Time-Space Tradeoffs For Probabilistic Proofs

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Ziyi Guan
ziyiguan@mit.edu
MIT

Omer Paneth
omerpa@tauex.tau.ac.il
Tel Aviv University

Nicholas Spooner
nspooner@cornell.edu
Cornell University

Abstract

Many recent constructions of probabilistic proofs achieve fast proving times but have high space complexity (much higher than that of the computation being proved). Empirically this arises due to the fact that error-correcting codes, a key ingredient of such constructions, suffer from limiting time-space tradeoffs. It remained open, however, whether such time-space tradeoffs exist for proofs themselves.

We establish time-space tradeoffs for probabilistically checkable proofs (PCPs) as well as for their multi-round extension, interactive oracle proofs (IOPs), with straightline knowledge soundness. Our results hold for machine computations that have sequential input access, the same model for which similar tradeoffs hold for error-correcting codes. We complement this result by describing an IOP for R1CS that avoids the tradeoff but requires random access to the R1CS witness.

Our techniques build on and extend results for codes of Cook and Moshkovitz (CCC 2024). A key difficulty that we encounter in establishing our tradeoff is that distance of the code plays a key role in prior results, while for probabilistic proofs there is no natural notion of distance. Nevertheless, we show that, assuming the existence of collision-resistant hash functions, we can establish time-space tradeoffs by carefully relating a small-space machine's information flow to the proof system's properties.

Keywords: probabilistically checkable proofs; interactive oracle proofs; lower bounds; space efficiency

Contents

1 Introduction **3**
- 1.1 Our results . . . 4
- 1.2 Related work . . . 6

2 Techniques **7**
- 2.1 Recall: time-space tradeoffs for codes . . . 7
- 2.2 Challenge for probabilistic proofs . . . 8
- 2.3 Our intuition: knowledge soundness induces "distance" . . . 8
- 2.4 Ruling out complexity-preserving PCPs . . . 9
- 2.5 The case of complexity-preserving IOPs . . . 11
- 2.6 Space-efficient IOPs with random input access . . . 12

3 Preliminaries **13**
- 3.1 Interactive oracle proofs . . . 13

4 Sequential-access machines with interaction **15**
- 4.1 Lemmas on sequential-access machines . . . 16

5 Time-space tradeoff for interactive oracle proofs **19**
- 5.1 Proof of Theorem 5.1 . . . 19
- 5.2 Proof of Lemma 5.3 . . . 21
- 5.3 Proof of Lemma 5.4 . . . 23
- 5.4 Proof of Lemma 5.5 . . . 23

6 Random-access machines with interaction **25**

7 Space-efficient interactive oracle proofs **26**
- 7.1 Notation and tools . . . 26
- 7.2 The main theorem . . . 28
- 7.3 The IOP . . . 29
- 7.4 Efficiency . . . 32
- 7.5 Proof of Theorem 7.4 . . . 35

Acknowledgments **36**

AI disclosure **36**

References **36**

A The Cook–Moshkovitz code **41**
- A.1 Rate, distance, and encoder . . . 41
- A.2 The generator is locally computable . . . 41

B Analysis of the subprotocols **43**
- B.1 The R1CS reduction . . . 43
- B.2 Folding . . . 44
- B.3 Code switching . . . 45

C Completeness and knowledge soundness of the IOP **46**

1 Introduction

Probabilistic proofs are a fundamental ingredient in constructions of succinct arguments. In most cases, the efficiency of a succinct argument is closely related to that of the underlying probabilistic proof. This has motivated much effort, across academia and industry, towards the design and analysis of highly-optimized probabilistic proofs. In particular, minimizing the resources needed to run the prover algorithm is essential to enable proving the correctness of large computations, and remains an active and challenging research goal.

In this paper we study the time and space efficiency of prover algorithms. The gold standard is *complexity preservation* [BC12]: proving a computation that runs in time t and space s via a prover algorithm that runs in time $T = \tilde{O}(t)$ and space $S = \tilde{O}(s)$, where the notation "$\tilde{O}$" hides $\mathsf{poly}(\log t)$ factors. The prover algorithm essentially preserves the time and space of the original computation: "if you can run it, you can prove it."

Prover time and space for PCPs and IOPs. We investigate complexity preservation for two important types of probabilistic proof: *probabilistically-checkable proofs* (PCPs) [BFLS91; FGLSS91] and *interactive oracle proofs* (IOPs) [BCS16; RRR16]. Mild forms of time-space efficiency are known for PCPs/IOPs:

- [BFLS91] yields PCPs where the prover algorithm runs in time $T = \mathsf{poly}(t)$ and space $S = \mathsf{poly}(s)$, so the prover's time and space incur a polynomial blowup compared to those of the original computation; the query complexity of the PCP verifier is $\mathsf{q} = \mathsf{poly}(\log t)$.
- [BBHV22] obtains IOPs where the prover algorithm runs in time $T = \tilde{O}(t)$ and space $S = \tilde{O}(s)$ (thus preserving the time and space of the original computation as desired) but the query complexity of the IOP verifier is $\mathsf{q} = O(t/s)$, which is large (but still sublinear).

Yet *no constructions* of complexity-preserving PCPs/IOPs with small verifier query complexity are known. Many known PCPs/IOPs achieve essentially optimal prover time ($\tilde{O}(t)$ or even $O(t)$) and query complexity ($O(\lambda)$ for soundness error $2^{-\lambda}$); however these constructions also have prover *space* $\Omega(t)$ (even when $s \ll t$). Informally, this is due to the time and space efficiency of codes: known PCP/IOP constructions encode the computation trace, which is of size at least t, using an error-correcting code C. Thus the efficiency of the prover in such constructions can be no better than the encoding time of C for a message of length at least t.

For modern constructions, this encoding turns out to be the bottleneck for time-space efficiency. Time-efficient IOPs with small query complexity typically rely on Reed–Solomon codes with constant relative distance, encoded using the Fast Fourier Transform, which runs in time $T = \tilde{\Theta}(n)$ and space $S = \Theta(n)$ for messages of length n. The time-space efficient IOP of [BBHV22] uses k-interleaved Reed–Solomon codes, which achieve smaller space complexity $\Theta(n/k)$ at the cost of reduced distance $\delta = \Theta(1/k)$ (and therefore larger query complexity). Moreover, these time-space complexities are tight in a sense: for any code C with relative distance δ, an encoder for C with sequential input access running in time T and space S must satisfy $TS = \Omega(\delta n^2)$, where n is the message size [CM24]. We elaborate on the sequential input access model in detail below; here we remark only that the model is fairly permissive and corresponds essentially to algorithms that maintain time-space efficiency under composition.

Codes vs. PCPs/IOPs. Any tradeoffs for codes are, at best, circumstantial evidence for analogous tradeoffs for PCPs/IOPs. First, while codes play a role in all known PCP/IOP constructions, we do not know if codes are inherent to PCPs/IOPs (there are no known approaches to construct a code from a general PCP/IOP). Second, and more importantly, the tradeoff for codes in [CM24] is *unconditional*, whereas any tradeoff for PCPs/IOPs would have to be *conditional*. Indeed, if $\mathsf{P} = \mathsf{NP}$ then the PCP/IOP verifier can decide the language itself, in which case $T = S = 0$ regardless of the original computation! Overall, tradeoffs for PCPs/IOPs would have to rely on some hardness assumption, leading to a qualitatively different question.

Our focus: tradeoffs for PCPs/IOPs. In this work we ask:

Can we establish (conditional) time-space tradeoffs **for PCPs/IOPs** *analogous to the (unconditional) tradeoffs known for codes?*

1.1 Our results

We establish prover time-space tradeoffs for IOPs (and therefore also PCPs), assuming the existence of collision-resistant hash functions. Our results apply to proof systems that satisfy standard notions of straightline knowledge soundness, whose prover algorithms have sequential input access. We complement our tradeoff result with a positive result in the random access model: based on the time-space efficient codes of [CM24], we obtain a time-space efficient IOP for R1CS where the prover has random access to the R1CS witness. Our results, both positive and negative, can be viewed as an analogue of the results of [CM24] in the setting of probabilistic proofs.

Prover computation model: machines with sequential input access. Our results hold in a machine model with these characteristics: (i) the machine accesses its input sequentially; (ii) the machine has a bounded working memory; (iii) the machine writes its output sequentially. Following [CM24] (which itself follows [SV77]), we permit a generous notion of sequential input access in which the machine has multiple heads that can move left/right or jump to one another's positions. We do not restrict complexity of the transition function: in a single time step, the machine may write to its memory *any* efficiently computable function of its entire memory, its entire random tape, its entire query tape, all messages it received in the interaction, and the input values at the head positions (provided it fits). We spell out the model in more detail in Section 4.

While sequential access may appear to be an unnatural limitation, it is arguably the correct input access model for space-bounded proof systems. The reason is that essentially all known PCP/IOP constructions take as input a *constraint satisfaction problem* (CSP): a SAT formula, a rank-one constraint system (R1CS), a system of quadratic equations, or similar. To transform a machine computation into the solution to a CSP requires a Cook–Levin-type reduction, which entails running the computation and writing down the machine state at each timestep. Such reductions can themselves be implemented in a complexity-preserving fashion [BCGT13], but implementing the proof system end-to-end amounts to a *composition of small-space algorithms*, which does not preserve time-space complexity for general input access models [PT25].It is thus unclear whether a PCP/IOP prover which accesses the input in a nonsequential fashion could be used to construct a complexity-preserving proof system.

Time-space tradeoffs for probabilistic proofs. We establish a time-space tradeoff for any IOP for general non-deterministic computations that satisfies straightline knowledge soundness, a standard property satisfied by almost all known IOPs. Informally, this notion of knowledge soundness states that, except for a small extraction error, if the IOP verifier accepts then an extractor can find a valid witness given the instance and interaction transcript. Our tradeoff is conditional on the existence of collision-resistant hash functions. Below we denote by $\mathsf{NTISP}[t, s]$ the class of languages decidable by a nondeterministic single-tape Turing machine running in time t and space s.

Theorem 1 (informal)**.** *Fix any $\varepsilon > 0$. Let $\mathsf{IOP} = (\mathcal{P}, \mathcal{V})$ be an IOP for $\mathsf{NTISP}[t, s]$ that satisfies straightline knowledge soundness (with negligible error and polynomial extraction time), where $s \geq n^{\varepsilon}$ and $t = \mathsf{poly}(n)$. Suppose that the prover $\mathcal{P}$ can be computed via a sequential-access machine with time complexity T and space complexity S. Assuming the existence of collision-resistant hash functions, it must be that*

$$T \cdot S = \Omega\left(\frac{t^{2-\varepsilon}}{\mathsf{q}}\right) \quad ,$$

where q is $\mathcal{V}$'s query complexity.

Alternatively, assuming the existence of subexponentially hard collision-resistant hash functions, when $s \geq (\log n)^c$ for sufficiently large c, it holds that

$$T \cdot S = \tilde{\Omega}\left(\frac{t^2}{\mathsf{q}}\right) \quad ,$$

where $\tilde{\Omega}$ hides $\mathsf{polylog}(n)$ factors.

We make a few remarks about this result.

- We allow the prover a substantially stronger computation model than that used by computation being proved; this only strengthens our negative result.
- The tradeoff is independent of the round complexity of the IOP, and whether the interaction is public- or private-coin. PCPs are a special case of IOPs, so Theorem 1 also applies to PCPs of knowledge.
- Theorem 1 establishes a *separation between PCPs/IOPs and MIPs*, because there exist MIPs that are complexity preserving with sequential input access [BC12; BTVW14; HR18].
- Theorem 1 implies the tightness of existing IOP constructions in the sequential access model (up to polylogarithmic factors assuming subexponential CRH). For [BBHV22] it holds that $T \cdot S \cdot \mathsf{q} = \tilde{O}(t) \cdot \tilde{O}(s) \cdot O(t/s) = \tilde{O}(t^2)$; for known quasilinear-time IOPs, $T \cdot S \cdot \mathsf{q} = \tilde{O}(t) \cdot \tilde{O}(t) \cdot \tilde{O}(1) = \tilde{O}(t^2)$.

Time-space efficient IOPs for R1CS in the random-access model. We complement our negative result by showing that allowing the prover *random access* to its input enables time-space efficient IOPs for constraint satisfaction problems.

Theorem 2 (informal)**.** *There is a public-coin IOP for R1CS with straightline knowledge soundness error $2^{-\lambda}$ whose prover, on instances of size N, with random access to its input, achieves time complexity $N^{1+o(1)} \cdot \mathsf{poly}(\lambda)$ and space complexity $\mathsf{poly}(\lambda, \log N)$, and whose verifier makes $\mathsf{poly}(\lambda, \log N)$ queries to oracles over the binary alphabet.*

As discussed above, it is unclear how to use this result to construct a complexity-preserving IOP: the reduction from NTISP to R1CS does not appear to compose with this proof system in a way that preserves its time-space complexity due to the random access. On the other hand, a sufficiently careful complexity-preserving reduction from NTISP to R1CS implies that a variant of Theorem 2 in which the prover has *sequential* access to the input would violate Theorem 1.

Open questions. Our work provides the first time-space tradeoffs for generic probabilistic proof systems, which are tight in the model we consider. In light of our main result, a number of natural questions arise.

- Theorem 1 requires *straightline knowledge soundness* of the PCP/IOP. Can the result be extended to IOP constructions with rewinding extractors, or PCPs/IOPs without knowledge soundness?
- Is the CRH assumption necessary? Can we obtain a tradeoff based on the existence of one-way functions, or even a worst-case assumption like $\mathsf{P} \neq \mathsf{NP}$?
- Are there constructions of PCPs/IOPs for NTISP whose prover algorithms beat our tradeoff using random access to their input? Or can Theorem 1 be extended to include prover algorithms with random access? (This is not precluded by Theorem 2 due to the aforementioned difficulties with composition.)

1.2 Related work

The focus of this work are time-space tradeoffs for PCPs/IOPs. Both PCPs and IOPs can be used to construct succinct arguments that make a black-box use of cryptographic hash functions typically modeled as random oracles [Kil92; Mic00; BCS16; CY24]. These succinct arguments are widely used in practice, and they are valued for their concrete efficiency and post-quantum security. Constructions of succinct arguments in the random oracle model (ROM) inherit the time and space efficiency of the PCP/IOP up to polylogarithmic factors (and dependence on a security parameter for the hash function), yet the lack of complexity-preserving PCPs/IOPs has caused a corresponding lack of complexity-preserving succinct arguments in the ROM.

There are approaches to constructing complexity-preserving succinct arguments via "structured" cryptography or a non-black-box use of cryptography (including an approach based on a non-black-box use of hash functions). While these lie outside the scope of our question, for context we review constructions of succinct arguments that achieve good time-and-space efficiency below.

Complexity-preserving succinct arguments are introduced in [BC12], which obtains a private-coin protocol by combining fully homomorphic encryption and multi-prover interactive proofs. [BCCT13] gives a general template for building complexity-preserving succinct *non-interactive* arguments (SNARG): starting from any SNARG with knowledge soundness, they show how to build incrementally-verifiable computation (IVC) [Val08], and then show how to obtain complexity-preserving SNARGs from IVC. A later line of work shows how to obtain IVC from simpler objects known as accumulation (or folding) schemes [BGH19; BCMS20; BCLMS21; KST22]. [CCS22; CCGOS23] obtains IVC with straightline extraction in algebraically structured variants of the ROM with non-black-box use of CRHs. [BHRRS20] construct complexity preserving succinct arguments (with a non-succinct verifier) based on prime-order groups, and [BHRRS21] construct a complexity-preserving succinct argument (with a succinct verifier) based on unknown-order groups. [FPP24] construct a complexity-preserving succinct argument based solely on non-black-box use of CRHs.

2 Techniques

Our starting point is the time-space tradeoff for codes, which we recall in Section 2.1. The argument relies on code distance, but a PCP prover need not map distinct witnesses to distant proof strings (Section 2.2). Our main idea, presented in Section 2.3, is that knowledge soundness provides a substitute for distance. Section 2.4 uses this idea to rule out complexity-preserving PCPs with sequential input access, and Section 2.5 extends the argument to IOPs. Finally, Section 2.6 turns to the upper bound and sketches our space-efficient IOP with random input access.

2.1 Recall: time-space tradeoffs for codes

Our proof of Theorem 1 builds upon the aforementioned result on time-space efficient encoders [CM24], which shows that, for any code C with relative distance δ and input message size n, if the encoder is a sequential-access machine with time T, space S, and h input tape heads, then

$$h \cdot T \cdot S = \Omega(\delta \cdot n^2) \ .$$

We review the proof from [CM24] below. For simplicity, we focus the rest of our discussion on the setting of having only one input tape head. We handle multiple heads in Sections 4 and 5.

The proof proceeds in two main steps. First, because the encoder is time-space efficient and accesses the input only sequentially, there exists a large portion of the input that does not affect the output. Second, this implies that there exist two distinct messages whose encodings are very similar, violating the minimum-distance property of the code.

Step 1: Most input bits do not affect the output. We divide the message m into consecutive intervals of size b where $b \gg S$ (assume for simplicity that b divides n). Observe that the encoder must "visit" an interval (i.e., move the head there) at least b/S times in order to write a different output for every different assignment to this interval. Since the encoder only has sequential access to the input, moving the tape head from one interval to another takes a long time. Hence, due to the time and space constraints of the encoder, there must exist some input interval that does not affect the output.

Formally, [CM24] introduces the notion of a *good restriction* for the encoder and shows its existence via a counting argument. A tuple $(m_{-i}, i) \in \{0,1\}^{n-b} \times [n/b]$ is a good restriction if there exists a large set $Y \subseteq \{0,1\}^b$ (the good assignment set) such that for every $m_i \in Y$, the encoder on input

$$m_{-i} \circ_i m_i \coloneqq m_{-i}[1, \ldots, (i-1) \cdot b] \| m_i \| m_{-i}[(i-1) \cdot b + 1, \ldots, n-b]$$

satisfies the following two properties:

- the number of times the tape head enters the i-th interval is much less than b/S (i.e., information about m_i is recorded fewer than b/S times);
- less than a δ fraction of the output bits are written when the tape head is inside the i-th interval (or even in its neighboring intervals).

Step 2: Small space implies small distance. Let (m_{-i}, i) be a good restriction and Y be the good assignment set. Note that the encoder's behavior depends solely on two things: (i) the content stored in the work space, which has size S; (ii) the location pointed to by the input tape head, which has size $\log n$. For appropriate parameters, another counting argument shows that there must exist $m_i, m_i' \in Y$ where $m_i \neq m_i'$, and the encoder behaves exactly the same on m and m' except when the tape head is near the i-th interval, where $m = m_{-i} \circ_i m_i$ and $m' = m_{-i} \circ_i m_i'$. Since the i-th interval affects less than a δ fraction of the output, this violates the minimum distance of the code.

2.2 Challenge for probabilistic proofs

Good PCPs/IOPs are not known to imply good codes, and so time-space tradeoffs for codes are only weak evidence for similar tradeoffs for PCPs/IOPs. As described in Section 2.1, the argument for codes relies crucially on *distance*: any two distinct input messages that lead to similar codewords yield a contradiction. One can view a PCP prover as an encoder, which transforms an NP witness w into a proof string π. However, this encoding need not have distance in the coding-theoretic sense. For example, a PCP for a relation R implies a PCP for the padded relation $R' = \{(x, (w\|y)) : (x, w) \in R, y \in \{0,1\}^t\}$ with the same parameters in which many witnesses are mapped to the same proof string.

Moreover, known time-space tradeoffs for codes are unconditional: they do not rely on any complexity-theoretic or cryptographic assumptions. We cannot hope to get similar unconditional results on PCPs, or probabilistic proofs in general: as mentioned in Section 1, if $\mathsf{P} = \mathsf{NP}$, we can construct a PCP where the prover has zero time and space complexity.

2.3 Our intuition: knowledge soundness induces "distance"

We base our lower bound on the existence of collision-resistant hash families (CRHF). In particular, we show that, if the space complexity of the PCP prover is too low, we can construct an adversary that breaks the CRHF.

Geometric interpretation of distance for codes. One can think about the distance of a code with blocklength n geometrically. Consider the space of all strings of length n, where the distance between two strings is the number of positions on which they differ. If the code has relative distance δ, then around each codeword one can draw a Hamming ball of radius $\delta n/2$ that contains no other valid codewords. Equivalently, these Hamming balls are disjoint, so each codeword is separated from all others by distance at least δn.

"Distance" from knowledge soundness. We propose to mimic this geometric interpretation using knowledge soundness of PCPs. Recall that a PCP for a relation R is a pair of algorithms $\mathsf{PCP} = (\mathcal{P}_{\mathsf{PCP}}, \mathcal{V}_{\mathsf{PCP}})$, where the prover produces an oracle string π and the verifier makes a few queries to π. We consider PCPs satisfying completeness and knowledge soundness.

- *Completeness*: For every $(\mathbb{x}, \mathbb{w}) \in R$,

$$\Pr\left[\mathcal{V}^{\pi}_{\mathsf{PCP}}(\mathbb{x}; \rho) = 1 \;\middle|\; \begin{array}{l} \pi \leftarrow \mathcal{P}_{\mathsf{PCP}}(\mathbb{x}, \mathbb{w}) \\ \rho \leftarrow \{0,1\}^{\mathsf{r}} \end{array}\right] \geq 1 - \alpha_{\mathsf{PCP}}(\mathbb{x}) \ .$$

- *Knowledge soundness*: There exists an efficient extractor $\mathcal{E}$ such that for every instance $\mathbb{x}$ and every (unbounded) adversary $\widetilde{\mathcal{P}}_{\mathsf{PCP}}$,

$$\Pr\left[(\mathbb{x}, \mathbb{w}) \in R \;\middle|\; \begin{array}{l} \widetilde{\pi} \leftarrow \widetilde{\mathcal{P}}_{\mathsf{PCP}}(\mathbb{x}) \\ \mathbb{w} \leftarrow \mathcal{E}(\mathbb{x}, \widetilde{\pi}) \end{array}\right] \geq \Pr\left[\mathcal{V}^{\widetilde{\pi}}_{\mathsf{PCP}}(\mathbb{x}; \rho) = 1 \;\middle|\; \begin{array}{l} \widetilde{\pi} \leftarrow \widetilde{\mathcal{P}}_{\mathsf{PCP}}(\mathbb{x}) \\ \rho \leftarrow \{0,1\}^{\mathsf{r}} \end{array}\right] - \kappa_{\mathsf{PCP}}(\mathbb{x}) \ .$$

  For simplicity, we assume the PCP extractor $\mathcal{E}$ is deterministic for the discussion in this section.

