Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 96 (2024)

Constant-Round Arguments for Batch-Verification and Bounded-Space Computations from One-Way Functions

Noga Amit*
UC Berkeley

Guy N. Rothblum†
Apple

Abstract

What are the minimal cryptographic assumptions that suffice for constructing efficient argument systems, and for which tasks? Recently, Amit and Rothblum [STOC 2023] showed that one-way functions suffice for constructing constant-round arguments for bounded-depth computations. In this work we ask: what other tasks have efficient argument systems based only on one-way functions? We show two positive results:

First, we construct a new argument system for batch-verification of k UP statements (NP statements with a unique witness) for witness relations that are verifiable in depth D. Taking M to be the length of a single witness, the communication complexity is $O(\log k) \cdot (M + k \cdot D \cdot n^{\sigma})$, where $\sigma > 0$ is an arbitrarily small constant. In particular, the communication is quasi-linear in the length of *a single witness*, so long as $k < M/(D \cdot n^{\sigma})$. The number of rounds is constant and the honest prover runs in polynomial time given witnesses for all k inputs' membership in the language.

Our second result is a constant-round doubly-efficient argument system for languages in P that are computable by bounded-space Turing machines. For this class of computations, we obtain an exponential improvement in the trade-off between the number of rounds and the (exponent of the) communication complexity, compared to known unconditionally sound protocols [Reingold, Rothblum and Rothblum, STOC 2016].

*Email: nogamit@berkeley.edu.
†Email: rothblum@alum.mit.edu.

ISSN 1433-8092

Contents

1 Introduction

A proof-system allows an untrusted prover to convince a verifier that a complex claim is true. The claim is usually framed as the membership of an input x in a language $\mathcal{L}$, where verification should be more efficient than deciding membership in $\mathcal{L}$. In a computationally sound *argument system* [BCC88], soundness is relaxed to hold only against polynomial-time cheating provers, under cryptographic assumptions. If $x \notin \mathcal{L}$, then no polynomial-time cheating prover should be able to get the verifier to accept (except with small probability). Understanding the cryptographic assumptions needed to construct argument systems, and the power of these protocols — i.e., which languages have argument systems, and how efficient can these protocols be — is a central question in the foundations of cryptography.

We study this question, focusing on efficient argument systems that have a constant number of rounds. Kilian's [Kil92] celebrated work showed that, assuming the existence of collision-resistant hash functions (CRHs), every language in NP (and, in particular, every language in P) has a 4-message argument system with sublinear communication and almost-linear verification time. Kilian's result demonstrated that CRHs are sufficient for constructing powerful argument systems that go well beyond what is known for *unconditionally sound* interactive proof systems (IPs) [GMR89] (and, in some regimes, beyond what is plausible for IPs [GH98, GVW02a]). However, it was not clear whether weaker cryptographic assumptions, such as the existence of one-way functions (OWFs, often referred to as a "minimal" assumption for cryptography), suffice for constructing powerful argument systems. This question is in line with a central theme in the theoretical study of cryptography: understanding the *minimal* assumptions needed for implementing cryptographic primitives.

Recently, Amit and Rothblum [AR23] showed that one-way functions suffice for constructing efficient argument systems for the class of bounded-depth polynomial-size computations. Their arguments have a constant number of rounds, and are *doubly-efficient*: the honest prover is efficient (polynomial), and the verifier is super-efficient (almost-linear). This demonstrated that one-way functions suffice for constructing argument systems whose efficiency is beyond what is known for IPs. In this work we ask: do OWFs suffice for constructing such arguments for other tasks?

1.1 Quasi-Linear Batch Verification for UP

Our first result is an interactive argument system for verifying the correctness of k UP statements. Recall that the complexity class UP (unambiguous non-deterministic polynomial-time) is the subset of NP statements for which correct statements have a *unique* witness. For example, a prover who wants to convince a verifier that k given integers are all RSA moduli (i.e., products of equal length primes). Multiple other examples arise from cryptography, where problems related to discrete-log or lattices all have unique solutions. We focus on UP problems where the witness verification circuit is of bounded depth.

A line of work has tried to construct (unconditionally sound) interactive proofs of minimal communication complexity for UP batch verification. Under complexity assumptions, it is known that the communication cannot be less than the length of a single witness [GH98, GVW02b], and thus the best one can aim for is linear or quasi-linear communication.[1] Known interactive proofs that achieve quasi-linear communication require polylogarithmically many rounds [RR20]. It is

[1] We refer the reader to [RRR16, Remark 2.2] for a through discussion.

natural to ask whether it is possible to achieve quasi-linear communication using only a *constant* number of rounds. We answer this question in the affirmative, assuming only the existence of OWF:

Theorem 1.1 (Batch Verification for UP)**.** *Assume one-way functions exist, and let $\sigma \in (0,1)$ be a constant. Let $\mathcal{L}$ be a language in* UP *with inputs of length n and witnesses of length $M = M(n) = poly(n)$, whose witness relation can be computed by* Logspace*-uniform circuits of fan-in 2, depth $D = D(n)$ and polynomial size. Let $k = k(n)$ be an ensemble of integers such that $1 \leq k \leq \mathsf{poly}(n)$. Then, there is an interactive argument that, on input $(x_1, \ldots, x_k) \in (\{0,1\}^n)^k$, verifies that $\forall i \in [k], x_i \in \mathcal{L}$.*

The protocol is public-coins and has perfect completeness and constant soundness error. The communication complexity is $O\left(\log k \cdot (M + k \cdot n^\sigma \cdot D)\right)$. The number of rounds is $O(1/\sigma^3)$. The running time of the verifier is $O\left(\log k \cdot (M + k \cdot n^\sigma \cdot (n + D))\right)$. The honest prover, given witnesses $(w_1, \ldots, w_k) \in (\{0,1\}^M)^k$ for the inputs' membership in $\mathcal{L}$, runs in time $\mathsf{poly}(n)$.

For simplicity, we take the security parameter to be a small polynomial in the input length throughout, obtaining security against $\mathsf{poly}(n)$-time adversaries. More generally, the communication, verifier runtime and prover runtime depend polynomially on the security parameter.

Theorem 1.1 is the first quasi-linear (in the witness length) constant-round batch-verification protocol for a rich subclass of UP that does not assume CRHs. Prior works constructed unconditionally sound constant-round (or $\mathsf{polylog}(k)$-round) batch-verification protocols for this class [RRR16, RRR18, RR20], but they require communication at least $(k^\sigma \cdot M)$. On the other hand, Theorem 1.1 assumes the existence of OWFs and also has an additive term that is (slightly) super-linear in k, so the improvement is most significant when k is sub-linear in the witness length M, i.e. $k \leq M/n^\sigma$. See the comparison with the best known results in Table 1.

class		# rounds	communication	verifier time
depth D, poly-size	(this)	$O(1/\sigma^3)$	$\tilde{O}\left(M + k \cdot n^\sigma \cdot D\right)$	$\tilde{O}\left(M + k \cdot n^\sigma \cdot (n + D)\right)$
NC	[RR20]	$\mathsf{polylog}(k)$	$\tilde{O}(M)$	$\tilde{O}\left(k \cdot n + M\right)$
space n^σ, poly-time	[RRR18]	$\mathsf{exp}(\tilde{O}(1/\sigma))$	$O(k^\sigma \cdot M^{1+\sigma})$	$\tilde{O}(k \cdot n + k^\sigma \cdot M^{1+\sigma})$

Table 1: Comparison of batch verification protocols for UP, where $\sigma \in (0,1)$ is a desired constant for bounding the communication. The first result is an argument, that assumes the existence of OWF, and the rest are unconditionally sound proofs. The first two protocols require Logspace-uniformity. In all protocols, the (honest) prover is efficient when given the k witnesses. The $\tilde{O}(\cdot)$ hides $\mathsf{polylog}(k, n)$ factors.

1.2 Constant-Round Arguments for Bounded Space

Our second result is a new construction of *doubly-efficient* and *constant-round* interactive arguments for languages computable by bounded-space Turing machines. We state the result for languages computable in polynomial time and refer the reader to Theorem 5.1 for the more general statement.

Theorem 1.2 (Constant-Round Interactive Arguments for Polynomial Time and Bounded Space)**.** *Assume one-way functions exist. Let $\mathcal{L}$ be a language that can be computed in time* $\mathsf{poly}(n)$ *and space $S = S(n)$. Then, for every constant $\sigma \in (0,1)$, the language $\mathcal{L}$ has an $O(1/\sigma^4)$-round*

public-coin interactive argument with perfect completeness and negligible soundness error. The communication complexity is $O\left(n^{\sigma} \cdot S^2\right)$. *The prover runs in time* $\mathsf{poly}(n)$ *and the verifier runs in time* $O\left(n^{1+\sigma} + n^{\sigma} \cdot S^2\right)$.

We compare this result to the work of Reingold, Rothblum and Rothblum [RRR16], who obtained constant-round unconditionally sound interactive proofs for the same class of computations. Our work builds on their techniques. Compared with their result, the main distinction is that in Theorem 1.2 we obtain an *exponential* improvement in tradeoff between rounds and communication: to get communication complexity $O\left(n^{\sigma} \cdot \mathsf{poly}(S)\right)$, the [RRR16] protocol uses $\mathsf{exp}(\tilde{O}(1/\sigma))$ many rounds, whereas the number of rounds in our work is $\mathsf{poly}(1/\sigma)$. We also improve the dependence on the space complexity and reduce the communication to $O\left(n^{\sigma} \cdot S^2\right)$. However, we stress that our protocol assumes the existence of one-way functions and is only computationally sound. In Table 2 we compare the power and complexity of the new result to an interactive *proof* for the *same* class of computations, and to an interactive *argument* for a *different* class of computations. In all protocols, the (honest) prover is efficient, which makes them doubly-efficient.

class	(in)	# rounds	communication	verifier time	soundness
space S, poly-time	(this)	$O(1/\sigma^4)$	$O\left(n^{\sigma} \cdot S^2\right)$	$O\left(n^{\sigma} \cdot S^2 + n^{1+\sigma}\right)$	argument
space S, poly-time	[RRR16]	$\mathsf{exp}(\tilde{O}(1/\sigma))$	$n^{\sigma} \cdot \mathsf{poly}(S)$	$n^{\sigma} \cdot \mathsf{poly}(S) + \tilde{O}(n)$	proof
depth D, poly-size	[AR23]	$O(1/\sigma^3)$	$O\left(n^{\sigma} \cdot D\right)$	$O\left(n^{\sigma} \cdot D + n^{1+\sigma}\right)$	argument

Table 2: Comparison with the [RRR16, AR23] protocols, where $\sigma \in (0,1)$ is a desired constant for bounding the communication. The two arguments assume the existence of OWFs.

Comparison to known argument systems. As noted above, Kilian [Kil92] shows that assuming the existence of CRHs, there exist 4-message doubly-efficient arguments with sublinear communication and almost-linear verification time for all of P (regardless of the space complexity required for the computation). In fact, this celebrated protocol can be used for all of NP, i.e., a much richer class of computations, and even for NP batching, since verifying the membership of k instances in an NP language is itself an NP problem.

Under stronger assumptions, such arguments exist even beyond NP, and some require only 2 messages or even no interaction at all (starting with Micali's CS proofs [Mic94], see also Kalai, Raz and Rothblum [KRR22] and Choudhuri, Jain and Jin [CJJ21], as well as the recent expositions by Thaler [Tha22] and Ishai [Ish20a, Ish20b], and the references therein). The vast majority of works on argument systems use assumptions that (at the very least) imply CRHs, and this holds w.r.t. the task of batch verification as well. One exception is works by Bitansky, Kalai and Paneth [BKP18] and by Komargodski, Naor and Yogev [KNY18], who construct argument systems based on the existence of the more relaxed primitive of multi-collision-resistant hash functions, though the recent work of Rothblum and Vasudevan [RV22] indicates that the gap between collision-resistance and multi-collision-resistance might not be wide.

The main distinction in our work is that we rely only on the existence of one-way functions, and achieve results that are applicable for restricted classes of computations (in particular, subclasses of P and NP). Whereas [AR23] use the model of Boolean circuits and focus on bounded-*depth* computation, in this work we use Turing machines and focus on bounded-*space* computations (see

Remark 1.3), and on batch verification for UP. Comparing these results to Kilian's 4-message protocol, the round complexity, while constant, is larger than Kilian's (let alone the subsequent works that further reduce the interaction using stronger assumptions). However, OWFs are generally considered to be a considerably more relaxed assumption than CRHs. Simon [Sim98] showed a black-box separation between the two notions. A natural question for future work is whether one-way functions suffice for constructing arguments that can be verified in almost-linear time for all of P, or even for all of NP.

Remark 1.3 (Comparison to [AR23])**.** *The [AR23] result can be used for bounded-*space *computations, using the simulation of a Turing machine by a low-depth circuit. Note, however, that for a space-S machine, this simulation results in a circuit of depth S^2 and size $\gg 2^S$. Plugging this into the [AR23] result would* not *give an efficient honest prover when $S = \omega(\log n)$, and in the context of argument systems this means that the honest prover is more powerful than the adversary. For* Logspace *computations, one can get an efficient honest prover, however the simulation by an* NC^2 *circuit might still incur a large polynomial overhead in the honest prover's running time.*

Succinct Zero-Knowledge Arguments. We also note that, assuming the existence of one-way functions, our protocols (which are public-coins) can be made zero-knowledge using standard techniques [IY87, BGG+88]. In particular, we can use Theorem 1.2 to construct succinct *constant-round* zero-knowledge arguments for NP statements from one-way functions, whenever the witness relation can be computed in bounded space. Succinctness means that the communication is nearly-linear in the witness length. The following corollary improves [RRR16, Theorem 2] in terms of the round complexity, reduced from $\mathsf{exp}(\tilde{O}(1/\sigma))$ to $(1/\sigma^4)$ (see the highlighted part in Table 2), but soundness holds only against polynomially bounded provers.

Corollary 1.4. *Assume one-way functions exist and let $\sigma \in (0,1)$ be a constant. Let $\mathcal{L}$ be an* NP *language, whose witness relation can be computed in time* $\mathsf{poly}(n)$ *and space $S = S(n)$ by a Turing machine that operates on inputs of length n and witnesses of length $M = \mathsf{poly}(n)$. The language $\mathcal{L}$ has a public-coin zero-knowledge interactive argument with perfect completeness, constant soundness error and $O(1/\sigma^4)$ rounds. The communication complexity is $n^\sigma \cdot O\left(M + S^2\right)$ and the verifier runs in time $n^\sigma \cdot O\left(n + M + S^2\right)$. The (honest) prover, given a valid witness, runs in time* $\mathsf{poly}(n)$.

2 Technical Overview

We outline the key ideas underlying our constant-round argument constructions (Theorems 1.1 and 1.2). We begin with a brief review on universal one-way hash functions (UOWHFs) and hash trees, that play a central role in both constructions.

2.1 Targeted Collision-Resistant Hash with Local Opening

UOWHFs. A family $\mathcal{H}$ of universal one-way hash functions (UOWHFs), introduced by Naor and Yung [NY89], is a family of shrinking functions with the following property: fixing an input x, and drawing a random hash function h from the family $\mathcal{H}$, it is hard to find a "second preimage" $x' \neq x$ s.t. $h(x') = h(x)$. This is sometimes referred to as second-preimage collision-resistance, or targeted collision-resistance. Note that the order of events is important: the input x should be fixed *before* the hash function $h \sim \mathcal{H}$ is selected. This is a considerable relaxation to collision-resistant

families, where even after h is chosen, it should be hard to find *any* collision (in a UOWHF, after h is revealed, it may well be possible to adaptively compute a pair x', x'' that collide, but they will not collide with any input that was fixed before h was chosen). Indeed, Rompel [Rom90] showed that UOWHFs can be constructed from any one-way function (Naor and Yung showed a construction from one-way permutations). Our construction uses a family of UOWHFs that map inputs in $\{0,1\}^{\kappa^2}$ to outputs in $\{0,1\}^{\kappa}$, where the security parameter κ is generally taken to be n^{δ} for a small constant $\delta \in (0,1)$, and the "shrinkage" factor is $1/\kappa = 1/n^{\delta}$.

