Zero-knowledge IOPs Approaching Witness Length

Noga Ron-Zewi
University of Haifa
noga@cs.haifa.ac.il

Mor Weiss
Bar-Ilan University
mor.weiss@biu.ac.il

Abstract

Interactive Oracle Proofs (IOPs) allow a probabilistic verifier interacting with a prover to verify the validity of an NP statement while reading only few bits from the prover messages. IOPs generalize standard Probabilistically-Checkable Proofs (PCPs) to the interactive setting, and in the few years since their introduction have already exhibited major improvements in main parameters of interest (such as the proof length and prover and verifier running times), which in turn led to significant improvements in constructions of succinct arguments. *Zero-Knowledge* (ZK) IOPs additionally guarantee that the view of any query-bounded (possibly malicious) verifier can be efficiently simulated. ZK-IOPs are the main building block of succinct *ZK* arguments which use the underlying cryptographic object (e.g., a collision-resistant hash function) *as a black box*.

In this work, we construct the first *ZK-IOPs approaching the witness length* for a natural NP problem. More specifically, we design constant-query and constant-round IOPs for 3SAT in which the total communication is $(1+\gamma)m$, where m is the number of variables and $\gamma > 0$ is an arbitrarily small constant, and ZK holds against verifiers querying m^β bits from the prover's messages, for a constant $\beta > 0$. This gives a ZK variant of a recent result of Ron-Zewi and Rothblum (FOCS '20), who construct (non-ZK) IOPs approaching the witness length for a large class of NP languages. Previous constructions of *ZK*-IOPs incurred an (unspecified) large constant multiplicative overhead in the proof length, even when restricting to ZK against the *honest* verifier.

We obtain our ZK-IOPs by improving the two main building blocks underlying most ZK-IOP constructions, namely ZK codes and ZK-IOPs for sumcheck. More specifically, we give the first ZK-IOPs for sumcheck that achieve both *sublinear* communication for sumchecking a *general* tensor code, and a ZK guarantee. We also show a strong ZK preservation property for tensors of ZK codes, which extends a recent result of Bootle, Chiesa, and Liu (EC '22). Given the central role of these objects in designing ZK-IOPs, these results might be of independent interest.

Contents

1 Introduction

In this work we design short zero-knowledge Interactive Oracle Proofs (IOPs), whose length approaches the witness length. Before describing our results, we first give some background on zero-knowledge IOPs, and the special case of (zero-knowledge) probabilistically-checkable proofs.

PCPs. Probabilistically Checkable Proofs (PCPs) allow a probabilistic verifier $\mathcal{V}$ with oracle access to a purported proof π to verify claims of the form "$x \in L$" while making only a few queries to π. The celebrated PCP theorem [ALM+92, AS92] asserts that any non-deterministic language in $\mathsf{NTIME}(N)$ has a PCP that can be verified with $O(1)$ queries to a $\mathsf{poly}(N)$-length proof. Since their introduction, PCPs have found far-reaching applications, most notably to hardness of approximation, and to constructions of *succinct arguments* [Kil92, Mic00], namely highly-efficient proof systems in which the communication is *sublinear* in N, but soundness only holds against *efficient* malicious provers.

Over the last decades, there has been a large body of work attempting to reduce the length of PCPs, which was initially a very large polynomial. In particular, an influential line of work has led to constant-query PCPs of quasi-linear length $\tilde{O}(N)$ for languages in $\mathsf{NTIME}(N)$ [BGH+05, Din07]. Despite this progress, determining the minimal length of constant-query PCPs, and in particular obtaining *linear* length PCPs, is still an important goal. Since this goal depends on the computational model – because transitions between different computational models typically incur poly-logarithmic overheads – the focus is usually on a *specific* NP language of interest, such as circuit SAT or related variants. While constant-query and linear-length PCPs are currently not known for such languages, more recently [BKK+16] constructed $O(n)$-length PCPs for circuit SAT over size-n circuits, albeit with a large query complexity on the order of n^{ϵ} (for an arbitrarily small $\epsilon > 0$).

In what follows, for a non-deterministic language L, we use n, m, N to denote the input length, witness length, and non-deterministic verification time of L, respectively.

Zero-knowledge PCPs. Zero-Knowledge PCPs (ZK-PCPs) [KPT97], introduced soon after the works of [AS92, ALM+92], are PCPs in which the proof is *randomized*, and has an additional *zero-knowledge* (ZK) guarantee. Specifically, any (possibly malicious and computationally-unbounded) verifier that is *t-restricted* – namely, can only query t proof bits, for an a-priori fixed bound t – learns only the validity of the claim, i.e., that $x \in L$. This is formalized in the simulation-based paradigm by requiring that for any such t-restricted verifier there exists a Probabilistic Polynomial Time (PPT) simulator that is given *only the input* x, but does not have access to the proof or the NP witness, and can simulate the *view* of the verifier, consisting of her input and answers to her oracle queries. One motivation for the study of ZK-PCPs is that they can be used to construct succinct arguments that are also *ZK* in the sense that a computationally-bounded verifier learns nothing except the validity of the claim, and which use the underlying cryptographic building block (e.g., a collision-resistent hash function) *as a black-box*.

However, despite over two decades of research, ZK-PCP constructions are still far from matching the efficiency of non-ZK PCPs, requiring for example a large polynomial proof length [KPT97, IW14, IWY16], large query complexity [IKOS07, HVW21], exponential prover running time [GOS24], adaptive *honest* verification [KPT97, IW14], or only guaranteeing non-efficient ZK simulation [IWY16]. These ZK-PCPs are obtained via generic constructions employing cryptographic building blocks such as locking schemes [KPT97, IW14], secure multi-party computation protocols [IKOS07, HVW21], and leakage-resilient circuits [IWY16]. Consequently, unlike standard PCPs, known ZK-PCPs do

not posses any "nice" algebraic structure (even when the generic construction is based on a non-ZK PCP that *does* have such structure [KPT97, IW14, IWY16]). One notable exception is the recent independent work of [GOS24] whose ZK-IOPs for #P cleverly achieve ZK while preserving the algebraic structure of the underlying (non-ZK) PCP. However, the prover in their construction runs in exponential time, and it is not clear if (and how) their results can be scaled-down to NP.

IOPs. Interactive Oracle Proofs (IOPs) [BCS16, RRR17] were recently introduced as a generalization of the PCP model that combines also aspects of *Interactive* Proofs (IPs) [GMR85].[1] In an IOP system, the verifier $\mathcal{V}$ interacts with a prover $\mathcal{P}$ (similar to IPs), and has *oracle access* to the prover messages (as in a PCP). The study of IOPs – which has seen rapid progress in the few years since its inception – is motivated from both theoretical and practical perspectives.

From a theoretical perspective, a recent sequence of works resulted in IOPs whose efficiency exceeds their PCP counterparts, as well as IOP constructions in parameter regimes that are widely believed to be impossible in the PCP setting. For example, [BCG+17a] constructed *constant-query* IOPs for circuit SAT whose length (i.e., total communication) is linear in the circuit size n, improving on the n^{ϵ}*-query* linear-length PCP of [BKK+16]. Moreover, while if $\mathsf{NP} \subseteq \mathsf{coNP}/\mathsf{poly}$ then there do not exist PCPs whose length is a fixed polynomial in the length m of the NP *witness* [FS11], such IOPs – of length $\mathsf{poly}(m)$ – exist for a large class of NP languages [KR08].

Furthermore, in a more recent work [RR20], Ron-Zewi and Rothblum obtained IOPs that improve on both aforementioned constructions. Specifically, they obtained constant-query IOPs for circuit SAT of length approaching the *circuit size* n (i.e., $(1+\gamma)\cdot n$ for an arbitrarily small constant $\gamma > 0$), improving on the IOPs of [BCG+17a] which obtained length $c \cdot n$ for a large unspecified constant $c > 1$. Moreover, for a large class of NP languages (specifically, languages that can be verified in polynomial time and bounded polynomial space) they managed to obtain proof length approaching the *witness length* m. The technical core of the construction of [RR20] was a new *code switching* technique (inspired by [Mei13]) that allows one to trade less efficient polynomial codes, commonly used in such proof systems, with more efficient *tensor codes*. This technique was later used in follow up works to obtain IOPs with *linear-time provers* [BCG20, RR22].

From a practical perspective, IOPs lie at the core of state-of-the-art succinct argument systems, and their improved efficiency (compared to PCPs) has been leveraged in several recent works [BBHR18, BCR+19, BBHR19, BGKS20, Set20, GLS+23, XZS22], including practical implementations [BCR+19, Set20, GLS+23, XZS22].

Zero-knowledge IOPs. Zero-Knowledge IOPs (ZK-IOPs) [BCGV16, BCF+16] generalize standard IOPs to the ZK setting, analogously to how ZK-PCPs generalize standard PCPs. Similarly to ZK-PCPs, ZK-IOPs can also be used to design and implement succinct *black-box ZK* arguments.

A large body of works have studied different aspects of ZK-IOPs, leading to ZK-IOP constructions that significantly improve over the best ZK-PCP constructions to date.[2] In more detail, this line of work was initiated by Ben-Sasson et al. [BCGV16, BCF+17] who constructed a 2-round ZK-IOP for $\mathsf{NTIME}(N)$ with quasi-linear length $\widetilde{O}(N)$ (and logarithmic query complexity). Follow-up

[1] Special cases of IOPs were considered earlier, in the Interactive PCP (IPCP) model of Kalai and Raz [KR08], and the duplex PCPs of Ben-Sasson et al. [BCGV16].

[2] We note that works on ZK-IOPs have considered different ZK guarantees – ranging from Honest-Verifier ZK (HVZK) to ZK against t-restricted verifiers for an arbitrary t – while attempting to optimize various efficiency measures. In this overview, we focus on constructions with full-fledged ZK and short proof lengths, which are most relevant to our work. See Section B for a more detailed discussion.

work focused on the *Rank*-1 *Constraint Satisfaction* (R1CS) problem,[3] and obtained ZK-IOPs with either linear length $O(n)$ (and logarithmic query complexity) [BCR+19], or linear-time provers in the arithmetic circuit model)over a large, super-constant, sized field) [BCL22]. Unlike the aforementioned ZK-PCP constructions, these ZK-IOP constructions utilized zero-knowledge variants of standard PCP techniques such as arithmetization, error-correcting codes, and the sumcheck protocol [LFKN92].

1.1 Our Results

Despite the impressive progress described above, prior to this work there was still a gap between short ZK- and non-ZK IOPs, even when restricting to the setting of *honest-verifier* ZK. Specifically, while for natural NP languages, IOPs whose length approaches the witness length were known [RR20], such *ZK*-IOPs (and even *HVZK*-IOPs) were not known, and constructing them was posed in [BCL22] as an interesting question for future research.

In this work, we answer this question affirmatively by constructing ZK-IOPs *approaching the witness length* for the natural NP language of 3SAT, i.e., the language consisting of satisfiable 3CNFs. The natural NP witness for this language is the satisfying assignment $w \in \{0,1\}^m$, where m denotes the number of variables, and our main result gives a constant-round and constant-query ZK-IOP for 3SAT whose length (i.e., total communication) approaches m. We chose to focus in this work on the language of 3SAT because it is a basic and natural NP language. However, we believe our techniques are general and versatile enough to apply also to other related NP languages, such as k-SAT for constant $k > 3$, circuit SAT, and R1CS.[4]

The properties of our ZK-IOP are summarized in the following theorem, where a t-ZK-IOP is an IOP which is ZK against any (possibly malicious, adaptive and computationally unbounded) t-restricted verifier.

Theorem 1.1 (ZK-IOPs approaching witness length– Informal, see Theorem 7.1)**.** *For any constant $\gamma > 0$ there exists a constant $\beta > 0$, so that there exists a constant-round m^β-ZK-IOP for 3SAT on m variables with constant soundness error, communication complexity $(1+\gamma) \cdot m$, and constant query and round complexities.*

The above theorem is a direct generalization of the result of [RR20] which gave an IOP for 3SAT (as well as a larger class of NP problems) with the same properties, but *without* the ZK guarantee. Note that using known reductions from 3SAT to R1CS, the results of [BCR+19, BCL22] only give ZK-IOPs for 3SAT whose length is *linear* in the *input* length n, which may be as large as m^3 (since a 3CNF over m variables may have $\Omega(m^3)$ different clauses).[5]

Though this is not the main focus of our work, our IOPs also achieve decent running times for the prover and the verifier. Specifically, the verifier runs in *sublinear time*, on the order of m^ϵ for an arbitrarily small constant $\epsilon > 0$, after a $\mathsf{poly}(m)$-time local preprocessing step; And the prover

[3] In the R1CS problem, the input consists of matrices A, B and C and a vector x, and the question is whether there exists a vector z such that $(Ax') \star (Bx') = Cx'$, where $\star$ represents pointwise multiplication, and $x' = (x, z)$. This problem is known to admit a linear-time reduction from circuit satisfiability.

[4] We note that since we care about very small factors in the proof length, our results do not immediately apply to these problems because standard NP reductions between these problems typically incur large constant overheads in the witness length (indeed, even for the language of circuit SAT, making certain structural choices such as changing the allowed logical gates or the fan-in may incur large constant overheads in the circuit size).

[5] While it is plausible that the techniques of [BCR+19, BCL22] could be used to give ZK-IOPs for 3SAT that are *linear* in the *witness* length m, they don't seem to provide ZK-IOPs *approaching* the witness length for any natural NP languages such as R1CS, 3SAT, or circuit SAT; Indeed, obtaining such ZK-IOPs approaching the witness length was posed as an interesting question for future research in [BCL22, page 11].

runs in *polynomial time*, given the witness, and a generator matrix for a certain code sampled at random, see Theorem 7.1.[6]

Our ZK-IOPs are obtained by combining the code switching technique of [RR20] with new zero-knowledge ingredients. Specifically, our ZK-IOPs rely on new strong zero-knowledge properties for general tensor codes, as well as a new zero-knowledge sumcheck protocol for general tensor codes with *sublinear* communication. Given the ubiquity of tensor codes and the sumcheck protocol in IOP constructions, we believe that these new zero-knowledge ingredients may be of independent interest, and may serve as building blocks in future IOP constructions. Next we describe these building blocks, our new constructions and how we use them to obtain our Main Theorem 1.1.

2 Technical Overview and Additional Results

In this section, we provide an overview of our ZK-IOP approaching the witness length, and the main new ingredients used to obtain it. Towards this, we first describe in Sections 2.1 and 2.2 below the main new ingredients used in the design of our ZK-IOP, which are new strong zero-knowledge properties of tensor codes, and a zero-knowledge sumcheck protocol for general tensor codes with *sublinear* communication. Then in Section 2.3 we explain how these new ingredients can be used to obtain our ZK-IOP approaching the witness length, and how to solve some additional challenges that arise in the design of these IOPs. Finally, in Section 2.4 we highlight some interesting directions for future research, based on our new ingredients.

2.1 Zero-Knowledge Properties of Tensor Codes

Our first main ingredient is establishing new strong zero-knowledge properties of tensor codes. These properties extend recent results on zero-knowledge properties of tensor codes [BCL22], and answer some open questions posed there. Before stating our results, we first give some background on zero-knowledge codes and tensor codes, and their importance in IOP design.

Zero-Knowledge Codes and Their Uses in IOP Constructions. Many (ZK-)IOP constructions follow a similar template to PCP constructions, in which the witness (or the entire computation) is encoded using an error-correcting code. At a high-level, the encoding typically serves two orthogonal purposes. First, it enables verification by exploiting a built-in mechanism of the code that assists verification (e.g., supporting the *sumcheck protocol*, see Section 2.2 below for more details). Second, to guarantee zero-knowledge, the encoding is replaced with a *randomized* encoding that is *zero-knowledge* in the sense that even malicious adversaries that query many codeword symbols learn nothing about the encoded message. We say that a code is *t-zero-knowledge* (t-ZK) if it is ZK (in the simulation-based paradigm) against malicious adversaries that can query at most t codeword symbols.

Verification of the IOP usually also requires checking that the prover sent a valid witness encoding. For this, the code is taken to be *locally testable*, which informally means that there exists a randomized oracle algorithm that can determine whether a given word w is a valid codeword, or rather far from any codeword, by making few queries to w. We say that a code is *q-locally testable* (q-LTC) if there exists such an algorithm (called a local tester) that makes at most q queries to w. Naturally, it is desirable that the ZK parameter of a code would be significantly larger than the

[6] We note that the sampling of the generator matrix can be handled by, e.g., allowing non-uniform provers, see Remark 6.2 for more details.

query complexity of the local tester, i.e., $t \gg q$, since even the *honest* verifier must at the very least make q queries to the purported codeword. Being slightly informal, in this overview, we refer to a code that is t-ZK and q-LTC for $t \gg q$ as a *ZK-LTC*.

Tensor Codes. Traditionally, PCP constructions used polynomial-based codes such as the *Low-Degree Extension* (LDE). Indeed, these codes are long-known to be locally testable [RS96], and to also admit an efficient sumcheck protocol [LFKN92]. These codes are also known to be ZK, and they in fact satisfy a stronger ZK property called *t-uniform ZK* which roughly states that any t symbols in a random encoding of the witness are *uniformly random*. (This should be contrasted with standard t-ZK, which only guarantees that the queried codeword symbols can be efficiently simulated without knowing the encoded message.)

The LDE encoding can be viewed as a special case of *tensor codes*, which we now define. Let $C_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}, C_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ be *linear* codes (i.e., C_1 and C_2 are linear maps over a finite field $\mathbb{F}$). Then the *tensor (product) code* $C_1 \otimes C_2$ is a code whose codewords are $n_1 \times n_2$ matrices with the constraint that each column is a codeword of C_1 and each row is a codeword of C_2. The definition naturally extends to higher dimensions by viewing codewords in the d-dimensional tensor $(C_0)^{\otimes d}$ of a *base code* C_0 as d-dimensional cubes, so that the restriction to any axis-parallel line is a codeword of C_0. Using this terminology, the LDE code corresponds to the special case of C_0 being the *Reed-Solomon (*RS*) code*.

A recent line of work, starting with [Mei13, RR20], has shown that in fact the LDE encoding can be replaced with more general tensor codes. Indeed, tensor codes admit the sumcheck protocol [Mei13], are locally testable [BS06, Vid15], and also have certain ZK properties [ISVW13, BCL22]. Moreover, as utilized in a recent series of works, tensor codes have several advantages over polynomial-based codes, such as achieving higher *rate* (i.e., shorter encoding length), smaller alphabet, or linear-time encoding, by picking a base code satisfying these properties.

Our Results. In a recent work, Bootle et al. [BCL22] define a "natural" randomized encoding function for linear codes and their tensors (See Definitions 3.18 and 3.30), and show that the (uniform) ZK of any pair of linear codes is preserved under tensor products with respect to this randomized encoding function. More accurately, they show that if C_1, C_2 have t_1- and t_2-ZK (t_1- and t_2-uniform ZK, resp.), then $C_1 \otimes C_2$ has $\min\{t_1, t_2\}$-ZK ($\min\{t_1, t_2\}$-uniform ZK, resp.). On the negative side, they show an example of codes C_1, C_2 that have t_1- and t_2-uniform ZK, but $C_1 \otimes C_2$ does not have $\max\{t_1, t_2\}$-*uniform* ZK, and pose the question of whether $C_1 \otimes C_2$ can have a larger ZK-threshold (such as $\max\{t_1, t_2\}$ or even $t_1 \cdot t_2$) for (standard, i.e., *non-uniform*) ZK [BCL20, Page 19].

We provide several extensions of the results of [BCL22]. These extensions answer the question posed in [BCL22], and also imply ZK local testing procedures for tensor codes that will be useful for obtaining our ZK-IOP of Theorem 1.1.

(1) Strong ZK properties of tensor codes. First, we strengthen the results of [BCL22] by showing that if C_1, C_2 have t_1- and t_2-ZK, then $C_1 \otimes C_2$ has ZK against an adversary that reads t_1 *entire rows* or t_2 *entire columns*. As a corollary, this implies that $C_1 \otimes C_2$ has $\max\{t_1, t_2\}$-(non-uniform)-ZK, answering *positively* the open question posed in [BCL22].

We also extend the above result on ZK against row/column adversaries to adversaries that query higher-dimensional axis-parallel subpaces in higher-dimensional tensors $(C_0)^{\otimes d}$. Since the local tester for tensor codes queries a random axis-parallel subspace [BS06, Vid15], this implies as

a corollary that if C_0 has 1-ZK, then its tensor product $(C_0)^{\otimes d}$ has a local testing procedure that has ZK against the *honest* tester.

Remark 2.1 (Connection to Secret Sharing)**.** *The fact that, for an appropriate ZK parameter, ZK holds against adversaries reading full rows or full columns is well known for the special case of* 2-dimensional *tensors* of RS*, which are used to design secret sharing schemes (under the name "bivariate Shamir"). Moreover, originating in [BGW88], this property of 2-dimensional tensors of RS was used extensively to design verifiable secret sharing schemes and secure multi-party computation protocols. The special case of* 2-dimensional *tensors of* general *linear codes was used in [CDM00] to design verifiable secret sharing and secure multiparty computation protocols.*

(2) Limitations on the ZK threshold of tensor codes. Second, we show an example of a code C_0 that has t-uniform ZK, but $C_0 \otimes C_0$ does not have $\omega(t)$-(non-uniform) ZK. This answers *negatively* the open question posed in [BCL22], by showing that $C_1 \otimes C_2$ does not necessarily have $(t_1 \cdot t_2)$-(non-uniform)-ZK, even if C_1 and C_2 have t_1- and t_2-uniform ZK. This also implies that even if C_0 has a high zero-knowledge threshold, then $(C_0)^{\otimes d}$ is not necessarily a ZK-LTC with respect to the aforementioned standard local tester which queries a random axis-parallel subspace. That is, the query complexity of this local tester may not be significantly smaller then the zero-knowledge threshold, meaning the local testing procedure is *not* ZK for a large class of *malicious* testers. [7]

To overcome the above limitation, we show that the tensor product of codes with uniform-ZK has (standard) ZK against *adaptive* row/column adversaries. Specifically, we prove that if C_1, C_2 have t_1- and t_2-uniform ZK, then the tensor product $C_1 \otimes C_2$ is ZK against adversaries that first make $t \leq \min\{t_1, t_2\}$ point queries, and only then adaptively decide whether to read $t_1 - t$ entire rows or $t_2 - t$ entire columns. As before, this result extends to higher-dimensional axis-parallel subspaces in higher-dimensional tensors. This property of codes with uniform-ZK is useful, because we show that it implies as a corollary that if C_0 has t-uniform ZK, then there exists a 2-round *constant query* ZK-IOP – with ZK against $(t-1)$-restricted *malicious* verifiers – for testing membership in the tensor product $(C_0)^{\otimes d}$. [8]

At a high level, the ZK-IOP verifier first executes the local tester for the tensor product, but instead of making the queries, the verifier sends the query set to the prover, and then both parties run a short constant-query PCPP to verify that the tester would have accepted on this query set. These ZK-IOPs for membership testing can be viewed as a relaxation of ZK-LTCs, where the latter correspond to the special case where there is no communication.

Proof Overview. In [BCL22], the ZK properties of tensor codes were shown using an *algebraic characterization* of linear codes having ZK against point queries, based on the generator matrix of the code and the distance of the dual code. We instead take an *algorithmic / simulation-based* approach that exploits the structure and symmetry properties of tensor codes to directly design simulators for these stronger ZK properties.

[7] Our negative result should be constructed with a result of [ISVW13], who showed that there *exists* a randomized encoding function for tensor codes - not necessarily the natural one considered in [BCL22] – so that for *any* linear base code $C_0 : \mathbb{F}^k \to \mathbb{F}^n$, $C_0 \otimes C_0$ has $\Omega(n^2)$-ZK with respect to this encoding function. This implies in turn that for *any* base code C_0, there *exists* a randomized encoding function with respect to which its tensor product $(C_0)^{\otimes d}$ is a ZK-LTC. However, this randomized encoding function does not suffice for our purposes since it is non-explicit, and we need the special structure of the randomized encoding function of [BCL22].

[8] In more detail, in the ZK-IOP the verifier first executes the local tester for the tensor product, but instead of making the queries, the verifier sends the query set to the prover, and then both parties run a short constant-query PCPP to verify that the tester would have accepted on this query set.

To show the negative result, we take the code C_0 to be the *punctured Reed-Solomon code* $\mathsf{PRS} : \mathbb{F}^k \to \mathbb{F}^n$, defined as follows. Let $I \subseteq \mathbb{F}$ be a fixed subset of field elements of size $|I| = k$. Given a message $\mathsf{m} \in \mathbb{F}^k$, let $f_{\mathsf{m}}(X)$ be the (unique) univariate polynomial of degree at most $k-1$ over $\mathbb{F}$ which satisfies that $f_{\mathsf{m}}(i) = \mathsf{m}(i)$ for any $i \in I$. The encoding of m is the evaluation table of $f_{\mathsf{m}}(X)$ on $\mathbb{F} \setminus I$. It follows by definition, that in the tensor product $\mathsf{PRS} \otimes \mathsf{PRS} : \mathbb{F}^{k \times k} \to \mathbb{F}^{n \times n}$, the message can be viewed as the evaluation table over $I \times I$ of a *bivariate* polynomial $g(X, Y)$ of *individual* degree at most $k-1$ over $\mathbb{F}$, while the codeword corresponds to the evaluation table of $g(X, Y)$ over $(\mathbb{F} \setminus I) \times (\mathbb{F} \setminus I)$.

The uniform-ZK property of the PRS code follows by the *MDS property* of this code which states that the value of any k entries in the codeword determines the codeword (and the encoded message). Our limitation result upper-bounding the ZK threshold of $\mathsf{PRS} \otimes \mathsf{PRS}$ relies on the *local decodability* property of bivariate polynomials. Specifically, while our results above show that the tensor product $\mathsf{PRS} \otimes \mathsf{PRS}$ has ZK against an adversary that queries *axis-parallel* lines, it is *not* ZK against lines in *arbitrary directions*! In particular, querying the line (i, i) for $i \in \mathbb{F} \setminus I$ reveals the value of the bivariate polynomial on all points (i, i) for $i \in \mathbb{F}$, which correspond to message entries.

2.2 A Zero-knowledge Sumcheck Protocol with Sublinear Communication

The second main building block in our ZK-IOPs approaching the witness length is a zero-knowledge sumcheck protocol with sublinear communication.

In the following, let $C : \mathbb{F}^k \to \mathbb{F}^n$ be an error-correcting code encoding length-k *messages* m into length-n *codewords* $C(\mathsf{m})$. The *sumcheck protocol* is an interactive proof that allows the verifier to check the validity of claims of the form "$\sum_{i \in [k]} \mathsf{m}(i) = \alpha$" for some value $\alpha \in \mathbb{F}$, given oracle access to the encoding $c = C(\mathsf{m})$ of m. Such a protocol was initially shown for polynomial-based codes (such as LDE) in [LFKN92], and was more recently extended to general tensor codes in [Mei13]. Quite amazingly, these protocols allow the verifier to check the validity of the claim "$\sum_{i \in [k]} \mathsf{m}(i) = \alpha$" by making only a *single* query to c, and with total communication that is *sublinear* in the codeword length n. As mentioned above, the sumcheck protocol is typically used to verify claims about correctness of the encoded computation. Ron-Zewi and Rothblum [RR20] (following Meir [Mei13]) used the sumcheck protocol also as the basis for their *code switching* technique.[9]

To obtain our ZK-IOP approaching the witness length we require a *zero-knowledge* version of the sumcheck protocol described above in which, roughly speaking, any t-restricted verifier $\mathcal{V}^*$ learns only few physical symbols of c, in addition to the correctness of the sumcheck claim. In prior works [BCGV16, BCF+17, BCG+17a, BCR+19, CHM+20] this property was achieved by executing the (non-ZK) sumcheck protocol on a *random shift* of the original codeword. More precisely, to test whether $\sum_{i \in [k]} m(i) = \alpha$, given oracle access to c, the sumcheck protocol is executed on a shifted codeword of the form $c' := R + \gamma \cdot c$, where R is a random codeword provided by the prover as an oracle, and γ is a random scalar provided by the verifier after she receives the oracle R (the verifier also needs to locally test R to make sure it is close to a valid codeword of C).

While the above method leads to a ZK sumcheck protocol which uses the underlying (non-ZK) sumcheck protocol as a *black box*, the communication becomes *linear* in the codeword length because of the need to send the random codeword R. Jumping ahead, applying this method in our ZK-IOP for 3SAT with c being the encoded witness, will *double* the communication, and in partic-

[9] We note that the code switching technique actually requires an extension of the sumcheck protocol, in which the goal is to verify claims of the form "$\sum_{i \in [k]} \lambda(i) \cdot \mathsf{m}(i) = \alpha$", where the coefficients $\lambda(i)$ have a certain tensor structure.

ular will not lead to a total communication which approaches the witness length. We remark that *sublinear*-communication sumcheck protocols are known for the special case of (certain families of) *polynomial* codes. Specifically, by exploiting special properties of these codes, Xie et al. [XZZ+19] design such a protocol based on the GKR protocol [GKR15], while Bootle et al. [BCL22] design such a protocol for *sparse* polynomial codes. However, these protocols do not seem to immediately apply to the setting of general tensor codes, and thus do not suffices for our purposes.

Our Results. We a design new sumcheck protocol with *sublinear* communication. We do so by exploiting the structure of Meir's protocol for general tensor codes [Mei13], instead of executing it as a black box on a masked codeword. This in turn leads to a protocol with *sublinear* communication, albeit with weaker ZK guarantees that we discuss below. In a nutshell, while in our sumcheck protocol $\mathcal{V}^*$ potentially learns more symbols of c compared to prior work, we show that this subset of symbols is *highly structured*. Importantly, these weaker ZK guarantees suffice for our purposes, given the stronger ZK properties of tensor codes (discussed in the previous section) that we are able to show. Our sumcheck protocol obtains two different flavors of zero-knowledge, depending on the properties of the underlying base code of the tensor product. Next we discuss these two guarantees.

(1) General LTC Base Codes. First, we show that if C_0 is an *arbitrary* q-LTC, then the view of any (possibly malicious and unbounded) t-restricted verifier $\mathcal{V}^*$ in our sumcheck protocol for the tensor product $C := (C_0)^{\otimes d}$ can be perfectly simulated given only $(t+q+1)$ "rows" of the given codeword $c \in (C_0)^{\otimes d}$, when viewed as a codeword in the two-dimensional tensor $C_0 \otimes (C_0)^{\otimes(d-1)} = (C_0)^{\otimes d}$, namely given codewords in $(C_0)^{\otimes(d-1)}$.

Theorem 2.2 (Sublinear-Communication ZK Sumcheck – Informal, see Theorem 5.2)**.** *For any constant integer $d > 1$ and a q-LTC $C_0 : \mathbb{F}^k \to \mathbb{F}^n$, there exists a constant round IOP with constant soundness error and communication complexity $O(n)$ for verifying a sumcheck of $(C_0)^{\otimes d}$, with the following ZK guarantee: There exists a black-box straight-line PPT simulator* Sim *that can perfectly simulate the view of any t-restricted verifier, given $t+q+1$ rows of $c \in (C_0) \otimes (C_0)^{\otimes d-1}$ of* Sim*'s choice. The verifier makes a single query to the given codeword c and $O(1)$ queries to the first prover's message, and reads the other prover messages in full.*

Remark 2.3. *We note that the protocol given in the above theorem also applies for verifying sums of the form "$\sum_{i\in[k]} \lambda(i) \cdot \mathsf{m}(i) = \alpha$", where the coefficients $\lambda(i)$ have a certain tensor structure. As shown in [RR20], such a generalized form of sumcheck implies a* code switching protocol*, in which one can simulate queries to a codeword $C(\mathsf{m})$ in a tensor code C by making a constant number of queries to the encoding $C'(\mathsf{m})$ of m via another (unrelated) tensor code C'. Thus, the above theorem also gives a sublinear-communication ZK protocol for code switching.*

Notice that the communication complexity of the sumcheck protocol given in the above theorem is indeed sublinear in the codeword length of $C = (C_0)^{\otimes d}$, which is n^d. We note that while the simulator in the above theorem makes q more queries then the verifier, we show that by increasing the honest verifier's query complexity to q (instead of constant), we can design a simulator that only makes $t+1$ row queries (see Remark 5.5). Instantiating the above Theorem 2.2 with a code C_0 that is a ZK-LTC (specifically, a q-LTC that has $(t+q+1)$-ZK), gives a (distributional)[10] sumcheck

[10]Roughly, distributional ZK guarantees that for any t-restricted verifier $\mathcal{V}^*$, and for any message m, the view of $\mathcal{V}^*$ when given oracle access to a random encoding c of m can be efficiently and perfectly simulated *without access to c* (see Definition 3.23).

protocol with *full-fledged* ZK against t-restricted verifiers (that is, $\mathcal{V}^*$ learns *nothing* about c except the sum α, see Corollary 5.10).

(2) Tensor Base Codes. Unfortunately, Theorem 2.2 does not lead to a sumcheck protocol with full-fledged ZK when instantiated with a base code C_0 that is not a ZK-LTC (i.e., in which the query complexity of the local tester is not sufficiently smaller than the ZK threshold). In particular, it does not provide a meaningful ZK guarantee for tensors of PRS, which are ubiquitous in ZK-IOP/PCP design, and are also used in this work. To overcome this issue, we consider the case that the base code C_0 is itself a low-dimensional tensor product of another code B, say $C_0 = B^{\otimes 3}$,[11] and show that for such codes the information leakage in our sumcheck protocol can be reduced. Specifically, we show that in this case the view of any t-restricted verifier can be simulated by a PPT simulator that first makes $t' \leq t$ point queries, and then queries $(t + 2 - t')$ axis-parallel hyperplanes in a certain direction from the codeword c, when viewed as a codeword in the tensor product $B^{\otimes(3d)} = (C_0)^{\otimes d}$ (the direction of the hyperplanes is chosen by the malicious verifier after making the point queries; See Theorem 5.2 for a formal statement).

As we have shown in Section 2.1 above, tensor products of uniform ZK codes have ZK against this type of queries, and consequently instantiating the above variant of our sumcheck protocol with a code $C_0 = B^{\otimes 3}$ for a t-uniform ZK code B (though not necessarily a ZK-*LTC*) gives full-fledged ZK against $(t - 2)$-restricted verifiers. Jumping ahead, this variant of our sumcheck protocol will be crucial for our ZK-IOP approaching the witness length, since the ZK-IOP includes executing a sumcheck protocol on a tensor of PRS which has uniform-ZK but is not a ZK-LTC.

Proof Overview. Both our results (for LTC/tensor base codes C_0) are obtained by utilizing the tensor structure of C. Specifically, our main observation is that masking with a random codeword *in the base code* C_0 suffices to hide c. For the sake of this overview, assume for simplicity that $d = 2$, i.e., our goal is to design a sumcheck protocol for the code $C := C_0 \otimes C_0$. In our protocol, the prover $\mathcal{P}$ samples a random $r \leftarrow \mathbb{F}^k$, and sets $R = C(\bar{r}) \in \mathbb{F}^{n \times n}$, where $\bar{r} \in \mathbb{F}^{k \times k}$ denotes the matrix whose first column is r, and all other entries are 0. A crucial point is that every entry $R(i, j)$ of R can be computed given a single entry in $\hat{r} := C_0(r) \in \mathbb{F}^n$. Therefore, $\mathcal{P}$ need not send R (whose length is n^2) to the verifier $\mathcal{V}$ - it suffices to only send $\hat{r}$ (whose length is n, which is in particular *sublinear* in the codeword length n^2 of C). The protocol then roughly proceeds as before: $\mathcal{P}$ sends $\hat{r} = C_0(r)$, and the parties execute the sumcheck protocol on the codeword $c' := R + \gamma \cdot c$, where γ is a random scalar provided by the verifier after she receives the oracle $\hat{r}$.

However, one additional point that needs to be handled is that a malicious prover may send a malformed $\hat{r}$ which is *not* a C_0-codeword. $\mathcal{V}$ therefore needs to check that $\hat{r}$ is a codeword of C_0. Notice that to do so while preserving ZK of the sumcheck protocol, we need the *base code* C_0 to be *locally testable*. Indeed, since $\hat{r}$ is used to mask c then, intuitively, ZK requires that $\hat{r}$ remain (mostly) hidden. If C_0 isn't locally testable, checking that $\hat{r} \in C_0$ would require fully reading $\hat{r}$, but then it would not mask c at all because $\mathcal{V}$ would fully know $\hat{r}$. Furthermore, to get constant query complexity (independent of the query complexity of C_0), we add an additional round in which $\mathcal{V}$ executes the local tester, but instead of making the queries herself, she sends the query set to $\mathcal{P}$, who responds with the answers v to the queries. $\mathcal{V}$ then checks that v is an accepting view of the tester, and that it is consistent with $\hat{r}$ (by making a single query to $\hat{r}$).

It is important to note that since the base code C_0 has *no* ZK guarantees, we cannot hope to claim that $\mathcal{V}^*$ learns *nothing* about m (except the sum α). Indeed, even a *single* query to c can reveal

[11] We use 3-dimensional tensor products because this is the smallest dimension for which the tensor product is known to be locally testable [BS06, Vid15].

non-trivial information (in particular, linear dependencies) on m. Therefore, following previous works on ZK sumcheck, we aim to *quantify* the amount (and type) of information which $\mathcal{V}^*$ learns during the sumcheck protocol – from the prover messages, and her queries to c and $\hat{r}$. While each of the t queries made to c and $\hat{r}$ can be simulated by making just a single point query to c, to consistently simulate $\mathcal{P}$'s later messages conditioned on these values, our simulator needs to obtain the whole row for each of these queries to $c, \hat{r}$. Our simulator makes q additional queries to c to simulate v during the local testing step described in the previous paragraph. This explains the structure and number of queries made by the simulator of Theorem 2.2.

For the special case that $C_0 = B^{\otimes 3}$ for some code B, we use the standard local tester for tensor codes – which queries a random axis-parallel two-dimensional plane [BS06, Vid15] – to test membership in $C_0 = B^{\otimes 3}$. In this case, the verifier queries during local testing are highly structured, which assist the simulation. Specifically, v can be simulated by querying a *single hyperplane* of c, when viewed as a codeword in $B^{\otimes 3} = C_0^{\otimes 6}$, instead of making q queries to c. More accurately, the simulator needs access to a stronger oracle which allows the adversary (the simulator, in this case) to first make point queries to c, and then adaptively pick the direction of the hyperplanes. (Indeed, the honest local tester picks an axis-parallel two-dimensional plane in a *random* direction, that is not known in advance. Consequently, *even the honest* verifier picks the direction adaptively, and a malicious verifier can additionally pick it *arbitrarily*.) This reduces the total number of row queries to $t + 2$.

2.3 ZK-IOPs Approaching the Witness Length

In this section, we outline how we obtain our ZK-IOPs approaching the witness length, by combining our new sublinear-communication ZK sumcheck protocol for tensor codes from the previous section, with our results on strong ZK properties for tensor codes discussed in Section 2.1. To this end, we first give an overview of the IOPs approaching the witness length of Ron-Zewi and Rothblum [RR22], and then discuss how our new ingredients can be used to add a zero-knowledge guarantee to this protocol. Finally, we highlight some additional challenges that arise in the design of our IOP.

IOPs for 3SAT. A main idea in PCP/IOP design – which goes back to the first PCP constructions [BFL91, BFLS91, Sud00, AS98, ALM+98], and was also used in many later PCPs and IOPs – is to encode formulas using low-degree polynomials (a.k.a, arithmetization). At a high level, a 3CNF formula φ over m variables, with a satisfying assignment $w \in \{0,1\}^m$, can be encoded using a single low-degree polynomial over a finite field $\mathbb{F}$ of size $|\mathbb{F}| \gg \log m$. This is done by first encoding w using a tensor of the Reed-Solomon code (i.e., the LDE encoding), where the encoding $\hat{w}$ of w is the unique s-variate polynomial of individual degree h (for appropriate s, h) such that $\hat{w}(i) = w_i$ for every $i \in \{0,1\}^{\log(m)} \equiv [m]$. The prover's goal is now to convince the verifier that $\hat{w}$ is an LDE of a satisfying assignment for φ. By exploiting the algebraic structure of the LDE encoding, this check can be carried out using the sumcheck protocol. More specifically, as pointed out in [RR20], the sumcheck protocol can be viewed as an *interactive reduction* that (roughly) allows a verifier interacting with a prover to reduce a claim of the form "w satisfies φ" to claims "$\hat{w}(\mathbf{i}_j) = v_j$, $j = 1, 2, 3$" for random and independent $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3$. Thus, combining the LDE encoding with the sumcheck protocol results in the following blueprint for an IOP $(\mathcal{P}, \mathcal{V})$ for 3SAT:

1. $\mathcal{P}$ computes the LDE $\hat{w}$ of w, and sends it to $\mathcal{V}$.

2. $\mathcal{V}$ checks that $\hat{w}$ is indeed (close to) the LDE of some assignment to the variables of φ, and otherwise rejects. (This requires local testability, which the LDE has through a low-degree test [RS96].)

3. The prover and verifier engage in the sumcheck protocol to reduce the claim "w satisfies φ" to the claims "$\hat{w}(\mathbf{i}_j) = v_j$, $j = 1, 2, 3$" for random and independent $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3$.

4. $\mathcal{V}$ reads $\hat{w}(\mathbf{i}_1), \hat{w}(\mathbf{i}_2), \hat{w}(\mathbf{i}_3)$ and accepts if and only if $\hat{w}(\mathbf{i}_j) = v_j$ for every $j \in \{1, 2, 3\}$.

IOPs approaching the witness length. The IOP approaching the witness length of [RR20] combines the blueprint described above with a new technique called *code switching* which they introduce (inspired by [Mei13]).[12] More specifically, the goal of [RR20] was to design IOPs whose total communication approaches the witness length m. Therefore, they could not afford to send the entire LDE encoding $\hat{w}$ for two main reasons. First, Reed-Solomon codes, and consequently also the LDE encoding, are defined over (large) super-constant alphabet size. Consequently, encoding the binary assignment w using the LDE encoding will incur a *super-constant overhead* which cannot lead to linear-length IOPs, let alone ones approaching the witness length. Second, even ignoring the alphabet-size issue, as pointed out in [Mei13, Mei14], the encoding requires a *multiplication property* which facilitates checking non-linear relations, and this property inherently requires rate at most $\frac{1}{2}$ [Ran13]. (In particular, the encoding will at least *double the length*, and consequently will not result in a proof approaching the witness length.)

Instead, they showed how to carry out the blueprint above using an encoding of the witness w in any *binary* tensor code C of *high-rate*, instead of the LDE encoding. More specifically, Ron-Zewi and Rothblum replace the LDE encoding in Step 1 of the blueprint above with the encoding $C(w)$ of w, and replace the low-degree test executed in Step 2 with a local test for C (using the fact that tensor codes are locally testable [BS06, Vid15]). Then, they execute the sumcheck of Step 3. (Crucially, this can be done even though $\mathcal{V}$ does not have access to $\hat{w}$.) Finally, to perform Step 4, $\mathcal{P}$ sends v_1, v_2, v_3 to $\mathcal{V}$ (recall that $\mathcal{V}$ does not have $\hat{w}$ and therefore cannot query these values herself). The parties now need to engage in an interaction which will prove to $\mathcal{V}$ that $\mathcal{P}$ sent the correct values $\hat{w}(\mathbf{i}_1), \hat{w}(\mathbf{i}_2), \hat{w}(\mathbf{i}_3)$. [RR20] show that checking each claim "$\hat{w}(\mathbf{i}_j) = v_j$" reduces to running (a variant of) the sumcheck protocol on $C(w)$.[13]

In summary, at a high-level the verifier in the IOP of [RR20] performs a local test on the high-rate encoding $C(w)$ of the witness, then engages in four executions of (variants of) the sumcheck protocol. This blueprint is described in Figure 1.

Our ZK-IOPs approaching the witness length. Recall that in a t-ZK-IOP, a (possibly malicious) verifier $\mathcal{V}^*$ that makes up to t queries to her oracles should learn nothing about the satisfying assignment w. Considering the blueprint of [RR20]'s IOP, $\mathcal{V}^*$ potentially obtains information about w from: (1) querying $C(w)$; and (2) messages which $\mathcal{P}$ sends in the executions of the sumcheck protocols on $\hat{w}$ and $C(w)$. We note that while we can easily overcome (1) by taking C to be a ZK code, and we can similarly prevent information leakage in (2) by using a sumcheck ZK-IOP, existing constructions of sumcheck ZK-IOPs (discussed in Section 2.2 above) either do not have

[12] As mentioned above, the IOPs of [RR20] are for the much larger class of NP relations for which membership can be decided in polynomial time and bounded polynomial space; however, we only describe here their IOPs for 3SAT.

[13] More accurately, they show that this reduces to executing a *generalized* version of the sumcheck protocol on $C(w)$, for verifying sums of the form $\sum_i \lambda(i) \cdot w(i) = \alpha$, where the coefficients $\lambda(i)$ have a certain tensor structure. In this overview, we mostly ignore this fact. This will not affect the rest of the overview since everything we describe for the standard sumcheck protocol naturally extends also to the generalized version. See Section 5 for further details.

Blueprint of an IOP for 3SAT

The IOP is executed between a prover $\mathcal{P}$ that has input a 3CNF formula φ on m variables, and a satisfying assignment $w \in \{0,1\}^m$ for φ, and a verifier $\mathcal{V}$ that has input φ. The IOP uses a high-rate binary tensor code C, and uses the sumcheck protocol as a sub-protocol.

1. $\mathcal{P}$ computes the high-rate encoding $C(w)$ of w, and sends it to $\mathcal{V}$.
2. $\mathcal{V}$ performs a local test to check that the purported codeword which $\mathcal{P}$ sent is close to C.
3. The prover and verifier engage in the sumcheck protocol to reduce the claim "w satisfies φ" to claims "$\hat{w}(\mathbf{i}_j) = v_j$", $j = 1, 2, 3$ for random and independent $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3$.
4. $\mathcal{P}$ sends v_1, v_2, v_3 to $\mathcal{V}$, and for every $j \in \{1, 2, 3\}$ the parties engage in a sumcheck protocol over $C(w)$ to check that $\hat{w}(\mathbf{i}_j) = v_j$.

Figure 1: Blueprint of an IOP for 3SAT using Code Switching [RR20]

sublinear communication [BCGV16, BCF+17, BCG+17a, BCR+19, CHM+20], or are specialized for *polynomial codes* [XZZ+19, BCL22]. The former cannot be used here because they would lead to communication complexity of at least $2|w|$ (because the prover also sends $C(w)$ which has size $\geq |w|$). The latter cannot be used because they do not readily extend to general tensor codes, whereas we execute the sumcheck ZK-IOP on a *binary* tensor code C of *high-rate* which is not a polynomial code.

Equipped with our results on sumcheck ZK-IOPs and on ZK codes from the previous sections, our short ZK-IOP for 3SAT now follows along the lines of [RR20]'s IOP for 3SAT (Figure 1). More specifically, in Step 1 the prover first generates a randomized encoding $\hat{w}$ of w via a tensor product of the PRS code (using the natural randomized encoding function for tensor codes). Then the prover generates another encoding of w via a tensor product of a high-rate binary ZK-LTC (whose existence is guaranteed by [ISVW13]), and only sends the latter encoding to the verifier. Then, in Step 3 we use our specialized sumcheck ZK-IOP for tensor base codes. Finally, in Step 4 we use our sumcheck ZK-IOP for general base codes.

Additional challenges in the design of our ZK-IOPs. Finally, we point out two additional challenges that arise in the above high-level approach.

(1) High-rate Encoding of Randomized Witness: The code switching technique of [RR20] relied on the fact that any point on the low-degree extension $\hat{w}$ is a linear combination $\sum_i \lambda(i)w(i)$ of the entries of $w \in \{0,1\}^m$, where the coefficients $\lambda(i)$ have a certain *tensor structure*. As mentioned above, the value of the sum $\sum_i \lambda(i)w(i)$ can then be verified by executing (a scaled version of) the sumcheck protocol on the encoding $C(w)$ of *the same* w via another unrelated tensor code C.

In our ZK-IOP, $\hat{w}$ is obtained by encoding w via a tensor product of the PRS code, using a natural *randomized* encoding function for tensor codes [BCL22] (some form of randomization is inherent to achieving ZK). To explain the issue that arises in the randomized setting, consider the special case of 2-dimensional tensors, where we view the message w as a $k \times k$ matrix. The natural randomized encoding function for tensor codes first extends w to a matrix $\bar{w}$ of size $\bar{k} \times \bar{k}$ for $\bar{k} > k$, by appending random and independent field elements in all entries outside of w, and then $\hat{w}$ is obtained by encoding $\bar{w}$ *deterministically* using the tensor product of PRS codes. But now to apply the code switching technique, the prover needs to provide the encoding $C(\bar{w})$ *of* $\bar{w}$ – whose

entries come from a *large field* – via the *binary* code C, which if done naively would cause a super-constant blowup. (As mentioned above, choosing C to be a binary code is necessary to guarantee short length in our ZK-IOP for 3SAT.)

To handle this, we choose the large field $\mathbb{F}$ to be an extension field of the binary field, and find a novel way to map the message $\bar{w} \in \mathbb{F}^{\bar{k} \times \bar{k}}$ into a *binary* message $w' \in \{0,1\}^{k' \times k'}$, for k' not much larger than $\bar{k}$, and to map the tensor coefficients $\lambda(i)$ into other tensor coefficients $\lambda'(i)$ satisfying the property that $\sum_i \lambda(i)\bar{w}(i) = \sum_i \lambda'(i)w'(i)$. Consequently, the prover can provide the encoding of the *binary* message w' via the *binary* code C, and verifying the linear combination $\sum_i \lambda(i)\bar{w}(i)$ reduces to executing the sumcheck protocol on $C(w')$ with tensor coefficients $\lambda'(i)$. (See Lemma 6.5 and Claim 6.3 for the definition and properties of w' and the coefficients $\lambda'(i)$.)

(2) Proving ZK of the Combined IOP: Roughly, the ZK property of our sumcheck IOP for tensor base codes guarantees that in Step 3, a (possibly malicious and unbounded) query-restricted verifier $\mathcal{V}^*$ only learns few axis-parallel subspaces of the randomized LDE encoding $\hat{w}$. Since LDEs have uniform ZK, this reveals no information (by our results from Section 2.1). Moreover, the ZK property of our sumcheck IOP for general base codes guarantees that in Step 4, $\mathcal{V}^*$ learns only few rows of $C(\hat{w})$, which reveal no information since C is a tensor of a ZK code (and therefore has ZK against row queries, by our results from Section 2.1). Slightly more accurately, a simulator Sim for the combined system can emulate multiple executions of the simulator Sim_Σ for the sumcheck IOP, using the simulators for C and the LDE to answer Sim_Σ's (row/axis-parallel subspace) oracle queries.

However, the actual proof is more intricate. Indeed, since the IOP for 3SAT described above involves multiple sequential calls to the internal sumcheck IOPs, the formal ZK proof requires arguing that ZK is preserved under this sequential composition. Moreover, the executions of the sumcheck IOPs with a malicious verifier $\mathcal{V}^*$ are in fact *interleaved*, because $\mathcal{V}^*$ may access an oracle message from an earlier sumcheck IOP execution in a later sumcheck IOP execution. Our ZK-IOP for sumcheck of Section 2.2 possesses the *stronger* guarantee of black-box straight-line simulation,[14] so it might be tempting to use the black-box, straight-line nature of the simulator to claim that ZK follows directly from [KLR06].

However, this implication is not immediate because: (1) [KLR06] require proving ZK *with auxiliary inputs*; and (2) their result holds in the plain model (i.e., does not account for settings – such as IOPs – in which parties have oracles). (1) can be handled by showing that in the IOP setting black-box ZK implies ZK with auxiliary inputs. This implication is known for standard ZK proofs [GO94]. The result of [GO94] does not account for oracles either, but can be extended to the IOP setting, as we show in Appendix A. (2) however is more problematic. Indeed, the result of [KLR06] is for the more general setting of *concurrent* composition of *general MPC protocols*, and extending it to the setting in which parties have oracles seems more complex than proving directly that the composed IOP for 3SAT has ZK. We therefore chose the latter route. This required careful bookkeeping because the 3 executions of the sumcheck protocol of Step 4 all use *the same* encoding $C(w)$, and so knowledge leaked on $C(w)$ via these executions may accumulate. This results in a more "correlated" execution compared to composition of standard ZKPs, and consequently makes the hybrid proof more involved (requiring a specific order in which sumcheck executions and codewords are replaced from real to simulated). However, by exploiting the *perfect* ZK property of the our sumcheck ZK-IOPs, and employing ideas from [KLR06], we were nonetheless able to make the hybrid proof work. See Section 6.5 for further details.

[14] Roughly, an IOP has black-box straight-line ZK if there exists a single PPT simulator Sim that can perfectly simulate the view of any t-restricted verifier $\mathcal{V}^*$, given oracle access to $\mathcal{V}^*$, and without rewinding the verifier.

2.4 Open Problems and Future Directions

While the focus of this work is on constructing short ZK-IOPs, we also provide new techniques for designing ZK variants of the two main building blocks underlying (ZK-)IOP constructions: tensor codes, and an IOP for the sumcheck problem. Since these primitives are ubiquitous in designing PCPs and IOPs, our techniques could potentially be used to improve other efficiency measures of these proof systems, as we now discuss.

Improved ZK-PCP Constructions. ZK incurs little to no overhead in the IOP setting, with recent ZK-IOP constructions coming close to their non-ZK counterparts in most parameters of interest. Unfortunately, as discussed in Section 1, this is not the case in the PCP setting, where ZK-PCPs incur large blowups in the proof length [KPT97, IW14, IWY16], exponential prover running time [GOS24], or a large query complexity [IKOS07, HVW21] or adaptivity [KPT97, IW14] of the honest verifier, compared to non-ZK PCPs.