Fix an instance $\mathbb{x} \in L(R)$, and consider the space Σ^{ℓ} of all PCP proof strings of length ℓ. For every valid witness $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in R$, define

$$B_{\mathbb{w}} \ := \ \{\widetilde{\pi} \in \Sigma^{\ell} : \mathcal{E}(\mathbb{x}, \widetilde{\pi}) = \mathbb{w}\} \ .$$

In other words, $B_{\mathbb{w}}$ is the set of all proof strings that extract to $\mathbb{w}$. We refer to it informally as the *ball* corresponding to $\mathbb{w}$ in the space of PCP proofs. Since the extractor $\mathcal{E}$ is deterministic, the collection $\{B_{\mathbb{w}}\}_{\mathbb{w}}$

forms a partition of all proof strings from which a witness can be extracted. Intuitively, we can define the "distance" between two distinct witnesses $\mathbb{w} \neq \mathbb{w}'$ to be

$$\min\left\{\Delta_H(\pi, \pi') : \pi \in B_{\mathbb{w}},\ \pi' \in B_{\mathbb{w}'}\right\}\ ,$$

namely, the minimum Hamming distance between any proof string that extracts to $\mathbb{w}$ and any proof string that extracts to $\mathbb{w}'$.

Syntactically, this new distance notion potentially allows us to apply an argument similar to the one outlined in Section 2.1. However, witnesses for an arbitrary relation R do not have a minimum distance property: they can be arbitrarily "close".

Thus, we consider a specific relation that enforces a minimum distance. In particular, we consider a relation R_H defined using a collision-resistant hash family:

$$R_H \coloneqq \left\{ \big((h, x), w\big) \;\middle|\; \begin{array}{l} h \in H \\ h(w) = x \end{array} \right\}\ .$$

Intuitively, if $\mathbb{w}$ and $\mathbb{w}'$ are too close, one can take a PCP proof $\widetilde{\pi}$ in $B_{\mathbb{w}}$ and modify it in very few locations to obtain another PCP proof $\widetilde{\pi}'$ in $B_{\mathbb{w}'}$. Then, we can find a hash collision using $\widetilde{\pi}$, contradicting the collision resistance of the hash family H.

2.4 Ruling out complexity-preserving PCPs

We show how to formalize the intuition above by adapting the proof in [CM24] to our distance setting. Let h be a collision-resistant hash function with input size n_{in} and output size n_{out}. Following the ideas from [CM24], we break any hash input $w \in \{0,1\}^{n_{\mathsf{in}}}$ into m consecutive intervals of size b (again, assume for simplicity that b divides n_{in}).

Step 1: Most witness bits do not affect the PCP string. We observe that one can analogously show that most bits in the witness w of the relation do not affect the PCP proof. In particular, we can argue that there exists a good restriction $(w_{-i}, i) \in \{0,1\}^{n_{\mathsf{in}}-b} \times [n_{\mathsf{in}}/b]$ and a large good assignment set $Y \subseteq \{0,1\}^b$ such that for every $w_i \in Y$ and $w \coloneqq w_{-i} \circ_i w_i$, the following holds:

- the number of times the tape head enters the i-th interval is much less than b/s (i.e., information about w_i is recorded fewer than b/s times);
- less than a δ fraction of the PCP string is written when the tape head is inside the i-th interval (or even in its neighboring intervals).

Step 2: Small space implies small distance. Then, we can argue that there must exist two hash inputs $w \coloneqq w_{-i} \circ_i w_i$ and $w' \coloneqq w_{-i} \circ_i w_i'$, where $w_i \neq w_i'$ are in the good assignment set Y, for which the PCP prover behaves exactly the same, except when the tape head is near the i-th interval. Let π be the honest proof output by $\mathcal{P}_{\mathsf{PCP}}((h, h(w)), w)$ and π' be the honest proof output by $\mathcal{P}_{\mathsf{PCP}}((h, h(w')), w')$. Since the i-th interval affects less than a δ fraction of the output bits, it follows that π and π' differ in less than a δ fraction of the locations.

Putting everything together: collision finder? According to the intuition above, we can construct an adversary A for the CRHF as follows:

$A(h)$:
1. Find a good restriction (w_{-i}, i). Find the good assignments w_i and w_i' described in Step 2 above.
2. Set $w \coloneqq w_{-i} \circ_i w_i$.

3. Compute the honest PCP string π with the PCP prover $\mathcal{P}_{\mathsf{PCP}}$ on instance $(h, h(w))$ and witness w. Let O be the set of output locations that are written when the tape head is near the i-th interval.
4. Construct $\widetilde{\pi}$ as follows:
$$\widetilde{\pi}[j] \coloneqq \begin{cases} \pi[j] & \text{if } j \notin \mathsf{O} \\ 0 & \text{otherwise} \end{cases} .$$
5. Compute $w^\star \leftarrow \mathcal{E}((h, h(w)), \widetilde{\pi})$.
6. Output $(w, w^\star)$.

In particular, we want to argue that $\widetilde{\pi}$ creates *ambiguity* for the PCP extractor $\mathcal{E}$: the information about the i-th interval of the witness is erased, and the proof could be in either the ball of $w = w_{-i} \circ_i w_i$ or the ball of $w' = w_{-i} \circ_i w'_i$. However, there are a few subtleties we need to handle before drawing this conclusion:

- By the definition of PCP knowledge soundness, the extractor $\mathcal{E}$ is guaranteed to output a valid witness only when $\widetilde{\pi}$ convinces the PCP verifier $\mathcal{V}_{\mathsf{PCP}}$ with high enough probability. Unfortunately, we only know that $\widetilde{\pi}$ is constructed by setting a δ fraction of the locations to 0. What if the PCP verifier queries these locations often? We cannot argue that $\mathcal{V}_{\mathsf{PCP}}$ accepts $\widetilde{\pi}$ with high probability, even if $\widetilde{\pi}$ differs only slightly from the honest proof π.
- To "confuse" the PCP extractor $\mathcal{E}$, it must be the case that $h(w) = h(w')$. Otherwise, when given $h(w)$ and $\widetilde{\pi}$, $\mathcal{E}$ would not consider outputting w', even if $\widetilde{\pi}$ is in $B_{w'}$.
- To contradict the collision resistance of the hash function, the adversary A has to be efficient. However, it is unclear how to efficiently implement Item 1 because the counting argument is existential.

PCP verifier query weights. We first argue that $\widetilde{\pi}$ still makes the PCP verifier accept with high probability. In particular, we argue that all locations modified in $\widetilde{\pi}$ are queried by the PCP verifier $\mathcal{V}_{\mathsf{PCP}}$ with small probability.

For every possible query $j \in [\ell]$ made by the PCP verifier, we define the *query weight* p_j to be the probability that $\mathcal{V}_{\mathsf{PCP}}$ queries location j. (This is a simplification of what we do in the actual proof, as this definition only makes sense for non-adaptive PCP verifiers, i.e., PCP verifiers whose queries do not depend on the proof string. We handle adaptive verifiers in the formal proof.) Let q be the query complexity of the PCP verifier. Then
$$\sum_{j \in [\ell]} p_j = \mathsf{q} \ .$$
One can modify the argument in Step 1 above to show that there exists a good restriction (w_{-i}, i) such that, for every w_i in the good assignment set Y, the set $\mathsf{O} \subseteq [\ell]$ of output locations that are written when the tape head is inside the i-th interval (or even in its neighboring intervals) satisfies
$$\sum_{j \in \mathsf{O}} p_j < \delta \mathsf{q} \ .$$
In other words, the probability that the PCP verifier $\mathcal{V}_{\mathsf{PCP}}$ queries any location in O is at most $\delta\mathsf{q}$. Since π is the honest proof that is accepted by $\mathcal{V}_{\mathsf{PCP}}$ with high probability, and with high probability $\mathcal{V}_{\mathsf{PCP}}$ cannot tell the difference between π and $\widetilde{\pi}$, we can conclude that $\mathcal{V}_{\mathsf{PCP}}$ accepts $\widetilde{\pi}$ with high probability.

Small space implies collision. Intuitively, if we set n_{out} to be much smaller than n_{in}, one can show that there must be a lot of hash collisions in expectation. In particular, when the set of good assignments $Y \subseteq \{0,1\}^b$ is large, we can show that for randomly sampled assignments $w_i \neq w'_i \in Y$, with high probability,
$$h(w) = h(w') \ .$$

However, the above approach is not compatible with the current counting argument. We still need to make sure that $\mathcal{P}_{\mathsf{PCP}}$ behaves exactly the same when running with input w and w' for $w \coloneqq w_{-i} \circ_i w_i$ and $w' \coloneqq w_{-i} \circ_i w'_i$, which is not guaranteed for randomly sampled $w_i \neq w'_i$.

Lifting to the average case. We modify the counting arguments to work for the average case. In particular, we modify the CRHF adversary A as follows:

$A(h)$:
1. Sample w uniformly at random from $\{0,1\}^{n_{\mathsf{in}}}$ and i uniformly from $[m]$.
2. Compute the honest PCP string π with the PCP prover $\mathcal{P}_{\mathsf{PCP}}$ on instance $(h, h(w))$ and witness w. Let O be the set of output locations that are written when the tape head is near the i-th interval.
3. Construct $\widetilde{\pi}$ as follows:
$$\widetilde{\pi}[j] \coloneqq \begin{cases} \pi[j] & \text{if } j \notin \mathsf{O} \\ 0 & \text{otherwise} \end{cases} .$$
4. Compute $w^\star \leftarrow \mathcal{E}((h, h(w)), \widetilde{\pi})$.
5. Output $(w, w^\star)$.

To argue that A finds a hash collision with high probability, we modify the counting arguments outlined in Steps 1 and 2 above to work for a randomly sampled w. By completeness of the PCP, we know that the PCP verifier accepts $\widetilde{\pi}$ with high probability, because the locations in O are queried with probability at most $\delta\mathsf{q}$. Therefore, according to the knowledge soundness of the PCP, $\mathcal{E}$ must output some valid witness with high probability. Observe that since $\widetilde{\pi}$ is close in Hamming distance to both π and π', intuitively it could be in either the ball B_w centered at w or the ball $B_{w'}$ centered at w'. Since w is randomly sampled, $\mathcal{E}$ outputs $w^\star \neq w$ with probability at least $1/2$. In other words, with high probability, the following holds:

$$h(w) = h(w^\star) \wedge w \neq w^\star \ .$$

2.5 The case of complexity-preserving IOPs

Interaction is known to offer significant speedups. For instance, there exist IOPs with linear-time provers [BCGGHJ17; BCG20; RR20; GLSTW23; BCL22; XZS22; RR22; BCGL22; BCFRRZ25; HR24; RW24; NST25; BMMS26; ARR25], whereas the best known asymptotic prover efficiency for PCPs remains superlinear [BS08]. Similarly, the sumcheck protocol admits linear-time and linear-space (or quasilinear-time and sublinear-space) provers in the interactive setting [CMT12; Baw+25], while its PCP realizations incur significantly worse efficiency [BFLS91; LFKN92].

We show, perhaps surprisingly, that in the sequential-access model, interaction (regardless of the round complexity) does not provide any extra advantage.

For codes, the approach from [CM24] does not directly extend to the interactive encoding setting, as interactive codes have a distance notion different from Hamming distance due to adaptivity. In fact, whether interaction helps with encoding in the sequential-access model remains an interesting open question.

To accurately model IOPs, we start by augmenting the sequential-access machine by allowing it to interact with another machine. Moreover, we even allow the IOP prover to have random access to the incoming messages. Nevertheless, this augmentation does not help as long as the access to the input tape remains sequential. We extend our proof outlined above to rule out complexity-preserving IOPs with knowledge soundness. We say that $\mathsf{IOP} = (\mathcal{P}, \mathcal{V})$ has knowledge soundness error κ if there exists an efficient extractor

$\mathcal{E}$ such that for every instance $\mathbb{x}$ and every adversary $\widetilde{\mathcal{P}}$,

$$\Pr\left[(\mathbb{x},\mathbb{w})\in R \;\middle|\; \begin{array}{l} b \xleftarrow{\widetilde{\pi}_1,\ldots,\widetilde{\pi}_{\mathsf{k}},\mathsf{vm}_1,\ldots,\mathsf{vm}_{\mathsf{k}}} \langle\widetilde{\mathcal{P}}(\mathbb{x}),\mathcal{V}(\mathbb{x})\rangle \\ \mathbb{w}\leftarrow\mathcal{E}(\mathbb{x},\widetilde{\pi}_1,\ldots,\widetilde{\pi}_{\mathsf{k}},\mathsf{vm}_1,\ldots,\mathsf{vm}_{\mathsf{k}}) \end{array}\right]$$
$$\geq \Pr\left[\langle\widetilde{\mathcal{P}}(\mathbb{x}),\mathcal{V}(\mathbb{x})\rangle=1\right]-\kappa(\mathbb{x}) \ ,$$

where k is the round complexity of IOP, $\widetilde{\pi}_i$ is the i-th proof string, and vm_i is the i-th verifier message. (Note that the IOP verifier can be private-coin.) The proof for ruling out complexity-preserving PCPs can be adapted to this setting. In particular, in the construction of the CRHF adversary A, instead of computing the honest proof directly, we now need to simulate the prover-verifier interaction as well. The counting arguments showing that A successfully finds a collision with high probability translate directly.

2.6 Space-efficient IOPs with random input access

We prove Theorem 2 with an IOP that builds on the code-based IOPs of [NA25; CFW26; BMMS26] and uses the Cook–Moshkovitz (CM) code [CM24]. An R1CS instance (Definition 3.4) consists of three sparse matrices A, B, C over a field $\mathbb{F}$, given by their nonzero entries, and asks for a vector z with first entry 1 such that $Az \circ Bz = Cz$, where $\circ$ is the entrywise product. Our IOP is for instances over a binary extension field $\mathbb{F}_{2^e}$ with $e \geq \lambda + \log N + O(1)$ and $e = O(\lambda + \log N)$, where λ is the security parameter and N is the instance size, and its prover receives the nonzero entries of each matrix sorted by row and also sorted by column. The verifier query complexity is $\mathsf{poly}(\lambda, \log N)$ bits, as in these prior IOPs. The CM code has a random-access encoder that runs in almost-linear time and polylogarithmic space. We show that if the code has a *locally computable generator*, the resulting IOP inherits the time and space complexity of the encoder.

Computing a row of the generator matrix. A recurring prover computation in these code-based IOPs is to generate the entries of a generator-matrix row $G_m[y, \cdot]$, given a message length m and a codeword position y. To perform this computation in almost-linear time and polylogarithmic space, it suffices to compute each entry $G_m[y, u]$ on demand in time $m^{o(1)}$ and space $\mathsf{polylog}(m)$, a property we call a *locally computable generator*.

An efficient encoder does not immediately give this property: encoding unit vectors generates columns of G_m, giving an $m^{2+o(1)}$-time bound for assembling a row. The standard transposition-based implementation [CFW26] generates a row in time $m^{1+o(1)}$ but uses linear space. We prove that the CM generator is locally computable (Lemma A.3).

Prover efficiency. The space efficiency of the rest of the protocol follows from the efficiency of the CM code and space-efficient implementations of basic protocols such as sumcheck [CMT12; Baw+25]. Thus random input access escapes the tradeoff established for sequential-access provers in Theorem 1. The full construction and analysis appear in Section 7.

3 Preliminaries

For integers $0 \le a \le b$, write $(a, b] \coloneqq \{a+1, \ldots, b\}$; in particular, $[b] = (0, b]$.

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

Definition 3.2. *A* **collision-resistant hash function** *is a tuple*

$$\mathsf{CRH} = (\mathsf{CRH.Gen}, \mathsf{CRH.Eval})$$

with the following syntax: (i) $\mathsf{CRH.Gen}$ *receives a security parameter* $\lambda \in \mathbb{N}$ *(in unary) and samples an evaluation key* ek*; (ii)* $\mathsf{CRH.Eval}$ *receives the evaluation key* ek *and a hash input* $w \in \{0,1\}^{n_{\mathsf{in}}(\lambda)}$ *and computes a hash output* $x \in \{0,1\}^{n_{\mathsf{out}}(\lambda)}$. CRH *has* **collision error** ϵ_{CRH} *if for every security parameter* $\lambda \in \mathbb{N}$*, adversary time* $t_{\mathsf{CRH}} \in \mathbb{N}$*, and* t_{CRH}*-time adversary* $\mathcal{A}_{\mathsf{CRH}}$*,*

$$\Pr\left[\begin{array}{l|l} w \neq w' & \mathsf{ek} \leftarrow \mathsf{CRH.Gen}(1^\lambda) \\ \wedge\, \mathsf{CRH.Eval}(\mathsf{ek}, w) = \mathsf{CRH.Eval}(\mathsf{ek}, w') & (w, w') \leftarrow \mathcal{A}_{\mathsf{CRH}}(\mathsf{ek}) \end{array}\right] \le \epsilon_{\mathsf{CRH}}(\lambda, t_{\mathsf{CRH}}).$$

Definition 3.3. *For* $\mathsf{CRH} = (\mathsf{CRH.Gen}, \mathsf{CRH.Eval})$ *as in Definition 3.2, the* CRH*-evaluation relation is*

$$R_{\mathsf{CRH}} \coloneqq \left\{ \big((\mathsf{ek}, x), w\big) : \begin{array}{l} \mathsf{ek} \in \mathsf{supp}(\mathsf{CRH.Gen}) \\ \wedge\, \mathsf{CRH.Eval}(\mathsf{ek}, w) = x \end{array} \right\}.$$

Definition 3.4. *Let* $\mathbb{F}$ *be a finite field and let* $M, N \ge 1$ *be integers. A* **rank-one constraint system (R1CS)** *over* $\mathbb{F}$ *with* M *constraints and assignment length* N *is a triple* (A, B, C) *of matrices in* $\mathbb{F}^{M \times N}$*. We index rows and columns starting at zero. The associated* **R1CS relation** *is*

$$R_{\mathsf{R1CS}} \coloneqq \left\{ (\mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathbb{x} = (\mathbb{F}, M, N, A, B, C) \text{ where } A, B, C \in \mathbb{F}^{M \times N} \\ \mathbb{w} \in \mathbb{F}^{N-1},\ z \coloneqq (1, \mathbb{w}) \\ (Az)[j] \cdot (Bz)[j] = (Cz)[j] \text{ for every } j \in \{0, \ldots, M-1\} \end{array} \right\}.$$

We refer to $z = (1, \mathbb{w})$ *as the* **assignment**.

3.1 Interactive oracle proofs

An **interactive oracle proof (IOP)** for a relation R is a pair $\mathsf{IOP} = (\mathcal{P}, \mathcal{V})$. The prover $\mathcal{P}$ receives an instance $\mathbb{x}$ and witness $\mathbb{w}$, and the verifier $\mathcal{V}$ receives $\mathbb{x}$. Both parties may be probabilistic. We use the *query-deferred* IOP syntax throughout this paper. The interaction phase consists of k rounds. In each round $i \in [\mathsf{k}]$:

- $\mathcal{P}$ commits to an immutable oracle message $\pi_i \in \Sigma^{\ell_i}$, computed from $(\mathbb{x}, \mathbb{w})$ and the preceding verifier messages $\mathsf{vm}_1, \ldots, \mathsf{vm}_{i-1}$; and
- $\mathcal{V}$ sends a message vm_i, computed from $\mathbb{x}$, its private randomness, and its current state, without querying any of the committed prover oracles.

After the interaction phase, $\mathcal{V}$ obtains oracle access to $\pi_1, \ldots, \pi_{\mathsf{k}}$, makes its possibly adaptive oracle queries, and outputs a decision bit $b \in \{0,1\}$.

We write $b \xleftarrow{\mathsf{T}} \langle \mathcal{P}(\mathbb{x}, \mathbb{w}), \mathcal{V}(\mathbb{x}) \rangle$, where $\mathsf{T} = (\pi_1, \mathsf{vm}_1, \ldots, \pi_{\mathsf{k}}, \mathsf{vm}_{\mathsf{k}})$, for the resulting decision and transcript. The transcript records the committed oracle strings for straightline extraction; the protocol verifier itself has only oracle access to them.

Definition 3.5. $\mathsf{IOP} = (\mathcal{P}, \mathcal{V})$ *has* **completeness error** α *if, for every* $(\mathbb{x}, \mathbb{w}) \in R$*,*

$$\Pr\left[\langle \mathcal{P}(\mathbb{x}, \mathbb{w}), \mathcal{V}(\mathbb{x})\rangle = 1\right] \geq 1 - \alpha(\mathbb{x}) \ .$$

Definition 3.6. $\mathsf{IOP} = (\mathcal{P}, \mathcal{V})$ *has* **straightline knowledge soundness error** κ **with extraction time** et *if there exists an extractor* $\mathcal{E}$ *that runs in time* et *such that, for every instance* $\mathbb{x}$*, auxiliary input* aux*, and malicious IOP prover* $\widetilde{\mathcal{P}}$*,*

$$\Pr\left[(\mathbb{x}, \mathbb{w}) \in R \;\middle|\; \begin{array}{l} b \xleftarrow{\mathsf{T}} \langle \widetilde{\mathcal{P}}(\mathbb{x}, \mathsf{aux}), \mathcal{V}(\mathbb{x})\rangle \\ \mathbb{w} \leftarrow \mathcal{E}(\mathbb{x}, \mathsf{T}) \end{array}\right] \geq \Pr\left[b = 1 \;\middle|\; b \leftarrow \langle \widetilde{\mathcal{P}}(\mathbb{x}, \mathsf{aux}), \mathcal{V}(\mathbb{x})\rangle\right] - \kappa(\mathbb{x}) \ .$$

We consider the following efficiency measures for $\mathsf{IOP} = (\mathcal{P}, \mathcal{V})$:

- pl denotes the number of symbols $\mathcal{P}$ writes in the prover-oracle messages $\pi_1, \ldots, \pi_{\mathsf{k}}$ (hence $\mathsf{pl} = \sum_{i \in [\mathsf{k}]} \ell_i$);
- q denotes the number of queries $\mathcal{V}$ makes to the prover-oracle messages $\pi_1, \ldots, \pi_{\mathsf{k}}$ during its query phase;
- $t_{\mathcal{V}}$ denotes the verifier's total running time, including its interaction and query phases.