A UOWHF tree. As observed in the work of [AR23], UOWHFs suffice for implementing a targeted collision-resistant hash with local opening. In particular, UOWHFs can be used in a hash tree to hash an M-bit string $x \in \{0,1\}^M$ to a short commitment (or hash root) $y \in \{0,1\}^{\kappa}$. The root y can later be used to *locally open* any desired bit x_i. Naor and Yung [NY89] observed that using a single UOWHF in a straightforward hash tree [Mer89, Dam89] might not be secure. However, the construction is secure if we use a separate hash function for each layer of the tree. The two important properties of this construction are:

1. **Local opening:** given any i, the local opening for $x[i]$ only requires sending $O(\kappa \cdot \log M)$ bits and can be verified in $\mathsf{poly}(\log M, \kappa)$ time.

2. **Targeted collision-resistance:** for any string x fixed before choosing the hash functions, taking $\mathsf{root}(x)$ to be the (correct) hash root according to x, it is hard to find an index i and a valid opening for any value of the i^{th} bit that is different from $x[i]$.

We comment that this construction is not a "standard" commitment scheme in the sense that it is not necessarily hiding, and, as described in the second item, only satisfies the relaxed "targeted" binding property when comparing to the one implied by CRH.

2.2 Batch Verification of UP Statements

We begin with the protocol for batching the verification of UP statements (NP statements that have at most one witness), which gives a taste of our new ideas and techniques. Consider a UP language $\mathcal{L}$ with witnesses of length $M = M(|x|)$. Our goal is to design an IA where, given k inputs $x_1, \ldots, x_k$, the verifier accepts only if $\forall i \in [k], x_i \in \mathcal{L}$ (and otherwise rejects w.h.p.). We also want the prover strategy to be *efficient*: the (honest) prover should run in polynomial time given witnesses to the inputs' membership in $\mathcal{L}$.

A naive solution is sending the k witnesses in their entirety. An honest prover, who knows the witnesses, runs in polynomial time, but the communication and verification grow with $(k \cdot M)$. Our goal is to *batch* the verification of these k UP statements via an IA with communication that is much smaller than $(k \cdot M)$. In what follows, we show an IA with communication that depends quasi-linearly on M, with an additive term that is slightly super-linear in k (but not in M). We assume that the honest prover $\mathcal{P}$ is given as input k witnesses, where $w_i \in \{0,1\}^M$ is a witness for the i^{th} input x_i.

Before stating the protocol, we highlight two of its main ideas. To start, suppose that all but one x_i are in $\mathcal{L}$, so each has a unique witness w_i. The protocol begins with the prover sending commitments to these witnesses, using the scheme described above. Assume further that for each $x_i \in \mathcal{L}$, the commitments (the "hash roots") are correct. Targeted collision-resistance implies that for each such i, the prover is now (computationally) committed to w_i. Next, the prover sends the

XOR of the k witnesses: the M-bit vector $\widetilde{chksum} = \bigoplus_{i\in[k]} w_i$. From now on, whenever the prover is asked to locally open the r^{th} bit of a witness, it will send openings for the r^{th} bit of each of the k witnesses (k openings in all), and the verifier checks that the XOR of these bits equals $\widetilde{chksum}[r]$.

We assumed that for all k rows but one the commitment is binding. Let $i^* \in [k]$ be the row where $x_{i^*} \notin \mathcal{L}$ and the commitment is not binding. Once the prover sends $\widetilde{chksum}$, it is also committed to how it opens each bit r in the alleged witness for the $i^{*\text{th}}$ row, i.e., the cheating prover has also "committed" to a fixed string for the $i^{*\text{th}}$ alleged witness. We can now run an IA for each row i to verify that $x_i \in \mathcal{L}$, where we use an IA that only makes a single query to (an encoding of) the witness, and is sound so long as the alleged witness string is fixed in advance. By the above, the cheating prover is committed to using a fixed (encoding of a) string for the $i^{*\text{th}}$ row's "witness", and the IA will reject. To obtain quasi-linear communication we need a high-rate encoding, which can be achieved using the code-switching technique of Ron-Zewi and Rothblum [RR19].

To make the above approach go through, we made two assumptions: that only one of the k inputs is not in $\mathcal{L}$, and that the prover is honest when committing to the witnesses of the rest of the inputs. In order to reduce the general case to this restricted one, we add a single step, *after* the prover sends the commitments: the verifier guesses a subset I of the inputs, such that w.h.p. I contains *exactly* one input on which the prover cheats. We define *cheating* on an input as: (i) it is not in $\mathcal{L}$, or (ii) it is in $\mathcal{L}$ but the prover does not send the correct commitment to its witness. The verifier sends the subset I to the prover, and they execute the rest of the protocol with respect to the inputs in I (while ignoring the rest). Note that here we need an IA for catching the prover if it cheats in the commitment it sent (even if the input x_{i^*} is in $\mathcal{L}$), see Section 4 for further details (we remark that this results in using different witness encodings in the checksum and in the commitment).

To see how I is chosen, suppose that the verifier knew the number b of "bad" inputs on which the prover is cheating (but does not know which rows are in the subset). If it chooses about (k/b) rows uniformly at random, then with constant probability there will be exactly one cheating row in the set. In fact, it suffices for the verifier to know b up to a multiplicative factor of 2. Thus, in the general case, the verifier can (approximately) cover all possibilities for b (up to a factor of 2) by running (in parallel) $\log k$ instantiations of the protocol, and trying all possible powers of 2. For each of these b's, the verifier chooses about (k/b) rows uniformly at random. This increases the verification and communication complexities by (only) a multiplicative $\log k$ factor.

The full protocol proceeds as follows:

1. $\mathcal{V}$ samples UOWHFs to instantiate the commitment scheme, and sends them to $\mathcal{P}$.
2. $\mathcal{P}$ commits to the k witnesses: it sends hash roots $y_1, \ldots, y_k$.
3. $\mathcal{V}$ samples a subset of indices $I \subseteq [k]$ as above and sends it to $\mathcal{P}$.
4. $\mathcal{P}$ constructs an $|I| \times M$ matrix A whose rows are the witnesses for any $i \in I$:

$$A = \begin{bmatrix} \text{—} & w_{i_1} & \text{—} \\ & \vdots & \\ \text{—} & w_{i_{|I|}} & \text{—} \end{bmatrix}.$$

$\mathcal{P}$ computes the parities of A's columns, and sends these to the verifier. We view this vector of parities (one per column of A) as a "checksum", denoted $chksum = \bigoplus_{i \in I} w_i$.

5. $\mathcal{V}$ receives a vector $\widetilde{chksum} \in \{0,1\}^M$.[2] For $i \in I$, both parties run an IA that verifies the validity of w_i and its commitment y_i. Each execution ends with $\mathcal{V}$ asking from $\mathcal{P}$ to open y_i at a single coordinate r (the same for all rows). Then, it runs two tests, and accepts only if they both pass:

   (a) **Openings Check.** All of the openings are valid w.r.t. the chosen UOWHFs;

   (b) **Checksum Consistency Check.** The r^{th} bit of $\widetilde{chksum}$ indeed equals the parity of the values claimed for the r^{th} column of A. That is:

$$\widetilde{chksum}[r] = \bigoplus_{i \in I} w_i[r].$$

Assuming that the IA performed in Step (5) is efficient, this batch verification protocol is also efficient: excluding this step, the communication complexity is only $M + k \cdot \mathsf{poly}(\kappa)$ bits — the cost of sending a single row and sending openings[3] for each entry of a single column — which is a considerable saving over the naive sound protocol that required $(k \cdot M)$ bits. The full protocol is slightly more involved, since the prover actually commits to the *encoding* of the witnesses, and uses the encoded witnesses when constructing A, in order to work with the IA.

Soundness of the UP batching protocol. The targeted collision-resistance binding property of the commitment scheme implies that once the prover sends a *correct* commitment for a certain witness, the verifier effectively gains oracle access to (the encoding of) this witness, because the prover cannot (locally) open the commitment to anything but the true value. By sampling the subset I in Step (3) as described above, we reduce soundness to the case where the prover only cheats on a single input, where *cheating* on an input is defined as: either the input has no witness (namely, $x_i \notin \mathcal{L}$), or it has a witness but the prover chooses to send a false commitment to it.

Hence, we restrict our attention to the case where the prover cheats on a single instance i^*. Note that for any $i \neq i^*$, the *unique* witness w_i is fixed in advance before the protocol begins, and moreover, that all openings are valid according to Step (5a). Since $\mathcal{P}^*$ sends the correct commitment y_i, targeted collision-resistance applies and $\mathcal{P}^*$ must open y_i according to w_i in Step (5) (i.e., when running the IA that verifies witnesses and commitments). We emphasize that $\mathcal{P}^*$ can send arbitrary and adaptive answers on the $i^{*\text{th}}$ instance, after seeing the column r: the purpose of the commitments is only forcing $\mathcal{P}^*$ to answer according to w_i for instances $i \neq i^*$.

Observe that $\mathcal{V}$ has full access to x_{i^*} and y_{i^*}, thus $\mathcal{P}^*$ can only be adaptive in its choice of the witness when running the $i^{*\text{th}}$ execution of the IA. We show that the checksum ties $\mathcal{P}^*$'s hands by ensuring that it uses a *fixed* string to determine its answers. Suppose that $\mathcal{P}^*$ makes the verifier accept with probability ε, conditioned on I containing a single index on which $\mathcal{P}^*$ cheats. We use $\mathcal{P}^*$ to construct a fixed string w that makes the verifier accept the input to the $i^{*\text{th}}$ execution of

[2] Here and throughout this work we use tildes to denote potentially-corrupted strings that the verifier receives from an untrusted prover.

[3] Recall that each opening is of size polynomial in the security parameter κ, and that we set $\kappa = n^{\delta}$.

the IA with probability ε. We derive w from the checksum value $\widetilde{chksum}$:

$$w = \widetilde{chksum} \oplus \left(\bigoplus_{i \neq i^*} w_i \right).$$

We claim that on input (x_{i^*}, w, y_{i^*}), the verifier will accept with probability ε. To see this, recall that $\mathcal{P}^*$ answers (i.e., opens the commitments) to rows $i \neq i^*$ according to w_i. Whenever $\mathcal{P}^*$ makes $\mathcal{V}$ accept, it must pass the checksum consistency check in Step (5b), and thus it must answer to the $i^{*\text{th}}$ execution of the IA according to w. On the other hand, $\mathcal{V}$ should accept (x_{i^*}, w, y_{i^*}) only if w is valid w.r.t. x_{i^*} and y_{i^*} is valid w.r.t. w. By the definition of i^*, at least one of these condition is not satisfied. We conclude that whenever $\mathcal{P}^*$ makes $\mathcal{V}$ accept, it also breaks the soundness guarantee of the IA, therefore ε is bounded by the IA's soundness error.

Lastly, note that there are two qualitative differences from [RRR16]'s UP batching protocol. The first is that they use PCPs as the matrix rows, instead of encoded witnesses, and a multiple-row checksum. Consequently, their checksum is of size at least $\mathsf{polylog}(k) \cdot \mathsf{poly}(M)$, since it is composed of $\mathsf{polylog}(k)$ rows, each containing a PCP of length $\mathsf{poly}(M)$. Alternatively, to achieve constant round complexity, the multiplicative factor grows to a small polynomial in k (instead of polylogarithmic). The second difference is that their construction is *recursive*. At each step of the recursion, they reduce the number of batched instances, while making sure that the prover cheats on a large enough fraction of the smaller subset. At the base of the recursion, the prover sends the witnesses explicitly for the remaining subset. Eliminating the recursion results in a better round complexity, or, a better trade-off between the number of rounds and the communication complexity.

To conclude, we remark that the IA performed in Step (5) is indeed efficient. We use [AR23]'s IA for bounded-depth computations, and for witness relations computable in depth $D(n)$, its communication and verification time are $D(n) \cdot \mathsf{poly}(\kappa)$ for a security parameter κ, set to n^δ for a small constant $\delta \in (0, 1)$. Its round complexity is $O(1/\delta^3)$, and its soundness error is negligible.

2.3 From UP Batching to IAs for Bounded-Space Computations

Probabilistically Checkable Interactive Proofs/Arguments (PCIP/PCIAs). To prove our main result, we use interactive protocols where the verifier only reads a few bits of the transcript in checking the validity of the statement, thus can be thought of as an interactive analogue of PCPs. We call these *probabilistically checkable interactive proofs* (PCIPs) in the statistically sound setting, or *probabilistically checkable interactive arguments* (PCIAs) in the computationally sound setting, and on this work focus on the latter. These were introduced in the work of Reingold *et al.* [RRR16], and in an independent work of Ben Sasson *et al.* [BCS16] by the name of *Interactive Oracle Proofs*.

A PCIA for a language $\mathcal{L}$ is an IA that is divided into two phases. In the *communication* phase, the prover and verifier interact for r rounds and generate a transcript, as in a standard interactive argument. Restricting our attention to public-coin protocols, all that the verifier does in this phase is send random strings $\beta_1, \ldots, \beta_r$ (one in each of the r rounds). In the *checking* phase, the verifier queries q bits of the messages sent by the prover and accepts or rejects. The verifier's running time in a PCIA is just the time for the checking phase (generating queries and deciding whether to accept). Thus, in a PCIA, the query complexity and the verifier's runtime can be much smaller than the transcript length.

Batch Verification of PCIAs. Our starting point for the proof of Theorem 1.2 is [RRR16]'s template for constructing IPs for bounded-space computations from a batch verification protocol for PCIPs. Assume we have a PCIA for verifying Turing machine computations of time T and space S, that we want to extend to verifying computations of time $(k \cdot T)$ and space S for some super-constant integer k. If the prover sends $k-1$ intermediate states of the machine, we reduce the verification of the $(k \cdot T)$-time computation to that of verifying k computations running in time T, namely, all we need is an efficient batch verification protocol for PCIAs.[4] Applying it repeatedly obtains PCIAs (and interactive arguments) for longer and longer computations. In what follows, we introduce an improved batch verification theorem for PCIAs, while using the targeted collision-resistant scheme defined in Section 2.1.

From UP batching to PCIA batching. First, we attempt to provide some intuition for the construction. Recall that the UP batching protocol starts with the prover committing to each of the k unique witnesses. Translating this idea to the PCIA setting, it is unclear what can the prover commit to: if its messages in the "base PCIA" (for length-T computations) were unique, then committing to them would work and bound its adaptivity in the rest of the protocol. However, the prover's messages in the base PCIA depend on the verifier's messages, and, in particular, are not fixed in advance. Instead, we view *the tableau* of the length-T computation as the analogue of the UP witness.[5] The protocol starts with the prover computing each of the k tableaux and committing to them. Observe that, since the computations are deterministic, each tableau is *unique*.

We want to use the tableau-commitments to bind the prover's messages in the "base PCIA". To achieve this, we require that the messages of the prover in the base PCIA can be computed efficiently given the tableau. In particular, this implies that we can verify claims about the messages sent in the base PCIA using an efficient consistency protocol that only makes a single query to (an encoding of) the tableau, so if the prover sends a correct (and thus binding) tableau-commitment, it cannot cheat about messages in the base PCIA without getting caught.

With this high-level idea in place, we can proceed analogously to the UP batching case. After receiving the tableau-commitments, we run a protocol where the prover sends checksums that are XORs of the messages of (a subset I of) the k base PCIAs. We then ask the prover to "open" the checksums at the coordinates queried by the PCIA verifier. We run k executions of the aforementioned consistency protocol to verify that the claimed message-values in each execution i of the base PCIAs are consistent with the i^{th} committed tableau. As above, soundness will hold so long as there is exactly one input where the prover is cheating, i.e., either the commitment is incorrect, or the claim about the length-T computation is false (among many of the details we are glossing over, we use additional machinery to make sure that we can catch cheating in the tableau-commitment). We can get to this situation by "guessing" a subset I of the rows as we did in the UP batching.

The PCIA batching protocol. The protocol starts with the prover sending $k-1$ intermediate states of the machine, and k commitments to the k tableaux of these computations (each is a sequence of T configurations). We view each pair of configurations at distance T from each other as an input to the "base" PCIA $(\mathcal{P}, \mathcal{V})$, such that there are k inputs overall. The verifier guesses a subset $I \subseteq [k]$ of these pairs (see above), and from now on we assume that the prover cheats

[4] In fact, we will use a batch verification protocol for a restricted form of PCIAs; see Section 5 for the full details.