Our results on tensors of ZK codes could potentially be used to construct better ZK-PCPs. More specifically, the witness encoding is a main source of leakage in PCPs (and IOPs). As discussed in Section 1.1, simply replacing the witness encoding with a *ZK* encoding does not guarantee ZK when a malicious verifier might query many codeword symbols compared to the honest verifier. Our results on tensors of ZK codes show that such tensors are ZK against a large set of *structured* queries. Thus, one could potentially achieve ZK in IOPs *and PCPs* by devising a code in which verifier queries reduce to such structured queries.

We stress that while a similar approach of "restricting" the *structure* of the verifier's queries was used before [KPT97, IW14], previous works applied a cryptographic primitive (called a "locking scheme") *on top* of the PCP, whereas the idea here would be to modify the witness encoding while preserving the algebraic structure of the underlying PCP. Thus, our investigation of properties of tensors of ZK codes might also lead to better *ZK-PCPs* whose efficiency matches the best non-ZK PCPs.

ZK-IOPs with Linear-Time Provers. Our sublinear-communication ZK-IOPs for sumcheck can be viewed as a derandomization of prior ZK-IOPs for sumcheck [BCGV16, BCF+16, BCF+17, BCG+17a, BCR+19, CHM+20, ZXZS20, BCL22]. By further exploiting the structure of the sumcheck protocol, the amount of randomness used in the protocol could potentially be further reduced, and we may also be able to obtain a ZK sumcheck with improved prover efficiency. This in turn can potentially be used to extend the linear-time (non-ZK) IOPs of [RR22] in the Boolean circuit model to the ZK setting.

Paper Organization. Preliminaries are given in Section 3. Our results on ZK properties of tensor codes are given in Section 4. Our sublinear-communication sumcheck ZK-IOPs are described in Section 5 (Figure 3), including a version with full-fledged (distributional) ZK for ZK codes (Section 5.6). Our main result on ZK-IOPs approaching witness length for 3SAT is described in Section 6 (see Figures 5- 7, including a "bare-bones" version of the protocol in Figure 4), and analyzed in Sections 6 and 7. Finally, we prove a general result on IOPs with black-box ZK in Appendix A.

3 Preliminaries

We will often view a string $x \in \Sigma^n$, over an alphabet Σ, as a function $x : [n] \to \Sigma$. In particular, the i-th entry of x is denoted $x(i)$. Similarly, for an integer $d > 1$, we will view a string $x \in \Sigma^{[n]^d}$ as a function $x : [n]^d \to \Sigma$, and will denote its entries by d-tuples $\mathbf{i} = (i_1, \ldots, i_d)$, where $i_1, \ldots, i_d \in [n]$. For $j \in [d]$, we let

$$(i_1, \ldots, i_{j-1}, *, i_{j+1}, \ldots, i_d) := \{(i_1, \ldots, i_{j-1}, \ell, i_{j+1}, \ldots, i_d) \mid \ell \in [n]\}$$

denote the axis-parallel line in direction j. For a pair of strings $x, y \in \Sigma^n$, we let $x \star y \in (\Sigma \times \Sigma)^n$ denote their *interleaving*, given by $(x \star y)(i) = (x(i), y(i))$ for any $i \in [n]$.

The relative distance between strings $x, y \in \Sigma^n$, over a finite alphabet Σ, is the fraction of coordinates $i \in [n]$ on which x and y differ, and is denoted by $\mathrm{dist}(x, y) := |\{i \in [n] : x(i) \neq y(i)\}| / n$. The relative distance of $x \in \Sigma^n$ from a non-empty set $S \subseteq \Sigma^n$ is $\mathrm{dist}(x, S) := \min_{y \in S} \mathrm{dist}(x, y)$. We say that x is ϵ-far (ϵ-close, respectively) from S if $\mathrm{dist}(x, S) > \epsilon$ ($\mathrm{dist}(x, S) \leq \epsilon$, respectively). For random variables X, Y, $X \equiv Y$ denotes that X, Y are identically distributed.

We will use finite fields extensively throughout this work. For the sake of efficient implementation of field operations, we need the field to be *constructible*:

Definition 3.1. *We say that an ensemble of finite fields $\mathcal{F} = (\mathbb{F}_n)_{n \in \mathbb{N}}$ is* constructible *if elements in $\mathbb{F}_n$ can be represented by $O(\log(|\mathbb{F}_n|))$ bits, and field operations (i.e., addition, subtraction, multiplication, inversion and sampling random elements) can all be performed in* $\mathrm{polylog}(|\mathbb{F}_n|)$ *time given this representation.*

Lemma 3.2 (see [Sho88])**.** *For every $S = S(n) \geq 1$, there exists a constructible field ensemble $\mathcal{F} = (\mathbb{F}_n)_n$, where each $\mathbb{F}_n$ has scharacteristic 2 and size $O(S(n))$.*

3.1 Interactive Oracle Proofs (IOPs) and Zero-Knowledge (ZK) IOPs

We next define the notion of *interactive oracle proofs*, due to [BCS16, RRR21]. We will give a general definition that applies to *promise problems*, which allows us to model more general settings in which the input has some particular structure (e.g., is encoded under an error-correcting code), and captures both P- and NP-languages as special cases. As we will often be interested in verifiers that run in *sub-linear time*, we will also view the input as being separated into two parts $x = (x_{\mathrm{exp}}, x_{\mathrm{imp}})$, where the verifier explicitly reads the explicit input x_{exp}, but only has *oracle access* to the implicit input x_{imp} (we will sometimes consider languages in which either the explicit or the implicit inputs are empty). We will also consider only *public-coin* IOPs (in which the verifier does not have any private randomness that is kept hidden from the prover) since all the IOPs we construct in this work will have this feature.

Notation 3.3. *We associate (NP) languages and relations to promise problems in the natural way. Specifically, for a language $\mathcal{L}$ we define the corresponding promise problem to be $\mathsf{YES}_{\mathcal{L}} := \mathcal{L}$, and $\mathsf{NO}_{\mathcal{L}} := \Sigma^* \setminus \mathcal{L}$. Similarly, for an NP relation $\mathcal{R} = \mathcal{R}(x, w)$ we define the corresponding promise problem to be $\mathsf{YES}_{\mathcal{R}} := \{x : \exists\, w \text{ s.t.} (x, w) \in \mathcal{R}\}$ and $\mathsf{NO}_{\mathcal{R}} := \{x : \forall w \text{ s.t. } |w| \leq p(|x|), (x, w) \notin \mathcal{R}\}$, where $p(n)$ is the polynomial such that for every $(x, w) \in \mathcal{R}$ we have $|w| \leq p(|x|)$.*

Definition 3.4 (Interactive Oracle Proof (IOP))**.** *An ℓ-*round (public coin) Interactive Oracle Proof (IOP) with soundness error ϵ for a promise problem $(\mathsf{YES}, \mathsf{NO})$ *is a pair $(\mathcal{P}, \mathcal{V})$ of probabilistic algorithms satisfying the following properties.*

- **Syntax.**

– ***Inputs:*** *Prover $\mathcal{P}$ receives as input $x = (x_{\text{exp}}, x_{\text{imp}})$. Verifier $\mathcal{V}$ receives as input x_{exp} and $1^{|x_{\text{imp}}|}$,*[15] *and is also given oracle access to x_{imp}.*

– **Protocol Structure.** *The protocol consists of a* Communication Phase *followed by a* Query Phase*. In the* Communication phase*, $\mathcal{P}$ and $\mathcal{V}$ interact for ℓ rounds, where $\mathcal{V}$'s messages are uniformly random strings $R_1, \ldots, R_\ell$, and each of $\mathcal{P}$'s messages is either an* explicit *message or an* oracle *message. In the* Query phase*, $\mathcal{V}$ reads $\mathcal{P}$'s explicit messages in full, and makes (non-adaptive) queries to the implicit input x_{imp} and to $\mathcal{P}$'s oracle messages. These queries are determined solely by the explicit input x_{exp} and $\mathcal{V}$'s randomness string $R = (R_1, \ldots, R_\ell)$. $\mathcal{V}$ then deterministically decides whether to accept or reject based on the answers to these queries, the explicit input x_{exp}, and the randomness strings $R_1, \ldots, R_l$.*

- **Semantics.** *The protocol satisfies the following semantic properties:*

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

  – **Soundness:** *If $x \in \mathsf{NO}$, then for any (possibly unbounded) prover strategy $\mathcal{P}^*$, when $\mathcal{V}$ interacts with $\mathcal{P}^*$, it accepts with probability at most ϵ.*

Remark 3.5 (IOPs with Pre-Processing)**.** *It will sometimes be useful to incorporate a local randomized* pre-processing *phase into the IOP execution. Specifically, in the pre-processing phase the verifier uses x_{exp} to generate a polynomially-long string, which she can later query during the query phase. In this work, we use a relatively weak form of pre-processing. Specifically, pre-processing is with public randomness, namely the randomness used for pre-processing is known to the prover (this can be achieved by adding a round in which the verifier sends this randomness to the prover, or includes it as part of her first message to the prover), and the locations of the verifier's queries to the pre-processed string depend solely on the randomness of $\mathcal{V}$. To distinguish IOPs with a pre-processing phase from standard IOPs (without pre-processing), we call the former a* pre-processing IOP*.*

Remark 3.6. *While IOPs are usually defined for P or NP languages, we chose to define these using promise prolems, as this will be useful for our construction of an IOP for the Sumcheck problem. Definition 3.4 captures the standard IOP definition for a language $\mathcal{L}$ or an NP relation $\mathcal{R}$ as a special case, by using the corresponding promise problems $(\mathsf{YES}_{\mathcal{L}}, \mathsf{NO}_{\mathcal{L}})$ and $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$, respectively (see Notation 3.3).*

Parameters of Interest. The key parameters associated with an IOP system are:

- **Round complexity:** the number of rounds ℓ.
- **Query Complexity:** the number of bits q that the verifier reads from the implicit input x_{imp} and $\mathcal{P}$'s messages (and the pre-processed string, in case of a pre-processing IOP).
- **Communication complexity:** the total length cc of $\mathcal{P}$'s messages. This is also referred to as the "proof length".
- **Alphabet size:** the size of the alphabet Σ over which the IOP is defined.
- **Verifier running time:** the total time $T_{\mathcal{V}}$ it takes for $\mathcal{V}$ to generate its randomness string $R = (R_1, \ldots, R_\ell)$, compute the query locations (given the explicit input x_{exp} and the randomness

[15] $\mathcal{V}$ needs input $1^{|x_{\text{imp}}|}$ in cases where $|x_{\text{exp}}| \ll |x_{\text{imp}}|$ (for example, this is the case in our sumcheck ZK-IOP of Section 5). In other cases, $|x_{\text{imp}}| = \text{poly}(|x_{\text{exp}}|)$, in which case $1^{|x_{\text{imp}}|}$ can be omitted from $\mathcal{V}$'s input (this is the case in our ZK-IOP for SAT of Section 6).

string R), and decide whether to accept or reject based on the query values, the explicit input x_{exp}, and the randomness string R. (We note that for pre-processing IOPs, the running time of preprocessing is accounted for separately.)

- **Preprocessing running time:** the total time T_{Pre} it takes for $\mathcal{V}$ to generate the preprocessed string.
- **Prover running time:** the total time $T_{\mathcal{P}}$ it takes for $\mathcal{P}$ to compute its messages. In the context of interactive oracle proofs for NP languages, we assume that the prover is also given as an auxiliary input a witness w proving that the input x belongs to the language.

We note that the notion of a PCP corresponds to the special case of an IOP, where the round complexity is $\ell = 1$.

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

Definition 3.7 (Interactive Oracle Proof of Proximity (IOPP))**.** *An ℓ-round IOP of α-proximity (α-IOPP) with soundness error ϵ for a pair language $\mathcal{L} \subseteq \{(x_{\text{exp}}, x_{\text{imp}}) \in \Sigma^* \times \Sigma^*\}$ is an ℓ-round IOP with soundness error ϵ for the promise problem $(\mathsf{YES}, \mathsf{NO})$, where $\mathsf{YES} = \mathcal{L}$ and $\mathsf{NO} = \{(x_{\text{exp}}, y) : y \text{ is } \alpha\text{-far from } \mathcal{L}_{x_{\text{exp}}}\}$.*

We refer to α as the proximity parameter of the IOPP. If $\mathcal{L}$ has no explicit input, then an ℓ-round α-IOPP with soundness error ϵ for $\mathcal{L}$ is an ℓ-round IOP with soundness error ϵ for the promise problem $(\mathsf{YES}, \mathsf{NO})$, where $\mathsf{YES} = \mathcal{L}$ and $\mathsf{NO} = \{y : y \text{ is } \alpha\text{-far from } \mathcal{L}\}$. Once more, we note that the notion of a PCPP corresponds to the special case of an IOPP, when the round complexity is $\ell = 1$.

For reducing our query complexity via composition, we shall use the following PCPP due to [Mie09].

Theorem 3.8 ([Mie09, Theorem 1])**.** *Let $\mathcal{L} \subseteq \Sigma^* \times \Sigma^*$ be a pair language decidable in time $T = T(m+n)$, where m, n are the explicit and implicit input lengths, respectively. Then for any constant $\alpha > 0$, there exists an α-PCPP over Σ^* with soundness error $\frac{1}{2}$ for $\mathcal{L}$ with length $\tilde{O}(T)$, constant query complexity, verifier running time $\mathsf{poly}(m, \log n, \log(T))$, and prover running time $\mathsf{poly}(m, n, T)$.*

Remark 3.9. *We remark that [Mie09, Theorem 1] does not explicitly state the prover's running time but it can be verified that the prover can be implemented in polynomial time. We also note that follow-up works obtain prover running time that is* quasi-linear *(rather than merely polynomial) in the original computation [BCGT13].*

3.1.1 IOPs with Zero-Knowledge

We now turn to defining a special case of IOPs with a *zero-knowledge* guarantee (in the simulation-based paradigm). Intuitively, these are IOPs for NP, in which *query-restricted* verifiers – that are restricted in the number of queries they can make to their oracles – learn nothing about the NP witness, and only learn few *physical* bits of x_{imp}. In particular, this notion naturally scales down to languages in P with a non-empty x_{imp} part. We first give some necessary preliminary definitions.

Definition 3.10 (t-Restricted Adversary)**.** *We say that an algorithm $\mathcal{A}$ with oracle access to oracles $\mathcal{O}_1, \ldots, \mathcal{O}_\ell$ is t-restricted if $\mathcal{A}$ makes at most t queries (in total) to $\mathcal{O}_1, \ldots, \mathcal{O}_\ell$.*

Definition 3.11 (Verifier View). *Let $\mathcal{R} = \mathcal{R}(x,w)$ be an NP relation with a corresponding promise problem $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$ (see Notation 3.3). Let $(\mathcal{P},\mathcal{V})$ be an IOP for $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$, and let $\mathcal{V}^*$ be a (possibly malicious) verifier interacting with $\mathcal{P}$ in the protocol. For every $((x_{\mathrm{exp}}, x_{\mathrm{imp}}), w) \in \mathcal{R}$, the* view $\mathsf{View}_{\mathcal{V}^*}(x_{\mathrm{exp}}, x_{\mathrm{imp}}, w)$ *in the interaction with $\mathcal{P}$ consists of $x_{\mathrm{exp}}, 1^{|x_{\mathrm{imp}}|}$, $\mathcal{V}^*$'s random coins, $\mathcal{P}$'s explicit messages, and the answers to $\mathcal{V}^*$'s queries to x_{imp} and $\mathcal{P}$'s oracle messages. Notice that $\mathsf{View}_{\mathcal{V}^*}(x_{\mathrm{exp}}, x_{\mathrm{imp}}, w)$ is a random variable which depends on the random coins of $\mathcal{P}$ and $\mathcal{V}^*$.*

We will distinguish between *adaptive* and *non-adaptive* verifiers. In this work, all our constructions have the feature that the honest verifier is non-adaptive, and zero-knowledge holds against adaptive malicious verifiers. We now formally define these notions:

Definition 3.12 (Adaptive and Non-Adaptive Verifiers). *We say that a verifier $\mathcal{V}^*$ with oracle access to $(\mathcal{O}_1, \ldots \mathcal{O}_\ell)$ is* non-adaptive *if her oracle queries are determined solely by her input and randomness (in particular, without loss of generality $\mathcal{V}^*$ makes all her oracle queries in a single round). Otherwise, we say that $\mathcal{V}^*$ is* adaptive *(in particular, each query of an adaptive $\mathcal{V}^*$ might depend on oracle answers to previous queries).*

Remark 3.13. *We note that by default, the honest IOP verifier is non-adaptive (this follows from Definition 3.4), and indeed the honest verifiers in all the ZK-IOP constructions described in this work are non-adaptive. However, the* malicious *verifier in the ZK properties described below might be adaptive.*

In this work we use several flavors of zero-knowledge, all in the simulation-based paradigm. The most commonly used is the following (stand-alone) ZK definition:

Definition 3.14 (Zero-Knowledge IOP (ZK-IOP)). *Let $t \in \mathbb{N}$ be a ZK parameter, and let $\mathcal{R} = \mathcal{R}(x,w)$ be an NP relation with a corresponding promise problem $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$. We say that an IOP $(\mathcal{P},\mathcal{V})$ for $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$ has t-ZK if for every t-restricted verifier $\mathcal{V}^*$ there exists a PPT simulator* Sim *such that for every $((x_{\mathrm{exp}}, x_{\mathrm{imp}}), w) \in \mathcal{R}$ we have*

$$\left(\mathsf{View}_{\mathcal{V}^*}(x_{\mathrm{exp}}, x_{\mathrm{imp}}, w), q_V\right)_{r_P, r_V} \equiv \left(r_V, \mathsf{Sim}^{x_{\mathrm{imp}}, \mathcal{V}^*(\cdot; r_V)}\left(x_{\mathrm{exp}}, 1^{|x_{\mathrm{imp}}|}\right), q_S\right)_{r_V, r_{\mathsf{Sim}}} \tag{1}$$

where $r_P, r_V, r_{\mathsf{Sim}}$ are the random coins of $\mathcal{P}, \mathcal{V}^*$ and Sim *(respectively); q_V denotes the number of queries which $\mathcal{V}^*$ makes to all her oracles (i.e., x_{imp} and $\mathcal{P}$'s oracle messages); and q_S denotes the number of queries which* Sim *makes to x_{imp}. In this case, we say that $(\mathcal{P},\mathcal{V})$ is a t-*ZK-IOP *for $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$.*

We say that $(\mathcal{P},\mathcal{V})$ has Honest-Verifier ZK (HVZK)*, denoted* HVZK-IOP*, if such a simulator* Sim *exists only for the* honest *verifier $\mathcal{V}$.*

Remarks on Definition 3.14. A few remarks are in order. First, notice that the verifier's randomness r_V is *not sampled by the simulator*, and not even given to it as input, but rather it is *prepended to* Sim*'s output*. This is because the (possibly computationally-unbounded) $\mathcal{V}^*$ might use super-polynomially many bits, and so these cannot be sampled by the PPT simulator. Instead, following previous works, $\mathcal{V}^*$'s randomness is sampled as in the real world and prepended to Sim's output. Second, Sim is *given oracle access to* $\mathcal{V}^*$ (with randomness r_V). This is inherent even though the order of quantifiers in the definition allows the description of Sim to depend on $\mathcal{V}^*$ (namely, Sim might have $\mathcal{V}^*$'s code hard-wired into it). Indeed, to simulate $\mathcal{V}^*$'s view, Sim needs to generate the messages which $\mathcal{P}$ sends to $\mathcal{V}^*$. Since these messages might depend on $\mathcal{V}^*$'s messages to $\mathcal{P}$, then Sim needs to generate $\mathcal{V}^*$'s messages on its own. While these messages are fully determined by $\mathcal{V}^*$'s code, randomness, and input, one might not be able to generate them *efficiently* (indeed, $\mathcal{V}^*$ might be computationally unbounded). Therefore, to enable Sim to determine the verifier's

messages, we give it oracle access to $\mathcal{V}^*$. We note that previous works defined ZK-IOPs without giving Sim oracle access to $\mathcal{V}^*$. However, their IOP constructions achieved a *stronger* ZK guarantee (specifically, black-box ZK, see Definition 3.15 below) in which the simulator is anyway given oracle access to $\mathcal{V}^*$. Finally, the simulator of Definition 3.14 is allowed to make oracle queries to x_{imp}. This is inherent because $\mathcal{V}^*$ may query x_{imp} directly. However, we restrict Sim's access to x_{imp}: it can only make as many queries as $\mathcal{V}^*$ makes *in total* to both x_{imp} and $\mathcal{P}$'s oracle messages. This is standard in the literature on zero-knowledge PCPPs and IOPs, e.g., in [IW14, BCF+16].

Next, we define the *stronger* notion of universal *black-box, straight-line* ZK. All ZK-IOP constructions provided in this work satisfy this stronger ZK property.

Definition 3.15 (Black-Box Straight-Line ZK)**.** *Let $t \in \mathbb{N}$ be a ZK parameter, and let $\mathcal{R} = \mathcal{R}(x,w)$ be an NP relation with a corresponding promise problem $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$. We say that an IOP $(\mathcal{P},\mathcal{V})$ for $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$ has* Black-Box (BB) Straight-Line t-ZK *if there exists a PPT simulator* Sim *such that for every t-restricted verifier $\mathcal{V}^*$ and every $((x_{\text{exp}}, x_{\text{imp}}), w) \in \mathcal{R}$, the following two distributions are identical.*

- $\left(\mathsf{View}_{\mathcal{V}^*}\left(x_{\text{exp}}, x_{\text{imp}}, w\right), q_V\right)_{r_P, r_V}$*: $\mathcal{V}^*$'s view when interacting with $\mathcal{P}$, where r_P, r_V are the random coins of $\mathcal{P}, \mathcal{V}^*$ (respectively), and q_V denotes the number of queries which $\mathcal{V}^*$ makes to all her oracles (i.e., x_{imp} and the oracle messages which $\mathcal{P}$ sent).*
- $\left(\mathsf{View}^{\mathsf{Sim}}_{\mathcal{V}^*}\left(x_{\text{exp}}, x_{\text{imp}}, w\right), q_{\mathsf{Sim}}\right)_{r_V, r_{\mathsf{Sim}}}$*: $\mathcal{V}^*$'s view when it interacts with* Sim *instead of with $\mathcal{P}$ and its oracles (namely, the prover messages are generated by* Sim*, and $\mathcal{V}^*$'s oracle queries are also answered by* Sim*), where r_V, r_{Sim} are the random coins of $\mathcal{V}^*$,* Sim *(respectively), and q_S denotes the number of queries which* Sim *makes to x_{imp}.*

Claim 3.16 (BB Straight-Line ZK implies ZK)**.** *If $(\mathcal{P},\mathcal{V})$ is an* IOP *with black-box straight-line t-ZK, the it has t-ZK.*

Proof: The proof follows from the fact that a simulator Sim for the black-box straight-line t-ZK property of $(\mathcal{P},\mathcal{V})$ can generate a simulated view by interacting with $\mathcal{V}^*$ (in a black-box, straight-line manner), to generate the transcript of the interaction (which includes the verifier's view, except for her randomness). The simulation is perfect because Sim's simulated answers perfectly emulate the prover's messages and oracle answers. ■

Our ZK-IOP for SAT (Section 6) will be constructed by composing several ZK-IOPs. To guarantee that ZK is preserved under this composition, we will need the following notion of *ZK with auxiliary inputs*, which generalizes the standard definition of ZK with auxiliary inputs for *interactive zero-knowledge proofs* (e.g., from [Gol01]) to the IOP setting.

Definition 3.17 (Zero-Knowledge with Auxiliary Inputs)**.** *Let $t \in \mathbb{N}$, and let $\mathcal{R} = \mathcal{R}(x,w)$ be an NP relation with a corresponding promise problem $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$. We say that an IOP $(\mathcal{P},\mathcal{V})$ for $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$ has t-*ZK with auxiliary inputs *if for every t-restricted verifier $\mathcal{V}^*$ there exist a polynomial $p(n)$, and a PPT simulator* Sim*, such that for every $((x_{\text{exp}}, x_{\text{imp}}), w) \in \mathcal{R}$, and for every* auxiliary input *$z \in \{0,1\}^{p(|x_{\text{exp}}|+|x_{\text{imp}}|)}$ we have*

$$\left(\mathsf{View}'_{\mathcal{V}^*}\left(x_{\text{exp}}, x_{\text{imp}}, w, z\right), q_V\right)_{r_P, r_V} \equiv \left(r_V, \mathsf{Sim}^{x_{\text{imp}}, \mathcal{V}^*(\cdot, r_V)}\left(x_{\text{exp}}, 1^{|x_{\text{imp}}|}, z\right), q_S\right)_{r_V, r_{\mathsf{Sim}}}$$

where $r_P, r_V, r_{\mathsf{Sim}}$ are the random coins of $\mathcal{P}, \mathcal{V}^*$ and Sim *(respectively); q_V denotes the number of queries which $\mathcal{V}^*$ makes to all her oracles (i.e., x_{imp} and the oracle messages which $\mathcal{P}$ sent); q_S denotes the number of queries which* Sim *makes to x_{imp}; and $\mathsf{View}'_{\mathcal{V}^*}\left(x_{\text{exp}}, x_{\text{imp}}, w, z\right)$ is defined similarly to Definition 3.11, but includes also the auxiliary input z.*

Remark on Definition 3.17. Notice that Definition 3.17 restricts the length of the auxiliary input z to be polynomial in the total length of the (explicit and implicit) input, whereas in standard ZK proofs with auxiliary inputs, no restriction is made on the length of z. However, such a bound exists implicitly also in standard ZK proofs, since there ZK is defined for *PPT* verifiers, and so $\mathcal{V}^*$ can only read a polynomial number of bits from her auxiliary input. Moreover, such a bound is necessary for IOPs since ZK is defined with respect to *unbounded* verifiers $\mathcal{V}^*$. Thus, if $|z|$ is not a-priori bounded by some polynomial, then the PPT Sim might not even be able to read all the bits of z which the (unbounded) verifier $\mathcal{V}^*$ reads.

We further note that ZK with auxiliary inputs is used (in the literature on ZK proofs, and in this paper) to allow for composition of ZK proofs/IOPs, where the auxiliary input is used to provide the verifier in an execution of the system with the view of verifiers in (say) previous executions of the system. These views include the verifiers' randomness, the messages received from the honest $\mathcal{P}$, and the oracle answers to the verifiers' queries. The latter two are bounded (by the polynomial time bound on the honest prover, and the query restriction on the verifiers). The former can be eliminated by having the verifier use its own randomness to emulate the verifiers in previous executions of the system, and then fixing an optimal choice of randomness for these emulations (and using an averaging argument).

Finally, we note that the notion of ZK with auxiliary inputs extends to the setting of black-box, straight-line ZK in the natural way, in which case the simulator $\mathsf{Sim}_{\mathsf{aux}}$ is not given z as input, and instead has oracle access to $\mathcal{V}^*$ with auxiliary input z.This is consistent with definitions of black-box ZK with auxiliary inputs for *interactive proofs* (see, e.g., [Gol01, Sec. 4.5.4.2]).[16]

3.2 Error-correcting codes

Let Σ be a finite alphabet, and k, n be positive integers (the message length and the block length, respectively). An (error-correcting) code is an injective map $C : \Sigma^k \to \Sigma^n$. The elements in the domain of C are called messages, and the elements in the image of C are called codewords. The rate of a code $C : \Sigma^k \to \Sigma^n$ is the ratio $\rho := \frac{k}{n}$. The relative distance $\mathrm{dist}(C)$ of C is the maximum $\delta > 0$ such that for every pair of distinct messages $\mathsf{m}, \mathsf{m}' \in \Sigma^k$ it holds that $\mathrm{dist}(C(\mathsf{m}), C(\mathsf{m}')) \geq \delta$.

We say that a code $C : \Sigma^k \to \Sigma^n$ is systematic if the message is a prefix of the corresponding codeword, i.e., for every $\mathsf{m} \in \Sigma^k$ there exists $z \in \Sigma^{n-k}$ such that $C(\mathsf{m}) = (\mathsf{m}, z)$. If $\Sigma = \mathbb{F}$ for some finite field $\mathbb{F}$, and C is a linear map between the vector spaces $\mathbb{F}^k$ and $\mathbb{F}^n$, then we say that C is linear. A generating matrix for a linear code $C : \mathbb{F}^k \to \mathbb{F}^n$ is an $n \times k$ matrix G over $\mathbb{F}$ so that $C(\mathsf{m}) = G \cdot \mathsf{m}$ for any $\mathsf{m} \in \mathbb{F}^k$. A parity-check matrix for C is an $(n-k) \times n$ matrix H over $\mathbb{F}$ so that $H \cdot y = 0$ for $y \in \mathbb{F}^n$ if and only if y is a codeword of C.

In this work we will typically want error-correcting codes that are defined for an infinite sequence of message lengths $\mathcal{I} \subseteq \mathbb{N}$. Thus, a code ensemble $\mathcal{C} = \{C_k : (\Sigma_k)^k \to (\Sigma_k)^n\}_{k \in \mathcal{I}}$ is a countable collection of error correcting codes, one for each message length $k \in \mathcal{I}$. We say that $\mathcal{C}$ has rate $\rho \in (0,1)$ and relative distance $\delta \in (0,1)$ if for any $k \in \mathcal{I}$, the code C_k has rate at least ρ and relative distance at least δ. We say that a code ensemble $\mathcal{C}$ is explicit if for any $k \in \mathcal{I}$, $C_k(\mathsf{m})$ can be computed in time $\mathsf{poly}(k)$ for any $\mathsf{m} \in (\Sigma_k)^k$. For linear codes, this is equivalent to the requirement that the generator matrix for C_k can be computed in time $\mathsf{poly}(k)$, and also implies that a

[16] Giving z to $\mathsf{Sim}'_{\mathsf{aux}}$ might violate the essence of black-box simulation, as observed in [KLR06]. Indeed, think of a "universal" verifier $\mathcal{V}_U$ who interprets its auxiliary input as the description of a verifier, which $\mathcal{V}_U$ then proceeds to emulate. In this case, giving z to the simulator in essence gives the simulator the description of the verifier it needs to simulate.

parity-check matrix for C_k can be computed in time $\mathsf{poly}(k)$. We say that a linear code ensemble $\mathcal{C}$ has an efficient randomized construction if the generator matrix for C_k can be found in time $\mathsf{poly}(k)$ with $\mathsf{negl}(k)$ failure probability (which also implies that a parity-check matrix for C_k can be found in time $\mathsf{poly}(k)$). For better readability, we sometimes abuse notation and write $\mathcal{C} : \Sigma^k \to \Sigma^n$ to denote that $\mathcal{C} = \{C_k : (\Sigma_k)^k \to (\Sigma_k)^n\}_{k \in \mathcal{I}}$.

3.2.1 Zero-Knowledge Codes

We now define a special type of codes with Zero-Knowledge (ZK) guarantees. Intuitively, these are codes associated with a *randomized* encoding function, in which few codeword symbols reveal no information about the message. We will use two flavors of this property, which hold against adaptive and non-adaptive adversaries. We first define the notion of a randomized encoding function.

Definition 3.18 (Randomized Encoding Function)**.** *Let $C : \Sigma^k \to \Sigma^n$ be a code. A* randomized encoding function for *C is a random map $\mathsf{Enc} : \Sigma^{k'} \to \mathrm{image}(C)$, which satisfies that for any $\mathsf{m} \neq \mathsf{m}' \in \Sigma^{k'}$, the distributions $\mathsf{Enc}(\mathsf{m})$ and $\mathsf{Enc}(\mathsf{m}')$ have disjoint supports.*

*Let $k' < k$ be a parameter. The k'-*randomized encoding function for *C is a randomized encoding function $\mathsf{Enc} : \Sigma^{k'} \to \Sigma^n$ which on input $\mathsf{m} \in \Sigma^{k'}$, samples $r \leftarrow \Sigma^{k-k'}$, and outputs $C(\mathsf{m}; r)$.*

The stronger flavor of ZK – which will be the default version we will use – requires ZK to hold against *adaptive* adversaries, that may make several rounds of queries to their oracle. In particular, each oracle query might dependent on the answers to previous oracle queries. The weaker notion of ZK holds only against *non-adaptive* adversaries, whose queries are determined solely by their input and randomness (independent of the oracle answers to previous queries). The formal definitions are similar to Definition 3.12.

Definition 3.19 (Zero-Knowledge (ZK) Code)**.** *Let $\mathcal{C} : \Sigma^k \to \Sigma^n$ be a code ensemble, and let $\mathsf{Enc} : \Sigma^{k'} \to \Sigma^n$ be a randomized encoding function for $\mathcal{C}$. We say that $\mathcal{C}$ has t-*ZK with respect to Enc *if there exists a PPT simulator Sim that takes as input $1^k, k'$,such that for every (possibly adaptive) t-restricted adversary $\mathcal{A}$, and every $\mathsf{m} \in \Sigma^{k'}$, the following distributions are identically distributed.*

- *The view $\mathsf{View}^R_{\mathcal{A}}(\mathsf{m})$ of $\mathcal{A}$ on input $1^k, k'$,and given oracle access to a random encoding $c \leftarrow \mathsf{Enc}(\mathsf{m})$.*
- *The view $\mathsf{View}^S_{\mathcal{A}}(\mathsf{m})$ of $\mathcal{A}$ on input $1^k, k'$, where $\mathcal{A}$'s oracle queries are answered by Sim.*[17]

If the above holds only against non-adaptive *t-restricted adversaries $\mathcal{A}$, then we say that the code has* non-adaptive *t-*ZK.

Remark 3.20. *We note that non-adaptive ZK codes were defined slightly differently in [ISVW13], requiring that for every $I \subseteq [n]$ of size $|I| \leq t$, and every pair $\mathsf{m}, \mathsf{m}' \in \Sigma^{k'}$ of messages, it holds that $\mathsf{Enc}(\mathsf{m})|_I \equiv \mathsf{Enc}(\mathsf{m}')|_I$. (Ishai et al. [ISVW13] also considered a statistical variant of ZK, but we only define the perfect version which we use in this work.) This was shown in [BCL22] to be equivalent to the ZK property of Definition 3.19 (using a simulator that answers the verifier queries via $\mathsf{Enc}(0)$).*[18]

We will also need a weaker version of a ZK code, in which the code is ZK only on a *subset of messages*. Formally,

[17] Notice that the simulator is black-box and straight-line; this is standard in the literature on ZK codes (see, e.g., [BCL22]).

[18] This equivalence holds when the code has a polynomial-time encoding function, which is the case for all codes considered in this work.

Definition 3.21 (ZK on a Subset of Messages). *Let $\mathcal{C} : \Sigma^k \to \Sigma^n$ be a code ensemble, and let* $\mathsf{Enc} : \Sigma^{k'} \to \Sigma^n$ *be a randomized encoding function for $\mathcal{C}$. For a set $S \subseteq \Sigma^{k'}$, we say that $\mathcal{C}$ has t-*ZK on S with respect to Enc *if Definition 3.19 holds for $\mathcal{C}$ when restricting the message* m *to satisfy* $\mathsf{m} \in S$.

An important special case of ZK codes, is uniform ZK codes in which any t coordinates in the randomized encoding are uniformly random.

Definition 3.22 (Uniform ZK Codes). *Let $C : \Sigma^k \to \Sigma^n$ be a code, and let* $\mathsf{Enc} : \Sigma^{k'} \to C$ *be a randomized encoding function for C. We say that C has t-*uniform ZK *with respect to* Enc *if for every* $\mathsf{m} \in \Sigma^{k'}$*, and every $I \subseteq [n]$ of size $|I| = t$, $c|_I$ is a uniformly random string in Σ^t, where $c \leftarrow \mathsf{Enc}(\mathsf{m})$.*

Distributional ZK-IOPs with Respect to ZK Codes. We also combine the notions of codes with a randomized encoding function and ZK-IOPs, and give a *distributional* ZK notion for IOPs which has an average-case flavor (compared to the worst-case definitions given in Section 3.1.1 above). Roughly, distributional ZK is defined with respect to a code and a randomized encoding function, and ZK is guaranteed only for implicit inputs which are random encodings of some message. The advantage of the average-case definition is that it potentially enables us to achieve a *stronger* ZK guarantee (We note that the distributional ZK definition naturally extends to black-box straight-line simulation, as in Definition 3.15.).

We are now ready to define distributional ZK for IOPs. In the following definition, the codeword c is the implicit input, meaning $\mathcal{V}^*$ has oracle access to c. Notice that unlike the worst-case ZK definitions on Section 3.1.1, the simulator Sim is not given access to c.

Definition 3.23 (Distributional Zero-Knowledge). *Let $\mathcal{R} = \mathcal{R}(x, w)$ be an NP relation with a corresponding promise problem $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$, and let $\mathcal{C}$ be a code with a randomized encoding function* Enc*. We say that an IOP $(\mathcal{P}, \mathcal{V})$ for $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$ has* distributional t-ZK with respect to $\mathcal{C}$ and Enc *if for every t-restricted verifier $\mathcal{V}^*$ there exists a PPT simulator* Sim *such that for every $(x_{\text{exp}}, \mathsf{m}, w)$ so that $\mathsf{Supp}(x_{\text{exp}}, \mathsf{Enc}(\mathsf{m}), w) \subseteq \mathsf{YES}_{\mathcal{R}}$, the following distributions are identical:*

- $\left(\mathsf{View}_{\mathcal{V}^*}\left(x_{\text{exp}}, c, w\right)\right)_{c \leftarrow \mathsf{Enc}(\mathsf{m}), r_P, r_V}$*, where r_P, r_V are the random coins of $\mathcal{P}, \mathcal{V}^*$ (respectively).*
- $\left(r_V, \mathsf{Sim}^{\mathcal{V}^*(\cdot; r_V)}\left(x_{\text{exp}}, 1^{|c|}\right)\right)_{c \leftarrow \mathsf{Enc}(\mathsf{m}), r_V, r_{\mathsf{Sim}}}$*, where r_V, r_{Sim} are the random coins of $\mathcal{V}^*$ and* Sim *(respectively).*

3.2.2 Locally Testable Codes

Intuitively, a code is said to be *locally testable* if, given a string $w \in \Sigma^n$, it is possible to determine whether w is a codeword of C, or rather far from C, by reading only a small part of w. There are two variants of LTCs in the literature, "weak" LTCs and "strong" LTCs, where the main difference is that weak LTCs are required to reject only words which are sufficiently far from C, while strong LTCs are required to reject any word w not in C with probability proportional to the relative distance of w from C. In this work, we will work exclusively with strong LTCs, since it is a simpler notion. We shall also consider an even stronger notion of *robust local testing*, in which if w is far from C, then the view of the local tester is far, on average, from an accepting view.

Definition 3.24 (Locally-Testable Code (LTC)). *We say that a code $C : \Sigma^k \to \Sigma^n$ is a (non-adaptive) q-*locally testable code *((q, μ)-LTC) if there exists a randomized oracle algorithm* TEST *which satisfies the following properties:*

- ***Input:*** TEST *gets oracle access to a string $w \in \Sigma^n$.*

- ***Query phase:*** TEST *picks a random subset* $I \subseteq [n]$ *of size* $|I| = q$*, according to some distribution, and queries* $w|_I$*.*
- ***Decision phase:*** TEST *outputs* $\psi_{\mathrm{TEST}}(w|_I)$ *for some predicate* $\psi_{\mathrm{TEST}} : \Sigma^q \to \{\mathsf{ACCEPT}, \mathsf{REJECT}\}$*.*
- ***Completeness:*** *If* w *is a codeword of* C*, then* TEST *accepts with probability* 1*.*
- ***Soundness:*** *If* w *is not a codeword of* C*, then* TEST *rejects with probability at least* $\mu \cdot \mathrm{dist}(w, C)$*.*

We say that C *is a* (q, μ)-robust locally testable code ((q, μ)-robust LTC) *if there exists a randomized oracle algorithm* TEST *which satisfies the above properties, except that the soundness condition is replaced with the following robustness condition:*

- ***Robustness:*** *If* w *is not a codeword of* C*, then, in expectation,* $w|_I$ *is* $(\mu \cdot \mathrm{dist}(w, C))$*-far from* $\psi^{-1}_{\mathrm{TEST}}(\mathsf{ACCEPT})$*.*

We say that the algorithm TEST *is a* (robust) local tester *of* C*. We say that a code ensemble* $\mathcal{C} : \Sigma^k \to \Sigma^n$ *is an* efficient (q, μ)-LTC (robust (q, μ)-LTC, resp.) *if there is a randomized oracle algorithm that on input* 1^k*, computes a* (q, μ)*-local tester (robust* (q, μ)*-local tester, resp.) for* $\mathcal{C}$ *in time* $\mathsf{poly}(k, \log(|\Sigma|))$*.*

3.2.3 Tensor codes

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

Definition 3.25 (Tensor Product Code)**.** *The* tensor (product) code *of linear codes* $C_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ *and* $C_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ *is the code* $C_1 \otimes C_2 : \mathbb{F}^{k_1 \times k_2} \to \mathbb{F}^{n_1 \times n_2}$*, where the encoding* $(C_1 \otimes C_2)(M)$ *of any message* $M \in \mathbb{F}^{k_1 \times k_2}$ *is obtained by first encoding each column of* M *with the code* C_1*, and then encoding each resulting row with the code* C_2*.*

Note that by linearity, the codewords of $C_1 \otimes C_2$ are $n_1 \times n_2$ matrices (over the field $\mathbb{F}$) whose columns belong to the code C_1, and whose rows belong to the code C_2. It is also known that the converse is true: any $n_1 \times n_2$ matrix, whose columns belong to the code C_1, and whose rows belong to the code C_2, is a codeword of $C_1 \otimes C_2$. Furthermore, swapping the order of encodings, encoding first each row of M with the code C_2, and then encoding each resulting column with the code C_1, results in the same codeword.

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

Fact 3.26. *Suppose that* $C_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$*,* $C_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ *are linear codes of rates* ρ_1, ρ_2 *and relative distances* δ_1, δ_2 *that can be encoded in times* T_1, T_2*, respectively. Then the tensor code* $C_1 \otimes C_2$ *is a linear code of rate* $\rho_1 \cdot \rho_2$ *and relative distance* $\delta_1 \cdot \delta_2$ *that can be encoded in time* $n_2 \cdot T_1 + n_1 \cdot T_2$*.*

For a linear code $C : \mathbb{F}^k \to \mathbb{F}^n$, let $C^{\otimes 1} := C$ and $C^{\otimes d} := C \otimes C^{\otimes(d-1)}$, for any $d \geq 2$. As in the 2-dimensional case, the codewords of $C^{\otimes d} : \mathbb{F}^{[k]^d} \to \mathbb{F}^{[n]^d}$ can be viewed as d-dimensional cubes, satisfying that their projection on any axis-parallel line is a codeword of C. Once more, we have that the converse is also true. Moreover, applying Fact 3.26 inductively yields the following corollary.

Corollary 3.27. *Suppose that* $C : \mathbb{F}^k \to \mathbb{F}^n$ *is a linear code of rate* ρ *and relative distance* δ *that can be encoded in time* T*. Then* $C^{\otimes d} : \mathbb{F}^{[k]^d} \to \mathbb{F}^{[n]^d}$ *is a linear code of rate* ρ^d *and relative distance* δ^d *that can be encoded in time* $td \cdot (n)^{d-1} \cdot T$*.*

For a pair of vectors $\lambda_1 \in \mathbb{F}^{k_1}$ and $\lambda_2 \in \mathbb{F}^{k_2}$, we denote by $\lambda_1 \otimes \lambda_2 \in \mathbb{F}^{k_1 \times k_2}$ the 2-dimensional tensor satisfying that $(\lambda_1 \otimes \lambda_2)(i,j) = \lambda_1(i) \cdot \lambda_2(j)$ for any $(i,j) \in [k_1] \times [k_2]$.

Fact 3.28. *Let $C_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ and $C_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ be linear codes. Then for any $(i_1, i_2) \in [k_1] \times [k_2]$, there exist a pair of vectors $\lambda_1 \in \mathbb{F}^{k_1}$ and $\lambda_2 \in \mathbb{F}^{k_2}$, so that for any $\mathsf{m} \in \mathbb{F}^{k_1 \times k_2}$, $\langle \lambda_1 \otimes \lambda_2, \mathsf{m} \rangle = c(i_1, i_2)$, where $c = (C_1 \otimes C_2)(\mathsf{m})$.*

Proof: Fix $(i_1, i_2) \in [k_1] \times [k_2]$. By linearity of C_1, there exists $\lambda_1 \in \mathbb{F}^{k_1}$, so that for any $\mathsf{m}_1 \in \mathbb{F}^{k_1}$, $c_1(i_1) = \langle \lambda_1, \mathsf{m}_1 \rangle$, where $c_1 = C_1(\mathsf{m}_1)$. Similarly, by linearity of C_2, there exists $\lambda_2 \in \mathbb{F}^{k_2}$, so that for any $\mathsf{m}_2 \in \mathbb{F}^{k_2}$, $c_2(i_2) = \langle \lambda_2, \mathsf{m}_2 \rangle$, where $c_2 = C_2(\mathsf{m}_2)$.

Next fix $\mathsf{m} \in \mathbb{F}^{k_1 \times k_2}$. Let $c' \in \mathbb{F}^{n_1 \times k_2}$ be the matrix obtained by encoding each column of m with the code C_1, and note that $c = (C_1 \otimes C_2)(\mathsf{m})$ is obtained by encoding each row of c' with the code C_2. Thus, we have that

$$\begin{aligned} c(i_1, i_2) &= \langle \lambda_2, c'(i_1, *) \rangle = \sum_{j=1}^{k_2} \lambda_2(j) \cdot c'(i_1, j) \\ &= \sum_{j=1}^{k_2} \lambda_2(j) \cdot \langle \lambda_1, \mathsf{m}(*, j) \rangle = \sum_{j=1}^{k_2} \lambda_2(j) \sum_{i=1}^{k_1} \lambda_1(i) \cdot \mathsf{m}(i, j) \\ &= \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \lambda_1(i) \cdot \lambda_2(j) \cdot \mathsf{m}(i, j) = \langle \lambda_1 \otimes \lambda_2, \mathsf{m} \rangle. \end{aligned}$$

■

For d vectors $\lambda_1, \ldots, \lambda_d \in \mathbb{F}^k$, we denote by $\lambda_1 \otimes \cdots \otimes \lambda_d \in \mathbb{F}^{[k]^d}$ the d-dimensional tensor satisfying that $(\lambda_1 \otimes \cdots \otimes \lambda_d)(\mathbf{i}) = \lambda_1(\mathbf{i}(1)) \cdots \lambda_d(\mathbf{i}(d)) = \prod_{j=1}^{d} \lambda_j(\mathbf{i}(j))$ for any $\mathbf{i} \in [k]^d$. Once more, by induction, the above fact yields the following corollary.

Corollary 3.29. *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code. Then for any $\mathbf{i} \in [n]^d$, there exist $\lambda_1, \ldots, \lambda_d \in \mathbb{F}^k$, so that for any $\mathsf{m} \in \mathbb{F}^{[k]^d}$, $\langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle = c(\mathbf{i})$, where $c = C^{\otimes d}(\mathsf{m})$.*

A main property of the tensor product operation that will be useful for us is that it roughly preserves both the zero knowledge and the local testing properties of the base code, as discussed next.

Zero-knowledge properties of tensor codes. For the ZK-preserving property, recall that ZK codes are associated with a specific randomized encoding function, and ZK holds with respect to that encoding function. Therefore, when defining the tensor product of ZK codes we must also specify how the randomized encoding function operates. We follow the definition of [BCL22].

Definition 3.30 (Randomized Encoding for Tensor Codes)**.** *Let $C_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ and $C_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ be linear codes, and let $k_1' \leq k_1$ and $k_2' \leq k_2$ be parameters. The $(k_1' \times k_2')$-*randomized encoding function for *$C_1 \otimes C_2$ is a function $\mathsf{Enc} : \mathbb{F}^{k_1' \times k_2'} \to \mathbb{F}^{n_1 \times n_2}$, defined as follows. On input $\mathsf{m}' \in \mathbb{F}^{k_1' \times k_2'}$ and $1^{k_1 \times k_2}$, Enc first extends m' to $\mathsf{m} \in \mathbb{F}^{k_1 \times k_2}$, by padding m' with random field elements, and then outputs $(C_1 \otimes C_2)(\mathsf{m})$.*

*The definition extends to higher dimensions as follows. Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code, and let $k' < k$ be a parameter. The $[k']^d$-*randomized encoding function for *$C^{\otimes d}$ is a function $\mathsf{Enc} : \mathbb{F}^{[k']^d} \to \mathbb{F}^{[n]^d}$, defined as follows. On input $\mathsf{m}' \in \mathbb{F}^{[k']^d}$ and 1^{k^d}, Enc first extends m' to $\mathsf{m} \in \mathbb{F}^{[k]^d}$, by padding m' with random field elements, and then outputs $C^{\otimes d}(\mathsf{m})$.*

The ZK-preservation property follows from the following theorem of [BCL22] (see also the full version [BCL20, Thm. 6.2]).

Theorem 3.31 (Tensor Product Preserves ZK, [BCL20], Theorem 6.2)**.** *Suppose that $\mathcal{C}_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ is a linear code ensemble that has t_1-ZK (t_1-uniform ZK, respectively) with respect to the k_1'-randomized encoding function, and that $\mathcal{C}_2 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ is a linear code ensemble that has t_2-ZK (t_2-uniform ZK, respectively) with respect to the k_2'-randomized encoding function. Then $\mathcal{C}_1 \otimes \mathcal{C}_2$ has $\min\{t_1, t_2\}$-ZK ($\min\{t_1, t_2\}$-uniform ZK) with respect to the $(k_1' \times k_2')$-randomized encoding function.*

The above theorem in particular implies the following corollary.

Corollary 3.32 ([BCL20], Corollary 6.3)**.** *Let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code ensemble that has t-ZK (t-uniform ZK, respectively) with respect to the k'-randomized encoding function. Then for any $d > 1$, $C^{\otimes d}$ has t-ZK (uniform t-ZK, respectively) with respect to the $[k']^d$-randomized encoding function.*

Local testing of tensor codes. The LTC-preserving property follows from the following local testing property of tensor codes from [Vid15].

Theorem 3.33 (Local Testing of Tensor Codes, [Vid15], Theorems 3.1 and 4.4)**.** *Suppose that $C : \mathbb{F}^k \to \mathbb{F}^n$ is a linear code of relative distance δ, and let $d > 1$. Let* TEST *be a local tester for $C^{\otimes d}$, which given a string $w \in \mathbb{F}^{[n]^d}$, picks a random axis-parallel line ℓ (axis-parallel two-dimensional plane P, resp.) in $[n]^d$, and accepts if and only if $w|_\ell$ is a codeword of C ($w|_P$ is a codeword of $C \otimes C$, resp.). Then* TEST *is an $(n, \delta^{O(d)})$-local tester ($(n^2, \delta^{O(d)})$-robust local tester, resp.) for $C^{\otimes d}$.*

The above lemma implies the following corollary.

Corollary 3.34 (Tensor Product Preserves Local Testing)**.** *Suppose that $C : \mathbb{F}^k \to \mathbb{F}^n$ is a linear code of relative distance δ, and that $C \otimes C$ is a (q, μ)-LTC ((q, μ)-robust LTC, resp.). Then for any $d > 1$, $C^{\otimes d}$ is an $(q, \mu \cdot \delta^{O(d)})$-LTC ($(q, \mu \cdot \delta^{O(d)})$-robust LTC, resp.). Moreover, if the (robust) local tester for $C \otimes C$ runs in time T, then the (robust) local tester for $C^{\otimes d}$ runs in time $O(T)$.*

3.2.4 Low-degree Extensions (LDEs) and Reed-Solomon (RS) Codes

A special case of tensor codes that will be of particular interest to us is the *low-degree extension*, defined as follows.

Lemma 3.35 (Low-Degree Extension (LDE), [AS98], Theorem 4.1.5)**.** *Let $\mathbb{F}$ be a finite field of size n, and identify the elements of $\mathbb{F}$ with $[n]$. For non-negative integers $k \leq n$ and d, and a function $w : [k]^d \to \mathbb{F}$, there exists a unique d-variate polynomial $\hat{w}$ over $\mathbb{F}$ of individual degree $k - 1$ that agrees with w on $[k]^d$. Moreover, $\hat{w}$ can be computed in time $n^{O(d)}$.*

The polynomial $\hat{w}$ is called the Low-Degree Extension (LDE) of w. It can be verified that the evaluation table of $\hat{w}$ over $\mathbb{F}$ is a codeword in the d-dimensional tensor product of the Reed-Solomon code $\mathsf{RS}_{k,n}$, defined as follows.

Definition 3.36 (Reed-Solomon (RS) Codes, [RS60])**.** *Let $\mathbb{F}$ be a finite field of size n, and identify the elements of $\mathbb{F}$ with $[n]$. Let $k \leq n$ be a non-negative integer. The (systematic)* Reed-Solomon (RS) *code $\mathsf{RS}_{k,n}$ is a map $\mathsf{RS}_{k,n} : \mathbb{F}^k \to \mathbb{F}^n$. The encoding of a message $\mathsf{m} \in \mathbb{F}^k$ is $(f_{\mathsf{m}}(i))_{i \in \mathbb{F}}$, where $f_{\mathsf{m}}(x)$ is the unique univariate polynomial over $\mathbb{F}$ of degree at most $k - 1$ which satisfies that $f_{\mathsf{m}}(i) = \mathsf{m}(i)$ for any $i \in [k]$. The* punctured Reed-Solomon *code $\mathsf{PRS}_{k,n}$ is a map $\mathsf{PRS}_{k,n} : \mathbb{F}^k \to \mathbb{F}^{n-k}$, where the encoding of a message $\mathsf{m} \in \mathbb{F}^k$ is $(f_{\mathsf{m}}(i))_{i \in \mathbb{F} \setminus [k]}$.*

Lemma 3.37. *For any non-negative integers $k \leq n$, $\mathsf{RS}_{k,n}$ is an explicit systematic linear code of rate $\frac{k}{n}$ and distance $n-k+1$, and $\mathsf{PRS}_{k,n}$ is an explicit linear code of rate $\frac{k}{n-k}$ and distance $n-2k+1$.*

Claim 3.38. *Let $\mathbb{F}$ be a finite field of size n, and identify the elements of $\mathbb{F}$ with $[n]$. Let $k \leq n$ be a non-negative integer, let $w : [k]^d \to \mathbb{F}$ be a function, and let $\hat{w}$ be the low-degree extension of w. Then the evaluation table $(\hat{w}(\mathbf{i}))_{\mathbf{i}\in\mathbb{F}^d}$ of $\hat{w}$ over $\mathbb{F}$ is a codeword in $(\mathsf{RS}_{k,n})^{\otimes d}$. Similarly, the evaluation table $(\hat{w}(\mathbf{i}))_{\mathbf{i}\in(\mathbb{F}\setminus[k])^d}$ of $\hat{w}$ over $\mathbb{F}$ is a codeword in $(\mathsf{PRS}_{k,n})^{\otimes d}$.*

We shall also need the following lemma which gives a family of "zero-tester polynomials" for the low-degree extension.