Definition 3.7 (Query weights)**.** *Let* $\mathbb{x}$ *be an instance,* $\mathbb{w}$ *a witness,* ζ *a fixed prover random tape, and* ρ *a fixed full verifier random tape. Let*

$$R_{\mathcal{V}}(\mathbb{x}, \mathbb{w}; \zeta, \rho) \subseteq [\mathsf{pl}]$$

be the set of locations in the concatenated prover oracles $\pi_1, \ldots, \pi_{\mathsf{k}}$ *that* $\mathcal{V}$ *queries during the query phase of* $\langle \mathcal{P}(\mathbb{x}, \mathbb{w}; \zeta), \mathcal{V}(\mathbb{x}; \rho)\rangle$*. For each* $q \in [\mathsf{pl}]$*, define*

$$\mathsf{w}_q(\mathbb{x}, \mathbb{w}; \zeta, \rho) := \mathbf{1}\left\{q \in R_{\mathcal{V}}(\mathbb{x}, \mathbb{w}; \zeta, \rho)\right\}.$$

For every $\mathbb{x}, \mathbb{w}, \zeta, \rho$*:*

$$\sum_{q \in [\mathsf{pl}]} \mathsf{w}_q(\mathbb{x}, \mathbb{w}; \zeta, \rho) = |R_{\mathcal{V}}(\mathbb{x}, \mathbb{w}; \zeta, \rho)| \leq \mathsf{q} \ .$$

4 Sequential-access machines with interaction

We define *sequential-access machines with interaction* by adapting the definitions of interactive Turing machines in [GMR89; Can01] and sequential-access machines in [CM24].

Definition 4.1. *A* **sequential-access machine** *is specified by a tuple*

$$M = (\Gamma, \mathsf{st}_{\mathrm{i}}, \mathsf{st}_{\mathrm{w}}, \mathsf{st}_{\mathrm{s}}, \mathsf{st}_{\mathrm{h}}, h, \Sigma_{\mathsf{out}}, \tau)$$

where:

- Γ *is the finite set of states, which in particular contains the initial state* st_{i}*, the write state* st_{w}*, the send state* st_{s}*, and the halt state* st_{h}*;*
- h *is the number of tape heads on the input tape;*
- Σ_{out} *is the output-tape alphabet, which contains a special symbol* $\sqcup \in \Sigma_{\mathsf{out}}$*;*
- τ *is the transition function, which takes as input the current state* $\mathsf{st} \in \Gamma$*, the symbols read by the input tape heads, the entire contents of the randomness tape, work tape, and receive tape, and outputs the next state* $\mathsf{st}' \in \Gamma$*, the next operation in* $\{\mathsf{Left}, \mathsf{Right}, \mathsf{Jump}, \mathsf{Write}\}$*, the input to that operation, and new contents for the entire work tape.*

The machine has an input tape with alphabet $\{0,1\} \cup \{\sqcup\}$*, a randomness tape with alphabet* $\{0,1\} \cup \{\sqcup\}$*, a work tape with alphabet* $\{0,1\} \cup \{\sqcup\}$*, a receive tape with alphabet* $\{0,1\} \cup \{\sqcup\}$*, and an output tape with alphabet* Σ_{out}*.*

The aforementioned operations of M work as follows.

- Left/Right: take one head location as input and move the specified head to the left/right by one.
- Jump: take two head locations as input, and move the first head to the location of the second head.
- Write: take a symbol in Σ_{out} as input and write the symbol at the end of the output tape.

The machine M has the following types of access to its tapes.

- *Input tape:* M has read-only sequential access to the input tape with h heads.
- *Randomness tape:* M has full read access to the randomness tape (it can read its entire content).
- *Work tape:* M has full read/write access to the work tape (it can read and write its entire content).
- *Receive tape:* M has full read access to the receive tape (it can read its entire content).
- *Output tape:* M has unidirectional write access to the output tape. Every time M is in state st_{w}, one symbol is written at the end of the output tape.

The machine M can interact with an external party. Whenever M enters st_{s}, the part of the output tape written since the preceding visit to st_{s} is sent to the external party. The external party may append a finite string to the receive tape.

At each time step, M performs exactly one of the following: (i) one move or jump operation on the input tape; (ii) one write operation on the output tape. M can access the work tape, read the receive tape, or send a message together with another operation.

We use the notation $\langle M(z;\zeta), F(z)\rangle$ to denote the execution of the machine M on input z with randomness ζ, while interacting with $F(z)$. Interactive parties in this section are deterministic; fixing the full random tape of a randomized interactive party induces such a party. We consider time and space complexity.

- *Time complexity.* We consider only machines M that halt on every input. We denote by $T_{M,\zeta,F}(z)$ the running time of M in the execution $\langle M(z;\zeta), F(z)\rangle$, i.e., the number of applications of the transition function to reach st_{h}.
- *Space complexity.* A configuration of M is a tuple $\mathsf{C} = (\mathsf{st}, \mathsf{mem}, \mathsf{recv}, (\mathsf{idx}_j)_{j\in[h]}, k)$ containing the current control state, the entire work- and receive-tape contents, the input-head locations, and the number of output symbols written so far. Its size is the length of a fixed binary encoding of this tuple, including a tag that reserves one codeword for a padding symbol $\perp$. We denote by $S_{M,\zeta,F}(z)$ the space complexity of M in the execution $\langle M(z;\zeta), F(z)\rangle$, i.e., the maximum size of any configuration until halting.

Both quantities depend on the party F. In particular, the receive tape is part of the configuration, so a party that sends long messages makes the space of M large. We therefore measure the complexity of an IOP prover in executions with the honest verifier.

Definition 4.2. *Let $\mathsf{IOP} = (\mathcal{P}, \mathcal{V})$ be an IOP for a relation R. A* **sequential-access IOP prover** *for IOP is a sequential-access machine $\mathcal{P}_\star$ such that, for every instance $\mathbb{x}$, witness $\mathbb{w}$, and random tapes ζ and ρ, the machine $\mathcal{P}_\star$ on input $\mathbb{w}$ with randomness ζ, with $\mathbb{x}$ written on its work tape at the start, interacting with the party $F \coloneqq \mathcal{V}(\mathbb{x};\rho)$, sends the messages of $\mathcal{P}(\mathbb{x},\mathbb{w};\zeta)$ in $\langle \mathcal{P}(\mathbb{x},\mathbb{w};\zeta), \mathcal{V}(\mathbb{x};\rho)\rangle$. For an instance size n_{out} and a witness size n_{in}, the* **time complexity** *T and the* **space complexity** *S of $\mathcal{P}_\star$ are the maxima of $T_{\mathcal{P}_\star,\zeta,F}(\mathbb{w})$ and of $S_{\mathcal{P}_\star,\zeta,F}(\mathbb{w})$ over all $\mathbb{x} \in \{0,1\}^{n_{\mathsf{out}}}$, $\mathbb{w} \in \{0,1\}^{n_{\mathsf{in}}}$, ζ, and ρ.*

4.1 Lemmas on sequential-access machines

Definition 4.3. *The* **distance** *between two sets $A, B \subseteq [n]$ is*

$$\Delta(A,B) \coloneqq \min_{x\in A, y\in B} |x-y| \quad .$$

Definition 4.4. *For an interval size $b \in \{1,\dots,n\}$, let $m \coloneqq \lfloor n/b \rfloor$ and $I_i = (b\cdot(i-1), b\cdot i]$ for every $i \in [m]$. The positions after $b\cdot m$ remain outside these intervals.*

Definition 4.5. *Let M be a sequential-access machine, $z \in \{0,1\}^n$ an input, $\zeta \in \{0,1\}^r$ a choice of randomness, F an interactive party, and $T \coloneqq T_{M,\zeta,F}(z)$. For every $t \in [T]$, let $H_{M,\zeta,F}(z,t) \subseteq [n]$ be the set of locations of the h heads on the input tape at time t. When the fixed choices are clear, we write this set as H_t.*

Definition 4.6. *Let M be a sequential-access machine, $z \in \{0,1\}^n$ an input, $\zeta \in \{0,1\}^r$ a choice of randomness, F an interactive party, and fix an interval size $b \in [n]$. Let $T \coloneqq T_{M,\zeta,F}(z)$.*

- *The* **marking of** *$\langle M(z;\zeta), F(z)\rangle$ is the sequence $a^b_{M,\zeta,F}(z) \coloneqq (a_0, a_1, \dots, a_T)$ of strings in $\{0,1\}^m$, where $a_0 \coloneqq 0^m$ and, for every $t \in [T]$,*

$$\forall\, i \in [m],\ a_t[i] \coloneqq \begin{cases} 1 & H_t \cap I_i \neq \emptyset \\ 0 & \Delta(H_t, I_i) \geq b \\ a_{t-1}[i] & \textit{otherwise} \end{cases} \quad .$$

- *An interval I_i is* covered *at time t if $a_t[i] = 1$.*

- $\langle M(z;\zeta), F(z)\rangle$ marks *(resp.,* unmarks*)* I_i *at time* t *if* $a_t[i] = 1 \wedge a_{t-1}[i] = 0$ *(resp.,* $a_t[i] = 0 \wedge a_{t-1}[i] = 1$*). There is a marking (resp., unmarking) at time* t *if* $\langle M(z;\zeta), F(z)\rangle$ *marks (resp., unmarks) any interval at time* t*.*

Lemma 4.7. *Let* M *be a sequential-access machine,* $z \in \{0,1\}^n$ *an input,* $\zeta \in \{0,1\}^r$ *a choice of randomness,* F *an interactive party, and* $T \coloneqq T_{M,\zeta,F}(z)$*. For every interval size* $b \in \{1,\ldots,n\}$*,* $\langle M(z;\zeta), F(z)\rangle$ *makes at most* $1 + \frac{T}{b}$ *markings and at most* $1 + \frac{T}{b}$ *unmarkings.*

Proof. The lemma is the corresponding lemma of [CM24] when the machine is augmented with randomness and interaction. The proof therein carries over with only syntactic changes. □

Definition 4.8. *Let* M *be a sequential-access machine with input size* n *and randomness complexity* r*, and fix an interval size* $b \in [n]$*. For every input* $z \in \{0,1\}^n$*, random tape* $\zeta \in \{0,1\}^r$*, interactive party* F*, and* $i \in [m]$*, define*

$$\mathsf{O}^b_{M,\zeta,F}(z,i) \subseteq \mathbb{N}$$

to be the set of positions written on the output tape by M *in* $\langle M(z;\zeta), F(z)\rangle$ *when* I_i *is covered. When* M*,* ζ*,* F*, and* b *are clear, we write this set as* $\mathsf{O}(z,i)$*.*

Definition 4.9. *For every* $n \in \mathbb{N}$*,* $b \in [n]$*,* $i \in [m]$*,* $z_{-i} \in \{0,1\}^{n-b}$*, and* $z_i \in \{0,1\}^b$*, we define* $z_{-i} \circ_i z_i$ *to be the string* $z \in \{0,1\}^n$ *whose* i*-th interval is* z_i *and the rest is* z_{-i} *(filling in bits from left to right).*

Definition 4.10 (Good input)**.** *Let* M *be a sequential-access machine with input size* n *and randomness complexity* r*, and fix an interval size* $b \in [n]$*. Fix a random tape* $\zeta \in \{0,1\}^r$*, a deterministic interactive party* F*,* $u \in \mathbb{N}$*, and* $\delta \in [0,1]$*.*

Let g *map every* $z \in \{0,1\}^n$ *to the finite-support weight assignment*

$$\mathbf{w}_{M,\zeta,F,\mathsf{g}}(z) \coloneqq \mathsf{g}_{M,\zeta,F}(z) = (\mathsf{w}_j(z) \in [0,1])_{j\in\mathbb{N}}.$$

When the fixed choices are clear, we write this assignment as $\mathbf{w}(z)$*. We require* $\mathsf{w}_j(z) = 0$ *for every position* j *not written in* $\langle M(z;\zeta), F(z)\rangle$*.*

For every $z \in \{0,1\}^n$ *and* $i \in [m]$*, define*

$$G^{b,u,\delta}_{M,\zeta,F,\mathsf{g}}(z,i) \in \{0,1\}$$

to be 1 *if and only if the following conditions hold:*

(i) $\langle M(z;\zeta), F(z)\rangle$ *unmarks* I_i *at most* u *times; and*

(ii) $\sum_{j\in\mathsf{O}(z,i)} \mathsf{w}_j(z) \le \delta \cdot \sum_{j\in\mathbb{N}} \mathsf{w}_j(z)$*.*

When the fixed choices are clear, we write this indicator as $G(z,i)$*.*

Lemma 4.11. *Let* M *be a sequential-access machine with input size* $n \in \mathbb{N}$ *and randomness complexity* r*, and let* $T \in \mathbb{N}$*. For every* $b \in [n]$*,* $u \ge 1$*, and* $\delta \in (0,1)$*, define*

$$p^{b,u,\delta}_{n,h,T} \coloneqq \max\left\{0, 1 - \frac{T+b}{b\cdot u\cdot m} - \frac{3h}{m\cdot\delta}\right\}.$$

When these parameters are clear, we write this quantity as p*. For every fixed input* $z \in \{0,1\}^n$*, random tape* $\zeta \in \{0,1\}^r$*, deterministic interactive party* F *with* $T_{M,\zeta,F}(z) \le T$*, and weight function* g*, let* G *denote the indicator in Definition 4.10 for these fixed choices. Then*

$$\Pr_{i\in[m]}[G(z,i) = 1] \ge p \ .$$

Proof. We prove the lower bound for any fixed input z, random tape $\zeta \in \{0,1\}^r$, deterministic interactive party F, and weight function g.

Many unmarkings have small probability. By Lemma 4.7 and $T_{M,\zeta,F}(z) \le T$, the number of unmarkings in $\langle M(z;\zeta), F(z)\rangle$ is at most $1 + T/b$. Hence the probability that a random interval is unmarked more than u times satisfies

$$\Pr_{i\in[m]}\left[\langle M(z;\zeta), F(z)\rangle \text{ unmarks } I_i \text{ more than } u \text{ times}\right] \le \frac{T+b}{b\cdot u\cdot m} \ .$$

A random input block has small influence on writes. Let $D_{\mathsf{O},M,\zeta,F}(z) \subseteq [T]$ be the time steps at which M writes an output symbol in $\langle M(z;\zeta), F(z)\rangle$, and let j_t be the output position written at time t. When the fixed choices are clear, we write this set as D_{O}. Because the output tape is append-only, the positions j_t are distinct. Define

$$\begin{aligned} W &\coloneqq \sum_{t\in D_{\mathsf{O}}} \mathsf{w}_{j_t}(z) = \sum_{j\in\mathbb{N}} \mathsf{w}_j(z) \ , \\ L_i &\coloneqq \sum_{t\in D_{\mathsf{O}}} \mathsf{w}_{j_t}(z)\cdot \mathbf{1}\{I_i \text{ is covered at time } t\} = \sum_{j\in \mathsf{O}(z,i)} \mathsf{w}_j(z) \ . \end{aligned}$$

At any time, each input head covers at most three intervals. Therefore,

$$\begin{aligned} \mathbb{E}_{i\in[m]}[L_i] &= \frac{1}{m}\sum_{t\in D_{\mathsf{O}}} \mathsf{w}_{j_t}(z) \sum_{i\in[m]} \mathbf{1}\{I_i \text{ is covered at time } t\} \\ &\le \frac{3h}{m}\sum_{t\in D_{\mathsf{O}}} \mathsf{w}_{j_t}(z) = \frac{3h}{m} W \ . \end{aligned}$$

If $W = 0$, condition (ii) of Definition 4.10 holds for every interval. If $W > 0$, Markov's inequality gives

$$\Pr_{i\in[m]}[L_i > \delta W] \le \frac{3h}{m\cdot\delta} \ .$$

Many good intervals. Applying the union bound for the fixed input gives

$$\begin{aligned} &\Pr_{i\in[m]}[G(z,i)=1] \\ &\ge 1 - \Pr_{i\in[m]}\left[\langle M(z;\zeta), F(z)\rangle \text{ unmarks } I_i \text{ more than } u \text{ times}\right] - \Pr_{i\in[m]}[L_i > \delta W] \\ &\ge 1 - \frac{T+b}{b\cdot u\cdot m} - \frac{3h}{m\cdot\delta} \ . \end{aligned}$$

□

5 Time-space tradeoff for interactive oracle proofs

Theorem 5.1. *Let* CRH *be a collision-resistant hash function with collision error* ϵ_{CRH}*, and let* R_{CRH} *be the corresponding* CRH*-evaluation relation. Let* $\mathsf{IOP} = (\mathcal{P}, \mathcal{V})$ *be an IOP for* R_{CRH} *with completeness error* α*, straightline knowledge-soundness error* κ*, extraction time* et*, and query complexity* q*. Suppose that* $\mathcal{P}_\star$ *is a sequential-access IOP prover for* IOP *(Definition 4.2) with* h *heads, time complexity* T*, and space complexity* S*. Then, for every* $\lambda \in \mathbb{N}$*,* $b \in [n_{\mathsf{in}}]$*,* $u \geq 1$*, and* $\delta \in (0,1)$ *such that* $\delta \cdot \mathsf{q} \leq 1$*,*

$$p^{b,u,\delta}_{n_{\mathsf{in}},h,T} \cdot (2 - \delta \cdot \mathsf{q}) - 2^{S \cdot u + n_{\mathsf{out}} - b} \leq 1 + 2(\overline{\epsilon}_{\mathsf{CRH}}(\lambda) + \alpha(\lambda) + \kappa(\lambda)) \ ,$$

where

- $p^{b,u,\delta}_{n_{\mathsf{in}},h,T}$ *is as defined in Lemma 4.11;*
- $t_{\mathsf{CRH}} \coloneqq O(n_{\mathsf{in}} + t_{\mathsf{CRH.Eval}} + T + t_{\mathcal{V}} + \mathsf{et})$ *and* $\overline{\epsilon}_{\mathsf{CRH}}(\lambda) \coloneqq \epsilon_{\mathsf{CRH}}(\lambda, t_{\mathsf{CRH}})$*;*
- $\alpha(\lambda) \coloneqq \max_{\mathsf{ek} \in \mathsf{supp}(\mathsf{CRH.Gen}(1^\lambda)), x \in \{0,1\}^{n_{\mathsf{out}}(\lambda)} : (\mathsf{ek}, x) \in L(R_{\mathsf{CRH}})} \alpha((\mathsf{ek}, x))$*;*
- $\kappa(\lambda) \coloneqq \max_{\mathsf{ek} \in \mathsf{supp}(\mathsf{CRH.Gen}(1^\lambda)), x \in \{0,1\}^{n_{\mathsf{out}}(\lambda)}} \kappa((\mathsf{ek}, x))$.

Corollary 5.2. *Under the same hypotheses as Theorem 5.1, suppose that* $\mathsf{q}, S, h \geq 1$ *and*

$$n_{\mathsf{in}} \geq 768 \mathsf{q} h \big(n_{\mathsf{out}} + S + 7 \big) \ .$$

Then

$$S \cdot T \cdot h \geq \frac{649}{5{,}898{,}240} \cdot \frac{n_{\mathsf{in}}^2}{\mathsf{q}} \cdot \max \left\{ 0, 1 - \frac{2304}{649} \big(\overline{\epsilon}_{\mathsf{CRH}}(\lambda) + \kappa(\lambda) + \alpha(\lambda) \big) \right\} \ .$$

Proof. Write $p \coloneqq p^{b,u,\delta}_{n_{\mathsf{in}},h,T}$, $q \coloneqq \mathsf{q}$, and $\epsilon \coloneqq \alpha(\lambda) + \kappa(\lambda) + \overline{\epsilon}_{\mathsf{CRH}}(\lambda)$, and set $\delta \coloneqq \frac{1}{3q}$, $b \coloneqq \lfloor \frac{\delta n_{\mathsf{in}}}{64h} \rfloor$, and $u \coloneqq \lfloor \frac{b - n_{\mathsf{out}} - 7}{S} \rfloor$. The assumed bound on n_{in} gives $b \geq \frac{n_{\mathsf{in}}}{384qh}$, $m \geq 192qh$, and $u \geq \frac{b}{4S}$, and the choice of u gives $2^{S \cdot u + n_{\mathsf{out}} - b} \leq 2^{-7}$. Thus Theorem 5.1 gives $p \leq \frac{3}{5}(1 + 2\epsilon + \frac{1}{128}) = \frac{387}{640} + \frac{6}{5}\epsilon$, while Lemma 4.11, using $m \geq 64h/\delta$, gives

$$p \geq \frac{61}{64} - \frac{T}{mbu} - \frac{1}{mu} \geq \frac{61}{64} - 3072 \frac{TShq}{n_{\mathsf{in}}^2} - 8 \frac{S}{n_{\mathsf{in}}} \geq \frac{181}{192} - 3072 \frac{TShq}{n_{\mathsf{in}}^2} \ .$$

Combining the two bounds on p proves the claim. □

5.1 Proof of Theorem 5.1

Fix ek. For every hash input w, write

$$x_w \coloneqq \mathsf{CRH.Eval}(\mathsf{ek}, w).$$

When $\mathcal{P}_\star$ is run on w, it computes $\mathcal{P}((\mathsf{ek}, x_w), w)$ with ek, x_w already on the work tape. For a fixed full verifier random tape ρ, define the deterministic verifier strategy

$$\mathcal{V}^\rho_\star(w) \coloneqq \mathcal{V}((\mathsf{ek}, x_w); \rho).$$

Thus every hash input is evaluated using its corresponding public instance. The weight function g outputs the fixed-tape query indicators from Definition 3.7 for this same execution, and the affected-output set is evaluated using the same prover and verifier tapes.

We use $\mathsf{IOP} = (\mathcal{P}, \mathcal{V})$ to construct an adversary $\mathcal{A}_{\mathsf{CRH}}$ for CRH and analyze its success probability. We first construct a subroutine $\mathsf{PostProcess}$.