[5] Note the difference from a UP witness: a tableau for polynomial time computations is computable in P (or, more generally, in poly(T) time), which does not necessary hold for a general UP witness.

only on one pair in I. Here, *cheating* means either that the machine does not move from the first configuration in the pair to the second one in exactly T steps, or that the commitment to the tableau of the computation is false.

Then, both parties proceed with r rounds that correspond to the r rounds of $(\mathcal{P}, \mathcal{V})$. In each round j, for each $i \in I$, let $\alpha_i^j \in \{0,1\}^a$ be the message that the (prescribed) "base" prover $\mathcal{P}$ would send on input x_i in round j given randomness $\beta^1, \ldots, \beta^j$ (that $\mathcal{V}_{\mathsf{batch}}$ already sent). The verifier sends random coins β^j as sent by $\mathcal{V}$ in the base protocol (the same random coins are used for all $|I|$ executions), and the prover $\mathcal{P}_{\mathsf{batch}}$ sends a checksum $chksum^j \in \{0,1\}^a$, defined as the parity of $\mathcal{P}$'s messages: $chksum^j = \bigoplus_{i \in I} \alpha_i^j$.

After these r rounds, $\mathcal{V}_{\mathsf{batch}}$ chooses random coins for $\mathcal{V}$'s checking phase (i.e., querying the input and deciding whether to accept), sends the queries $Q \subset [r] \times [a]$ to $\mathcal{P}_{\mathsf{batch}}$, and receives answers $(\phi_i : Q \to \{0,1\})_{i \in I}$ to these queries. $\mathcal{V}_{\mathsf{batch}}$ accepts if and only if: (i) $\mathcal{V}$ would accept the answers in all $|I|$ protocols, and (ii) the answers are consistent with the checksums sent in rounds $1, \ldots, r$.

The last component of the batching protocol is running an IA for any $i \in I$ that verifies the validity of the prover's answers in the i^{th} PCIA — namely, that it answers according to the prescribed strategy — and of the i^{th} commitment.[6] During these executions, $\mathcal{V}_{\mathsf{batch}}$ asks from $\mathcal{P}_{\mathsf{batch}}$ to open the commitment at a single coordinate, and accepts only if all openings are valid. We stress that this step is important only for $i \in I$ for which the prover sends a correct commitment. In order for the IA to be efficient, we assume that given the tableau of the length-T computation, it is possible to compute $\mathcal{P}$'s answers in the base protocol by a low-depth circuit. We call this class of honest provers *low-depth provers*, and make sure that $\mathcal{P}_{\mathsf{batch}}$ (and the provers in the rest of the protocols that we use or construct) satisfies this property.

The round complexity of the protocol is dominated by the IA's round complexity, which is $O(1/\delta^3)$ where n^δ is the security parameter. Since the number of iterative applications of the batching protocol is eventually set to $O(1/\delta)$, we derive the $O(1/\delta^4)$ round complexity stated in Theorem 1.2 (where $\sigma = O(\delta)$). We note that there is hope that this result can be achieved without adding any cryptographic assumption.

Soundness of the PCIA batching protocol. The proof of soundness is similar to the analogous proof of the UP batch verification. Taking i^* to denote the single instance in I where the prover cheats, we get that $\forall i \neq i^*$ the commitment is correct. Due to the targeted binding property of the scheme and the soundness guarantee of the IA, the prover answers to the queries of the i^{th} protocol according to the prescribed strategy $\forall i \neq i^*$. Then, when it sends the checksum value $\widetilde{chksum}^j$, it implicitly commits to messages sent for the $i^{*\text{th}}$ protocol, since it must also pass the checksum consistency check. This means that ϕ_{i^*} is fixed before the $i^{*\text{th}}$ execution begins, for any chosen query set Q. Thus, if $\mathcal{P}^*_{\mathsf{batch}}$ could get $\mathcal{V}_{\mathsf{batch}}$ to accept, we could derive a cheating prover for the base protocol, breaking its soundness. We note that it is critically important for this argument that $\mathcal{P}^*_{\mathsf{batch}}$ sends $\widetilde{chksum}^j$ (and commits to the messages in round j), before it knows the random coins β^{j+1} that will be chosen by the verifier for round $j+1$. Moreover, we stress that the binding

[6] After $\mathcal{V}_{\mathsf{batch}}$ guesses the subset, it must reject in the case that there is a single pair where $\mathcal{P}^*_{\mathsf{batch}}$ cheats, even if the cheating is due to sending a false commitment and the pair of configurations represents a correct length-T computation of the machine. Notice that the verifier may not catch this if we follow the protocol as described above. Thus, in the actual protocol, we run two IAs: the first only verifies the correctness of the commitments w.r.t. the tableaux, and it is run before the PCIA executions, and the second is the one described here, excluding the check of the commitments. See Section 5 for the full details.

property of the scheme, namely, targeted collision-resistance (see Item 2), applies in this setting as well due to the uniqueness of the tableaux.

Comparison to RRR's batching protocol. To conclude, let us spell out the two main differences between this protocol and the one constructed in [RRR16], for readers who are familiar with it. The first is that the RRR protocol has a recursive structure. At each recursive step, the task of (interactively) batching k protocols is reduced to that of batching $k' \ll k$ protocols. At the base of the recursion, when sufficiently few protocols are left, the prover and verifier simply execute them. In order to verify that the prover's answers in the k' protocols are consistent with the prescribed strategy, they design an interactive protocol where the prover interactively convinces the verifier that it can "complete" the messages it sent until round j to a full transcript, when the verifier uses fresh randomness in each execution, and run this protocol $\forall j \in [r]$.

In our setting, we are able to avoid the recursion, by narrowing down to a subset with only one instance i^* on which the prover cheats and reducing the correctness of all instances in I to the correctness of i^*. Moreover, we avoid the r executions (r per $i \in I$) of the interactive protocol that verifies consistency between the prover's answers to i^* and the prescribed strategy, by a single execution of the IA described above (one per $i \in I$) and while relying on the low-depth property. Note that guessing a subset in the RRR setting would not work, since without the commitment scheme, nothing promises that the prover uses the prescribed messages to answer the queries in the rest of the PCIA executions $i \neq i^*$. Eliminating the recursion and the r executions of the interactive protocol results in a better round complexity. It also implies shorter prover messages, however, this does not significantly improve the communication complexity.

The second is that the class of protocols to which RRR's batching theorem applies is more restricted. In their work, they introduce the notion of *unambiguous interactive proof*. These are proofs where the prover has a unique strategy to convince a verifier (similarly to a unique witness). They require from their protocols to satisfy unambiguity, and since the construction uses iterative applications of the batching protocol, requiring the base protocol to be unambiguous implies that the batching protocol should be unambiguous as well. Unlike the RRR protocol, we do not require unambiguity, which significantly simplifies the proof. We are able to avoid this technical overhead by the low-depth property that we require from the honest prover.

Organization. Preliminaries and technical definitions are in Section 3. In Section 4, we prove the UP batching protocol. The full proof of Theorem 1.2, that is, interactive arguments for bounded-space computations, is in Section 5 and Appendix B.

3 Preliminaries

For a string $x \in \Sigma^n$ and an index $i \in [n]$, we denote by $x[i]$ the i^{th} entry in x. For a set $I \subseteq [n]$, we use $x|_I$ to describe the sequence of entries in x corresponding to coordinates in I, and "$\circ$" for string concatenation.

For any pair of distributions, $D_1 \equiv D_2$ means that they are identical. We use $x \in_R D$ to indicate that the element x was drawn uniformly at random from the distribution D. For a multiset (i.e., a set where repetitions are allowed) of elements I, we use $I \subseteq_R D$ to denote that for each $x \in I$, x is sampled uniformly at random from D, and all of these samples are independent.

A Useful Probabilistic Claim. The following proposition argues that if we have a fixed set of a known size, that has a subset T of an unknown size, then we have a logarithmic (in the set's size) success probability in guessing a correct 2-approximation of T's size. Further, this implies that, with a logarithmic success probability, a random subset I will intersect T at a single point.

Proposition 3.1. *Fix a subset $T \subseteq [k]$ of size $t \in [k]$ and assume that $k \geq 2$. If we sample an integer $b \in_R \{0, \ldots, \lceil \log k \rceil\}$ and then sample $d = \lceil \frac{k}{2^b} \rceil$ indices $I \subseteq_R [k]$ independently (such that $|I| = d$) that may repeat themselves (i.e., we allow duplicates), then,*

$$\Pr_{\substack{b \in_R \{0,\ldots,\lceil \log k \rceil\} \\ I \subseteq_R [k]}} \left[|T \cap I| = 1 \right] \geq \frac{1}{30 \log k}.$$

Proof. Our goal is to lower bound the probability that the random subset I has exactly one index which is in the subset T, and that the rest of the $(d-1)$ indices in I are in $[k] \setminus T$. We view 2^b as a guess for t up to a factor of 2. First, notice that

$$\Pr[t \leq 2^b < 2t] = \Pr[b = \lceil \log t \rceil] \geq \frac{1}{\lceil \log k \rceil + 1} \geq \frac{1}{\log k + 2} \geq \frac{1}{3 \log k},$$

since $\lceil \log t \rceil \in \{0, \ldots, \log k\}$, and b is sampled uniformly from a distribution of size $(\lceil \log k \rceil + 1)$. For the last inequality, we used the fact that $k \geq 2$.

Next, we assume that $t \leq 2^b < 2t$ and find the conditional probability that the size of the intersection is 1. We split into two cases. If $t \geq k/2$:

$$\Pr\left[|T \cap I| = 1\right] \geq \Pr\left[|T \cap I| = 1 \mid b = \lceil \log k \rceil\right] = \Pr\left[|T \cap I| = 1 \mid d = 1\right] = \frac{t}{k} \geq \frac{k/2}{k} = \frac{1}{2}.$$

If $t < k/2$:

$$\Pr\left[|T \cap I| = 1\right] = \Pr[\exists i^* \in I \text{ s.t. } i \in T] \cdot \Pr[\forall i \in I \setminus \{i^*\},\ i \notin T] = d \cdot \frac{t}{k}(1 - \frac{t}{k})^{d-1}.$$

To lower bound this term, we use the assumption that $t \leq 2^b < 2t$. Since $k/2^b \leq d < k/2^b + 1$, it holds that $k/2t < d < k/t + 1$ and therefore

$$d \cdot \frac{t}{k}(1 - \frac{t}{k})^{d-1} \geq \frac{k}{2t} \cdot \frac{t}{k}(1 - \frac{t}{k})^{k/t+1-1} = \frac{1}{2}(1 - \frac{t}{k})^{k/t} > \frac{1}{10},$$

where the last inequality follows from the fact that $\forall x \in [0, \frac{1}{2}]$ $(1-x)^{\frac{1}{x}} > 0.2$. This means that in both cases, the probability is lower bounded by $1/10$, and the proposition follows. □

3.1 Multivariate Polynomials and Low Degree Extensions

We recall some important facts on multivariate polynomials (see [Sud95] for a far more detailed introduction). A basic fact, captured by the Schwartz-Zippel lemma, is that low degree polynomials cannot have too many roots.

Lemma 3.2 (Schwartz-Zippel Lemma)**.** *Let $P : \mathbb{F}^m \to \mathbb{F}$ be a* non-zero *polynomial of total degree d. Then,*

$$\Pr_{x \in \mathbb{F}^m}[P(x) = 0] \leq \frac{d}{|\mathbb{F}|}.$$

An immediate corollary of the Schwartz-Zippel Lemma is that two distinct polynomials $P, Q : \mathbb{F}^m \to \mathbb{F}$ of total degree d may agree on at most a $\frac{d}{|\mathbb{F}|}$-fraction of their domain $\mathbb{F}^m$.

Throughout this work, we consider fields in which operations can be implemented efficiently (i.e., in poly-logarithmic time in the field size). Formally we define such fields as follows.

Definition 3.3. *We say that an ensemble of finite fields $\mathbb{F} = (\mathbb{F}_n)_{n\in\mathbb{N}}$ is* constructible *if elements in $\mathbb{F}_n$ can be represented by $O(\log(|\mathbb{F}_n|))$ bits and field operations (i.e., addition, subtraction, multiplication, inversion and sampling random elements) can all be performed in $\mathsf{polylog}(|\mathbb{F}_n|)$ time given this representation.*

A well known fact is that for every $S = S(n)$, there exists a *constructible* field ensemble of size $O(S)$ and its representation can be found in $\mathsf{polylog}(S)$ time using a randomized algorithm (see, e.g., [Gol08, Appendix G.3] for details). Furthermore, if the ensemble extends $\mathbb{GF}[2]$, then the representation can be found deterministically within this time.

In the following we consider polynomials over a constructible field. In this work we will use the following well-known local testability and decodability properties of polynomials.

Lemma 3.4 (Self-Correction Procedure (cf. [GS92, Sud95])**.** *Let $\mathbb{F}$ be a constructible field ensemble. Let $\delta < 1/3$, $\varepsilon \in (0, 1]$, and $d, m \geq 1$. There exists an algorithm that, given input $x \in \mathbb{F}^m$ and oracle access to an m-variate function $P : \mathbb{F}^m \to \mathbb{F}$ that is δ-close to a polynomial P' of total degree d, runs in time $O(d \cdot m \cdot \log(|\mathbb{F}|) \cdot \log(1/\varepsilon))$ makes $O(d \cdot \log(1/\varepsilon))$ oracle queries and outputs $P'(x)$ with probability $1 - \varepsilon$. Furthermore, if P itself has total degree d, then given $x \in \mathbb{F}^m$, the algorithm outputs $P(x)$ with probability 1.*

We will also need a variant of the low degree test that tests the individual degree of the polynomial (rather than total degree). Such a test is implicit in [GS06, Section 5.4.2] (see also [GR15, Theorem A.9]).

Lemma 3.5 (Individual Degree Test)**.** *Let $\mathbb{F}$ be a constructible field ensemble. Let $\delta \in (0, 1/2)$, $\varepsilon \in (0, 1]$ and $d, m \in \mathbb{N}$ such that $d \cdot m < |\mathbb{F}|/10$. There exists a randomized algorithm that, given oracle access to an m-variate polynomial $P : \mathbb{F}^m \to \mathbb{F}$, runs in time $(d \cdot m \cdot \log(|\mathbb{F}|) \cdot \mathsf{poly}(1/\delta) \cdot \log(1/\varepsilon))$, makes $(d \cdot m \cdot \mathsf{poly}(1/\delta) \cdot \log(1/\varepsilon))$ oracle queries and:*

1. *Accepts every function that is a polynomial of* individual *degree d with probability 1; and*
2. *Rejects functions that are δ-far from every polynomial of individual degree d with probability at least $1 - \varepsilon$.*

Low Degree Extension. Let $\mathbb{H} \subseteq \mathbb{F}$ be (ensembles of) finite fields. Fix an integer $m \in \mathbb{N}$. A basic fact is that for every function $\phi : \mathbb{H}^m \to \mathbb{F}$, there exists a unique extension of ϕ into a function $\hat{\phi} : \mathbb{F}^m \to \mathbb{F}$ (which agrees with ϕ on $\mathbb{H}^m$; i.e., $\hat{\phi}|_{\mathbb{H}^m} \equiv \phi$), such that $\hat{\phi}$ is an m-variate polynomial of individual degree at most $(|\mathbb{H}| - 1)$. Moreover, there exists a collection of $|\mathbb{H}|^m$ functions $\{\hat{\tau}_x\}_{x\in\mathbb{H}^m}$ such that each $\hat{\tau}_x : \mathbb{F}^m \to \mathbb{F}$ is the m-variate polynomial of degree $(|\mathbb{H}| - 1)$ in each variable defined as

$$\hat{\tau}_x(z) \stackrel{\text{def}}{=} \prod_{i\in[m]} \prod_{h\in\mathbb{H}\setminus\{x_i\}} \frac{z_i - h}{x_i - h},$$

and for every function $\phi : \mathbb{H}^m \to \mathbb{F}$ it holds that

$$\hat{\phi}(z_1, \ldots, z_m) = \sum_{x \in \mathbb{H}^m} \hat{\tau}_x(z_1, \ldots, z_m) \cdot \phi(x).$$

The function $\hat{\phi}$ is called the *low degree extension* of ϕ (with respect to $\mathbb{F}$, $\mathbb{H}$ and m).