Lemma 3.39 ("Zero-Tester" Polynomials, [AS98], Theorem 4.2.1.2)**.** *Let $\mathbb{F}$ be a finite field of size n, and identify the elements of $\mathbb{F}$ with $[n]$. Let k, d be non-negative integers so that $8dk < n$. Then there exists a family $\mathcal{Q}$ of d-variate polynomials over $\mathbb{F}$ of individual degree at most $k-1$, where $|\mathcal{Q}| \leq n^d$, so that if $P : [k]^d \to \mathbb{F}$ is not identically zero, then*

$$\Pr_{Q\leftarrow\mathcal{Q}}\left[\sum_{\mathbf{i}\in[k]^d} Q(\mathbf{i})\cdot P(\mathbf{i}) = 0\right] \leq \frac{1}{2}.$$

Moreover, the family $\mathcal{Q}$ is constructible in time $n^{O(d)}$.

4 Zero-Knowledge Properties of Tensor Codes

In this section, we show several zero-knowledge properties of tensor codes. These properties imply in turn several different ZK local testing procedures for tensor codes that will be useful for obtaining our zero-knowledge IOPs.

In Section 4.1, we generalize and strengthen a result of [BCL22] on zero-knowledge preservation under tensor products. More specifically, [BCL22] show that if $\mathcal{C}_1$ and $\mathcal{C}_2$ have t_1- and t_2-ZK, respectively (with respect to the k_1', k_2'-randomized encoding function, respectively), then their tensor product $\mathcal{C}_1 \otimes \mathcal{C}_2$ has $\min\{t_1, t_2\}$-ZK (with respect to the $(k_1' \times k_2')$-randomized encoding function). We strengthen their result and show that $\mathcal{C}_1 \otimes \mathcal{C}_2$ in fact satisfies a stronger ZK property, namely, it has ZK against an adversary that reads t_1 *entire rows* or t_2 *entire columns*. As a corollary, this implies that the tensor product $\mathcal{C}^{\otimes d}$ has a local testing procedure that has ZK against the *honest tester*, as well as a 2-round ZK IOPP (with ZK against *malicious* verifiers) for testing membership in tensor product, assuming that the base code $\mathcal{C}$ has a stronger uniform ZK guarantee.

In Section 4.2, we investigate whether $\mathcal{C}_1 \otimes \mathcal{C}_2$ can generally have $\Omega(t_1 \cdot t_2)$-ZK. We show an example of a code $\mathcal{C}$ that has t-ZK, but its tensor product $\mathcal{C} \otimes \mathcal{C}$ does not have $\Omega(t^2)$-ZK, which answers an open question posed by [BCL22]. On the other hand, note that the results of [ISVW13] imply that for *any* linear code $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$, there *exists* a randomized encoding function (not necessarily the $(k' \times k')$-randomized encoding function) with respect to which $\mathcal{C} \otimes \mathcal{C}$ has $\Omega(n^2)$-ZK. As a corollary, this implies in turn that for any $d > 1$ and any linear code $\mathcal{C}$, there exists a randomized encoding function with respect to which the tensor product $\mathcal{C}^{\otimes d}$ is a ZK-LTC (i.e., it is an LTC in which the zero-knowledge parameter is significantly larger than the query complexity).

Finally, in Section 4.3 we present a transformation that extends a tensor code into a tensor code over a larger field, while preserving its zero-knowledge properties.

4.1 ZK Against Line Queries

In this section, we show that the tensor product has ZK against an adversary that reads *entire rows or columns*. We then use these properties to deduce that the tensor product has two different ZK local testing procedures.

More specifically, in Section 4.1.1 below we show that if $\mathcal{C}_1$ and $\mathcal{C}_2$ have t_1- and t_2-ZK, respectively, then their tensor product $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against an adversary that reads t_1 *entire rows* or t_2 *entire columns*. As a corollary, this implies that if $\mathcal{C}$ has 1-ZK, then $\mathcal{C}^{\otimes d}$ has a local testing procedure that has ZK against the *honest tester*.

Then in Section 4.1.2, we show that if $\mathcal{C}_1$ and $\mathcal{C}_2$ have *uniform* t_1- and t_2-ZK, then their tensor product $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against an adversary that can *adaptively choose* whether it will read t_1 rows or t_2 columns, after making point queries to $\mathcal{C}_1 \otimes \mathcal{C}_2$. As a corollary, this implies in turn that if $\mathcal{C}$ has t-uniform ZK, then there exists a 2-round *constant query* ZK IOPP of sublinear length for testing membership in $\mathcal{C}^{\otimes d}$ (against a *malicious* verifier).

4.1.1 ZK Against Line Queries

In this section, we show that if $\mathcal{C}_1$ and $\mathcal{C}_2$ are ZK codes then their tensor product $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against adversaries that read *entire rows or columns*, and use this to conclude that if $\mathcal{C}$ is even 1-ZK, then $\mathcal{C}^{\otimes d}$ has a local testing procedure with ZK against the *honest tester*. We start by formally defining the stronger notion of zero-knowledge for rows/columns.

Definition 4.1 (Row/Column Restricted Adversary)**.** *Let $C_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ and $C_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ be linear codes, and let c be a codeword of $C_1 \otimes C_2 : \mathbb{F}^{k_1 \times k_2} \to \mathbb{F}^{n_1 \times n_2}$. A* row oracle for c *is an oracle that on input $i \in [n_1]$ outputs the i'th row $c(i, *)$ of c. We say that an algorithm $\mathcal{A}$ with access to a row oracle for c is t-*row restricted *if $\mathcal{A}$ makes at most t oracle queries. Similarly, a* column oracle for c *on input $j \in [n_2]$ outputs the j'th column $c(*, j)$ of c, and an algorithm $\mathcal{A}$ with access to a column oracle for c is t-*column restricted *if $\mathcal{A}$ makes at most t oracle queries.*

Theorem 4.2 (ZK Against Row/Column-Restricted Adversaries)**.** *Let $\mathcal{C}_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ and $\mathcal{C}_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ be linear code ensembles, such that $\mathcal{C}_1$ ($\mathcal{C}_2$, resp.) has t-ZK with respect to the k_1'-randomized encoding function (k_2'-randomized encoding function, resp.). Then $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against t-row restricted adversaries (t-column restricted adversaries, resp.), with respect to the $(k_1' \times k_2')$-randomized encoding function.*

The above theorem is an immediate consequence of the following lemma which says that if $\mathcal{C}_1$ has t-ZK, then for any code $\mathcal{C}_2$, the tensor product $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against t-row restricted adversaries. By the symmetry of encoding of the encoding function of tensor codes, this also implies that if $\mathcal{C}_2$ has t_2-ZK then $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against t-column restricted adversaries.

Lemma 4.3. *Let $\mathcal{C}_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ and $\mathcal{C}_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ be linear code ensembles, so that $\mathcal{C}_1$ has t-ZK with respect to the k_1'-randomized encoding function. Then $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against t-row restricted adversaries with respect to the $(k_1' \times k_2)$-randomized encoding function.*

Remark 4.4. *Lemma 4.3 guarantees that if $\mathcal{C}_1$ has ZK, then $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against* row restricted *adversaries. As will be evident in the proof below, the simulator uses* only *the fact that the codeword was generated by* first *encoding using $\mathcal{C}_1$, and then encoding using $\mathcal{C}_2$, and does not otherwise rely on the identity of the code used to encode the rows or columns. Put differently, had we computed the tensor by first encoding the rows using $\mathcal{C}_2$, then encoding the columns using $\mathcal{C}_1$, the ZK of $\mathcal{C}_2$ would imply ZK against* column restricted *adversaries. Crucially, both of these encoding procedures (first encoding the columns*

using $\mathcal{C}_1$, or first encoding the rows using $\mathcal{C}_2$) result in the same codeword, so we conclude that ZK of $\mathcal{C}_2$ implies that $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against column-restricted adversaries.

This important property of the proof – namely, that the simulator relies only on which code was applied first – will be used in Section 4.1.2 below.

Before we prove the above lemma, we show how it implies the above Theorem 4.2.

Proof of Theorem 4.2: By symmetry of the encoding function for tensor codes (since the order of encodings between rows and columns can be swapped), it suffices to prove the theorem for t_1-row restricted adversaries (cf. Remark 4.4).

Let Sim be the simulator given by Lemma 4.3 for t_1-row restricted adversaries with respect to the $(k_1' \times k_2)$-randomized encoding function. We shall show a black-box straight-line simulator Sim′ for t_1-row restricted adversaries with respect to the $(k_1' \times k_2')$-randomized encoding function. On input $1^{k_1 \times k_2}, k_1' \times k_2'$, and given oracle access to a t_1-row restricted adversary $\mathcal{A}$, the simulator Sim′ runs the simulator Sim on input $1^{k_1 \times k_2}, k_1' \times k_2$, relaying messages between $\mathcal{A}$ and Sim.

Let $\mathsf{m} \in \mathbb{F}^{k_1' \times k_2'}$, let $r \in \mathbb{F}^{k_1' \times (k_2 - k_2')}$, and let $(\mathsf{m}, r) \in \mathbb{F}^{k_1' \times k_2}$ denote the matrix which equals m in the first k_2' columns and r in the last $k_2 - k_2'$ columns. For every such r, Lemma 4.3 guarantees that the real view of $\mathcal{A}$, when given oracle access to a random encoding of (m, r) generated with the $(k_1' \times k_2)$-randomized encoding function, is identically distributed to $\mathcal{A}$'s simulated view when its oracle queries are answered according to Sim. In particular, because $\mathcal{A}$'s view in the simulation with Sim′ is exactly its view in the simulation with Sim, this implies that – conditioned on r being the randomness padded to the rows of m – the real-world view of $\mathcal{A}$, when given oracle access to a random encoding of m generated with the $(k_1' \times k_2')$-randomized encoding function, is identically distributed to its view when $\mathcal{A}$'s oracle queries are answered by Sim′. Using a standard conditional probability argument, this implies that the real and simulated views of $\mathcal{A}$ are identically distributed (even without conditioning on r). ■

We now turn to the proof of Lemma 4.3.

Proof of Lemma 4.3: Let Sim be the simulator for $\mathcal{C}_1$ whose existence follows from the t-ZK of $\mathcal{C}_1$. We will describe a black-box, straight-line simulator Sim′ that uses Sim and the adversary as black boxes.

The simulator Sim′ on input $1^{k_1 \times k_2}, k_1' \times k_2$, and given oracle access to a t-row restricted adversary $\mathcal{A}$, operates as follows.

1. Initializes k_2 random and independent instantiations of the simulator Sim for the column code $\mathcal{C}_1$ with input $1^{k_1}, k_1'$.
2. Interacts with $\mathcal{A}$, answering each row query i of $\mathcal{A}$ as follows. Sim′ queries each of the executions of Sim on i, and obtains simulated codeword symbols $c_i^1, \ldots, c_i^{k_2}$. It then honestly encodes $(c_i^1, \ldots, c_i^{k_2})$ using the code $\mathcal{C}_2$, to obtain a codeword $c(i)$, which it provides to $\mathcal{A}$ as the answer of the row oracle.

We claim that for every $\mathsf{m} \in \mathbb{F}^{k_1' \times k_2}$, the view $\mathsf{View}_{\mathcal{A}}^{R}(\mathsf{m})$ of $\mathcal{A}$ when given oracle access to an actual encoding of m is distributed identically to its view $\mathsf{View}_{\mathcal{A}}^{S}(\mathsf{m})$ when $\mathcal{A}$'s oracle answers are generated by Sim′. We prove this using a hybrid argument.

For $0 \leq \ell \leq k_2$, we define a hybrid distribution $\mathcal{H}_\ell$ which is obtained by answering $\mathcal{A}$'s oracle queries by emulating Sim′, but generating $c_i^1, \ldots, c_i^\ell$ in Step 2 by *honestly* encoding the first ℓ columns of m (this is done by first sampling the randomness required for the randomized encoding of $\mathcal{C}_1 \otimes \mathcal{C}_2$, and then encoding the first ℓ columns via $\mathcal{C}_1$), and simulating the rest of the symbols

$c_i^{\ell+1}, \ldots, c_i^{k_2}$ using Sim. Notice that $\mathcal{H}_0 = \mathsf{View}^S_{\mathcal{A}}(\mathsf{m})$, and $\mathcal{H}_{k_2} = \mathsf{View}^R_{\mathcal{A}}(\mathsf{m})$. Therefore, it suffices to prove that $\mathcal{H}_\ell \equiv \mathcal{H}_{\ell-1}$ for every $\ell \in [k_2]$. We show that this follows from the t-ZK of $\mathcal{C}_1$, otherwise $\mathcal{A}$ can be used to design an adversary $\mathcal{A}'$ that distinguishes between t real and simulated symbols in encodings of a *single* column of m (i.e., in a single $\mathcal{C}_1$-encoding).

Assume towards negation that there exists an $\ell^* \in [k_2]$ such that $\mathcal{H}_{\ell^*} \not\equiv \mathcal{H}_{\ell^*-1}$. We describe an (interactive) t-restricted adversary $\mathcal{A}'$ whose views – given oracle access to either real or simulated symbols in an encoding generated by $\mathcal{C}_1$ – are not identically distributed, which contradicts the t-ZK of $\mathcal{C}_1$. More specifically, we will show that this holds when $\mathcal{A}'$'s oracle is to a (real or simulated) encoding of the ℓ^*th column of m.

$\mathcal{A}'$ has m and ℓ^* hard-wired into it. It samples randomness for the randomized encoding of $\mathcal{C}_1 \otimes \mathcal{C}_2$, and honestly encodes columns $1, \ldots, \ell^* - 1$ of m using $\mathcal{C}_1$ with the sampled randomness. It additionally instantiates $k_2 - \ell^*$ random and independent executions of Sim (for columns $\ell^* + 1, \ldots, k_2$). Then, $\mathcal{A}'$ emulates $\mathcal{A}$, answering each query i of $\mathcal{A}$ as follows. (1) $\mathcal{A}'$ queries its own oracle on i (this emulates the ith symbol in the encoding of the ℓ^*'th column of m). (2) $\mathcal{A}'$ uses the encoded columns $1, \ldots, \ell^* - 1$ generated above to determine the ith symbols in the encodings of columns $1, \ldots, \ell^* - 1$. (3) $\mathcal{A}'$ uses the $k_2 - \ell^*$ executions of Sim to emulate the ith symbol in each of the columns $\ell^* + 1, \ldots, k_2$. (4) $\mathcal{A}'$ honestly encodes the string of ith symbols generated in Steps (1)-(3) with $\mathcal{C}_2$, and provides the resultant row as the answer to $\mathcal{A}$'s query.

Notice that if $\mathcal{A}'$'s oracle is to an actual encoding, then the view of $\mathcal{A}$ is distributed according to $\mathcal{H}_{\ell^*}$, otherwise it is distributed according to $\mathcal{H}_{\ell^*-1}$. Therefore, since $\mathcal{H}_{\ell^*-1} \not\equiv \mathcal{H}_{\ell^*}$ then the views of $\mathcal{A}'$ (with oracle access to real or simulated symbols in an encoding of the ℓ^*'th column of m) are not identically distributed. This contradicts the t-ZK of $\mathcal{C}_1$, because $\mathcal{A}'$ is t-restricted ($\mathcal{A}'$ makes a single query to its oracle for every row query of $\mathcal{A}$, and $\mathcal{A}$ is t-row restricted). ■

Honest-Tester ZK Local Testing for Tensor Codes. The results of this section can be used to show that if $\mathcal{C}$ has 1-ZK, then $\mathcal{C}^{\otimes d}$ has a local testing procedure that has ZK against the *honest tester*. To show this, we need an extension of Theorem 4.2 which says that $\mathcal{C}^{\otimes d}$ has ZK against an adversary that queries axis-parallel *subspaces* in the same direction. To state this formally, we first extend the notion of row/column restricted adversaries (Definition 4.1) to adversaries that query axis-parallel subspaces in the same direction.

In what follows, for a non-empty subset $J \subsetneq [d]$ of coordinates, and values $\mathbf{j} \in [n]^J$ to these coordinates, let $V_{J \leftarrow \mathbf{j}}$ denote the (axis-parallel) subspace of $[n]^d$ that is obtained by fixing the coordinates in J to $\mathbf{j}$, that is, $V_{J \leftarrow \mathbf{j}} = \{\mathbf{i} \in [n]^d \mid \mathbf{i}|_J = \mathbf{j}\}$. Note that for the special case of $d = 2$, a row $(i, *)$ (column $(*, j)$, respectively) corresponds to the special case that $J = \{1\}$ ($J = \{2\}$, respectively) and $\mathbf{j} = i$ ($\mathbf{j} = j$, respectively).

Definition 4.5 (Subspace-Restricted Adversary)**.** *Let $d > 1$ be an integer, let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code, and let c be a codeword of $C^{\otimes d} : \mathbb{F}^{[k]^d} \to \mathbb{F}^{[n]^d}$. Let $J \subsetneq [d]$ be a non-empty subset of coordinates. A* subspace oracle in direction J for c *is an oracle that on input $\mathbf{j} \in [n]^{[d]\setminus J}$ outputs the restriction of c to the subspace $V_{([d]\setminus J) \leftarrow \mathbf{j}}$. We say that an algorithm $\mathcal{A}$ with access to a subspace oracle in direction J for c is* (t, J)-restricted *if $\mathcal{A}$ makes at most t oracle queries.*

The following corollary is a consequence of Lemma 4.3 above.

Corollary 4.6. *Let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code ensemble that has t-ZK with respect to the k'-randomized encoding function for some $k' < k$.Then for any $d > 1$, and for any non-empty subset $J \subsetneq [d]$, $\mathcal{C}^{\otimes d}$ has ZK against (t, J)-restricted adversaries with respect to the $[k']^d$-randomized encoding function.*

Proof: Let $\zeta := |J|$. By symmetry of the encoding function for tensor codes (since the order of encodings in different dimensions can be swapped, cf. Remark 4.4), we may assume without loss of generality that $J = \{d-\zeta+1, d-\zeta+2, \ldots, d\}$. By properties of tensor codes, we can further view any codeword $c \in C^{\otimes d}$, generated using the $[k']^d$-randomized encoding function, as a codeword of the two-dimensional tensor $C^{\otimes(d-\zeta)} \otimes C^{\otimes\zeta}$, generated using the $([k']^{d-\zeta} \times [k']^{\zeta})$-randomized encoding function. Moreover, by Corollary 3.32, we have that $C^{\otimes(d-\zeta)}$ has t-ZK with respect to the $[k']^{d-\zeta}$-randomized encoding function. By Theorem 4.2, this implies in turn that $C^{\otimes(d-\zeta)} \otimes C^{\otimes\zeta}$ has ZK against t-row restricted adversaries with respect to the $([k']^{d-\zeta} \times [k']^{\zeta})$-randomized encoding function. But this is equivalent to saying that $C^{\otimes d}$ has ZK against (t, J)-restricted adversaries with respect to the $[k']^d$-randomized encoding function. ■

Recall that by Theorem 3.33, there exists a (robust) local tester for $C^{\otimes d}$ which queries only a single random axis-parallel line (two-dimensional plane, respectively). Consequently, the above corollary implies that if the base code is 1-ZK, then this test has ZK against the honest tester.

Corollary 4.7 (Local Testing with Honest-Tester ZK for Tensor Codes.)**.** *Let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be an explicit linear code ensemble of relative distance δ, that has 1-ZK with respect to the k'-randomized encoding function. Then for any $d > 1$, $\mathcal{C}^{\otimes d} : \mathbb{F}^{[k]^d} \to \mathbb{F}^{[n]^d}$ is an $(n, \delta^{O(d)})$-LTC and $(n^2, \delta^{O(d)})$-robust LTC that has ZK with respect to the $[k']^d$-randomized encoding function against the honest tester.*

4.1.2 ZK Against Adaptive Line Queries

In the previous section, we showed that if $\mathcal{C}_1$ has t_1-ZK and $\mathcal{C}_2$ has t_2-ZK, then the tensor product $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against adversaries that query *either* t_1 rows *or* t_2 columns. In this section, we prove a stronger guarantee against an adversary that can *adaptively choose* whether its oracle will be a row or a column oracle, *after* making point queries to $\mathcal{C}_1 \otimes \mathcal{C}_2$. However, the results of this section will only apply to codes $\mathcal{C}_1$ and $\mathcal{C}_2$ that satisfy the stronger uniform-ZK property. We then use this to show that if $\mathcal{C}$ has t-uniform ZK, then there exists a 2-round *constant query* ZK IOPP of sublinear length for testing membership in $\mathcal{C}^{\otimes d}$. We start by formally defining the stronger notion of *adaptive* line-restricted adversaries.

Definition 4.8 (Adaptive Line-Restricted Adversary)**.** *Let $\mathcal{C}_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ and $\mathcal{C}_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ be linear codes, and let c be a codeword of $C_1 \otimes C_2 : \mathbb{F}^{k_1 \times k_2} \to \mathbb{F}^{n_1 \times n_2}$. We say that a 2-phase algorithm $\mathcal{A}$ with access to a point, row, and column oracle for c is* adaptive t-line restricted *if the following holds. In the first phase, $\mathcal{A}$ makes $0 \leq t' \leq t$* point queries *to c. Then, it chooses $\ell \in \{1, 2\}$, and the first phase ends. Next, if $\ell = 1$ ($\ell = 2$, respectively), then $\mathcal{A}$ is given the entire row $c(i, *) \in \mathcal{C}_2$ (column $c(*, j) \in \mathcal{C}_1$, respectively) for each query (i, j) which $\mathcal{A}$ made in the first phase. In the second phase, if $\ell = 1$ ($\ell = 2$, respectively), then $\mathcal{A}$ makes at most $t - t'$ queries to its row (column, respectively) oracle.*

The following theorem states that the tensor product of two uniform ZK codes guarantees ZK against adaptive line-restricted adversaries.

Theorem 4.9 (Tensor Product of Uniform ZK Codes has ZK against Adaptive Line-Restricted Adversaries)**.** *Let $\mathcal{C}_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ and $\mathcal{C}_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ be linear codes that have t_1- and t_2-uniform ZK with respect to the k_1'- and k_2'-randomized encoding function, respectively. Then $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against adaptive t-line restricted adversaries with respect to the $(k_1' \times k_2')$-randomized encoding function, where $t = \min\{t_1, t_2\}$.*

The above theorem is an immediate consequence of Lemma 4.10 below, which shows that in the special case that $\mathcal{C}_1$ and $\mathcal{C}_2$ have uniform ZK, one can obtain a simulator for row-restricted adversaries with special properties. In more detail, in what follows, we say that a 2-phase algorithm

$\mathcal{A}$ with access to a point and row oracle for a codeword c of $\mathcal{C}_1 \otimes \mathcal{C}_2$ is adaptive t-row restricted if $\mathcal{A}$ always chooses $\ell = 1$ in Definition 4.8.

Lemma 4.10. *Let $\mathcal{C}_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ and $\mathcal{C}_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ be linear codes that have t_1- and t_2-uniform ZK with respect to the k_1'- and k_2'-randomized encoding function, respectively, and let $t := \min\{t_1, t_2\}$. Then $\mathcal{C}_1 \otimes \mathcal{C}_2$ has ZK against adaptive t-row restricted adversaries with respect to the $(k_1' \times k_2')$-randomized encoding function, with the following two-phase simulator $\mathsf{Sim} = (\mathsf{Sim}', \mathsf{Sim}'')$. Sim' takes $1^{k_1 \times k_2}, k_1' \times k_2'$ as input, and answers $\mathcal{A}$'s queries in the first phase uniformly at random. Let Q denote the set of queries which $\mathcal{A}$ makes in the first phase, and $\{c'_q\}_{q \in Q}$ denote the answers given by Sim'. Sim'' takes as input $1^{k_1 \times k_2}, k_1' \times k_2', Q$ and $\{c'_q\}_{q \in Q}$, and completes the simulation with $\mathcal{A}$.*

Before we prove the above lemma, we show how it implies the above Theorem 4.9.

Proof of Theorem 4.9: We describe a simulator Sim for an adaptive t-line restricted adversary. The simulator Sim answers all point queries in the first phase uniformly and independently at random. Then, if $\ell = 1$, Sim completes the simulation using the simulator for t-adaptive row restricted adversaries, guaranteed by Lemma 4.10. If $\ell = 2$, then by symmetry of the encoding function for tensor codes (since the order of encodings between rows and columns can be swapped, cf. Remark 4.4), Sim can complete the simulation using the simulator for t-adaptive row restricted adversaries for the code obtained by first encoding the rows (using $\mathcal{C}_2$). ∎

We now turn to the proof of Lemma 4.10.

Proof of Lemma 4.9: Let $\mathsf{m} \in \mathbb{F}^{k_1 \times k_2}$ and $c \leftarrow \mathsf{Enc}(\mathsf{m})$, where Enc is the $(k_1' \times k_2')$-randomized encoding function for $C_1 \otimes C_2$. Let $\mathcal{A}$ be an adaptive t-row restricted adversary with respect to $\mathcal{C}_1 \otimes \mathcal{C}_2$. Let Sim' be a simulator which takes $1^{k_1 \times k_2}, k_1' \times k_2'$ as input, and answers $\mathcal{A}$'s queries in the first phase uniformly at random. Let Q denote the set of queries which $\mathcal{A}$ makes in the first phase, and $\{c'_q\}_{q \in Q}$ denote the answers given by Sim'. We describe a simulator Sim'', which takes as input $1^{k_1 \times k_2}, k_1' \times k_2', Q$ and $\{c'_q\}_{q \in Q}$, and completes the simulation with $\mathcal{A}$.

Claim: Q is identically distributed in both worlds, and $c|_Q$ is uniformly random. We will show that $c|_Q$ is uniformly random (i.e., distributed identically to the simulated answers given by Sim') in two steps. First, we will show that $c|_{Q'}$ is uniformly random for every $Q', |Q'| \leq t$. Then, we will show that the queries Q which $\mathcal{A}$ makes in the real world and the simulation are identically distributed. Intuitively, to prove the first step we will use the t_1-uniform ZK of $\mathcal{C}_1$ to claim that the joint distribution of columns $k_2' + 1, \ldots, k_2$ in the (at most t_1) queried rows is uniformly random. Therefore, in each of the queried rows the row encoding in $\mathcal{C}_2$ was generated using independent uniformly random field elements, and so the t_2-uniform ZK of $\mathcal{C}_2$ guarantees that the (at most t_2) entries queried in each of these rows is uniformly random. The formal proof follows.

Recall that in the first phase of the execution $\mathcal{A}$ makes at most $t' \leq t = \min\{t_1, t_2\}$ point queries. Therefore, it queries at most t_1 rows, and at most t_2 entries in each of these rows. Recall that m is encoded by first extending it to $\mathsf{m}' \in \mathbb{F}^{k_1 \times k_2}$ by padding m with random field elements, then encoding each column with $\mathcal{C}_1$ to obtain $\mathsf{m}'' \in \mathbb{F}^{n_1 \times k_2}$, and finally encoding each row of m'' with $\mathcal{C}_2$. Let $\mathcal{I}' := \{i \in [n_1] \,:\, \exists j \in [n_2], (i,j) \in Q'\}$ (intuitively, these are the rows which are queried in Q'). Since $|\mathcal{I}'| \leq t_1$, then by the t_1-uniform ZK of $\mathcal{C}_1$, for every $j \in [k_2]$ we have that $\mathsf{m}''(*, j)|_{\mathcal{I}'}$ – namely, the restriction of the j'th column of m'' to the rows in $\mathcal{I}'$ – is uniformly distributed. Moreover, since each column is encoded randomly and independently, the joint distribution $(\mathsf{m}''(*, j)|_{\mathcal{I}'})_{j \in [k_2]}$ is also uniformly random.[19] In particular, the joint distribution $(\mathsf{m}''(*, j)|_{\mathcal{I}'})_{j \in \{k_2'+1, \ldots, k_2\}}$ is uniformly

[19] We note that for this part of the proof we only need to analyze the distribution of the last $k_2 - k_2'$ columns of m'',

random. Therefore, each row $i \in \mathcal{I}'$ of m'' is encoded randomly *and independently* (with uniform randomness $(\mathsf{m}''(i,j))_{j\in\{k_2'+1,\ldots,k_2\}}$), so the t_2-uniform ZK of $\mathcal{C}_2$ guarantees that $c|_{Q'}$ is uniformly random. This completes the first step.

As for the second step, the queries which $\mathcal{A}$ makes are determined by $\mathcal{A}$'s randomness, and the answers to its previous queries. In particular, its first query is determined by $\mathcal{A}$'s randomness and is therefore identically distributed in the real world and the simulation. By induction over the number $t' \leq t$ of queries which $\mathcal{A}$ makes in the first phase, we have that the queries Q which $\mathcal{A}$ makes in the real world and the simulation are identically distributed, and $c'|_Q \equiv c|_Q$. Therefore, it suffices to prove the claim conditioned on Q and $c|_Q$.

Simulating a Row Oracle. It remains to show how to complete the simulation by defining Sim''. The simulator Sim'', on input $1^{k_1\times k_2}, k_1' \times k_2', Q$ and $\{c_q'\}_{q\in Q}$, operates as follows. Let $\mathcal{I} := \{i \in [n_1] \ : \ \exists j \in [n_2], (i,j) \in Q\}$ (recall that these are the rows which were queried in Q), and for every $i \in \mathcal{I}$, let $\mathcal{Q}_i := \{(i,j) \in Q\}$ (these are the entries which $\mathcal{A}$ queried from the i'th row in the first phase). For every $i \in \mathcal{I}$, Sim' samples a uniformly random codeword $c_i' \in \mathcal{C}_2$, conditioned on the event that $c_i'|_{\mathcal{Q}_i} = \{c_q'\}_{q\in\mathcal{Q}_i}$. The linearity of $\mathcal{C}_2$ guarantees that this sampling can be done efficiently by solving a system of linear equations, and the t_2-uniformity of $\mathcal{C}_2$ guarantees that the system has a solution. Sim'' then gives c_i' to $\mathcal{A}$ as the i'th row.

Next, Sim'' answers adaptive row queries i of $\mathcal{A}$ as follows.[20] Sim samples $\tilde{\mathsf{m}}_i \leftarrow \mathbb{F}^{k_2}$, deterministically encodes $c_i' := \mathcal{C}_2(\tilde{\mathsf{m}}_i)$ and provides c_i' as the oracle answer.

We now claim that the distribution induced by Sim'''s simulated answers is identical to the real world, conditioned on $c|_Q$ and Q. We prove this claim it two steps. First, we prove that the simulated rows $(c_i')_{i\in\mathcal{I}}$ which Sim'' provided at the onset of the second phase of the execution are distributed identically to the real world. Then, we prove that the simulated rows Sim'' provided in the second phase of the execution are distributed identically to the real world, conditioned on Q and $(c_i')_{i\in\mathcal{I}}$.

$(c_i')_{i\in\mathcal{I}} \equiv (c(i,*))_{i\in\mathcal{I}}$. Recall that we have already proven above that $(\mathsf{m}''(*,j)|_{\mathcal{I}'})_{j\in\{k_2'+1,\ldots,k_2\}}$ is uniformly random, meaning each row $i \in \mathcal{I}$ of m'' was encoded *independently* using *uniform randomness*. Therefore, it suffices to prove that $c(i,*) \equiv c_i'$ for every $i \in \mathcal{I}$ separately. Fix an $i \in \mathcal{I}$. As we have shown above, $\mathsf{m}''(i,*) \in \mathbb{F}^{k_2}$ (i.e., the message encoded in the i'th row) is uniformly random. Therefore, $c(i,*)$ is distributed as a random $\mathcal{C}_2$-encoding of a random message. Therefore, to show that $c_i' \equiv c(i,*)$, it suffices to show that for every pair of messages $\mathsf{m}_i, \mathsf{m}_i'$,

$$\left|\{\tilde{c} \ : \ \tilde{c} \in \mathsf{Supp}(\mathsf{Enc}_2(\mathsf{m}_i)) \wedge \tilde{c}|_{\mathcal{Q}_i} = c|_{\mathcal{Q}_i}\}\right| = \left|\left\{\tilde{c} \ : \ \tilde{c} \in \mathsf{Supp}(\mathsf{Enc}_2(\mathsf{m}_i')) \wedge \tilde{c}|_{\mathcal{Q}_i} = c|_{\mathcal{Q}_i}\right\}\right| \qquad (2)$$

where Enc_2 is the k_2'-randomized encoding function. Indeed, if Eq. (2) holds then sampling a random message and then sampling a random encoding of the message (conditioned on $c|_{\mathcal{Q}_i}$) is identical to directly sampling a codeword that is consistent with $c|_{\mathcal{Q}_i}$. Eq. (2) follows from the t_2-uniform ZK of $\mathcal{C}_2$ (which guarantees that the probability of observing $c|_{\mathcal{Q}_i}$ in a random encoding of $\mathsf{m}_i, \mathsf{m}_i'$ is equally likely), and the injectivity of $\mathcal{C}_2$'s generator matrix (which implies that each possible choice of randomness encoding for Enc_2 induces a unique codeword).

The real and simulated second-phase oracle answers are identically distributed. We show that for every $t'' \leq t - t'$, given the t'''th query i of $\mathcal{A}$ in the second phase, the simulated oracle answer of Sim'' is distributed identically to $c(i,*)$, conditioned on the oracle answers given earlier in the

but later in the proof we will use the fact that the entire rows $(\mathsf{m}''(*,j)|_{\mathcal{I}'})_{j\in[k_2]}$ are uniformly random.

[20] We assume without loss of generality that $i \notin \mathcal{I}$, otherwise the i'th row has already been determined and Sim'' gives it to $\mathcal{A}$ as the oracle answer.

simulation. By induction over t'' this will prove the claim since we have already proven that at the onset of the second phase, the simulation is perfect.

Fix $t'' \leq t - t'$. As noted above, we can assume without loss of generality that the i'th row has not been queried before (not even in the first phase of the execution). The t_1-uniform ZK of $\mathcal{C}_1$ therefore guarantees that $\mathsf{m}''(i,*)$ is uniformly random even when conditioned on the rows of m'' that were already queried by $\mathcal{A}$ in the execution. Therefore, $c(i,*)$ is a $\mathcal{C}_2$-encoding of a random message in $\mathbb{F}^{k_2}$, exactly as it is sampled in the simulation. ■

2-Round ZK IOPP for Testing Membership in Tensor Codes. The results of this section can be used to show that if $\mathcal{C}$ has t-uniform ZK, then there exists a 2-round distributional ZK IOPP for testing membership in $\mathcal{C}^{\otimes d}$ with sublinear length and constant query complexity. As before, to show this we first need to extend Definition 4.8 and Theorem 4.9 to higher dimensions.

Definition 4.11 (Adaptive Subspace-Restricted Adversary)**.** *Let $d > 1$ be an integer, let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code, and let c be a codeword of $\mathcal{C}^{\otimes d}$. We say that a 2-phase algorithm $\mathcal{A}$ is* adaptive t-subspace restricted *if the following holds. In the first phase, $\mathcal{A}$ is given oracle access to a point oracle for c, and makes $0 \leq t' \leq t$ point queries to c. Then, it chooses a non-empty subset $J \subsetneq [d]$, and the first phase ends. Next, $\mathcal{A}$ is given the restriction of c to the whole subspace $V_{([d]\setminus J)\leftarrow \mathbf{j}}$ for each query $\mathbf{i} \in [n]^d$ which $\mathcal{A}$ made in the first phase, where $\mathbf{j} = \mathbf{i}|_{[d]\setminus J}$. In the second phase, $\mathcal{A}$ is given oracle access to a subspace oracle in direction J for c, and makes at most $t - t'$ queries to the subspace oracle.*

The following corollary is a consequence of Lemma 4.10 above.

Corollary 4.12. *Let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code that has t-uniform ZK with respect to the k'-randomized encoding function. Then for every $d > 1$, $\mathcal{C}^{\otimes d}$ has ZK against adaptive t-subspace restricted adversaries with respect to the $[k']^d$-randomized encoding function.*

Proof: We describe a 2-phase PPT simulator $\mathsf{Sim} = (\mathsf{Sim}', \mathsf{Sim}'')$ that can perfectly simulate the answers to oracle queries of any adaptive t-subspace restricted adversary $\mathcal{A}$ to a codeword $c \in \mathcal{C}^{\otimes d}$. Recall that Sim', on input $1^{[k]^d}, [k']^d$, answers $\mathcal{A}$'s point queries to c in the first phase of the execution uniformly at random. Let $Q \in [n]^d$ denote the set of queries which $\mathcal{A}$ made in this phase, and let $J \subsetneq [d]$ denote the non-empty subset of coordinaes which $\mathcal{A}$ chose.

Let $\zeta := |J|$.By symmetry of the encoding function for tensor codes (since the order of encodings in different dimensions can be swapped, cf. Remark 4.4), we may assume without loss of generality that $J = \{d-\zeta+1, d-\zeta+2, \ldots, d\}$. By properties of tensor codes, we can further view any codeword $c \in C^{\otimes d}$, generated using the $[k']^d$-randomized encoding function, as a codeword of the two-dimensional tensor $C^{\otimes(d-\zeta)} \otimes C^{\otimes\zeta}$, generated using the $([k']^{d-\zeta} \times [k']^{\zeta})$-randomized encoding function. Moreover, by Corollary 3.32, we have that $C^{\otimes(d-\zeta)}, C^{\otimes(\zeta)}$ have t-uniform ZK with respect to the $[k']^{d-\zeta}$, $[k']^{\zeta}$-randomized encoding functions, respectively.

Therefore, Lemma 4.10 guarantees that there exists a PPT simulator Sim'' that, on input $1^{[k]^d}, [k']^d, Q$ and the answers given by Sim' in the first phase of the execution, can complete the simulation and provide $\mathcal{A}$ with rows in $C^{\otimes(d-\zeta)} \otimes C^{\otimes\zeta}$. In the second phase of the execution, Sim uses Sim'' to provide $\mathcal{A}$ with the corresponding subspaces for all point queries which $\mathcal{A}$ made in the first phase, and to answer $\mathcal{A}$'s subspace queries, noting that querying the restriction of c to a subspace in direction J is equivalent to querying a row of $C^{\otimes(d-\zeta)} \otimes C^{\otimes\zeta}$. Finally, Lemma 4.10 guarantees that the simulation induced by this process is perfect. ■

Next we show that the above corollary implies a 2-round ZK IOPP for checking membership in $\mathcal{C}^{\otimes d}$.

Corollary 4.13 (2-round Distributional ZK-IOPP for testing membership in tensor codes)**.** *The following holds for any constant $\delta, \alpha > 0$ and $d > 1$. Let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be an explicit linear code ensemble of relative distance δ, that has t-uniform ZK with respect to the k'-randomized encoding function. Suppose furthermore that membership in $\mathcal{C}^{\otimes 2}$ can be checked in time $T = T(n)$.*

Then there exists a 2-round α-IOPP over $\mathbb{F}$ with constant soundness error for the language $\mathcal{L} = \text{image}(\mathcal{C}^{\otimes d})$ (where the input is the implicit input and there is no explicit input), that has distributional $(t-1)$-ZK with respect to $\mathcal{C}$ and the $[k']^d$-randomized encoding function. The IOP has constant query complexity and communication complexity $\tilde{O}(T)$.

Proof: The IOPP proceeds as follows. The verifier first runs the query phase of the (n^2, μ)-robust local tester TEST for the code $\mathcal{C}^{\otimes d}$ for $\mu := \delta^{O(d)}$, given by Theorem 3.33, but instead of making the queries, the verifier sends the query set $I \subseteq [n]^d$ of size $|I| = n^2$ to the prover. The prover and the verifier then run the $\frac{\alpha \cdot \mu}{2}$-PCPP over $\mathbb{F}$, given by Theorem 3.8, for the language $\psi_{\text{TEST}}^{-1}(\mathsf{ACCEPT})$ on implicit input is $c|_I$ (and with no explicit input).

It can be verified that the round complexity, query complexity, and communication complexity are all as stated. Completeness is also straightforward. To see that the soundness property holds, suppose that c is α-far from $\mathcal{C}^{\otimes d}$. Then by the properties of the robust local tester, with probability at least $\frac{\alpha \cdot \mu}{2}$, it holds that $c|_I$ is $\frac{\alpha \cdot \mu}{2}$-far from $\psi_{\text{TEST}}^{-1}(\mathsf{ACCEPT})$, in which case the PCPP verifier will reject with probability at least $\frac{1}{2}$. So overall, in this case the verifier will reject with probability at least $\frac{\alpha \cdot \mu}{4} = \Omega(1)$.

Finally, to see that the ZK property holds, let Sim be the simulator against adaptive t-subspace restricted adversaries, guaranteed by Corollary 4.12 for the code $\mathcal{C}^{\otimes d}$ with respect to the $[k']^d$-randomized encoding function. We describe a simulator Sim′ for the IOPP, which operates as follows for every $(t-1)$-restricted verifier $\mathcal{V}^*$. Sim′ interacts with $\mathcal{V}^*$, forwarding her queries to Sim and providing the simulated codeword symbols as the answers of the oracle. At some point, $\mathcal{V}^*$ sends a query subset I, which is a two-dimensional axis-parallel subspace. Sim′ then sets I to be the subspace of the adaptive subspace-restricted adversary for Sim, and obtains from Sim a simulated $c|_I$. Sim′ honestly generated the PCPP proof π for $c|_I$, and uses π to answer any queries of $\mathcal{V}^*$ to the PCPP. Any further queries $i \in [n]^d$ of $\mathcal{V}^*$ to c are answered by querying Sim on the two-dimensional axis-parallel subspace in the direction of I that contains i, and forwarding the answer to $\mathcal{V}^*$. Notice that Sim′ emulates an adaptive t-subspace restricted adversary for Sim (Sim′ makes one additional query compared to $\mathcal{V}^*$, to simulate $c|_I$), and so indistinguishability of the real and simulated views follows directly from Corollary 4.12. ■

4.2 ZK Threshold of Tensor Product

In this section, we investigate whether $\mathcal{C}_1 \otimes \mathcal{C}_2$ can generally have $\Omega(t_1 \cdot t_2)$-ZK in the case that $\mathcal{C}_1$ and $\mathcal{C}_2$ have t_1, t_2-ZK, respectively.

In Section 4.2.1, we show an example of a code ensemble $\mathcal{C}$ that has t-ZK (with respect to the k'-randomized encoding function), but its tensor product $\mathcal{C} \otimes \mathcal{C}$ does not have $\Omega(t^2)$-ZK (with respect to the $(k' \times k')$-randomized encoding function); this answers an open question posed by [BCL22].

On the other hand, in Section 4.2.2 we note that the results of [ISVW13] imply that for *any* linear code $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$, there *exists* a randomized encoding function (not necessarily the $(k' \times k')$-randomized encoding function) with respect to which $\mathcal{C} \otimes \mathcal{C}$ has $\Omega(n^2)$-ZK. As a corollary, this implies in turn that for any linear code $\mathcal{C}$, there exists a randomized encoding function with respect to which the tensor product $\mathcal{C}^{\otimes d}$ is a ZK-LTC (i.e., it is an LTC in which the zero-knowledge parameter is significantly larger than the query complexity).

4.2.1 Limitations on the ZK Threshold

Recall that Theorem 3.31 (proven in [BCL22, Theorem 6.2]) says that if $\mathcal{C}_1$ and $\mathcal{C}_2$ have t_1- and t_2-ZK, respectively (with respect to the k'_1, k'_2-randomized encoding function, respectively), then their tensor product $\mathcal{C}_1 \otimes \mathcal{C}_2$ has $\min\{t_1, t_2\}$-ZK (with respect to the $(k'_1 \times k'_2)$-randomized encoding function). However, [BCL22, Page 21] remarks that one might hope to improve this theorem to obtain $\Omega(t_1 \cdot t_2)$-ZK.

In this section, we show a counter example to the above hope. Specifically, we show a code $\mathcal{C}$ that has t-ZK (even *t-uniform* ZK), but its tensor product $\mathcal{C} \otimes \mathcal{C}$ does not have $\Omega(t^2)$-ZK (even against *non-adaptive* adversaries). The code satisfying this property is the punctured Reed-Solomon (PRS) code (cf. Definition 3.36).

We first show that PRS has uniform ZK.

Lemma 4.14. *Let $\mathbb{F}$ be a finite field of size n, and identify the elements of $\mathbb{F}$ with $[n]$. Let $k \leq \frac{n}{2}$ be a non-negative integer, and let $\mathsf{PRS}_{k,n} : \mathbb{F}^{[k]} \to \mathbb{F}^{[n]\setminus[k]}$ be the Punctured Reed-Solomon code. Then for any non-negative integer $k' \leq k$, $\mathsf{PRS}_{k,n}$ has $(k-k')$-uniform ZK with respect to the k'-randomized encoding function.*

Proof: Fix $\mathsf{m} \in \mathbb{F}^{k'}$, and let $c = \mathsf{PRS}_{k,n}(\mathsf{m}, r)$ for a uniformly random string $r \in \mathbb{F}^{k-k'}$. Let $I \subseteq [n] \setminus [k]$ be an arbitrary subset of size $k - k'$. We would like to show that $c|_I$ is uniformly random. To show this, it suffices to show that for any string $v \in \mathbb{F}^{k-k'}$, there exists (a unique) $r \in \mathbb{F}^{k-k'}$ so that $\mathsf{PRS}_{k,n}(\mathsf{m}, r)|_I = v$.

To show the above, fix $v \in \mathbb{F}^{k-k'}$. Let f be the unique univariate polynomial over $\mathbb{F}$ of degree at most $k-1$ which satisfies that $f(i) = \mathsf{m}(i)$ for any $i \in [k']$ and $f(i) = v(i)$ for any $i \in I$ (such a polynomial exists since $|[k'] \cup I| = k' + (k - k') = k$). Let $r := (f(i))_{i \in [k]\setminus[k']}$. By the definition of $\mathsf{PRS}_{k,n}$, it follows that $\mathsf{PRS}_{k,n}(\mathsf{m}, r) = (f'(i))_{i\in[n]\setminus[k]}$, where $f'(i)$ is the unique univariate polynomial over $\mathbb{F}$ of degree at most $k-1$ which satisfies that $f'(i) = f(i)$ for any $i \in [k]$. But since $f(i)$ is a univariate polynomial of degree at most $k-1$ over $\mathbb{F}$, we must have that $f'(i) = f(i)$ for every $i \in [n]$,and consequently $\mathsf{PRS}_{k,n}(\mathsf{m}, r)|_I = (f(i))_{i\in I} = v$. We conclude that for any string $v \in \mathbb{F}^{k-k'}$, there exists (a unique) $r \in \mathbb{F}^{k-k'}$ so that $\mathsf{PRS}_{k,n}(\mathsf{m}, r)|_I = v$, which completes the proof of the lemma. ■

The next Lemma bounds the ZK parameter of the tensor product of PRS codes.

Lemma 4.15. *Let $\mathbb{F}$ be a finite field of size n, and identify the elements of $\mathbb{F}$ with $[n]$. Let $k' \leq k$ be a non-negative integer so that $3k - k' \leq n$, and let $\mathsf{PRS}_{k,n} : \mathbb{F}^{[k]} \to \mathbb{F}^{[n]\setminus[k]}$ be the punctured Reed-Solomon code. Then $\mathsf{PRS}_{k,n} \otimes \mathsf{PRS}_{k,n}$ does not have (non-adaptive) $(2k-k')$-ZK with respect to the $(k' \times k')$-randomized encoding function.*

Proof: By Remark 3.20, it suffices to show a subset $I \subseteq ([n] \setminus [k]) \times ([n] \setminus [k])$ of size $|I| = 2k - k'$, and a pair of messages $\mathsf{m}_1, \mathsf{m}_2 \in \mathbb{F}^{k' \times k'}$, so that $\mathsf{Enc}(\mathsf{m}_1)|_I \not\equiv \mathsf{Enc}(\mathsf{m}_2)|_I$, where Enc is the $(k' \times k')$-randomized encoding function. Let $I = \{(i,i) \mid i \in [3k - k'] \setminus [k]\}$ (recalling our assumption that $3k - k' \leq n$), let $\mathsf{m}_1 \in \mathbb{F}^{k'\times k'}$ be the all zeros message, and let $\mathsf{m}_2 \in \mathbb{F}^{k'\times k'}$ be the message satisfying that $\mathsf{m}_2(1,1) = 1$ and $\mathsf{m}_2(i,j) = 0$ otherwise. We shall show below that the distributions $\mathsf{Enc}(\mathsf{m}_1)|_I$ and $\mathsf{Enc}(\mathsf{m}_2)|_I$ have disjoint support, and consequently $\mathsf{Enc}(\mathsf{m}_1)|_I \not\equiv \mathsf{Enc}(\mathsf{m}_2)|_I$.

To show the above, suppose on the contrary that there exist $r_1, r_2 \in \mathbb{F}^{k-k'}$ so that $c_1|_I = c_2|_I$, where $c_1 = \mathsf{PRS}_{k,n}(\mathsf{m}_1, r_1)$ and $c_2 = \mathsf{PRS}_{k,n}(\mathsf{m}_2, r_2)$. By Claim 3.38, there exist bivariate polynomials $g_1(x,y), g_2(x,y)$ of individual degree at most $k-1$ over $\mathbb{F}$ so that $g_1(i,j) = \mathsf{m}_1(i,j)$ ($g_2(i,j) = \mathsf{m}_2(i,j)$, respectively) for any $(i,j) \in [k'] \times [k']$, and $g_1(i,j) = c_1(i,j)$ ($g_2(i,j) = c_2(i,j)$, respectively) for any $(i,j) \in ([n] \setminus [k]) \times ([n] \setminus [k])$.

Let $h_1(x) = g_1(x, x)$ and $h_2(x) = g_2(x, x)$. Then $h_1(x)$ and $h_2(x)$ are univariate polynomials of degree at most $2k - 2$ over $\mathbb{F}$. Moreover, $h_1(i) = h_2(i)$ for any $i \in [k'] \setminus \{1\}$ by assumption that $\mathsf{m}_1(i, i) = \mathsf{m}_2(i, i) = 0$ for any $i \in [k'] \setminus \{1\}$, and $h_1(i) = h_2(i)$ for any $i \in [3k - k'] \setminus [k]$ by assumption that $c_1|_I = c_2|_I$. So we conclude that $h_1(x)$ and $h_2(x)$ are two univariate polynomials of degree at most $2k - 2$ over $\mathbb{F}$ that agree on at least $2k - 1$ points, and so we must have that $h_1(x) = h_2(x)$ as polynomials in $\mathbb{F}[X]$. But this implies in turn that $\mathsf{m}_1(1, 1) = h_1(1) = h_2(1) = \mathsf{m}_2(1, 1)$, a contradiction. ■

The above Lemmas 4.14 and 4.15 imply the following Corollary.

Corollary 4.16. *Let $\mathbb{F}$ be a finite field of size n, and identify the elements of $\mathbb{F}$ with $[n]$. Let $k' \leq k \leq \frac{n}{2}$ be non-negative integers, and let $\mathsf{PRS}_{k,n} : \mathbb{F}^{[k]} \to \mathbb{F}^{[n] \setminus [k]}$ be the Punctured Reed-Solomon code. Then the following holds for any $\epsilon > 0$:*

- *(Low-rate setting, $k' = \epsilon k$) If $3k \leq n$, then $\mathsf{PRS}_{k,n}$ has $((1 - \epsilon)k)$-uniform ZK with respect to the (ϵk)-randomized encoding function, but $\mathsf{PRS}_{k,n} \otimes \mathsf{PRS}_{k,n}$ does not have (non-adaptive) $2k$-ZK with respect to the $((\epsilon k) \times (\epsilon k))$-randomized encoding function.*
- *(High-rate setting, $k' = (1 - \epsilon)k$) If $(2 + \epsilon)k \leq n$, then $\mathsf{PRS}_{k,n}$ has ϵk-uniform ZK with respect to the $((1 - \epsilon)k)$-randomized encoding function, but $\mathsf{PRS}_{k,n} \otimes \mathsf{PRS}_{k,n}$ does not have (non-adaptive) $(1 + \epsilon)k$-ZK with respect to the $(((1 - \epsilon)k) \times (1 - \epsilon)k))$-randomized encoding function.*

4.2.2 Linear ZK Threshold

In this section, we restate a result from [ISVW13] which says that for *any* linear code $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$, there *exists* a randomized encoding function (not necessarily the $(k' \times k')$-randomized encoding function) with respect to which $\mathcal{C} \otimes \mathcal{C}$ has $\Omega(n^2)$-ZK. This result implies as a corollary that for any linear code $\mathcal{C}$, there exists a randomized encoding function with respect to which the tensor product $\mathcal{C}^{\otimes d}$ is a ZK-LTC. As we shall need a more quantitative version of this result (specifically, codes of high rate), for completeness we provide below a more quantitative statement and formal proof.