$\mathsf{PostProcess}(\mathsf{T}, \mathsf{O})$:

1. Parse T as $(\pi_1, \mathsf{vm}_1, \ldots, \pi_{\mathsf{k}}, \mathsf{vm}_{\mathsf{k}})$.
2. Set $(\pi'_1, \ldots, \pi'_{\mathsf{k}})$ equal to $(\pi_1, \ldots, \pi_{\mathsf{k}})$ after replacing every location in O with a fixed public symbol $\sigma_0 \in \Sigma$.
3. Let $\mathsf{T}' \coloneqq (\pi'_1, \mathsf{vm}_1, \ldots, \pi'_{\mathsf{k}}, \mathsf{vm}_{\mathsf{k}})$.
4. Output T'.

Because all IOP queries are deferred, this replacement leaves the verifier messages unchanged. The adversary $\mathcal{A}_{\mathsf{CRH}}$ is constructed as follows:

$\mathcal{A}_{\mathsf{CRH}}(\mathsf{ek})$:

1. Sample $w \leftarrow \{0,1\}^{n_{\mathsf{in}}}$ and $i \leftarrow [m]$.
2. Compute $x \coloneqq \mathsf{CRH.Eval}(\mathsf{ek}, w)$.
3. Sample independent random tapes $\zeta \leftarrow \{0,1\}^r$ for $\mathcal{P}$ and $\rho \leftarrow \{0,1\}^{\mathsf{vr}}$ for $\mathcal{V}$.
4. Simulate $\langle \mathcal{P}((\mathsf{ek}, x), w; \zeta), \mathcal{V}((\mathsf{ek}, x); \rho) \rangle$. This defines several quantities:
   - the interaction transcript $\mathsf{T} = (\pi_1, \mathsf{vm}_1, \ldots, \pi_{\mathsf{k}}, \mathsf{vm}_{\mathsf{k}})$, and
   - the locations O_i in $(\pi_1, \ldots, \pi_{\mathsf{k}})$ written in the output when I_i is covered.
5. Compute $\mathsf{T}' \coloneqq \mathsf{PostProcess}(\mathsf{T}, \mathsf{O}_i)$.
6. Compute $w' \leftarrow \mathcal{E}((\mathsf{ek}, x), \mathsf{T}')$.
7. Output (w, w').

The running time of $\mathcal{A}_{\mathsf{CRH}}$ is at most t_{CRH}: this accounts for sampling the hash input, evaluating the hash function, simulating both IOP parties and the verifier's query phase, and running the extractor in time et.

Let exp denote the experiment in which $\mathsf{ek} \leftarrow \mathsf{CRH.Gen}(1^\lambda)$ is sampled and $\mathcal{A}_{\mathsf{CRH}}(\mathsf{ek})$ is run; we aim to lower-bound the following probability:

$$\Pr_{\mathsf{exp}}[w \neq w' \wedge \mathsf{CRH.Eval}(\mathsf{ek}, w) = \mathsf{CRH.Eval}(\mathsf{ek}, w')] \ .$$

In exp we view the sampled input as $w = w_{-i} \circ_i w_i$ with $w_{-i} \leftarrow \{0,1\}^{n_{\mathsf{in}}-b}$ and $w_i \leftarrow \{0,1\}^b$ uniform and independent of i, which is the same distribution. Besides the transcript T and the set O_i, the simulation in step 4 determines the set $R_{\mathcal{V}}$ of prover-oracle locations queried by $\mathcal{V}$ during its query phase; all three are evaluated under the sampled tapes ζ and ρ, which we use only to analyze the experiment and which are not passed to the extractor. For every completion $a \in \{0,1\}^b$, write

$$w_a \coloneqq w_{-i} \circ_i a \text{ and } x_a \coloneqq \mathsf{CRH.Eval}(\mathsf{ek}, w_a) \ .$$

We use the following definitions.

- For every $w_{-i} \in \{0,1\}^{n_{\mathsf{in}}-b}$, define

$$Y(\mathsf{ek}, w_{-i}, \zeta, \rho, \mathsf{g}, i, u, \delta) \coloneqq \left\{ a \in \{0,1\}^b : G^{b,u,\delta}_{\mathcal{P}_\star, \zeta, \mathcal{V}^\rho_\star, \mathsf{g}}(w_a, i) = 1 \right\} \ ; \tag{1}$$

  the execution for each a uses the public instance (ek, x_a). For each a, the weights and affected-output set are evaluated in that same execution. When the arguments are clear, we write this set as Y.

- We also define the ordered boundary-configuration sequence

$$\mathcal{C}(\mathsf{ek}, w, \zeta, \rho, i) \coloneqq (\mathsf{C}_{t_1}, \ldots, \mathsf{C}_{t_r}, \perp, \ldots, \perp) \ , \tag{2}$$

  where $t_1 < \cdots < t_r$ are exactly the times at which $\mathcal{P}_\star$ unmarks I_i in the execution on w and its corresponding public instance, and C_{t_j} is the full configuration immediately after that unmarking. For a good input, $r \leq u$; append $\perp$ until the sequence has u entries, and encode every entry using S bits.

Let Valid, Distinct, and Ambiguous be the following events in exp:

$$\begin{aligned}
\mathsf{Valid} &:= \{\mathsf{CRH.Eval}(\mathsf{ek}, w) = \mathsf{CRH.Eval}(\mathsf{ek}, w')\} \ , \\
\mathsf{Distinct} &:= \{w \neq w'\} \ , \\
\mathsf{Ambiguous} &:= \left\{ \begin{array}{l} w_i \in Y \ \wedge \ \exists\, a' \in Y \setminus \{w_i\} : \\ x_{a'} = x_{w_i} \ \wedge \ \mathcal{C}(\mathsf{ek}, w_{a'}, \zeta, \rho, i) = \mathcal{C}(\mathsf{ek}, w, \zeta, \rho, i) \end{array} \right\} .
\end{aligned}$$

We lower-bound the target probability using the following lemmas.

Lemma 5.3. *For every $\delta \in (0,1)$ such that $\delta \cdot \mathsf{q} \leq 1$,*

$$\Pr_{\mathsf{exp}}[\mathsf{Valid}] \geq p^{b,u,\delta}_{n_{\mathsf{in}},h,T}\,(1 - \delta\mathsf{q}) - \alpha(\lambda) - \kappa(\lambda) \ .$$

Lemma 5.4.

$$\Pr_{\mathsf{exp}}[\mathsf{Ambiguous}] \geq p^{b,u,\delta}_{n_{\mathsf{in}},h,T} - 2^{Su+n_{\mathsf{out}}-b} \ .$$

Lemma 5.5.

$$\Pr_{\mathsf{exp}}[\mathsf{Distinct} \cap \mathsf{Valid} \cap \mathsf{Ambiguous}] \geq \frac{1}{2} \Pr_{\mathsf{exp}}[\mathsf{Valid} \cap \mathsf{Ambiguous}] \ .$$

By the three lemmas and $\Pr[\mathsf{Valid} \cap \mathsf{Ambiguous}] \geq \Pr[\mathsf{Valid}] + \Pr[\mathsf{Ambiguous}] - 1$,

$$\begin{aligned}
\Pr_{\mathsf{exp}}[\mathsf{Distinct} \cap \mathsf{Valid}] &\geq \frac{1}{2} \Pr_{\mathsf{exp}}[\mathsf{Valid} \cap \mathsf{Ambiguous}] \\
&\geq \frac{1}{2} \left(p^{b,u,\delta}_{n_{\mathsf{in}},h,T}(2 - \delta\mathsf{q}) - 2^{Su+n_{\mathsf{out}}-b} - 1 - \alpha(\lambda) - \kappa(\lambda) \right) \ .
\end{aligned}$$

The left-hand side is at most $\overline{\epsilon}_{\mathsf{CRH}}(\lambda)$. Rearranging, and then increasing the right-hand side, gives

$$p^{b,u,\delta}_{n_{\mathsf{in}},h,T}(2 - \delta\mathsf{q}) - 2^{Su+n_{\mathsf{out}}-b} \leq 1 + 2(\overline{\epsilon}_{\mathsf{CRH}}(\lambda) + \alpha(\lambda) + \kappa(\lambda)) \ ,$$

which proves Theorem 5.1.

5.2 Proof of Lemma 5.3

By knowledge soundness, it suffices to lower-bound the verifier's acceptance probability in the malicious execution. Consider the following instance distribution $\mathcal{D}$:

$\mathcal{D}$:
1. Sample $\mathsf{ek} \leftarrow \mathsf{CRH.Gen}(1^\lambda)$.
2. Sample $w \leftarrow \{0,1\}^{n_{\mathsf{in}}}$ and $i \leftarrow [m]$.
3. Compute $x := \mathsf{CRH.Eval}(\mathsf{ek}, w)$.
4. Output $(\mathbb{x} := (\mathsf{ek}, x), \mathsf{aux} := (w, i))$.

We construct a malicious IOP prover $\widetilde{\mathcal{P}}$ as a stateful next-oracle function. Its persistent state includes the sampled input w, the input-block index i, the honest prover's coins, and the honest prover state st_j:

- $\widetilde{\mathcal{P}}(\mathbb{x}, (w, i))$:
  1. Parse $\mathbb{x}$ as (ek, x).

2. Initialize $S_1 \coloneqq \emptyset$.
3. Simulate the first next-oracle computation of $\mathcal{P}((\mathsf{ek}, x), w)$ to obtain (π_1, st_1):
   (a) whenever $\mathcal{P}$ writes while I_i is covered, add the current round-local position in π_1 to S_1.
4. Let π_1' equal π_1 with the locations in S_1 replaced by σ_0.
5. Commit π_1' and retain (w, i, st_1).

- For $j = 2, \ldots, \mathsf{k}$, $\widetilde{\mathcal{P}}(\mathsf{vm}_{j-1}, (w, i, \mathsf{st}_{j-1}))$:
  1. Initialize $S_j \coloneqq \emptyset$.
  2. Resume the honest prover on vm_{j-1} from st_{j-1} to obtain (π_j, st_j):
     (a) whenever $\mathcal{P}$ writes while I_i is covered, add the current round-local position in π_j to S_j.
  3. Let π_j' equal π_j with the locations in S_j replaced by σ_0.
  4. Commit π_j' and retain (w, i, st_j).

The prover $\widetilde{\mathcal{P}}$ runs in time $O(T)$. Because the verifier's oracle queries are deferred until all rounds are complete, its round messages in this execution are identical to those in the corresponding honest execution with the same tapes. For every fixed tape pair, its transcript therefore equals T' in exp; averaging over the tapes gives identical transcript distributions.

By this transcript coupling and the knowledge soundness of the IOP (Definition 3.6), averaged over $\mathcal{D}$,

$$\begin{aligned}
\Pr_{\mathsf{exp}}\left[\mathsf{CRH.Eval}(\mathsf{ek}, w) = \mathsf{CRH.Eval}(\mathsf{ek}, w')\right] &= \Pr_{\mathsf{exp}}\left[((\mathsf{ek}, x), w') \in R_{\mathsf{CRH}}\right] \\
&= \Pr\left[(\mathbb{x}, w') \in R_{\mathsf{CRH}} \;\middle|\; \begin{array}{l} (\mathbb{x}, \mathsf{aux}) \leftarrow \mathcal{D} \\ b \xleftarrow{\mathsf{T}} \langle \widetilde{\mathcal{P}}(\mathbb{x}, \mathsf{aux}), \mathcal{V}(\mathbb{x})\rangle \\ w' \leftarrow \mathcal{E}(\mathbb{x}, \mathsf{T}) \end{array}\right] \\
&\geq \Pr\left[b = 1 \;\middle|\; \begin{array}{l} (\mathbb{x}, \mathsf{aux}) \leftarrow \mathcal{D} \\ b \xleftarrow{\mathsf{T}} \langle \widetilde{\mathcal{P}}(\mathbb{x}, \mathsf{aux}), \mathcal{V}(\mathbb{x})\rangle \end{array}\right] - \kappa(\lambda) \ .
\end{aligned}$$

Couple the honest and malicious executions using the same prover and verifier tapes. Their verifier messages agree under this coupling, regardless of the committed-oracle entries, because all oracle queries are deferred until after the interaction phase. During the query phase, their query–answer histories agree until the verifier first queries a location in O_i. Hence their decision bits agree whenever no such location is queried, and we deduce that

$$\begin{aligned}
\Pr\left[b = 1 \;\middle|\; \begin{array}{l} (\mathbb{x}, \mathsf{aux}) \leftarrow \mathcal{D} \\ b \xleftarrow{\mathsf{T}} \langle \widetilde{\mathcal{P}}(\mathbb{x}, \mathsf{aux}), \mathcal{V}(\mathbb{x})\rangle \end{array}\right] &\geq \Pr_{\mathsf{exp}}\left[\begin{array}{l} b = 1 \\ \wedge\, R_{\mathcal{V}} \cap \mathsf{O}_i = \emptyset \end{array}\right] \\
&= \Pr_{\mathsf{exp}}[b = 1] - \Pr_{\mathsf{exp}}\left[\begin{array}{l} b = 1 \\ \wedge\, R_{\mathcal{V}} \cap \mathsf{O}_i \neq \emptyset \end{array}\right] \\
&\geq 1 - \alpha(\lambda) - \Pr_{\mathsf{exp}}[R_{\mathcal{V}} \cap \mathsf{O}_i \neq \emptyset] \ ,
\end{aligned}$$

where the last inequality follows from completeness of the IOP (Definition 3.5). Let

$$\mathsf{Good} \coloneqq \{w_i \in Y\} \, .$$

For every fixed outcome of exp satisfying Good, the weights from Definition 3.7 are precisely the indicators of the locations in $R_{\mathcal{V}}$. In the next display we suppress their arguments, writing w_j for $\mathsf{w}_j((\mathsf{ek}, x), w; \zeta, \rho)$.

Since O_i and these weights are evaluated under the same fixed tapes,

$$\mathbf{1}\{R_{\mathcal{V}} \cap \mathsf{O}_i \neq \emptyset\} \leq \sum_{j \in \mathsf{O}_i} \mathsf{w}_j \leq \delta \sum_{j \in \mathbb{N}} \mathsf{w}_j \leq \delta \cdot \mathsf{q} \ .$$

Averaging this pointwise bound over all the randomness in exp gives

$$\begin{aligned}
\Pr_{\mathsf{exp}}\left[R_{\mathcal{V}} \cap \mathsf{O}_i \neq \emptyset\right] &\leq \delta \cdot \mathsf{q} \cdot \Pr_{\mathsf{exp}}[\mathsf{Good}] + \Pr_{\mathsf{exp}}[\neg \mathsf{Good}] \\
&= 1 - \Pr_{\mathsf{exp}}[\mathsf{Good}] \cdot (1 - \delta \cdot \mathsf{q}) \\
&\leq 1 - p^{b,u,\delta}_{n_{\mathsf{in}},h,T} \cdot (1 - \delta \cdot \mathsf{q}) \ .
\end{aligned}$$

We conclude that the target probability is lower-bounded by

$$p^{b,u,\delta}_{n_{\mathsf{in}},h,T} \cdot (1 - \delta \cdot \mathsf{q}) - \alpha(\lambda) - \kappa(\lambda) \ .$$

5.3 Proof of Lemma 5.4

Fix $\mathsf{ek}, w_{-i}, i, \zeta$, and ρ. For each $a \in Y$, write

$$\mathcal{C}_a \coloneqq \mathcal{C}(\mathsf{ek}, w_a, \zeta, \rho, i).$$

There are at most 2^{Su} possible boundary-configuration sequences. For every hash output x and boundary-configuration sequence C, define

$$W_{x,C} \coloneqq \{a \in Y : x_a = x \text{ and } \mathcal{C}_a = C\}.$$

Thus the good assignments are divided into at most $2^{n_{\mathsf{out}}+Su}$ groups. Since the sampled completion is uniform in $\{0,1\}^b$,

$$\begin{aligned}
\Pr[\mathsf{Ambiguous} \mid \mathsf{ek}, w_{-i}, i, \zeta, \rho] &= 2^{-b} \sum_{x,C} |W_{x,C}| \mathbf{1}\{|W_{x,C}| \geq 2\} \\
&= 2^{-b} \left(|Y| - \sum_{x,C} \mathbf{1}\{|W_{x,C}| = 1\} \right) \\
&\geq \frac{|Y|}{2^b} - 2^{n_{\mathsf{out}}+Su-b}.
\end{aligned}$$

Averaging over the fixed choices, we find that $\mathbb{E}[|Y|/2^b]$ is the probability that the sampled hash input is good. The sampled hash input is uniform and independent of i, so Lemma 4.11 gives a lower bound of $p^{b,u,\delta}_{n_{\mathsf{in}},h,T}$, proving the lemma.

5.4 Proof of Lemma 5.5

Use the groups $W_{x,C}$ from the preceding proof. Fix $\mathsf{ek}, w_{-i}, i, \zeta$, and ρ, and fix the extractor's randomness. Consider a group $W_{x,C}$ with at least two assignments. All assignments in this group have the same public instance and the same boundary-configuration sequence.

Because all IOP queries are deferred, the verifier messages are the same. The initial configurations agree, and the full configurations agree each time I_i is unmarked; while it is uncovered, the executions read the same input symbols and write the same symbols at the same locations. Symbols written while I_i is covered are replaced by σ_0, and the fixed prover-oracle lengths handle a final covered part. Thus all assignments in $W_{x,C}$ give the same postprocessed IOP transcript, so the extractor gives the same output for all of them.

The event Valid has the same value for all assignments in the group. If it holds, the extractor output can equal the sampled hash input for at most one assignment. Hence $\mathsf{Distinct}$ holds for at least half of the group. Summing over the groups and averaging over the fixed choices proves

$$\Pr_{\mathsf{exp}}[\mathsf{Distinct} \cap \mathsf{Valid} \cap \mathsf{Ambiguous}] \geq \frac{1}{2} \Pr_{\mathsf{exp}}[\mathsf{Valid} \cap \mathsf{Ambiguous}] \ .$$

6 Random-access machines with interaction

The time-space tradeoff of Section 5 concerns provers that access their input sequentially. In Section 7 we complement it with an IOP whose prover has *random access* to its input and runs in almost-linear time and polylogarithmic space. For that result we use the standard word RAM [BCGT13; BBHV22], in which each step is a single operation on words; in contrast, a step of the machine of Section 4 may compute an arbitrary function of its work tape. The machine interacts with an external party as in Section 4.

Definition 6.1. *A* **random-access machine with interaction** *has a word size* w*. It consists of a finite program, a program counter, and a constant number* $\mathsf{k}_{\mathsf{reg}}$ *of registers, each holding a word of* w *bits, and it has the following tapes.*

- Input tape, randomness tape, receive tape: *read-only tapes with alphabet* $\{0,1\}$*. In one step the machine can read the bit at an address held in a register.*
- Work tape: *a tape whose cells hold one word each. In one step the machine can load or store the cell at an address held in a register.*
- Output tape: *a write-only tape with alphabet* Σ_{out}*. In one step the machine can append one symbol.*

In one step the machine can also apply an arithmetic, Boolean, comparison, or shift operation to registers, branch on the content of a register, or send*. A send delivers to an external party the part of the output tape written since the previous send, and the external party may reply by appending a string to the receive tape.*

Addresses are words, so the word size must be large enough to address every tape. All our machines have word size $\mathsf{w} = O(\log(n+T))$, where n is the input length and T is the running time.

Consider the execution $\langle \mathsf{M}(z;\zeta), F(z)\rangle$ of a machine M on input z with random tape ζ, interacting with a deterministic party F. (As in Section 4, fixing the random tape of a randomized party gives a deterministic one.)

- *Time complexity.* The *time* of the execution is the number of steps until the machine halts.
- *Space complexity.* The *space* of the execution is the total number of bits in the registers, the program counter, the work-tape cells that the machine accesses, and the receive tape at the end of the execution. As in Section 4, the receive tape counts towards space, while the input, randomness, and output tapes do not.

As in Definition 4.2, we say that the prover of an IOP *runs in time* T *and space* S if every execution with the honest verifier, for every choice of the verifier's random tape, has time at most T and space at most S.

7 Space-efficient interactive oracle proofs

We construct the IOP of Theorem 2: an IOP for R1CS (Definition 3.4) whose prover is a random-access machine (Section 6) that runs in almost-linear time and polylogarithmic space, and whose verifier makes polylogarithmically many queries.

The IOP builds on recent code-based IOPs [NA25; CFW26; BMMS26]. The prover encodes the R1CS assignment with a linear code. The parties then alternate between *folding*, which halves the length of the encoded message, and *code switching*, which replaces the encoding of the folded message with a fresh one that can be folded again, until the message has constant length. Any linear code of constant rate and distance can be used. However, prior analyses consider only the prover's running time, and their provers use linear space. We identify two properties of a code that let the IOP prover run in small space: an encoder that is efficient in time and space given random access to the message, and a generator matrix each of whose entries can be computed in small time and space. We state our result for every code with these properties (Theorem 7.4). We then show that the code of Cook and Moshkovitz [CM24] has them (Theorem 7.5), which gives the IOP of Theorem 2 (Corollary 7.6).

Section 7.1 sets up notation and recalls standard tools. Section 7.2 states our results. Section 7.3 describes the IOP, Section 7.4 analyzes its efficiency, and Section 7.5 proves the main theorem of this section. The security of the IOP is analyzed in Appendices B and C, and the Cook–Moshkovitz code in Appendix A.

7.1 Notation and tools

Throughout this section, $\mathbb{F}$ is a finite field and logarithms are in base 2.

Linear codes. A *linear code* with message length m and block length K is an injective linear map $\mathcal{C}\colon \mathbb{F}^m \to \mathbb{F}^K$. Its *generator matrix* is the matrix $G \in \mathbb{F}^{K\times m}$ such that $\mathcal{C}(x) = Gx$. We index the coordinates of a message by $\{0,\dots,m-1\}$ and the positions of a codeword by $[K]$. The *relative distance* between two words f, f' of length K, over any alphabet, is

$$\mathsf{dist}(f,f') \coloneqq \frac{|\{y \in [K] : f[y] \neq f'[y]\}|}{K} \ ,$$

and the relative distance of $\mathcal{C}$ is the minimum of $\mathsf{dist}(\mathcal{C}(x),\mathcal{C}(x'))$ over all pairs of distinct messages x, x'.