We also define the individual degree $|\mathbb{H}| - 1$ polynomial $\hat{\tau} : \mathbb{F}^m \times \mathbb{F}^m \to \mathbb{F}$ as:

$$\hat{\tau}(x, z) \stackrel{\text{def}}{=} \prod_{i \in [m]} \prod_{h \in \mathbb{H} \setminus \{0\}} \frac{z_i - x_i - h}{h}. \tag{1}$$

Observe that for every $x \in \mathbb{H}^m$ it holds that $\hat{\tau}(x, \cdot) \equiv \hat{\tau}_x(\cdot)$.

One way to compute the low degree extension is a process known as *Lagrange Interpolation*. The Lagrange interpolant polynomials can be computed for a given set of evaluation points $\mathbb{H}$ in $O(m \cdot |\mathbb{H}|^2)$ time.

Proposition 3.6 ([AR23, Proposition 3.11])**.** *Let $\mathbb{H} \subseteq \mathbb{F}$ be constructible field ensembles. Let $\phi : \mathbb{H}^m \to \mathbb{F}$ and suppose that ϕ can be evaluated by a Turing machine in time t. Then, there exists a Turing machine that, given as an input a point $z \in \mathbb{F}^m$, runs in time $|\mathbb{H}|^m \cdot O(m \cdot |\mathbb{H}|^2) + O(t)$ and outputs the value $\hat{\phi}(z)$.*

The Low Degree Extension as an Error Correcting Code. The low degree extension can be viewed as an error correcting code applied to bit strings. Formally, fix some canonical ordering of the elements in $\mathbb{H}$. For every integer $n \in \mathbb{N}$, we identify the set $[n]$ with the set $\mathbb{H}^{\log_{|\mathbb{H}|}(n)}$ by taking the representation of $i \in [n]$ in base $|\mathbb{H}|$. Consider the function $\mathsf{LDE}_{\mathbb{F},\mathbb{H}} : \{0,1\}^n \to \mathbb{F}^{|\mathbb{F}|^m}$, where $m = \log_{|\mathbb{H}|}(n)$,[7] that given a string $\phi \in \{0,1\}^n$, views ϕ as a function $\phi : \mathbb{H}^m \to \{0,1\}$, by identifying $[n]$ with $\mathbb{H}^m$ as above, and outputs the truth table of the low degree extension $\hat{\phi} : \mathbb{F}^m \to \mathbb{F}$ of ϕ, represented as an $|\mathbb{F}|^m$ dimensional vector.

We use the notation $\mathsf{LDE}(\phi) = \mathsf{LDE}_{\mathbb{F},\mathbb{H}}(\phi)$ or $\hat{\phi}$ to denote the unique low degree extension (with degree $(|\mathbb{H}| - 1)$ in each variable) for inputs of varying lengths. That is, $m = m(n)$ is implicitly taken to be the appropriate number of variables required for any input of length n and a field of size $|\mathbb{H}|$. Once $|\mathbb{H}|$ is set, m is well defined for every input length. Typically, m will be clear from the context and otherwise we define it.

A Low Degree Extension for Multiple Strings. We will also need to extend the $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}$ encoding to encode multiple strings (of varying lengths). A natural way of doing so is by simply concatenating the strings and applying $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}$ to the concatenated string. However, we take a slightly different approach, which will be useful both for composing encodings of individual strings to a single joint encoding, and for decomposing such a joint encoding into individual encodings. We remark that this approach leverages the fact that the low degree extension is a tensor code [Wol65].

Given strings $x_1 \in \{0,1\}^{n_1}, \ldots, x_k \in \{0,1\}^{n_k}$, the encoding $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1, \ldots, x_k)$ is defined as follows. Let $m_k = \log_{|\mathbb{H}|}(k)$, $m^{(i)} = \log_{|\mathbb{H}|}(n_i)$, $m_{\max} = \max_{i \in [k]} m^{(i)}$ and $m = m_k + m_{max}$. We identify $[k]$ with $\mathbb{H}^{m_k}$ and identify $[\max(|x_i|)]$ with $\mathbb{H}^{m_{max}}$ by viewing the respective integers in base $|\mathbb{H}|$. Let $P : \mathbb{H}^m \to \{0,1\}$ be a function such that for every $z \in \mathbb{H}^{m_k}$, and every $(w, y) \in$

[7] We assume n is a power of $|\mathbb{H}|$, otherwise we can pad the input with zeros.

$\mathbb{H}^{m^{(z)}} \times \mathbb{H}^{m-m_k-m^{(z)}}$, it holds that: $P(z,w,y)$ is equal to the w^{th} bit of x_z if $y = 0^{m-m_k-m^{(z)}}$, and $P(z,w,y) = 0$ otherwise (i.e., if $y \neq 0^{m-m_k-m^{(z)}}$). We define $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1,\ldots,x_k)$ as the low degree extension of P with respect to $\mathbb{F}$, $\mathbb{H}$ and m.

Proposition 3.7 ([RRR16, Proposition 3.8])**.** *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$ and $|\mathbb{H}| \geq \log(|\mathbb{F}|)$. There exist procedures* Compose *and* Decompose *as follows:*

- **Compose***: Given oracle access to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1),\ldots,\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_k)$ and a point $z \in \mathbb{F}^m$, where m is as above, the procedure* Compose *makes a single query to each $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_i)$ (k queries in total), runs in time $k \cdot \mathsf{poly}(|\mathbb{H}|, \sum_i \log_{|\mathbb{H}|}(n_i), \log(k))$ and outputs the z^{th} entry of $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1,\ldots,x_k)$.*
- **Decompose***: Given oracle access to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1,\ldots,x_k)$, some $i \in [k]$ and a point $w \in \mathbb{F}^{m^{(i)}}$, where $m^{(i)}$ is as above, the procedure* Decompose *makes a single oracle query, runs in time $\mathsf{poly}(|\mathbb{H}|, \log(k), \sum_i \log_{|\mathbb{H}|}(n_i))$ and outputs the w^{th} entry of $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_i)$.*

Remark 3.8. *An immediate corollary from Proposition 3.7 is that, in the interactive setting, we can split a claim about $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1,\ldots,x_k)$ into k claims about $(\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_i))_{i\in[k]}$. This is done by realizing the oracle access by interaction with the prover: it sends over the results of the k queries that the verifier makes in the* Compose *procedure. The verifier runs in time $k \cdot \mathsf{poly}(|\mathbb{H}|, \sum_i \log_{|\mathbb{H}|}(n_i), \log(k))$, the prover runs in time $k \cdot \mathsf{poly}(|\mathbb{H}|, \sum_i \log_{|\mathbb{H}|}(n_i) \cdot n_i, \log(k))$, the communication complexity is $k \cdot \log(|\mathbb{F}|)$ and the number of rounds is $1/2$ (i.e., a single message).*

The following proposition allows us to check if an LDE codeword is zero except for a small set of coordinates.

Proposition 3.9 ([RRR16, Proposition 3.10])**.** *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$, where $|\mathbb{H}| \geq \log(|\mathbb{F}|)$ and let $m = m(n)$. There exists a randomized algorithm $\mathcal{A}$ that is given as explicit input a set $S \subseteq \mathbb{H}^m$ and as implicit input an individual degree $|\mathbb{H}|-1$ polynomial $P : \mathbb{F}^m \to \mathbb{F}$. The algorithm runs in time $|S| \cdot \mathsf{poly}(|\mathbb{H}|, m, \log(|\mathbb{F}|))$ and makes at most $|S|+1$ queries. If $P|_{\mathbb{H}^m \setminus S} \equiv 0$, then $\mathcal{A}$ accepts and otherwise, with probability $1 - \frac{|\mathbb{H}| \cdot m}{|\mathbb{F}|}$ it rejects.*

Furthermore, the queries that $\mathcal{A}$ makes depend only on the set S and its random coins.

Next, the following proposition implements the standard interactive process of reducing the task of proving many claims about the low degree extension of a string to the task of proving a single claim about it. The high level idea is to consider a low degree curve passing through all the points that the verifier wishes to read. The prover specifies the values for all the points on the curve and the verifier outputs a random point on the curve and its alleged value. Soundness follows from the fact that composing a low degree curve with a low degree polynomial results in a low degree univariate polynomial.

Proposition 3.10. *Fix some input w and a set of t claims, denoted $(\chi_i, \theta_i)_{i\in[t]} \in (\mathbb{F}^m \times \mathbb{F})^t$, about $\hat{w} = \mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w)$. Then, for every $\varepsilon \in (0,1]$, there is a δ-sound $(r,a,b,\mathcal{P}\mathsf{time},\mathcal{V}\mathsf{time})$-$\mathsf{IP}$ that outputs a single claim (χ,θ) such that:*

- **Completeness.** *If $\forall i \in [t], \hat{w}[\chi_i] = \theta_i$, then $\hat{w}[\chi] = \theta$.*

- **Soundness.** *If $\exists i \in [t]$ such that $\hat{w}[\chi_i] \neq \theta_i$, then, for every prover strategy, with probability $1 - \delta$ over the verifier coin tosses, it holds that $\hat{w}[\chi] \neq \theta$.*

The protocol's complexities are:

- $\delta = (m \cdot |\mathbb{H}| \cdot t^{\varepsilon})/(\varepsilon \cdot |\mathbb{F}|)$.
- $r = 1/\varepsilon$.
- $a = (1/\varepsilon) \cdot m \cdot |\mathbb{H}| \cdot t^{1+\varepsilon} \cdot \log(|\mathbb{F}|)$.
- $b = (1/\varepsilon) \cdot \log(|\mathbb{F}|)$.
- $\mathcal{P}\mathsf{time} = \mathsf{poly}(|\mathbb{F}|^m)$.
- $\mathcal{V}\mathsf{time} = (1/\varepsilon) \cdot m \cdot |\mathbb{H}| \cdot t^{1+\varepsilon} \cdot \log(|\mathbb{F}|)$.

Proof. We split the set $(\chi_i, \theta_i)_{i \in [t]}$ into $t^{1-\varepsilon}$ "batches" of size at most t^{ε} each. We run the following process on each of the batches, and repeat applying it on the results for $1/\varepsilon$ iterations:

1. Let $(\rho_i)_{i \in [t^{\varepsilon}]} \in \mathbb{F}^{t^{\varepsilon}}$ be some canonical set of distinct fixed elements known to the prover and to the verifier. For each batch $(\chi_i, \theta_i)_{i \in [t^{\varepsilon}]}$, let $\gamma : \mathbb{F} \to \mathbb{F}^m$ be the unique degree-$(t^{\varepsilon} - 1)$ curve that passes through the batch of points $(\chi_i)_{i \in [t^{\varepsilon}]}$, such that $\forall i, \gamma(\rho_i) = \chi_i$. The prover sends the function $\hat{w} \circ \gamma : \mathbb{F} \to \mathbb{F}$ to the verifier.
2. Upon receiving a function $f : \mathbb{F} \to \mathbb{F}$ from the prover (supposedly, $f = \hat{w} \circ \gamma$), the verifier checks that f is a polynomial of degree $m \cdot (|\mathbb{H}| - 1) \cdot (t^{\varepsilon} - 1)$, and that $\forall i, f(\rho_i) = \theta_i$. If these tests pass, then the verifier chooses a random element $\rho \in \mathbb{F}$ and sends it to the prover.
3. The prover and the verifier continue to the next iteration, such that the batch $(\chi_i, \theta_i)_{i \in [t^{\varepsilon}]}$ was reduced to the single claim $(\gamma(\rho), f(\rho))$.

Completeness is trivial. For soundness, imagine a t^{ε}-ary tree, where each internal node represents a claim that is the result of the foregoing process when applied to its children. The leaves represent the original claims $(\chi_i, \theta_i)_{i \in [t]}$. Assume that the i^{th} claim is false, i.e., that $\hat{w}[\chi_i] \neq \theta_i$, and assume that for some cheating prover strategy, the probability that the output claim is correct is s. That is,

$$s = \Pr[\hat{w}[\chi] = \theta].$$

Observe that the i^{th} leaf represents a false claim but the root represents a correct claim, and consider the unique path that connects the root and the i^{th} leaf. There must exist some node in this path that represents a correct claim, but at least one of its children represents a false claim. We focus on the interactive process that is applied to these children. In this process, the only way for the prover to pass the tests in Step (2) is by sending a function f such that $f \neq \hat{w} \circ \gamma$ but $f(\rho) = (\hat{w} \circ \gamma)(\rho)$. This implies that f and $\hat{w} \circ \gamma$ are two distinct polynomials of degree at most $m \cdot (|\mathbb{H}| - 1) \cdot (t^{\varepsilon} - 1)$, and thus by the Schwartz-Zippel Lemma (see Lemma 3.2), we get that

$$\Pr[f(\rho) = \hat{w} \circ \gamma(\rho)] \leq \frac{m \cdot (|\mathbb{H}| - 1) \cdot (t^{\varepsilon} - 1)}{|\mathbb{F}|} \leq \frac{m \cdot |\mathbb{H}| \cdot t^{\varepsilon}}{|\mathbb{F}|}.$$

By a union bound over all $1/\varepsilon$ nodes in the path, we get that

$$s \leq \frac{m \cdot |\mathbb{H}| \cdot t^{\varepsilon}}{\varepsilon \cdot |\mathbb{F}|}.$$

Notice that sending or evaluating a polynomial over $\mathbb{F}$ with (total) degree at most $m \cdot (|\mathbb{H}| - 1) \cdot (t^{\varepsilon} - 1)$ takes $m \cdot (|\mathbb{H}| - 1) \cdot (t^{\varepsilon} - 1) \cdot \log(|\mathbb{F}|) < m \cdot |\mathbb{H}| \cdot t^{\varepsilon} \cdot \log(|\mathbb{F}|)$ bits or time. To conclude, observe that there are at most $t/(t^{\varepsilon} - 1) \leq t/\varepsilon$ nodes in the tree and that there are $2/\varepsilon$ messages, and the other complexity measures follow by construction. □

3.2 UOWHF and Merkle Tree

Definition 3.11 (UOWHF [NY89])**.** *Let κ be a security parameter, and let $\{n_{1_i}\}$ and $\{n_{0_i}\}$ be two polynomially related increasing sequences that depend on κ, such that for all $i, n_{0_i} \leq n_{1_i}$. Let $\mathcal{H}_k$ be a collection of functions such that for all $h \in \mathcal{H}_k, h : \{0,1\}^{n_{1_k}} \to \{0,1\}^{n_{0_k}}$.*

Let A be a probabilistic $\mathsf{poly}(\kappa)$-time algorithm that on input 1^k outputs $x^{(1)} \in \{0,1\}^{n_{1_k}}$ that we call an initial value*, then given a random $h \in \mathcal{H}_k$ attempts to find $x^{(2)} \in \{0,1\}^{n_{1_k}}$ such that $h(x^{(1)}) = h(x^{(2)})$ but $x^{(1)} \neq x^{(2)}$. Such an $\mathcal{H} = \mathcal{H}_k$ is called a* family of universal one-way hash functions (UOWHFs) *if for all $\mathsf{poly}(\kappa)$-time probabilistic algorithms A, the following holds for sufficiently large k:*

1. *If $x^{(1)} \in \{0,1\}^{n_{1_k}}$ is A's initial value, then $\Pr[A(h, x^{(1)}) = x^{(2)}, h(x^{(1)}) = h(x^{(2)}), x^{(2)} \neq x^{(1)}] = negl(n_{0_k})$ where the probability is taken over all $h \in \mathcal{H}$ and the random choices of A.*

2. *$\forall h \in \mathcal{H}$ there is a description of h of length polynomial in n_{1_k}, such that given h's description and x, $h(x)$ is computable in polynomial time.*

3. *$\mathcal{H}$ is accessible: there exists an efficient algorithm G such that on input 1^k, G generates uniformly at random a description of $h \in \mathcal{H}$.*

We note that we treat $\mathcal{H}$ as a collection of descriptions *of functions.*

Note that we allow A to be non-uniform, namely, we assume that the UOWHFs are secure against non-uniform adversaries. This means that the argument systems that we construct in this work will be secure against non-uniform provers.