The aforementioned result is an immediate consequence of the following theorem from [ISVW13], which says that *any* linear code can be turned into a ZK code by changing the encoding function (but not the set of codewords).

Theorem 4.17 ([ISVW13], Theorem 1)**.** *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code with a generator matrix G. Let k' and t be non-negative integers which satisfy that $H_q(\frac{t}{n}) < \frac{k-k'}{n}$, where*

$$H_q(x) = x \log_q((q - 1)/x) + (1 - x) \log_q(1/(1 - x))$$

is the q-ary entropy function. Then there exists an invertible $k \times k$ binary matrix G' so that the code $C' : \mathbb{F}^k \to \mathbb{F}^n$, given by $C'(\mathsf{m}) = (G \cdot G')(\mathsf{m})$ for any $\mathsf{m} \in \mathbb{F}^k$, is t-ZK with respect to the k'-randomized encoding function. Moreover, G' can be constructed in probabilistic polynomial time except with negligible failure probability given k', t, and G.

Remark 4.18. *The proof of [ISVW13, Thm. 1] shows that for every subset $I \subseteq [n]$ of size t, and for any pair of messages $\mathsf{m}, \mathsf{m}' \in \mathbb{F}^{k'}$ it holds that $\mathsf{Enc}(\mathsf{m})|_I \equiv \mathsf{Enc}(\mathsf{m}')|_I$, where Enc is the k'-randomized encoding function for C'. Note that this in particular implies t-ZK, as noted in Remark 3.20.*

As observed in [ISVW13], it follows from the above theorem that there exists a randomized encoding function with respect to which the tensor product $\mathcal{C}^{\otimes d}$ is a ZK-LTC (i.e., it is an LTC in which the zero-knowledge parameter is significantly larger than the query complexity).

Corollary 4.19 (ZK-LTC Tensor Codes, [ISVW13])**.** *Let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be an explicit linear code ensemble of relative distance δ and rate $1-\gamma$. Then for any integer $d > 1$, there exists an efficient randomized construction of a linear code $\mathcal{C}' : \mathbb{F}^{k^d} \to \mathbb{F}^{[n]^d}$ so that $\mathrm{image}(\mathcal{C}') = \mathrm{image}(\mathcal{C}^{\otimes d})$, and $\mathcal{C}'$ is an $(n, \delta^{O(d)})$-LTC and $(n^2, \delta^{O(d)})$-robust LTC that has $(\gamma' \cdot n^d)$-ZK with respect to the $((1-\gamma)\cdot k^d)$-randomized encoding function* Enc*, where $\gamma' = H_q^{-1}(\gamma \cdot (1-\gamma)^d)$.*

In particular, $\mathcal{C}^{\otimes d}$ is an $(n, \delta^{O(d)})$-LTC and $(n^2, \delta^{O(d)})$-robust LTC that has $(\gamma' \cdot n^d)$-ZK with respect to the randomized encoding function Enc*.*

Proof: Let $\mathcal{C}'$ be the code given by Theorem 4.17 for the code $\mathcal{C}^{\otimes d}$, $k' = (1-\gamma)\cdot k^d$, and $t = H_q^{-1}(\gamma \cdot (1-\gamma)^d) \cdot n^d$.

Then we clearly have that $\mathrm{image}(\mathcal{C}') = \mathrm{image}(\mathcal{C}^{\otimes d})$. Moreover, by Theorem 3.33, $\mathcal{C}^{\otimes d}$ is an $(n, \delta^{O(d)})$-LTC and $(n^2, \delta^{O(d)})$-robust LTC, and so the same holds for the code $\mathcal{C}'$ since it has the same set of codewords as $\mathcal{C}^{\otimes d}$. Finally, our choice of parameters implies that

$$\frac{k^d - k'}{n^d} = \frac{\gamma k^d}{n^d} =^{\text{Fact 3.26}} \gamma \cdot (1-\gamma)^d = H_q\left(\frac{t}{n^d}\right),$$

and so by Theorem 4.17, $\mathcal{C}'$ is t-ZK with respect to the k'-randomized encoding function.

The 'in particular' part follows by noting that Enc is also a randomized encoding function for $\mathcal{C}^{\otimes d}$ since $\mathrm{image}(\mathcal{C}') = \mathrm{image}(\mathcal{C}^{\otimes d})$.

■

We note that it is not known how to explicitly construct the randomized encoding function Enc for $\mathcal{C}^{\otimes d}$ given by the above corollary, and obtaining such an explicit randomized encoding function is an interesting question for future research.

4.3 Code Extension

In this section we present a transformation that extends a tensor code over a field $\mathbb{H}$ into a tensor code over a larger field $\mathbb{F}$, while preserving its zero-knowledge properties.

We start by setting some notation. Let $\mathbb{H} \subseteq \mathbb{F}$ be finite fields, and let A be a basis for $\mathbb{F}$ over $\mathbb{H}$. We view $\mathbb{F}$ as a $b := \log_{|\mathbb{H}|}(|\mathbb{F}|)$ dimensional vector space over $\mathbb{H}$ in the natural way (e.g., A can be the standard basis). Every element in $\mathbb{F}$ can be expressed as a linear combination of the basis elements. Extending this fact to vectors, any $z \in \mathbb{F}^n$ can be expressed as $z = \sum_{a \in A} a \cdot z^{(a)}$, for some $\{z^{(a)}\}_{a \in A}$, where $z^{(a)} \in \mathbb{H}^n$ for every $a \in A$ are determined by z.

Definition 4.20 (Extended Code)**.** *Let $C : \mathbb{H}^k \to \mathbb{H}^n$ be a code, let $\mathbb{F}$ be a finite extension field of $\mathbb{H}$, and let A be a basis for $\mathbb{F}$ over $\mathbb{H}$. The A-*extension of *C is the code $\widetilde{C} : \mathbb{F}^k \to \mathbb{F}^n$, given by $\widetilde{C}(\mathsf{m}) = \sum_{a \in A} a \cdot C(\mathsf{m}^{(a)})$ for any $\mathsf{m} \in \mathbb{F}^k$.*

Lemma 4.21. *Suppose that $C : \mathbb{H}^k \to \mathbb{H}^n$ is a linear code of relative distance δ. Let $\mathbb{F}$ be a finite extension field of $\mathbb{H}$, let A be a basis for $\mathbb{F}$ over $\mathbb{H}$, and let $b := |A| = \log_{|\mathbb{H}|}(|\mathbb{F}|)$. Let $\widetilde{C} : \mathbb{F}^k \to \mathbb{F}^n$ be the A-extension of C. Then the following hold:*

- *$\widetilde{C}$ is a linear code of relative distance at least δ.*
- *$(\widetilde{C})^{\otimes d} = \widetilde{C^{\otimes d}}$.*
- *If $1 \in A$, then $\widetilde{C}(\mathsf{m}) = C(\mathsf{m})$ for any $\mathsf{m} \in \mathbb{H}^k$. In particular, if $\mathcal{C} : \mathbb{H}^k \to \mathbb{H}^n$ is a linear code ensemble that has t-ZK with respect to a randomized encoding function* Enc*, then $\widetilde{\mathcal{C}}$ has ZK on $\mathbb{H}^k$ with respect to* Enc *(cf. Definition 3.21).*

Proof: We prove the three bullets of the lemma.

$\widetilde{C}$ **is a linear code.** We need show that for any $\mathsf{m}, \mathsf{m}_1, \mathsf{m}_2 \in \mathbb{F}^k$, and any $\lambda \in \mathbb{F}$, we have $\widetilde{C}(\lambda \cdot \mathsf{m}) = \lambda \cdot \widetilde{C}(\mathsf{m})$ and $\widetilde{C}(\mathsf{m}_1 + \mathsf{m}_2) = \widetilde{C}(\mathsf{m}_1) + \widetilde{C}(\mathsf{m}_2)$. First note that by linearity of C we have that:

$$\begin{aligned}
&\widetilde{C}(\mathsf{m}_1 + \mathsf{m}_2) = \sum_{a \in A} a \cdot C((\mathsf{m}_1 + \mathsf{m}_2)^{(a)}) = \sum_{a \in A} a \cdot C(\mathsf{m}_1^{(a)} + \mathsf{m}_2^{(a)}) \\
&= \sum_{a \in A} a \cdot \left(C(\mathsf{m}_1^{(a)}) + C(\mathsf{m}_2^{(a)}) \right) = \sum_{a \in A} a \cdot C(\mathsf{m}_1^{(a)}) + \sum_{a \in A} a \cdot C(\mathsf{m}_2^{(a)}) \\
&= \widetilde{C}(\mathsf{m}_1) + \widetilde{C}(\mathsf{m}_2).
\end{aligned}$$

Additionally, for any $\lambda \in \mathbb{F}$ and $\mathsf{m} \in \mathbb{F}^k$, we have that:

$$\begin{aligned}
&\widetilde{C}(\lambda \cdot \mathsf{m}) = \widetilde{C}\left(\lambda \sum_{a \in A} a \cdot \mathsf{m}^{(a)} \right) = \widetilde{C}\left(\sum_{a \in A} (\lambda \cdot a) \cdot \mathsf{m}^{(a)} \right) \\
&= \widetilde{C}\left(\sum_{a \in A} \left(\sum_{a' \in A} a' \cdot (\lambda \cdot a)^{(a')} \right) \cdot \mathsf{m}^{(a)} \right) = \widetilde{C}\left(\sum_{a' \in A} a' \left(\sum_{a \in A} (\lambda \cdot a)^{(a')} \cdot \mathsf{m}^{(a)} \right) \right) \\
&=^{(*)} \sum_{a' \in A} a' \cdot C\left(\sum_{a \in A} (\lambda \cdot a)^{(a')} \cdot \mathsf{m}^{(a)} \right) = \sum_{a' \in A} a' \sum_{a \in A} (\lambda \cdot a)^{(a')} C(\mathsf{m}^{(a)}) \\
&= \sum_{a \in A} C(\mathsf{m}^{(a)}) \left(\sum_{a' \in A} a' \cdot (\lambda \cdot a)^{(a')} \right) = \sum_{a \in A} C(\mathsf{m}^{(a)}) \cdot (\lambda \cdot a) \\
&= \lambda \sum_{a \in A} a \cdot C(\mathsf{m}^{(a)}) = \lambda \cdot \widetilde{C}(\mathsf{m}).
\end{aligned}$$

where the equality denoted by (*) follows by the definition of $\widetilde{C}$, noting that $\sum_{a \in A} (\lambda \cdot a)^{(a')} \cdot \mathsf{m}^{(a)} \in \mathbb{H}$ for any $a' \in A$.

$\widetilde{C}$ **has relative distance at least** δ**.** Suppose that $\mathsf{m}_1, \mathsf{m}_2 \in \mathbb{F}^k$ are a pair of distinct messages, and let $c_1 = \widetilde{C}(\mathsf{m}_1)$ and $c_2 = \widetilde{C}(\mathsf{m}_2)$. Then there exists an $a \in A$ so that $\mathsf{m}_1^{(a)} \neq \mathsf{m}_2^{(a)}$. Since C has relative distance δ, we have that $C(\mathsf{m}_1^{(a)})$ and $C(\mathsf{m}_2^{(a)})$ differ by at least a δ-fraction of the entries. But by the definition of $\widetilde{C}$, this implies in turn that $c_1^{(a)} = C(\mathsf{m}^{(a)})$ and $c_2^{(a)} = C(\mathsf{m}_2^{(a)})$ differ by at least a δ-fraction of the entries, and so also c_1 and c_2 differ by at least a δ-fraction of the entries.

Finally, the last two bullets are an immediate consequence of the definition of the extended code. ■

5 Sublinear length ZK-IOP for sumcheck

In this section, we present our sublinear sumcheck protocol, and analyze its properties. We first formally define the sumcheck relation, and its corresponding promise problem.

Definition 5.1 (Sumcheck Relation)**.** *Let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code ensemble, and let $d \in \mathbb{N}$. The sumcheck relation $\mathcal{R}_{\mathsf{SChk}}^{\mathsf{YES}}$ for $\mathcal{C}^{\otimes d}$ consists of all tuples of the form $((\lambda_1, \ldots, \lambda_d, \alpha), c)$ such that:*

- $c = C^{\otimes d}(\mathsf{m})$ *for some* $\mathsf{m} \in \mathbb{F}^{[k]^d}$.
- $\lambda_1, \ldots, \lambda_d \in \mathbb{F}^k$.

- $\langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle = \alpha$.

Here, $(\lambda_1, \ldots, \lambda_d, \alpha)$ is the explicit input, and c is the implicit input.

$\mathcal{R}^{\mathsf{NO}}_{\mathsf{SChk}}$ is defined similarly to $\mathcal{R}^{\mathsf{YES}}_{\mathsf{SChk}}$ except that tuples $((\lambda_1, \ldots, \lambda_d, \alpha), c)$ in $\mathcal{R}^{\mathsf{NO}}_{\mathsf{SChk}}$ satisfy that $\langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle \neq \alpha$ (in particular, $c \in C^{\otimes d}$).

The sumcheck promise problem for $\mathcal{C}^{\otimes d}$ is the pair $\left(\mathcal{R}^{\mathsf{YES}}_{\mathsf{SChk}}, \mathcal{R}^{\mathsf{NO}}_{\mathsf{SChk}}\right)$.

The main result of this section is the following theorem:

Theorem 5.2. *The following holds for any integer $d > 1$ and constants $\delta, \mu > 0$. Let $\{\mathbb{F}_k\}_{k \in \mathbb{N}}$ be a constructible finite field ensemble, and let $\mathcal{C} = \{C_k : (\mathbb{F}_k)^k \to (\mathbb{F}_k)^n\}_{k \in \mathbb{N}}$ be an explicit linear code ensemble of relative distance δ that is an efficient (q_k, μ)-robust LTC.*

Then there exists a $(d+2)$-round IOP $(\mathcal{P}, \mathcal{V})$ for the sumcheck promise problem $\left(\mathcal{R}^{\mathsf{YES}}_{\mathsf{SChk}}, \mathcal{R}^{\mathsf{NO}}_{\mathsf{SChk}}\right)$ for $\mathcal{C}^{\otimes d}$, with soundness error $1 - \min\{\frac{\delta^2 \cdot \mu^2}{16}, \frac{\delta^d}{4}\}$ with a verifier that queries a single symbol from c and constant number of symbols from the prover's first message (and reads the other prover messages in full). The communication complexity of the system is $O(d \cdot n \cdot \log(|\mathbb{F}_k|))$, the verifier's running time is $\mathsf{poly}(d, k, \log(|\mathbb{F}_k|)) + \mathsf{poly}(q_k, \log(|\mathbb{F}_k|))$, and the prover's running time is $n^d \cdot \mathsf{poly}(d, k, \log(|\mathbb{F}_k|))$.

Furthermore, the system has the following zero-knowledge guarantees:

1. **ZK (General Codes).** *For any natural t there exists a black-box straight-line PPT simulator Sim which is $((t + q_k + 1), \{2, \ldots, d\})$-restricted such that for every t-restricted verifier $\mathcal{V}^*$, and for every $((\lambda_1, \ldots, \lambda_d, \alpha), c) \in \mathcal{R}^{\mathsf{YES}}_{\mathsf{SChk}}$,*

$$\left(\mathsf{View}^{\mathsf{Sim}}_{\mathcal{V}^*}\left((\lambda_1, \ldots, \lambda_d, \alpha), c\right), q_S - q_k - 1\right)_{r_V, r_{\mathsf{Sim}}} \equiv \left(\mathsf{View}_{\mathcal{V}^*}\left((\lambda_1, \ldots, \lambda_d, \alpha), c\right), q_V\right)_{r_P, r_V}$$

   where $r_P, r_V, r_{\mathsf{Sim}}$ are the random coins of the prover, verifier $\mathcal{V}^*$, and Sim, respectively; q_S denotes the number of queries which Sim makes to its row oracle; q_V denotes the number of queries which $\mathcal{V}^*$ makes to her oracles c and the first prover message; and $\mathsf{View}^{\mathsf{Sim}}_{\mathcal{V}^*}$ is as defined in Definition 3.15.

2. **ZK (Tensor Codes).** *If $C_k = (B_k)^{\otimes 3}$ for some code B_k, and the robust local tester is the two-dimensional plane tester given by Theorem 3.33, then for any natural t PPT simulator Sim which is adaptive $(t+2)$-subspace restricted with respect to the code $(B_k)^{\otimes 3d}$ such that for every t-restricted verifier $\mathcal{V}^*$, and for every $((\lambda_1, \ldots, \lambda_d, \alpha), c) \in \mathcal{R}^{\mathsf{YES}}_{\mathsf{SChk}}$,*

$$\left(\mathsf{View}^{\mathsf{Sim}}_{\mathcal{V}^*}\left((\lambda_1, \ldots, \lambda_d, \alpha), c\right), q_S - 2\right)_{r_V, r_{\mathsf{Sim}}} \equiv \left(\mathsf{View}_{\mathcal{V}^*}\left((\lambda_1, \ldots, \lambda_d, \alpha), c\right), q_V\right)_{r_P, r_V} \quad (3)$$

   where $r_P, r_V, r_{\mathsf{Sim}}$ are the random coins of the prover, verifier $\mathcal{V}^*$, and Sim, respectively; q_S denotes the number of queries which Sim makes to its oracle; and q_V denotes the number of queries which $\mathcal{V}^*$ makes to her oracles c and the first prover's message. Moreover, during the simulation, the subspaces that Sim chooses for its oracle are in directions $[3d] \setminus \{1\}$, $[3d] \setminus \{2\}$, or $[3d] \setminus \{3\}$.

We note that the only difference between the ZK property of Theorem 5.2 and the notion of black-box straight-line ZK (Definition 3.15) is that the simulator is given a *line/subspace* oracle for the implicit input c of the verifier, instead of a standard (point) oracle for c.

Remark 5.3 (On the Two flavors of ZK of Our Sumcheck ZK-IOP.)**.** *Our sumcheck ZK-IOP achieves two different flavors of zero-knowledge: a ZK guarantee for general base codes with an additive loss in the ZK parameter, and a ZK guarantee tailored for tensor base codes with a reduced loss in the ZK parameter. We present both flavors because they are incomparable: the former achieves a weaker ZK guarantee, but is applicable to a wider class of codes. Both variants are used in our ZK-IOPs of Section 6. Specifically, the*

former is applied to a high-rate code, which is chosen specifically such that it is a ZK LTC (i.e., the query complexity of the local tester is lower than the ZK LTC parameter). The latter is applied specifically to the LDE encoding, which is not a ZK LTC. Therefore, using the version for general codes would not provide any ZK guarantees. Instead, we utilize the tensor structure of LDEs and use the sumcheck ZK-IOP version for tensor codes to eliminate the loss in the ZK parameter.

Theorem 5.2 will follow as a corollary of the following, more general, theorem, which summarizes the properties of our sumcheck ZK-IOP:

Theorem 5.4 (Sublinear length ZK-IOP for Sumcheck)**.** *The following holds for any integer $d > 1$ and constant $\delta, \mu > 0$. Let $\{\mathbb{F}_k\}_{k\in\mathbb{N}}$ be a constructible finite field ensemble, and let $\{C_k : (\mathbb{F}_k)^k \to (\mathbb{F}_k)^n\}_{k\in\mathbb{N}}$ be an explicit linear code ensemble of relative distance δ that is an efficient (q_k, μ)-robust LTC.*

Then there exists an interactive oracle proof $(\mathcal{P}, \mathcal{V})$ satisfying the following properties:

- **Prover $\mathcal{P}$'s Input:** *$\mathcal{P}$ receives as input d vectors $\lambda_1, \ldots, \lambda_d \in (\mathbb{F}_k)^k$, a scalar $\alpha \in \mathbb{F}_k$, and a codeword $c = (C_k)^{\otimes d}(\mathsf{m}) \in (\mathbb{F}_k)^{[n]^d}$.*
- **Verifier $\mathcal{V}$'s Input:** *$\mathcal{V}$ receives as input $\lambda_1, \ldots, \lambda_d$ and α, and also receives oracle access to c.*
- **Communication phase:** *$\mathcal{P}$ and $\mathcal{V}$ interact for $d + 2$ rounds, where $\mathcal{P}$'s messages are strings over $\mathbb{F}$, and $\mathcal{V}$'s messages are uniformly random strings.*
- **Query phase:** *At the end of the interaction, $\mathcal{V}$ computes a uniformly random point $(i_1, \ldots, i_d) \in [n]^d$ (depending only on $\mathcal{V}$'s randomness string), queries a constant number of field elements out of $\mathcal{P}$'s first message, and fully reads the rest of $\mathcal{P}$'s messages. She then either accepts, rejects, or outputs $(i_1, \ldots, i_d)$ and a scalar $\alpha' \in \mathbb{F}_k$.*[21]
- **Completeness:** *If $((\lambda_1, \ldots, \lambda_d, \alpha), c) \in \mathcal{R}^{\mathsf{YES}}_{\mathsf{SChk}}$ then when $\mathcal{V}$ interacts with $\mathcal{P}$, with probability 1, $\mathcal{V}$ either accepts, or outputs $(i_1, \ldots, i_d)$ and α' such that $c(i_1, \ldots, i_d) = \alpha'$.*
- **Soundness:** *If $((\lambda_1, \ldots, \lambda_d, \alpha), c) \in \mathcal{R}^{\mathsf{NO}}_{\mathsf{SChk}}$ then for any prover strategy $\mathcal{P}^*$, when $\mathcal{V}$ interacts with $\mathcal{P}^*$, then with probability at least $\min\{\frac{\delta^2\cdot\mu^2}{16}, \frac{\delta^d}{4}\}$, $\mathcal{V}$ either rejects, or outputs $(i_1, \ldots, i_d)$ and α' such that $c(i_1, \ldots, i_d) \neq \alpha'$.*
- **Zero knowledge:** *the system has the two ZK properties specified in Theorem 5.2.*

The interactive oracle proof $(\mathcal{P}, \mathcal{V})$ has the same communication complexity and verifier and prover running times as stated in Theorem 5.2.

Remark 5.5 (Tighter ZK simulation in Theorem 5.4)**.** *Notice that for general codes, the simulator in Theorems 5.2 and 5.4 makes $q_k + 1$ more oracle queries than $\mathcal{V}^*$ (because $q_S = q_V + q_k + 1$). This is because the local tester for $C^{\otimes d}$ has no ZK guarantees. Therefore, to simulate the prover messages in these internal systems, the simulator needs to generate the tester's/prover's inputs in these systems, which are $u|_I$ and $\gamma \cdot c(i_1, *) + R(i_1, *)$, respectively. The former can be simulated by reading $|I| = q_k$ rows of c, whereas the latter requires a single row of c.*

We note that the number of rows the simulator reads can be reduced to $q_V + 1$ by changing the IOP of Figure 3 as follows: the honest verifier $\mathcal{V}$ directly runs the local tester in Step 1b (instead of obtaining v from $\mathcal{P}$ in Step 1(b)ii). This will increase the query complexity of the honest verifier by an additive q_k, but now the simulator need not fully simulate $u|_I$, so it will only need to query $q_V + 1$ rows of c.

[21] We note that $\mathcal{V}$ does not make any queries to c, but the malicious verifier in the ZK property might query c.

In the setting of general codes, we chose the former option, since it guarantees the property that the verifier makes a constant number of queries to each prover's oracle message, which will in turn simplify the query reduction (composition) step. We stress however that in the case of tensor codes, it is crucial for us that the verifier does not directly run the local tester. This is because in this setting, the query complexity of the local tester may be larger than the zero-knowledge threshold of the code, and consequently we cannot afford to have the verifier make these queries directly.

Proof of Theorem 5.2. An IOP $(\mathcal{P}_{\mathsf{SChk}}, \mathcal{V}_{\mathsf{SChk}})$ satisfying the properties stated in Theorem 5.2 can be obtained from the IOP $(\mathcal{P}, \mathcal{V})$ of Theorem 5.4 by having $\mathcal{P}_{\mathsf{SChk}}, \mathcal{V}_{\mathsf{SChk}}$ emulate $\mathcal{P}, \mathcal{V}$, and when $\mathcal{V}$ terminates, if she outputs $(i_1, \ldots, i_d)$ and α' then $\mathcal{V}_{\mathsf{SChk}}$ queries $c(i_1, \ldots, i_d)$ and accepts if $c(i_1, \ldots, i_d) = \alpha'$, otherwise $\mathcal{V}_{\mathsf{SChk}}$ rejects. Then completeness and soundness, as well as the structure and complexities stated in Theorem 5.2 follow directly from the properties stated in Theorem 5.4. The ZK property also follows from Theorem 5.2 since the additional query of the honest verifier does not affect the behaviour of a malicious verifier (and the increased query complexity of the honest verifier is still within the ZK bound t). ■

To prove Theorem 5.4 we rely on the following (non zero-knowledge) IOP for sumcheck, given in [RR20] (see also the full version [RR19]).

Theorem 5.6 (IOP for Sumcheck, [RR19], Lemma 6.1)**.** *The following holds for any integer $d > 1$ and $\delta > 0$. Let $\{\mathbb{F}_k\}_{k\in\mathbb{N}}$ be a constructible finite field ensemble, and let $\{C_k : (\mathbb{F}_k)^k \to (\mathbb{F}_k)^n\}_{k\in\mathbb{N}}$ be an explicit linear code ensemble of relative distance δ.*

Then there exists an interactive proof $(\mathcal{P}, \mathcal{V})$ satisfying the following properties:

- **Prover $\mathcal{P}$'s Input:** *$\mathcal{P}$ receives as input d vectors $\lambda_1, \ldots, \lambda_d \in (\mathbb{F}_k)^k$, a scalar $\alpha \in \mathbb{F}_k$, and a codeword $c = (C_k)^{\otimes d}(\mathsf{m}) \in (\mathbb{F}_k)^{[n]^d}$.*
- **Verifier $\mathcal{V}$'s Input:** *$\mathcal{V}$ receives as input $\lambda_1, \ldots, \lambda_d$ and α.*[22]
- **Communication phase:** *$\mathcal{P}$ and $\mathcal{V}$ interact for d rounds, where $\mathcal{P}$'s messages are strings over $\mathbb{F}$, and $\mathcal{V}$'s messages are uniformly random strings.*
- **Query phase:** *At the end of the interaction, $\mathcal{V}$ computes a uniformly random point $(i_1, \ldots, i_d) \in [n]^d$ (depending on $\mathcal{V}$'s randomness string) and reads all of $\mathcal{P}$'s messages. It then either rejects, or outputs $(i_1, \ldots, i_d)$ and a scalar $\alpha' \in \mathbb{F}_k$.*
- **Completeness:** *If $\langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle = \alpha$, then when $\mathcal{V}$ interacts with $\mathcal{P}$, with probability 1, $\mathcal{V}$ outputs $(i_1, \ldots, i_d)$ and α' such that $c(i_1, \ldots, i_d) = \alpha'$.*
- **Soundness:** *If $\langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle \neq \alpha$, then for any prover strategy $\mathcal{P}^*$, when $\mathcal{V}$ interacts with $\mathcal{P}^*$, then with probability at least δ^d, $\mathcal{V}$ either rejects, or outputs $(i_1, \ldots, i_d)$ and α' such that $c(i_1, \ldots, i_d) \neq \alpha'$.*

  The interactive oracle proof $(\mathcal{P}, \mathcal{V})$ has communication complexity $d \cdot n \cdot \log(|\mathbb{F}_k|)$, verifier running time $\mathsf{poly}(d, k, \log(|\mathbb{F}_k|))$, and prover running time $n^d \cdot \mathsf{poly}(d, k, \log(|\mathbb{F}_k|))$.

Remark 5.7. *We note the following changes in the above Theorem 5.6, compared to [RR19, Lemma 6.1]:*

1. *[RR19, Lemma 6.1] was proven for the special case where $\mathbb{F}_k$ is the binary field (and the vectors $\lambda_1, \ldots, \lambda_d$ are possibly over a larger field of characteristic 2). However, it can be verified that the proof can be extended to any finite field.*

[22] We note that in this protocol, $\mathcal{V}$ does not receive oracle access to c.

2. *[RR19, Lemma 6.1] was proven for the special case where C_k is a systematic code. However, the protocol can be extended to any linear code, by modifying the check in step (2a) to be that $\tilde{w}_\ell = C_n(y)$ for $y \in \{0,1\}^n$ which satisfies that $\sum_{j\in[n]} \lambda_\ell(j) \cdot y(j) = b_{\ell-1}$. Such a y can be efficiently found by solving a system of linear equations, given a parity-check matrix for the code.*

3. *In [RR19, Lemma 6.1], the verifier receives oracle access to c, and in the query phase the verifier computes a uniformly random point $(i_1, \ldots, i_d) \in [n]^d$ (depending on $\mathcal{V}$'s randomness string), reads all of $\mathcal{P}$'s messages, and based on these it either rejects, or computes a value $\alpha' \in \{0,1\}$. The verifier then queries $c(i_1, \ldots, i_d)$ and accepts if and only if $c(i_1, \ldots, i_d) = \alpha'$. In the current protocol, the verifier does not receive oracle access to c, and in the query phase it just outputs $(i_1, \ldots, i_d)$ and α', but does not make the actual query to c.*

5.1 Warmup: the 2-Dimensional Case

In this section we describe a special case of our sumcheck ZK-IOP for the 2-dimensional setting for sums of the form $\sum_{i\in[k]} \mathsf{m}(i) = \alpha$ (i.e., with no tensor coefficients). The full protocol is given in Section 5.2 below. We begin with a more detailed overview of the sumcheck protocol of [RR20], then highlight the differences introduced in our protocol.

Sumcheck for Tensor Codes of [RR20]. Let $C_0 : \mathbb{F}^k \to \mathbb{F}^n$ be a linear base code, and let $C = C_0 \otimes C_0$ be its two-dimensional tensor product. Ron-Zewi and Rothblum's sumcheck for a codeword $c = C(\mathsf{m})$ and the claim $\sum_{i,j\in[k]} \mathsf{m}(i,j) = \alpha$ (where $\mathsf{m}(i,j)$ denotes the entry in row i and column j in m, when viewed as a $k \times k$ matrix), operates as follows:

1. $\mathcal{P}$ computes the sum $z := \sum_{j\in[k]} C_0(\mathsf{m}(*,j))$ of the columns' encodings with the code C_0, where $\mathsf{m}(*,j)$ denotes the j'th column of m, when viewed as a $k \times k$ matrix.

   Then, $\mathcal{P}$ sends z to $\mathcal{V}$ as an explicit message, i.e., which $\mathcal{V}$ reads in full.

2. $\mathcal{V}$ checks that $z = C_0(y)$ for some $y \in \mathbb{F}^k$ such that $\sum_{i\in[k]} y(i) = \alpha$, and rejects otherwise.
   ▷ *Notice that by linearity, $z = C_0(\sum_{j\in[k]} \mathsf{m}(*,j))$, where*

$$\sum_{i\in[k]} \sum_{j\in[k]} \mathsf{m}(*,j) = \sum_{i,j\in[k]} \mathsf{m}(i,j) = \alpha.$$

3. $\mathcal{V}$ samples a random row $i_0 \leftarrow [n]$ and sends i_0 to $\mathcal{P}$.

4. $\mathcal{P}$ sends to $\mathcal{V}$ the i_0th row $z' := c(i_0, *)$ of c, where c is viewed as an $n \times n$ matrix.

5. $\mathcal{V}$ checks that $z' = C_0(y')$ for $y' \in \mathbb{F}^k$ such that $\sum_{j\in[k]} y'(j) = z(i_0)$.
   ▷ *Notice that by the definition of the encoding function for the tensor product, $c(i_0, *) = C_0(y')$ for $y' \in \mathbb{F}^k$ whose jth entry is $C_0(\mathsf{m}(*,j))(i_0)$. Consequently,*

$$\sum_{j\in[k]} y'(j) = \sum_{j\in[k]} C_0(\mathsf{m}(*,j))(i_0) = z(i_0).$$

6. $\mathcal{V}$ samples a random $j_0 \leftarrow [n]$, queries $c(i_0, j_0)$, and accepts if and only if $z'(j_0) = c(i_0, j_0)$.

Our Sumcheck ZK-IOPs with Sublinear Communication. Intuitively, the IOP of [RR20] reveals to the verifier the sum of columns in c, as well as a single row of c (when viewed as an $n \times n$ matrix). As discussed in Section 2.2, we obtain ZK by masking these with a random codeword *in the base code* C_0. This added mask requires the verifier to run an additional local test (to verify the mask is close to the base code). Our ZK-IOP for 2-dimensional sumchecks is given in Figure 2, where we mark modifications over the IOP of [RR20] in red.

Sublinear-Communication Sumcheck ZK-IOPs for 2-dimensional Tensor Codes

Let $C_0 : \mathbb{F}^k \to \mathbb{F}^n$ be a q-LTC, and $C = C_0 \otimes C_0 : \mathbb{F}^{k \times k} \to \mathbb{F}^{n \times n}$. The IOP is executed between a prover $\mathcal{P}$ with input $c = C(\mathsf{m})$ for $\mathsf{m} \in \mathbb{F}^{k \times k}$, and a verifier $\mathcal{V}$ that has input $\alpha := \sum_{i,j \in [k]} \mathsf{m}(i,j)$ and oracle access to c. The goal is to verify that $c = C(\mathsf{m})$ for a message m satisfying that $\sum_{i,j \in [k]} \mathsf{m}(i,j) = \alpha$.

1. $\mathcal{P}$ samples $r \leftarrow \mathbb{F}^k$, and sends the oracle message $\hat{r} := C_0(r)$ and the explicit message $\beta := \sum_{i \in [k]} r(i)$ to $\mathcal{V}$.
   $\triangleright$ Let $\bar{r} \in \mathbb{F}^{k \times k}$ denote the matrix whose first column is r, and all other entries are 0, and let $R := C(\bar{r})$.
2. $\mathcal{V}$ runs the local tester of C_0 to obtain a set $I \subseteq [n], |I| = q$ of queries, and sends I to $\mathcal{P}$. $\mathcal{P}$ then sends $v := \hat{r}|_I$ to $\mathcal{V}$, and $\mathcal{V}$ checks that the local tester accepts v, and checks consistency of v with $\hat{r}$ at a random point. If one of the tests fails then $\mathcal{V}$ rejects.
3. $\mathcal{V}$ picks $\gamma \leftarrow \mathbb{F}$ and sends γ to $\mathcal{P}$.
   Both parties locally set $c' := \gamma \cdot c + R = C(\gamma \cdot \mathsf{m} + \bar{r})$.
4. $\mathcal{P}$ computes $z := \sum_{j \in [k]} C_0(\gamma \cdot \mathsf{m}(*, j) + \bar{r}(*, j))$. Then, $\mathcal{P}$ sends z to $\mathcal{V}$ as an explicit message.
5. $\mathcal{V}$ checks that $z = C_0(y)$ for some $y \in \mathbb{F}^k$ such that $\sum_{i \in [k]} y(i) = \gamma \cdot \alpha + \beta$, and rejects otherwise.
6. $\mathcal{V}$ samples a random row $i_0 \leftarrow [n]$ and sends i_0 to $\mathcal{P}$.
7. $\mathcal{P}$ sends to $\mathcal{V}$ the i_0th row $z' := c'(i_0, *)$.
8. $\mathcal{V}$ checks that $z' = C_0(y')$ for $y' \in \mathbb{F}^k$ such that $\sum_{j \in [k]} y'(j) = z(i_0)$.
9. $\mathcal{V}$ samples a random $j_0 \leftarrow [n]$, queries $c(i_0, j_0)$ and $\hat{r}(i_0)$, and accepts if and only if $z'(j_0) = \gamma \cdot c(i_0, j_0) + C_0(\hat{r}(i_0), 0, \ldots, 0)(j_0)$.

Figure 2: Sumcheck ZK-IOP for 2-Dimensional Tensor Codes (Special Case of Figure 3)

5.2 The Full Sumcheck Protocol

The IOP system $(\mathcal{P}, \mathcal{V})$ described in Theorem 5.4 is given in Figure 3 below. For simplicity, we describe a protocol in which the communication and query phase are interleaved, but it can be verified that all of the verifier's queries can also be deferred to the end of the protocol. We also omit the subscript k (when clear from the context) for better readability.

It can be verified that the protocol has the required structure, and that the communication complexity and verifier and prover running times are as claimed (noting that $R(i_1, \ldots, i_d) = (C^{\otimes(d-1)}(w(i_1), 0, \ldots, 0))(i_2, \ldots, i_d)$ can be computed in time $d \cdot \mathsf{polylog}(|\mathbb{F}_k|)$, given $w(i_1)$ and a generator matrix for C). Next we analyze the completeness, soundness, and zero-knowledge

properties of this protocol.

5.3 Completeness

Suppose that $\langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle = \alpha$. We shall show that in this case, with probability 1, $\mathcal{V}$ either accepts, or outputs $(i_1, \ldots, i_d) \in [k]^d$ and $\alpha'' \in \mathbb{F}$ so that $c(i_1, \ldots, i_d) = \alpha''$.

First note that by the properties of the local tester, we clearly have that $\psi_{\mathrm{TEST}}(u|_I) = \mathsf{ACCEPT}$, and so the verifier will not reject in Steps 1(b)iii and 1(b)iv.

Next observe that by linearity of C, $z = C(y)$ for $y = \sum_{j \in [k']} \lambda'(j)(\gamma \cdot \mathsf{m}(*, j) + \bar{r}(*, j))$, which satisfies that

$$\begin{aligned} \sum_{i \in [k]} \lambda_1(i) \cdot y(i) &= \sum_{i \in [k], j \in [k']} \lambda_1(i) \cdot \lambda'(j)(\gamma \cdot \mathsf{m}(*, j) + \bar{r}(*, j)) \\ &= \gamma \sum_{i \in [k], j \in [k']} \lambda_1(i) \cdot \lambda'(j) \cdot \mathsf{m}(i, j) + \lambda'(1) \sum_{i \in [k]} \lambda_1(i) \cdot r(i) = \gamma \cdot \alpha + \beta. \end{aligned} \tag{4}$$

Consequently, the verifier does not reject in Step 2b.

Furthermore, we have that $c' = C'(\mathsf{m}')$ for $\mathsf{m}' \in \mathbb{F}^{k'}$, given by $\mathsf{m}'(j) = C(\gamma \cdot \mathsf{m}(*, j) + \bar{r}(*, j))(i_1)$ for $j \in [k']$, which satisfies that

$$\langle \lambda_2 \otimes \cdots \otimes \lambda_d, \mathsf{m}' \rangle = \langle \lambda', \mathsf{m}' \rangle = \sum_{j \in [k']} \lambda'(j) \cdot C(\gamma \cdot \mathsf{m}(*, j) + \bar{r}(*, j))(i_1) = z(i_1). \tag{5}$$

Consequently, by Theorem 5.6, with probability 1, $\mathcal{V}'$ will output $(i_2, \ldots, i_d)$ and α' so that $\gamma \cdot c(i_1, \ldots, i_d) + R(i_1, \ldots, i_d) = \alpha'$ (and in particular, will not reject in Step 3a).

Finally, note that by the definition of R, we have that

$$\sigma = (C'(u(i_1), 0, \ldots, 0))(i_2, \ldots, i_d) = C'((C(r))(i_1), 0, \ldots, 0))(i_2, \ldots, i_d) = R(i_1, \ldots, i_d). \tag{6}$$

Consequently, we have that $\sigma = \alpha'$ if $\gamma = 0$, and $c(i_1, \ldots, i_d) = (\alpha' - \sigma)/\gamma = \alpha''$ if $\gamma \neq 0$, and so with probability 1, $\mathcal{V}$ will either accept or output $(i_1, \ldots, i_d)$ and α'' so that $c(i_1, \ldots, i_d) = \alpha''$ in Step 3c.

5.4 Soundness

Suppose that $\langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle = \alpha^*$ for some $\alpha^* \neq \alpha$. We shall show that in this case, for any prover strategy $\mathcal{P}^*$, with probability at least $\min\{\frac{\delta^2 \cdot \mu^2}{16}, \frac{\delta^d}{4}\}$,$\mathcal{V}$ either rejects, or outputs $(i_1, \ldots, i_d) \in [k]^d$ and $\alpha'' \in \mathbb{F}$ so that $c(i_1, \ldots, i_d) \neq \alpha''$.

Let $\beta^* \in \mathbb{F}$ and $u^* \in \mathbb{F}^k$ denote $\mathcal{P}^*$'s messages in Step 1a, and let v^* denote $\mathcal{P}^*$'s message in Step 1(b)ii. First note that we may assume that $\psi_{\mathrm{TEST}}(v^*) = \mathsf{ACCEPT}$, since otherwise $\mathcal{V}$ clearly rejects on Step 1(b)iii. Next observe that if u^* is $\frac{\delta}{2}$-far from C, then by the properties of the robust local tester, with probability at least $\frac{\delta \cdot \mu}{4}$, we have that $u^*|_I$ is $\frac{\delta \cdot \mu}{4}$-far from $\psi_{\mathrm{TEST}}^{-1}(\mathsf{ACCEPT})$. Assuming that this event holds, our assumption that $\psi_{\mathrm{TEST}}(v^*) = \mathsf{ACCEPT}$ implies that v^* is $\frac{\delta \cdot \mu}{4}$-far from $u^*|_I$. But in this case, $\mathcal{V}$ will reject in Step 1(b)iii with probability at least $\frac{\delta \cdot \mu}{4}$ (and overall $\mathcal{V}$ will reject with probability $\geq \frac{\delta^2 \mu^2}{16}$). Hence we may assume that u^* is $\frac{\delta}{2}$-close to a codeword $C(r)$.

Let β, $\bar{r}$, and R be defined as in Step 1a with respect to the string r. Next observe that since $\alpha^* \neq \alpha$, with probability at least $1 - \frac{1}{|\mathbb{F}|} \geq \frac{1}{2}$ over the choice of γ in Step 1c, we have that $\gamma \cdot (\alpha - \alpha^*) \neq \beta - \beta^*$. In what follows, assume that the latter event holds, which implies in turn that

$$\gamma \cdot \alpha + \beta^* \neq \gamma \cdot \alpha^* + \beta. \tag{7}$$

Let z^* denote $\mathcal{P}$'s messages in Step 2a. We may assume that $z^* = C(y^*)$ for $y^* \in \mathbb{F}^k$ which satisfies that $\sum_{i\in[k]} \lambda_1(i) \cdot y^*(i) = \gamma \cdot \alpha + \beta^*$, since otherwise $\mathcal{V}$ clearly rejects in Step 2b. On the other hand, by (4), for z which is defined as in Step 2a with respect to r, we have that $z = C(y)$ for $y \in \mathbb{F}^k$ which satisfies that $\sum_{i\in[k]} \lambda_1(i) \cdot y(i) = \gamma \cdot \alpha^* + \beta$. By (7), we conclude that z and z^* are two distinct codewords of C, and so they differ by at least a δ-fraction of the coordinates. By this, and by assumption that u^* is $\frac{\delta}{2}$-close to $C(r)$, with probability at least $\frac{\delta}{2}$ over the choice of i_1 in Step 2c, it holds that $z(i_1) \neq z^*(i_1)$ and $u(i_1) = (C(r))(i_1)$. Next assume that these events hold.

Next recall that by (5), we have that $c' = C'(\mathsf{m}')$ for $\mathsf{m}' \in \mathbb{F}^{k'}$ which satisfies that $\langle \lambda_2 \otimes \cdots \otimes \lambda_d, \mathsf{m}' \rangle = z(i_1)$. But by assumption that $z^*(i_1) \neq z(i_1)$, and by Theorem 5.6, this implies in turn that with probability at least δ^{d-1}, $\mathcal{V}'$ either rejects, or outputs $(i_2, \ldots, i_d)$ and α' so that $c'(i_2, \ldots, i_d) \neq \alpha'$. Assume that this event holds. If $\mathcal{V}'$ rejects, then $\mathcal{V}$ also rejects and aborts, and so we may assume that $\mathcal{V}'$ outputs $(i_2, \ldots, i_d)$ and α' so that $c'(i_2, \ldots, i_d) \neq \alpha'$, where $c'(i_2, \ldots, i_d) = \gamma \cdot c(i_1, \ldots, i_d) + R(i_1, \ldots, i_d)$.

Finally, by (6) and by assumption that $u(i_i) = (C(r))(i_1)$, we have that $\sigma = R(i_1, \ldots, i_d)$, which implies in turn that $\sigma \neq \alpha'$ if $\gamma = 0$, and $c(i_1, \ldots, i_d) \neq (\alpha' - \sigma)/\gamma$ if $\gamma \neq 0$. So in this case, $\mathcal{V}$ will either reject or output $(i_1, \ldots, i_d)$ and α'' so that $c(i_1, \ldots, i_d) \neq \alpha''$ on Step 3c.

We conclude that in the case that $\langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle \neq \alpha$, $\mathcal{V}$ rejects with probability at least $\min\{\frac{\delta^2 \cdot \mu^2}{16}, \frac{\delta^d}{4}\}$.

5.5 Zero knowledge

In this section we prove the two zero-knowledge properties stated in Theorem 5.4. Specifically, in Section 5.5.1 we provide a simulator for general codes C, whereas in Section 5.5.2 we describe a simulator for C which is itself a tensor product $C = B^{\otimes 3}$, where we exploit the tensor product to reduce the query complexity of the simulator.

5.5.1 Zero Knowledge for General Codes

Proof of ZK Item (1) in Theorem 5.4: Let t such that $t + q < k$. We describe a $((t + q + 1), \{2, \ldots, d\})$-restricted black-box straight-line simulator Sim that perfectly simulates the view $\mathsf{View}_{\mathcal{V}^*}(c)$ of any t-restricted (possibly malicious) verifier $\mathcal{V}^*$. This view consists of her randomness, her inputs $1^{[k]^d}, \lambda_1, \ldots, \lambda_d$ and $\alpha := \langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle$, the explicit message v, z received from the honest prover $\mathcal{P}$ in Steps 1(b)ii and 2a, the explicit messages received from $\mathcal{P}$ when executing the internal IOP in Step 3a, and the oracle answers to $\mathcal{V}^*$'s queries to c and u. (We note that while the *honest* verifier does not make any queries to c, and only makes a constant number of queries to u – to check consistency with v in Step 1(b)iv, to check z in Step 2b, and to perform Step 3b – the *malicious* $\mathcal{V}^*$ might make $\leq t$ queries to c, u.) In fact, the simulator which we design will have a stronger property: it will perfectly simulate $\beta = \lambda'(1) \sum_{i\in[k]} \lambda_1(i) r(i)$, as well as the *entire* codeword $\gamma \cdot c(i_1, *) + R(i_1, *) \in C^{\otimes(d-1)}$ from Step 3a. Since $\mathcal{V}^*$'s view in Step 3a can be efficiently generated from $\gamma \cdot c(i_1, *) + R(i_1, *)$ by honestly emulating the interaction between $\mathcal{V}^*$ and the honest prover $\mathcal{P}'$, this will show that $\mathcal{V}^*$'s view in Step 3a can be simulated, even though $(\mathcal{P}', \mathcal{V}')$ is *not* ZK.

We now describe the simulator. Notice that the oracle given to Sim is a row oracle for the tensor product $C \otimes C^{\otimes(d-1)}$, so in the following we refer to the oracle as a "row oracle". Sim, on input $1^{[k]^d}, \lambda_1, \ldots, \lambda_d$ and $\alpha := \langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle$, and given access to a row oracle for $c \in C \otimes C^{\otimes(d-1)}$, and black-box access to $\mathcal{V}^*$, operates as follows:

1. Initializes sets $Q_c = Q_r = \emptyset$ (intuitively, these are the sets of queries which $\mathcal{V}^*$ makes to her oracles c, u, respectively).

2. Answers any query $q = (i, j) \in [n] \times [n]^{d-1}$ of $\mathcal{V}^*$ to c by querying $c(i, *)$ from Sim's row oracle, sending $c(q)$ to $\mathcal{V}^*$, and adding q to Q_c.[23]

3. Samples $r \leftarrow \mathbb{F}^k$, generates β, u and R as the honest prover does (in Step 1a in Figure 3). If at some point during the simulation $\mathcal{V}^*$ queries β, then Sim sends β to $\mathcal{V}^*$.

4. Answers queries of $\mathcal{V}^*$ to c or u as follows. Queries to c are answered as in Step 2. Queries i to u are answered by adding i to Q_r, sending $u(i)$ to $\mathcal{V}^*$, and additionally querying $c(i, *)$.

5. At some point, Sim receives $I \subseteq [n]$ of size $|I| = q$ from $\mathcal{V}^*$. Then, for every $i \in I$, Sim adds i to Q_r and queries $c(i, *)$ from its row oracle. Then, Sim sends $v := u|_I$ to $\mathcal{V}^*$.

6. At some point, Sim receives $\gamma \in \mathbb{F}$ from $\mathcal{V}^*$.

7. Let $Q := Q_r \cup \left\{ i : \exists j \in [n]^{d-1}, (i, j) \in Q_c \right\}$. Sim Samples $y' \in \mathbb{F}^k$ uniformly at random subject to the following constraints:

   (a) for every $i \in Q$,

   $$C(y')(i) = \lambda'(1) \cdot u(i) + \gamma \cdot \left(\sum_{j \in [k']} \lambda'(j) \cdot C(\mathsf{m}(*, j)) \right)(i). \tag{8}$$

   Notice that for every $i \in Q$, Sim *knows $c(i, *)$. Therefore,* Sim *can decode $c(i, *)$, which is exactly $C(\mathsf{m}(i, *))$. Thus,* Sim *can compute the right-hand side of Equation (8).*

   (b) $\sum_{i \in [k]} \lambda_1(i) \cdot y'(i) = \gamma \cdot \alpha + \beta$.

   Let $z' := C(y')$. Sim sends z' to $\mathcal{V}^*$.

8. Answers queries of $\mathcal{V}^*$ to c or u as follows. Queries to c are answered as in Step 2. Queries i to u are answered by querying $c(i, *)$, and then sending $\left(z'(i) - \gamma \cdot \sum_{j \in [k']} \lambda'(j) \cdot C(\mathsf{m}(i, j)) \right) / \lambda'(1)$[24] to $\mathcal{V}^*$ as the oracle answer. ($C(\mathsf{m}(i, j))$ can be efficiently computed from $c(i, j)$ as explained in Step 7 above.)

9. At some point, Sim obtains from $\mathcal{V}^*$ an index $i_1 \in [n]$. Sim then queries its oracle to obtain $c(i_1, *)$, computes $u(i_1)$ as in Step 8, computes $R'(i_1, *) = C'(u(i_1), 0, \ldots, 0)$, and sets $c'' := \gamma \cdot c(i_1, *) + R'(i_1, *)$ to be the codeword on which the IOP of Step 3a in Figure 3 is executed. It then executes Step 3a in Figure 3 with $\mathcal{V}^*$ by honestly emulating the prover $\mathcal{P}'$ on input c''. Let M denote the prover messages sent during this phase. Throughout this execution, and after it ends, Sim continues to answer $\mathcal{V}^*$'s oracle queries as in Step 8 above.

It follows directly from the description of Sim that it is straight-line and uses $\mathcal{V}^*$ in a black-box manner. Therefore, to complete the proof, we need to prove that Sim is PPT, $(t + q + 1, \{2, \ldots, d\})$-restricted, and that $\mathcal{V}^*$'s view when interacting with Sim is distributed identically to its real-world view.

[23] We note that Sim could actually answer an entire *row query* of $\mathcal{V}^*$ to c. This fact will be used in Lemma 5.9 below.

[24] We note that this expression is well defined because we have assumed that $\lambda'(1) \neq 0$.

Sim **is** $(t+q+1)$**-restricted.** Sim queries its oracle in three cases: (1) when $\mathcal{V}^*$ queries c or u (there are at most t such queries, because $\mathcal{V}^*$ is t-restricted); (2) in Step 5, to generate v (Sim makes q queries); and (3) in Step 9, where Sim makes a single query. Overall, Sim makes at most $t+q+1$ queries.

Sim **is PPT.** All simulation steps are clearly $\mathsf{poly}(k)$-time, except for sampling y' in Step 7. We now explain why this too can be done in $\mathsf{poly}(k)$ time. Since the code is linear, each codeword symbol $C(y')(i)$ can be written as an $\mathbb{F}$-linear combination of the message symbols $y'(1),\ldots,y'(k)$. Therefore, the $|Q|+1$ equations in Step 1 can be replaced with $|Q|+1$ linear equations on the message symbols $y'(1),\ldots,y'(k)$. Moreover, since (1) r (and consequently u and β) were sampled as in the real world, and (2) the oracle answers to c are answered using the *actual* codeword symbols, such a solution y' exists, namely the set of linear equations has a solution (since it has a solution in the real world). Therefore, a random solution can be found efficiently by solving a system of linear equations, and picking a random element in the subspace solution.

Simulated and Real Views are Identically Distributed. Let $\mathsf{View}_R, \mathsf{View}_S$ denote the real and simulated views, respectively. It follows directly from the description of Sim that $q_S = q_V + q + 1$, where q_S, q_V are the *actual* number of queries which Sim, $\mathcal{V}^*$ make to their oracles (respectively). Therefore, it suffices to prove that $\mathsf{View}_S \equiv \mathsf{View}_R$. Moreover, since β and $\mathcal{V}^*$'s randomness rand are identically distributed in both worlds, we can condition both views on these values.

We claim first that the oracle queries which $\mathcal{V}^*$ makes before sending γ (i.e., before Step 1c in the real world, and before Step 6 in the simulation) are identically distributed. Indeed, each query is a function of the randomness rand of $\mathcal{V}^*$ (this is distributed identically in both worlds), and the oracle answers to previous queries. These queries are either to u or c, which are identically distributed in the real world and the simulation. Therefore, the oracle answers are identically distributed in both cases, and it follows by induction over the number $t' \leq t$ of queries which $\mathcal{V}^*$ makes to c, u (before Steps 1c in the IOP and 6 in the simulation) that the queries are identically distributed. Since γ is a function of rand and the oracle answers to queries made before Step 1c in the IOP and Step 6 in the simulation, γ is also identically distributed in $\mathsf{View}_R, \mathsf{View}_S$, and we can condition both views on it. Therefore, the constraints in Step 7 of the simulation are identically distributed to the constraints induced by the real-world c, u, γ.