We split a vector $x \in \mathbb{F}^{2m}$ into the vectors $x^{(0)}, x^{(1)} \in \mathbb{F}^m$ of its even-indexed and odd-indexed entries, that is, $x^{(b)}[v] \coloneqq x[2v+b]$. Similarly, for a word $f \in (\mathbb{F}^2)^K$, whose positions hold pairs of field elements, we write $f^{(0)}, f^{(1)} \in \mathbb{F}^K$ for the words of its first and second entries. The *interleaved code* $\mathcal{C}^{\equiv 2}$ encodes the two halves of its message separately: $\mathcal{C}^{\equiv 2}(x)$ is the word $f \in (\mathbb{F}^2)^K$ with

$$f^{(0)} = \mathcal{C}(x^{(0)}) \quad \text{and} \quad f^{(1)} = \mathcal{C}(x^{(1)}) \ .$$

The distance between two words in $(\mathbb{F}^2)^K$ counts the positions at which the pairs differ, and $\mathcal{C}^{\equiv 2}$ has the same relative distance as $\mathcal{C}$.

For $\alpha \in \mathbb{F}$, the *fold* of a vector $x \in \mathbb{F}^{2m}$ is the vector $\mathsf{Fold}(x,\alpha) \coloneqq (1-\alpha)\cdot x^{(0)} + \alpha \cdot x^{(1)} \in \mathbb{F}^m$, and the fold of a word $f \in (\mathbb{F}^2)^K$ is the word $\mathsf{Fold}(f,\alpha) \coloneqq (1-\alpha)\cdot f^{(0)} + \alpha \cdot f^{(1)} \in \mathbb{F}^K$. Folding commutes with encoding: by linearity of $\mathcal{C}$,

$$\mathsf{Fold}\big(\mathcal{C}^{\equiv 2}(x),\alpha\big) = \mathcal{C}\big(\mathsf{Fold}(x,\alpha)\big) \ . \tag{3}$$

Definition 7.1. *A* **code family** *over $\mathbb{F}$ with* smallest message length $n_\star$*, where $n_\star \geq 2$ is a power of two, is a collection $\mathcal{C} = \{\mathcal{C}_m\}_m$ of linear codes $\mathcal{C}_m\colon \mathbb{F}^m \to \mathbb{F}^{K(m)}$, one for every power of two $m \geq n_\star$. It has* rate ϱ

and relative distance δ *if, for every such* m, $K(m) \le m/\varrho$ *and* $\mathcal{C}_m$ *has relative distance at least* δ. *We denote by* $G_m \in \mathbb{F}^{K(m)\times m}$ *the generator matrix of* $\mathcal{C}_m$.

Definition 7.2. *Let* $\mathcal{C}$ *be a code family.*

- *An* **encoder** *for* $\mathcal{C}$ *with time* T_{enc} *and space* S_{enc} *is a random-access machine that, given* m *and a message* $x \in \mathbb{F}^m$ *on its input tape, writes the symbols* $\mathcal{C}_m(x)[1], \dots, \mathcal{C}_m(x)[K(m)]$ *on its output tape, in this order, in time* $T_{\mathsf{enc}}(m)$ *and space* $S_{\mathsf{enc}}(m)$.
- $\mathcal{C}$ *has a* $(T_{\mathsf{loc}}, S_{\mathsf{loc}})$**-locally computable generator** *if there is a random-access machine, which we call an* entry algorithm *for* $\mathcal{C}$, *that, given* m, *a position* $y \in [K(m)]$, *and a coordinate* $u \in \{0, \dots, m-1\}$, *outputs the entry* $G_m[y, u]$ *in time* $T_{\mathsf{loc}}(m)$ *and space* $S_{\mathsf{loc}}(m)$. *If* $T_{\mathsf{loc}}(m) = m^{o(1)}$ *and* $S_{\mathsf{loc}}(m) = \mathsf{polylog}(m)$, *we simply say that* $\mathcal{C}$ *has a* **locally computable generator**.

Machines represent an element of $\mathbb{F}$ *by* $\lceil \log|\mathbb{F}| \rceil$ *bits. We always assume that* $K(m)$ *can be computed from* m *in time* $\mathsf{polylog}(m)$, *and that* $T_{\mathsf{enc}}(m)/m$, $S_{\mathsf{enc}}(m)$, $T_{\mathsf{loc}}(m)$, *and* $S_{\mathsf{loc}}(m)$ *are nondecreasing in* m.

The encoder reads its message from an input tape, in any order. This is what the IOP prover needs: it never writes down the message that it encodes, which is a fold of the R1CS assignment, and instead answers each read of the encoder by recomputing the requested coordinate from the assignment (Lemma 7.12). The entry algorithm is needed in code switching, where both parties use rows $G_m[y, \cdot]$ of the generator matrix. The encoder does not give rows: running it on a unit vector gives a column.

Multilinear extensions. We identify an integer $u \in \{0, \dots, 2^k - 1\}$ with its binary representation $(u_1, \dots, u_k) \in \{0,1\}^k$, least significant bit first, so that $u = \sum_{j=1}^{k} 2^{j-1} u_j$. In this way, a vector $g \in \mathbb{F}^{2^k}$ is also a function $g\colon \{0,1\}^k \to \mathbb{F}$. For $q, u \in \mathbb{F}^k$ let

$$\mathrm{eq}(q, u) \coloneqq \prod_{j=1}^{k} \big(q_j u_j + (1 - q_j)(1 - u_j)\big) \ .$$

The *multilinear extension* of g is the polynomial $\tilde{g}(X) \coloneqq \sum_{u \in \{0,1\}^k} \mathrm{eq}(X, u) \cdot g[u]$ in the variables $X = (X_1, \dots, X_k)$; it is the unique multilinear polynomial that agrees with g on $\{0,1\}^k$. We write $\mathrm{eq}(q, \cdot)$ for the vector $(\mathrm{eq}(q, u))_{u \in \{0,1\}^k} \in \mathbb{F}^{2^k}$, so that $\tilde{g}(q) = \langle \mathrm{eq}(q, \cdot), g \rangle$. For a matrix $D \in \mathbb{F}^{2^k \times 2^{k'}}$ we similarly write $\tilde{D}(q, q') \coloneqq \sum_{a,u} \mathrm{eq}(q, a) \cdot \mathrm{eq}(q', u) \cdot D[a, u]$.

Folding a vector fixes the first variable of its multilinear extension: $\mathsf{Fold}(g, \alpha)[v] = \tilde{g}(\alpha, v)$ for every $v \in \{0,1\}^{k-1}$. In particular,

$$\mathsf{Fold}\big(\mathrm{eq}(q, \cdot), \alpha\big) = \mathrm{eq}(q_1, \alpha) \cdot \mathrm{eq}\big((q_2, \dots, q_k), \cdot\big) \ . \tag{4}$$

We use the Schwartz–Zippel lemma: a nonzero polynomial of total degree d vanishes at a uniformly random point with probability at most $d/|\mathbb{F}|$.

Sumcheck. Let $p \in \mathbb{F}[X_1, \dots, X_k]$ be a polynomial of degree at most d in each variable, and let $H \in \mathbb{F}$. The sumcheck protocol [LFKN92] for the equation $\sum_{u \in \{0,1\}^k} p(u) = H$ has k rounds. In round j, the prover sends a univariate polynomial p_j of degree at most d, and the verifier replies with a uniformly random $r_j \in \mathbb{F}$. The honest prover sends

$$p_j(X) \coloneqq \sum_{u \in \{0,1\}^{k-j}} p(r_1, \dots, r_{j-1}, X, u) \ .$$

The verifier checks that $p_1(0) + p_1(1) = H$ and that $p_j(0) + p_j(1) = p_{j-1}(r_{j-1})$ for $j = 2, \dots, k$. The protocol outputs the point $r \coloneqq (r_1, \dots, r_k)$ and the value $c \coloneqq p_k(r_k)$. The verifier never evaluates p. Instead, the protocol reduces the equation about the sum to the equation $p(r) = c$, which remains to be checked.

Lemma 7.3 ([LFKN92; Tha23]). *If $\sum_{u\in\{0,1\}^k} p(u) = H$ and the prover is honest, then all checks pass and $c = p(r)$. If $\sum_{u\in\{0,1\}^k} p(u) \neq H$, then, for every prover, the probability that all checks pass and $c = p(r)$ is at most $kd/|\mathbb{F}|$.*

7.2 The main theorem

R1CS instances. Consider an R1CS instance $\mathbb{x} = (\mathbb{F}, M, N, A, B, C)$ and a witness $\mathbb{w}$ as in Definition 3.4. We say that a vector $z \in \mathbb{F}^N$ is a *satisfying assignment* for $\mathbb{x}$ if $z[0] = 1$ and $Az \circ Bz = Cz$, where $\circ$ is the entrywise product. Thus $(\mathbb{x}, \mathbb{w}) \in R_{\mathsf{R1CS}}$ if and only if $z = (1, \mathbb{w})$ is a satisfying assignment. We assume that $M \geq 2$ and N are powers of two, which can be arranged by padding. Each matrix $D \in \{A, B, C\}$ is given as the list D^{row} of its nonzero entries, as triples $(a, u, D[a, u])$ sorted by row a and then by column u. We measure the size of the instance by

$$N_{\mathrm{tot}} \coloneqq N + M + Z \quad \text{where} \quad Z \coloneqq \mathrm{nnz}(A) + \mathrm{nnz}(B) + \mathrm{nnz}(C)$$

and $\mathrm{nnz}(D)$ is the number of nonzero entries of D.

The input of the prover consists of $\mathbb{x}$, $\mathbb{w}$, and, for each matrix D, the list D^{col} of the same triples sorted by column and then by row. The lists D^{col} are determined by $\mathbb{x}$, but computing them requires sorting, which cannot be done in almost-linear time and polylogarithmic space [BC82; Bea91]. We therefore include them in the prover's input, as in [BCHO22].

The IOP for general codes. The main result of this section holds for every code family that has an encoder and a locally computable generator.

Theorem 7.4. *Let $\mathcal{C}$ be a code family over $\mathbb{F}$ with smallest message length $n_\star$, rate ϱ, and relative distance δ, that has an encoder with time T_{enc} and space S_{enc} and a $(T_{\mathsf{loc}}, S_{\mathsf{loc}})$-locally computable generator. Let $\lambda \in \mathbb{N}$. Then there is a public-coin IOP for R_{R1CS}, for instances over $\mathbb{F}$ with M constraints and assignment length N that satisfy*

$$|\mathbb{F}| \geq 2^{\lambda+1} \cdot \left(\frac{N}{\varrho} + 4\log M + 2\log^2 N\right) , \tag{5}$$

with alphabet $\mathbb{F}$ and the following parameters:

- *perfect completeness;*
- *straightline knowledge soundness error is $2^{-\lambda}$, with extraction time $\mathsf{poly}(N, \log|\mathbb{F}|) \cdot T_{\mathsf{loc}}(N)$;*
- *round complexity is $O(\log M + \log^2 N)$;*
- *proof length is $O(N + \log M)$ elements of $\mathbb{F}$;*
- *query complexity is $O(\lambda \log N + \log^2 N + \log M)$;*
- *the prover is a random-access machine with input $(\mathbb{x}, \mathbb{w}, A^{\mathrm{col}}, B^{\mathrm{col}}, C^{\mathrm{col}})$ that runs in time*

$$\big(N_{\mathrm{tot}} + N \cdot T_{\mathsf{loc}}(N) + T_{\mathsf{enc}}(N)\big) \cdot \mathsf{poly}(\log|\mathbb{F}|, \log N_{\mathrm{tot}})$$

  and space $O(S_{\mathsf{enc}}(N) + S_{\mathsf{loc}}(N)) + \mathsf{poly}(\log|\mathbb{F}|, \log N_{\mathrm{tot}})$;
- *the verifier runs in time $(N_{\mathrm{tot}} + N \cdot T_{\mathsf{loc}}(N)) \cdot \mathsf{poly}(\log|\mathbb{F}|, \log N_{\mathrm{tot}})$.*

The constants hidden by $O(\cdot)$ and $\mathsf{poly}(\cdot)$ depend only on $n_\star$, ϱ, and δ.

The condition in Equation (5) bounds the knowledge soundness error: the term N/ϱ comes from the folds, and the other two terms come from the sumchecks.

The Cook–Moshkovitz code. We instantiate Theorem 7.4 with the code of [CM24]. This is a binary code, which we use over binary extension fields.

Theorem 7.5. *There are constants $\varrho, \delta > 0$ such that, for every $e \geq 1$, there is a code family $\mathsf{CM} = \{\mathsf{CM}_m\}_m$ over $\mathbb{F}_{2^e}$ with smallest message length 2, rate ϱ, and relative distance δ, such that:*

- *the generator matrices of CM have entries in $\mathbb{F}_2$ and do not depend on e;*
- *CM has an encoder with time $T_{\mathsf{enc}}(m) = m \cdot 2^{O((\log\log m)^3)} \cdot \mathsf{poly}(e)$ and space $S_{\mathsf{enc}}(m) = O(e \cdot \log^2 m)$;*
- *CM has a locally computable generator, with $T_{\mathsf{loc}}(m) = 2^{O((\log\log m)^3)}$ and $S_{\mathsf{loc}}(m) = \mathsf{polylog}(m)$.*

The code, its distance, and its encoder are due to [CM24]. The locally computable generator is new (Lemma A.3). We prove Theorem 7.5 in Appendix A.

Corollary 7.6. *There is a constant c_0 such that the following holds for all $\lambda, e \in \mathbb{N}$. There is a public-coin IOP for R_{R1CS}, for instances over $\mathbb{F}_{2^e}$ with M constraints, assignment length N, and size $N_{\mathrm{tot}} \leq 2^{e-\lambda-c_0}$, with alphabet $\{0,1\}$, perfect completeness, and straightline knowledge soundness error $2^{-\lambda}$ with extraction time $\mathsf{poly}(N_{\mathrm{tot}}, e)$, in which:*

- *the prover is a random-access machine with input $(\mathbb{x}, \mathbb{w}, A^{\mathrm{col}}, B^{\mathrm{col}}, C^{\mathrm{col}})$ that runs in time $N_{\mathrm{tot}}^{1+o(1)} \cdot \mathsf{poly}(e)$ and space $\mathsf{poly}(e, \log N_{\mathrm{tot}})$;*
- *the verifier makes $\mathsf{poly}(e, \log N_{\mathrm{tot}})$ queries and runs in time $N_{\mathrm{tot}}^{1+o(1)} \cdot \mathsf{poly}(e)$;*
- *the proof length is $O(e \cdot (N + \log M))$ bits and the round complexity is $O(\log M + \log^2 N)$.*

If $e = O(\lambda + \log N_{\mathrm{tot}})$, then every $\mathsf{poly}(e)$ above is $\mathsf{poly}(\lambda, \log N_{\mathrm{tot}})$.

Proof. Apply Theorem 7.4 to the code family of Theorem 7.5. Since $N/\varrho + 4\log M + 2\log^2 N = O(N_{\mathrm{tot}})$, the condition in Equation (5) holds once c_0 is large enough. We have $\log|\mathbb{F}| = e$, $T_{\mathsf{enc}}(N) = N^{1+o(1)} \cdot \mathsf{poly}(e)$, and $T_{\mathsf{loc}}(N) = N^{o(1)}$, which gives the stated time and space bounds. Finally, we write each element of $\mathbb{F}_{2^e}$ as e bits. This makes the alphabet binary, and multiplies the proof length and the query complexity by e. □

Theorem 2 is Corollary 7.6 with $e = \Theta(\lambda + \log N_{\mathrm{tot}})$.

7.3 The IOP

Fix a code family $\mathcal{C}$ as in Theorem 7.4. We describe the IOP for instances with assignment length $N = n_\star \cdot 2^L$, for an integer $L \geq 1$. Since N and $n_\star$ are powers of two, this only requires $N \geq 2n_\star$.

Claims. Every intermediate statement in the IOP is a linear constraint on a message that the prover has encoded.

Definition 7.7. *Let $\mathcal{C} \colon \mathbb{F}^m \to \mathbb{F}^K$ be a linear code. A* **claim** *for $\mathcal{C}^{\equiv 2}$ is a triple (f, s, μ) consisting of a word $f \in (\mathbb{F}^2)^K$, a* weight *$s \in \mathbb{F}^{2m}$, and a* target *$\mu \in \mathbb{F}$. For $\Delta \in [0,1)$, a* **Δ-witness** *for the claim is a message $x \in \mathbb{F}^{2m}$ such that*

$$\mathsf{dist}\big(f, \mathcal{C}^{\equiv 2}(x)\big) \leq \Delta \quad \textit{and} \quad \langle s, x\rangle = \mu \ .$$

Claims for $\mathcal{C}$, and their Δ-witnesses, are defined in the same way, with $f \in \mathbb{F}^K$ and $s, x \in \mathbb{F}^m$.

A claim (f, s, μ) states that f encodes a message x with $\langle s, x\rangle = \mu$. In the IOP, f is a proof string sent by the prover, or the fold of one, and s and μ are determined by the transcript. For the honest prover the message is a 0-witness: f is exactly its encoding. The verifier reads f at few positions, so against other provers we can only establish that f is close to an encoding, that is, that the claim has a Δ-witness for some $\Delta > 0$. Our analysis works for every $\Delta < \delta/3$, and we fix $\Delta := \delta/4$. Since $2\Delta < \delta$, a claim has at most one Δ-witness: two of them would give two distinct codewords within relative distance 2Δ of each other.

Levels. The IOP has levels $i = 0, \ldots, L$. Let $N_i \coloneqq N/2^i$ and, for $i < L$, let $m_i \coloneqq N_i/2$; in particular $m_{L-1} = N_L = n_\star$. The assignment at level i is the vector $z_i \in \mathbb{F}^{N_i}$ defined by

$$z_0 \coloneqq z \quad \text{and} \quad z_{i+1} \coloneqq \mathsf{Fold}(z_i, \alpha_i) \ ,$$

where $\alpha_i \in \mathbb{F}$ is a challenge sent by the verifier. At every level $i < L$ the prover sends the proof string $f_i \coloneqq \mathcal{C}^{\equiv 2}_{m_i}(z_i)$, and the parties hold a claim (f_i, s_i, μ_i) for $\mathcal{C}^{\equiv 2}_{m_i}$ of which z_i is a 0-witness. The IOP is built from three subprotocols, which we describe in turn. The first produces the claim at level 0, and the other two take a claim from one level to the next.

From R1CS to a claim. The first subprotocol reduces the R1CS instance to a claim about the encoded assignment, as in [Set20]. A sumcheck over the rows of the matrices checks the constraints $Az \circ Bz = Cz$. It leaves three linear constraints on z, with weight vectors $\tilde{A}(q_{\mathsf{row}}, \cdot)$, $\tilde{B}(q_{\mathsf{row}}, \cdot)$, and $\tilde{C}(q_{\mathsf{row}}, \cdot)$ for a random point q_{row}. These vectors have no short description, so a second sumcheck, over the columns of the matrices, reduces the three constraints to one whose weight vector is $\mathrm{eq}(q_{\mathsf{col}}, \cdot)$ for a random point q_{col}, and thus is described by the $\log N$ coordinates of q_{col}.

Construction 7.8 (R1CS reduction)**.** The prover receives $(\mathbb{x}, \mathbb{w})$ and the verifier receives $\mathbb{x}$.

1. *Encoding.* The prover sends $f_0 \coloneqq \mathcal{C}^{\equiv 2}_{m_0}(z)$.
2. *Row sumcheck.* The verifier sends $\eta \leftarrow \mathbb{F}^{\log M}$. The parties run the sumcheck protocol for the equation

$$\sum_{a \in \{0,1\}^{\log M}} \mathrm{eq}(\eta, a) \cdot \left(\widetilde{Az}(a) \cdot \widetilde{Bz}(a) - \widetilde{Cz}(a) \right) = 0 \ ,$$

   whose summand has degree at most 3 in each variable. Let q_{row} and c_{row} be the output point and value. The prover sends $\nu_D \coloneqq \widetilde{Dz}(q_{\mathsf{row}})$ for each $D \in \{A, B, C\}$. The verifier checks that $c_{\mathsf{row}} = \mathrm{eq}(\eta, q_{\mathsf{row}}) \cdot (\nu_A \nu_B - \nu_C)$.
3. *Column sumcheck.* The verifier sends $\xi_A, \xi_B, \xi_C \leftarrow \mathbb{F}$. Let $\ell \in \mathbb{F}^N$ be the vector with entries

$$\ell[u] \coloneqq \sum_{D \in \{A,B,C\}} \xi_D \cdot \tilde{D}(q_{\mathsf{row}}, u) \ .$$

   The parties run the sumcheck protocol for the equation

$$\sum_{u \in \{0,1\}^{\log N}} \tilde{\ell}(u) \cdot \tilde{z}(u) = \sum_D \xi_D \nu_D \ ,$$

   whose summand has degree at most 2 in each variable. Let q_{col} and c_{col} be the output point and value. The prover sends $\nu_z \coloneqq \tilde{z}(q_{\mathsf{col}})$. The verifier checks that $c_{\mathsf{col}} = \tilde{\ell}(q_{\mathsf{col}}) \cdot \nu_z$, where it computes $\tilde{\ell}(q_{\mathsf{col}}) = \sum_D \xi_D \cdot \tilde{D}(q_{\mathsf{row}}, q_{\mathsf{col}})$ from the lists D^{row}.
4. *Constraint* $z[0] = 1$. The verifier sends $\sigma \leftarrow \mathbb{F}$.

The output is the claim (f_0, s_0, μ_0) for $\mathcal{C}^{\equiv 2}_{m_0}$ with

$$s_0 \coloneqq \mathrm{eq}(q_{\mathsf{col}}, \cdot) + \sigma \cdot \mathrm{eq}(0^{\log N}, \cdot) \quad \text{and} \quad \mu_0 \coloneqq \nu_z + \sigma \ .$$

If $Az \circ Bz = Cz$ then the equation of the row sumcheck holds for every η, and otherwise it fails for most η. The column sumcheck uses that $\widetilde{Dz}(q_{\mathsf{row}}) = \sum_u \tilde{D}(q_{\mathsf{row}}, u) \cdot z[u]$, so that $\sum_D \xi_D \cdot \widetilde{Dz}(q_{\mathsf{row}}) = \langle \ell, z \rangle$. The last step adds the constraint $z[0] = 1$: for every vector z we have $\langle s_0, z \rangle = \tilde{z}(q_{\mathsf{col}}) + \sigma \cdot z[0]$.