Furthermore, throughout this work we take $\kappa = n_{0_k}$. We comment that we could take κ to be smaller, i.e., $(n_{0_k})^{\varepsilon}$ for $\varepsilon \in (0,1)$, however taking $\kappa = n_{0_k}$ is sufficient. This means that the UOWHF families that we will consider will be secure against polynomial time adversaries (i.e., $\mathsf{poly}(\kappa) = \mathsf{poly}(n_{0_k})$).

As mentioned in the Overview (see Section 2), Rompel gave the first construction of UOWHFs from arbitrary one-way functions, while Katz and Koo gave the first full proof for Rompel's construction.

Theorem 3.12 ([Rom90, KK05])**.** *The existence of one-way functions implies the existence of universal one-way hash functions.*

In what follows, we define a Merkle tree and a valid path in the standard way. Definition 3.15 gives a relaxed security property (when comparing to CRHs) of a commitment scheme when based on a Merkle tree, called a "2-PLOSC". Proposition 3.16 shows that instantiating a Merkle tree with a family of UOWHFs results in a 2-PLOSC, and thus the existence of one-way functions suffices for constructing a 2-PLOSC. As mentioned in the overview, this construction can be thought of a targeted-collision-resistant hash with local opening. Although we refer to it by the term "commitment", it is not a standard commitment scheme in the sense that it is not necessarily hiding, and that its security property is only a *targeted* binding property.

Construction 3.13 (Merkle Tree)**.** *Fix $N, n_{in}, n_{out} \in \mathbb{N}$ and a finite alphabet Σ. Set*[8] *$L = \frac{N}{n_{in}}$, and let $\ell = \ell(N, n_{in}, n_{out})$ to be defined below. Given a string $z \in \Sigma^N$ and functions $h_1, \ldots, h_\ell : \Sigma^{n_{in}} \to \Sigma^{n_{out}}$, the* Merkle Tree *$T = T(z, h_1, \ldots, h_\ell)$ is defined in the following manner:*

- *The tree has $(\ell + 1)$ layers. The first layer is the root and the last layer is the leaves;*
- *There are L leaves. For $j = 1, \ldots, L$, the j^{th} leaf is denoted by $c_{(\ell+1,j)}$ and contains $z[(j-1)n_{in}] \ldots z[j \cdot n_{in} - 1]$;*
- *For $i \in [\ell]$, the i^{th} layer is denoted by w_i and is created by applying h_i on the $(i+1)^{st}$ layer: The j^{th} block in the i^{th} layer is denoted by $c_{(i,j)}$ and contains $h_i(c_{(i+1,j)})$ for $j = 1, \ldots, \frac{|w_{i+1}|}{n_{in}}$;*
- *The root is denoted by $y = h_1(c_{(2,1)})$, thus $w_1 = y$.*

We comment about the indexing of the tree nodes and blocks. As the tree is (n_{in}/n_{out})-ary, it is most convenient for us to index "chunks" of n_{in} nodes, which we call a *block*: namely, for each fixed tree layer i, the blocks in this layer are indexed as $c_{(i,1)}, \ldots, c_{(i,M)}$ for $M = |w_i|/n_{in}$. In other words, each tree node is coupled together with its $(n_{in}/n_{out} - 1)$ siblings. We stress that we do not give an explicit notation for each tree *node*, only to blocks of nodes. In particular, this means that there is no explicit indexing for the image of each block, e.g., for $h_i(c_{(i+1,j)})$, but we will not need one. In comparison to a binary tree, this indexing is equivalent to coupling each pair of nodes (that are siblings, i.e., mapped together to the next layer by the hash function) to a block, and only index the block.

Definition 3.14 (Valid Path)**.** *Let $N, n_{in}, n_{out}, \Sigma, L, \ell$ and $h_1, \ldots, h_\ell$ be as in Construction 3.13. Given a string $y \in \Sigma^{n_{out}}$ and a leaf index $q' \in [L]$, a path $p = (p_1, \ldots, p_{\ell+1})$ is called a* valid path from q' to y *if it satisfies the following properties:*

- $\forall i \in [\ell + 1] : p_i \in \Sigma^{n_{in}}$;
- $\forall i \in [\ell] : p_i = h_i(p_{i+1})$;
- $p_1 = y$.

Definition 3.15 (2-PLOSC)**.** *Let $N, n_{in}, n_{out}, \Sigma, \ell$ and $h_1, \ldots, h_\ell$ be as in Construction 3.13. Assume that $n_{out} \ll N$. A* Second-Preimage Locally Openable Succinct[9] Commitment using a Merkle tree *for strings in Σ^N is defined via a pair of probabilistic polynomial-time algorithms $(\mathcal{S}, \mathcal{R})$ such that:*

[8] We assume that N, as well as any other layer length, is a multiple of n_{in}. This is always possible by padding, and does not hurt the security since the all-zero string is a fixed string, and therefore the UOWHF security condition applies to it as well.

[9] We call this scheme *succinct* because the output of the commit phase is small.

- **Commit Phase:**
  - $\mathcal{S}$ *chooses a string* $z \in \Sigma^N$;
  - $\mathcal{R}$ *sends hash functions* $h_1, \ldots, h_\ell : \Sigma^{n_{in}} \to \Sigma^{n_{out}}$;
  - $\mathcal{S}$ *sends a commitment* $y \in \Sigma^{n_{out}}$*: the (correct) hash root of the Merkle tree* $T = T(z, h_1, \ldots, h_\ell)$.
- **Local-Opening Phase:** $\mathcal{S}$ *outputs an index* $q \in [N]$ *and an opening of a leaf* $q' = \left\lceil \frac{q}{n_{in}} \right\rceil$, *that is, a path* $p = (p_1, \ldots, p_{\ell+1})$.
- **Security:** *For a sufficiently large* n_{in} *and for all* $q' = \left\lceil \frac{q}{n_{in}} \right\rceil$,

$$\Pr_{\substack{\mathcal{S} \text{ coins} \\ h_1,\ldots,h_\ell}} [p \text{ is a valid path from } q' \text{ to } y \text{ and } p_{\ell+1} \neq z[(q'-1)n_{in}] \ldots z[q' \cdot n_{in} - 1]] = negl(n_{out}).$$

The following proposition shows that a UOWHF tree is a 2-PLOSC. The full proof is given in [AR23].

Proposition 3.16 ([AR23, Claim 3.6])**.** *In the setting of Definition 3.15, if* R *samples* $h_1, \ldots, h_\ell$ *uniformly at random from a* UOWHF*s family*[10] $\mathcal{H} : \Sigma^{n_{in}} \to \Sigma^{n_{out}}$*, and* n_{in} *and* N *are polynomially related, then the commitment scheme is secure. In other words, given the functions, the leaves and the correct root for them, it is impossible for* $\mathcal{S}$ *to find an index and a second valid opening for it.*

Remark 3.17 (The alphabet of the UOWHF tree)**.** *As already mentioned in Footnote 10, in this work we will only use* $\Sigma = \mathbb{GF}[2]$*. However, the strings that we want to hash — that is, to use as leaves for the* UOWHF *tree — are, in some cases, over a finite field* $\mathbb{F}$ *(that extends* $\mathbb{GF}[2]$*). Thus, given some* $z \in \mathbb{F}^N$ *that we wish to hash, the leaves are actually going to be* $z' \in \{0,1\}^{N \cdot \log(|\mathbb{F}|)}$ *where* z' *is the binary representation of* z*,*[11] *and this detail is going to be implicit in what follows.*

In this setting, a valid opening may consist of sending $\log(|\mathbb{F}|)$ *paths, one per each bit in the binary representation of the leaf index (the one to be opened). However, since these bits are* consecutive *in* z'*, and since a block size* n_{in} *will be bigger than the binary representation of an index (namely,* $n_{in} > \log(|\mathbb{F}|)$*), an opening will only consist up to two paths. To facilitate the reading, we also omit this detail, and only refer to a single path per opening, and to leaf indices in* $[N]$.

Throughout this work, the protocols we construct extensively use the commitment scheme described above. We formally define the language $\mathcal{L}_{\mathsf{hashroot}}$, the language of valid commitments, and then prove that it is computed by a sufficiently small and uniform family of Boolean circuits. Notice that we always hash (i.e., use as leaves) the *low degree extension* of the string being committed to. This will become useful in the interactive setting, when delegating the verification of a commitment to the prover.

Definition 3.18. *The language* $\mathcal{L}_{\mathsf{hashroot}}$ *is parameterized by an ensemble* $(\mathbb{F}, \mathbb{H})_n$*, that defines the low degree extension encoding* $\mathsf{LDE} = \mathsf{LDE}_{\mathbb{F},\mathbb{H}}$*, by the integers* $(n_{in}, n_{out}, n_h, M)_n$*, and by a family*

[10] Definition 3.11 considers a UOWHF as a Boolean function. Here we allow the functions to be over some finite alphabet Σ, and the definition is extended in the natural way. In our protocol, however, we take $\Sigma = \mathbb{GF}[2]$.

[11] Since $\mathbb{F}$ extends $\mathbb{GF}[2]$, we can take the natural binary representation: the vector of coefficients of x.

of functions $\mathcal{H} : \{0,1\}^{n_{in}} \to \{0,1\}^{n_{out}}$. *Take* $\ell + 1$ *to be the number of layers in the Merkle tree (see Construction 3.13) whose leaves are* $\mathsf{LDE}(w)$,[12] *with respect to* $\mathcal{H}$.

The explicit input to the language is $h = h_1 \circ ... \circ h_\ell$, *the concatenation of the length-*n_h *descriptions of* ℓ *functions chosen from* $\mathcal{H}$, *and a string* $y \in \{0,1\}^{n_{out}}$. *The holographic input is a string* $w \in \{0,1\}^M$, *where* $M \leq \mathsf{poly}(n)$.

YES *instances of the language are all triplets* $\big(w, (y, h)\big)$ *such that* y *is the correct hash root of* $\mathsf{LDE}(w)$ *with respect to* h.

The following proposition shows that $\mathcal{L}_{\mathsf{hashroot}}$ is computable by an $\ell \cdot \mathsf{poly}(n_{in})$-depth family of circuits of polynomial size.

Proposition 3.19. *The language* $\mathcal{L}_{\mathsf{hashroot}}$ *as defined in Definition 3.18 is computable by* Logspace*-uniform Boolean circuits with fan-in 2, of depth* $\ell \cdot \mathsf{poly}(n_{in})$ *and size* $\mathsf{poly}(n)$.

Proof. First, since computing each of the UOWHFs can be done by a $\mathsf{poly}(n_{in})$-time Turing machine, it can also be done by a Logspace-uniform circuit with fan-in 2 of size $\mathsf{poly}(n_{in})$ (by considering the circuit computing the tableau of the machine). This means that a hash root is computable by a fan-in 2, depth $\ell \cdot \mathsf{poly}(n_{in})$ and size $\mathsf{poly}(n, M) = \mathsf{poly}(n)$ circuit, by considering the circuit computing the UOWHF tree. It follows that this circuit is also Logspace-uniform, since it merely computes a Merkle tree w.r.t. the UOWHFs, which are computable by Logspace-uniform circuits as described above. □

3.3 Interactive Arguments

An interactive protocol consists of a pair $(\mathcal{P}, \mathcal{V})$ of interactive Turing machines that are run on a common input x, whose length we denote by $n = |x|$. The first machine, which is deterministic, is called *the prover* and is denoted by $\mathcal{P}$, and the second machine, which is probabilistic, is called *the verifier* and is denoted by $\mathcal{V}$.

An $(r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}, \Sigma)$-interactive protocol, is an interactive protocol in which, for inputs of length n, the parties interact for $r = r(n)$ rounds and each message sent from $\mathcal{P}$ to $\mathcal{V}$ (resp., $\mathcal{V}$ to $\mathcal{P}$) is in Σ^a (resp., Σ^b), where $\Sigma = \Sigma(n)$ is an alphabet (whose size may depend on n). The verifier runs in time $\mathcal{V}\mathsf{time}$ and the prover runs in time $\mathcal{P}\mathsf{time}$. We typically omit Σ from the notation and refer to $(r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$ interactive protocols when Σ is clear from the context.

In an $(r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}, \Sigma)$-interactive protocol, in each round $i \in [\ell]$, first $\mathcal{P}$ sends a message $\alpha^{(i)} \in \Sigma^a$ to $\mathcal{V}$ and then $\mathcal{V}$ sends a message $\beta^{(i)} \in \Sigma^b$ to $\mathcal{P}$. At the end of the interaction $\mathcal{V}$ runs a (deterministic) Turing machine on input $\big(x, r, \big(\alpha^{(1)}, \ldots, \alpha^{(\ell)}\big)\big)$, where r is its random string and outputs the result. Abusing notation, we denote the result of the computation by $\mathcal{V}\big(x, r, \big(\alpha^{(1)}, \ldots, \alpha^{(\ell)}\big)\big)$. We also denote by $\mathcal{P}\big(x, i, \big(\beta^{(1)}, \ldots, \beta^{(i-1)}\big)\big)$ the message sent by $\mathcal{P}$ in round i given input x and receiving the messages $\beta^{(1)}, \ldots, \beta^{(i-1)}$ from $\mathcal{V}$ in rounds $1, \ldots, i-1$, respectively. We emphasize that $\mathcal{P}$'s messages depend only on the input x and on the messages that it received from $\mathcal{V}$ in previous rounds.

The communication complexity of an $(r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}, \Sigma)$-interactive protocol is the total number of bits transmitted. Namely, $(\ell \cdot (b + a) \cdot \log_2(|\Sigma|))$.

[12] We refer to $\mathsf{LDE}(w)$ as the leaves of the tree, although it is not a binary string. See Remark 3.17 for details.

Public-coin Protocols. In this work we focus on public-coin interactive protocols, which are interactive protocols in which each message $\beta^{(i)}$ sent from the verifier to the prover is a uniformly distributed random string in Σ^b. At the end of the protocol, $\mathcal{V}$ decides whether to accept or reject as a function of x and the messages $\alpha^{(1)}, \beta^{(1)}, \ldots, \alpha^{(r)}, \beta^{(r)}$.

Definition 3.20 (ε-sound $(r, a, b, \mathcal{P}\text{time}, \mathcal{V}\text{time})$-IA)**.** *An $(r, a, b, \mathcal{P}\text{time}, \mathcal{V}\text{time})$-interactive protocol $(\mathcal{P}, \mathcal{V})$ (as above) is an ε-*sound *$(r, a, b, \mathcal{P}\text{time}, \mathcal{V}\text{time})$-*Interactive Argument (IA) *for $\mathcal{L}$ if:*

- **Completeness:** *For every $x \in \mathcal{L}$, if $\mathcal{V}$ interacts with $\mathcal{P}$ on common input x, then $\mathcal{V}$ accepts with probability* 1.[13]
- **ε-Soundness:** *For every $x \notin \mathcal{L}$ and every computationally bounded cheating prover strategy $\mathcal{P}^*$, the verifier $\mathcal{V}$ accepts when interacting with $\mathcal{P}^*$ with probability less than $\varepsilon(|x|)$, where $\varepsilon = \varepsilon(n)$ is called the* soundness error *of the argument system.*

We remark that our definition of interactive arguments emphasizes the parameters of the argument system (to a higher degree than is commonly done in the literature). This is mainly because throughout this work we apply transformations to interactive arguments and we need to carefully keep track of the effect of these transformations on each one of these parameters.

Parallel Repetition. The following theorem shows that, in case of public-coin arguments, the soundness error can be reduced at an exponential rate using parallel repetition.

Theorem 3.21 ([BIN97, Hai09, CP15])**.** *Let κ be a security parameter. Running γ parallel repetitions of a public-coin argument with a soundness error $s \in (0, 1)$ reduces the error at an exponential rate to $s^\gamma + negl(\kappa)$. The perfect completeness still holds.*

3.4 HIAs and a constant-round HIA for bounded-depth computations

In this work, we use an argument system of a special type, called a *Holographic Interactive Argument* (HIA), a notion formally defined in the work of Gur and Rothblum [GR17]. A HIA is defined similarly to a standard interactive argument, except that the verifier, rather than being given access to the input, is given access to its *encoding*. As a matter of fact, for the low degree extension encoding (see Section 3.1), reading just a single point r from the encoded input suffices for the verifier. Alternatively, instead of having the verifier actually read the (encoded) input at the point r, the verifier outputs a claim about the point, i.e., it outputs r together with a value v that it would have expected to see, had it actually queried the (encoded) input at r.