It remains to prove that when conditioned on γ, β, $\mathcal{V}^*$'s randomness rand, and the oracle answer to queries in Q (defined in Step 7 of the simulation) then: (1) $z \equiv z'$, (2) the simulated messages M in Step 9 of the simulation are identically distributed to $\mathcal{P}''$s messages in Step 3a of Figure 3, and (3) the queries – and oracle answers to the queries – which $\mathcal{V}^*$ makes after receiving z (in the real world) or z' (in the simulation) are identically distributed.

Lemma 5.8. *It suffices to prove that* $z \equiv z'$.

Proof: Assume that $z \equiv z'$, and we show that in this case items (2) and (3) above are also identically distributed in the real world and the simulation.

We show first that the queries – and oracle answers to the queries – which $\mathcal{V}^*$ makes after receiving z (z', respectively), *but before* Step 3a in Figure 3 (Step 9 in the simulation, respectively) are identically distributed. Answers to queries made to c are clearly identical in both worlds (because Sim answers them using its own oracle). Since c is fixed and $\lambda'(1) \neq 0$, then (for a fixed γ) there is a bijection between $z = \lambda'(1)\cdot u + \gamma \cdot \sum_{j\in[k']} \lambda'(j)\cdot C(\mathsf{m}(*,j))$ in the real world (z' in the simulation, respectively) and u, and we have assumed that $z \equiv z'$, then answers to oracle queries to u are also identically distributed. $\mathcal{V}^*$'s queries are determined by her randomness rand, β, z (in the real world, or z' in the simulation) and oracle answers to her previous queries. Since we have conditioned on rand, β, we assume that $z \equiv z'$, and we have shown that the oracle answers are identically distributed, then (by induction on the number of queries which $\mathcal{V}^*$ makes) the oracle

queries are also identically distributed. Since all these determine i_1, it is also identically distributed in the real world and the simulation.

We are now ready to prove that the simulated messages M from Step 9 (i.e., item (2)) are distributed identically to the real world. These messages are fully determined by c'' (which was defined in Step 9 of the simulation.) c'' is fully determined by i_1, $u(i_1)$, γ, and $c(i_1, *)$. We have already shown above that i_1 is identically distributed in both worlds. $u(i_1)$ is determined by i_1, $c(i_1, *)$, γ, and z (in the real world,[25] or z' in the simulation). Since we have already shown above that all these are identically distributed, we conclude that the simulated M is distributed identically to the real world.

Next, we prove that the oracle queries and answers made during and after Step 3a in Figure 3 (or Step 9 in the simulation) are identically distributed in both worlds, which proves item (3). These oracle answers and queries are determined by rand, β, M (which are all identically distributed in both worlds), z (in the real world, or z' in the simulation), and the queried locations in c and u (in the real world) or z' (in the simulation, since it is used to answer queries to u). Since: (a) we have conditioned on rand, β, (b) we assume that $z \equiv z'$, (c) we have already shown that in this case M is identically distributed in the real and simulated views, and (d) for a fixed γ there is a bijection between z and u in the real world, these are all identically distributed (here again we use the fact that answers to queries to c are identical in both views because Sim answers them using its own oracle). This concludes the proof of Lemma 5.8. ∎

In summary, we have shown that to complete the proof, it suffices to prove that $z \equiv z'$. Recall that we have already shown that $u|_Q$ is distributed identically in $\mathsf{View}_R, \mathsf{View}_S$, and this holds even if $z \not\equiv z'$ (because $u|_Q$ was fixed before z, z' were determined). Therefore, it suffices to prove that $z \equiv z'$ conditioned on $u|_Q$. We do so by showing that z, z' are identically distributed to an alternative distribution z'', which we now define.

Defining the alternative distribution z''. Let $V := \left\{v \in C \ : \ v|_Q = \vec{0}\right\}$, and notice that V is a linear subspace. Let $t' := |Q|$, and $S := \left\{s \in \mathbb{F}^{t'} \ : \ \exists c'' \in C, c''|_Q = s\right\}$. For every $s \in S$ fix an arbitrary $c_s \in C$ which satisfies $c_s|_Q = s$, and let $C_s := \{c_s + v \ : \ v \in V\}$. Notice that $\{C_s \ : \ s \in S\}$ is a partition of C (here, we use the fact that C is a linear code).
z'' is sampled as follows:

1. Set $s \in S$ as follows: $s = \lambda'(1) \cdot u|_Q + \left(\gamma \cdot \sum_{j \in [k']} \lambda'(j) \cdot C\left(\mathsf{m}(*, j)\right)\right)|_Q$.
2. Sample $z'' \leftarrow C_s$ subject to the constraint that z'' encodes a message $y'' \in \mathbb{F}^k$ such that $\sum_{i \in [k]} \lambda_1(i) \cdot y''(i) = \gamma \cdot \alpha + \beta$.

We now show that z'' is identically distributed to both z and z'.

z', z'' are identically distributed. Both distributions are over codewords which satisfy the following constraints: (1) consistent with $\lambda'(1) \cdot u|_Q + \left(\gamma \cdot \sum_{j \in [k']} \lambda'(j) \cdot C\left(\mathsf{m}(*, j)\right)\right)|_Q$; and (2) encode a message $y^\star$ satisfying $\sum_{i \in [k]} \lambda_1(i) \cdot y^\star(i) = \gamma \cdot \alpha + \beta$. The only difference is that z' is sampled by sampling directly a message y' satisfying these two constraints, then encoding y' to obtain z', whereas z'' is sampled by first determining s, then sampling a random codeword $z'' \in C_s$ which satisfies constraint (2). Since m is fixed (because c is fixed), and we have conditioned on $u|_Q$ and γ, this fixes s. Therefore, sampling a random y' which satisfy both constraints induces the same distribution as sampling a codeword $z'' \in C_s$ which satisfies constraint (2) (since s is fixed), which is exactly the distribution of z'' (when s is fixed).

[25] This is because, as noted above, for a fixed γ there is a bijection between z and u.

z, z'' are identically distributed. Recall that m is fixed, and that we have fixed $u|_Q$ and γ, and this (as noted above) fixes s. Therefore, to prove that $z \equiv z''$ it suffices to prove that z is distributed uniformly at random over C_s, subject to the constraint that

$$(\star) \quad \exists y \ : \ z = C(y) \wedge \sum_{i \in [k]} \lambda_1(i) \cdot y(i) = \gamma \cdot \alpha + \beta$$

(recall that β is also fixed). We proceed to prove that this is indeed the case. Notice that if s is fixed then so is $s' := \left(s - \left(\gamma \cdot \sum_{j \in [k']} \lambda'(j) \cdot C\left(\mathsf{m}(*, j)\right)\right)|_Q\right) / \lambda'(1)$.[26] Since $u \in C_{s'}$ and s' is fixed, sampling a random u in C subject to the constraints $u|_Q$ and β is exactly the distribution of sampling a random u in $C_{s'}$ subject to the constraint that

$$(\star\star) \quad \exists r \ : \ u = C(r) \wedge \beta = \lambda'(1) \sum_{i \in [k]} \lambda_1(i) \cdot r(i).$$

Since $C_{s'} = \{c_{s'} + v \ : \ v \in V\}$ and C is linear, sampling $u \leftarrow C_{s'}$ subject to $(\star\star)$ is equivalent to setting $u := c_{s'} + v$, for v sampled from V uniformly at random subject to the constraint that

$$(\star\star\star) \quad \exists m_v \ : \ v = C(m_v) \wedge \beta = \lambda'(1) \sum_{i \in [k]} \lambda_1(i) \cdot (m_{s'}(i) + m_v(i))$$

where $m_{s'}$ is the message encoded in $c_{s'}$ (i.e., $c_{s'} = C(m_{s'})$). Therefore, $z = \gamma \cdot \sum_{j \in [k']} \lambda'(j) \cdot C\left(\mathsf{m}(*, j)\right) + \lambda'(1) \cdot (c_{s'} + v) = s'' + \lambda'(1) \cdot v$ for $s'' := \gamma \cdot \sum_{j \in [k']} \lambda'(j) \cdot C\left(\mathsf{m}(*, j)\right) + \lambda'(1) \cdot c_{s'}$ which is a fixed codeword in C_s. Since $v \leftarrow V$ subject to $(\star\star\star)$, and $\lambda'(1) \neq 0$, we conclude that z is also uniformly distributed over C_s subject to $(\star)$. ■

Zero-Knowledge against Row Queries to c. We note that the sumcheck ZK-IOP of Figure 3 provides a stronger ZK guarantee than stated in Theorem 5.4: it has ZK against verifiers $\mathcal{V}^*$ that make at most t queries to the prover's first message and *to a row oracle for* $c \in C_k \otimes C_k^{\otimes(d-1)}$. This fact will be useful when the sumcheck ZK-IOP is used to design a ZK-IOP for SAT in Section 6.

Lemma 5.9. *The IOP of Figure 3 has the following guarantee. For any natural t such that $t + q_k < k$ there exists a black-box straight-line PPT simulator Sim which is $((t + q_k + 1), \{2, \ldots, d\})$-restricted such that the following holds. For every $((\lambda_1, \ldots, \lambda_d, \alpha), c) \in \mathcal{R}^{\mathsf{YES}}_{\mathsf{SChk}}$, and for every verifier $\mathcal{V}^*$ that makes at most t queries to c, the prover's first message, and to a row oracle for $c \in C_k \otimes C_k^{\otimes(d-1)}$, we have*

$$\left(\mathsf{View}^{\mathsf{Sim}}_{\mathcal{V}^*}\left((\lambda_1, \ldots, \lambda_d, \alpha), c\right), q_S - q_k - 1\right)_{r_V, r_{\mathsf{Sim}}} \equiv \left(\mathsf{View}_{\mathcal{V}^*}\left((\lambda_1, \ldots, \lambda_d, \alpha), c\right), q_V\right)_{r_P, r_V} \tag{9}$$

where $r_P, r_V, r_{\mathsf{Sim}}$ are the random coins of the prover, verifier $\mathcal{V}^*$, and Sim, respectively; q_S denotes the number of queries which Sim makes to its row oracle; q_V denotes the number of queries which $\mathcal{V}^*$ makes to her oracles; and $\mathsf{View}^{\mathsf{Sim}}_{\mathcal{V}^*}$ is as defined in Definition 3.15.

Proof (Sketch). We repeat the proof of the ZK property in Theorem 5.4, altering the simulator Sim to answer row queries i of $\mathcal{V}$ in Steps 2, 4, and 8 with the full row $c(i, *)$ which Sim queried from its own oracle. We now need to show that the real and simulated views are identically distributed. In the proof of Theorem 5.4, we used the fact that the answers to $\mathcal{V}^*$'s queries to c are

[26] Notice that s' is well defined because we have assumed $\lambda'(1) \neq 0$.

identical in the real world and the simulation, because in the simulation they are answered using a (row) oracle to c. This is still true even when $\mathcal{V}^*$ makes row queries to $c \in C_k \otimes C_k^{\otimes(d-1)}$, because in the proof of Theorem 5.4 any (point-wise) query of $\mathcal{V}^*$ to c induced a full row-query of Sim to c. The rest of the proof holds unchanged. ■

5.5.2 Zero-Knowledge for Tensor Codes

In this section we prove that for codes C which are themselves a tensor product of another code B, our Sumcheck ZK-IOP of Figure 3 can have a smaller gap in the query complexities of the malicious verifier $\mathcal{V}^*$ and the ZK simulator. At a high level, by exploiting the tensor structure of C, we can employ a more structured local testing procedure whose "leakage" is contained in rows *of the code* B. Since such rows have cost 1 in our sumcheck protocol (i.e., the simulator can access a row oracle, so each row costs the simulator a single oracle query), we are able to reduce the query complexity of the simulator. We now turn to the formal proof.

Proof of ZK Item (2) in Theorem 5.4: Let t such that $(t+1) \cdot n^{2/3} < k$. We show how to modify the simulator Sim of Section 5.5.1 to obtain an adaptive $(t+2)$-subspace restricted (see Definition 4.11) straight-line black-box simulator Sim′ that perfectly simulates the view $\mathsf{View}_{\mathcal{V}^*}(c)$ of any t-restricted (possibly malicious) verifier $\mathcal{V}^*$. We note that the subspace oracle of Sim is defined *with respect to* $B^{\otimes 3d}$.

Sim′, on input $1^{[k]^d}$, $\lambda_1, \ldots, \lambda_d$ and $\alpha := \langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle$, operates similarly to Sim, with the following modifications:

- **Revised Step 2:** Answers any query $q \in [n]^d$ of $\mathcal{V}^*$ to c by querying $c(q)$ from Sim's oracle (recall that $C^{\otimes d} = B^{\otimes 3d}$, so a point in c is a point in a codeword of $B^{\otimes 3d}$, so Sim's oracle can be used to answer point queries to c), sending $c(q)$ to $\mathcal{V}^*$, and adding q to Q_c.
- **Revised Step 4:** queries to c are answered according to the revised Step 2 described above. Queries to u are answered by adding i to Q_r and sending $u(i)$ to $\mathcal{V}^*$. (Notice that on queries to u, Sim′ *does not* query its oracle for c; these queries will be made in the revised Step 5 below.)
- **Revised Step 5:** At some point, Sim receives from $\mathcal{V}^*$ the queries of the local tester. Because the local tester is the two-dimensional plane tester given by Theorem 3.33, this set of queries is an axis-parallel two-dimensional plane P, namely P is parallel to the ℓ'th axis for some $\ell \in \{1,2,3\}$. Sim sends $v' := u|_P$ to $\mathcal{V}^*$, and adds an arbitrary point on P to Q_r.

  Then, Sim sets the subspace of its oracle to $[3d] \setminus \{\ell\}$. Recall that at this point, for every query $q = (i_1, \ldots, i_{3d}) \in \left[n^{1/3}\right]^{3d}$ which $\mathcal{V}$ made to c in the revised Steps 2 or 4 of the simulation, Sim is given the restriction of c to the entire subspace $c_{\{\ell\} \leftarrow i_\ell}$. In addition, for every $i \in Q_r$, Sim queries the corresponding subspace $c_{\{\ell\} \leftarrow i}$.

  (For example, if $\ell = 1$ then for every $q = (i_1, \ldots, i_{3d}) \in Q_c$ Sim is given $(c(i_1, i'_2, \ldots, i'_{3d})_{(i'_2, \ldots, i'_{3d}) \in \left[n^{1/3}\right]^{3d-1}}$. Then, for every $i = (i_1, i_2, i_3) \in Q_r$, Sim queries its oracle on i_1 to obtain $(c(i_1, i'_2, \ldots, i'_{3d})_{(i'_2, \ldots, i'_{3d}) \in \left[n^{1/3}\right]^{3d-1}}$.[27])

[27] Notice, in particular, that Sim reads $c|_P$.

- **Answering queries to c or u after the revised Step 5 (but before the revised Step 7):** Sim answers queries $q = (i_1, \ldots, i_{3d}) \in \left[n^{1/3}\right]^{3d}$ to c by querying the subspace oracle on i_ℓ, sending $c(q)$ to $\mathcal{V}^*$, and adding q to Q_c. Queries $i = (i_1, i_2, i_3) \in \left(\left[n^{1/3}\right]\right)^3$ to u are answered by adding i to Q_r, sending $u(i)$ to $\mathcal{V}^*$, and additionally querying the subspace oracle on i_ℓ.
- **Revised Step 7:** Let

$$Q := \left\{ i_\ell \in \left[n^{1/3}\right] \ : \exists i \in \left[n^{1/3}\right]^3 \text{ s.t. } i_\ell \text{ it the } \ell' \text{th coordinate in } i, \text{ and } i \in Q_r \right\}$$
$$\cup \left\{ i_\ell \in \left[n^{1/3}\right] \ : \ \exists i_1, \ldots, i_{\ell-1}, i_{\ell+1}, \ldots, i_{3d} \in [n]^{1/3}, (i_1, \ldots, i_{3d}) \in Q_c \right\}$$

  and $\tilde{Q} := \left\{ (i_1, i_2, i_3) \in \left(\left[n^{1/3}\right]\right)^3 \ : \ i_\ell \in Q \right\}$.

  Sim Samples $y' \in \mathbb{F}^k$ uniformly at random subject to the following constraints:

  1. for every $i \in \tilde{Q}$,

$$C(y')(i) = \lambda'(1) \cdot u(i) + \gamma \cdot \left(\sum_{j \in [k']} \lambda'(j) \cdot C(\mathsf{m}((*, j)) \right)(i).$$

     (For every $i \in \tilde{Q}$, the simulator obtained from its oracle a subspace containing the ith row $c(i, *)$, so similar to the original Step 7, Sim can efficiently compute $C(\mathsf{m}(*, j))(i)$ and the right-hand side of the equation above.)
  2. $\sum_{i \in [k]} \lambda_1(i) \cdot y'(i) = \gamma \cdot \alpha + \beta$.

  Let $z' := C(y')$. Sim sends z' to $\mathcal{V}^*$.
- **Revised Steps 8 and 9:** the only difference in these steps is that Sim$'$ uses its subspace oracle (instead of a row oracle) to answer queries to u and to generate $u(i_1)$.

The proof that the simulation is correct follows similarly to the proof for general codes in Section 5.5.1, so we only sketch the differences.

Sim **is adaptive** $(t+2)$**-subspace restricted.** First, it follows directly from the definition of Sim that it is a 2-phase algorithm (the first phase ends at the onset of revised Step 5 of the simulation). As for the query bound, Sim queries its oracle in three cases: (1) when $\mathcal{V}^*$ queries c or u (there are at most t such queries, because $\mathcal{V}^*$ is t-restricted); (2) a single query in the revised Step 5 (indeed, Sim adds a point on P to Q_r, and then reads the corresponding subspace from c); and (3) a single query in the revised Step 9, to generate the i_1'th row of c. Overall, Sim makes at most $t+2$ queries.

Sim **is PPT.** We only need to explain why sampling y' in the revised Step 7 can be done efficiently (all other steps are clearly efficient). Each codeword symbol $C(y')(i)$ can be written as an $\mathbb{F}$-linear combination of the message symbols $y'(1), \ldots, y'(k)$ because the code is linear. Therefore, the $\left|\tilde{Q}\right| + 1 \leq |Q| \cdot n^{2/3} + 1$ equations in the revised Step 7 can be replaced with $|Q| \cdot n^{2/3} + 1$ linear equations on the message symbols $y'(1), \ldots, y'(k)$. Moreover, since (1) r (and consequently u and β) were sampled as in the real world, and (2) the oracle answers to c are answered using the *actual* codeword symbols, such a solution y' exists, namely the set of linear equations has a solution (since it has a solution in the real world). Therefore, a random solution can be found efficiently.

Simulated and Real Views are Identically Distributed. Let $\mathsf{View}_R, \mathsf{View}_S$ denote the real and simulated views, respectively. It follows directly from the description of Sim that $q_S = q_V + 2$, where q_S, q_V are the *actual* number of queries which Sim, $\mathcal{V}^*$ make to their oracles (respectively). Therefore, it suffices to prove that $\mathsf{View}_S \equiv \mathsf{View}_R$. Moreover, since β and $\mathcal{V}^*$'s randomness rand are identically distributed in both worlds, we can condition both views on these values.

$v = u|_P$ is identically distributed in both views. Similarly to the analysis of Section 5.5.1, the oracle queries which $\mathcal{V}^*$ makes before sending P (i.e., before Step 1(b)i in the real world, and before the revised Step 5.5.2 in the simulation), are identically distributed. Similarly, P is identically distributed in both views. Since $v = u|_P$, and u is identically distributed in both views, then v is also identically distributed in $\mathsf{View}_R, \mathsf{View}_S$. We can therefore condition both views on P, v.

γ is identically distributed in both views. Similarly to the analysis of Section 5.5.1, the oracle queries and answers which $\mathcal{V}^*$ makes before sending γ (i.e., before Step 1c in the real world, and before Step 6 in the simulation) are identically distributed. Since γ is a function of rand, v and the oracle answers to queries made before Step 1c in the IOP and Step 6 in the simulation, γ is also identically distributed in $\mathsf{View}_R, \mathsf{View}_S$, and we can condition both views on it. Therefore, the constraints in the revised Step 7 of the simulation are identically distributed to the constraints induced by the real-world c, u, γ. In particular, we have that $u|_{\widetilde{Q}}$ is identically distributed in the real world and the simulation. ($\tilde{Q}$ was defined in the revised Step 7 of the simulation.) We will use this observation below.

It remains to prove that when conditioned on $\gamma, \beta, v, u|_{\widetilde{Q}}$, $\mathcal{V}^*$'s randomness rand, and the oracle answer to queries in Q, then: (1) $z \equiv z'$, (2) the simulated messages M in the revised Step 9 of the simulation are identically distributed to $\mathcal{P}''$s messages in Step 3a of Figure 3, and (3) the queries – and oracle answers to the queries – which $\mathcal{V}^*$ makes after receiving z (in the real world) or z' (in the simulation) are identically distributed. We can show, similar to the analysis of Section 5.5.1, that to prove items (2) and (3), it suffices to prove that $z \equiv z'$.

We now show that $z \equiv z'$. Recall that $\widetilde{Q} = \left\{(i_1, i_2, i_3) \in \left(\left[n^{1/3}\right]\right)^3 \ : \ i_\ell \in Q\right\}$, and that we have already shown that $u|_{\widetilde{Q}}$ is distributed identically in $\mathsf{View}_R, \mathsf{View}_S$, and this holds even if $z \not\equiv z'$ (because $u|_{\widetilde{Q}}$ was fixed before z, z' were determined). Therefore, it suffices to prove that $z \equiv z'$ conditioned on $u|_{\widetilde{Q}}$. We do so by showing that z, z' are identically distributed to an alternative distribution z'', which is defined similarly to the alternative distribution of Section 5.5.1, except that we use $\tilde{Q}$ instead of Q. Formally:

The alternative distribution z''. Let $V := \left\{v \in C \ : \ v|_{\widetilde{Q}} = \vec{0}\right\}$. Let $t' := \left|\widetilde{Q}\right|$, and $S := \left\{s \in \mathbb{F}^{t'} \ : \ \exists c'' \in C, c''|_{\widetilde{Q}} = s\right\}$. For every $s \in S$ fix an arbitrary $c_s \in C$ which satisfies $c_s|_{\widetilde{Q}} = s$, and let $C_s := \{c_s + v \ : \ v \in V\}$.
z'' is sampled as follows:

1. Set $s \in S$ as follows: $s = \lambda'(1) \cdot u|_{\widetilde{Q}} + \left(\gamma \cdot \sum_{j \in [k']} \lambda'(j) \cdot C\left(\mathsf{m}(*, j)\right)\right)|_{\widetilde{Q}}$.
2. Sample $z'' \leftarrow C_s$ subject to the constraint that z'' encodes a message $y'' \in \mathbb{F}^k$ such that $\sum_{i \in [k]} \lambda_1(i) \cdot y''(i) = \gamma \cdot \alpha + \beta$.

The fact that z'' is identically distributed to both z and z' follows exactly as in Section 5.5.1, when Q is replaced with $\tilde{Q}$. ■

5.6 Distributional ZK

In this section, we show that if the code $\mathcal{C}$ used in the sumcheck IOP of Figure 3 has $(t+q+1)$-ZK (where q is the query complexity of the local tester for $\mathcal{C}$), then the resultant protocol has distributional t-ZK with respect to $\mathcal{C}$.

Corollary 5.10 (Sublinear-Length Distributional ZK-IOP for Sumcheck)**.** *Let $t \in \mathbb{N}$ be a ZK parameter, let $\alpha \in \mathbb{F}$, and let $\lambda_1, \ldots, \lambda_d \in \mathbb{F}^k$. If $\mathcal{C}$ has $(t+q+1)$-ZK with respect to the k'-randomized encoding function* Enc *for some $k' < k$, then the IOP of figure 3 has distributional t-ZK with respect to $\mathcal{C}$ and* Enc*. Moreover, the ZK property holds with a black-box, straight-line simulator.*

Proof: Assume that $\mathcal{C}$ has $(t+q+1)$-ZK with respect to the k'-randomized encoding function. By Theorem 4.2, $\mathcal{C}^{\otimes d} = \mathcal{C} \otimes \mathcal{C}^{\otimes d-1}$ has ZK against $(t+q+1)$-row-restricted adversaries with respect to the $[k']^d$-randomized encoding function Enc^d, and let Sim_C denote the corresponding PPT simulator.

We describe a simulator Sim', such that for every $\mathsf{m} \in \mathbb{F}$ and $\lambda_1, \ldots, \lambda_d \in \mathbb{F}^k$, Sim' perfectly simulates the view of any (possibly malicious and unbounded) t-restricted verifier $\mathcal{V}^*$ in the IOP of Figure 3, when $\mathcal{V}^*$'s oracle is $c \leftarrow \mathsf{Enc}^d(\mathsf{m})$.

Sim', on input $1^{[k]^d}$, $\lambda_1, \ldots, \lambda_d$ and $\alpha := \langle \lambda_1 \otimes \cdots \otimes \lambda_d, \mathsf{m} \rangle$, and given black-box access to $\mathcal{V}^*$, operates as follows. It emulates the simulator Sim whose existence is guaranteed by Theorem 5.4, where the input of Sim is $1^{[k]^d}$, $\lambda_1, \ldots, \lambda_d$ and α. During this emulation, Sim' relays messages between Sim and $\mathcal{V}^*$, and answers row queries of Sim to c using Sim_C.

Clearly, Sim' is PPT and straight-line (because Sim and Sim_C are). We now prove that the simulated and real world views are identically distributed. We do so by showing that both distributions are identical to a hybrid distribution $\mathcal{H}$ generated by the following "hybrid" simulator $\mathsf{Sim}_{\mathcal{H}}$. $\mathsf{Sim}_{\mathcal{H}}$ has input $1^{[k]^d}, \lambda_1, \ldots, \lambda_d$ and α, and is also given oracle access to a row oracle for $c \in \mathcal{C} \otimes \mathcal{C}^{\otimes(d-1)}$, where $c \leftarrow \mathsf{Enc}^d(\mathsf{m})$. $\mathsf{Sim}_{\mathcal{H}}$ emulates Sim as Sim' does, but answers Sim's row queries by querying its row oracle. $\mathcal{H}$ is the distribution induced over $\mathcal{V}^*$'s view by this process.

$\mathcal{H}$ is identically distributed to the real-world view. Since the oracles of $\mathcal{V}^*$ and $\mathsf{Sim}_{\mathcal{H}}$ are identically distributed, it suffices to prove the claim conditioned on c. In this case, $\mathcal{H}$ is distributed identically to the output of the $(t+q+1)$-row restricted simulator of Theorem 5.4. Therefore, Theorem 5.4 guarantees that $\mathcal{H}$ is distributed identically to the real world view of $\mathcal{V}^*$ (conditioned on c).

$\mathcal{H}$ is identically distributed to the simulated view. We prove this by a reduction to the ZK of $\mathcal{C}^{\otimes d}$ against $(t+q+1)$-row-restricted adversaries. Assume towards negation that $\mathcal{H}$ and $\mathsf{View}^{\mathsf{Sim}'}_{\mathcal{V}^*}(\lambda_1, \ldots, \lambda_d, \alpha, c)$ are not identically distributed, and let $\mathcal{D}$ be a distinguisher that can distinguish between then with some advantage $\epsilon > 0$.

We define a distinguisher $\mathcal{D}'$ that obtains the same advantage ϵ in distinguishing between $(t+q+1)$ real and simulated (by Sim_C) rows of c, where $c \leftarrow \mathsf{Enc}^d(\mathsf{m})$. This contradicts the ZK of $\mathcal{C}^{\otimes d}$ against $(t+q+1)$-row-restricted adversaries, which in turn contradicts the $(t+q+1)$-ZK of $\mathcal{C}$.

$\mathcal{D}'$ has oracle access to an oracle that answers (up to) $(t+q+1)$ adaptive row queries, either according to the actual rows of c, or according to the simulated answers of Sim_C. It samples a random string r_V for $\mathcal{V}^*$, and then emulates Sim by relaying messages between Sim and $\mathcal{V}^*(1^{[k]^d}, \lambda_1, \ldots, \lambda_d, \alpha; r_V)$, and forwards Sim's oracle queries to $\mathcal{D}'$'s own oracle. When Sim terminates, $\mathcal{D}'$ runs $\mathcal{D}$ on $\mathcal{V}^*$'s simulated view obtained in this process. Notice that when $\mathcal{D}'$'s oracle is to a row-oracle for $c \in \mathcal{C} \otimes \mathcal{C}^{\otimes(d-1)}$ then $\mathcal{D}'$ runs $\mathcal{D}$ on a sample from $\mathcal{H}$, otherwise it runs $\mathcal{D}$

on a sample from the simulated distribution (generated by Sim'). Therefore, $\mathcal{D}'$ obtains the same distinguishing advantage ϵ as $\mathcal{D}$. ■

6 ZK-IOP approaching witness length for 3SAT

In this section, we use the zero-knowledge properties of tensor codes shown in Section 4, and the sumcheck ZK-IOP of Section 5, to construct a ZK-IOP for 3-SAT whose proof length approaches the witness length.

Theorem 6.1 (ZK-IOP for 3-SAT)**.** *For any constant $\beta, \gamma > 0$, there exists a constant-round pre-processing IOP $(\mathcal{P}, \mathcal{V})$ for 3-SAT on n variables of constant soundness error, which satisfies the following:*

- *The prover's first message is a binary string of length $(1+\gamma) \cdot n$, and the rest of the prover's messages are strings of length n^β over some finite field $\mathbb{F}$ of size 2^s for $s = O(\log n)$.*
- *The verifier queries a constant number of bits from the first prover's message, queries a constant number of field elements from the rest of the prover's oracles, and reads the rest of the prover's messages in full.*
- *There exists a constant $\beta' > 0$, so that the IOP has ZK against any malicious verifier that reads at most $n^{\beta'}$ entries from each prover's oracle, and reads the rest of the prover's messages in full.*
- *The verifier's running time is $n^{O(\beta)}$, after a local preprocessing step of running time $\mathsf{poly}(n)$. The prover's running time is $\mathsf{poly}(n)$, given a witness, and a generating matrix for the ZK-LTC C_2 generated in Step 1c of Figure 5.*

Remark 6.2. *The ZK-IOP described in Theorem 6.1 above is with a non-uniform prover, where the (only) reason for non-uniformity is that the prover is given a "good" generator matrix for C_2 (i.e., a generator matrix with respect to which C_2 is ZK). The ZK-IOP of Theorem 6.1 can be made to be uniform in one of the following ways:*

- *Allowing the prover to run in exponential time. This allows the prover to find a "good" generator matrix for C_2. Notice that letting the prover choose the generator matrix does not affect soundness, since the verifier can efficiently check that the matrix provided by the prover generates the code C_2.*
- *Alternatively, we could settle for statistical ZK instead of perfect ZK, and have the prover sample a generator matrix for C_2 (this generator matrix can be sampled efficiently by Corollary 4.19). This does not affect completeness or soundness, but results in statistical ZK since there is a negligible probability that the prover will sample a "bad" matrix. However, conditioned on this event not happening, ZK will be perfect.*

6.1 High-Level Overview of the Protocol

Our ZK-IOP satisfying the properties stated in Theorem 6.1 is given in Figures 5, 6, and 7 below. Before we proceed to the full description of the protocol, we first give a high-level overview of its structure. We first describe basic ideas that appear in many prior works on probabilistic proof systems, specifically encoding the assignment as a polynomial, and replacing the original claim with a sumcheck claim. We then explain the components and techniques which are new to this work, and enable achieving ZK with proofs approaching the witness length.

6.1.1 Prior Techniques in (Zero-Knowledge) PCP and IOP Design

Arithmetization: Representing 3-SAT Formulas as Polynomials. Let φ be a 3-CNF formula with n variables, and let $w \in \{0,1\}^n$ be a satisfying assignment for φ. We follow the standard paradigm of representing φ and w as multi-variate low-degree polynomials [BFL91, BFLS91, Sud00].

Specifically, let $k, d \in \mathbb{N}$ be such that $k^d = \lceil n \rceil$, where d is a sufficiently large constant, and where we identify $[n]$ with $[k]^d$. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| = \Theta(k) > k$, where we identify $[k]$ with some fixed subset of $\mathbb{F}$. We interpret w as a function $w : [k]^d \to \mathbb{F}$. Standard PCP/IOP constructions use the LDE encoding $\hat{w}$ of w (see lemma 3.35 in Section 3.2.4), which is the *unique* d-variate polynomial of individual degree at most k over $\mathbb{F}$ that represents w. Let $I_\varphi : [k]^{3d+3} \to \{0,1\}$ be a function that represents the structure of φ, namely for every $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3 \in [k]^d$ and every $b_1, b_2, b_3 \in [k]$, $I_\varphi(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) = 1$ if and only if the clause $(x_{\mathbf{i}_1} = b_1) \vee (x_{\mathbf{i}_2} = b_2) \vee (x_{\mathbf{i}_3} = b_3)$ appears in φ. Let $\hat{I}_\varphi$ be the LDE of I_φ, which is the unique $(3d+3)$-variate polynomial of individual degree at most k over $\mathbb{F}$ that represents I_φ.

Finally, we let $P_{\varphi,w}$ be the $(3d+3)$-variate polynomial over $\mathbb{F}$ given by:

$$P_{\varphi,w}(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) = \hat{I}_\varphi(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) \cdot \Pi_{j=1}^{3} (\hat{w}(\mathbf{i}_j) - b_j).$$

Importantly, w satisfies φ if and only if $P_{\varphi,w}$ vanishes on $[k]^{3d+3}$. This representation is useful because checking that $P_{\varphi,w}$ vanishes on $[k]^{3d+3}$ reduces to checking that a related $(3d+3)$-variate polynomial P of individual degree at most $3k$ sums to 0 on $[k]^{3d+3}$ (see Lemma 3.39).

The process of representing the satisfying assignment w as a polynomial P – known as *arithmetization* – is precisely the reason that the LDE encoding is so useful in designing proof systems: it enables to *uniquely* represent any string, such that verifying whether w satisfies φ reduces to checking the sum of a polynomial on a subcube. Similar to other PCPs/IOPs, the high-level idea is then to use a sumcheck procedure to verify that P indeed sums to 0 on $[k]^{3d+3}$. The sumcheck procedure exploits a different viewpoint of the polynomial P, as we now explain.

Polynomial View vs. Tensor View. The arithmetization described above defines the encoding P of w in terms of polynomials. However, to verify that P indeed sums to 0 on $[k]^{3d+3}$, it is more useful to adopt a coding/tensor view of P. Specifically, notice that the truth table of P on $\mathbb{F}^{3d+3}$ can be viewed as a codeword $c_1 \in \mathsf{RS}^{\otimes(3d+3)}$. This viewpoint is useful because tensor codes are particularly amenable to local testing and sumchecking (as well as to obtaining ZK). Thus, once we arithmetize w by "encoding" it into P, we switch from the polynomial view to the tensor view.

Viewing the truth table of P as a Reed-Solomon (RS) codeword c_1, we can use the sumcheck ZK-IOP of Theorem 5.4 (Section 5) to verify that P sums to 0 on $[k]^{3d+3}$. Recall that the verifier in the sumcheck ZK-IOP of Theorem 5.4 might output a point $\mathbf{i}$ on c_1 and a value α, such that it should hold that $c_1(\mathbf{i}) = \alpha$. Therefore, to complete the verification, the verifier $\mathcal{V}$ needs oracle access to c_1. In fact, it suffices for $\mathcal{V}$ to have oracle access to $\hat{w}$, since $\mathcal{V}$ can locally compute any point of c_1 by querying 3 points of $\hat{w}$.

However, the length overhead incurred by the RS encoding is too large, because it is defined over a large super-constant alphabet size, and because the code needs to satisfy the multiplication property which inherently requires rate at most $\frac{1}{2}$. Since we are interested in designing a ZK-IOP which *approaches the witness length*, we cannot afford to send the oracle $\hat{w}$ to the verifier.

Using Both Low-Rate and High-Rate Encodings (a-la [RR20]). This problem was solved by Ron-Zewi and Rothblum [RR20] (inspired by [Mei13]) using the so-called *code-switching technique*. Roughly, in Ron-Zewi and Rothblum's IOPs the prover $\mathcal{P}$ sends to $\mathcal{V}$ an encoding c_2 of

w, generated using a *high-rate binary* tensor code $(C_2)^{\otimes d}$. Since $(C_2)^{\otimes d}$ has high rate, we can afford to send w_2 to the verifier. Crucially, one can verify the value of any point on $\hat{w}$ by executing a sumcheck over c_2. The fact that $(C_2)^{\otimes d}$ is a tensor code guarantees that it is both locally testable and sumcheckable. In summary, the high-level idea of [RR20] is for $\mathcal{P}, \mathcal{V}$ to execute (a scaled version of) the sumcheck over the low-rate encoding c_1, which results in a point $\mathbf{i}$ and a value α. Let $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3$ be such that $c_1(\mathbf{i})$ can be generated from $\hat{w}(\mathbf{i}_1), \hat{w}(\mathbf{i}_2), \hat{w}(\mathbf{i}_3)$. Then $\mathcal{P}$ sends $\alpha_j := \hat{w}(\mathbf{i}_j), 1 \leq j \leq 3$ to $\mathcal{V}$, and the parties then engage in a sumcheck over c_2 to check the purported values of $\hat{w}(\mathbf{i}_j), 1 \leq j \leq 3$.

Obtaining Zero-Knowledge: Randomized (and Punctured) LDE. Another issue which arises when designing ZK-IOPs, which is orthogonal to the efficiency issue discussed in the previous paragraph (i.e., that we cannot afford to send the LDE to $\mathcal{V}$), is the fact that the LDE encoding $\hat{w}$ itself reveals non-trivial information on w. This issue was addressed already in [BCGV16], who replace the unique LDE encoding $\hat{w}$ with a *randomized* version of this encoding, generated by applying the standard LDE encoding to a *randomly padded* version of w. More specifically, for $k_1 > k$ (whose value is set in Step 1a of Figure 5), let $w_1 : [k_1]^d \to \mathbb{F}$ be such that $w_1(\mathbf{i}) = w(\mathbf{i})$ for every $\mathbf{i} \in [k]^d$, and the rest of the entries in $w_1(\mathbf{i})$ are sampled uniformly at random and independently. Let $\hat{w}_1$ be the LDE encoding of w_1 (the unique d-variate polynomial of individual degree at most k_1 over $\mathbb{F}$ representing w_1), and let c_1 be the truth table on $\mathbb{F}^{3d+3}$ of

$$P_{\varphi,w}\left(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3\right) = \hat{I}_{\varphi}\left(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3\right) \cdot \Pi_{j=1}^{3}\left(\hat{w}_1\left(\mathbf{i}_j\right) - b_j\right).$$

We additionally take the base code C_2 of the tensor code $(C_2)^{\otimes d}$ to be a high-rate binary *ZK code*, and let c_2 be the encoding of a randomized version $w_2 : [k_2]^d \to \{0,1\}$ of w_1 via the code $(C_2)^{\otimes d}$, to guarantee that c_2 does not leak information about w.

It is important to notice that using a randomized LDE encoding and a ZK code C_2 still does not prevent information leakage, even to the *honest* verifier. More specifically, the verifier can learn non-trivial information about the witness w via (1) direct queries to the LDE encoding $\hat{w}_1$, or (2) from information leaked in the sumcheck protocols for c_1 and c_2. We explain (1) now, and discuss (2) in Section 6.1.2 below.

To see why $\mathcal{V}$ can learn information from direct queries to $\hat{w}_1$, recall that while $\hat{w}_1$ is not given directly to the verifier, the prover does send $\hat{w}_1(\mathbf{i}_j), 1 \leq j \leq 3$ to her after the sumcheck protocol on c_1. If, for example, the sumcheck verifier $\mathcal{V}_{\mathsf{in}}$ outputs a point $(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3)$ on c_1 such that, e.g., $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3 \in [k]^d$, then $\mathcal{V}$ learns a non-trivial function of $w(\mathbf{i}_1), w(\mathbf{i}_2), w(\mathbf{i}_3)$. Similar to prior works, we prevent this information leakage by using a *punctured* version of the RS code in which codewords contain only evaluations of the (polynomial defined by the) message in $\mathbb{F} \setminus [k]$ (cf. Definition 3.36). This guarantees that by accessing the codeword one cannot reconstruct values of P in $[k]^{3d+3}$.

6.1.2 New Techniques for ZK Proofs Approaching the Witness Length

As discussed in the previous section, using a punctured and randomized version of the LDE encoding, as well as a ZK code C_2, guarantees that direct queries into the LDE or c_2 do not leak information on w. However, the parties also execute sumcheck IOPs over c_1 and c_2, and so to guarantee ZK, we must ensure that no information is leaked in this execution. As explained in Section 2.2, prior works used a sumcheck ZK-IOP with a strong ZK guarantee that was achieved by fully masking the checked codeword. Combining such a sumcheck ZK-IOP with a ZK code seamlessly ensured ZK. However, since we aim for proofs approaching the witness length, we cannot afford to fully mask the checked codeword, and consequently our sumcheck ZK-IOP of

Section 5 achieves a weaker ZK guarantee, i.e., there is an additive loss in the ZK parameter due to local testing.

Specifically, to handle ZK against t-restricted verifiers, the base code of the tensor product should have ZK against $(t+q)$-*restricted verifiers*, where q is the query complexity of the local tester for the code. Thus, obtaining a non-trivial ZK guarantee necessitates a gap between the query complexity of the local tester, and the ZK parameter of the underlying code. Put differently, the underlying base code should be a ZK-LTC. This requirement of the code used in the sumcheck ZK-IOP was not needed in prior works due to the use of a sumcehck IOP with stronger ZK guarantees (which, however, inherently did *not* have sublinear communication).

To deal with the above, we choose the base code C_2 to be the ZK-LTC given in [ISVW13] (which can be constructed probabilistically). Unfortunately, the ZK-LTC requirement turns out to be problematic for the low-rate code. Indeed, our IOP construction (as well as many works in the field of PCPs and IOPs) crucially relies on the properties of the LDE encoding, namely its polynomial view discussed above. This view is inherently tied to the *specific encoding procedure* used to map a message w into a codeword c_1. Consequently, we cannot apply the transformation of [ISVW13] (which adds a ZK guarantee to any linear code) since it modifies the encoding procedure (while keeping the set of codewords in tact). Thus, the only hope is that the PRS code (with the specific encoding procedure discussed above) is a ZK-LTC. Unfortunately, this is not the case, as we show in Lemma 4.15 (Section 4.2.1). Our solution is to take a different route altogether, designing a specialized sumcheck ZK-IOP with a smaller loss in the ZK parameter.

Sublinear-Communication Zero-Knowledge via Specialized Sumcheck. Recall that to employ arithmetization, we are forced to use the PRS code with a specific encoding procedure, which is not a ZK-LTC, and consequently using the sublinear-communication sumcheck ZK-IOP of Section 5 (for general codes) might violate ZK of the full ZK-IOP for 3-SAT. Instead, our main observation in this context is that we can obtain ZK by *exploiting the highly-structured nature* of the PRS encoding, and combining it with a *specialized* version of our sumcheck ZK-IOP. More specifically, we take the base code C_1 for the sumcheck ZK-IOP to be *a tensor code* $C_1 = \mathsf{PRS}^{\otimes 9}$ (instead of taking the base code to be just PRS). Such tensor codes have local testers whose queries are highly structured (see Theorem 3.33), and therefore admit a ZK sumcheck procedure with a smaller (constant) additive loss in the ZK parameter (see Theorem 5.4, version for tensor base codes). Intuitively, the choice of 9 in the tensor power is made because executing the version of Theorem 5.4 for tensor base codes requires that $C_1 = B^{\otimes 3}$ for some code B, and $B = \mathsf{PRS}^{\otimes 3}$ is required in order to correctly map $\hat{w}_1$ into c_1, as discussed below.

Bookkeeping Dimensions: "Correctly" Mapping $\hat{w}_1$ into c_1. Employing the specialized sumcheck ZK-IOP for tensor codes introduces a new problem. Indeed, the smaller loss in the ZK parameter in the specialized sumcheck ZK-IOP comes at the price that the sumcheck verifier $\mathcal{V}_{\mathsf{in}}$ might now learn *hyperplanes* of c_1, when viewed as a codeword of $B^{\otimes(d+1)} = \mathsf{PRS}^{\otimes(3d+3)}$. Thus, to get ZK we must guarantee that *full hyperplanes* of c_1 reveal no information on w. This should be contrasted with previous works, where fully masking the sumchecked codeword guaranteed that $\mathcal{V}_{\mathsf{in}}$ learns only *few points* on c_1, which translate to few points on $\hat{w}_1$, regardless of how points on $\hat{w}_1$ are mapped to points on c_1.

Luckily, because PRS has uniform ZK, we show in Corollary 4.12 (Section 4.2) that tensors of PRS (in particular, $\hat{w}_1$) have ZK against adversaries reading hyperplanes. Therefore, it suffices to ensure that a hyperplane of c_1 translates into few hyperplanes of $\hat{w}_1$. For this, we carefully map the dimensions of the three values of $\hat{w}_1$ – used to define each value of c_1 – to the dimensions of

c_1.

Code Switching with ZK Codes. We complete the overview by discussing one technical issue which arises when applying the code switching technique over ZK codes. The code switching technique of [RR20] relied on the fact that any point in the LDE $\hat{w}$ is a linear combination $\sum_i \lambda(i)w(i)$ of the entries of the witness $w \in \{0,1\}^{[k]^d}$, where the coefficients $\lambda(i)$ have a certain *tensor structure*. As mentioned above, the value of the sum $\sum_i \lambda(i)w(i)$ can then be verified by executing (a scaled version of) the sumcheck protocol on the encoding $C(w)$ of the witness w via another unrelated tensor code C.

However, in our ZK-IOP, $\hat{w}_1$ is obtained by encoding a *randomized version* $w_1 \in \mathbb{F}^{[k_1]^d}$ of w via the LDE. But now to apply the code switching technique, the prover needs to provide the encoding $(C_2)^{\otimes d}(w_1)$ of w_1, whose entries come from a *large field*, via the *binary* code $(C_2)^{\otimes d}$. To handle this, we choose the large field $\mathbb{F}$ to be an extension field of the binary field, and find a novel way to map the message w_1 into a *binary* message w_2, whose length is not much longer than that of w_1, and to map the tensor coefficients $\lambda(i)$ into other tensor coefficients $\lambda'(i)$ satisfying the property that $\sum_i \lambda(i)w_1(i) = \sum_i \lambda'(i)w_2(i)$. Consequently, the prover can provide the encoding of the *binary* message w_2 via the *binary* code $(C_2)^{\otimes d}$, and verifying the linear combination $\sum_i \lambda(i)w_1(i)$ reduces to executing the sumcheck protocol on $(C_2)^{\otimes d}(w_2)$ with tensor coefficients $\lambda'(i)$. We further incorporate in w_2 also fresh randomness not used in the generation of w_1 to guarantee that the encoding of w_2 via $(C_2)^{\otimes d}$ would be ZK. (see Lemma 6.5 and Claim 6.3 for the definition and properties of w_2 and the coefficients $\lambda'(i)$.)

6.2 The "Bare-Bones" Protocol and the Full Protocol

In Figure 4, we give a "bare-bones" description of our ZK-IOP for 3-SAT. This bare-bones protocol captures the main steps in the ZK-IOP, while omitting many details (e.g., on how exactly the high- and low-rate encodings are computed). The detailed (and accurate) description is given in Figures 5-7 below. Since the full protocol includes local computations and various settings of parameters, we emphasize interactive steps (direct messages exchanged between the parties, or queries to prover oracle messages) in purple.

Main Differences between the Bare-Bones Protocol and the Actual ZK-IOP. The "bare bones" protocol of Figure 4 omits several details. We now explain the missing details and where they appear in the full protocol (Figures 5-7).

- **Setting of parameters, and explicit description of encoding.** The bare-bones protocol does not explain how $k_1, k_2, n_1, n_2, \mathbb{F}$ are chosen. Also, it does not describe exactly how $\hat{w}_1, c_2$ are generated. Intuitively, $\hat{w}_1$ is the low-degree extension of w padded with randomness r, and c_2 is a random C_2-encoding of this padded message (w, r). However, to preserve the relationship between $\hat{w}_1, c_2$ needed by the protocol (namely, that one can compute entries of $\hat{w}_1$ using a scaled sum over entries of c_2), the encoding must be defined more carefully (see steps 3a and 4a in figure 6). We also note that for ease of notation, in the final ZK-IOP we describe the encoding in a reversed order: first generating the randomized c_2, then generating $\hat{w}_1$. However, as we show in Lemma 6.5 below, this is equivalent to the description provided in the bare-bones protocol.

- **Local testing of c_2.** Soundness of the ZK-IOP uses the fact that the purported codeword c_2 sent by $\mathcal{P}$ in Step 3a is indeed close to the code C_2. In the bare-bones protocol, this is

guaranteed by having $\mathcal{V}$ directly performs a local test on v, by running the local tester TEST and querying c_2 at the appropriate locations (see Step 3 in Figure 4). In the full protocol, we instead have the prover provide the queried entries of c_2, and $\mathcal{V}$ checks consistency with c_2 at a single point (See Steps 3b-3e).

- **Sumcheck and code switching over randomized LDE.** The bare-bones protocol does not specify how to set the tensor coefficients such that executing the sumcheck (code switching, respectively) over a randomized LDE encoding $\hat{w}_1$ (randomized encoding c_2, respectively) eliminates unnecessary encoding randomness. This is specified in the full protocol in Steps 5a and 6a (Figure 7).

The Full Protocol. The formal description of the protocol is given in Figure 5. In the protocol, we let $d := \lceil d_0/\beta \rceil$, where $d_0 \geq 1$ is a sufficiently large absolute constant, and $d+1$ is divisible by 3. For simplicity we assume that $n := k^d$ for some integer k. This can be obtained by letting $k := \lceil n^{1/d} \rceil$, and replacing n with $\bar{n} := k^d = (1+o(1)) \cdot n$.

We note that while for simplicity, we chose to describe the protocol with interleaved communication and query phase, all verifier queries can be deferred to the end of the protocol. It can be verified that the protocol has the required properties, for a sufficiently large absolute constant $d_0 \geq 1$. Next we prove completeness, soundness, and zero-knowledge properties of this protocol.

6.3 Completeness

Suppose that w is a satisfying assignment for φ, we shall show that in this case $\mathcal{V}$ accepts with probability 1.

First note that by the properties of the local tester, we clearly have that $\psi_{\mathrm{TEST}}(c_2|_I) = \mathsf{ACCEPT}$, and so the verifier will not reject in Steps 3d and 3e.

We now claim that

$$\sum_{\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3 \in [k]^d, b_1,b_2,b_3 \in [k]} P\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) = 0. \tag{10}$$

To see the above, note that by the definition of P_{φ,w_1} in Step 4a, and by our setting of $w_1(\mathbf{i}) = w_2(\mathbf{i}) = w(\mathbf{i})$ for any $\mathbf{i} \in [k]^d$ in Steps 3a and 4a, we have that

$$P_{\varphi,w_1}\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) = I_{\varphi}\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) \cdot \Pi_{j=1}^{3}\left(w(\mathbf{i}_j) - b_j\right)$$

for any $\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3 \in [k]^d$ and $b_1,b_2,b_3 \in [k]$. Furthermore, by the definition of I_{φ} in Step 2a, and by assumption that w is a satisfying assignment for φ, the above expression is zero for any $\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3 \in [k]^d$ and $b_1,b_2,b_3 \in [k]$. But by the definition of $P = Q \cdot P_{\varphi,w}$ in Step 4b, this implies in turn that $P\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) = 0$ for any $\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3 \in [k]^d$ and $b_1,b_2,b_3 \in [k]$, and so (10) is satisfied.

Next observe that by (10), and by the definition of w_3 and $\lambda_1,\ldots,\lambda_{(d+1)/3}$ in Steps 4b and 5a, respectively, we have that

$$\begin{aligned}
\langle \lambda_1 \otimes \cdots \otimes \lambda_{(d+1)/3}, w_3 \rangle &= \sum_{\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3 \in [k]^d, b_1,b_2,b_3 \in [k]} w_3\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) \\
&= \sum_{\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3 \in [k]^d, b_1,b_2,b_3 \in [k]} P\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) = 0.
\end{aligned}$$

Consequently, by Theorem 5.4, with probability 1, the outcome of the protocol $(\mathcal{P}_{\mathsf{in}}, \mathcal{V}_{\mathsf{in}})$ in Step 5b is either that $\mathcal{V}_{\mathsf{in}}$ accepts, or a point $(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) \in (\mathbb{F} \setminus [3k_1])^{3d+3}$ and a value $\alpha' \in \mathbb{F}$ so that

$$P(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) = \alpha'. \tag{11}$$

In the former case, $\mathcal{V}$ also accepts, and so we are done. Hence we may assume that (11) holds.

Claim 6.3 below shows that

$$\langle \lambda'_{j,1} \otimes \cdots \otimes \lambda'_{j,d}, w_2 \rangle = \hat{w}_1(\mathbf{i}_j)$$

for any $j \in \{1,2,3\}$. Consequently, by Theorem 5.4, for any $j \in \{1,2,3\}$, with probability 1, the outcome of the protocol $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$ on Step 6c is either that $\mathcal{V}'_{\mathsf{in}}$ accepts, or a point $\mathbf{i}'_j \in [n_2]^d$ and a value $\alpha'_j \in \mathbb{F}$ so that $c_2(\mathbf{i}'_j) = \alpha'_j$. So $\mathcal{V}$ does not reject in Step 6c in any of the iterations.