Folding. At level $i < L$, folding turns the claim (f_i, s_i, μ_i) for $\mathcal{C}^{\equiv 2}_{m_i}$ into a claim for $\mathcal{C}_{m_i}$ about the folded assignment z_{i+1}. The new word and the new weight are the folds of f_i and s_i. The new target is determined by a sumcheck in a single variable, for the polynomial

$$h_i(X) \coloneqq \big\langle \mathsf{Fold}(s_i, X), \mathsf{Fold}(z_i, X) \big\rangle \ ,$$

which has degree at most 2 and satisfies $h_i(0) + h_i(1) = \langle s_i^{(0)}, z_i^{(0)} \rangle + \langle s_i^{(1)}, z_i^{(1)} \rangle = \langle s_i, z_i \rangle$.

Construction 7.9 (Fold at level i)**.** The parties hold a claim (f_i, s_i, μ_i) for $\mathcal{C}^{\equiv 2}_{m_i}$, and the prover holds z_i.
1. The prover sends the polynomial $h_i \in \mathbb{F}[X]$, of degree at most 2.
2. The verifier checks that $h_i(0) + h_i(1) = \mu_i$, and sends $\alpha_i \leftarrow \mathbb{F}$.

The output is the claim (f'_i, s'_i, μ'_i) for $\mathcal{C}_{m_i}$ with

$$f'_i \coloneqq \mathsf{Fold}(f_i, \alpha_i) \ , \quad s'_i \coloneqq \mathsf{Fold}(s_i, \alpha_i) \ , \quad \mu'_i \coloneqq h_i(\alpha_i) \ .$$

The word f'_i is not sent: whenever the verifier needs $f'_i[y]$, it queries the pair $f_i[y]$ and folds it. In an honest execution f'_i is the encoding of z_{i+1}, by Equation (3).

Code switching. Folding takes a claim for the interleaved code $\mathcal{C}^{\equiv 2}_{m_i}$ and produces a claim for $\mathcal{C}_{m_i}$, which is not interleaved, so its output cannot be folded again. Code switching restores the interleaved form: at level $i < L - 1$, it turns the claim (f'_i, s'_i, μ'_i) for $\mathcal{C}_{m_i}$ into a claim for $\mathcal{C}^{\equiv 2}_{m_{i+1}}$ about the same message z_{i+1}, which the prover encodes again in a new proof string f_{i+1}. (The name refers to the change of code.) The verifier must check that f_{i+1} encodes the same message as f'_i. It reads f'_i at a few random positions y and requires that $\mathcal{C}_{m_i}(z_{i+1})[y] = f'_i[y]$. Each requirement is a linear constraint on z_{i+1} whose coefficients are the row $G_{m_i}[y, \cdot]$ of the generator matrix. The verifier combines these constraints with random coefficients, and a sumcheck reduces the combined constraint to the value of $\tilde{z}_{i+1}$ at a random point. This is a constraint of the same form as those produced by the R1CS reduction, so the next level can fold it.

Construction 7.10 (Code switch at level i)**.** The parties hold a claim (f'_i, s'_i, μ'_i) for $\mathcal{C}_{m_i}$, and the prover holds z_{i+1}. The number t of sampled positions is a parameter.
1. The prover sends $f_{i+1} \coloneqq \mathcal{C}^{\equiv 2}_{m_{i+1}}(z_{i+1})$.
2. The verifier sends positions $y_1, \ldots, y_t \leftarrow [K(m_i)]$ and coefficients $\beta_1, \ldots, \beta_t \leftarrow \mathbb{F}$. Let

$$a_i \coloneqq \sum_{j=1}^{t} \beta_j \cdot G_{m_i}[y_j, \cdot] \in \mathbb{F}^{m_i} \quad \text{and} \quad \tau_i \coloneqq \sum_{j=1}^{t} \beta_j \cdot f'_i[y_j] \ .$$

3. The parties run the sumcheck protocol for the equation $\sum_{u \in \{0,1\}^{\log m_i}} \tilde{a}_i(u) \cdot \tilde{z}_{i+1}(u) = \tau_i$, whose summand has degree at most 2 in each variable. Let q_i and c_i be the output point and value. The prover sends $\nu_i \coloneqq \tilde{z}_{i+1}(q_i)$. The verifier checks that $c_i = \tilde{a}_i(q_i) \cdot \nu_i$, where it computes $\tilde{a}_i(q_i)$ from the entries of G_{m_i}.
4. The verifier sends $\gamma_i \leftarrow \mathbb{F}$.

The output is the claim $(f_{i+1}, s_{i+1}, \mu_{i+1})$ for $\mathcal{C}^{\equiv 2}_{m_{i+1}}$ with

$$s_{i+1} \coloneqq s'_i + \gamma_i \cdot \mathrm{eq}(q_i, \cdot) \quad \text{and} \quad \mu_{i+1} \coloneqq \mu'_i + \gamma_i \cdot \nu_i \ .$$

In an honest execution $f'_i = \mathcal{C}_{m_i}(z_{i+1})$, so $\tau_i = \langle a_i, z_{i+1} \rangle$ and the equation of the sumcheck holds.

The protocol. The IOP runs the R1CS reduction and then goes through the levels. At the last level the folded assignment has constant length, so the prover sends it and the verifier checks the last claim directly.

Construction 7.11 (The IOP)**.** The prover receives $(\mathbb{x}, \mathbb{w})$ and the verifier receives $\mathbb{x}$.

1. The parties run the R1CS reduction (Construction 7.8), which outputs the claim (f_0, s_0, μ_0).
2. For $i = 0, \ldots, L-1$: the parties run the fold at level i (Construction 7.9) and, if $i < L-1$, the code switch at level i (Construction 7.10).
3. The prover sends $z_L \in \mathbb{F}^{n_\star}$. The verifier checks that $f'_{L-1} = \mathcal{C}_{n_\star}(z_L)$ and that $\langle s'_{L-1}, z_L \rangle = \mu'_{L-1}$.
4. The verifier accepts if all checks pass.

In the third step the verifier reads all of f_{L-1}, which has constant length because $m_{L-1} = n_\star$.

All messages of the verifier are uniformly random and independent of the prover's messages. Hence the IOP is public-coin, and the verifier can postpone all its queries and checks until the interaction is over, as required by the syntax of Section 3.1. Every prover message is a proof string over $\mathbb{F}$. The verifier reads the short ones in full: these are the polynomials of the sumchecks and of the folds, the values ν, and the vector z_L.

7.4 Efficiency

We count *operations*, which are field operations and operations on indices, and *cells*, each of which stores a field element or an index. On a random-access machine, an operation takes $\mathsf{poly}(\log|\mathbb{F}|, \log N_{\mathrm{tot}})$ steps and a cell takes $O(\log|\mathbb{F}| + \log N_{\mathrm{tot}})$ bits; the bounds of Theorem 7.4 allow for these factors.

The prover cannot afford to store any of the vectors that appear in the protocol: the assignments z_i, the weights s_i, the vector ℓ, and the vectors a_i all have length up to N. Instead, it recomputes their entries whenever they are needed. The next two lemmas show that this is cheap for the assignments and for the weights.

Lemma 7.12. *For every $i \le L$ and $u \in \{0, \ldots, N_i - 1\}$,*

$$z_i[u] = \sum_{w \in \{0,1\}^i} \mathrm{eq}\big((\alpha_0, \ldots, \alpha_{i-1}), w\big) \cdot z[2^i \cdot u + w] \ .$$

Moreover, given $\alpha_0, \ldots, \alpha_{i-1}$ and random access to z, one entry of z_i can be computed with $O(2^i)$ operations and $O(i)$ cells, and all entries of z_i, in increasing order, with $O(N)$ operations and $O(i)$ cells.

Proof. The formula holds for $i = 0$, and it follows for $i+1$ from $z_{i+1}[u] = (1-\alpha_i) \cdot z_i[2u] + \alpha_i \cdot z_i[2u+1]$. To compute an entry, the prover goes over $w = 0, \ldots, 2^i - 1$ and adds up the terms. It maintains the i prefix products of the factors of $\mathrm{eq}((\alpha_0, \ldots, \alpha_{i-1}), w)$, ordered from the most significant bit of w. Moving from w to $w+1$ changes only the prefix products that involve the bits that flip, and there are $O(1)$ of these on average, as in a binary counter. □

Lemma 7.13. *For every $i < L$, the weight s_i is a linear combination of $i+2$ vectors of the form $\mathrm{eq}(q, \cdot)$ with $q \in \mathbb{F}^{\log N_i}$, and s'_i is a linear combination of $i+2$ such vectors with $q \in \mathbb{F}^{\log m_i}$. The points and the coefficients are determined by the verifier's messages, and each fold updates them with $O(i)$ operations.*

Proof. The weight s_0 has two terms. By Equation (4), folding maps a term $\theta \cdot \mathrm{eq}(q, \cdot)$ to the term $\theta \cdot \mathrm{eq}(q_1, \alpha_i) \cdot \mathrm{eq}((q_2, q_3, \ldots), \cdot)$, and code switching adds the term $\gamma_i \cdot \mathrm{eq}(q_i, \cdot)$. □

By Lemma 7.13, each party can store a weight in $O(\log^2 N)$ cells and compute any of its entries with $O(\log^2 N)$ operations.

Sumcheck in small space. In all our uses of the sumcheck protocol, the polynomial p is a fixed polynomial in $O(1)$ multilinear extensions, and the honest prover can be implemented with one pass per round over the

underlying vectors. This one-pass prover is due to [CMT12]; see also [BCHO22; Baw+25]. Lemma 7.14 states it in our cost model; the proof is short, so we include it. For vectors $g_1, \dots, g_n \in \mathbb{F}^{2^k}$, a *scan* is a procedure that outputs the tuples $(g_1[u], \dots, g_n[u])$ for $u = 0, 1, \dots, 2^k - 1$, in this order.

Lemma 7.14. *Suppose that $|\mathbb{F}| \geq 4$ and that $p = P(\tilde{g}_1, \dots, \tilde{g}_n)$, where $g_1, \dots, g_n \in \mathbb{F}^{2^k}$ and P is a fixed polynomial in $n = O(1)$ variables of total degree at most 3. Given a scan of $g_1, \dots, g_n$, the honest sumcheck prover for p can be implemented with k scans, $O(k \cdot 2^k)$ further operations, and $O(k)$ further cells. Within the same bounds, it also obtains the values $\tilde{g}_1(r), \dots, \tilde{g}_n(r)$ at the output point r.*

Proof. Fix a round j and the challenges $r_1, \dots, r_{j-1}$ of the previous rounds. For $i \in [n]$, $b \in \{0,1\}$, and $w \in \{0,1\}^{k-j}$, define

$$A_i(b, w) \coloneqq \sum_{u \in \{0,1\}^{j-1}} \mathrm{eq}\big((r_1, \dots, r_{j-1}), u\big) \cdot g_i[(u, b, w)] = \tilde{g}_i(r_1, \dots, r_{j-1}, b, w) \ .$$

Since $\tilde{g}_i$ is multilinear, $\tilde{g}_i(r_1, \dots, r_{j-1}, X, w) = (1-X) \cdot A_i(0, w) + X \cdot A_i(1, w)$. Therefore the polynomial of the honest prover in round j is

$$p_j(X) = \sum_{w \in \{0,1\}^{k-j}} P\Big((1-X) \cdot A_1(0, w) + X \cdot A_1(1, w),\ \dots,\ (1-X) \cdot A_n(0, w) + X \cdot A_n(1, w)\Big) \ .$$

Since u holds the least significant bits of the index (u, b, w), a scan visits the indices with the same (b, w) consecutively, and those with $(0, w)$ immediately before those with $(1, w)$. So in one scan the prover computes, for each w in turn, the $2n$ values $A_i(b, w)$, and adds the summand for w, evaluated at four fixed points of $\mathbb{F}$, to four accumulators. It generates the coefficients $\mathrm{eq}((r_1, \dots, r_{j-1}), u)$ for $u = 0, 1, \dots$ with prefix products, as in the proof of Lemma 7.12. At the end of the scan it interpolates p_j, which has degree at most 3, from its values at the four points. Each of the k rounds thus takes one scan, $O(2^k)$ further operations, and $O(k)$ cells.

In round k there is a single w, and the prover has computed the values $A_i(b) = \tilde{g}_i(r_1, \dots, r_{k-1}, b)$ for $b \in \{0,1\}$. After receiving r_k, it obtains $\tilde{g}_i(r) = (1 - r_k) \cdot A_i(0) + r_k \cdot A_i(1)$. □

The prover. We now put these tools together.

Lemma 7.15 (Prover)**.** *The honest prover can be implemented by a random-access machine with input $(\mathbb{x}, \mathbb{w}, A^{\mathrm{col}}, B^{\mathrm{col}}, C^{\mathrm{col}})$ that runs in time*

$$\big(N_{\mathrm{tot}} + t \cdot N \cdot T_{\mathsf{loc}}(N) + T_{\mathsf{enc}}(N)\big) \cdot \mathsf{poly}(\log|\mathbb{F}|, \log N_{\mathrm{tot}})$$

and space $O(S_{\mathsf{enc}}(N) + S_{\mathsf{loc}}(N)) + t \cdot \mathsf{poly}(\log|\mathbb{F}|, \log N_{\mathrm{tot}})$.

Proof. The prover runs two other machines as subroutines: the encoder, and the entry algorithm of Definition 7.2, which computes entries $G_m[y, u]$ of the generator matrix. It keeps the work tape of such a machine in a dictionary indexed by address, inside a region of its own work tape that it reuses from one run to the next. This costs a polylogarithmic factor in time and a constant factor in space. The table below lists the cost of each part of the prover, and we justify its rows in turn. Time is in operations and space is in cells, except that the costs of the encoder and of the entry algorithm are in their own steps and bits.

	time	space
proof string f_0	$2T_{\mathsf{enc}}(m_0)$ encoder steps	$2S_{\mathsf{enc}}(m_0)$ bits
row sumcheck	$O((Z+M)\log M)$	$O(\log M)$
column sumcheck	$O((Z\log M+N)\log N)$	$O(\log N_{\mathrm{tot}})$
fold at level i	$O(N\log^2 N)$	$O(\log^2 N)$
proof string f_{i+1}	$2T_{\mathsf{enc}}(m_{i+1})$ encoder steps, each with $O(2^{i+1})$ operations	$2S_{\mathsf{enc}}(m_{i+1})$ bits and $O(i)$ cells
code switch at level i	$O((N+t\cdot m_i)\log N)$, and $t\cdot m_i\log m_i$ calls to the entry algorithm	$S_{\mathsf{loc}}(m_i)$ bits and $O(\log N)$ cells
vector z_L	$O(N)$	$O(L)$

Proof strings. To produce $f_{i+1}=\mathcal{C}^{\equiv 2}_{m_{i+1}}(z_{i+1})$, the prover simulates two copies of the encoder for $\mathcal{C}_{m_{i+1}}$, one on each half of z_{i+1}. It alternates between the two copies so that they output their symbols at the same pace, and writes each pair of symbols to the output tape. The input tapes of the two copies are not stored: whenever a copy reads from its input tape, the prover computes the entry of z_{i+1} that is read, with $O(2^{i+1})$ operations (Lemma 7.12). The proof string f_0 is produced in the same way, with reads answered by reading z.

Sumchecks of the R1CS reduction. The summand of the row sumcheck is $P(\tilde{g},\widetilde{Az},\widetilde{Bz},\widetilde{Cz})$ for the vector $g\coloneqq \mathrm{eq}(\eta,\cdot)$ and the polynomial $P(x_0,x_1,x_2,x_3)\coloneqq x_0\cdot(x_1x_2-x_3)$, so Lemma 7.14 applies. A scan goes over the rows $a=0,\dots,M-1$. It reads the three lists D^{row} sequentially and, for each triple $(a,u,D[a,u])$, reads $z[u]$ and adds $D[a,u]\cdot z[u]$ to the entry $(Dz)[a]$. It computes $\mathrm{eq}(\eta,a)$ with prefix products as in Lemma 7.12. A scan thus takes $O(Z+M)$ operations, and there are $\log M$ scans. The values ν_D are obtained from the last scan, as in the second part of Lemma 7.14. The column sumcheck is similar, for the vectors ℓ and z. A scan goes over the columns $u=0,\dots,N-1$ and reads the three lists D^{col} sequentially. For each triple $(a,u,D[a,u])$ it adds $\xi_D\cdot D[a,u]\cdot\mathrm{eq}(q_{\mathsf{row}},a)$ to $\ell[u]$, computing $\mathrm{eq}(q_{\mathsf{row}},a)$ with $O(\log M)$ operations.

Folding. The coefficients of h_i are sums, over $v\in\{0,\dots,m_i-1\}$, of products of entries of s_i and z_i at the indices $2v$ and $2v+1$. The prover computes them in one pass over the entries of z_i, which takes $O(N)$ operations by Lemma 7.12, computing each entry of s_i with $O(\log^2 N)$ operations by Lemma 7.13.

Code switching. The prover runs the sumcheck as in Lemma 7.14, for the vectors a_i and z_{i+1}. A scan produces the entries of z_{i+1} with $O(N)$ operations, and each entry $a_i[u]=\sum_j\beta_j\cdot G_{m_i}[y_j,u]$ with t calls to the entry algorithm and $O(t)$ operations. There are $\log m_i$ scans, and the value ν_i is obtained from the last one.

Total. We sum the rows of the table over the levels. Since $T_{\mathsf{enc}}(m)/m$ is nondecreasing and $m_{i+1}=N/2^{i+2}$, we have $2^{i+2}\cdot T_{\mathsf{enc}}(m_{i+1})\le T_{\mathsf{enc}}(N)$, so the proof strings cost $O(T_{\mathsf{enc}}(N)\cdot\log N)$ operations in total. Since T_{loc} is nondecreasing and $\sum_i m_i\le N$, the calls to the entry algorithm take at most $t\cdot N\log N\cdot T_{\mathsf{loc}}(N)$ steps. Everything else takes $O(N_{\mathrm{tot}}\log^3 N_{\mathrm{tot}}+t\cdot N\log N)$ operations. This gives the stated time bound. The space is the maximum over the rows of the table, plus the space for the receive tape, which holds the verifier's messages. These consist of $O(\log M+L\cdot(t+\log N))$ field elements and positions. □

Lemma 7.16 (Verifier and communication)**.** *The IOP has $O(\log M+\log^2 N)$ rounds and proof length $2N/\varrho+O(\log M+\log^2 N)$. The verifier makes $O(t\cdot\log N+\log^2 N+\log M)$ queries and runs in time*

$$\big(N_{\mathrm{tot}}+t\cdot N\cdot T_{\mathsf{loc}}(N)\big)\cdot\mathsf{poly}(\log|\mathbb{F}|,\log N_{\mathrm{tot}})\ .$$

Proof. The R1CS reduction has $\log M + \log N + O(1)$ rounds, a fold has one round, and the code switch at level i has $\log m_i + O(1)$ rounds. The proof strings $f_0, \ldots, f_{L-1}$ have $2\sum_{i<L} K(m_i) \leq 2N/\varrho$ symbols in total, because $\sum_{i<L} m_i < N$. All other prover messages are polynomials of sumchecks and folds, values ν, and the vector z_L. These have $O(\log M + \log^2 N)$ symbols in total, and the verifier reads them all. In addition, the verifier reads t positions of f_i for every $i < L - 1$, and all of f_{L-1}.

The verifier computes each $\tilde{D}(q_{\mathsf{row}}, q_{\mathsf{col}})$ from the list D^{row}, with $O(\log N_{\mathrm{tot}})$ operations per nonzero entry. In the code switch at level i it computes $\tilde{a}_i(q_i) = \sum_u \mathrm{eq}(q_i, u) \cdot \sum_j \beta_j \cdot G_{m_i}[y_j, u]$ with $t \cdot m_i$ calls to the entry algorithm. It maintains the weights with $O(L^2)$ operations (Lemma 7.13), and its checks in the last step involve vectors of constant length. All its other checks take $O(1)$ operations per symbol that it reads. □

7.5 Proof of Theorem 7.4

The IOP is Construction 7.11, and it is public-coin. For instances with $N < 2n_\star$, the prover instead sends the witness, which has constant length, and the verifier checks the constraints.

Completeness is Lemma C.1. For knowledge soundness, we set the number of sampled positions to $t = O(\lambda + \log\log N)$ as in Lemma C.2. This gives knowledge soundness error $2^{-\lambda}$ and the stated extraction time whenever Equation (5) holds. With this choice of t, the efficiency parameters follow from Lemmas 7.15 and 7.16, because Equation (5) implies $\lambda < \log|\mathbb{F}|$ and hence $t = \mathsf{poly}(\log|\mathbb{F}|, \log N_{\mathrm{tot}})$.

Acknowledgments

This work was partially supported by the Ethereum Foundation.

AI disclosure

This paper has two main results: the time-space tradeoff for IOPs (Theorem 1) and the complexity-preserving IOP construction (Theorem 2). The ideas behind both results were generated entirely by the human authors. The proof of Theorem 1 was written entirely by the human authors. The proof of Theorem 2 was written in interactive sessions between the human authors and AI tools. AI tools were also used to polish the text throughout the paper.

References

[AER24] Guillermo Angeris, Alex Evans, and Gyumin Roh. *A Note on Ligero and Logarithmic Randomness*. Cryptology ePrint Archive, Report 2024/1399. 2024. URL: https://eprint.iacr.org/2024/1399.

[ARR25] Noor Athamnah, Noga Ron-Zewi, and Ron D. Rothblum. "Linear Prover IOPs in Log Star Rounds". In: *Proceedings of the 23rd Theory of Cryptography Conference*. TCC '25. 2025, pp. 335–368. DOI: 10.1007/978-3-032-12287-2_12.

[BBHV22] Laasya Bangalore, Rishabh Bhadauria, Carmit Hazay, and Muthuramakrishnan Venkitasubramaniam. "On Black-Box Constructions of Time and Space Efficient Sublinear Arguments from Symmetric-Key Primitives". In: *Proceedings of the 20th Theory of Cryptography Conference*. TCC '22. 2022, pp. 417–446. DOI: 10.1007/978-3-031-22318-1_15.

[BC12] Nir Bitansky and Alessandro Chiesa. "Succinct Arguments from Multi-Prover Interactive Proofs and Their Efficiency Benefits". In: *Proceedings of the 32nd Annual International Cryptology Conference*. CRYPTO '12. 2012, pp. 255–272. DOI: 10.1007/978-3-642-32009-5_16.