We give the definition of a HIA for pair languages. On input (x, w), we interpret x as the explicit input and w as the holographic input. The prover gets them while the verifier only gets $(x, |w|)$. The claim about the encoding of the input is only about the holographic input w. We comment that the original definition was given in the information theoretic setting (namely, for an interactive proof), but it extends naturally to interactive arguments by bounding the computational resources of the cheating prover.

[13] One could allow an error also in the completeness condition. For simplicity, and since all our protocols do not have such an error, we require perfect completeness.

Definition 3.22 (Holographic Interactive Proof/Argument (HIP/HIA)). *Fix finite fields $\mathbb{H} \subseteq \mathbb{F}$ and a low degree extension encoding $\mathsf{LDE} = \mathsf{LDE}_{\mathbb{F},\mathbb{H}}$.*

A Holographic Interactive Proof/Argument *for a pair language $\mathcal{L}$, with respect to the low degree extension* LDE*, is an interactive protocol with two parties: a computationally unbounded (reps., bounded) prover $\mathcal{P}$ and a computationally bounded verifier $\mathcal{V}$. Both parties get as input $x \in \{0,1\}^{n_{exp}}$. The prover also gets $w \in \{0,1\}^{n_{hol}}$ whereas the verifier only gets $|w| = n_{hol}$.*

At the end of the interaction, either the verifier rejects or it outputs a coordinate $r \in \mathbb{F}^{m(n_{hol})}$, and a value $v \in \mathbb{F}$, such that:

- **Completeness.** *If $(x,w) \in \mathcal{L}$ and the prover honestly follows the protocol, then $\mathsf{LDE}(w)[r] = v$.*
- **Soundness.** *If $(x,w) \notin \mathcal{L}$, then for any unbounded (reps., bounded) cheating prover, with probability at least $1/2$ over the verifier's coins, $\mathsf{LDE}(w)[r] \neq v$.*

Next, we present the protocol of [AR23, Theorem 5.1], referred to as flat-GKR (since it takes the [GKR08] interactive proof and "flattens" its rounds complexity). It is a constant-round HIA for any language computable by bounded-depth circuits, either arithmetic (over some finite field $\mathbb{F}$ that extends $\mathbb{GF}[2]$) or Boolean. A few remarks regarding the theorem statement are in order. First, in the original protocol, $|\mathbb{F}|$ is taken to be a sufficiently large polynomial, and the soundness error is always smaller than $1/\mathsf{poly}(n)$. Then, a tighter analysis is presented in Appendix A.2.1 of that work, allowing to take a smaller field size $|\mathbb{F}| \geq \Theta(n^{2\delta})$. This way, the soundness error is only as small as $1/n^{\delta}$. Nonetheless, we can reduce the error to be as small as $1/p(n)$ for any known polynomial $p(n)$: assume $p(n) \leq n^{\nu}$ for some $\nu \in \mathbb{N}$ and perform $O(\nu/\delta)$ parallel repetitions. These will not increase the complexity measures, however, using Theorem 3.21, will reduce the soundness error to $1/p(n)$ as desired.

Secondly, in what follows we state the holographic version of the protocol, for pair languages, since it is the one that will be in use throughout this work. Note that we let the soundness error and the complexities depend on n, which is the length of the explicit input.

Lastly, notice that, in order to comply with Definition 3.20, we extend the result such that the length of prover's and verifier's messages are visible. We do so by trivially bounding these with the communication complexity.

Theorem 3.23 (flat-GKR [AR23]). *Assume one-way functions exist, and let $\delta \in (0,1)$ be a constant. Let $\mathbb{H} \subseteq \mathbb{F}$ be (ensembles of) extension fields of $\mathbb{GF}[2]$, where $|\mathbb{H}| = \Theta(n^{\delta})$ and $|\mathbb{F}| \geq \Theta(n^{2\delta})$.*

Let $\mathcal{L}$ be a pair language with inputs of the form $(x,w) \in \{0,1\}^n \times \{0,1\}^M$. Assume that $\mathcal{L}$ is computable by Logspace*-uniform circuits with fan-in 2, of depth $D = D(n,M)$ and size $\mathsf{poly}(n,M)$. There is an ε-sound doubly-efficient $(r,a,b,\mathcal{P}\mathsf{time},\mathcal{V}\mathsf{time})$-HIA for $\mathcal{L}$ w.r.t. the low degree extension $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}$, that is public-coin and has perfect completeness, with the following parameters:*

- $\varepsilon = 1/p(n)$, *for any* $p(n) \leq \mathsf{poly}(n)$,
- $r = O(1/\delta^3)$.
- $a = D \cdot n^{O(\delta)}$.
- $b = D \cdot n^{O(\delta)}$.
- $\mathcal{P}\mathsf{time} = \mathsf{poly}(n,M)$.

- $\mathcal{V}\mathsf{time} = (D+n) \cdot n^{O(\delta)}$.

Note that, as discussed after Definition 3.11, we assume that the one-way functions are secure against non-uniform adversaries, and consequently get that the HIA is secure against non-uniform provers.

Remark 3.24 (flat-GKR on multi-output circuits)**.** *The protocol also applies to cases where we want to verify the output of a Boolean or arithmetic circuit that has multiple (say $\ell \geq 1$) output wires. By convention, we will verify that the value of all output wires is 1 (where 1 is either a Boolean value or a field element). The choice of 1^ℓ is arbitrary; any vector that is known to the verifier would work (however, it is crucial that the verifier knows this output vector in advance).*

Proposition 3.25 ($\mathcal{P}$'s messages in flat-GKR are computable in low-depth)**.** *Let $\mathbb{F}$ be an extension field of $\mathbb{GF}[2]$ where $|\mathbb{F}| = \Theta(n^{2\delta})$, and let $\mathcal{L}$ be a pair language with inputs of the form $(x,w) \in \{0,1\}^n \times \{0,1\}^M$. Assume that $\mathcal{L}$ is computable by* Logspace*-uniform circuits with fan-in 2, of depth $D = D(n,M)$ and size* poly(n,M)*. Then, the prover's messages in the* flat-GKR *protocol are computable by* Logspace*-uniform arithmetic circuits over $\mathbb{F}$ with fan-in 2, of depth* poly(D, n^δ) *and size* poly(n,M)*.*

The proof of Proposition 3.25 is deferred to Appendix B.1.

3.5 Code Switching for Tensor Codes

In this section we present a "code-switching" technique, introduced in the work of Ron-Zewi and Rothblum [RR19]. Loosely speaking, this technique allows to "switch" between different tensor codes, using the Sumcheck protocol [LFKN92].

As already mentioned in Section 3.1, the low degree extension encoding is a tensor code (in fact, it is the m^{th} tensor of a the Reed-Solomon code, that we do not define here). In their work, [RR19] define another family of tensor codes, of a higher rate, that supports a certain "code-switching" property. This allows them to use a (low-rate) code (namely, the low degree extension) to check the correctness of an NP statement, but then use a high-rate tensor code for actually encoding the witness. We follow this approach, which allows us to save in communication complexity.

We denote this high-rate tensor code by $\mathcal{R}$ and index it by an integer $n \in \mathbb{N}$. Taking $M = M(n)$ to denote its message length, the encoding satisfies a few properties that will come in hand in Section 4: it is linear, locally-testable, computable in NC (see Proposition A.1), and satisfies the "code-switching" property, which means that claims about a string encoded under the low degree extension can be converted to claims about the encoding of the string under $\mathcal{R}$. Most importantly, the encoding of an M-bit string only requires $O(M)$ bits. The existence of $\mathcal{R}$ as stated next was proved in the work of [RR19]; see Appendix A.1 for more details about this encoding and its construction.

Theorem 3.26 (The encoding $\mathcal{R}$ [RR19])**.** *Let $\gamma \in (0,1)$ be a constant and let t be a positive constant integer. There exists a locally-testable linear encoding $\mathcal{R} : \{0,1\}^M \to \{0,1\}^{M'}$ with rate at least $1-\gamma$, such that $M' = \left(\frac{1}{1-\gamma} \cdot M\right)$. Its relative distance is $\left(\frac{\gamma}{t}\right)^{O(t)}$, and the encoding can be computed in* NC*.*

We note that $\mathcal{R}$ has many other useful properties, and Theorem 3.26 merely highlights a few. The following lemma captures the local testability of the code. In what follows, we always take $t = O(1/\delta)$ for a constant $\delta \in (0,1)$.

Lemma 3.27 (Local testing for $\mathcal{R}$ [RR19, Lemma 7.2])**.** *Let $n \in \mathbb{N}$ and $M = M(n) = \mathsf{poly}(n)$. Let $\delta, \gamma \in (0,1)$ be two constants and take $\mathcal{R}$ as defined in Theorem 3.26.*

For any $\varepsilon > 0$, there exists a randomized algorithm A that gets oracle access to $z \in \{0,1\}^M$ and a proximity parameter $\alpha \in (0,1)$. It makes $O\left(n^{2\delta} \cdot \gamma \cdot \delta \cdot \frac{1}{\alpha} \cdot \log\left(\frac{1}{\varepsilon}\right)\right)$ queries, such that:

- **Completeness.** *If z is a codeword of $\mathcal{R}$, then A accepts w.p. 1.*
- **Soundness.** *If z is α-far from the code $\mathcal{R}$, then A rejects w.p. at least $1 - \varepsilon$.*

HIPs with respect to $\mathcal{R}$. We extend the definition of a holographic interactive proof (HIP, see Definition 3.22) for the encoding $\mathcal{R}$. As discussed in Section 3.4, the original definition leverages the fact that multiple claims about the low degree extension of a string can be (soundly) reduced to a single claim about it (see Proposition 3.10). Consequently, reading a *single* point in the low degree extension of the holographic input suffices for the verifier. However, this does not hold in general for other families of error-correcting codes. We generalize Definition 3.22 in the natural way: a HIP w.r.t. the encoding $\mathcal{R}$, denoted $\mathsf{HIP}_{\mathcal{R}}$, outputs *a sequence* of claims $(\chi_j, \theta_j)_{j \in [q]}$ about the encoding of its input under $\mathcal{R}$. We account for q in the verification time.

The following lemma uses this generalized notion. We refer to it as the "code-switching $\mathsf{HIP}_{\mathcal{R}}$" since it reduces a claim about the encoding of a string under another code (in particular, the LDE encoding) to multiple claims about the encoding of the string under $\mathcal{R}$. It is essentially a restatement of [RR19, Lemma 7.1]. The full details are deferred to Appendix A.2.

Lemma 3.28 (Code-Switching $\mathsf{HIP}_{\mathcal{R}}$)**.** *Let $\delta, \gamma \in (0,1)$ be two constants. Take $\mathbb{H} \subseteq \mathbb{F}$ to be (ensembles of) extension fields of $\mathbb{GF}[2]$, such that $\mathbb{H} \subseteq \mathbb{F}$, and take $m = \log_{|\mathbb{H}|}(M)$. Let $\mathsf{LDE} = \mathsf{LDE}_{\mathbb{F},\mathbb{H}} : \{0,1\}^{|\mathbb{H}|^m} \to \mathbb{F}^{|\mathbb{F}|^m}$ be the low degree extension encoding, and let $\mathcal{R} : \{0,1\}^M \to \{0,1\}^{M'}$ be the encoding defined in Theorem 3.26, with rate parameter γ, such that $M' = \left(\frac{1}{1-\gamma} \cdot M\right)$.*

Then, there exists a $\mathsf{HIP}_{\mathcal{R}}$, with explicit input $(r, v) \in \mathbb{F}^m \times \mathbb{F}$ and holographic input $w \in \{0,1\}^M$. It outputs $q = q(\delta, \gamma, n) = \mathsf{polylog}(n)$ claims $(\chi_j, \theta_j)_{j \in [q]} \in \left(\{0,1\}^{\log(M')} \times \{0,1\}\right)^q$, such that:

- **Completeness.** *If $\mathsf{LDE}(w)[r] = v$ and the prover honestly follows the protocol, then, for any $j \in [q]$, $\mathcal{R}(w)[\chi_j] = \theta_j$.*
- **Soundness.** *If $\mathsf{LDE}(w)[r] \neq v$, then, for any unbounded cheating prover, with probability at least $1/n$ over the verifier's coins $\exists j \in [q]$ such that $\mathcal{R}(w)[\chi_j] \neq \theta_j$.*

The number of rounds is $O(1/\delta)$, the communication complexity is $O\left(q \cdot \log|\mathbb{F}| \cdot |\mathbb{H}|\right)$, the verifier runs in time $\mathsf{poly}\left(q, \log|\mathbb{F}|, |\mathbb{H}|, m\right)$ and the prover runs in time $O(M) \cdot q \cdot \mathsf{polylog}(|\mathbb{F}|)$.

3.6 Probabilistically Checkable Interactive Arguments (PCIAs)

Loosely speaking, Probabilistically Checkable Interactive Proofs (PCIPs), also known as Interactive Oracle Proofs, are public-coin interactive protocols in which the verifier only queries the input and transcript at few locations. This notion was introduced by two independent works [RRR16, BCS16], and is naturally extended to interactive arguments by bounding the computational resources of the cheating prover. We view a PCIA as a two-step process: first, an interactive protocol is executed, where the verifier sends messages (which are merely random strings) to the prover, and receives

in return messages from the prover. In the second step, the verifier queries just a few points in the transcript and input (without any further interaction with the prover), and decides whether to accept or reject. When referring to the verifier's running time we will refer only to the running time in the second step. In particular, the running time will typically be sub-linear in the transcript length (which is obviously impossible if we counted also the first step).

Definition 3.29 (ε-sound $(q_T, q_I, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIA)**.** *A* public-coin $(r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-IA *is an* ε-sound $(q_I, q_T, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$ Probabilistically Checkable Interactive Argument (PCIA) for $\mathcal{L}$ *if the interaction consists of the following three phases:*

1. **Communication Phase:** *First, the two parties interact for r rounds, in which $\mathcal{V}$ only sends random strings (of length b). No further interaction takes place after this phase.*
2. **Query Phase:** *$\mathcal{V}$ makes adaptive queries to its input and output. The number of queries to the transcript (resp., input) is at most $q_T = q_T(n)$ (resp., $q_I = q_I(n)$).*
3. **Decision Phase:** *Based on the answers to its queries and the random messages that it sent in the communication phase, $\mathcal{V}$ decides whether to accept or reject.*

In contrast to interactive protocols, here $\mathcal{V}\mathsf{time}$ *refers to $\mathcal{V}$'s running time only in the* query *and* decision *phases.*

We stress that the query complexity is not multiplied by $\log_2(|\Sigma|)$ because the transcript is over the alphabet Σ.

Holographic Access to Input. In Definition 3.29 we bound the verifier's queries both to the transcript *and to the input*. It is natural for the verifier to read its entire input, making a linear number of queries and achieving $q_I = n$, whereas the number of queries to the transcript is typically sublinear. For example, this is the case in classical PCP proofs for NP (which can be viewed as single-message PCIPs).

If the input is encoded under an error-correcting code, it is often possible for the verifier to read only a sub-linear portion of the input. Indeed, it is known in the PCP literature [BFLS91] that query access to the low degree extension of the input suffices for sub-linear running time in the PCP setting, and the same is true also for PCIPs. Following [BFLS91, GR17] and as discussed in Section 3.4, we refer to this type of access to the input as *holographic*.

In the setting of a sub-linear verifier, which will be our focus in much of this work, the numbers of queries to the transcript and input are both sublinear. Often, we do not need to distinguish between the two bounds, using a single parameter q to bound both quantities. For convenience, in what follows we use the notation $(q, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIA to refer to a PCIA where the numbers of verifier queries to the transcript and to the input are both bounded by q (i.e., a $(q, q, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIA as per Definition 3.29).