Finally, by the definition of P in Step 4b, and by our assumption (11), this implies in turn that

$$\begin{aligned}
&Q(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) \cdot \hat{I}_\varphi(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) \cdot \Pi_{j=1}^3 (\alpha_j - b_j) \\
&= Q(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) \cdot \hat{I}_\varphi(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) \cdot \Pi_{j=1}^3 (\hat{w}_1(\mathbf{i}_j) - b_j) \\
&= P(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) = \alpha',
\end{aligned}$$

(the top equality holds because the honest prover $\mathcal{P}$ sends $\alpha_j = \hat{w}_1(\mathbf{i}_j)$) and so $\mathcal{V}$ will accept on Step 7.

Claim 6.3. *For any $j \in \{1,2,3\}$,*

$$\langle \lambda'_{j,1} \otimes \cdots \otimes \lambda'_{j,d}, w_2 \rangle = \hat{w}_1(\mathbf{i}_j).$$

Proof: For any $j \in \{1,2,3\}$, we have that:

$$\begin{aligned}
&\langle \lambda'_{j,1} \otimes \cdots \otimes \lambda'_{j,d}, w_2 \rangle \\
= &\sum_{\mathbf{i} \in [k]^d} \lambda'_{j,1}(\mathbf{i}(1)) \cdots \lambda'_{j,d}(\mathbf{i}(d)) \cdot w(\mathbf{i}) + \sum_{\mathbf{i} \in [k_1]^d \setminus [k]^d} \sum_{h \in [s]} \lambda'_{j,1}(\mathbf{i}^{(h)}(1)) \cdots \lambda'_{j,d}(\mathbf{i}^{(h)}(d)) \cdot w_2(\mathbf{i}^{(h)}) \\
&+ \sum_{\mathbf{i} \in [k_2]^d \setminus [\bar{k}_1]^d} \lambda'_{j,1}(\mathbf{i}(1)) \cdots \lambda'_{j,d}(\mathbf{i}(d)) \cdot w_2(\mathbf{i}) \\
= &\sum_{\mathbf{i} \in [k]^d} \lambda_{j,1}(\mathbf{i}(1)) \cdots \lambda_{j,d}(\mathbf{i}(d)) \cdot w(\mathbf{i}) + \sum_{\mathbf{i} \in [k_1]^d \setminus [k]^d} \lambda_{j,1}(\mathbf{i}(1)) \cdots \lambda_{j,d}(\mathbf{i}(d)) \sum_{h \in [s]} a_h \cdot w_2(\mathbf{i}^{(h)}) \\
= &\sum_{\mathbf{i} \in [k_1]^d} \lambda_{j,1}(\mathbf{i}(1)) \cdots \lambda_{j,d}(\mathbf{i}(d)) \cdot w_1(\mathbf{i}) \\
= &\langle \lambda_{j,1} \otimes \cdots \otimes \lambda_{j,d}, w_1 \rangle = \hat{w}_1(\mathbf{i}_j),
\end{aligned}$$

where the first identity follows by the definition w_2 in Step 3a, the second identity follows by the definition of $\lambda'_{j,1}, \ldots, \lambda'_{j,d}$ in Step 6a (recalling our assumption that $a_1 = 1$), and the third identity follows by the definition of w_1 in Step 4a. ■

6.4 Soundness

Suppose that φ is not satisfiable, we shall show that in this case $\mathcal{V}$ rejects with probability at least $(\frac{\gamma}{d})^{O(d)}$. In what follows, let $\delta := (\frac{\gamma}{d})^{O(1)}$ denote the relative distance of C_2, and let $\mu := (\frac{\gamma}{d})^{O(1)}$

denote the robustness parameter of the $\sqrt{n_2}$-query robust local tester for the code C_2, given by Theorem 3.33 (recalling that $\text{image}(C_2) = \text{image}((C_0)^{\otimes 4})$).

Let $z^* \in \{0,1\}^{[k_2]^d}$ denote $\mathcal{P}$'s message in Step 3a, and let v^* denote $\mathcal{P}$'s messages in Step 3c. First note that we may assume that $\psi_{\text{TEST}}(v^*) = \textsf{ACCEPT}$, since otherwise $\mathcal{V}$ clearly rejects on Step 3d. Next observe that if z^* is $\frac{\delta^d \cdot \mu^2}{32}$-far from $(C_2)^{\otimes d}$, then by the properties of the robust local tester for $(C_2)^{\otimes d}$, with probability at least $(\frac{\gamma}{d})^{O(d)}$, we have that $z^*|_I$ is $(\frac{\gamma}{d})^{O(d)}$-far from $\psi^{-1}_{\text{TEST}}(\textsf{ACCEPT})$. Assuming that this event holds, our assumption that $\psi_{\text{TEST}}(v^*) = \textsf{ACCEPT}$ implies that v^* is $(\frac{\gamma}{d})^{O(d)}$-far from $z^*|_I$. But in this case, $\mathcal{V}$ will reject in Step 3e with probability at least $(\frac{\gamma}{d})^{O(d)}$ (and overall $\mathcal{V}$ will reject with probability $(\frac{\gamma}{d})^{O(d)}$). Hence we may assume that z^* is $\frac{\delta^d \cdot \mu^2}{32}$-close to a codeword $c_2^* = (C_2)^{\otimes d}(w_2^*)$. In what follows, let $w^* \in \{0,1\}^{[k]^d}$ be the restriction of w_2^* to $[k]^d$, let $w_1^* \in \mathbb{F}^{[k_1]^d}$ be defined as in Step 4a with respect to w_2^*, and let $P^* = Q \cdot P_{\varphi, w_1^*}$.

We now claim that with probability at least $\frac{1}{2}$ over the choice of Q in Step 4b, it holds that

$$\sum_{\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3 \in [k]^d, b_1,b_2,b_3 \in [k]} P^*\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) \neq 0. \tag{12}$$

To see the above, note that by the definition of P_{φ,w_1^*}, and by our setting of $w_1^*(\mathbf{i}) = w_2^*(\mathbf{i}) = w^*(\mathbf{i})$ for any $\mathbf{i} \in [k]^d$, we have that

$$P_{\varphi,w_1^*}\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) = I_\varphi\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) \cdot \Pi_{j=1}^3 \left(w^*(\mathbf{i}_j) - b_j\right)$$

for any $\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3 \in [k]^d$ and $b_1,b_2,b_3 \in [k]$. Furthermore, by the definition of I_φ in Step 2a, and by assumption that w^* is not a satisfying assignment for φ, there exist $\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3 \in [k]^d$ and $b_1,b_2,b_3 \in [k]$ so that $P_{\varphi,w_1^*}\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) \neq 0$. Moreover, by our setting of parameters, we have that $8 \cdot (3d+3) \cdot k = 24(d+1)k \leq n_1$, and consequently Lemma 3.39 implies that $P^* = Q \cdot P_{\varphi,w_1^*}$ satisfies (12) with probability at least $\frac{1}{2}$ over the choice of Q. In what follows, assume that (12) holds.

Next observe that by our assumption (12) and the definition $\lambda_1,\ldots,\lambda_{(d+1)/3}$ (in Step 5a), we have that

$$\begin{aligned}\langle \lambda_1 \otimes \cdots \otimes \lambda_{(d+1)/3}, w_3^* \rangle &= \sum_{\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3 \in [k]^d, b_1,b_2,b_3 \in [k]} w_3^*\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) \\ &= \sum_{\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3 \in [k]^d, b_1,b_2,b_3 \in [k]} P^*\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) \neq 0,\end{aligned}$$

where $w_3^* \in \mathbb{F}^{[3k_1]^{3d+3}}$ is the evaluation table of P^* on $[3k_1]^{3d+3}$. Moreover, by our setting of parameters, we have that $3k_1 = 3(1+\frac{1}{s^2})k \leq 6k \leq \frac{n_1}{4}$, and so by Lemma 3.37 and Theorem 3.33, $C_1 = (\textsf{PRS}_{3k_1,n_1})^{\otimes 9}$ has relative distance at least $\frac{1}{2}$, and is an $(n_1^2, \Omega(1))$-robust LTC. Consequently, applying Theorem 5.4 with implicit input $c_1^* = (C_1)^{\otimes (d+1)/3}(w_3^*)$, with probability at least $2^{-O(d)}$, the outcome of the protocol $(\mathcal{P}_{\textsf{in}}, \mathcal{V}_{\textsf{in}})$ in Step 5b is either that $\mathcal{V}_{\textsf{in}}$ rejects, or a point $(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3) \in (\mathbb{F} \setminus [3k_1])^{3d+3}$ and a value $\alpha' \in \mathbb{F}$ so that

$$P^*\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right) \neq \alpha'. \tag{13}$$

In the former case, $\mathcal{V}$ also rejects, and so we are done. Hence we may assume that (13) holds.

Next suppose that in Step 6a, $\mathcal{P}$ sends $\alpha_j^* \neq \hat{w_1^*}(\mathbf{i}_j)$ for some $j \in \{1,2,3\}$. We shall show that in this case $\mathcal{V}$ rejects with probability at least $(\frac{\gamma}{d})^{O(d)}$. To this end, first recall that by Claim 6.3, we have that

$$\langle \lambda'_{j,1} \otimes \cdots \otimes \lambda'_{j,d}, w_2^* \rangle = \hat{w}_1^*(\mathbf{i}_j) \neq \alpha_j^*.$$

Moreover, by Lemma 4.21, the A-extension $\widetilde{C}_2$ of C_2 also has relative distance at least δ, and a $(\sqrt{n_2}, \mu)$-robust local tester (since $\text{image}(\widetilde{C}_2) = \text{image}\left(\widetilde{(C_0)^{\otimes 4}}\right) = \text{image}((\widetilde{C}_0)^{\otimes 4})$), and $c_2^* = (C_2)^{\otimes d}(w_2^*) = (\widetilde{C}_2)^{\otimes d}(w_2^*)$ by assumption that $1 \in A$.

Consequently, applying Theorem 5.4 with implicit input $c_2^* = (\widetilde{C}_2)^{\otimes d}(w_2^*)$, with probability at least $\frac{\delta^d \cdot \mu^2}{16}$, the outcome of the protocol $(\mathcal{P}'_{\text{in}}, \mathcal{V}'_{\text{in}})$ in Step 6c is either that $\mathcal{V}'_{\text{in}}$ rejects, or a point $\mathbf{i}'_j \in [n_2]^d$ and a value $\alpha'_j \in \mathbb{F}$ so that $c_2^*(\mathbf{i}'_j) \neq \alpha'_j$. Moreover, $\mathbf{i}'_j$ is a uniform random point (depending on $\mathcal{V}'_{\text{in}}$'s randomness string) that is computed by $\mathcal{V}'_{\text{in}}$ at the end of the interaction.

Let E_1 be the event that $\mathcal{V}'_{\text{in}}$ either rejects, or outputs $\mathbf{i}'_j \in [n_2]^d$ and $\alpha'_j \in \mathbb{F}$ so that $c_2^*(\mathbf{i}'_j) \neq \alpha'_j$. Let E_2 be the event that $z^*(\mathbf{i}'_j) = c_2^*(\mathbf{i}'_j)$. Then by the above, we have that $\Pr[E_1] \geq \frac{\delta^d \cdot \mu^2}{16}$, and by assumption that z^* is $\frac{\delta^d \cdot \mu^2}{32}$-close to c_2^*, and that $\mathbf{i}'_j$ is a uniform random point, we have that $\Pr[E_2] \geq 1 - \frac{\delta^d \cdot \mu^2}{32}$. Consequently, by a union bound, with probability at least $\frac{\delta^d \cdot \mu^2}{32} = (\frac{\gamma}{d})^{O(d)}$, both events E_1 and E_2 hold. Next assume that both events E_1 and E_2 hold. If $\mathcal{V}'_{\text{in}}$ rejects, then $\mathcal{V}$ also rejects, and so we are done. Otherwise, our assumption implies that $z^*(\mathbf{i}'_j) = c_2^*(\mathbf{i}'_j) \neq \alpha'_j$, and so $\mathcal{V}$ will reject in Step 6c. Hence we may assume that $\mathcal{P}$ sends $\alpha_j^* = \hat{w}_1^*(\mathbf{i}_j)$ for every $j \in \{1, 2, 3\}$.

Finally, by our assumption (13), this implies in turn that

$$\begin{aligned}
&Q\left(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3\right) \cdot \hat{I}_\varphi\left(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3\right) \cdot \Pi_{j=1}^3 \left(\alpha_j^* - b_j\right) \\
&= Q\left(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3\right) \cdot \hat{I}_\varphi\left(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3\right) \cdot \Pi_{j=1}^3 \left(\hat{w}_1^*(\mathbf{i}_j) - b_j\right) \\
&= P^*\left(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3\right) \neq \alpha',
\end{aligned}$$

and so $\mathcal{V}$ will reject in Step 7.

We conclude that in the case that φ is unsatisfiable, $\mathcal{V}$ will reject with probability at least $(\frac{\gamma}{d})^{O(d)}$.

6.5 Zero-Knowledge

In this section we prove that the ZK-IOP of Figures 5-7 has t-ZK for $t = n^{\beta'}$. Since our ZK-IOP for 3-SAT runs several ZK sub-procedures and ZK codes, this requires careful analysis. In particular, the internal ZK-IOPs should have ZK *with auxiliary inputs*. In the following, we first state the exact ZK with auxiliary input property needed by the internal ZK-IOPs, then discuss the effect of their composition on the ZK property, and the properties needed from the underlying ZK codes, and finally prove the ZK property stated in Theorem 6.1.

The ZK with Auxiliary Inputs Requirement from the Internal ZK-IOPs. To show the zero-knowledge property of our protocol, we shall use the following lemma which, roughly, states that if an IOP has t-ZK with a black-box straight-line simulator, then it also has t-ZK with auxiliary inputs. A similar result is known for standard interactive proofs [GO94]. The following lemma shows a slightly stronger implication, which allows the simulator to access a *different* oracle than x_{imp}, and to make a different (possibly larger) number of oracle queries.[28] Its proof is given in Appendix A.

[28] Jumping ahead, we note that this generalization is useful because the lemma will be applied to the simulators from Theorem 5.4 (for general and tensor codes), who receive a *row/subspace oracle* instead of the actual codeword given as implicit input to the verifier, and make slightly more queries than the verifier. We also note that while for some choices of oracles $\mathcal{O}$ and functions f the implication might be trivial (e.g., this might be the case if the simulator is given oracle access to the NP witness and/or f is sufficiently small to allow the simulator to fully read its oracle), still the implication of the lemma is meaningful for some choices of the oracle and the function f (as we show below in the proof of Theorem 6.1).

Lemma 6.4. *Let $t \in \mathbb{N}$, let $\mathcal{R}$ be an NP relation with corresponding promise problem $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$, and let $(\mathcal{P}, \mathcal{V})$ be an IOP for $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$. Assume that there exists a black-box straight-line simulator Sim with oracle $\mathcal{O}$, and a function $f : \mathbb{N} \to \mathbb{N}$, such that for every t-restricted verifier $\mathcal{V}^*$, and every $((x_{\exp}, x_{\mathrm{imp}}), w) \in \mathcal{R}$, we have*

$$\left(\mathsf{View}_{\mathcal{V}^*}\left(x_{\exp}, x_{\mathrm{imp}}, w\right), q_{\mathcal{V}^*}\right)_{r_P, r_V} \equiv \left(\mathsf{View}_{\mathcal{V}^*}^{\mathsf{Sim}^{\mathcal{O}}}\left(x_{\exp}, x_{\mathrm{imp}}, w\right), f(q_S)\right)_{r_V, r_{\mathsf{Sim}}}$$

where $r_P, r_V, r_{\mathsf{Sim}}$ are the random coins of $\mathcal{P}, \mathcal{V}^*$ and Sim (respectively); and $q_{\mathcal{V}^*}$ (q_S, resp.) denotes the number of queries which $\mathcal{V}^*$ (Sim, resp.) makes to all her oracles ($\mathcal{O}$, resp.). Then for every t-restricted verifier $\mathcal{V}^*_{\mathsf{aux}}$ with auxiliary input there exist a polynomial $p(n)$, and a black-box straight-line simulator $\mathsf{Sim}_{\mathsf{aux}}$, such that for every $((x_{\exp}, x_{\mathrm{imp}}), w) \in \mathcal{R}$ and every auxiliary input $z \in \{0,1\}^{p(|x_{\exp}|+|x_{\mathrm{imp}}|)}$, we have

$$\left(\mathsf{View}'_{\mathcal{V}^*_{\mathsf{aux}}}\left(x_{\exp}, x_{\mathrm{imp}}, w, z\right), q'_V\right)_{r_P, r'_V} \equiv \left(\mathsf{View}^{\mathsf{Sim}^{\mathcal{O}}_{\mathsf{aux}}}_{\mathcal{V}^*_{\mathsf{aux}}(\cdot,z)}\left(x_{\exp}, x_{\mathrm{imp}}, w\right), f(q'_S)\right)_{r'_V, r'_{\mathsf{Sim}}} \tag{14}$$

where $r_P, r'_V, r'_{\mathsf{Sim}}$ are the random coins of $\mathcal{P}, \mathcal{V}^*_{\mathsf{aux}}$ and $\mathsf{Sim}_{\mathsf{aux}}$ (respectively); q'_V (q'_S, resp.) denotes the number of queries which $\mathcal{V}^*_{\mathsf{aux}}$ ($\mathsf{Sim}_{\mathsf{aux}}$, resp.) makes to all her oracles ($\mathcal{O}$, resp.), and $\mathsf{View}^{\mathsf{Sim}^{\mathcal{O}}_{\mathsf{aux}}}_{\mathcal{V}^*_{\mathsf{aux}}(\cdot,z)}\left(x_{\exp}, x_{\mathrm{imp}}, w\right)$ is the view of $\mathcal{V}^*_{\mathsf{aux}}$ when it has auxiliary input *z, and interacts with $\mathsf{Sim}_{\mathsf{aux}}$ (instead of $\mathcal{P}$).*[29]

Composition of internal ZK-IOPs. Our ZK-IOP executes 4 internal ZK-IOPs (in Steps 5a and 6b in Figure 7). We stress that while these systems are executed *sequentially*, this in effect induces an *interleaved* (concurrent) execution of the internal systems. This is because in an execution of an internal ZK-IOP, a malicious verifier $\mathcal{V}^*$ might continue making oracle queries to prover oracle messages sent in *previous* internal ZK-IOP executions. Thus, the executions of the internal ZK-IOPs might not terminate until the end of the execution of the composed system of Figures 5-7, and so they are executed concurrently.

ZK Codes Used by the ZK-IOP for 3-SAT. The ZK-IOP of Figures 5-7 uses several codes, and two codewords – c_1 and c_2 – which encode randomized messages w_1, w_2, respectively, which depend on the witness w. In proving ZK, it would be more useful to employ a different – though equivalent – definition of w_1, w_2. The next lemma formally describes this alternative view.

Lemma 6.5 (Alternative View of w_1, w_2.)**.** *The definition of w_1, w_2 in Steps 4a and 3a (respectively) in Figure 6 is equivalent to the following:*

- *Define $w_1 \in \mathbb{F}^{[k_1]^d}$ as follows. For any $\mathbf{i} \in [k]^d$, $w_1(\mathbf{i}) = w(\mathbf{i})$. For any $\mathbf{i} \in [k_1]^d \setminus [k]^d$, $w_1(\mathbf{i})$ is a uniformly random element in $\mathbb{F}$, sampled independently at random.*
- *Define $\bar{w}_2 \in \{0,1\}^{[\bar{k}_1]^d}$ such that $\bar{w}_2$ is consistent with w_1. That is:*
  1. *For every $\mathbf{i} \in [k]^d$, $\bar{w}_2(\mathbf{i}) = w_1(\mathbf{i})$.*
  2. *For every $\mathbf{i} \in [\bar{k}_1]^d \setminus [k]^d$, let $\ell_0 \in [d]$ be the first index so that $\mathbf{i}(\ell_0) \notin [k]$. If there exists another index $\ell \neq \ell_0$ so that $\mathbf{i}(\ell) \notin [k]$ and $\mathbf{i}(\ell) - k \not\equiv 1 \mod s$, then $\bar{w}_2(\mathbf{i}) = 0$. Otherwise, there exist $h \in [s]$ and $\mathbf{j} \in [k_1]^d \setminus [k]^d$ such that $\mathbf{j}^{(h)} = \mathbf{i}$, and $\bar{w}_2(\mathbf{i})$ is the projection of $w_1(\mathbf{j})$ on the basis element a_h.*

[29] We remind the reader that $\mathsf{View}'_{\mathcal{V}^*_{\mathsf{aux}}}\left(x_{\exp}, x_{\mathrm{imp}}, w, z\right)$ is defined similarly to Definition 3.11, but includes also the auxiliary input z.

- *Define $w_2 \in \{0,1\}^{[k_2]^d}$ as follows. For any $\mathbf{i} \in [\bar{k}_1]^d$, $w_2(\mathbf{i}) = \bar{w}_2(\mathbf{i})$. For any $\mathbf{i} \in [k_2]^d \setminus [\bar{k}_1]^d$, $w_2(\mathbf{i})$ is a uniformly random bit, sampled independently at random.*

Proof: We first show that $w_1(\mathbf{i})$ is distributed as claimed. To this end, first note that $w_1(\mathbf{i}) = w(\mathbf{i})$ for any $\mathbf{i} \in [k]^d$ by our setting of $w_1(\mathbf{i}) = w_2(\mathbf{i})$ and $w_2(\mathbf{i}) = w(\mathbf{i})$ for any $\mathbf{i} \in [k]^d$ in Steps 4a and 3a, respectively.

Next assume that $\mathbf{i} \notin [k]^d$, and let $\ell_0 \in [d]$ be the first index so that $\mathbf{i}(\ell_0) \notin [k]$. Then for any $h \in [s]$, $\mathbf{i}^{(h)}$ (defined in Step 4a) is a vector in $[\bar{k}_1]^d$, satisfying that ℓ_0 is the first index $\ell \in [d]$ so that $\mathbf{i}^{(h)}(\ell) \notin [k]$, and $\mathbf{i}(\ell) - k \equiv 1 \mod s$ for any other index $\ell \neq \ell_0$ so that $\mathbf{i}^{(h)}(\ell) \notin [k]$. Consequently, for any $h \in [s]$, $w_2(\mathbf{i}^{(h)})$ is set to be a uniform random bit in Step 3a. Moreover, for any $h \neq h' \in [s]$, we have that $\mathbf{i}^{(h)}(\ell_0) = k + s \cdot (\mathbf{i}(\ell_0) - k - 1) + h \neq k + s \cdot (\mathbf{i}(\ell_0) - k - 1) + h' = \mathbf{i}^{(h')}(\ell_0)$, and so $\mathbf{i}^{(h)} \neq \mathbf{i}^{(h')}$. So $w_2(\mathbf{i}^{(1)}), \ldots, w_2(\mathbf{i}^{(s)})$ are uniform and independent random bits, which implies in turn that $w_1(\mathbf{i}) = \sum_{h \in [s]} a_h \cdot w_2(\mathbf{i}^{(h)})$ is set to be a uniform random element in $\mathbb{F}$ in Step 4a. Moreover, for any $\mathbf{i} \neq \mathbf{i}' \notin [k]^d$ and $h, h' \in [s]$, we have that $\mathbf{i}^{(h)} \neq (\mathbf{i}')^{(h')}$, and consequently $w_1(\mathbf{i})$ for $\mathbf{i} \notin [k]^d$ are independent. So w_1 is distributed as claimed.

Next we show that w_2 is distributed as claimed. For this, first note that for any $\mathbf{i} \in [k]^d$, we clearly have that $w_2(\mathbf{i}) = \bar{w}_2(\mathbf{i}) = w_1(\mathbf{i})$. Also, for any $\mathbf{i} \in [k_2]^d \setminus [\bar{k}_1]^d$, $w_2(\mathbf{i})$ is a uniformly random bit, sampled independently at random, by our setting of w_2 in Step 3a, and since w_1 only depends on values of $w_2(\mathbf{i})$ for $\mathbf{i} \in [\bar{k}_1]^d$.

Next assume that $\mathbf{i} \in [\bar{k}_1]^d \setminus [k]^d$, and let $\ell_0 \in [d]$ be the first index so that $\mathbf{i}(\ell_0) \notin [k]$. If there exists another index $\ell \neq \ell_0$ so that $\mathbf{i}(\ell) \notin [k]$ and $\mathbf{i}(\ell) - k \not\equiv 1 \mod s$, then $w_2(\mathbf{i}) = \bar{w}_2(\mathbf{i}) = 0$, by our setting in Step 3a. Otherwise, let $h \in [s]$ be such that $\mathbf{i}(\ell_0) - k \equiv h \mod s$, and let $\mathbf{j} \in [k_1]^d \setminus [k]^d$ be given by $\mathbf{j}(\ell) = \mathbf{i}(\ell)$ if $\mathbf{i}(\ell) \in [k]$, $\mathbf{j}(\ell_0) = \frac{\mathbf{i}(\ell) - k - h}{s} + k + 1$, and $\mathbf{j}(\ell) = \frac{\mathbf{i}(\ell) - k - 1}{s} + k + 1$ otherwise. Then $\mathbf{j}^{(h)} = \mathbf{i}$, and $w_1(\mathbf{j}) = \sum_{h \in [s]} a_h \cdot w_2(\mathbf{j}^{(h)})$ by our setting of w_1 in Step 4a. Consequently, $w_2(\mathbf{i}) = w_2(\mathbf{j}^{(h)})$ is indeed the projection of $w_1(\mathbf{j})$ on the basis element a_h. So w_2 is also distributed as claimed. ■

In the next remark, we summarize the properties needed from the codes C_1, C_2.

Remark 6.6 (ZK Codes used in Theorem 6.1)**.** *The ZK-IOP of Figures 5- 7 implicitly uses two codes C_1, C_2. We now explicitly describe their ZK properties.*

ZK property of the high-rate encoding C_2. *Recall from Step 1c on Figure 5, that $C_2 : \{0,1\}^{k_2} \to \{0,1\}^{n_2}$ has $\tilde{t}$-ZK with respect to the $\bar{k}_1$-randomized encoding function for $\tilde{t} := (\frac{\gamma}{d})^{O(1)} \cdot n_2$. Then by Corollary 4.6, $(C_2)^{\otimes d}$ (with the $[\bar{k}_1]^d$-randomized encoding function) has ZK against $\tilde{t}$-row restricted adversaries that have oracle access to a row oracle for $c_2 \in C_2 \otimes (C_2)^{\otimes(d-1)}$ (Here, we use the alternative view of w_2 given in Lemma 6.5 to view c_2 as generated using the $[\bar{k}_1]^d$-randomized encoding function). Moreover, by Lemma 4.21, the extended code $(\widetilde{C}_2)^{\otimes d} = \widetilde{C}_2 \otimes (\widetilde{C}_2)^{\otimes(d-1)}$ has ZK on $\{0,1\}^{[k_2]^d}$ with respect to the $[\bar{k}_1]^d$-randomized encoding function, against $\tilde{t}$-row restricted adversaries that have oracle access to a row oracle for c_2.*

ZK property of the low-rate encoding C_1. *Recall from Step 4a in Figure 6, that $\hat{w}_1 \in (\mathsf{RS}_{k_1,n_1})^{\otimes d}$. In the following, we abuse notation and use $\hat{w}_1$ to denote the* punctured *codeword* in $(\mathsf{PRS}_{k_1,n_1})^{\otimes d}$ *obtained by puncturing the first k_1 coordinates in the codeword of the base code PRS_{k_1,n_1}. Notice that since $\mathcal{P}$ never sends $\hat{w}_1$ to $\mathcal{V}^*$, and the values $\hat{w}_1(\mathbf{i}_j)$ defined in Step 6 in Figure 7 satisfy $\mathbf{i}_j \in [[n_1] \setminus [k_1]]^d$ (because they are derived from w_3, which is itself punctured in the first $3k_1$ coordinates), we can replace $\hat{w}_1$ with its punctured version without effecting the protocol.*

By Lemma 4.14, PRS_{k_1,n_1} has $(k_1 - k)$-uniform ZK with respect to the k-randomized encoding function. By Corollary 4.12, this implies in turn that the LDE encoding $\hat{w}_1$ has ZK against adaptive $(k_1 - k)$-subspace

restricted adversaries with respect to the $[k]^d$-randomized encoding function. (Here, we use the alternative view of w_1 given in Lemma 6.5 to view c_1 as generated using the $[k]^d$-randomized encoding function.)

Finally, recall from Step 4b in Figure 6 that $c_1 \in B^{\otimes(d+1)}$, and note that the restriction of c_1 to any oracle query made to a subspace oracle in direction $[d+1] \setminus \{\ell\}$ with respect ot the code $B^{\otimes(d+1)}$ is a codeword of $B^{\otimes d}$. Moreover, if $\ell \neq d+1$, then this subspace can be generated given 3 queries to an oracle in direction $[d] \setminus \{\ell\}$ for $\hat{w}_1$. Indeed, dimension $d+1$ in $c_1 \in B^{\otimes(d+1)}$ corresponds to the values of b_1, b_2, b_3. Therefore, for $\ell \neq d+1$, a subspace oracle in direction $[d+1] \setminus \{\ell\}$ for $c_1 \in B^{\otimes(d+1)}$ can be generated given the restrictions of $\hat{w}_1$ to 3 subspaces in direction $[d] \setminus \{\ell\}$, since these can then be used to generate the value of P on the subspace (because b_1, b_2, b_3 are public).

The following is our main technical lemma.

Lemma 6.7 (ZK property of IOP for 3-SAT). *Let $t \in \mathbb{N}$ be a parameter, and let $q := \sqrt{n_2}$ denote the query complexity of the local tester for C_2 and $(C_2)^{\otimes d}$. (cf., remark on Step 1c in Figure 6). Suppose that C_2 has $(t+4q+3)$-ZK with respect to the $\bar{k}_1$-randomized encoding function, and PRS_{k_1,n_1} has $3(t+3)$-uniform ZK with respect to the k-randomized encoding function. Then the ZK-IOP of Figures 5-7 has t-ZK.*

Before proving the above lemma, we show how it implies the ZK property stated in Theorem 6.1.

Proof of ZK Property in Theorem 6.1: Let $t := n^{1/(2d)}$. By Remark 6.6, C_2 has t_2-ZK with respect to the $\bar{k}_1$-randomized encoding function for $t_2 = (\frac{\gamma}{d})^{O(1)} \cdot n_2$, while PRS_{k_1,n_1} has t_1-uniform ZK with respect to the k-randomized encoding function for $t_1 = k_1 - k$.

By our setting of parameters in Step 1a of Figure 5, for sufficiently large n, we have that

$$t_2 - 4q - 3 = \left(\frac{\gamma}{d}\right)^{O(1)} \cdot n_2 - 4\sqrt{n_2} - 3 = \Omega(n_2) \geq \Omega(k) = \Omega(n^{1/d}) \geq t,$$

and

$$\frac{t_1}{3} - 3 = \frac{k_1 - k}{3} - 3 = \Omega(k_1 - k) = \Omega\left(\frac{k}{s^2}\right) = \Omega\left(\frac{n^{1/d}}{\log^2 n}\right) \geq t.$$

We conclude that for sufficiently large n, $t_2 \geq t + 4q + 3$ and $t_1 \geq 3(t+3)$, and consequently Lemma 6.7 implies that the ZK-IOP has t-ZK for $t := n^{1/(2d)}$. Theorem 6.1 follows recalling that d was chosen to be a sufficiently large constant (depending on the constant β).

6.5.1 Proof of Main Technical Lemma 6.7

We now turn to the proof of our Main Technical Lemma 6.7. In the following, we assume that t is such that: (1) C_2 has $(t+4q+3)$-ZK and (2) PRS_{k_1,n_1} has $3(t+3)$-uniform ZK. Let $\mathcal{V}^*$ be a t-restricted (possibly malicious) verifier in the IOP of Figures 5-7. We first make some necessary remarks, then describe the simulator and prove that the simulation is perfect.

Answering queries to c_2. Let Sim_C be the black-box straight-line simulator for adaptive $(t+4q+3)$-row restricted adversaries that have oracle access to a row oracle for $c_2 \in \widetilde{C}_2 \otimes \widetilde{C}_2^{\otimes(d-1)} \cap \widetilde{C}_2^{\otimes d}(\{0,1\}^{[k_2]^d})$ (the existence of Sim_C follows from the $(t+4q+3)$-ZK of C_2 by Remark 6.6). We answer $\mathcal{V}^*$'s queries to c_2 using Sim_C, as follows.

- Initialize a set $\mathcal{Q}_2 := \emptyset$. (Throughout the simulation, $\mathcal{Q}_2 \subseteq [n_2]$ will consist of the rows of c_2 – when viewed as a codeword in $\widetilde{C}_2 \otimes \widetilde{C}_2^{\otimes(d-1)}$ – to which $\mathcal{V}^*$ made a query.)

- Answer queries $(l_1, l_2) \in [n_2] \times [n_2]^{d-1}$ of $\mathcal{V}^*$ to c_2 as follows:
  - If $l_1 \notin \mathcal{Q}_2$ then add l_1 to $\mathcal{Q}_2$. Then, use Sim_C to emulate $c_2(l_1, *)$ (i.e., the l_1'st row of c_2), and store $c_2(l_1, *)$.
  - Retrieve the simulated row $c_2(l_1, *)$, and output $c_2(l_1, l_2)$ as the oracle answer.

We note that this procedure can also answer *row* queries to c_2. This will be used in the simulation below.

Answering queries to c_1. [30] Let $\mathsf{Sim}_{\mathsf{LDE}}$ be the black-box straight-line simulator that can perfectly simulate the answers to queries of any adaptive $3(t+3)$-subspace restricted adversary $\mathcal{A}$ with oracle access to $\hat{w}_1$ (the existence of $\mathsf{Sim}_{\mathsf{LDE}}$ follows from the $3(t+3)$-uniform ZK of $\mathsf{PRS}^{\otimes d}_{k_1,n_1}$ using Remark 6.6). Recall that the execution of $\mathcal{A}$ consists of two phases, where in the first phase $\mathcal{A}$ makes point queries, then $\mathcal{A}$ picks a subset $J \subseteq [d]$, and in the second phase $\mathcal{A}$ can query a subspace oracle in direction J. In the proof below we will only need to consider adversaries $\mathcal{A}$ that pick $J = [d] \setminus \{\ell\}$ for some ℓ. The simulation for $c_1 \in B^{\otimes(d+1)}$ will follow a similar 2-phase structure. We will only support restrictions to subspaces in direction $[d+1] \setminus \{\ell\}$ for $\ell \neq d+1$ since these could be reduced to an axis of $\hat{w}_1$, as explained in Remark 6.6. Our simulation of the ZK-IOP for 3-SAT will obey this constraint.

We answer queries to $c_1 \in \left(\mathsf{PRS}^{\otimes 3}_{3k_1,n_1}\right)^{\otimes(d+1)}$ using $\mathsf{Sim}_{\mathsf{LDE}}$, as follows.

- Initialize a set $\mathcal{Q} := \emptyset$. (Intuitively, $\mathcal{Q}$ will store the point queries which the adversary made in the first phase of the simulation.)
- Answer point-queries $(l_1, \ldots, l_{d+1})$ (where $l_h = (l_h^1, l_h^2, l_h^3) \in ([n_1 - 3k_1]^3)$ for every $h \in [d+1]$) made in the first phase of the simulation as follows.
  - Add $(l_1, \ldots, l_{d+1})$ to $\mathcal{Q}$.[31]
  - For every $1 \leq j \leq 3$, use $\mathsf{Sim}_{\mathsf{LDE}}$ to emulate $\hat{w}_1(l_1^j, \ldots, l_d^j)$, and use the simulated values, together with l_{d+1} and Q, to generate $c_1(l_1, \ldots, l_{d+1})$ (which is the value of P at some point).
- When the simulation enters its second phase, the simulator obtains a subspace $[d] \setminus \{\ell\}$. For every $(l_1, \ldots, l_{d+1}) \in \mathcal{Q}$ and every $j \in \{1,2,3\}$, it then obtains from $\mathsf{Sim}_{\mathsf{LDE}}$ the lines $(\hat{w}_1(i'_1, \ldots, i'_{\ell-1}, l_\ell^j, i'_{\ell+1}, \ldots, i'_d))_{i'_1,\ldots,i'_{\ell-1},i'_{\ell+1},\ldots,i'_d \in [n_1-k_1]}$, and uses them to compute the subspace $(c_1(i'_1, \ldots, i'_{\ell-1}, l_\ell, i'_{\ell+1}, \ldots, i'_{d+1}))_{i'_1,\ldots,i'_{\ell-1},i'_{\ell+1},\ldots,i'_d \in [n_1-3k_1]^3}$ (as explained in Remark 6.6, for any $\ell \leq d$, any query to a subspace oracle in direction $[d+1] \setminus \{\ell\}$ for c_1 can be computed with 3 queries to a subspace oracle in direction $[d] \setminus \{\ell\}$ for $\hat{w}_1$). The simulator then provides these lines to the adversary.
- In the second phase of the simulation, the simulator answers any query $i = (i^1, i^2, i^3) \in [n_1 - 3k_1]^3$ of the adversary by using $\mathsf{Sim}_{\mathsf{LDE}}$ to simulate the lines $(\hat{w}_1(i'_1, \ldots, i'_{\ell-1}, i^j, i'_{\ell+1}, \ldots, i'_d))_{i'_1,\ldots,i'_{\ell-1},i'_{\ell+1},\ldots,i'_d \in [n_1-k_1]}, j \in \{1,2,3\}$, and using these lines to generate the line $(c_1(i_1, \ldots, i'_{\ell-1}, i, i'_{\ell+1}, \ldots, i'_{d+1}))_{i'_1,\ldots,i'_{\ell-1},i'_{\ell+1},\ldots,i'_d \in [n_1-3k_1]^3}$ as described above.

[30] We note that while $\mathcal{V}^*$ cannot directly query c_1 (since it is not given an oracle to c_1), as part of the simulation we define helper sub-procedures which might query c_1. We will therefore need to handle such queries during the simulation. This will be made clearer when we define the simulator Sim for $\mathcal{V}^*$ later in the proof.

[31] We assume that $\mathcal{A}$ never makes repeated queries. This is without loss of generality since repeated queries could be answered consistently using the previously generated simulated answers.

ZK Property of the Internal sumcheck ZK-IOP $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$ (High-Rate Encoding). Let $c \in C_2^{\otimes d}$, and let k and q denote the message length, and query complexity of the local tester, of C_2. Recall from Lemma 5.9 that for any t, the sumcheck ZK-IOP $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$ has black-box straight-line ZK against t-restricted adversaries that are given a point and row oracle for $c \in \widetilde{C}_2 \otimes \widetilde{C}_2^{\otimes(d-1)}$, with a $(t+q+1)$-row restricted simulator. Therefore, by Lemma 6.4, for every t, there exists a black-box straight-line $(t+q+1)$-row restricted simulator $\mathsf{Sim}'_{\mathsf{in}}$ that can simulate the view of any t-restricted verifier $\mathcal{V}^*_{\mathsf{in}}$ *with auxiliary inputs* in the system $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$, where $\mathcal{V}^*_{\mathsf{in}}$ can make both point-wise and row queries to c. In the simulation (described below) we will use the simulator $\mathsf{Sim}'_{\mathsf{in}}$ as a sub-procedure.

ZK Property of the Internal sumcheck ZK-IOP $(\mathcal{P}_{\mathsf{in}}, \mathcal{V}_{\mathsf{in}})$ (Low-Rate Encoding). Let $c_1 \in B^{\otimes(d+1)}$, and let k, n denote the message and block lengths of C_1, respectively. Recall from Theorem 5.4 that for any t, the sumcheck ZK-IOP $(\mathcal{P}_{\mathsf{in}}, \mathcal{V}_{\mathsf{in}})$ has black-box straight-line ZK against t-restricted adversaries with an adaptive $(t+2)$-subspace restricted simulator, that only queries subspaces in direction $[d+1] \setminus \{\ell\}$ for $\ell \in \{1,2,3\}$. Therefore, by Lemma 6.4, for every t, there exists a black-box straight-line adaptive $(t+2)$-subspace restricted simulator $\mathsf{Sim}_{\mathsf{in}}$ that can perfectly simulate the view of any t-restricted verifier $\mathcal{V}^*_{\mathsf{in}}$ *with auxiliary inputs* in the system $(\mathcal{P}_{\mathsf{in}}, \mathcal{V}_{\mathsf{in}})$, and the simulation is perfect. In the simulation (described next) we will use the simulator $\mathsf{Sim}_{\mathsf{in}}$ as a sub-procedure.

The Simulator. We are now ready to describe the simulator Sim for $\mathcal{V}^*$. Sim will employ the simulators $\mathsf{Sim}_C, \mathsf{Sim}_{\mathsf{LDE}}$ defined above. Sim also uses the black-box simulators $\mathsf{Sim}_{\mathsf{in}}, \mathsf{Sim}'_{\mathsf{in}}$ (in the setting of auxiliary inputs) for $(\mathcal{P}_{\mathsf{in}}, \mathcal{V}_{\mathsf{in}})$ and $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$, respectively, whose existence is guaranteed by Lemma 6.4 (see above). More specifically, Sim will use 3 instantiations of $\mathsf{Sim}'_{\mathsf{in}}$ for the three ZK-IOP executions of Step 6b. To differentiate between them, we denote the ith execution of $\mathsf{Sim}'_{\mathsf{in}}$ in Step 6b by Sim_i.

Recall that $\mathcal{V}^*$'s view consists of her randomness rand, the restriction v received from $\mathcal{P}$ (Step 3c in Figure 6), the direct messages she received from $\mathcal{P}$ during the sumcheck IOP for c_1 (Step 5a in Figure 7), the codeword symbols $(\hat{w}_1(\mathbf{i}_j))_{1\le j\le 3}$ received from $\mathcal{P}$ (Step 6a in Figure 7), the direct messages she received from $\mathcal{P}$ during the sumcheck ZK-IOP executions for $(\hat{w}_1(\mathbf{i}_j))_{1\le j\le 3}$ (Step 6b in Figure 7), and the oracle answers to her (at most t) queries to c_2 and the oracle messages which $\mathcal{P}$ sent to her in Steps 5a and 6b in Figure 7.

Sim, on input a 3-CNF formula φ, and given oracle access to $\mathcal{V}^*(\cdot; \mathsf{rand})$ (where rand is the randomness of $\mathcal{V}^*$) operates as follows:

1. Generates randomness rand_C for Sim_C. Throughout the simulation, rand_C is used to emulate Sim_C.
2. **Answering queries to c_2.** Uses Sim_C (with input $1^{k_2^d}, \bar{k}_1$) to answer $\mathcal{V}^*$'s queries to c_2 up to (excluding) Step 7 of the simulation.
3. **Simulating v.** Obtains from $\mathcal{V}^*$ a set $I \subseteq [n_2]^d$ of size q, simulates $c_2|_I$ using Sim_C, via the method described above, and sends the outcome to $\mathcal{V}^*$.
4. Obtains from $\mathcal{V}^*$ an index Q of a zero-tester polynomial. This determines a codeword c_1 (though it is not known to Sim).
5. **Emulating $\mathcal{V}^*$'s view in the ZK-IOP of Step 5a.** Runs $\mathsf{Sim}_{\mathsf{in}}$ to emulate $\mathcal{V}^*$'s view in the sumcheck ZK-IOP for c_1 (Step 5a in Figure 7). More specifically, Sim initializes $\mathsf{Sim}_{\mathsf{in}}$ with

input $1^{(3k_1)^{3d+3}}$, $\alpha = 0$, and $\lambda_1, \ldots, \lambda_{(d+1)/3}$ as specified in Step 5a, where $\mathsf{Sim}_{\mathsf{in}}$ interacts with a verifier whose auxiliary input is rand_C. (We note that while we do not explicitly define this verifier, this is not a problem because $\mathsf{Sim}_{\mathsf{in}}$ is black-box.) During its emulation, $\mathsf{Sim}_{\mathsf{in}}$ interacts with its verifier in a black-box manner. Sim emulates this interaction by relaying messages between $\mathsf{Sim}_{\mathsf{in}}$ and $\mathcal{V}^*$. Throughout the emulation of $\mathsf{Sim}_{\mathsf{in}}$, its oracle queries to c_1 are answered by emulating $\mathsf{Sim}_{\mathsf{LDE}}$, using the method described above. (Recall that $\mathsf{Sim}_{\mathsf{in}}$ is adaptive subspace restricted.) We stress that during this step, $\mathcal{V}^*$ might continue sending queries to c_2, which are answered as described in Step 2.

6. **Emulating $\hat{w}_1(\mathbf{i}_j)$, $j = 1, 2, 3$.** At some point, Sim obtains $(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) \in (\mathbb{F} \setminus [3k_1])^{3d+3}$ and α' from $\mathcal{V}^*$. $\mathcal{V}^*$ now expects to obtain from $\mathcal{P}$ the values $\hat{w}_1(\mathbf{i}_1), \hat{w}_1(\mathbf{i}_2)$ and $\hat{w}_1(\mathbf{i}_3)$. Sim uses $\mathsf{Sim}_{\mathsf{LDE}}$ to simulate the corresponding subspaces. (That is, the emulation of $\mathsf{Sim}_{\mathsf{in}}$ in Step 5 determined a subspace oracle in some direction J for c_1; for every $1 \leq j \leq 3$, Sim now queries its oracle on the subspace in direction J that contains the point $\mathbf{i}_j$.) Then Sim computes simulated values $\hat{w}'_1(\mathbf{i}_1), \hat{w}'_1(\mathbf{i}_2), \hat{w}'_1(\mathbf{i}_3)$ from the simulated subspaces, and gives these values to $\mathcal{V}^*$ as the prover's messages.

7. **Simulating $\mathcal{V}^*$'s view in the ZK-IOPs of Step 6b.** Simulates $\mathcal{V}^*$'s view in the 3 sequential calls to the sumcheck ZK-IOP in Step 6b of Figure 7, by sequentially performing the following for $j = 1, 2, 3$.[32]

   Sim emulates Sim_j with input $1^{k_2^d}$, $\hat{w}'_1(\mathbf{i}_j)$, and $\lambda'_{j,1}, \ldots, \lambda'_{j,d}$ (as specified in Step 6b), where Sim_j interacts with a verifier whose auxiliary input consists of $\mathcal{V}^*$'s view so far in the interaction, except for rand. (This view consists of $v, \hat{w}(\mathbf{i}_1), \hat{w}(\mathbf{i}_2), \hat{w}(\mathbf{i}_3)$, the answers to $\mathcal{V}^*$'s queries to c_2, and the simulated direct messages – and answers to $\mathcal{V}^*$'s queries to the simulated oracle messages – generated in the emulations of $\mathsf{Sim}_{\mathsf{in}}$ and $(\mathsf{Sim}_{j'})_{j'<j}$.)[33] The oracle queries of Sim_j are answered using the simulator Sim_C as in Step 2 of the simulation.

   We stress that during this step, Sim emulates up to 4 different simulators. More specifically, throughout the emulation of Sim_j, Sim relays messages between Sim_j and $\mathcal{V}^*$ (to implement the oracle access which Sim_j has to its verifier). However, queries of $\mathcal{V}^*$ to oracles sent *before* the sumcheck ZK-IOP for $\hat{w}_1(\mathbf{i}_j)$ are answered using the appropriate simulator, instead of being forwarded to Sim_j. For example, if during the emulation of Sim_1 the verifier $\mathcal{V}^*$ queries an oracle sent in Step 5a, then the answer is simulated using $\mathsf{Sim}_{\mathsf{in}}$.

   We also need to explain how $\mathcal{V}^*$'s queries to c_2 are answered in this step. For every $1 \leq j \leq 3$, once the emulation of Sim_j has been initialized, Sim_j is used to answer the queries of $\mathcal{V}^*$ to c_2. That is, once Sim_1 has been initialized, $\mathcal{V}^*$'s queries to c_2 are forwarded to Sim_1, and his answers are provided to $\mathcal{V}^*$ as the oracle's answer; Once Sim_2 has been initialized, $\mathcal{V}^*$'s queries to c_2 are sent to Sim_2 (instead of Sim_1); and once Sim_3 has been initialized, all following queries to c_2 are sent to it.

We now prove that for every $\varphi \in 3-\mathsf{SAT}$ with a satisfying assignment w, the real-world view in an execution of the IOP of Figures 5-7 on φ, w, and the simulated view obtained when $\mathcal{V}^*$ interacts with Sim, are identically distributed. We do so through a sequence of hybrids. Since the randomness rand of the verifier $\mathcal{V}^*$ is identically distributed in the real world and the simulation,

[32] By "sequentially", we mean that Sim initializes Sim_j only when the interactive phase in the emulation of Sim_{j-1} has ended (i.e., after Sim_{j-1} sent the last prover message to the verifier). However, as will become clear below, the emulations of the Sim_j's are actually interleaved.

[33] Notice that we do not include rand as part of the auxiliary input. It is potentially too long to include in z, which should be polynomially bounded.

we condition all hybrids on rand. To simplify the presentation, we refer to the 4 internal ZK-IOPs used in Steps 5a and 6b as ZK-IOP', ZK-IOP-1, ZK-IOP-2 and ZK-IOP-3. We are now ready to define the hybrids.

$\mathcal{H}_0$**:** this is the real-world view of $\mathcal{V}^*$ (conditioned on rand).

$\mathcal{H}_1$**:** in $\mathcal{H}_1$, we replace the real-world view of $\mathcal{V}^*$ during the execution of ZK-IOP-3 (i.e., the 3rd sumcheck ZK-IOP of Step 6b, executed for $\hat{w}_1(\mathbf{i}_3)$) with the simulated view generated by Sim_3. More specifically, $\mathcal{H}_1$ is generated by honestly executing the ZK-IOP of Figures 5-7 with $\mathcal{V}^*$ up to the execution of ZK-IOP-3, and then simulating the interaction in ZK-IOP-3 by letting $\mathcal{V}^*$ interact with Sim_3 (instead of $\mathcal{P}'_{\mathsf{in}}$), and having Sim_3 answer $\mathcal{V}^*$'s oracle queries to c_2. In this execution, the input to Sim_3 is $1^{k_2^d}$, $\hat{w}_1(\mathbf{i}_3)$ and $\lambda'_{3,1},\ldots,\lambda'_{3,d}$, and Sim_3 has oracle access to a row oracle for $c_2 \in \widetilde{C}_2 \otimes \widetilde{C}_2^{\otimes(d-1)}$. We stress that oracle queries which $\mathcal{V}^*$ makes to oracle messages sent by the prover *before* ZK-IOP-3 are answered according to the actual oracles that were already generated by $\mathcal{P}$ in this process. We also note that though we have not explicitly defined the verifier for the internal ZK-IOP system $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$ with which Sim_3 is executed, this is not a problem because Sim_3 is black-box.

Claim: *$\mathcal{H}_1 \equiv \mathcal{H}_0$ by the ZK with auxiliary inputs of the sumcheck ZK-IOP (Theorem 5.4 and Lemma 6.4).*

Proof: Assume towards contradiction that the claim does not hold. Then there exists a choice of randomness rand_P for the honest prover up to the execution of ZK-IOP-3, such that $\mathcal{H}_1 \not\equiv \mathcal{H}_0$, conditioned on the prover using randomness rand_P. We will use this to define a verifier $\mathcal{V}_3$ in the execution of ZK-IOP-3, for which Sim_3's simulated answers do not induce a perfect emulation of $\mathcal{V}_3$'s view.

The verifier $\mathcal{V}_3$ has input $1^{k_2^d}$, α_3 and $\lambda'_{3,1},\ldots,\lambda'_{3,d}$, and oracle access to c_2. In addition, $\mathcal{V}_3$ has an auxiliary input z, which consists of $\mathcal{V}^*$'s view at the onset of the execution of ZK-IOP-3, except for $\mathcal{V}^*$'s randomness. (That is, the auxiliary input consists of the explicit messages $\mathcal{V}^*$ received from the prover until the onset of ZK-IOP-3, as well as the answers to $\mathcal{V}^*$'s oracle queries to c_2 and the oracle messages which the prover sent until then.[34]) $\mathcal{V}_3$ uses its auxiliary input to emulate the interaction between the honest prover $\mathcal{P}$ and the verifier $\mathcal{V}^*$ in the ZK-IOP of Figures 5-7, up to (excluding) ZK-IOP-3 (in particular, $\mathcal{V}_3$ uses part of its own random string for this emulation). We stress that during this emulation, any oracle queries of $\mathcal{V}^*$ to c_2 are answered according to $\mathcal{V}_3$'s auxiliary input. When the emulation reaches ZK-IOP-3 (in Step 6b), $\mathcal{V}_3$ continues emulating $\mathcal{V}^*$, but forwards $\mathcal{V}^*$'s messages to the prover $\mathcal{P}$ to $\mathcal{V}_3$'s own prover $\mathcal{P}'_{\mathsf{in}}$, and forwards $\mathcal{V}^*$'s oracle queries to c_2 to $\mathcal{V}_3$'s own oracle. Notice that once the execution of ZK-IOP-3 begins, the verifier $\mathcal{V}^*$ does not send any further messages to the provers in the previous ZK-IOP executions. However, $\mathcal{V}^*$ can still query the oracle messages sent by the provers in these previous executions. $\mathcal{V}_3$ answers these queries at random. Let t_3 denote the number of queries which $\mathcal{V}_3$ makes to its oracles, and notice that $t_3 \leq t$. The ZK property of $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$ guarantees that in the execution with $\mathcal{V}_3$, Sim_3 is (t_3+q+1)-row restricted, and the simulation is perfect.