[BC82] Allan Borodin and Stephen A. Cook. "A Time-Space Tradeoff for Sorting on a General Sequential Model of Computation". In: *SIAM Journal on Computing* 11.2 (1982), pp. 287–297. DOI: 10.1137/0211022.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data". In: *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*. STOC '13. 2013, pp. 111–120. DOI: 10.1145/2488608.2488623.

[BCFRRZ25] Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. "Blaze: Fast SNARKs from Interleaved RAA Codes". In: *Proceedings of the 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '25. 2025, pp. 123–152. DOI: 10.1007/978-3-031-91134-7_5.

[BCFW25] Benedikt Bünz, Alessandro Chiesa, Giacomo Fenzi, and William Wang. "Linear-Time Accumulation Schemes". In: *Proceedings of the 23rd Theory of Cryptography Conference*. TCC '25. 2025, pp. 369–399. DOI: 10.1007/978-3-032-12287-2_13.

[BCG20] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. "Linear-Time Arguments with Sublinear Verification from Tensor Codes". In: *Proceedings of the 18th Theory of Cryptography Conference*. TCC '20. 2020, pp. 19–46. DOI: 10.1007/978-3-030-64378-2_2.

[BCGGHJ17] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. "Linear-Time Zero-Knowledge Proofs for Arithmetic Circuit Satisfiability". In: *Proceedings of the 23rd International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '17. 2017, pp. 336–365. DOI: 10.1007/978-3-319-70700-6_12.

[BCGL22] Jonathan Bootle, Alessandro Chiesa, Ziyi Guan, and Siqi Liu. *Linear-Time Probabilistic Proofs with Sublinear Verification for Algebraic Automata Over Every Field*. Cryptology ePrint Archive, Report 2022/1056. 2022. URL: https://eprint.iacr.org/2022/1056.

[BCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. "Fast Reductions from RAMs to Delegatable Succinct Constraint Satisfaction Problems". In: *Proceedings of the 4th Innovations in Theoretical Computer Science Conference*. ITCS '13. 2013, pp. 401–414. DOI: 10.1145/2422436.2422481.

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. "Gemini: Elastic SNARKs for Diverse Environments". In: *Proceedings of the 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '22. 2022, pp. 427–457. DOI: 10.1007/978-3-031-07085-3_15.

[BCL22] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. "Zero-Knowledge IOPs with Linear-Time Prover and Polylogarithmic-Time Verifier". In: *Proceedings of the 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '22. 2022, pp. 275–304. DOI: 10.1007/978-3-031-07085-3_10.

[BCLMS21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. "Proof-Carrying Data Without Succinct Arguments". In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO '21. 2021, pp. 681–710. DOI: 10.1007/978-3-030-84242-0_24.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. "Recursive Proof Composition from Accumulation Schemes". In: *Proceedings of the 18th Theory of Cryptography Conference*. TCC '20. 2020, pp. 1–18. DOI: 10.1007/978-3-030-64378-2_1.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC '16-B. 2016, pp. 31–60. DOI: 10.1007/978-3-662-53644-5_2.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. "Checking Computations in Polylogarithmic Time". In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*. STOC '91. 1991, pp. 21–32. DOI: 10.1145/103418.103428.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. *Halo: Recursive Proof Composition Without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. 2019. URL: https://eprint.iacr.org/2019/1021.

[BHRRS20] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. "Public-Coin Zero-Knowledge Arguments with (Almost) Minimal Time and Space Overheads". In: *Proceedings of the 18th Theory of Cryptography Conference*. TCC '20. 2020, pp. 168–197. DOI: 10.1007/978-3-030-64378-2_7.

[BHRRS21] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. "Time- and Space-Efficient Arguments from Groups of Unknown Order". In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO '21. 2021, pp. 123–152. DOI: 10.1007/978-3-030-84259-8_5.

[BMMS26] Anubhav Baweja, Pratyush Mishra, Tushar Mopuri, and Matan Shtepel. "FICS and FACS: Fast IOPPs and Accumulation via Code-Switching". In: *Proceedings of the 46th Annual International Cryptology Conference*. CRYPTO '26. 2026, pp. 557–591. DOI: 10.1007/978-3-032-35424-2_19.

[BS08] Eli Ben-Sasson and Madhu Sudan. "Short PCPs with Polylog Query Complexity". In: *SIAM Journal on Computing* 38.2 (2008), pp. 551–607. DOI: 10.1137/050646445.

[BTVW14] Andrew J. Blumberg, Justin Thaler, Victor Vu, and Michael Walfish. *Verifiable Computation Using Multiple Provers*. Cryptology ePrint Archive, Report 2014/846. 2014. URL: https://eprint.iacr.org/2014/846.

[Baw+25] Anubhav Baweja, Alessandro Chiesa, Elisabetta Fedele, Giacomo Fenzi, Pratyush Mishra, Tushar Mopuri, and Andrew Zitek-Estrada. "Time-Space Trade-Offs for Sumcheck". In: *Proceedings of the 23rd Theory of Cryptography Conference*. TCC '25. 2025, pp. 37–70. DOI: 10.1007/978-3-032-12290-2_2.

[Bea91] Paul Beame. "A General Sequential Time-Space Tradeoff for Finding Unique Elements". In: *SIAM Journal on Computing* 20.2 (1991), pp. 270–277. DOI: 10.1137/0220017.

[CCGOS23] Megan Chen, Alessandro Chiesa, Tom Gur, Jack O'Connor, and Nicholas Spooner. "Proof-Carrying Data from Arithmetized Random Oracles". In: *Proceedings of the 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '23. 2023, pp. 379–404. DOI: 10.1007/978-3-031-30617-4_13.

[CCS22] Megan Chen, Alessandro Chiesa, and Nicholas Spooner. "On Succinct Non-Interactive Arguments in Relativized Worlds". In: *Proceedings of the 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '22. 2022, pp. 336–366. DOI: 10.1007/978-3-031-07085-3_12.

[CFW26] Alessandro Chiesa, Giacomo Fenzi, and Guy Weissenberg. *Zero-Knowledge IOPPs for Constrained Interleaved Codes*. Cryptology ePrint Archive, Report 2026/391. 2026. URL: https://eprint.iacr.org/2026/391.

[CM24] Joshua Cook and Dana Moshkovitz. "Explicit Time and Space Efficient Encoders Exist Only with Random Access". In: *Proceedings of the 39th Computational Complexity Conference*. CCC '24. 2024, 5:1–5:54. DOI: 10.4230/LIPIcs.CCC.2024.5.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. "Practical Verified Computation with Streaming Interactive Proofs". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS '12. 2012, pp. 90–112. DOI: 10.1145/2090236.2090245.

[CY24] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024. URL: https://github.com/hash-based-snargs-book/hash-based-snargs-book.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*. FOCS '01. 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.

[DP24] Benjamin E. Diamond and Jim Posen. "Proximity Testing with Logarithmic Randomness". In: *IACR Communications in Cryptology* 1.1 (2024). DOI: 10.62056/aksdkp10.

[FGLSS91] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. "Approximating Clique Is Almost NP-Complete (Preliminary Version)". In: *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*. FOCS '91. 1991, pp. 2–12. DOI: 10.1109/SFCS.1991.185341.

[FPP24] Cody Freitag, Omer Paneth, and Rafael Pass. "Public-Coin, Complexity-Preserving, Succinct Arguments of Knowledge for NP from Collision-Resistance". In: *Proceedings of the 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '24. 2024, pp. 112–141. DOI: 10.1007/978-3-031-58737-5_5.

[GLSTW23] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. "Brakedown: Linear-Time and Field-Agnostic SNARKs for R1CS". In: *Proceedings of the 43rd Annual International Cryptology Conference*. CRYPTO '23. 2023, pp. 193–226. DOI: 10.1007/978-3-031-38545-2_7.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18.1 (1989). Preliminary version appeared in STOC '85, pp. 186–208. DOI: 10.1137/0218012.

[HR18] Justin Holmgren and Ron D. Rothblum. "Delegating Computations with (Almost) Minimal Time and Space Overhead". In: *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '18. 2018, pp. 124–135. DOI: 10.1109/FOCS.2018.00021.

[HR24] Justin Holmgren and Ron D. Rothblum. "Linear-Size Boolean Circuits for Multiselection". In: *Proceedings of the 39th Computational Complexity Conference*. CCC '24. 2024, 11:1–11:20. DOI: 10.4230/LIPIcs.CCC.2024.11.

[KST22] Abhiram Kothapalli, Srinath T. V. Setty, and Ioanna Tzialla. "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes". In: *Proceedings of the 42nd Annual International Cryptology Conference*. CRYPTO '22. 2022, pp. 359–388. DOI: 10.1007/978-3-031-15985-5_13.

[Kil92] Joe Kilian. "A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)". In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. STOC '92. 1992, pp. 723–732. DOI: 10.1145/129712.129782.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. "Algebraic Methods for Interactive Proof Systems". In: *Journal of the ACM* 39.4 (1992), pp. 859–868. DOI: 10.1145/146585.146605.

[Mic00] Silvio Micali. "Computationally Sound Proofs". In: *SIAM Journal on Computing* 30.4 (2000). Preliminary version appeared in FOCS '94, pp. 1253–1298. DOI: 10.1137/S0097539795284959.

[NA25] Andrija Novakovic and Guillermo Angeris. *Ligerito: A Small and Concretely Fast Polynomial Commitment Scheme*. Cryptology ePrint Archive, Report 2025/1187. 2025. URL: https://eprint.iacr.org/2025/1187.

[NST25] Vineet Nair, Ashish Sharma, and Bhargav Thankey. "BrakingBase—A Linear Prover, Poly-Logarithmic Verifier, Field-Agnostic Polynomial Commitment Scheme". In: *Proceedings of the 31st International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '25. 2025, pp. 427–455. DOI: 10.1007/978-981-95-5116-3_14.

[PT25] Edward Pyne and Roei Tell. *Composing Low-Space Algorithms*. Electronic Colloquium on Computational Complexity, Report TR25-140. 2025. URL: https://eccc.weizmann.ac.il/report/2025/140/.

[RR20] Noga Ron-Zewi and Ron D. Rothblum. "Local Proofs Approaching the Witness Length (Extended Abstract)". In: *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science*. FOCS '20. 2020, pp. 846–857. DOI: 10.1109/FOCS46700.2020.00083.

[RR22] Noga Ron-Zewi and Ron D. Rothblum. "Proving as Fast as Computing: Succinct Arguments with Constant Prover Overhead". In: *Proceedings of the 54th Annual ACM Symposium on Theory of Computing*. STOC '22. 2022, pp. 1353–1363. DOI: 10.1145/3519935.3519956.

[RRR16] Omer Reingold, Ron D. Rothblum, and Guy N. Rothblum. "Constant-Round Interactive Proofs for Delegating Computation". In: *Proceedings of the 48th Annual ACM Symposium on Theory of Computing*. STOC '16. 2016, pp. 49–62. DOI: 10.1145/2897518.2897652.

[RW24] Noga Ron-Zewi and Mor Weiss. "Zero-Knowledge IOPs Approaching Witness Length". In: *Proceedings of the 44th Annual International Cryptology Conference*. CRYPTO '24. 2024, pp. 105–137. DOI: 10.1007/978-3-031-68403-6_4.

[SV77] Walter J. Savitch and Paul M. B. Vitányi. "Linear Time Simulation of Multihead Turing Machines with Head-to-Head Jumps". In: *Proceedings of the 4th International Colloquium on Automata, Languages, and Programming*. ICALP '77. 1977, pp. 453–464. DOI: 10.1007/3-540-08342-1_35.

[Set20] Srinath Setty. "Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup". In: *Proceedings of the 40th Annual International Cryptology Conference*. CRYPTO '20. 2020, pp. 704–737. DOI: 10.1007/978-3-030-56877-1_25.

[Tha23] Justin Thaler. *Proofs, Arguments, and Zero-Knowledge*. Author manuscript, version of July 18, 2023. 2023. URL: https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf.

[Val08] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *Proceedings of the 5th Theory of Cryptography Conference*. TCC '08. 2008, pp. 1–18. DOI: 10.1007/978-3-540-78524-8_1.

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. "Orion: Zero Knowledge Proof with Linear Prover Time". In: *Proceedings of the 42nd Annual International Cryptology Conference*. CRYPTO '22. 2022, pp. 299–328. DOI: 10.1007/978-3-031-15985-5_11.

A The Cook–Moshkovitz code

We prove Theorem 7.5. The code family is the binary code of Cook and Moshkovitz [CM24], used over an extension field. Its rate, distance, and encoder come directly from their work (Lemma A.2). The locally computable generator requires inspecting their construction (Lemma A.3). Theorem 7.5 follows from these two lemmas.

A.1 Rate, distance, and encoder

We call the *blocks* of a string of bits its consecutive substrings of b bits, for a constant b fixed by the following theorem.

Theorem A.1 ([CM24, Theorem 1])**.** *There is a constant b and, for every power of two $m \geq 2$, a matrix $G_m \in \mathbb{F}_2^{K(m)\times m}$, where $K(m) = O(m)$ is a multiple of b, such that for every nonzero $x \in \mathbb{F}_2^m$ at least half of the blocks of $G_m x$ are nonzero. Moreover, there is a random-access machine that, given m and x on its input tape, writes $G_m x$ on its output tape in time $m \cdot 2^{O((\log\log m)^3)}$ and space $O(\log^2 m)$.*

The encoder of [CM24] produces its output block by block [CM24, proof of Theorem 19], so it can write it on a write-only output tape. Their construction leaves a few choices open: a prime in a given range, the linear code used by a mixer, and the treatment of exceptional seeds of an extractor. We fix each of them in a canonical way (for instance, the smallest admissible prime), so that the matrices G_m are well defined. For $e \geq 1$ and every power of two $m \geq 2$, we define

$$\mathsf{CM}_m \colon \mathbb{F}_{2^e}^m \to \mathbb{F}_{2^e}^{K(m)} \quad \text{by} \quad \mathsf{CM}_m(x) \coloneqq G_m x \ .$$

Lemma A.2. *The code family $\mathsf{CM} = \{\mathsf{CM}_m\}_m$ has constant rate and relative distance at least $1/(2b)$. It has an encoder with time $m \cdot 2^{O((\log\log m)^3)} \cdot \mathsf{poly}(e)$ and space $O(e \cdot \log^2 m)$.*

Proof. The rate is constant because $K(m) = O(m)$. For the distance and the encoder, fix a basis $\theta_1, \ldots, \theta_e$ of $\mathbb{F}_{2^e}$ over $\mathbb{F}_2$, and write a message as $x = \sum_i \theta_i \cdot x_i$ with $x_i \in \mathbb{F}_2^m$. Since G_m has entries in $\mathbb{F}_2$,

$$\mathsf{CM}_m(x) = \sum_{i=1}^{e} \theta_i \cdot G_m x_i \ .$$

The elements θ_i are linearly independent over $\mathbb{F}_2$, so the entry of $\mathsf{CM}_m(x)$ at a position y is nonzero whenever $(G_m x_i)[y] = 1$ for some i. If $x \neq 0$ then $x_i \neq 0$ for some i. By Theorem A.1, at least half of the $K(m)/b$ blocks of $G_m x_i$ are nonzero, so $G_m x_i$ has at least $K(m)/(2b)$ nonzero bits. Hence $\mathsf{CM}_m(x)$ has at least $K(m)/(2b)$ nonzero entries. In particular CM_m is injective.

The encoder simulates e copies of the machine of Theorem A.1, the i-th copy on the message x_i. When this copy reads bit u of its message, the encoder reads $x[u]$ from its own input tape and returns its i-th coordinate in the basis. The encoder alternates between the copies so that they output their bits at the same pace, and it outputs the symbol $\sum_i \theta_i \cdot (G_m x_i)[y]$ once all copies have produced their bit at position y. □

A.2 The generator is locally computable

Lemma A.3. *The code family CM has a $(2^{O((\log\log m)^3)}, \mathsf{polylog}(m))$-locally computable generator.*

The entry $G_m[y, u]$ is the bit at position y of the encoding of the u-th unit vector. Running the encoder of Theorem A.1 on this vector takes time $m^{1+o(1)}$, which is too slow for a single entry. Running only the part of the encoder that produces the bit at position y is not enough either: in the construction of [CM24], some output bits are the XOR of $\Omega(m)$ message bits, and the encoder computes such a bit by enumerating all of them. For these bits we ask the question in the other direction. Instead of enumerating the message bits that feed into an output bit, we test whether the message bit u is one of them.

Proof. For m below a sufficiently large constant, the entry algorithm has the matrix G_m hardwired. Fix a larger m, let $n \coloneqq \log m$, and identify the coordinates of a message with $\{0,1\}^n$. We first recall how [CM24] construct the map $x \mapsto G_m x$, and then describe the entry algorithm.

The construction. The map is assembled from $\mathbb{F}_2$-linear maps of four kinds. The first two are called *weight fixers*: they are designed to map messages whose Hamming weight lies in a given range to strings of large weight. Here only the way in which each output bit depends on the message matters.

- *Light fixers* [CM24, Theorem 21 and Lemma 22]. For every k from 0 up to a threshold $n - O(\log^3 n)$, there is a map $\Phi_k \colon \{0,1\}^n \times \{0,1\}^{d_k} \to \{0,1\}^{n_k}$, with $d_k = O(\log^3 n)$ and $n_k \le n + d_k + O(1)$, that takes a coordinate u and a *seed* s. It is the first component of the lossless condenser of [CM24, Theorem 5] for sets of size 2^k. The k-th light fixer maps a message x to the string of 2^{n_k} bits whose bit $v \in \{0,1\}^{n_k}$ is

$$\bigoplus_{(u,s)\,:\,\Phi_k(u,s)=v} x[u] \ . \tag{6}$$

- *Heavy fixer* [CM24, Theorem 18]. It is defined by a recursion of depth $O(\log^3 n)$ over expanders of constant degree. Each of its output bits is the XOR of a list of $2^{O(\log^3 n)}$ message bits, possibly with repetitions, and the list can be enumerated by a depth-first traversal of the recursion, in time $\mathsf{polylog}(m)$ per element and space $\mathsf{polylog}(m)$ [CM24, Lemmas 42 and 43].

- *Amplifiers* [CM24, Theorem 24 and Corollary 26]. An amplifier maps a string to a string of blocks of constant length, where each block is a tuple of bits of the input. The positions of these bits are obtained by inverting an explicit extractor, in time $\mathsf{polylog}(m)$ [CM24, Lemma 25].

- *Mixers* [CM24, Theorem 19]. A mixer takes several strings of blocks, repeats each block of each string so that all strings have the same number of blocks, and, for every j, applies a fixed linear code to the tuple of their j-th blocks.

These maps are combined as follows [CM24, Lemma 29 and proof of Theorem 1]. Each light fixer is followed by an amplifier, and the resulting strings are mixed, using a linear code on $O(\log m)$ bits whose generator matrix can be computed in time $\mathsf{polylog}(m)$. The heavy fixer is also followed by an amplifier. Finally, the two strings obtained in this way are mixed with the trivial code, which concatenates its inputs.

The entry algorithm. Given y and u, the algorithm first determines, by arithmetic on indices, whether the bit of $G_m x$ at position y comes from the heavy or from the light part of the last mixer, and which bit of that part it is.

In the heavy case, the bit is a bit of the amplified heavy fixer, and hence a copy of one output bit of the heavy fixer, which the algorithm finds by inverting the extractor. The algorithm enumerates the list of message bits whose XOR is that output bit, and outputs the parity of the number of occurrences of $x[u]$ in the list.

In the light case, the bit is an output bit of the linear code used for mixing, and hence the XOR of $O(\log m)$ input bits of that code, which the algorithm finds by computing a row of the generator matrix of that code. Each of these input bits is a bit of an amplified light fixer, and hence a copy of the output bit v of the k-th light fixer, for some k and v that the algorithm finds by inverting the extractor. By Equation (6), $x[u]$ occurs in this output bit once for every seed s such that $\Phi_k(u, s) = v$. The algorithm counts these seeds by evaluating $\Phi_k(u, s)$ for all $s \in \{0,1\}^{d_k}$. It adds up the counts over the $O(\log m)$ pairs (k, v), and outputs the parity of the total.

In both cases the algorithm obtains $G_m[y, u]$ as a sum over $\mathbb{F}_2$ of $2^{O(\log^3 n)}$ terms, which it adds one at a time without storing them.

Efficiency. The heavy case takes time $2^{O(\log^3 n)}$. The light case takes $O(\log m) \cdot 2^{O(\log^3 n)}$ evaluations of the maps Φ_k. Here [CM24] bound the complexity of *inverting* the condenser, whereas we evaluate it in the forward direction, which takes time $\mathsf{poly}(n)$ and space $O(n^2)$. Indeed, the condenser is a composition of three kinds of maps on strings of $O(n)$ bits [CM24, Figure 5]. The first is a Trevisan-style extractor [CM24, Lemma 39]: for a fixed seed it is a linear map, and their inversion algorithm starts by computing its matrix in time $\mathsf{poly}(n)$ and space $O(n^2)$. The second is a condenser based on multiplicity codes [CM24, Lemmas 33–35 and Corollary 36], which evaluates a polynomial and its derivatives at a point. The third is an extractor based on hashing [CM24, Lemma 37], which computes one multiplication and one addition in $\mathbb{F}_{2^n}$. Hence the entry algorithm runs in time $2^{O(\log^3 n)} = 2^{O((\log\log m)^3)}$ and space $\mathsf{polylog}(m)$. □

B Analysis of the subprotocols

We analyze the security of the three subprotocols of Section 7.3. Throughout, $\mathcal{C}$ is a code family with relative distance δ, and Δ is a proximity parameter with $\Delta < \delta/3$. Provers are arbitrary and, without loss of generality, deterministic, so all probabilities are over the randomness of the verifier.

B.1 The R1CS reduction

The next lemma says that the output claim of the R1CS reduction cannot hold for a vector that is not a satisfying assignment. In Appendix C we apply it to the message encoded by the proof string f_0.

Lemma B.1. *Consider the R1CS reduction (Construction 7.8) on an instance $\mathbb{x}$ with M constraints and assignment length N, and let $z^\star \in \mathbb{F}^N$ be a vector that is not a satisfying assignment for $\mathbb{x}$. For every prover, the probability that all checks pass and $\langle s_0, z^\star \rangle = \mu_0$ is at most*

$$\varepsilon_{\mathsf{r1cs}} \coloneqq \frac{4\log M + 2\log N + 1}{|\mathbb{F}|} \ .$$

Proof. We distinguish two cases.