Definition 3.30 (Holographic PCIA)**.** *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible finite ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$. A* Holographic ε-sound $(q_I, q_T, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$ Probabilistically Checkable Interactive Argument (PCIA) for $\mathcal{L}$ *with $(\mathbb{H}, \mathbb{F})$-encoded input, is defined similarly to a $(q_I, q_T, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIA (see Definition 3.29), except that the verifier has oracle access to the low degree extension $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$ of the input x. The parameter q_I corresponds to the number of queries made to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$.*

Notice that in Section 3.4 we restrict the number of the queries to the encoded (holographic) input to a single query. In many cases, this facilitates the composition of such protocols: the HIA outputs a claim about the holographic input, and after the HIA execution ends, this claim is verified in some manner by another protocol. Since we use the LDE encoding, a holographic PCIA can satisfy this property as well (namely, we can always restrict it to $q_I = 1$) without losing its power.[14] However, since we construct PCIAs (i.e., instead of using them as a black-box), it will be more convenient for us to allow the verifier to query (the encoding of) the input multiple times, and account on these queries with the parameter q_I.

Input-Oblivious PCIAs. The PCIAs that we construct will have a restricted type of query access to their transcript (and we will leverage this query access in our proof). Intuitively, we would like the verifier's queries to the transcript to depend only on its random string (and not on previous queries to the input and transcript).

Definition 3.31. *We say that a* PCIP $(\mathcal{P}, \mathcal{V})$ *makes* input-oblivious queries *if for every two inputs* $x_1, x_2 \in \{0,1\}^n$ *and every random string* ρ*, the queries that the verifier* $\mathcal{V}$ *makes to the transcript on input* x_1*, and* x_2*, both with the random string* ρ*, are the same.*

PCIA for Pair Languages. Following Ben-Sasson *et al.* [BGH+06] we consider pair languages which are languages whose input is divided into two parts. We typically think of an input (w, x) to a pair language $\mathcal{L}$ as being composed of an explicit input w (to which the verifier has explicit access) and an implicit input x (to which the verifier only has implicit access). An ε-sound $(q_I, q_T, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIA for a *pair language* $\mathcal{L}$ is defined similarly to Definition 3.29, except that both the verifier and the prover have explicit access to the explicit part of the input (and the completeness and soundness condition are modified to accommodate this).

We extend the notion of input-oblivious queries (see Definition 3.31) to PCIPs for pair languages but allow the verifier to use the explicit input to generate its queries. More specifically, a PCIP for a pair language $\mathcal{L}$ makes input-oblivious queries if for every two implicit input x_1 and x_2 and explicit input w, for every random string ρ the verifier makes the same queries to the transcript on input (w, x_1) and on input (w, x_2).

Holographic PCIA for Pair Languages. The definition of *Holographic* PCIAs (Definition 3.30) is extended analogously for pair languages: in a holographic PCIA for a pair language, which is the model we use in this work, both parties have full access to the explicit input, where the verifier only has *holographic* access to the implicit input. In other words, it is a restricted notion of general PCIAs for pair languages, where we add the assumption that the implicit input is encoded. Note that, in this setting, we still refer to the implicit input — the one that the verifier have *holographic* access to — as *implicit*, to be consistent with the definition of a pair language. Indeed, this is somewhat inconsistent with the definition of a holographic PCIA (for languages with a single input), where we call the input to which the verifier have holographic access to as holographic.

[14] A procedure like the one described in Proposition 3.10, that reduces multiple claims about the LDE of a string into a single one, could have been used to reduce q_I claims into a single one, that the PCIA would output.

3.7 PCIAs w.r.t. Encoded Provers and Low-Depth Honest Prover

Encoded Provers. Notice that in the PCIA setting, full soundness cannot be obtained with small query complexity: If a cheating prover $\mathcal{P}^*$ changes just one bit of the i^{th} message, and the verifier only makes a small number of queries to the message, this change will likely go unnoticed and soundness is lost. To reconcile these two notions, we follow [RRR16] and restrict the family of cheating provers such that every message sent by the cheating prover (as well as by the prescribed prover) is a codeword in a high-distance error-correcting code (the low degree extension, see Section 3.1). We refer to this notion as PCIA *w.r.t. encoded provers*.

Definition 3.32 (Encoded Prover)**.** *We say that a prover strategy $\mathcal{P}$ is* encoded *if every message that $\mathcal{P}$ sends is a $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codeword, where $m = \log_{|\mathbb{F}|}(a)$, and a is the length of each prover message.*

We comment that RRR's definition for encoded provers is a strict generalization of the aforementioned one.[15]

The following proposition shows that we can (trivially) transform a PCIA into an IA with a slight loss in parameters. It follows directly from applying a low degree test on each of the prover's messages, and then using a self-correction procedure to encode its messages. The other direction (i.e., transforming an IA into a PCIA) is approached in Remark 5.3. Before stating it, we recall that the communication complexity of a $(r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}')$-IA was defined as $(\ell \cdot (b + a) \cdot \log_2(|\Sigma|))$.

Proposition 3.33 (From PCIA w.r.t. encoded provers to interactive arguments)**.** *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$. Let $\mathcal{L}$ be a language with inputs of length n, and let $\xi = \xi(n) \in (0, 1]$ be an error parameter. If $\mathcal{L}$ has an ε-sound $(q, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIA w.r.t. encoded provers, then $\mathcal{L}$ has an ε'-sound $(r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}')$ interactive argument, where*

$$\varepsilon' = \varepsilon + \xi,$$

and

$$\begin{aligned}\mathcal{V}\mathsf{time}' = \mathcal{V}\mathsf{time} &+ n \cdot \mathsf{poly}(|\mathbb{H}|, \log(|\mathbb{F}|), \log_{|\mathbb{H}|}(n)) \\ &+ O(|\mathbb{H}| \cdot m \cdot \log(|\mathbb{F}|) \cdot \log(2r \cdot q/\xi)) \cdot (r + q)\,.\end{aligned}$$

Proof. Let $(\mathcal{P}, \mathcal{V})$ be an an ε-unambiguous $(q, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$ PCIA for $\mathcal{L}$ w.r.t. encoded provers. Consider a protocol $(\mathcal{P}', \mathcal{V}')$ in which $\mathcal{P}'$ just emulates $\mathcal{P}$ and $\mathcal{V}'$ first computes the low degree extension $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$ of the main input x (or, to achieve a HIA (see Definition 3.22), uses its oracle access to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$) and emulates $(\mathcal{P}, \mathcal{V})$ w.r.t. that low degree extension. In order to emulate $\mathcal{V}$'s queries, the verifier $\mathcal{V}'$ first checks that each message $\alpha \in \mathbb{F}^a$ that it receives is an $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codeword by running an individual degree test Lemma 3.5, w.r.t. individual degree $|\mathbb{H}| - 1$, proximity parameter $1/100$ and soundness parameter $\xi/2r$. By Lemma 3.5 this can be done in time $r \cdot O(|\mathbb{H}| \cdot m \cdot \log(|\mathbb{F}|) \cdot \log(2r \cdot q/\xi))$.

[15] In particular, each prover messages is composed of a *sequence* of $g = g(n)$ strings, each of which is an LDE codeword. Using $g > 1$ is necessary there since, in their construction of the batching PCIP, the checksum has multiple rows, each of which is a codeword (as each checksum row is a linear combination of prover messages, which are all codewords). Had the prover sent the checksum in a single message, this message should have been an encoding of codewords, which incurs a polynomial blowup.

Then, it runs the self-correction procedure of Lemma 3.4 on q points, each w.r.t. total degree $(|\mathbb{H}|-1)\cdot m$, the same proximity parameter used for the low degree test and soundness parameter $\xi/2q$. This takes time $q \cdot O(|\mathbb{H}| \cdot m \cdot \log|\mathbb{F}| \cdot \log(2r \cdot q/\xi))$.

Completeness is trivial. For ε-soundness, observe that if a cheating prover for the PCIP sends any message that is far from an "encoded message" (i.e. a message that is not in the $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}$ code), then the verifier $\mathcal{V}'$ rejects with high probability when performing the low degree test on that message. Otherwise (i.e., the messages are close to valid codewords), since $\mathcal{V}'$ uses the self-correction procedure, we can essentially treat the messages as being valid encoded messages, which means that the prover is behaving like an encoded prover. Soundness now follows from the soundness of the underlying PCIA w.r.t. encoded provers, thus $\mathcal{V}'$ rejects with probability $1-\varepsilon$.

All the parameters of $(\mathcal{P}',\mathcal{V}')$ are the same as in $(\mathcal{P},\mathcal{V})$ except for the soundness error and the verifier's running time. The former increases by an additive factor of $r\cdot(\xi/2r)+q\cdot(\xi/2q)\leq\xi$ by a union bound over r rounds (for the low degree text) and the q queries (for the self-correction), where the latter increases by an additive factor of

$$n\cdot\mathsf{poly}(|\mathbb{H}|,\log_{|\mathbb{H}|}(n))+O(|\mathbb{H}|\cdot m\cdot\log(|\mathbb{F}|)\cdot\log(2r\cdot q/\xi))\cdot(r+q)\,,$$

due to computing the low degree extension of the input (see Proposition 3.6) and running the low degree tests of Lemma 3.5 and the self-correction procedures of Lemma 3.4. We note again that the first term is eliminated in case of a HIA (i.e., there is no need to compute the low degree extension of the input). □

Low-Depth Honest Prover. In fact, we will require an additional property from the PCIAs that we build: that the *honest* prover $\mathcal{P}$ is *low-depth*. We stress that being encoded is a property of both the cheating and the honest prover, whereas being low-depth is only a property of the honest prover. This means that low-depth does *not* restrict the PCIAs' soundness (i.e., to hold against a restricted class of cheating provers, as in the case of encoded provers).

Loosely speaking, *low-depth* means that given the tableau of the computation (that checks membership in the language that the PCIA computes) and the verifier's randomness, the prover's answers to the protocol can be computed by a low-depth circuit. However, in the batching protocol constructed in Section 5, the language to be batched includes verifying a claim about the tableau of another computation, whereas we want the input to the low-depth circuit to be this tableau (instead of the bigger tableau that includes verifying the claim). Therefore, we define the low-depth property of the honest prover w.r.t. a PCIA for this language. First, we formally define the aforementioned language.

Definition 3.34 (The language $\mathcal{L}_t^{\mathcal{M}}$)**.** *Let $n\in N$, and let $\mathbb{H}\subseteq\mathbb{F}$ be (ensembles of) finite fields. For a space $S=S(n)$ Turing machine $\mathcal{M}$ and a time bound $t=t(n)$, the pair language $\mathcal{L}_t^{\mathcal{M}}$ gets a claim (ω,θ) as explicit input and (x,u,v) as implicit input (see Section 3.6 for the precise technical definition of a holographic* PCIA *for pair languages). It is defined with respect to the low degree extension encoding $\mathsf{LDE}=\mathsf{LDE}_{\mathbb{F},\mathbb{H}}$ as follows:*

$$\mathcal{L}_t^{\mathcal{M}}\stackrel{def}{=}\left\{((\omega,\theta),(x,u,v)):\begin{array}{c}\text{On input } x\in\{0,1\}^n,\ \mathcal{M} \text{ moves from configuration}\\ u\in\{0,1\}^{O(S)} \text{ to configuration } v\in\{0,1\}^{O(S)}\\ \text{in exactly } t \text{ steps and } \mathsf{LDE}(\mathcal{T}_{u,v})[\omega]=\theta\end{array}\right\},$$

where $\mathcal{T}_{u,v} \in \{0,1\}^{t \cdot O(S)}$ *stands for the tableau of the computation from* u *to* v*, namely, the sequence of configurations starting from* u *and ending with* v*.*

Notice that the definition of $\mathcal{T}_{u,v}$ assumes that indeed the first condition is met. The low-depth property w.r.t. $\mathcal{L}_t^{\mathcal{M}}$ is formally captured by the following definition. Note that, since we only consider *encoded* provers, the circuit computing its messages is an *arithmetic* circuit. Moreover, since we use the flat-GKR theorem w.r.t. this circuit, we also require that this circuit is of fan-in 2, of polynomial size and sufficiently uniform.

Definition 3.35 (Low-Depth Prover in a PCIA for $\mathcal{L}_t$)**.** *Let* $\mathbb{F}$ *be a finite field. Let* $\mathcal{L} \in \mathsf{DTIME}(T(n))$ *be a language with inputs of length* n*, and let* $\mathcal{T}$ *denote the tableau of the length-*$T(n)$ *computation of verifying membership in* $\mathcal{L}$ *w.r.t. a Turing machine* $\mathcal{M}$*.*

For $\mu = \mu(n)$*, we say that a prover strategy* $\mathcal{P}$ *of an* r*-round* PCIA $(\mathcal{P}, \mathcal{V})$ *for* $\mathcal{L}_t^{\mathcal{M}}$ *(see Definition 3.34) is* μ-low-depth *if the sequence of messages that* $\mathcal{P}$ *sends in the* PCIA *for* $\mathcal{L}_t^{\mathcal{M}}$ *is computable by* Logspace*-uniform arithmetic circuits over* $\mathbb{F}$ *with fan-in 2, of depth* $\mathsf{poly}(\mu)$ *and size* $\mathsf{poly}(n)$*, whose input is the input to* $\mathcal{L}$*, the tableau* $\mathcal{T}$ *and a full sequence of* $\mathcal{V}$*'s random coins* $\beta_1, \ldots, \beta_r$*, and output is the sequence of message sent by* $\mathcal{P}$*.*

Throughout this work, we will always require that the provers in the PCIAs are low-depth, and indeed, all of the PCIAs we construct are for $\mathcal{L}_t^{\mathcal{M}}$ (when plugging in different time bounds t).

Query reduction. An important ingredient in constructing our PCIA is a "query reduction" transformation for PCIPs (w.r.t. encoded provers) that reduces the verifier's query complexity and running time. The following is essentially a restatement of [RRR16, Lemma 8.2] for PCIAs, when omitting the unambiguity condition (which is unnecessary in this work) and taking $g = 1$ (see Footnote 15). Moreover, the soundness error is refined from the original $\mathsf{poly}(|\mathbb{H}|)/|\mathbb{F}|$, by a more careful analysis (and without any changes in the protocol).[16]

Lemma 3.36 (Query-reduction for PCIPs w.r.t. encoded provers, [RRR16, Lemma 8.2])**.** *Take* $\mathbb{H}$ *and* $\mathbb{F}$ *to be constructible field ensembles where* $|\mathbb{F}| = \mathsf{poly}(|\mathbb{H}|)$*. Let* $\sigma = \sigma(n) \in (0,1)$ *be a* reduction parameter*.*

Let $(\mathcal{P}, \mathcal{V})$ *be an* ε*-sound* $(q, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP *for a language* $\mathcal{L}$ *w.r.t. encoded provers and with input-oblivious queries, where* $\log(\max(n, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})) \leq |\mathbb{H}| \leq \min(q, \mathcal{V}\mathsf{time})^{\sigma}$*.*

There exists an ε_{Q}*-sound* $(q_{\mathsf{Q}}, r_{\mathsf{Q}}, a_{\mathsf{Q}}, b_{\mathsf{Q}}, \mathcal{P}\mathsf{time}_{\mathsf{Q}}, \mathcal{V}\mathsf{time}_{\mathsf{Q}})$-PCIP *protocol* $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$ *for the language* $\mathcal{L}$ *w.r.t. provers and with input-oblivious queries, where:*

- $\varepsilon_{\mathsf{Q}} = \varepsilon + O\left((|\mathbb{H}| \cdot m)^2\right)/|\mathbb{F}|$.
- $q_{\mathsf{Q}} = \mathsf{poly}(\mathcal{V}\mathsf{time})^{\sigma} + O(r \cdot \log|\mathbb{F}|)$.
- $r_{\mathsf{Q}} = r + O(1/\sigma)$.
- $a_{\mathsf{Q}} = \max\left(a, \mathsf{poly}(r, b, \mathcal{V}\mathsf{time}, |\mathbb{H}|)\right)$.
- $b_{\mathsf{Q}} = \max\left(b, O(|\mathbb{H}| \cdot \log|\mathbb{F}|)\right)$.

[16] Indeed, the soundness error in RRR's query reduction originates in executions of the Sumcheck protocol [LFKN92], in which the soundness error is dominated by the degree of the polynomial that is given as input. In all cases, the polynomials are LDE codewords, thus their degree does not exceed $(|\mathbb{H}| \cdot m)$, and they are never composed with more than one other polynomial. See [RRR16, Lemma 7.3] for further details.