Notice that when z is $\mathcal{V}^*$'s view up to (excluding) the execution of ZK-IOP-3, then the only differences between the real-world view of $\mathcal{V}_3$ and $\mathcal{H}_0$ are that: (1) $\mathcal{V}_3$'s view contains $\mathcal{V}_3$'s randomness instead of $\mathcal{V}^*$'s randomness (and the randomness used to emulate $\mathcal{V}^*$ might not

[34] The explicit messages include $v, \alpha_1, \alpha_2, \alpha_3$, and messages sent as part of the executions of the ZK-IOP in Step 5a, and the ZK-IOPs for $\hat{w}_1(\mathbf{i}_1)$ and $\hat{w}_1(\mathbf{i}_2)$ in Step 6b. The oracle queries are to oracles sent by the prover in these sub-executions, as well as to c_2.

be rand); and (2) $\mathcal{H}_0$ also contains the answers to queries which $\mathcal{V}^*$ makes – *during* the execution of ZK-IOP-3 – to oracles sent *before* the onset of ZK-IOP-3. However, the answers which $\mathcal{V}_3$ provides to these queries are determined by its randomness, and can therefore be computed from $\mathcal{V}_3$'s view (since $\mathcal{V}_3$'s view contains its randomness). Moreover, the portion rand' of $\mathcal{V}_3$'s randomness which is used to emulate $\mathcal{V}^*$ can also be computed from $\mathcal{V}_3$'s view. Let g_3 denote the function that on input a view, computes from it the answers to the aforementioned queries, and the portion rand' of randomness used to emulate $\mathcal{V}^*$, concatenates these values to the view, and erases the randomness reported in the view. (We stress that g_3 erases the randomness of $\mathcal{V}_3$, not rand'.)

To reach a contradiction, it suffices to show that there exist a c_2, an auxiliary input z, and a choice of randomness rand_3 for $\mathcal{V}_3$, such that when $\mathcal{V}_3$ has input $1^{k_2^d}$, α_3 and $\lambda'_{3,1},\ldots,\lambda'_{3,d}$, implicit input c_2, auxiliary input z, and uses randomness rand_3, the simulation of Sim_3 is not perfect. Since we have conditioned on the randomness rand of $\mathcal{V}^*$, every randomness of the prover $\mathcal{P}$ fully defines the emulation of $\mathcal{V}^*$ up to the execution of ZK-IOP-3. We set z to be the view of $\mathcal{V}^*$ in this emulation, when the prover's randomness is consistent with rand_P. Finally, we choose the randomness rand_3 at random subject to: (1) $\mathsf{rand}' = \mathsf{rand}$, and (2) the random oracle answers which $\mathcal{V}_3$ provides to $\mathcal{V}^*$ once the execution of ZK-IOP-3 begins (i.e., answers to queries to oracles provided in previous stages of the emulation) are consistent with the oracles the prover generates when using randomness rand_P. Notice that in this case, applying g_3 to $\mathcal{V}_3$'s view gives the hybrid $\mathcal{H}_0$ (when conditioned on rand_P), whereas applying g_3 on the simulated view gives $\mathcal{H}_1$ (conditioned on rand_P). Since $\mathcal{H}_0, \mathcal{H}_1$ are not identically distributed (when conditioned on rand_P, by the negation assumption), this implies that the real and simulated views are not identically distributed in this case, which contradicts the ZK with auxiliary inputs of $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$ against t-restricted adversaries.

$\mathcal{H}_2$: in $\mathcal{H}_2$, we replace the real-world view of $\mathcal{V}^*$ during the execution of ZK-IOP-2 with the simulated view generated by Sim_2. In particular, once the execution of ZK-IOP-2 begins then Sim_2 is used to answer all of $\mathcal{V}^*$'s oracle queries until the execution of ZK-IOP-3 begins. Then, as in $\mathcal{H}_1$, the view of $\mathcal{V}^*$ during ZK-IOP-3 is generated using Sim_3. In the emulation of Sim_2, its input is $1^{k_2^d}$, $\hat{w}_1(\mathbf{i}_2)$ and $\lambda'_{2,1},\ldots,\lambda'_{2,d}$, and Sim_2 has oracle access to a row oracle for $c_2 \in \widetilde{C}_2 \otimes \widetilde{C}_2^{\otimes(d-1)}$.

Claim: *$\mathcal{H}_1 \equiv \mathcal{H}_2$ by the ZK with auxiliary inputs of the sumcheck ZK-IOP against adversaries with row oracles (Lemmas 5.9 and 6.4.)*

Proof: The proof is similar to the proof that $\mathcal{H}_1 \equiv \mathcal{H}_0$, but we need to additionally account for how the output of Sim_3 is generated from $\mathcal{V}^*$'s (real or simulated) view in ZK-IOP-2. The formal proof follows.

Assume towards contradiction that the claim does not hold. Then there exists a choice of randomness rand_P for the honest prover up to the execution of ZK-IOP-2, such that $\mathcal{H}_2 \not\equiv \mathcal{H}_1$, conditioned on the prover using randomness rand_P. (Notice that rand_P, in particular, also fixes c_2.) We use this to define a verifier $\mathcal{V}_2$ in the execution of ZK-IOP-2, for which Sim_2's simulated answers do not induce a perfect emulation of $\mathcal{V}_2$'s view. $\mathcal{V}_2$ has input $1^{k_2^d}$, α_2 and $\lambda'_{2,1},\ldots,\lambda'_{2,d}$, auxiliary input z which consists of $\mathcal{V}^*$'s view at the onset of the execution of ZK-IOP-2 (except for $\mathcal{V}^*$'s randomness), and has oracle access to c_2 and to a row oracle for c_2 (when c_2 is interpreted as a codeword of $\widetilde{C}_2 \otimes \widetilde{C}_2^{\otimes(d-1)}$). $\mathcal{V}_2$ uses its auxiliary input to emulate $\mathcal{V}^*$ up to the execution of ZK-IOP-2 (similar to how $\mathcal{V}_3$ emulates $\mathcal{V}^*$). Then, $\mathcal{V}_2$ emulates $\mathcal{V}^*$ in ZK-IOP-2 by interacting with its own oracle c_2 and prover $\mathcal{P}'_{\mathsf{in}}$. Once the execution of

ZK-IOP-2 terminates, namely once the last message is sent between the parties of ZK-IOP-2, $\mathcal{V}_2$ uses Sim_3 to continue the emulation of $\mathcal{V}^*$. Specifically, $\mathcal{V}_2$ relays messages in ZK-IOP-3 between $\mathcal{V}^*$ and Sim_3 (this includes forwarding any c_2-queries of $\mathcal{V}^*$ to Sim_3). Queries of $\mathcal{V}^*$ to oracle messages sent in ZK-IOP-2 are forwarded to $\mathcal{V}_2$'s own oracles, and queries of $\mathcal{V}^*$ to prover oracle messages sent before ZK-IOP-2 are answered at random. Oracle queries of Sim_3 are answered using $\mathcal{V}_2$'s own row oracle.

Let t_2 denote the number of queries which $\mathcal{V}_2$ makes to its oracles (except the row oracle for c_2), and notice that $\mathcal{V}_2$ makes $t_3 + q + 1$ additional queries to its row oracle during the emulation of Sim_3. The ZK property of $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$ guarantees that in the execution with $\mathcal{V}_2$, Sim_2 is $(t_3 + t_2 + 2q + 2)$-row restricted, and the simulation is perfect.

Notice that when the auxiliary input of Sim_2 consists of $\mathcal{V}^*$'s view up to the execution of ZK-IOP-2 (except for rand), then there are three difference between $\mathcal{V}_2$'s view and $\mathcal{H}_1$. The first two are similar to the differences between the $\mathcal{V}_3$'s view and $\mathcal{H}_0$, namely Sim_2's view contains $\mathcal{V}_2$'s randomness instead of the portion rand'' used to emulate $\mathcal{V}^*$ (where possibly $\mathsf{rand}'' \neq \mathsf{rand}$), and $\mathcal{H}_1$ contains also the answers to queries which $\mathcal{V}^*$ made during ZK-IOP-2 and ZK-IOP-3 to oracles sent previously in its emulation. The third difference is that $\mathcal{H}_1$ contains also $\mathcal{V}^*$'s view in ZK-IOP-3, whereas $\mathcal{V}_2$'s view contains only the answers to row-queries which $\mathcal{V}_2$ made during Sim_3's emulation. (Everything else generated during this emulation was performed internally by $\mathcal{V}_2$ with no externally sent messages/oracle queries, and is therefore not included in the view.) However, the simulated view of $\mathcal{V}^*$ in ZK-IOP-3 can be generated from $\mathcal{V}_2$'s view. Indeed, we can extract from $\mathcal{V}_2$'s randomness (which is included in its view) the randomness used to emulate Sim_3. We can then emulate Sim_3 with this randomness, and answer Sim_3's row queries using the answers which are included in $\mathcal{V}_2$'s view. Let g_2 be the function that given a view first generates the simulated view of $\mathcal{V}^*$ in ZK-IOP-3 as described above, erases the answers to $\mathcal{V}_2$'s row queries, and then applies a function that operates similarly to g_3 (defined in the proof that $\mathcal{H}_1 \equiv \mathcal{H}_0$).[35]

We now show that there exist c_2, z, and rand_2 such that when $\mathcal{V}_2$ has input $1^{k_2^d}$, α_2 and $\lambda'_{2,1}, \ldots, \lambda'_{2,d}$, implicit input c_2, auxiliary input z, and uses randomness rand_2, the simulation of Sim_2 is not perfect. This will contradict the ZK with auxiliary inputs of $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$ against t-restricted adversaries. We set z to be the view of $\mathcal{V}^*$ in this emulation, when the prover's randomness is rand_P (this also determines c_2). Furthermore, we choose the randomness rand_2 at random subject to: (1) $\mathsf{rand}'' = \mathsf{rand}$, and (2) the random oracle answers which $\mathcal{V}_2$ provides to $\mathcal{V}^*$ once the execution of ZK-IOP-2 begins (i.e., answers to queries to oracles provided in previous stages of the emulation) are consistent with the oracles the prover generates when using randomness rand_P. Notice that in this case, applying g_2 to $\mathcal{V}_2$'s view (the simulated view generated when interacting with Sim_2, respectively) gives the hybrid $\mathcal{H}_1$ ($\mathcal{H}_2$, respectively) when conditioned on rand_P. Since by the negation assumption $\mathcal{H}_2 \not\equiv \mathcal{H}_1$ when conditioned on rand_P, we have reached a contradiction.

$\mathcal{H}_3$: in $\mathcal{H}_3$, we replace the real-world view of $\mathcal{V}^*$ during the execution of ZK-IOP-1 with the simulated view generated by Sim_1. This is done similarly to $\mathcal{H}_2$, namely: (1) once the execution of ZK-IOP-1 begins then Sim_1 is used to answer all of $\mathcal{V}^*$'s oracle queries to c_2 until the execution of ZK-IOP-2 begins; (2) the views of $\mathcal{V}^*$ during ZK-IOP-2 and ZK-IOP-3 are generated using Sim_2 and Sim_3, respectively; and (3) in the emulation of Sim_1, its input is $1^{k_2^d}$, $\hat{w}_1(\mathbf{i}_1)$ and $\lambda'_{1,1}, \ldots, \lambda'_{1,d}$, and it has oracle access to a row oracle for $c_2 \in \widetilde{C}_2 \otimes \widetilde{C}_2^{\otimes(d-1)}$.

[35] The difference from g_3 is that we need to keep also the portion of $\mathcal{V}_2$'s randomness used to emulate $\mathcal{V}^*$ in ZK-IOP-2, and the answers to $\mathcal{V}^*$'s queries – during ZK-IOP-2 – to previously-generated prover oracle messages.

Claim: $\mathcal{H}_2 \equiv \mathcal{H}_3$ *by the ZK with auxiliary inputs of the sumcheck ZK-IOP against adversaries with row oracles (Lemmas 5.9 and 6.4.)*

Proof (sketch): The proof is similar to the proof that $\mathcal{H}_1 \equiv \mathcal{H}_2$ above, where we define a verifier $\mathcal{V}_1$ in the execution of ZK-IOP-1 for which Sim_1's simulation is not perfect. We therefore only survey the differences compared to the proof that $\mathcal{H}_1 \equiv \mathcal{H}_2$. First, $\mathcal{V}_1$'s input is $1^{k_2^d}$, α_1 and $\lambda'_{1,1}, \ldots, \lambda'_{1,d}$, and its auxiliary input does not contain $\mathcal{V}^*$'s view in ZK-IOP-1. Second, $\mathcal{V}_1$ emulates both Sim_2 and Sim_3, forwarding their row-queries to its own row oracle.[36] Third, if t_1 denotes the number of queries which $\mathcal{V}_1$ makes to its oracles (except the row oracle for c_2), then it additionally makes at most $(t_3 + q + 1) + (t_2 + q + 1) = t_3 + t_2 + 2q + 2$ row queries to emulate Sim_2 and Sim_3. The ZK property of $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$ guarantees that Sim_1 makes at most $t_1 + t_2 + t_3 + 3q + 3 \leq t + 3q + 3$ row oracle queries, and the simulation is perfect. Finally, $\mathcal{H}_2, \mathcal{H}_3$ can be generated from the real or simulated views (respectively) by applying a function g_1 that first emulates Sim_2 (with the appropriate part of $\mathcal{V}_1$'s randomness) to generate the simulated view of $\mathcal{V}^*$ during ZK-IOP-2, then applies a function which operates similarly to g_2. (The difference from g_2 is that we need to keep also the portion of $\mathcal{V}_1$'s randomness used to emulate $\mathcal{V}^*$ in ZK-IOP-1, and the answers to $\mathcal{V}^*$'s queries – during ZK-IOP-1 – to previously-generated prover oracle messages.)

$\mathcal{H}_4$**:** in $\mathcal{H}_4$, we replace the oracle answers to (point queries of $\mathcal{V}^*$, and row queries of $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ to) c_2 with simulated codeword symbols generated by Sim_C.

Claim: $\mathcal{H}_4 \equiv \mathcal{H}_3$ *by the black-box* $(t + 4q + 3)$*-ZK of* C_2.

Proof: Assume towards contradiction that the claim does not hold, then there exist φ, w for which the hybrids $\mathcal{H}_3, \mathcal{H}_4$ are not identically distributed. By an averaging argument, there exists a w_1 (consistent with w) such that $\mathcal{H}_3, \mathcal{H}_4$ are not identically distributed, even conditioned on w_1. We construct a (non uniform) $(t + 4q + 3)$-row restricted adversary $\mathcal{A}$ that distinguishes between the real and simulated oracle answers of a row oracle for $c_2 \in \widetilde{C}_2 \otimes \widetilde{C}_2^{\otimes d-1}$. Recall that using the alternative view of w_1, w_2 given in Lemma 6.5, c_2 can be viewed as a random encoding of $\bar{w}_2 \in \{0,1\}^*$ (which is fully determined by w_1). Therefore, by Remark 6.6, the existence of $\mathcal{A}$ contradicts the black-box $(t + 4q + 3)$-ZK of C_2.

$\mathcal{A}$ has φ, w_1 and rand hard-wired into it, as well as the code of the honest prover $\mathcal{P}$. $\mathcal{A}$ also has black-box access to $\mathcal{V}^*$.[37] $\mathcal{A}$ provides rand as the randomness of $\mathcal{V}^*$, and interacts with $\mathcal{V}^*$, by emulating the honest prover $\mathcal{P}$ on input φ, w up to (excluding) Step 6b in Figure 7. (This can be done because w_1 fully defines w.) In particular, during this emulation $\mathcal{V}^*$ provides $\mathcal{A}$ with a query set $I \subseteq [n_2]^d$, and $\mathcal{A}$ then generates $v = c_1|_I$ by querying its oracle on I (more accurately, for every $i \in I$, $\mathcal{A}$ queries the row of c_2 to which i belongs). $\mathcal{A}$ answers $\mathcal{V}^*$'s queries to c_2 by querying its row oracle (which answers according to an actual codeword, or with simulated answers), and then computing the corresponding entry of c_2 (similar to how Sim does this in Step 2 of the simulation). Once the interaction with $\mathcal{V}^*$ reaches Step 6b, $\mathcal{A}$ generates the prover messages during the following steps of the execution by emulating $\mathsf{Sim}_1, \mathsf{Sim}_2$ and Sim_3 in the same way that Sim does so in Step 7, except that oracle queries of $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ are forwarded to $\mathcal{A}$'s own oracle. (Recall that these queries are to rows of

[36] The interleaving of these simulators is as specified in the description of Sim. In particular, Sim_2 is used *only* to emulate $\mathcal{V}^*$'s view in ZK-IOP-2, namely – unlike $\mathcal{V}_2$ defined in the proof of $\mathcal{H}_2 \equiv \mathcal{H}_1$ – the emulated verifier of Sim_2 does *not* internally emulate Sim_3.

[37] This can be thought of as having $\mathcal{V}^*$ hard-wired into $\mathcal{A}$. However, $\mathcal{A}$ does not use the code of $\mathcal{V}^*$, only its input-output behavior.

$c_2 \in \widetilde{C}_2 \otimes \widetilde{C}_2^{\otimes(d-1)}$.[38]) We note that once the emulation reaches Step 6b, oracle queries of $\mathcal{V}^*$ to c_2 are answered using the appropriate simulator (according to the description in Step 7 of the simulation.) When $\mathcal{V}^*$ terminates, $\mathcal{A}$ outputs φ, rand, all messages which (the emulated) $\mathcal{P}$ and $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ sent to $\mathcal{V}^*$ in the emulation, and the answers to $\mathcal{V}^*$'s queries to c_2 and the oracle messages of $\mathcal{P}, \mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$. Notice that $\mathcal{A}$ is well defined, because v is determined by $c_2|_I$, and all other messages sent by the prover $\mathcal{P}$ up to (excluding) Step 6b can be computed from φ, w_1 alone, and are otherwise independent of c_2.

Notice that $\mathcal{A}$ is $(t+3q+3)$-restricted. Indeed, let t'' denotes the number of queries which $\mathcal{V}^*$ makes to c_2 before Step 6b, and notice that since $\mathcal{V}^*$ is t-restricted then $t''+t_1+t_2+t_3 \leq t$. $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ make *in total* $(t_1+t_2+t_3+3q+3)$-row queries, so $\mathcal{A}$ makes a total of at most $t''+t_1+t_2+t_3+3q+3+q \leq t+4q+3$ oracle queries to its row oracle. (That is, the number of queries of $\mathcal{A}$ is q more than the *total* number of queries which $\mathcal{V}^*$ and $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ make; this is because $\mathcal{A}$ also queries $c_2|_I$.) Moreover, if $\mathcal{A}$ is given oracle access to an actual codeword then its output is distributed identically to $\mathcal{H}_3$ (conditioned on w_1), otherwise its output is distributed according to $\mathcal{H}_4$ (conditioned on w_1). Since these distributions are not identical, the inputs of $\mathcal{A}$ – namely, the answers to $\mathcal{A}$'s oracle queries – cannot be identically distributed. This contradicts the black-box $(t+4q+3)$-ZK of C_2.

$\mathcal{H}_5$: in $\mathcal{H}_5$, we replace the real-world view of $\mathcal{V}^*$ during the execution of ZK-IOP′ with the simulated view generated by $\mathsf{Sim}_{\mathsf{in}}$. More specifically, $\mathsf{Sim}_{\mathsf{in}}$ has input $1^{[3k_1]^{3d+3}}$, $\alpha = 0$ and $\lambda_1, \ldots, \lambda_{(d+1)/3}$ as specified in Step 5a in Figure 7, and it is adaptive subspace-restricted (for $c_1 \in B^{\otimes(3\cdot(d+1)/3)}$, where $B = \mathsf{PRS}^{\otimes 3}_{3k_1,n_1}$). This emulation is interleaved with the emulations of $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ as specified in the description of Sim. (That is, once the simulation of ZK-IOP′ terminates, namely $\mathcal{V}^*$ provides α' and $(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3)$, then we initialize an emulation of Sim_1 and so on.)

Claim: *$\mathcal{H}_5 \equiv \mathcal{H}_4$ by the ZK with auxiliary inputs of the sumcheck ZK-IOP (Theorem 5.4 and Lemma 6.4).*

Proof: The proof is similar to the proof that $\mathcal{H}_0 \equiv \mathcal{H}_1$, where we also need to account for how the outputs of $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ are generated from the (real or simulated) view of $\mathcal{V}^*$ in ZK-IOP′.

It suffices to prove the claim conditioned on the random string rand_P of the prover $\mathcal{P}$ up to the execution of ZK-IOP′, and the random string rand_C used to emulate Sim_C. We use the latter to define a verifier $\mathcal{V}'$ in the execution of ZK-IOP′, whose view $\mathsf{Sim}_{\mathsf{in}}$ fails to perfectly simulate.

$\mathcal{V}'$ has input $1^{[3k_1]^{3d+3}}$, $\alpha = 0$ and $\lambda_1, \ldots, \lambda_{(d+1)/3}$, and its auxiliary input is rand_C. In addition, $\mathcal{V}'$ has oracle access to c_1 (though it does not use it). $\mathcal{V}'$ emulates $\mathcal{V}^*$ in the following way. $\mathcal{V}'$ uses part of its own random string to emulate the randomness of $\mathcal{V}^*$, and interprets another part of it as an LDE $\hat{w}'_1$. $\mathcal{V}'$ uses Sim_C, with rand_C as the randomness for Sim_C, to answer $\mathcal{V}^*$'s queries to c_2, similar to Step 2 in the simulation. When $\mathcal{V}^*$ sends a query set I, $\mathcal{V}'$ generates v as in Step 3 of the simulation. In the execution of ZK-IOP′, $\mathcal{V}'$ relays $\mathcal{V}^*$'s queries to $\mathcal{V}''$s own prover. When $\mathcal{V}^*$ provides α' and $(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3)$, $\mathcal{V}'$ sends $\hat{w}'_1(\mathbf{i}_1), \hat{w}'_1(\mathbf{i}_2), \hat{w}'_1(\mathbf{i}_3)$ to $\mathcal{V}^*$ as the messages which $\mathcal{P}$ sent in Step 6a in Figure 7. Then, $\mathcal{V}'$ continues the emulation of $\mathcal{V}^*$ by emulating $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ as in Step 7 in the simulation. In particular, any queries which $\mathcal{V}^*$

[38] We note that $\mathcal{A}$ answers repeated queries consistently. Specifically, $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ might query rows to which $\mathcal{V}^*$ has already made a query; in this case $\mathcal{A}$ has already obtained the row from its oracle, and uses it to answer these queries.

makes to c_2 are answered as described above, and $\mathcal{V}'$ answers any queries which $\mathcal{V}^*$ makes to prover oracle messages sent in ZK-IOP' by querying $\mathcal{V}''$'s own oracles. Row queries of $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ are answered using Sim_C.

Let t' denote the number of queries which $\mathcal{V}'$ makes to its oracles (since $\mathcal{V}'$ does not query c_1, these are all to prover oracle messages sent in ZK-IOP'). The ZK property of $(\mathcal{P}_{\mathsf{in}}, \mathcal{V}_{\mathsf{in}})$ guarantees that in the execution with $\mathcal{V}'$, $\mathsf{Sim}_{\mathsf{in}}$ is adaptive $(t'+2)$-subspace restricted, and the simulation is perfect.

Notice that when the auxiliary input rand_C of $\mathcal{V}'$ is the randomness used for Sim_C in $\mathcal{H}_4$, then the differences between $\mathcal{V}''$'s view and $\mathcal{H}_4$ are: (1) $\mathcal{V}''$'s view contains its entire randomness, as well as rand_C, whereas $\mathcal{H}_4$ only contains the randomness of $\mathcal{V}^*$; (2) $\mathcal{H}_4$ contains also the answers to $\mathcal{V}^*$'s queries to c_2, the restriction v, the values $\alpha'_1, \alpha'_2, \alpha'_3$, and $\mathcal{V}^*$'s view in ZKIOP-1, ZK-IOP-2, and ZK-IOP-3. However, the values in (2) can be computed from $\mathcal{V}''$'s view. Indeed, v and the answers to $\mathcal{V}''$'s queries to c_2 can be answered by emulating Sim_C with randomness rand_C (rand_C is $\mathcal{V}''$'s auxiliary input, and therefore part of its view). $\alpha'_1, \alpha'_2, \alpha'_3$ are determined by $\mathcal{V}''$'s randomness. $\mathcal{V}^*$'s views in ZK-IOP-1, ZK-IOP-2, and ZK-IOP-3 can be computed from $\mathcal{V}''$'s view by emulating $\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3$ (respectively) with the part of $\mathcal{V}''$'s randomness used for these emulations (these emulations include interaction with $\mathcal{V}^*$, which can be emulated using the appropriate randomness from $\mathcal{V}''$'s view), and answering their oracle queries using Sim_C (with randomness rand_C). Let g' denote the function that on input a view generates these values, and additionally erases the other parts of $\mathcal{V}''$'s randomness. Notice that when g' is applied to the simulated view (obtained when interacting with $\mathsf{Sim}_{\mathsf{in}}$), concatenated with $\mathcal{V}''$'s randomness and rand_C, then we get $\mathcal{H}_5$.

To reach a contradiction, it suffices to show that there exist a c_1, and a choice of randomness rand' for $\mathcal{V}'$, such that when $\mathcal{V}'$ has input $1^{[3k_1]^{3d+3}}$, $\alpha = 0$ and $\lambda_1, \ldots, \lambda_{(d+1)/3}$, implicit input c_1, auxiliary input rand_C, and uses randomness rand', the simulation of $\mathsf{Sim}_{\mathsf{in}}$ is not perfect. (We note that during its simulation, $\mathsf{Sim}_{\mathsf{in}}$ might query rows of c_1; these are answered using the actual codeword c_1, which is fully determined by $\hat{w}_1$.) Since we have conditioned on the randomness rand_P of $\mathcal{P}$, this fixes c_1 and $\hat{w}_1$. We set the randomness of $\mathcal{V}'$ such that the portion which $\mathcal{V}'$ uses to emulate $\mathcal{V}^*$ is equal to rand, and the portion which $\mathcal{V}'$ interprets as an LDE $\hat{w}'_1$ is equal to $\hat{w}_1$. Notice that in this case, applying g' to $\mathcal{V}''$'s view gives the hybrid $\mathcal{H}_4$ (when conditioned on $\mathsf{rand}_P, \mathsf{rand}_C$), whereas applying g' on the simulated view – prepended with $\mathsf{rand}', \mathsf{rand}_C$ – gives $\mathcal{H}_5$ (conditioned on $\mathsf{rand}_P, \mathsf{rand}_C$). If $\mathcal{H}_4, \mathcal{H}_5$ are not identically distributed (when conditioned on $\mathsf{rand}_P, \mathsf{rand}_C$), then the real and simulated views are not identically distributed, which contradicts the ZK with auxiliary inputs of $(\mathcal{P}_{\mathsf{in}}, \mathcal{V}_{\mathsf{in}})$ against t'-restricted adversaries.

$\mathcal{H}_6$**:** in $\mathcal{H}_6$, we replace $(\hat{w}_1(\mathbf{i}_j))_{1 \le j \le 3}$ and the answers to $\mathsf{Sim}_{\mathsf{in}}$'s (at most $t+2$) subspace queries, with the simulated codeword symbols $(\hat{w}'_1(\mathbf{i}_j))_{1 \le j \le 3}$, and the simulated answers (computed from simulated subspaces of $\hat{w}_1$), generated by $\mathsf{Sim}_{\mathsf{LDE}}$ in Steps 6 and 5 of the simulation (respectively).

Claim: *$\mathcal{H}_6 \equiv \mathcal{H}_5$ because the LDE encoding $\hat{w}_1$ has black-box $3(t+3)$-ZK.*

Proof: The proof is similar to the proof that $\mathcal{H}_4 \equiv \mathcal{H}_3$. $\mathcal{H}_5$ (respectively, $\mathcal{H}_6$) can be generated by making $3(t+3)$ adaptive queries to a (real or simulated) subspace oracle for $\hat{w}_1$. This can be done by executing the same process as the one used to generate $\mathcal{H}_5$, except that we answer $\mathsf{Sim}_{\mathsf{in}}$'s oracle queries to c_1 by querying a (real or simulated) subspace oracle for $\hat{w}_1$, and use the method described in the beginning of the proof (in the paragraph "answering

queries to c_1") to generate the symbols/subspaces of c_1. As part of this process, we also obtain $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3$. We then query the appropriate subspaces of $\hat{w}_1$ from the row oracle (similar to how these are generated in Step 6 of the simulation). Overall, this processes makes at most 3 subspace queries directly to $\hat{w}_1$, and $t+2$ subspace queries to c_1, which can be answered using $3(t+2)$ subspace queries to $\hat{w}_1$ (cf. Remark 6.6). Therefore, overall, we make at most $3(t+3)$ subspace queries to $\hat{w}_1$. Moreover, if the oracle is to $\hat{w}_1$ then we obtain $\mathcal{H}_5$, and if it is answered by $\mathsf{Sim}_{\mathsf{LDE}}$'s simulated answers then we obtain $\mathcal{H}_6$. Therefore, $\mathcal{H}_5 \equiv \mathcal{H}_6$ because these hybrids can be generated by applying the same (randomized) function to the (real or simulated) view of an adaptive $3(t+3)$-subspace restricted adversary, and the LDE encoding has ZK against such adversaries by Remark 6.6.

We conclude the proof by noting that $\mathcal{H}_6$ is exactly the simulated view. ■

7 Reducing query complexity

In this section, we reduce the query complexity in the IOP for SAT of Theorem 6.1 to a constant. Specifically, we prove the following theorem.

Theorem 7.1 (ZK-IOP for SAT with constant query complexity)**.** *For any constant $\beta, \gamma > 0$, there exists a constant-round pre-processing IOP $(\mathcal{P}, \mathcal{V})$ for 3-SAT on n variables of constant soundness error, communication complexity $(1+\gamma) \cdot n$, and constant query complexity, that has $n^{\beta'}$-ZK for some $\beta' > 0$.*

The verifier's running time is $n^{O(\beta)}$, after a local preprocessing step of running time $\mathsf{poly}(n)$, and the prover's running time is $\mathsf{poly}(n)$ (given a witness and a generating matrix for the ZK-LTC C_2 generated in Step 1c of Figure 5, see Remark 6.2). Moreover, the first prover's message is of length $(1+\gamma) \cdot n$, and the rest of the prover's messages are of length n^{β}.

The above theorem is an immediate consequence of Theorem 6.1 and the following *composition* lemma which shows how to reduce the query complexity of a ZK-IOP of a certain structure to a constant. This lemma is a ZK analogue of Lemma 7.3 from [RR19]. A more general ZK composition theorem was proven in [BCL22], but since we care about very small factors in the length we use here a more specialized composition method that uses special properties of our protocol.

Lemma 7.2. *Let $\ell = O(1)$, and let $(\mathcal{P}, \mathcal{V})$ be an ℓ-round pre-processing IOP with soundness error $1-\epsilon$ for a promise problem $(\mathsf{YES}, \mathsf{NO})$ with no implicit input, satisfying the following properties:*

1. *The prover's first message is a binary string of length cc_0, and the rest of the prover's messages are strings of length cc over some finite field $\mathbb{F}$.*
2. *The verifier queries a constant number of bits from the first prover's message, queries a constant number of field elements from the rest of the prover's oracles, and reads the rest of the prover's explicit messages in full.*
3. *The IOP has black-box straight-line ZK against any malicious verifier that reads at most t entries from each prover's oracle, and reads the rest of the prover's messages in full.*
4. *The verifier's running time is $T_{\mathcal{V}}(n) \leq \mathsf{poly}(n)$, after a local preprocessing step of running time $\mathsf{poly}(n)$, and the prover's running time is $\mathsf{poly}(n)$.*

Then there exists an $(\ell+1)$-round pre-processing IOP $(\mathcal{P}', \mathcal{V}')$ (over the binary alphabet) for $(\mathsf{YES}, \mathsf{NO})$ with soundness error $1 - \frac{\epsilon}{2}$, communication complexity $\mathrm{cc}_0 + \mathsf{poly}(T_{\mathcal{V}}(n))$, and constant query complexity, that has black-box straight-line $(t - O(1))$-ZK. The verifier's running time is $\mathsf{poly}(T_{\mathcal{V}}(n))$, after a

local preprocessing step of running time $\mathsf{poly}(n)$*, and the prover's running time is* $\mathsf{poly}(n)$*. Moreover, the first prover's message is a binary string of length* cc_0*, and the rest of the prover's messages are of length* $\mathsf{poly}(T_{\mathcal{V}}(n))$*.*

We prove the above lemma in two stages: We first show in Lemma 7.3 below how to reduce the query complexity to a constant, albeit over the alphabet $\mathbb{F}$, and then in Lemma 7.4 we show how to turn the alphabet to binary. The above Lemma 7.2 is an immediate consequence of these two lemmas.

Lemma 7.3. *Let* $(\mathcal{P},\mathcal{V})$ *be as in Lemma 7.2. Then there exists an* $(\ell+1)$*-round pre-processing* IOP $(\mathcal{P}',\mathcal{V}')$ *over* $\mathbb{F}$ *for* $(\mathsf{YES},\mathsf{NO})$ *with soundness error* $1-\frac{\epsilon}{2}$*, communication complexity* $\mathrm{cc}_0+\mathsf{poly}(T_{\mathcal{V}}(n))$*, and constant query complexity, that has black-box straight-line* $(t-O(1))$*-ZK. The verifier's running time is* $\mathsf{poly}(T_{\mathcal{V}}(n))$*, after a local preprocessing step of running time* $\mathsf{poly}(n)$*, and the prover's running time is* $\mathsf{poly}(n)$*. Moreover, the first prover's message is a binary string over* $\mathbb{F}$ *of length* cc_0*, and the rest of the prover's messages are strings over* $\mathbb{F}$ *of length* $\mathsf{poly}(T_{\mathcal{V}}(n))$*.*

Proof: Notice that if all prover messages are oracles, then the lemma holds with $(\mathcal{P}',\mathcal{V}')=(\mathcal{P},\mathcal{V})$. Therefore, we can assume that the interaction includes at least one explicit message, in which case $\mathcal{T}_{\mathcal{V}}(n)\geq \mathrm{cc}$ (the verifier needs to read the entire explicit message). The reminder of the proof is for this case.

We start by setting some notation. Let $\ell=O(1)$ denote the round complexity of $(\mathcal{P},\mathcal{V})$, and let $I_O\subseteq[\ell]$ denote the subset of all rounds in which $\mathcal{P}$ sends an oracle. Let $q=O(1)$ denote the total number of queries made to all of the oracles. Let E be any efficient code ensemble over $\mathbb{F}$ of constant rate $\rho>0$ and constant relative distance $\delta>0$. For an element $a\in\mathbb{F}$, and an integer $d\geq 1$, let $a^{(d)}\in\mathbb{F}^d$ denote the string which consists of the concatenation of d copies of a.

Let $\mathcal{L}_{\mathcal{V}}\subseteq\mathbb{F}^*$ denote the language consisting of all strings of the form $(x_{\mathrm{exp}},x_{\mathrm{imp}})$, where $x_{\mathrm{exp}}\in\{0,1\}^*$ (x_{exp} represents part of a view of $\mathcal{V}$, see below), $x_{\mathrm{imp}}=((a_1)^{(\mathrm{cc}/\rho)},\ldots,(a_q)^{(\mathrm{cc}/\rho)},(E(m_i))_{i\notin I_O})$ for $a_j\in\mathbb{F}$ and $m_i\in\mathbb{F}^{\mathrm{cc}}$, and $\mathcal{V}$ accepts, when given $m_i, i\notin I_O$ as the explicit prover messages, $(a_1,\ldots,a_q)$ as the answers to her queries to $\mathcal{P}$'s oracles, and when the rest of her view (i.e., her randomness and the answers to her queries to the pre-processed string) is x_{exp}.

The protocol $(\mathcal{P}',\mathcal{V}')$ is obtained from $(\mathcal{P},\mathcal{V})$ via the following modifications. Let $m_1,\ldots,m_\ell$ denote the prover messages in $(\mathcal{P},\mathcal{V})$. In the communication phase, for $i=1,2,\ldots,\ell$, if $i\in I_O$, then $\mathcal{P}'$ sends the oralce $w_i=m_i$, and otherwise, $\mathcal{P}'$ sends the oracle $w_i=E(m_i)\in\mathbb{F}^{\mathrm{cc}/\rho}$. Then, in the query phase, $\mathcal{V}'$ generates $\mathcal{V}$'s queries to the pre-processed string, and queries these locations. $\mathcal{P}'$ uses $\mathcal{V}'$'s messages to determine the queries which $\mathcal{V}$ makes to the pre-processed string and the oracles sent by $\mathcal{P}$. Let $z\in\{0,1\}^*$ denote the messages which $\mathcal{V}'$ sent to $\mathcal{P}'$ before the query phase (i.e., randomness used for pre-processing and the messages in the ℓ rounds of $(\mathcal{P},\mathcal{V})$), as well as the answers to her queries to the pre-processed string), let $a_1,\ldots,a_q$ denote the answers to the oracle queries, and let $w:=((a_1)^{(\mathrm{cc}/\rho)},\ldots,(a_q)^{(\mathrm{cc}/\rho)},(w_i)_{i\notin I_O})$. $\mathcal{P}'$ and $\mathcal{V}'$ run the $\frac{\delta}{2(\ell+q)}$-PCPP over $\mathbb{F}$ for $\mathcal{L}_{\mathcal{V}}$, given by Theorem 3.8, on explicit input z and implicit input w, and $\mathcal{V}'$ accepts if and only if the PCPP verifier accepts. More specifically, queries of the PCPP verifier $\mathcal{V}_{\mathsf{PCPP}}$ to the proof are answered using $\mathcal{V}'$'s PCPP oracle, queries of $\mathcal{V}_{\mathsf{PCPP}}$ to any of the $w_i, i\notin I_O$ are answered using $\mathcal{V}'$'s oracle w_i, and queries to the l'th entry in one of the $(a_j)^{\mathrm{cc}/\rho}$ strings are answered by querying a_j and providing this as the oracle's answer.

It can be verified that the round complexity, query complexity, verifier running time, and prover running time of the protocol $(\mathcal{P}',\mathcal{V}')$ are all as claimed, and that the requirements in the 'moreover' part are satisfied. Note that $\mathcal{T}_{\mathcal{V}}\geq(\ell-|I_O|)\cdot\mathrm{cc}$, and so the communication com-

plexity is also as claimed. Completeness is also straightforward. Next we show soundness and zero-knowledge.

Soundness: Assume that $x \in \mathsf{NO}$. Let $w_1^*, \ldots, w_\ell^*$ denote the first ℓ messages of $\mathcal{P}'$. Let $z \in \{0,1\}^*$ denote the messages which $\mathcal{V}'$ sent to $\mathcal{P}'$ before the query phase (i.e., randomness used for pre-processing and the messages in the ℓ rounds of $(\mathcal{P}, \mathcal{V})$), as well as the answers to her queries to the pre-processed string). Let $a_1, \ldots, a_q$ be the answers to the queries of $\mathcal{V}$ to $(w_i^*)_{i \in I_O}$, and let $w^* = ((a_1)^{(\mathrm{cc}/\rho)}, \ldots, (a_q)^{(\mathrm{cc}/\rho)}, (w_i^*)_{i \notin I_O})$. We shall show that with probability at least ϵ, w^* is $\frac{\delta}{2(\ell+q)}$-far from $(\mathcal{L}_\mathcal{V})_z$, and so by Theorem 3.8, the PCPP verifier, and consequently also $\mathcal{V}'$, will reject with probability at least $\frac{1}{2}$ (and hence the total rejection probability is at least $\frac{\epsilon}{2}$).

For $i \notin I_O$, let $c_i \in \mathbb{F}^{\mathrm{cc}/\rho}$ be the codeword of E that is closest to w_i^*, and let $w^{**} = ((a_1)^{(\mathrm{cc}/\rho)}, \ldots, (a_q)^{(\mathrm{cc}/\rho)}, (c_i)_{i \notin I_O})$. First note that by the soundness property of the IOP $(\mathcal{P}, \mathcal{V})$, with probability at least ϵ, we have that $w^{**} \notin (\mathcal{L}_\mathcal{V})_z$. In what follows, assume that this event holds. Suppose that $\tilde{w} = (u_1, \ldots, u_q, (v_i)_{i \notin I_O})$ is $\frac{\delta}{2(\ell+q)}$-close to w^*, where $u_j, v_i \in \mathbb{F}^{\mathrm{cc}/\rho}$ for all $j \in [q]$ and $i \notin I_O$. We shall show that $\tilde{w} \notin (\mathcal{L}_\mathcal{V})_z$, and so w^* is at least $\frac{\delta}{2(\ell+q)}$-far from $(\mathcal{L}_\mathcal{V})_z$.

To see why $\tilde{w} \notin (\mathcal{L}_\mathcal{V})_z$ for $\tilde{2}$ as above, note first that if there exists $j \in [q]$ so that the entries in u_j are non-identical, then clearly $\tilde{w} \notin (\mathcal{L}_\mathcal{V})_z$. Hence we may assume that for any $j \in [q]$, all entries in u_j are identical. Moreover, by assumption that $\tilde{w}$ is $\frac{\delta}{2(\ell+q)}$-close to w^*, we must have that for any $j \in [q]$, $u_j = (a_j)^{(\mathrm{cc}/\rho)}$. Indeed, if there exists j^* such that $u_{j^*} = (a')^{(\mathrm{cc}/\rho)}$ for $a' \neq a_j$, then the distance between $w^*, \tilde{w}$ is at least

$$\frac{\mathrm{cc}/\rho}{(q + (\ell - |I_O|)) \cdot \frac{\mathrm{cc}}{\rho}} = \frac{1}{q + (\ell - |I_O|)} > \frac{1}{q+\ell} \geq \frac{\delta}{2(q+\ell)}.$$

Similarly, if there exists $i \notin I_O$ so that v_i is not a codeword of E, then clearly $\tilde{w} \notin (\mathcal{L}_\mathcal{V})_z$. Hence we may assume that all v_i are codewords of E. Moreover, by assumption that $\tilde{w}$ is $\frac{\delta}{2(\ell+q)}$-close to w^*, we must have that $\mathrm{dist}(v_i, w_i^*) < \frac{\delta}{2}$ for any $i \notin I_O$. But since E has relative distance at least δ, this implies in turn that v_i is the closest codeword to w_i^*, and so $v_i = c_i$. But in this case we have that $\tilde{w} = w^{**} \notin (\mathcal{L}_\mathcal{V})_z$.

Zero-knowledge: We describe a simulator Sim' that uses (in a black-box, straight-line manner) a simulator Sim for the IOP $(\mathcal{P}, \mathcal{V})$ (whose existence follows from the lemma's assumptions), and perfectly simulates the prover and oracles for any $(t-q)$-restricted verifier $\mathcal{V}^*$.

Sim', on input x_{exp}, operates as follows. It runs Sim with input x_{exp}, relaying messages between Sim and $\mathcal{V}^*$ in the first ℓ rounds of the execution (notice that $\mathcal{V}^*$'s messages in these rounds define a verifier $\mathcal{V}^{**}$ for Sim in the system $(\mathcal{P}, \mathcal{V})$). Additionally, Sim' forwards oracle queries of $\mathcal{V}^*$ to Sim, and sends the answers back to $\mathcal{V}^*$. In the final round of the execution, Sim' uses the random messages received from $\mathcal{V}^*$ in previous rounds to determine the queries which the honest verifier $\mathcal{V}'$ would have made to the pre-processed string, as well as her q oracle queries. (These are the queries which the honest prover $\mathcal{P}'$ would have used to generate the PCPP in the last round.) Sim' forwards the q oracle queries to Sim and obtains from it the simulated answers $a'_1, \ldots, a'_q$. Let $(m'_i)_{i \notin I_O}$ denote the simulated messages which Sim provided in this execution, and for every $i \notin I_O$, let $w'_i := E(m'_i)$. Then Sim' honestly generates the PCPP π for $w' := ((a'_1)^{\mathrm{cc}/\rho}, \ldots, (a'_q)^{\mathrm{cc}/\rho}, (w'_i)_{i \notin I_O})$, and uses π to answer $\mathcal{V}^*$'s oracle queries to the PCPP. We note that Sim' can indeed generate π, because it fully knows w', z. (To see why Sim' knows z, notice that Sim' knows the messages sent by $\mathcal{V}^*$. These, together with x_{exp}, determine the pre-processed string which $\mathcal{V}'$ would have generated, and the queries she would have made to it.)

Clearly, Sim' is PPT (because Sim and the PCPP prover are). Moreover, Sim' forwards at most $(t-q)+q$ oracle queries to Sim (the $\leq t-q$ queries which $\mathcal{V}^*$ makes to her oracles directly, and the q queries which Sim' makes in the last round to generate the PCPP). Therefore, the black-box straight-line t-ZK of $(\mathcal{P},\mathcal{V})$ guarantees that in its emulation, Sim perfectly simulates the explicit prover messages and oracles answers. Since all the messages which Sim' sent to $\mathcal{V}^*$ are fully determined by the simulated values provided by Sim, the simulation is perfect.

■

The next lemma states that we can obtain a constant number of queries even over the binary alphabet:

Lemma 7.4. *Let $\ell = O(1)$, and let $\mathbb{F}$ be a finite field of characteristic 2. Let $(\mathcal{P},\mathcal{V})$ be an ℓ-round pre-processing IOP over $\mathbb{F}$ with soundness error $1-\epsilon$ for a promise problem $(\mathsf{YES},\mathsf{NO})$ with no implicit input, with constant query complexity, black-box straight-line t-ZK, and properties (1) and (4) from Lemma 7.2.*

Then there exists an $(\ell+1)$-round pre-processing IOP $(\mathcal{P}',\mathcal{V}')$ (over the binary alphabet) for $(\mathsf{YES},\mathsf{NO})$ with soundness error $1-\frac{\epsilon}{2}$, communication complexity $\mathrm{cc}_0+\mathsf{poly}(T_{\mathcal{V}}(n))$, and constant query complexity, that has black-box straight-line $(t-O(1))$-ZK. The verifier's running time is $\mathsf{poly}(T_{\mathcal{V}}(n))$, after a local preprocessing step of running time $\mathsf{poly}(n)$, and the prover's running time is $\mathsf{poly}(n)$. Moreover, the first prover's message is a binary string of length cc_0, and the rest of the prover's messages are of length $\mathsf{poly}(T_{\mathcal{V}}(n))$.

Proof: As before, we start by setting some notation. Let $s := \log(|\mathbb{F}|)$. Let $q_0 = O(1)$ denote the number of queries made to the first prover message, and let $q = O(1)$ denote the total number of queries made to the rest of the prover messages. Let E be any efficient binary code ensemble of constant rate $\rho > 0$ and constant relative distance $\delta > 0$. For a bit $b \in \{0,1\}$, and an integer $d \geq 1$, let $b^{(d)} \in \{0,1\}^d$ denote the string which consists of the concatenation of d copies of b. Slightly abusing notation, we view each element of $\mathbb{F}$ as a binary string of length $\log(\mathbb{F}|)$ in the natural way.

Let $\mathcal{L}_{\mathcal{V}} \subseteq \{0,1\}^*$ denote the language consisting of all strings of the form $(x_{\mathrm{exp}}, x_{\mathrm{imp}})$, where $x_{\mathrm{exp}} \in \{0,1\}^*$, $x_{\mathrm{imp}} = ((b_1)^{(s/\rho)},\ldots,(b_{q_0})^{(s/\rho)}, E(a_1),\ldots,E(a_q))$ for $b_j \in \{0,1\}$ and $a_i \in \mathbb{F}$, and $\mathcal{V}$ accepts, when given $(b_1,\ldots,b_{q_0})$ as the answers to her queries to the first prover message, $(a_1,\ldots,a_q)$ as the answers to her oracle queries to the rest of the prover messages, and when the rest of her view (i.e., her randomness and the answers to her queries to the pre-processed string) is x_{exp}.

The protocol $(\mathcal{P}',\mathcal{V}')$ is obtained from $(\mathcal{P},\mathcal{V})$ via the following modifications. Let $m_1,\ldots,m_\ell$ denote the prover messages in $(\mathcal{P},\mathcal{V})$. In the communication phase, in the first round, $\mathcal{P}'$ sends $w_1 = m_1 \in \{0,1\}^{\mathrm{cc}_0}$, and in rounds $i = 2,\ldots,\ell$, $\mathcal{P}'$ sends $w_i \in \{0,1\}^{\mathrm{cc}\cdot s/\rho}$ that is obtained by encoding each $\mathbb{F}$-entry of m_i with E. Then, in the query phase, $\mathcal{V}'$ generates $\mathcal{V}$'s queries to the pre-processed string, and queries these locations. $\mathcal{P}'$ uses $\mathcal{V}'$'s messages to determine the queries which $\mathcal{V}$ makes to the pre-processed string and the oracles sent by $\mathcal{P}$. Let $z \in \{0,1\}^*$ denote the messages which $\mathcal{V}'$ sent to $\mathcal{P}'$ before the query phase (i.e., randomness used for pre-processing and the messages in the ℓ rounds of $(\mathcal{P},\mathcal{V})$), as well as the answers to her queries to the pre-processed string. Let $b_1,\ldots,b_{q_0} \in \{0,1\}$ denote the answers to her queries to the first prover message, and let $e_1,\ldots,e_q \in \{0,1\}^{s/\rho}$ denote the blocks in $w_2,\ldots,w_\ell$ which corresponding to the encoding of the answers to $\mathcal{V}$'s queries to the rest of the prover messages. Let $w := ((b_1)^{(s/\rho)},\ldots,(b_{q_0})^{(s/\rho)},e_1,\ldots,e_q)$. $\mathcal{P}'$ and $\mathcal{V}'$ run the $\frac{\delta}{2(q_0+q)}$-PCPP (over the binary alphabet) for $\mathcal{L}_{\mathcal{V}}$, given by Theorem 3.8, on explicit input z and implicit input w, and $\mathcal{V}'$ accepts if and only if the PCPP verifier accepts. More specifically, queries of the PCPP verifier $\mathcal{V}_{\mathsf{PCPP}}$ to the proof $(e_1,\ldots,e_q$, respectively) are answered using $\mathcal{V}'$'s PCPP oracle (the appropriate w_i, respectively),

and queries to the l'th entry in one of the $(b_j)^{s/\rho}$ strings are answered by querying b_j from the first prover message, and providing this as the oracle's answer.

It can be verified that the round complexity, communication complexity, query complexity, verifier running time, and prover running time of the protocol $(\mathcal{P}', \mathcal{V}')$ are all as claimed, and that the requirements in the 'moreover' part are satisfied. Completeness is also straightforward. Next we show soundness and zero-knowledge. The proof are similar to those given in the proof of Lemma 7.3.

Soundness: Assume that $x \in \mathsf{NO}$. Let $w_1^*, \ldots, w_\ell^*$ denote the first ℓ messages of $\mathcal{P}'$. Let $z \in \{0,1\}^*$ denote the messages which $\mathcal{V}'$ sent to $\mathcal{P}'$ before the query phase, and the answers to her queries to the pre-processed string. Let $b_1, \ldots, b_{q_0}$ be the answers to the queries of $\mathcal{V}$ to the first prover message, and let $e_1, \ldots, e_q \in \{0,1\}^{s/\rho}$ be the blocks in $w_2^*, \ldots, w_\ell^*$ corresponding to the encoding of the answers to $\mathcal{V}$'s queries to the rest of the prover messages. Let $w^* = ((b_1)^{(s/\rho)}, \ldots, (b_{q_0})^{(s/\rho)}, e_1, \ldots, e_q)$. We shall show that with probability at least ϵ, w^* is $\frac{\delta}{2(q_0+q)}$-far from $(\mathcal{L}_\mathcal{V})_z$, and so by Theorem 3.8, the PCPP verifier, and consequently also $\mathcal{V}'$, will reject with probability at least $\frac{1}{2}$ (and hence the total rejection probability is at least $\frac{\epsilon}{2}$).

For $i \in [q]$, let $c_i \in \{0,1\}^{s/\rho}$ be the codeword of E that is closest to e_i, and let $w^{**} = ((b_1)^{(s/\rho)}, \ldots, (b_{q_0})^{(s/\rho)}, c_1, \ldots, c_q)$. First note that by the soundness property of the IOP $(\mathcal{P}, \mathcal{V})$, with probability at least ϵ, we have that $w^{**} \notin (\mathcal{L}_\mathcal{V})_z$. In what follows, assume that this event holds. Suppose that $\tilde{w} = (u_1, \ldots, u_{q_0}, v_1, \ldots, v_q)$ is $\frac{\delta}{2(q_0+q)}$-close to w^*, where $u_j, v_i \in \{0,1\}^{s/\rho}$ for all $j \in [q_0]$ and $i \in [q]$. We shall show that $\tilde{w} \notin (\mathcal{L}_\mathcal{V})_z$, and so w^* is at least $\frac{\delta}{2(q_0+q)}$-far from $(\mathcal{L}_\mathcal{V})_z$.

To see why $\tilde{w} \notin (\mathcal{L}_\mathcal{V})_z$ for $\tilde{2}$ as above, note first that if there exists $j \in [q_0]$ so that the entries in u_j are non-identical, then clearly $\tilde{w} \notin (\mathcal{L}_\mathcal{V})_z$. Hence we may assume that for any $j \in [q_0]$, all entries in u_j are identical. Moreover, by the assumption that $\tilde{w}$ is $\frac{\delta}{2(q_0+q)}$-close to w^*, we must have that for any $j \in [q_0]$, $u_j = (b_j)^{(s/\rho)}$. Similarly, if there exists $i \in [q]$ so that v_i is not a codeword of E, then clearly $\tilde{w} \notin (\mathcal{L}_\mathcal{V})_z$. Hence we may assume that all v_i are codewords of E. Moreover, by the assumption that $\tilde{w}$ is $\frac{\delta}{2(q_0+q)}$-close to w^*, we must have that $\mathrm{dist}(v_i, e_i) < \frac{\delta}{2}$ for any $i \in [q]$. But since E has relative distance at least δ, this implies in turn that v_i is the closest codeword to e_i, and so $c_i = v_i$. But in this case we have that $\tilde{w} = w^{**} \notin (\mathcal{L}_\mathcal{V})_z$.

Zero-knowledge: We describe a simulator Sim' that uses (in a black-box, straight-line manner) a simulator Sim for the IOP $(\mathcal{P}, \mathcal{V})$ (whose existence follows from the lemma's assumptions), and perfectly simulates the prover and oracles for any $(t - q_0 - q)$-restricted verifier $\mathcal{V}^*$.