Case 1: $z^\star[0] \neq 1$. The equation $\langle s_0, z^\star \rangle = \mu_0$ reads $(\tilde{z}^\star(q_{\mathsf{col}}) - \nu_z) + \sigma \cdot (z^\star[0] - 1) = 0$. All quantities in it other than σ are fixed before σ is sampled, so it holds with probability at most $1/|\mathbb{F}|$.

Case 2: $z^\star[0] = 1$. Now $\langle s_0, z^\star \rangle = \mu_0$ if and only if $\nu_z = \tilde{z}^\star(q_{\mathsf{col}})$, and the vector $g \coloneqq Az^\star \circ Bz^\star - Cz^\star \in \mathbb{F}^M$ is nonzero. Let $p^\star$ be the summand of the row sumcheck with $z^\star$ in place of z, and consider the following events.

- E_1: $\tilde{g}(\eta) = 0$.
- E_2: $\tilde{g}(\eta) \neq 0$, the checks of the row sumcheck pass, and $c_{\mathsf{row}} = p^\star(q_{\mathsf{row}})$.

- E_3: $\nu_D \neq \widetilde{Dz^\star}(q_{\mathsf{row}})$ for some D, and $\sum_D \xi_D \nu_D = \sum_D \xi_D \cdot \widetilde{Dz^\star}(q_{\mathsf{row}})$.
- E_4: $\sum_D \xi_D \nu_D \neq \langle \ell, z^\star \rangle$, the checks of the column sumcheck pass, and $c_{\mathsf{col}} = \tilde{\ell}(q_{\mathsf{col}}) \cdot \tilde{z}^\star(q_{\mathsf{col}})$.

We have $\Pr[E_1] \leq (\log M)/|\mathbb{F}|$ by the Schwartz–Zippel lemma. The sum of $p^\star$ over $\{0,1\}^{\log M}$ is $\tilde{g}(\eta)$, so in E_2 the row sumcheck is run on a false equation, and $\Pr[E_2] \leq (3 \log M)/|\mathbb{F}|$ by Lemma 7.3. The coefficients ξ_D are sampled after the values ν_D are sent, so $\Pr[E_3] \leq 1/|\mathbb{F}|$. Since $\langle \ell, z^\star \rangle = \sum_D \xi_D \cdot \widetilde{Dz^\star}(q_{\mathsf{row}})$ is the sum of $\tilde{\ell} \cdot \tilde{z}^\star$ over $\{0,1\}^{\log N}$, in E_4 the column sumcheck is run on a false equation, and $\Pr[E_4] \leq (2 \log N)/|\mathbb{F}|$.

Suppose that none of these events occurs, all checks pass, and $\nu_z = \tilde{z}^\star(q_{\mathsf{col}})$. Then $c_{\mathsf{row}} \neq p^\star(q_{\mathsf{row}})$ because E_1 and E_2 do not occur. The verifier's check on c_{row} then implies that $\nu_D \neq \widetilde{Dz^\star}(q_{\mathsf{row}})$ for some D. Since E_3 and E_4 do not occur, $c_{\mathsf{col}} \neq \tilde{\ell}(q_{\mathsf{col}}) \cdot \tilde{z}^\star(q_{\mathsf{col}}) = \tilde{\ell}(q_{\mathsf{col}}) \cdot \nu_z$, which contradicts the verifier's check on c_{col}. Hence the probability in the statement is at most $\Pr[E_1] + \Pr[E_2] + \Pr[E_3] + \Pr[E_4] \leq \varepsilon_{\mathsf{r1cs}}$. □

B.2 Folding

Folding is sound because of a *proximity gap*: if a word of $(\mathbb{F}^2)^K$ is far from the interleaved code, then its fold is far from the code for all but a few challenges. This was shown by [DP24]; we use the bound of [AER24].

Lemma B.2 ([AER24, Section 4])**.** *Let $\mathcal{C} \colon \mathbb{F}^m \to \mathbb{F}^K$ be a linear code with relative distance δ, and let $\Delta < \delta/3$. Let $f \in (\mathbb{F}^2)^K$ be a word such that $\mathsf{dist}(f, \mathcal{C}^{\equiv 2}(x)) > \Delta$ for every $x \in \mathbb{F}^{2m}$. Then*

$$\Pr_{\alpha \leftarrow \mathbb{F}}\Big[\exists\, x' \in \mathbb{F}^m :\ \mathsf{dist}\big(\mathsf{Fold}(f,\alpha), \mathcal{C}(x')\big) \leq \Delta\Big] \leq \frac{\Delta K + 1}{|\mathbb{F}|} \ .$$

To extract, we need to go from a message whose codeword is close to the fold of a word back to a message whose codeword is close to the word itself. The following procedure does this by solving linear systems. As in [BCFW25; CFW26], it does not need a decoding algorithm for the code.

$\mathsf{Recover}_{\mathcal{C}}(f, \alpha, x')$, for a linear code $\mathcal{C} \colon \mathbb{F}^m \to \mathbb{F}^K$ with relative distance at least δ:

1. Let S be the set of positions $y \in [K]$ such that $\mathsf{Fold}(f,\alpha)[y] = \mathcal{C}(x')[y]$. If $|S| \leq (1-\delta) \cdot K$, output $\perp$.
2. For each $b \in \{0,1\}$, find $x_b \in \mathbb{F}^m$ such that $\mathcal{C}(x_b)[y] = f^{(b)}[y]$ for every $y \in S$. If there is no such x_b, output $\perp$.
3. Output the vector $x \in \mathbb{F}^{2m}$ with $x^{(0)} = x_0$ and $x^{(1)} = x_1$.

The vector x_b in the second step is unique if it exists, because two codewords of $\mathcal{C}$ that agree on more than $(1-\delta) \cdot K$ positions are equal.

The next lemma is the usual analysis of folding (see [BCFW25; CFW26]), stated for our claims.

Lemma B.3. *Consider the fold at level i (Construction 7.9) with input claim (f_i, s_i, μ_i). Let $\mathsf{Bad}_{\mathsf{fold},i}$ be the event that the check passes, the output claim has a Δ-witness x', and $\mathsf{Recover}_{\mathcal{C}_{m_i}}(f_i, \alpha_i, x')$ is not a Δ-witness for the input claim. For every prover,*

$$\Pr\big[\mathsf{Bad}_{\mathsf{fold},i}\big] \leq \varepsilon_{\mathsf{fold}}(m_i) \coloneqq \frac{\Delta \cdot K(m_i) + 2}{|\mathbb{F}|} \ .$$

Proof. Fix the prover and hence the polynomial h_i that it sends. To lighten notation we drop the index i, and we write $\mathcal{C}$ for $\mathcal{C}_{m_i}$ and K for $K(m_i)$. We distinguish two cases according to the input claim (f, s, μ).

Case 1: no codeword is close to f. Suppose that $\mathsf{dist}(f, \mathcal{C}^{\equiv 2}(x)) > \Delta$ for every x. By Lemma B.2, except with probability $(\Delta K + 1)/|\mathbb{F}|$ over α, no codeword of $\mathcal{C}$ is within relative distance Δ of $\mathsf{Fold}(f, \alpha)$. In that case the output claim has no Δ-witness, and $\mathsf{Bad}_{\mathsf{fold}}$ does not occur.

Case 2: a codeword is close to f. Suppose that $\mathsf{dist}(f, f^\star) \le \Delta$ for the codeword $f^\star := \mathcal{C}^{\equiv 2}(x^\star)$ of some message $x^\star$, which is then unique. Let $J \subseteq [K]$ be the set of positions at which f and $f^\star$ differ, so that $|J| \le \Delta K$. For every α, the words $\mathsf{Fold}(f, \alpha)$ and $\mathsf{Fold}(f^\star, \alpha)$ agree outside J, and $\mathsf{Fold}(f^\star, \alpha) = \mathcal{C}(\mathsf{Fold}(x^\star, \alpha))$ by Equation (3). Hence $\mathsf{Fold}(x^\star, \alpha)$ is the only message whose codeword is within relative distance Δ of $\mathsf{Fold}(f, \alpha)$, and so it is the only possible Δ-witness for the output claim. Let

$$h^\star(X) := \big\langle \mathsf{Fold}(s, X), \mathsf{Fold}(x^\star, X) \big\rangle \ ,$$

which is a polynomial of degree at most 2 with $h^\star(0) + h^\star(1) = \langle s, x^\star \rangle$. The message $\mathsf{Fold}(x^\star, \alpha)$ is a Δ-witness for the output claim if and only if $h^\star(\alpha) = h(\alpha)$. There are two sub-cases.

(a) $\langle s, x^\star \rangle \neq \mu$. If the check $h(0) + h(1) = \mu$ passes, then $h \neq h^\star$, and so $h(\alpha) = h^\star(\alpha)$ with probability at most $2/|\mathbb{F}|$. When $h(\alpha) \neq h^\star(\alpha)$, the output claim has no Δ-witness, and $\mathsf{Bad}_{\mathsf{fold}}$ does not occur.

(b) $\langle s, x^\star \rangle = \mu$. Then $x^\star$ is a Δ-witness for the input claim. So it suffices to show that, except with probability $\Delta K/|\mathbb{F}|$, the procedure $\mathsf{Recover}_{\mathcal{C}}$ on input $(f, \alpha, \mathsf{Fold}(x^\star, \alpha))$ outputs $x^\star$. For $y \in J$, the pairs $f[y]$ and $f^\star[y]$ differ, so

$$\mathsf{Fold}(f, \alpha)[y] - \mathsf{Fold}(f^\star, \alpha)[y] = (1 - \alpha) \cdot \big(f^{(0)}[y] - f^{\star(0)}[y]\big) + \alpha \cdot \big(f^{(1)}[y] - f^{\star(1)}[y]\big)$$

is a nonzero polynomial of degree at most 1 in α, which vanishes for at most one value of α. By a union bound, except with probability $|J|/|\mathbb{F}|$, the set S computed by $\mathsf{Recover}$ is exactly $[K] \setminus J$. In that case $|S| \ge K - \Delta K > (1 - \delta) \cdot K$, and for each $b \in \{0, 1\}$ the system in the second step is satisfied by $(x^\star)^{(b)}$, because $f^{(b)}$ and $\mathcal{C}((x^\star)^{(b)})$ agree on S. Since the solution is unique, $\mathsf{Recover}$ outputs $x^\star$.

In every case, $\mathsf{Bad}_{\mathsf{fold}}$ occurs with probability at most $(\Delta K + 2)/|\mathbb{F}|$. □

B.3 Code switching

The next lemma is the usual analysis of code switching (see [NA25; CFW26]), stated for our claims.

Lemma B.4. *Consider the code switch at level i (Construction 7.10) with input claim (f_i', s_i', μ_i'). Let $\mathsf{Bad}_{\mathsf{switch},i}$ be the event that all checks pass, the output claim has a Δ-witness x', and x' is not a Δ-witness for the input claim. For every prover,*

$$\Pr\big[\mathsf{Bad}_{\mathsf{switch},i}\big] \le \varepsilon_{\mathsf{switch}}(m_i) := (1 - \Delta)^t + \frac{2 \log m_i + 2}{|\mathbb{F}|} \ .$$

Proof. Fix the prover and hence its first message f_{i+1}. If no message x satisfies $\mathsf{dist}(f_{i+1}, \mathcal{C}^{\equiv 2}_{m_{i+1}}(x)) \le \Delta$, then the output claim has no Δ-witness and $\mathsf{Bad}_{\mathsf{switch},i}$ does not occur. Otherwise let x' be the unique such message. It is the only possible Δ-witness for the output claim, and it is fixed before the verifier sends any message of this subprotocol. Let J be the set of positions at which f_i' and $\mathcal{C}_{m_i}(x')$ differ, and consider the following events.

- E_1: $|J| > \Delta \cdot K(m_i)$, and none of the positions $y_1, \ldots, y_t$ is in J.
- E_2: some position y_j is in J, and $\tau_i = \langle a_i, x' \rangle$.
- E_3: $\tau_i \neq \langle a_i, x' \rangle$, the checks of the sumcheck pass, and $c_i = \tilde{a}_i(q_i) \cdot \tilde{x}'(q_i)$.
- E_4: $(\langle s_i', x' \rangle, \tilde{x}'(q_i)) \neq (\mu_i', \nu_i)$, and $\langle s_{i+1}, x' \rangle = \mu_{i+1}$.

The positions are uniform and independent, so $\Pr[E_1] \le (1-\Delta)^t$. Since $\langle a_i, x' \rangle = \sum_j \beta_j \cdot \mathcal{C}_{m_i}(x')[y_j]$, we have

$$\tau_i - \langle a_i, x' \rangle = \sum_{j=1}^{t} \beta_j \cdot \left(f_i'[y_j] - \mathcal{C}_{m_i}(x')[y_j] \right) \ ,$$

which is a nonzero linear form in the coefficients β_j when some y_j is in J. Hence $\Pr[E_2] \le 1/|\mathbb{F}|$. In E_3 the sumcheck is run on a false equation about the polynomial $\tilde{a}_i \cdot \tilde{x}'$, which is fixed once the positions and coefficients are sent, so $\Pr[E_3] \le (2 \log m_i)/|\mathbb{F}|$ by Lemma 7.3. Finally,

$$\langle s_{i+1}, x' \rangle - \mu_{i+1} = \left(\langle s_i', x' \rangle - \mu_i' \right) + \gamma_i \cdot \left(\tilde{x}'(q_i) - \nu_i \right) \ ,$$

which vanishes for at most one value of γ_i unless both differences are zero. Hence $\Pr[E_4] \le 1/|\mathbb{F}|$.

Suppose that none of these events occurs, all checks pass, and x' is a Δ-witness for the output claim, so that $\langle s_{i+1}, x' \rangle = \mu_{i+1}$. Since E_4 does not occur, $\langle s_i', x' \rangle = \mu_i'$ and $\nu_i = \tilde{x}'(q_i)$. The verifier's check $c_i = \tilde{a}_i(q_i) \cdot \nu_i$ then gives $c_i = \tilde{a}_i(q_i) \cdot \tilde{x}'(q_i)$, so $\tau_i = \langle a_i, x' \rangle$ because E_3 does not occur. Then no position y_j is in J because E_2 does not occur, and so $|J| \le \Delta \cdot K(m_i)$ because E_1 does not occur. Therefore x' is a Δ-witness for the input claim, and $\mathsf{Bad}_{\mathsf{switch},i}$ does not occur. □

C Completeness and knowledge soundness of the IOP

We prove that the IOP of Construction 7.11 is complete and knowledge sound.

Lemma C.1 (Completeness)**.** *The IOP has perfect completeness.*

Proof. Let $(\mathbb{x}, \mathbb{w}) \in R_{\mathsf{R1CS}}$, so that $z = (1, \mathbb{w})$ is a satisfying assignment, and consider an execution with the honest prover.

R1CS reduction. Every term of the sum in the row sumcheck is zero, so its equation holds. By Lemma 7.3, its checks pass and c_{row} is the value of the summand at q_{row}, which is $\mathrm{eq}(\eta, q_{\mathsf{row}}) \cdot (\nu_A \nu_B - \nu_C)$. The equation of the column sumcheck holds because $\langle \ell, z \rangle = \sum_D \xi_D \cdot \widetilde{Dz}(q_{\mathsf{row}})$, so its checks pass and $c_{\mathsf{col}} = \tilde{\ell}(q_{\mathsf{col}}) \cdot \nu_z$. Finally $\langle s_0, z \rangle = \tilde{z}(q_{\mathsf{col}}) + \sigma \cdot z[0] = \mu_0$, so $z_0 = z$ is a 0-witness for (f_0, s_0, μ_0).

Levels. Suppose that z_i is a 0-witness for (f_i, s_i, μ_i). In the fold, the check passes because $h_i(0) + h_i(1) = \langle s_i, z_i \rangle = \mu_i$. Moreover $f_i' = \mathcal{C}_{m_i}(z_{i+1})$ by Equation (3), and $\langle s_i', z_{i+1} \rangle = h_i(\alpha_i) = \mu_i'$ by definition of h_i, so z_{i+1} is a 0-witness for (f_i', s_i', μ_i'). In the code switch, $\tau_i = \sum_j \beta_j \cdot \mathcal{C}_{m_i}(z_{i+1})[y_j] = \langle a_i, z_{i+1} \rangle$, so the equation of the sumcheck holds. By Lemma 7.3, its checks pass and $c_i = \tilde{a}_i(q_i) \cdot \nu_i$. Finally $\langle s_{i+1}, z_{i+1} \rangle = \mu_i' + \gamma_i \cdot \tilde{z}_{i+1}(q_i) = \mu_{i+1}$, so z_{i+1} is a 0-witness for $(f_{i+1}, s_{i+1}, \mu_{i+1})$.

Last step. By induction, z_L is a 0-witness for $(f_{L-1}', s_{L-1}', \mu_{L-1}')$. This means that $f_{L-1}' = \mathcal{C}_{n_\star}(z_L)$ and $\langle s_{L-1}', z_L \rangle = \mu_{L-1}'$, which are the final checks of the verifier. □

Lemma C.2 (Knowledge soundness)**.** *Let $\Delta = \delta/4$ and $t := \lceil (\lambda + 1 + \log L)/\log(1/(1-\Delta)) \rceil$, and suppose that Equation* (5) *holds. Then the IOP has straightline knowledge soundness error $2^{-\lambda}$ (Definition 3.6), with an extractor that runs in time $\mathsf{poly}(N, \log|\mathbb{F}|) \cdot T_{\mathsf{loc}}(N)$.*

Note that $t = O(\lambda + \log\log N)$, because Δ is a constant and $L \le \log N$.

Proof. **The extractor.** The transcript contains the proof strings $f_0, \ldots, f_{L-1}$, the challenges $\alpha_0, \ldots, \alpha_{L-1}$, and the vector z_L sent by the prover.

$\mathcal{E}(\mathbb{x}, \mathsf{T})$:
1. Set $\hat{z}_L \coloneqq z_L$.
2. For $i = L-1, \ldots, 0$: compute $\hat{z}_i \coloneqq \mathsf{Recover}_{\mathcal{C}_{m_i}}(f_i, \alpha_i, \hat{z}_{i+1})$, and output $\perp$ if $\hat{z}_i = \perp$.
3. Output the last $N-1$ entries of $\hat{z}_0$.

Analysis. Fix an instance and a prover, which we may assume to be deterministic. Let $z^\star$ be the unique vector such that $\mathsf{dist}(f_0, \mathcal{C}_{m_0}^{\equiv 2}(z^\star)) \le \Delta$, if there is one. It is determined by f_0, the first message of the prover. Let $\mathsf{Bad}_{\mathsf{r1cs}}$ be the event that $z^\star$ exists and is not a satisfying assignment, all checks of the R1CS reduction pass, and $\langle s_0, z^\star \rangle = \mu_0$. Recall the events $\mathsf{Bad}_{\mathsf{fold},i}$ for $i < L$ (Lemma B.3) and $\mathsf{Bad}_{\mathsf{switch},i}$ for $i < L-1$ (Lemma B.4).

We claim that if the verifier accepts and none of these events occurs, then the extractor outputs a valid witness. To see this, we show by backward induction on i that $\hat{z}_{i+1}$ is a Δ-witness for (f'_i, s'_i, μ'_i) and that $\hat{z}_i$ is a Δ-witness for (f_i, s_i, μ_i).

- For $i = L-1$, the final checks of the verifier say that $\hat{z}_L = z_L$ is a 0-witness for $(f'_{L-1}, s'_{L-1}, \mu'_{L-1})$.
- If $\hat{z}_{i+1}$ is a Δ-witness for (f'_i, s'_i, μ'_i), then $\hat{z}_i$ is a Δ-witness for (f_i, s_i, μ_i), because the check of the fold at level i passes and $\mathsf{Bad}_{\mathsf{fold},i}$ does not occur.
- If $\hat{z}_i$ is a Δ-witness for (f_i, s_i, μ_i) and $i \ge 1$, then $\hat{z}_i$ is a Δ-witness for $(f'_{i-1}, s'_{i-1}, \mu'_{i-1})$, because the checks of the code switch at level $i-1$ pass and $\mathsf{Bad}_{\mathsf{switch},i-1}$ does not occur.

In particular $\hat{z}_0$ is a Δ-witness for (f_0, s_0, μ_0). So $z^\star$ exists and equals $\hat{z}_0$, and $\langle s_0, z^\star \rangle = \mu_0$. Since $\mathsf{Bad}_{\mathsf{r1cs}}$ does not occur, $z^\star$ is a satisfying assignment, and the extractor outputs a valid witness.

Error. We have $\Pr[\mathsf{Bad}_{\mathsf{r1cs}}] \le \varepsilon_{\mathsf{r1cs}}$ by Lemma B.1, applied to $z^\star$. Lemmas B.3 and B.4 hold for every input claim and every prover, so they bound the probability of $\mathsf{Bad}_{\mathsf{fold},i}$ and of $\mathsf{Bad}_{\mathsf{switch},i}$ conditioned on the transcript up to the beginning of the corresponding subprotocol. By a union bound, the probability that the verifier accepts and the extractor does not output a valid witness is at most

$$\varepsilon_{\mathsf{r1cs}} + \sum_{i<L} \varepsilon_{\mathsf{fold}}(m_i) + \sum_{i<L-1} \varepsilon_{\mathsf{switch}}(m_i) \le L \cdot (1-\Delta)^t + \frac{1}{|\mathbb{F}|} \cdot \left(\frac{N}{\varrho} + 4 \log M + 2 \log^2 N \right) .$$

For the inequality, note that $\Delta \cdot \sum_{i<L} K(m_i) \le \sum_{i<L} m_i/\varrho < N/\varrho$, and that the remaining terms of the numerators add up to at most $4 \log M + 2 \log^2 N$ because $\log m_i = \log N - i - 1$ and $L \le \log N - 1$. By the choice of t and by Equation (5), each of the two terms on the right-hand side is at most $2^{-\lambda-1}$. This gives knowledge soundness error $2^{-\lambda}$ in the sense of Definition 3.6.

Extraction time. The call to Recover at level i computes the generator matrix G_{m_i}, which has $K(m_i) \cdot m_i$ entries, and solves two linear systems with at most $K(m_i)$ equations in m_i unknowns by Gaussian elimination. Over all levels, this takes $O(N^2)$ calls to the entry algorithm and $O(N^3)$ operations. □