Sim', on input x_{exp}, operates as follows. It runs Sim with input x_{exp}, relaying messages between Sim and $\mathcal{V}^*$ in the first ℓ rounds of the execution (notice that $\mathcal{V}^*$'s messages in these rounds define a verifier $\mathcal{V}^{**}$ for Sim in the system $(\mathcal{P}, \mathcal{V})$). Additionally, Sim' forwards oracle queries of $\mathcal{V}^*$ to Sim, and sends the answers back to $\mathcal{V}^*$. In the final round of the execution, Sim' uses the random messages received from $\mathcal{V}^*$ in previous rounds to determine the queries which the honest verifier $\mathcal{V}'$ would have made to the pre-processed string, as well as her q_0 (q, respectively) oracle queries to the first prover message (the rest of the prover messages, respectively). Sim' forwards the $q_0 + q$ oracle queries to Sim and obtains from it the simulated answers $b'_1, \ldots, b'_{q_0} \in \{0,1\}$ and $a'_1, \ldots, a'_q \in \mathbb{F}$. For every $1 \leq i \leq q$, Sim' computes $e'_i := E(a'_i)$ (here, we view a'_i as a length-s bitstring). Then Sim' honestly generates the PCPP π for $w' := ((b'_1)^{s/\rho}, \ldots, (b'_{q_0})^{s/\rho}, e'_1, \ldots, e'_q)$, and uses π to answer $\mathcal{V}^*$'s oracle queries to the PCPP. We note that Sim' can indeed generate π, because it fully knows w', z.

Clearly, Sim' is PPT (because Sim and the PCPP prover are). Moreover, Sim' forwards at most $(t - q_0 - q) + q_0 + q$ oracle queries to Sim (the $\leq t - q_0 - q$ queries which $\mathcal{V}^*$ makes to her oracles directly, and the $q_0 + q$ queries which Sim' sends in the last round to generate the PCPP). Therefore, the black-box straight-line t-ZK of $(\mathcal{P}, \mathcal{V})$ guarantees that in its emulation, Sim perfectly simulates the explicit prover messages and oracles answers. Since all the messages which Sim' sent to $\mathcal{V}^*$ are fully determined by the simulated values provided by Sim, the simulation is perfect. ■

Acknowledgments

The first author is supported by the Milgrom family grant for Collaboration between the Technion and University of Haifa, by ISF grant 735/20, and by the European Union (ERC, ECCC, 101076663). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. The second author is supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, pages 2087–2104, 2017.

[ALM+92] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *FOCS*, pages 14–23, 1992.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and intractability of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[AS92] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *FOCS*, pages 2–13, 1992.

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

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In *ICALP*, pages 14:1–14:17, 2018.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *CRYPTO*, pages 701–732, 2019.

[BCF+16] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. On probabilistic checking in perfect zero knowledge. *Electron. Colloquium Comput. Complex.*, TR16-156, 2016.

[BCF+17] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Zero knowledge protocols from succinct constraint detection. In *TCC, Proceedings, Part II*, pages 172–206, 2017.

[BCG+17a] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Interactive oracle proofs with constant rate and query complexity. In *ICALP*, pages 40:1–40:15, 2017.

[BCG+17b] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In *ASIACRYPT, Proceedings, Part III*, pages 336–365, 2017.

[BCG20] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. In *TCC*, pages 19–46, 2020.

[BCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *STOC*, pages 585–594, 2013.

[BCGV16] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. Quasi-linear size zero knowledge from linear-algebraic PCPs. In *TCC 2016-A, Proceedings, Part II*, pages 33–64, 2016.

[BCL20] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. Zero-knowledge IOPs with linear-time prover and polylogarithmic-time verifier. Cryptology ePrint Archive, Report 2020/1527, 2020. https://eprint.iacr.org/2020/1527.

[BCL22] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. Zero-knowledge IOPs with linear-time prover and polylogarithmic-time verifier. In *EUROCRYPT*, pages 275–304, 2022.

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

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *TCC*, pages 31–60, 2016.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BFLS91] László Babai, Lance Fortnow, Leonid Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31, 1991.

[BGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Short PCPs verifiable in polylogarithmic time. In *CCC*, pages 120–134, 2005.

[BGKS20] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: sampling outside the box improves soundness. In *ITCS*, pages 5:1–5:32, 2020.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[BKK+16] Eli Ben-Sasson, Yohay Kaplan, Swastik Kopparty, Or Meir, and Henning Stichtenoth. Constant rate PCPs for circuit-SAT with sublinear query complexity. *Journal of the ACM*, 63(4):32:1–32:57, 2016.

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

[CCG+07] Hao Chen, Ronald Cramer, Shafi Goldwasser, Robbert de Haan, and Vinod Vaikuntanathan. Secure computation from random error correcting codes. In *EUROCRYPT*, pages 291–310, 2007.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *EUROCRYPT*, pages 316–334, 2000.

[CDN15] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[CFS17] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. Cryptology ePrint Archive, Report 2017/305, 2017. http://eprint.iacr.org/2017/305.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *EUROCRYPT, Proceedings, Part I*, pages 738–768, 2020.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *EUROCRYPT, Proceedings, Part I*, pages 769–793, 2020.

[DFK+92] Cynthia Dwork, Uriel Feige, Joe Kilian, Moni Naor, and Shmuel Safra. Low communication 2-prover zero-knowledge proofs for NP. In *CRYPTO*, pages 215–227, 1992.

[DGR99] Scott E. Decatur, Oded Goldreich, and Dana Ron. Computational sample complexity. *SIAM J. Comput.*, 29(3):854–879, 1999.

[DGR20] Scott E. Decatur, Oded Goldreich, and Dana Ron. A probabilistic error-correcting scheme that provides partial secrecy. In *Computational Complexity and Property Testing - On the Interplay Between Randomness and Computation*, pages 1–8. Springer, 2020.

[Din07] Irit Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3):12, 2007.

[DSW06] Irit Dinur, Madhu Sudan, and Avi Wigderson. Robust local testability of tensor products of LDPC codes. In *RANDOM*, pages 304–315, 2006.

[FMSS04] Jon Feldman, Tal Malkin, Rocco A. Servedio, and Cliff Stein. Secure network coding via filtered secret sharing. In *Proceedings of the 42nd Annual Allerton Conference on Communication, Control, and Computing*, 2004.

[FS11] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *Journal of the ACM*, 62(4):27:1–27:64, 2015.

[GLS+23] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In *CRYPTO, Proceedings, Part II*, pages 193–226, 2023.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304, 1985.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptol.*, 7(1):1–32, 1994.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001.

[GOS24] Tom Gur, Jack O'Connor, and Nicholas Spooner. Perfect zero-knowledge PCPs for #P. Cryptology ePrint Archive, Report 2024/462 (to appear at STOC'24), 2024. http://eprint.iacr.org/2024/462.

[HVW21] Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. ZK-PCPs from leakage-resilient secret sharing. In *ITC*, pages 6:1–6:21, 2021.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30, 2007.

[IMS12] Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. On efficient zero-knowledge PCPs. In *TCC*, pages 151–168, 2012.

[ISVW13] Yuval Ishai, Amit Sahai, Michael Viderman, and Mor Weiss. Zero knowledge LTCs and their applications. In *RANDOM*, pages 607–622, 2013.

[IW14] Yuval Ishai and Mor Weiss. Probabilistically checkable proofs of proximity with zero-knowledge. In *TCC*, pages 121–145, 2014.

[IWY16] Yuval Ishai, Mor Weiss, and Guang Yang. Making the best of a leaky situation: Zero-knowledge PCPs from leakage-resilient circuits. In *TCC 2016-A, Proceedings, Part II*, pages 3–32, 2016.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *STOC*, pages 723–732, 1992.

[KLR06] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In *STOC*, pages 109–118. ACM, 2006.

[KPT97] Joe Kilian, Erez Petrank, and Gábor Tardos. Probabilistically checkable proofs with zero knowledge. In *STOC*, pages 496–505, 1997.

[KR08] Yael Tauman Kalai and Ran Raz. Interactive PCP. In *ICALP*, pages 536–547, 2008.

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

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

[Mei14] Or Meir. Combinatorial PCPs with efficient verifiers. *Computational Complexity*, 23(3):355–478, 2014.

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

[Mie09] Thilo Mie. Short PCPPs verifiable in polylogarithmic time with $O(1)$ queries. *Annals of Mathematics and Artificial Intelligence*, 56(3-4):313–338, 2009.

[Ran13] Hugues Randriambololona. An upper bound of singleton type for componentwise products of linear codes. *IEEE Transactions on Information Theory*, 59(12):7936–7939, 2013.

[RR19] Noga Ron-Zewi and Ron Rothblum. Local proofs approaching the witness length. *Electronint Colloquium on Computational Complexity*, TR19-127, Revision 2, 2019.

[RR20] Noga Ron-Zewi and Ron D. Rothblum. Local proofs approaching the witness length [extended abstract]. In *FOCS*, pages 846–857, 2020.

[RR22] Noga Ron-Zewi and Ron Rothblum. Proving as fast as computing: Succinct arguments with constant prover overhead. In *STOC*, pages 1353–1363, 2022.

[RRR17] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum, 2017. Personal Communication.

[RRR21] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. *SIAM Journal on Computing*, 50(3), 2021.

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

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal of Computing*, 25(2):252–271, 1996.

[Set20] Srinath T. V. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *CRYPTO, Proceedings, Part III*, pages 704–737, 2020.

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

[Sud00] Madhu Sudan. Probabilistically checkable proofs - lecture notes, 2000. Available at http://madhu.seas.harvard.edu/MIT/pcp/pcp.ps.

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

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

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 299–328. Springer, 2022.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *CRYPTO*, pages 733–764. Springer, 2019.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *S&P*, pages 859–876, 2020.

A Black-Box ZK Implies ZK with Auxiliary Inputs for IOPs

In this section we prove Lemma 6.4, and use it to conclude that black-box straight-line ZK implies ZK with auxiliary inputs for IOPs.

Proof of Lemma 6.4. The proof follows the same outline as the proof of the corresponding claim for IPs, given in Goldreich and Oren [GO94], accounting also for oracle queries which $\mathsf{Sim}, \mathcal{V}^*$ make during the execution. However, since the original IOP system $(\mathcal{P}, \mathcal{V})$ has *perfect* ZK, we are able to describe a *black-box* straight-line simulator for the system with auxiliary inputs (cf. Remark A.2 below).

Let $\mathsf{Sim}^{\mathcal{O}}$ be a black-box straight-line PPT simulator for $(\mathcal{P}, \mathcal{V})$ whose existence is guaranteed from the lemma's assumptions. We will construct a simulator $\mathsf{Sim}_{\mathsf{aux}}$ for t-restricted verifiers $\mathcal{V}^*_{\mathsf{aux}}$ with auxiliary inputs. Before describing $\mathsf{Sim}_{\mathsf{aux}}$, we first set some notation.

Let $\mathcal{V}^*_{\mathsf{aux}}$ be a t-restricted verifier with auxiliary input. For every possible choice $z \in \{0,1\}^{p(|x_{\mathrm{exp}}|+|x_{\mathrm{imp}}|)}$ of auxiliary input for $\mathcal{V}^*_{\mathsf{aux}}$, we define a t-restricted verifier $\mathcal{V}^*_z$ (*without* auxiliary input) that has z and $\mathcal{V}^*_{\mathsf{aux}}$'s code hard-wired into it. $\mathcal{V}^*_z$ emulates $\mathcal{V}^*_{\mathsf{aux}}$, giving z to $\mathcal{V}^*_{\mathsf{aux}}$ as the auxiliary input. Crucially, even though $\mathcal{V}^*_{\mathsf{aux}}$ might be computationally unbounded, black-box access to $\mathcal{V}^*_z$ can be emulated *efficiently* given oracle access to $\mathcal{V}^*_{\mathsf{aux}}$ (who has auxiliary input z), and this emulation is perfect conditioned on the auxiliary input of $\mathcal{V}^*_{\mathsf{aux}}$ coinciding with the hard-wired value z of $\mathcal{V}^*_z$.

We now define the simulator $\mathsf{Sim}_{\mathsf{aux}}$. $\mathsf{Sim}_{\mathsf{aux}}$, on input $x_{\mathrm{exp}}, 1^{|x_{\mathrm{imp}}|}$, and given oracle access to $\mathcal{O}$ and to $\mathcal{V}^*_{\mathsf{aux}}(\cdot, z; r'_V)$ (where r'_V is the randomness of $\mathcal{V}_{\mathsf{aux}}$), emulates Sim, answering oracle queries of Sim to $\mathcal{O}$ and the verifier, respectively, by forwarding the queries to $\mathsf{Sim}_{\mathsf{aux}}$'s own oracle $\mathcal{O}$, and to $\mathcal{V}^*_{\mathsf{aux}}$, respectively.

Clearly, $\mathsf{Sim}_{\mathsf{aux}}$ is PPT, and makes the same number of queries to $\mathcal{O}$ as Sim. Moreover, since Sim is black-box and straight-line, so is $\mathsf{Sim}_{\mathsf{aux}}$. Furthermore, notice that when $\mathsf{Sim}_{\mathsf{aux}}$ has oracle access to $\mathcal{V}^*_{\mathsf{aux}}$ with auxiliary input z, then Sim is emulated with oracle access to $\mathcal{V}^*_z$.

We claim that for every $((x_{\mathrm{exp}}, x_{\mathrm{imp}}), w) \in \mathcal{R}$ and every auxiliary input $z \in \{0,1\}^{p(|x_{\mathrm{exp}}|+|x_{\mathrm{imp}}|)}$, $\mathsf{Sim}_{\mathsf{aux}}$ perfectly simulates $\mathcal{V}^*_{\mathsf{aux}}$'s view. By the black-box straight-line ZK property of $(\mathcal{P}, \mathcal{V})$ (in the lemma's assumption) we have, for every $((x_{\mathrm{exp}}, x_{\mathrm{imp}}), w) \in \mathcal{R}$, and every $z \in \{0,1\}^{p(|x_{\mathrm{exp}}|+|x_{\mathrm{imp}}|)}$:

$$\left(\mathsf{View}_{\mathcal{V}^*_z}\left(x_{\mathrm{exp}}, x_{\mathrm{imp}}, w\right), q_V\right)_{r_P, r_V} \equiv \left(\mathsf{View}^{\mathsf{Sim}}_{\mathcal{V}^*_z}\left(x_{\mathrm{exp}}, x_{\mathrm{imp}}, w\right), f(q_S)\right)_{r_V, r_{\mathsf{Sim}}}$$

where $r_P, r_V, r_{\mathsf{Sim}}$ are the random strings of $\mathcal{P}, \mathcal{V}^*_z$ and Sim, respectively, and q_V (q_S, respectively) is the number of queries which $\mathcal{V}^*_z$ (Sim, respectively) makes to all her oracles (to $\mathcal{O}$, respectively). This implies that for every $z \in \{0,1\}^{p(|x_{\mathrm{exp}}|+|x_{\mathrm{imp}}|)}$:

$$\left(\mathsf{View}_{\mathcal{V}^*_z}\left(x_{\mathrm{exp}}, x_{\mathrm{imp}}, w\right), z, q_V\right)_{r_P, r_V} \equiv \left(\mathsf{View}^{\mathsf{Sim}}_{\mathcal{V}^*_z}\left(x_{\mathrm{exp}}, x_{\mathrm{imp}}, w\right), z, f(q_S)\right)_{r_V, r_{\mathsf{Sim}}}.$$

By the definition of $\mathcal{V}^*_z$ we have, for every $((x_{\mathrm{exp}}, x_{\mathrm{imp}}), w) \in \mathcal{R}$, and every $z \in \{0,1\}^{p(|x_{\mathrm{exp}}|+|x_{\mathrm{imp}}|)}$:

$$\left(\mathsf{View}'_{\mathcal{V}^*_{\mathsf{aux}}}\left(x_{\mathrm{exp}}, x_{\mathrm{imp}}, w, z\right), q'_V\right)_{r_P, r'_V} = \left(\mathsf{View}_{\mathcal{V}^*_z}\left(x_{\mathrm{exp}}, x_{\mathrm{imp}}, w\right), z, q_V\right)_{r_P, r_V}$$

where r'_V is the random string of $\mathcal{V}^*_{\mathsf{aux}}$ (here, we use the fact that $\mathcal{V}^*_z$ emulates $\mathcal{V}^*_{\mathsf{aux}}$ with $\mathcal{V}^*_z$'s own randomness), and q'_V is the number of oracle queries $\mathcal{V}^*_{\mathsf{aux}}$ makes. Moreover

$$\left(\mathsf{View}^{\mathsf{Sim}_{\mathsf{aux}}}_{\mathcal{V}^*_{\mathsf{aux}}(\cdot,z)}\left(x_{\mathrm{exp}}, x_{\mathrm{imp}}, w\right), f(q'_S)\right)_{r'_V, r'_{\mathsf{Sim}}} \equiv \left(\mathsf{View}^{\mathsf{Sim}}_{\mathcal{V}^*_z}\left(x_{\mathrm{exp}}, x_{\mathrm{imp}}, w\right), f(q_S)\right)_{r_{\mathsf{Sim}}, r_V},$$

where r'_{Sim} is the randomness of $\mathsf{Sim}_{\mathsf{aux}}$ (used also as the randomness of Sim), and q'_S is the number of queries it makes to $\mathcal{O}$. Therefore, for every $((x_{\exp}, x_{\mathrm{imp}}), w) \in \mathcal{R}$, and every $z \in \{0,1\}^{p(|x_{\exp}|+|x_{\mathrm{imp}}|)}$, we have

$$\left(\mathsf{View}'_{\mathcal{V}^*_{\mathsf{aux}}}\left(x_{\exp}, x_{\mathrm{imp}}, w, z\right), q'_V\right)_{r_P, r'_V} \equiv \left(\mathsf{View}^{\mathsf{Sim}_{\mathsf{aux}}}_{\mathcal{V}^*_{\mathsf{aux}}(\cdot,z)}\left(x_{\exp}, x_{\mathrm{imp}}, w\right), f(q'_S)\right)_{r'_V, r'_{\mathsf{Sim}}}.$$

■

Setting $\mathcal{O} = x_{\mathrm{imp}}$, and f to the identity function, in Lemma 6.4, we immediately obtain the following:

Corollary A.1. *Let $t \in \mathbb{N}$, let $\mathcal{R}$ be an NP relation with corresponding promise problem $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$, and let $(\mathcal{P}, \mathcal{V})$ be an IOP for $(\mathsf{YES}_{\mathcal{R}}, \mathsf{NO}_{\mathcal{R}})$. If $(\mathcal{P}, \mathcal{V})$ has black-box straight-line t-ZK, then it has t-ZK with auxiliary inputs.*

Remark A.2. *Three remarks are in order. First, notice that we do not give the auxiliary input z to the black-box simulator $\mathsf{Sim}'_{\mathsf{aux}}$ (cf. remarks on Definition 3.17).*

Second, the standard definition of black-box ZK with auxiliary inputs (for interactive proofs) gives the distinguisher oracle access to $\mathcal{V}^*_{\mathsf{aux}}$, but does not *give it the auxiliary input. However, Eq. (14) implies black-box ZK with auxiliary inputs (when z is not provided as input to the distinguisher) by truncating z from both sides of Eq. (14). (We do not give the distinguisher access to $\mathcal{V}^*_{\mathsf{aux}}$.)*

Third, the result of Goldreich and Oren [GO94] shows that for interactive proofs, black-box ZK implies ZK with auxiliary inputs, but with a non-black box *simulator. This is because in [GO94], $\mathcal{V}^*_z$ cannot be emulated with black-box access to $\mathcal{V}^*_{\mathsf{aux}}$, and the reason is that they need $\mathcal{V}_z$ to have the additional feature that she answers queries of the form "reveal your auxiliary input" by replying with the hard-wired z (this cannot be obtained with black-box access to $\mathcal{V}^*_{\mathsf{aux}}$). However, this additional feature is* not *used in the simulation, rather it is used to prove that the simulated output of $\mathsf{Sim}_{\mathsf{aux}}$, and the real view of $\mathcal{V}^*_{\mathsf{aux}}$, are* computationally *indistinguishable. More specifically, the proof uses a PPT distinguisher $\mathcal{D}_{\mathsf{aux}}$ between the real view of $\mathcal{V}^*_{\mathsf{aux}}$ and the output of $\mathsf{Sim}_{\mathsf{aux}}$ to construct a PPT distinguisher $\mathcal{D}$ between the view of $\mathcal{V}_z$ and the output of Sim. However, emulating $\mathcal{D}_{\mathsf{aux}}$ requires giving it the auxiliary input z of $\mathcal{V}^*_{\mathsf{aux}}$, which $\mathcal{D}$ does not have. Since $\mathcal{D}$ is PPT, [GO94] need to "program" $\mathcal{V}_z$ to provide z upon request. In our setting of* perfect *ZK, distinguishers are computationally unbounded (and potentially non-uniform), so we can hard-wire z into the distinguisher. We further note that the (interactive) distinguisher (against black-box ZK) in the proof of [GO94] accesses $\mathcal{V}^*_z$ only in two cases – to obtain z, and to answer queries of the distinguisher against ZK with auxiliary inputs.*

B Related Works

In the following, for a non-deterministic language, we denote by n, m, and N its *input length*, *witness length*, and *non-deterministic running time*, respectively.

Probabilistically-Checkable Proofs (PCPs). As mentioned in the introduction, the celebrated PCP theorem [ALM+92, AS92] asserts that any NP language has a constant-query PCP with constant soundness error. The works of [ALM+92, AS92] obtain $\mathsf{poly}(N)$ proof length for languages in $\mathsf{NTIME}(N)$, whereas [BGH+05, Din07] obtain much shorter proofs of length $N \cdot \mathsf{poly}\log(N)$ (the former with $\mathsf{poly}\log(N)$ query complexity, the latter with $O(1)$ queries). More recently, Ben-Sasson et al [BKK+16] constructed PCPs for circuit SAT of length $O(n)$, where n is the circuit size, albeit with a large query complexity, on the order of n^ϵ for an arbitrarily small constant $\epsilon > 0$.

Zero-knowledge PCPs (ZK-PCPs). Zero-Knowledge PCPs (ZK-PCPs) were introduced by Kilian, Petrank and Tardos [KPT97] who construct, for any $t \in \mathbb{N}$, a t-ZK-PCP for NP, i.e., with ZK against t-restricted verifiers. Their ZK-PCP has proofs of length $\mathsf{poly}(N, t)$ and an honest verifier that makes $\mathsf{poly}\log(N, t)$ *adaptive* queries to the proof oracle (and obtains a negligible soundness error). Their construction combines an earlier construction of a PCP with Honest-Verifier Zero-Knowledge (HVZK) of [DFK+92], and a cryptographic building block called a locking scheme. These building blocks were later improved by [IW14] and [IMS12], respectively, who give conceptually-simpler constructions of these objects. Ishai et al. [IWY16] design a t-ZK-PCP with $\mathsf{poly}(N, t)$-length proofs and a *non-adaptive* honest verifier, but the ZK simulator in their construction is not efficient (i.e., they obtain *witness indistinguishability*). Ishai et al. [IKOS07] design a t-ZK-PCP over a large alphabet (which implies an HVZK-PCP over the binary alphabet) with a *non-adaptive* honest verifier, based on secure multi-party computation protocols. To obtain $\mathsf{negl}(t)$ soundness error, the *honest* verifier in their construction must also make $\Omega(t)$ queries; this was reduced to $\sqrt{t}$ in [HVW21]. The works of [IKOS07, HVW21] obtain perfect ZK, whereas [KPT97, IW14, IWY16] obtain statistical ZK.

Interactive Oracle Proofs (IOPs). IOPs were first introduced by Reingold, Rothblum and Rothblum [RRR17] and Ben-Sasson, Chiesa and Spooner [BCS16], but special cases of IOPs were considered even earlier, in the Interactive PCP (IPCP) model of Kalai and Raz [KR08], and the duplex PCPs of Ben-Sasson et al. [BCGV16]. As discussed in Section 1, the study of IOPs has seen a rapid progress in the few years since its introduction, and is motivated from both theoretical and practical perspectives.

In terms of proof length, the shortest known IOPs are those of [RR20] who construct, for a large class of NP problems (concretely, any problem that can be verified in polynomial time and bounded polynomial space), a constant round and constant query IOPs whose proof length is $(1+\gamma)\cdot m$ (where m is the witness length) with constant soundness error. One of the main results of this work is a ZK variant of their result for the language 3-SAT. The technical core of the construction of [RR20] was a new *code switching* technique, inspired by [Mei13], that allows one to trade less efficient polynomial codes, commonly used in such proof systems, with more efficient *tensor codes*. This technique was later used in follow up works to obtain IOPs with *linear-time provers* [BCG20, RR22].

Zero-Knowledge IOPs (ZK-IOPs). Works on ZK-IOPs have considered different ZK guarantees, ranging from HVZK to ZK against t-restricted verifiers for an arbitrary t, and various efficiency measures. We focus here on works with full-fledged ZK or short (linear or near-linear) proof lengths.

This line of works was initialized by Ben-Sasson et al. [BCGV16] who construct 2-round IOPs for $\mathsf{NTIME}(N)$ with perfect t-ZK for any $t \in \mathbb{N}$, with $\widetilde{O}(N + t)$-length proofs and $\mathsf{poly}\log(N + t)$ query complexity. (This should be contrasted with the best ZK-PCP constructions to date, that either obtain *statistical* ZK with proofs of large *polynomial* length [KPT97, IW14, IWY16], or have a large query complexity [IKOS07, HVW21].) The simulator in the ZK-IOPs of [BCGV16] runs in $\mathsf{poly}(N + t)$ time, so they obtain efficient ZK simulation only for NP. This was later improved by [BCF+17] to ZK-IOPs for all of NEXP with the same parameters (and efficient simulation). The prover and verifier running times were further improved in subsequent work [BBHR19, CHM+20]. Follow-up work [BCR+19] design ZK-IOPs for the related problem of R1CS with ZK against t-restricted adversaries of proof length $O(n + t)$ and query complexity $O(\log(n + t))$, where n is the instance length. This was later extended to the holographic setting

by [COS20].

[AHIV17] design 2-round linear-length ZK-IPCPs for arithmetic circuit SAT in which the honest verifier queries $O(\sqrt{n})$ proof symbols. Bootle et al. [BCG+17b] give IOPs with HVZK for R1CS (over a large super-constant size field) with an $O(n)$-time prover and $O(\sqrt{n})$ query complexity. This was later improved in [BCL22], who construct HVZK-IOPs for R1CS with an $O(n)$-time prover and a polylogarithmic-time verifier, as well as full-fledged ZK – against malicious verifiers that make at most n^ϵ oracle queries, for some constant $\epsilon > 0$ – where the verification time increases to n^ϵ.

Many of the aforementioned works also construct (and, in some cases, implement) black-box ZK succinct (non-interactive) argument systems based on their ZK-IOP constructions.

ZK Sumcheck IOPs. A main ingredient in many of the ZK-IOP constructions mentioned above (e.g., [BCGV16, BCF+17, BCG+17a, BCR+19, CHM+20, ZXZS20, BCL22]) is a zero-knowledge IOP for the (univariate or multivariate) sumcheck problem. (In fact, many of these works design sumcheck ZK-IOPs for specific codes, such as RS or RM.) Such a sumcheck ZK-IOP (in fact, ZK-IPCP) was first given in [CFS17], though sumcheck IOPs with weaker ZK guarantees were given already in [BCGV16, BCF+16]. More specifically, [BCGV16, BCF+16] design sumcheck IOPs for polynomial codes (such as RS or RM) in which the view of a query-restricted verifier (that can query *both* the codeword and prover messages) can be simulated by making the same number of queries *to the codeword alone*.[39] This should be contrasted with our sumcheck IOP which, when applied to general (non-ZK) tensor codes leaks *rows* of the codeword. While the leakage in our sumcheck IOP is larger, it applies to general codes and has sublinear communication. We stress that the weaker ZK guarantee of our sumcheck IOP still suffices for constructing ZK-IOPs for 3SAT.

The main idea underlying all these aforementioned works (except for [XZZ+19, BCL22], see below) it to obtain ZK by applying a (standard, non-ZK) sumcheck IOP on a *random shift* of the tested codeword. That is, to test the sum $\sum_i \mathsf{m}(i)$ of a codeword $c = C(\mathsf{m})$, the prover first sends a uniformly random codeword $r \leftarrow C$, the verifier sends a random $\gamma \in \mathbb{F}$, and the parties then engage in a sumcheck IOP for the codeword $c' := \gamma \cdot c + r$. Intuitively, when combined with a ZK encoding of the witness (e.g., an LDE generated from an augmented witness that was padded with randomness), this guarantees ZK because r fully masks c. (See Further discussion in Section 2.2.) We stress that these sumcheck ZK-IOPs do *not* have sublinear communication. Indeed, the communication is at least n (where n is the length of the tested codeword) because the prover sends the oracle message r.

We note that [XZZ+19, BCL22] follow a similar approach, but are able to obtain *sublinear*-communication sumcheck ZK-IOPs for *specific* codes, by exploiting properties of these codes. Xie et al. [XZZ+19] design a zero-knowledge IOP based on the GKR protocol [GKR15]. Their construction includes a sublinear-communication sumcheck ZK-IOP which exploits the structure of the LDE encoding used in [GKR15] to enable using a short random masking. Bootle et al. [BCL22] design (as a main building block in their ZK-IOP construction) a sublinear-communication sumcheck ZK-IOP on *sparse* polynomial codes (i.e., ones in which the codeword corresponds to the evaluations of a polynomial with few monomials). Both results are tailored to the structure of polynomial codes (and the work of [BCL22] needs the further assumption that the polynomial is sparse), and it is not clear if or how these ideas can be generalized to sumchecking a generic tensor code, as we do in this work.

[39] The simulator in [BCGV16] is only efficient for codes with polynomial-size domains; this was improved in [BCF+16] whose simulation is efficient even for codes with an exponential-size domain.

Zero-Knowledge Codes. Zero-Knowledge codes are often used implicity in the context of secret sharing (most notably for Shamir's secret sharing) as well as for the design of ZK-IOPs (e.g., in [BCGV16, BCF+16, BCF+17, CFS17, BBHR19, BCL22]).

ZK codes were first explicitly defined by [DGR20, DGR99], who construct binary ZK codes with constant rate that are also able to correct a constant fraction of errors. Their construction works in two steps. First, they defined a property of encoding matrices such that any generator matrix G possessing the property induces a code that has ZK with respect to the encoding defined by G. Then, they gave an explicit construction of a generator matrix satisfying the property. Feldman et al. [FMSS04] showed that for any linear code with "sufficiently good" parameters, a generator matrix of the code – that additionally satisfies the matrix property of [DGR20, DGR99] – can be found probabilistically (except with negligible failure probability). This was later generalized by [ISVW13], who used this to show that the tensor product is a ZK-LTC with respect to *some* randomized encoding function which can be found probabilistically. Ishai et al. [ISVW13] also gave an efficient *explicit* transformation from any linear code to a ZK code (albeit with a *statistical* ZK property). They also show that for *any* (not necessarily linear) code (with sufficiently good parameters), there exists a random encoding function that is ZK for the code. The codes constructed in all these works are ZK against adversaries querying a constant fraction of codeword symbols.

Bootle et al. [BCL22] studied ZK and uniformity under general tensor products. They show that t-ZK (t-uniform ZK, respectively) is preserved under tensor products, specifically that if C_1, C_2 are codes such that C_i is t_i-ZK (t_i-uniform ZK, respectively) then $C_1 \otimes C_2$ is $\min\{t_1, t_2\}$-ZK ($\min\{t_1, t_2\}$-uniform ZK, respectively). (This should be contrasted with [BCGV16, BCF+16, BCF+17, CFS17, BBHR19, BCL22]) that focused specifically on tensor products of *Reed-Solomon*.) They use this property to give a probabilistic construction of $\Omega(k)$-uniform ZK codes that are linear-time encodable. They also give an algebraic characterization of t-ZK and t-uniform ZK of a code C, via the generating matrix and the distance of the dual codes. This characterization strengthens the result of [CDN15] (which was for the special case of Massey's scheme), and of [CCG+07, ISVW13] (who showed only one direction of these implications).

Sublinear-Length Sumcheck ZK-IOP

- ***Explicit input:*** $\lambda_1, \ldots, \lambda_d \in \mathbb{F}^k$ and $\alpha \in \mathbb{F}$.
- ***Implicit input:*** A codeword $c = C^{\otimes d}(\mathsf{m}) \in \mathbb{F}^{[n]^d}$, where $\mathsf{m} \in \mathbb{F}^{[k]^d}$.

$\triangleright$ Let $k' := k^{d-1}$, $n' := n^{d-1}$, $C' := C^{\otimes(d-1)}$, and $\lambda' := \lambda_2 \otimes \cdots \otimes \lambda_d$. In what follows, we view c as a codeword in $C \otimes C'$, where the goal is to verify that $\langle \lambda_1 \otimes \lambda', \mathsf{m} \rangle = \alpha$. Without loss of generality, assume that $\lambda'(1) \neq 0$.

1. **Masking the tested codeword c.**
   (a) $\mathcal{P}$ samples $r \leftarrow \mathbb{F}^k$, and sends $\beta := \lambda'(1) \sum_{i \in [k]} \lambda_1(i) \cdot r(i) \in \mathbb{F}$ and the oracle $u := C(r) \in \mathbb{F}^n$ to $\mathcal{V}$.
   $\triangleright$ Let $\bar{r} \in \mathbb{F}^{k \times k'}$ denote the matrix whose first column is r, and the rest of its entries are all zero, and let $R := (C \otimes C')(\bar{r})$.
   (b) **Local testing of the mask.**
      i. $\mathcal{V}$ runs the query phase of the robust local tester TEST for the code C, but instead of making the queries, $\mathcal{V}$ sends the query set $I \subseteq [n]$ of size $|I| = q$ to $\mathcal{P}$.
      ii. $\mathcal{P}$ sends $v := u|_I : I \to \mathbb{F}$ to $\mathcal{V}$.
      iii. $\mathcal{V}$ checks that $\psi_{\mathrm{TEST}}(v) = \mathsf{ACCEPT}$; If not, then it rejects and aborts.
      iv. $\mathcal{V}$ picks a uniform random point $i \in I$, queries $u(i)$, and checks that $u(i) = v(i)$; If not, then it rejects and aborts.
   (c) $\mathcal{V}$ samples $\gamma \leftarrow \mathbb{F}$ and sends γ to $\mathcal{P}$.
2. **Sumcheck over the random linear combination.** The parties now engage in a protocol whose goal is to verify that
$$\sum_{i \in [k], j \in [k']} \lambda_1(i) \cdot \lambda'(j) \cdot (\gamma \cdot \mathsf{m}(i,j) + \bar{r}(i,j)) = \gamma \cdot \alpha + \beta.$$
   (a) $\mathcal{P}$ computes
$$z := \sum_{j \in [k']} \lambda'(j) \cdot C(\gamma \cdot \mathsf{m}(*,j) + \bar{r}(*,j)) = \lambda'(1) \cdot u + \gamma \sum_{j \in [k']} \lambda'(j) \cdot C(\mathsf{m}(*,j)) \in \mathbb{F}^n,$$
   and sends z to $\mathcal{V}$.
   (b) $\mathcal{V}$ queries β, and checks that $z = C(y)$ for $y \in \mathbb{F}^k$ which satisfies that $\sum_{i \in [k]} \lambda_1(i) \cdot y(i) = \gamma \cdot \alpha + \beta$. If not, then $\mathcal{V}$ rejects and aborts.
   (c) $\mathcal{V}$ picks a random $i_1 \leftarrow [n]$, and sends i_1 to $\mathcal{P}$.
3. **Restricting to a row.**
   (a) $\mathcal{P}$ and $\mathcal{V}$ engage in the (non zero-knowledge) IOP for sumcheck $(\mathcal{P}', \mathcal{V}')$ given by Theorem 5.6 with the explicit input $\lambda_2, \ldots, \lambda_d \in \mathbb{F}^k$ and $z(i_1) \in \mathbb{F}$ and the implicit input $c' := \gamma \cdot c(i_1, *) + R(i_1, *) \in C^{\otimes(d-1)}$. If $\mathcal{V}'$ rejects, then $\mathcal{V}$ rejects and aborts; Otherwise, let $(i_2, \ldots, i_d) \in [n]^{d-1}$ and $\alpha' \in \mathbb{F}$ be the output of this protocol.
   (b) $\mathcal{V}$ queries $u(i_1)$, and computes $\sigma := (C'(u(i_1), 0, \ldots, 0))(i_2, \ldots, i_d) \in \mathbb{F}$.
   (c) i. If $\gamma = 0$ and $\sigma \neq \alpha'$, then $\mathcal{V}$ outputs reject.
      ii. Otherwise, if $\gamma = 0$ and $\sigma = \alpha'$, then $\mathcal{V}$ outputs accept.
      iii. Otherwise, $\mathcal{V}$ outputs $(i_1, \ldots, i_d)$ and $\alpha'' := (\alpha' - \sigma)/\gamma$.

Figure 3: Sublinear-Length ZK-IOP for Sumcheck

ZK-IOP for 3SAT - Bare-Bones Protocol

- ***Prover $\mathcal{P}$'s input:*** A satisfiable 3-CNF formula φ with $n := k^d$ variables for some integer $d > 1$, and a satisfying assignment $w \in \{0,1\}^{[k]^d}$ for φ.
- ***Verifier $\mathcal{V}$'s input:*** The 3-CNF φ.

1. **Parameters.** Let $k_1, k_2, n_1, n_2 \in \mathbb{N}$ such that $k < k_1 < k_2$ (the exact value of these parameters is set in Figures 5-7), and let $\mathbb{F}$ be a sufficiently large finite field. Let $C_2 : \{0,1\}^{k_2} \to \{0,1\}^{n_2}$ be a high-rate ZK-LTC, and let $C_1 : \mathbb{F}^{(3k_1)^9} \to \mathbb{F}^{n_1^9}$ be $C_1 := \mathsf{PRS}^{\otimes 9}_{3k_1,n_1}$.
2. **Witness Encodings.** The prover generates two randomized encodings of the witness w:
   (a) **Low-Rate Encoding.** (This encoding is never sent to $\mathcal{V}$.)
      i. $\mathcal{P}$ generates a randomized low-degree extension $\hat{w}_1$ of w.
      ii. Let $\hat{I}_\varphi$ be the low-degree extension of the function $I_\varphi : [k]^{3d+3} \to \{0,1\}$ which is 1 if and only if the clause $x_{\mathbf{i}_1} = b_1 \vee x_{\mathbf{i}_2} = b_2 \vee x_{\mathbf{i}_3} = b_3$ exists in φ.
      iii. **Arithmetization of φ:** let
      $$P_{\varphi,w_1}(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3) := \hat{I}_\varphi(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3) \cdot \Pi_{j=1}^3 (\hat{w}_1(\mathbf{i}_j) - b_j).$$
      iv. Let Q be sampled uniformly at random by $\mathcal{V}$, out of the family of zero-tester polynomials given by Lemma 3.39. $\mathcal{V}$ sends Q to $\mathcal{P}$, who computes
      $$P(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3) := Q(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3) \cdot P_{\varphi,w_1}(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3).$$
      v. The low-rate encoding of w is the codeword $c_1 = C_1^{\otimes(d+1)/3}(w_3) \in \mathsf{PRS}_{3k_1,n_1}$, where w_3 is the evaluation table of P on $[3k_1]^{3d+3}$.
   (b) **High-Rate Encoding.** $\mathcal{P}$ generates a randomized C_2-encoding c_2 of $\hat{w}_1$, and sends c_2 to $\mathcal{V}$.
3. **Local Testing of c_2.** $\mathcal{V}$ runs the local tester of C_2 on c_2.
4. **Sumcheck over LDE – Checking that $\varphi \in 3-\mathsf{SAT}$.** $\mathcal{P}$ and $\mathcal{V}$ execute the sumcheck ZK-IOP $(\mathcal{P}_{\mathsf{in}}, \mathcal{V}_{\mathsf{in}})$, given by Theorem 5.4 (version for tensor codes) on c_1 with input α. If $\mathcal{V}_{\mathsf{in}}$ accepts or rejects, then $\mathcal{V}$ does the same and aborts; Otherwise, the outcome of this execution is an evaluation point $(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3) \in (\mathbb{F} \setminus [3k_1])^{3d+3}$ and a value $\alpha' \in \mathbb{F}$.
5. **Sumcheck over High-Rate Encoding – Checking value of $P(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3)$.** Recall that $P(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3)$ can be generated from $\hat{w}_1(\mathbf{i}_1), \hat{w}_1(\mathbf{i}_2)$, and $\hat{w}_1(\mathbf{i}_3)$. In the next step, $\mathcal{V}$ and $\mathcal{P}$ engage in the sumcheck ZK-IOP $(\mathcal{P}'_{\mathsf{in}}, \mathcal{V}'_{\mathsf{in}})$ given by Theorem 5.4 (version for general codes) on c_2 with input α_j. If $\mathcal{V}'_{\mathsf{in}}$ rejects, or the outcome of this execution is $\mathbf{i}'_j \in [n_2]^d, \alpha'_j \in \mathbb{F}$, such that $c_2(\mathbf{i}'_j) \neq \alpha'_j$, then $\mathcal{V}$ rejects and aborts.
6. **Output.** $\mathcal{V}$ queries $\hat{I}_\varphi(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3)$ and $Q(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3)$, and accepts if and only if
$$Q(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3) \cdot \hat{I}_\varphi(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3) \cdot \Pi_{j=1}^3 (\alpha_j - b_j) = \alpha'.$$

Figure 4: A "Bare-Bones" Description of the ZK-IOP for 3SAT

ZK-IOP for 3SAT – Part 1

- ***Prover $\mathcal{P}$'s input:*** A satisfiable 3-CNF formula φ with $n := k^d$ variables for some integer $d > 1$, and a satisfying assignment $w \in \{0,1\}^{[k]^d}$ for φ.
- ***Verifier $\mathcal{V}$'s input:*** The 3-CNF φ.

1. **Parameters and codes.**

   (a) Let n_1 be the smallest power of 2 so that $24(d+1)k \leq n_1$. Let $\mathbb{F}$ be a finite field of size n_1, and identify $\mathbb{F}$ with $[n_1]$. Let $A = \{a_1, \ldots, a_s\}$ be a basis for $\mathbb{F}$ over the binary field, where $s = \log n_1$ and $a_1 = 1$. Let $k_1 = (1 + \frac{1}{s^2}) \cdot k$, $\bar{k}_1 = k + s \cdot (k_1 - k) = (1 + \frac{1}{s})k$, $k_2 = \bar{k}_1/(1 - \frac{\gamma}{8d})$, and $n_2 = k_2/(1 - \frac{\gamma}{8d})$.

   (b) Let $B := (\mathsf{PRS}_{3k_1,n_1})^{\otimes 3}$ (cf., Definition 3.36) and $C_1 = B^{\otimes 3}$.

   (c) Let $C_0 : \{0,1\}^{(k_2)^{1/4}} \to \{0,1\}^{(n_2)^{1/4}}$ be any explicit binary linear code of rate $1 - \frac{\gamma}{8d}$ and relative distance $(\frac{\gamma}{d})^{O(1)}$, and let $C_2 : \{0,1\}^{k_2} \to \{0,1\}^{n_2}$ be the binary linear ZK-LTC so that $\mathrm{image}(C_2) = \mathrm{image}((C_0)^{\otimes 4})$, given by Corollary 4.19.

2. **Local preprocessing.**

   (a) Let $I_\varphi : [k]^{3d+3} \to \{0,1\}$ be the function which satisfies that $I_\varphi(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) = 1$ for $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3 \in [k]^d$ and $b_1, b_2, b_3 \in [k]$ if and only if $b_1, b_2, b_3 \in \{0,1\}$ and the clause $x_{\mathbf{i}_1} = b_1 \vee x_{\mathbf{i}_2} = b_2 \vee x_{\mathbf{i}_3} = b_3$ exists in φ. $\mathcal{V}$ computes the low degree extension $\hat{I}_\varphi$ of I_φ over $\mathbb{F}$ (cf., Lemma 3.35).

   (b) $\mathcal{V}$ samples a random $(3d+3)$-variate polynomial Q over $\mathbb{F}$ of individual degree at most $k-1$ out of the family of zero-tester polynomials, given by Lemma 3.39, and computes the evaluation table of Q over $\mathbb{F}$.

Figure 5: ZK-IOP Approaching Witness Length for 3SAT (Part 1)

ZK-IOP for 3SAT – Part 2

3. **High-rate encoding.**

   (a) $\mathcal{P}$ generates $w_2 \in \{0,1\}^{[k_2]^d}$ as follows. For any $\mathbf{i} \in [k]^d$, $w_2(\mathbf{i}) = w(\mathbf{i})$. For any $\mathbf{i} \in [k_2]^d \setminus [\bar{k}_1]^d$, $w_2(\mathbf{i})$ is a uniform random bit, sampled independently at random. Next suppose that $\mathbf{i} \in [\bar{k}_1]^d \setminus [k]^d$, and let $\ell_0 \in [d]$ be the first index so that $\mathbf{i}(\ell_0) \notin [k]$. If there exists another index $\ell \neq \ell_0$ so that $\mathbf{i}(\ell) \notin [k]$ and $\mathbf{i}(\ell) - k \not\equiv 1 \mod s$, then $w_2(\mathbf{i}) = 0$. Otherwise, $w_2(\mathbf{i})$ is a uniform random bit, sampled independently at random.
   $\mathcal{P}$ sends $c_2 := (C_2)^{\otimes d}(w_2)$ to $\mathcal{V}$.

   (b) $\mathcal{V}$ runs the $(\sqrt{n_2}, (\frac{\gamma}{d})^{O(d)})$-robust local tester TEST for the code $(C_2)^{\otimes d}$, given by Theorem 3.33 (such a local tester exists since $\mathrm{image}((C_2)^{\otimes d}) = \mathrm{image}((C_0)^{\otimes(4d)})$), but instead of making the queries, $\mathcal{V}$ sends the query set $I \subseteq [n_2]^d$ of size $|I| = \sqrt{n_2}$ to $\mathcal{P}$.

   (c) $\mathcal{P}$ sends $v := c_2|_I : I \to \mathbb{F}$.

   (d) $\mathcal{V}$ checks that $\psi_{\mathrm{TEST}}(v) = \mathsf{ACCEPT}$; If not, then it rejects and aborts.

   (e) $\mathcal{V}$ picks a uniform random point $i \in I$, queries $c_2(i)$, and checks that $c_2(i) = v(i)$; If not, then it rejects and aborts.

4. **Arithmetization.**

   (a) $\mathcal{P}$ generates $w_1 \in \mathbb{F}^{[k_1]^d}$ as follows. For any $\mathbf{i} \in [k]^d$, $w_1(\mathbf{i}) = w_2(\mathbf{i})$. Next suppose that $\mathbf{i} \in [k_1]^d \setminus [k]^d$, and let $\ell_0 \in [d]$ be the first index so that $\mathbf{i}(\ell_0) \notin [k]$. For $h \in [s]$, let $\mathbf{i}^{(h)} \in [\bar{k}_1]^d$ be the vector which satisfies that $\mathbf{i}^{(h)}(\ell) = \mathbf{i}(\ell)$ if $\mathbf{i}(\ell) \in [k]$, $\mathbf{i}^{(h)}(\ell_0) = k + s \cdot (\mathbf{i}(\ell_0) - k - 1) + h$, and $\mathbf{i}^{(h)}(\ell) = k + s \cdot (\mathbf{i}(\ell) - k - 1) + 1$ otherwise. Then $w_1(\mathbf{i}) = \sum_{h \in [s]} a_h \cdot w_2(\mathbf{i}^{(h)})$.
   $\mathcal{P}$ computes the low degree extension $\hat{w}_1$ of w_1 over $\mathbb{F}$ (cf., Lemma 3.35), and the polynomial

$$P_{\varphi, w_1}(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) := \hat{I}_\varphi(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) \cdot \Pi_{j=1}^{3} (\hat{w}_1(\mathbf{i}_j) - b_j) .$$

   (b) $\mathcal{V}$ sends (the index of) Q, generated in Step 2b, to $\mathcal{P}$.
   $\mathcal{P}$ computes

$$P(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) := Q(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) \cdot P_{\varphi, w_1}(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) .$$

   $\triangleright$ Note that P is a $(3d+3)$-variate polynomial over $\mathbb{F}$ of individual degree at most $3k_1$. In particular, the evaluation table of P on $(\mathbb{F} \setminus [3k_1])^{3d+3}$ can be viewed as a codeword

$$c_1 \in (\mathsf{PRS}_{3k_1, n_1})^{\otimes(3d+3)}(w_3) = B^{\otimes(d+1)}(w_3) = (C_1)^{\otimes(d+1)/3}(w_3),$$

   where $w_3 \in \mathbb{F}^{[3k_1]^{3d+3}}$ is the evaluation table of P on $[3k_1]^{3d+3}$, and

$$P(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3) = (B^{\otimes(d+1)}(w_3))((\mathbf{i}_1, b_1) \star (\mathbf{i}_2, b_2) \star (\mathbf{i}_3, b_3))$$

   for any $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3, b_1, b_2, b_3 \in [3k_1]^{3d+3}$.

Figure 6: ZK-IOP Approaching Witness Length for 3SAT (Part 2)

ZK-IOP for 3SAT – Part 3

5. **Sumcheck over LDE.**

   The prover and verifier now engage in the sumcheck IOP given by Theorem 5.4 to prove that

   $$\sum_{\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3\in[k]^d,b_1,b_2,b_3\in[k]} P\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right)=0.$$

   (a) $\mathcal{P}$ and $\mathcal{V}$ execute the sumcheck ZK-IOP $(\mathcal{P}_{\mathsf{in}},\mathcal{V}_{\mathsf{in}})$, given by Theorem 5.4 (for tensor codes with respect to the code B), with the explicit input being $\alpha=0$ and $\lambda_1,\ldots,\lambda_{(d+1)/3}\in\mathbb{F}^{[3k_1]^9}$, where for any $\ell\in[(d+1)/3]$, $\lambda_\ell(i)=1$ for any $i\in[k]^9$, and $\lambda_\ell(i)=0$ otherwise, and the implicit input being $c_1=(C_1)^{\otimes(d+1)/3}(w_3)$.

   (b) If $\mathcal{V}_{\mathsf{in}}$ accepts or rejects, then $\mathcal{V}$ does the same and aborts; Otherwise, the outcome of this execution is an evaluation point $(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3)\in(\mathbb{F}\setminus[3k_1])^{3d+3}$ and a value $\alpha'\in\mathbb{F}$.

6. **Sumcheck over high-rate code.**

   Recall that $P\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right)$ can be generated from $\hat{w}_1(\mathbf{i}_1),\hat{w}_1(\mathbf{i}_2)$, and $\hat{w}_1(\mathbf{i}_3)$. In the next step, $\mathcal{V}$ and $\mathcal{P}$ engage in the sumcheck IOP given by Theorem 5.4 with the goal of computing the values $\hat{w}_1(\mathbf{i}_1),\hat{w}_1(\mathbf{i}_2)$, and $\hat{w}_1(\mathbf{i}_3)$.

   For $j=1,2,3$, sequentially:

   (a) $\mathcal{P}$ sends $\alpha_j:=\hat{w}_1(\mathbf{i}_j)$ to $\mathcal{V}$.

   Let $\lambda_{j,1},\ldots,\lambda_{j,d}\in\mathbb{F}^{k_1}$ be the vectors satisfying that $\hat{w}_1(\mathbf{i}_j)=\langle\lambda_{j,1}\otimes\cdots\otimes\lambda_{j,d},w_1\rangle$, given by Corollary 3.29. For $\ell=1,\ldots,d$, let $\lambda'_{j,\ell}\in\mathbb{F}^{k_2}$ be given by $\lambda'_{j,\ell}(i)=\lambda_{j,\ell}(i)$ for $i\in[k]$, $\lambda'_{j,\ell}(i)=0$ for $i\in[k_2]\setminus[\bar{k}_1]$, and $\lambda'_{j,\ell}(k+s\cdot(i-k-1)+h)=a_h\cdot\lambda_{j,\ell}(i)$ for any $i\in[\bar{k}_1]\setminus[k]$ and $h\in[s]$.

   (b) $\mathcal{P}$ and $\mathcal{V}$ emulate the sumcheck ZK-IOP $(\mathcal{P}'_{\mathsf{in}},\mathcal{V}'_{\mathsf{in}})$ given by Theorem 5.4 (for general codes) with the explicit input being α_j and $\lambda'_{j,1},\ldots,\lambda'_{j,d}\in\mathbb{F}^{k_2}$, and the implicit input being $c_2=(\widetilde{C}_2)^{\otimes d}(w_2)$, where $\widetilde{C}_2$ is the A-extension of C_2 (cf., Definition 4.20).

   (c) If $\mathcal{V}'_{\mathsf{in}}$ rejects, then $\mathcal{V}$ rejects and aborts; Otherwise, if the outcome of this execution is an evaluation point $\mathbf{i}'_j\in[n_2]^d$ and a value $\alpha'_j\in\mathbb{F}$, $\mathcal{V}$ queries $c_2(\mathbf{i}'_j)$, and checks that $c_2(\mathbf{i}'_j)=\alpha'_j$. If not, then $\mathcal{V}$ rejects and aborts.

7. **Output.**

   $\mathcal{V}$ queries $\hat{I}_\varphi(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3)$, $Q(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3)$, generated in Steps 2a and 2b, respectively, and accepts if and only if

   $$Q\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right)\cdot\hat{I}_\varphi\left(\mathbf{i}_1,\mathbf{i}_2,\mathbf{i}_3,b_1,b_2,b_3\right)\cdot\Pi_{j=1}^{3}\left(\alpha_j-b_j\right)=\alpha'.$$

Figure 7: ZK-IOP Approaching Witness Length for 3SAT (Part 3)