- $\mathcal{P}\mathsf{time}_{\mathsf{Q}} = \mathcal{P}\mathsf{time} + \mathsf{poly}(q, r, b, \mathcal{V}\mathsf{time})$.
- $\mathcal{V}\mathsf{time}_{\mathsf{Q}} = \mathsf{poly}(\mathcal{V}\mathsf{time})^{\sigma} + (\mathsf{poly}(b, r, |\mathbb{H}|))$.

Furthermore, for any $\mu = \mu(n) \geq \mathsf{polylog}(n)$, if $\mathcal{P}$ is μ-low-depth, then $\mathcal{P}_{\mathsf{Qreduce}}$ is also μ-low-depth.

Two remarks are in order. The first is that the furthermore clause of Lemma 3.36 follows by the same arguments used in Proposition 3.25: All of the computations preformed by the prover boil down to computing the encoding of polynomial-length strings (under the low degree extension encoding) and to preforming arithmetic operations over $\mathbb{F}$, and these can be preformed by an NC circuit. The only exceptions are the first two steps of the query reduction protocol, where both parties execute the original protocol $(\mathcal{P}, \mathcal{V})$, and then $\mathcal{P}_{\mathsf{Qreduce}}$ sends the encoding of $\mathcal{V}$'s view, that is, the queries and the query answers. Since we assume that $\mathcal{P}$ is low-depth, the honest $\mathcal{P}_{\mathsf{Qreduce}}$ can preform these by a μ-depth circuit.

The second remark is about the queries made in the protocol, that are input-oblivious. RRR settle for a relaxation of Definition 3.31, in which the verifier's queries to the transcript depend only on its random string when interacting with the *prescribed* prover (see [RRR16, Definition 4.7]). In this work, we consider the more natural definition, in which the condition holds also when interacting with the cheating prover. To make the query reduction protocol satisfy the stronger definition, we revisit the only step where the verifier's queries may be adaptive, that is, Step (4) of $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$. There, we add an additional step where $\mathcal{V}_{\mathsf{Qreduce}}$ checks that $\tilde{Q}_\alpha$ is the correct query set w.r.t. the randomness it has chosen for generating Q, and rejects otherwise. This promises that the query addresses does not depend on the transcript or the input. This change is possible without hurting the complexities as stated above.

4 Batch Verification for UP

In this section we construct a constant-round doubly-efficient interactive argument for batch verification of UP statements. Recall that such a theorem allows the verification of k UP witnesses in a much more efficient way than k independent verifications, and while maintaining the soundness error. As in the standard definition for doubly-efficient protocols for NP or UP, the *honest* prover gets the witnesses for the inputs' membership in $\mathcal{L}$, whereas w.l.o.g. the *cheating* prover does not get any, although there may be an input that has a witness (since the cheating prover is non-uniform, it can get the existing witnesses as auxiliary input).

Theorem 4.1 (Batch Verification for UP)**.** *Assume one-way functions exist, and let $\delta \in (0, 1)$ be a constant. Let $\mathcal{L}$ be a language in UP with inputs of length n and witnesses of length $M = M(n) = poly(n)$, whose witness relation can be computed by $\mathsf{Logspace}$-uniform circuits of fan-in 2, depth $D(n)$ and polynomial size. Let $k = k(n)$ be an ensemble of integers such that $1 \leq k \leq \mathsf{poly}(n)$. There is a constant-round doubly-efficient argument that, on input $(x_1, \ldots, x_k) \in (\{0,1\}^n)^k$, verifies that $\forall i \in [k], x_i \in \mathcal{L}$. The protocol is public-coins and has perfect completeness and constant soundness error. The protocol's complexities are:*

- *constant round complexity $O(1/\delta^3)$,*
- *communication complexity $\log k \cdot O\left(M + k \cdot n^{O(\delta)} \cdot D\right)$,*
- *verifier runtime $\log k \cdot O\left(M + k \cdot n^{O(\delta)} \cdot (n + D)\right)$.*

- *the honest prover, given witnesses* $(w_1, \ldots, w_k) \in (\{0,1\}^M)^k$ *for the inputs' membership in* $\mathcal{L}$*, runs in time* $\mathsf{poly}(n)$*,*
- *assuming the existence of one-way functions, the protocol is sound against malicious cheating provers running in time* $\mathsf{poly}(n)$*.*

Theorem 1.1 follows from taking δ to be a small enough constant multiple of the desired σ, so that the $n^{O(\delta)}$ term in the verification time and the communication complexity ends up being n^{σ}.

Remark 4.2. *Take* κ *to denote the security parameter (we always take* $\kappa = n^{\delta}$*). Since the protocol is public-coins, we can use a parallel repetition theorem for public-coin interactive arguments (see Theorem 3.21). Repeating the protocol for* κ *times in parallel will reduce the soundness error to be negligible, while increasing the* $O(\log k \cdot M)$ *term in the communication complexity and verification time by a multiplicative factor of* κ*.*

Fix finite fields $\mathbb{F}, \mathbb{H}$ (we formally set them below) and a low degree extension encoding $\mathsf{LDE} = \mathsf{LDE}_{\mathbb{F},\mathbb{H}}$. For every $i \in [k]$, we use the notation $\hat{w}_i = \mathsf{LDE}(w_i)$, i.e., the low degree extension of each witness. Its length is denoted by $|\hat{w}_i| = \hat{M}$.

We further fix the encoding $\mathcal{R}$ from Theorem 3.26 w.r.t. an arbitrary constant $\gamma \in (0,1)$ (e.g., $\gamma = 1/10$). Recall that $\mathcal{R}$ maps M-bit long strings to M'-bit long strings, where $M' \stackrel{\text{def}}{=} M/(1-\gamma)$. We define the matrix A is the following manner: the i^{th} row of A is $\mathcal{R}(w_i)$. Given any subset of rows $(\mathcal{R}(w_{i_1}), \ldots, \mathcal{R}(w_{i_d}))$, induced by a subset of indices $I = \{i_1, \ldots i_d\} \subseteq [k]$, where $|I| = d$ and $d \leq k$, we denote by A_I the following $(d \times M')$-dimensional matrix:

$$A_I = \begin{bmatrix} \text{---} \ \mathcal{R}(w_{i_1}) \ \text{---} \\ \vdots \\ \text{---} \ \mathcal{R}(w_{i_d}) \ \text{---} \end{bmatrix}. \tag{2}$$

For any matrix A_I, we define $C_I = C_I(A_I) \in \{0,1\}^{M'}$ as the XOR of A_I's rows, i.e.,

$$\forall j \in [M'], \ C_I[j] = \bigoplus_{i \in I} \mathcal{R}(w_i)[j].$$

We view this vector as a "checksum", and therefore refer to C_I with this terminology (in the overview, it was denoted by *chksum*, but here we abbreviate to C_I).

Assume that there exists a family of UOWHFs

$$\mathcal{H} : \{0,1\}^{n_{in}} \to \{0,1\}^{n_{out}}.$$

We denote by n_h the length of the description of each function. For every $i \in [k]$, we take $(\ell+1)$ to be the depth of the Merkle tree (see Construction 3.13) whose leaves are $\hat{w}_i$, w.r.t. $\mathcal{H}$. Given a sequence of ℓ functions sampled from $\mathcal{H}$, we take $h = h_1 \circ \cdots \circ h_\ell$ to be the concatenation of their descriptions.

We define the pair language $\mathcal{L}'$ w.r.t. $\mathcal{L}$ and Definition 3.18:

$$((x, y, h), w) \in \mathcal{L}' \iff (x, w) \in \mathcal{L} \wedge ((y, h), w) \in \mathcal{L}_{\mathsf{hashroot}},$$

where $(x, y, h) \in \{0,1\}^n \times \{0,1\}^{n_{out}} \times \{0,1\}^{n_h \cdot \ell}$ is the explicit input and $w \in \{0,1\}^M$ is the holographic input. Namely, $\mathcal{L}'$ is an "augmented" version of $\mathcal{L}$: a pair $((x, y, h), w)$ is in $\mathcal{L}'$ if w is the unique NP-witness of x for proving its membership in $\mathcal{L}$, *and* y is the correct hash root of $\hat{w}$.

Lastly, we define the pair language $\mathcal{L}_{\mathcal{R}}$ w.r.t. the encoding $\mathcal{R}$ (see Theorem 3.26):

$$\big((\chi_j, \theta_j)_{j\in[q]}, w\big) \in \mathcal{L}_{\mathcal{R}} \iff \forall j \in [q],\ \mathcal{R}(w)[\chi_j] = \theta_j,$$

where $(\chi_j, \theta_j)_{j\in[q]} \in \Big(\{0,1\}^{\log(M')} \times \{0,1\}\Big)^q$ is the explicit input and $w \in \{0,1\}^M$ is the holographic input. Notice that $\mathcal{L}_{\mathcal{R}}$ is computable in Logspace-uniform NC, due to Theorem 3.26 (see Proposition A.1 for the proof).

We proceed with the full description of the UP batching protocol.

The UP Batching Protocol $(\mathcal{P}, \mathcal{V})$

Prover's Input: $x_1, \ldots, x_k \in (\{0,1\}^n)^k$ and $w_1, \ldots, w_k \in (\{0,1\}^M)^k$.
Verifier's Input: $x_1, \ldots, x_k \in (\{0,1\}^n)^k$.

1. Let $\mathcal{H}$ be a UOWHF family, as per Definition 3.11. $\mathcal{V}$ samples $h_1, \ldots, h_\ell \in_R \mathcal{H}$ and sends their description h to $\mathcal{P}$.
2. $\mathcal{P}$ commits to the witnesses: it creates k Merkle trees as in Construction 3.13, using h and $(\hat{w}_i)_{i\in[k]}$ (where $\hat{w} = \mathsf{LDE}(w)$), and sends their roots $(y_i)_{i\in[k]}$ to $\mathcal{V}$.
3. $\mathcal{V}$ samples an integer $b \in_R \{0, \ldots, \lceil \log k \rceil\}$ and defines $d = \lceil k/2^b \rceil$. Then, it samples uniformly at random d independent indices $I \subseteq_R [k]$, and sends I to $\mathcal{P}$.
4. $\mathcal{P}$ creates A_I and a checksum C_I as described in Eq. (2) and sends C_I to $\mathcal{V}$.
5. $\mathcal{V}$ receives C_I, tests that it is a codeword of $\mathcal{R}$ using Lemma 3.27 and rejects otherwise.
6. $\mathcal{V}$ and $\mathcal{P}$ run d parallel and independent invocations (one per each $i \in I$) of the following, where $\mathcal{V}$ uses the same random tape for each of them:
   (a) the flat-GKR HIA of Theorem 3.23 w.r.t. $\mathcal{L}'$, with (x_i, y_i, h) as explicit input and w_i as holographic input. It outputs a claim about $\hat{w}_i$, denoted (r', v'_i).
   (b) the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$ of Lemma 3.28, with (r', v'_i) as explicit input and w_i as holographic input. It outputs q claims about $\mathcal{R}(w_i)$, denoted $(\chi_j, \theta^i_j)_{j\in[q]}$.
   (c) the flat-GKR HIA of Theorem 3.23 w.r.t. $\mathcal{L}_{\mathcal{R}}$, with $(\chi_j, \theta^i_j)_{j\in[q]}$ as explicit input and w_i as holographic input. It outputs a claim about $\hat{w}_i$, denoted (r, v_i).
7. $\mathcal{P}$ opens the commitments for $(\hat{w}_i[r])_{i\in I}$: it sends the paths $(p^i)_{i\in I}$.
8. $\mathcal{V}$ receives $(p^i)_{i\in I}$. It runs three tests and accepts only if they all pass:
   (a) **Consistency Check:** $\forall i \in I$, the claimed value for $\hat{w}_i[r]$ is consistent with v_i.
   (b) **Checksum Check:** The claimed values $(\theta^i_j)_{i\in I, j\in[q]}$ are consistent with C_I.
   (c) **Paths Check:** $\forall i \in I,\ p^i$ is a valid opening.

Remark 4.3. *When using the same random tape in parallel runs, we do not harm the completeness or soundness of the protocol, while promising that the coordinates r, r' are the same in each run. This is crucial for checking the consistency between the checksum and the prover's answers while maintaining low communication complexity.*

Since the protocol is public-coins, we can use a parallel repetition theorem for public-coin interactive arguments (see Theorem 3.21). The full protocol runs in parallel $\gamma = O(\log k)$ copies of the protocol described above, and accepts only if all of the copies accept. Looking ahead, we prove that the soundness error is $s = 1 - 1/\Theta(\log k)$, hence taking γ repetitions will reduce the overall soundness error to be as small as any constant.

Parameter Setting. Let $n \in \mathbb{N}$ and $\delta \in (0,1)$. Take $\mathbb{H} \subseteq \mathbb{F}$ to be (ensembles of) extension fields of $\mathbb{GF}[2]$, such that $|\mathbb{H}| = \Theta(n^{\delta})$ and $|\mathbb{F}| = \Theta(n^{2\delta})$. We define

$$n_{in} = n^{2\delta}, \; n_{out} = n^{\delta},$$

and a security parameter $\kappa = n_{out}$ (in Definition 3.11, we take k to be the smallest integer such that $n_{1_k} = n_{in}$ and $n_{0_k} = n_{out}$). The UOWHF family we use is secure against polynomial time adversaries, and the time it takes to compute each function is $\mathsf{poly}(n_{in}) = n^{O(\delta)}$. Note that if OWFs exist, then there exist UOWHFs satisfying these properties.

Once the fields are set, the length of $|\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w)| = |\hat{w}| = \hat{M}$ for $|w| = M$ is set: $\hat{M} = M^2$. Given M, we can find ℓ (the depth of the UOWHF tree whose leaves are $\hat{w}$): it is the unique solution to the equation

$$\left(\frac{n_{out}}{n_{in}}\right)^{\ell-1} \cdot \hat{M} = n_{out},$$

which implies that $(n^{\delta})^{\ell} = \hat{M}$, thus $\ell = \log_{n^{\delta}}(\hat{M})$. This implies that $\ell = O(1/\delta)$ for all $M = \mathsf{poly}(n)$. In fact, due to Remark 3.17, the length of the leaves is $\left(\log|\mathbb{F}| \cdot M^2\right)$, but this is also $\mathsf{poly}(n)$, so the above analysis still applies.

Proof of Theorem 4.1.

Completeness. Let $(\mathcal{P}, \mathcal{V})$ denote a single run of the protocol (before performing the repetitions). Assume that there exist witnesses $w_1, \ldots, w_k$ such that $\forall i \in [k], x_i \in \mathcal{L}$.

First, notice that if $(\mathcal{R}(w_i))_{i\in[k]}$ are all legal codewords, then the checksum C_I, which is the XOR (which is a sum) of a subset of them, is also a legal codeword, thanks to the linearity of the encoding $\mathcal{R}$ (see Theorem 3.26). This means that an honest $\mathcal{P}$ sends a checksum C_I that passes the test in Step (5). Moreover, it is evident that an honest prover passes the three final checks if it follows the steps of the protocol, by the perfect completeness of the $\mathsf{flat\text{-}GKR\ HIA}$ (see Theorem 3.23) and the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$ (see Lemma 3.28). Thus, the verifier accepts in all of the parallel executions with probability 1.

Soundness. First, let us introduce some useful notation for the two lasts steps of the protocol:

- In Step (7), let $q = \left\lceil \frac{r}{n_{in}} \right\rceil$ be the leaf index (in the UOWHF tree) that $\mathcal{P}^*$ computes in order to open the commitments for the r^{th} coordinate of $(\hat{w}_i)_{i\in I}$.

- For every row $i \in I$, let $p^i = \left(p^i_1, \ldots, p^i_{\ell+1}\right)$ be the tree path that corresponds to q, namely, the opening for $\hat{w}_i[r]$.
- Let $\xi_i = p^i_{\ell+1}[q]$ (by the previous item, $p^i_{\ell+1}$ is the last node in the path that the prover sends for opening the r^{th} coordinate of $\hat{w}_i$). If the prover is honest, $\xi_i = \hat{w}_i[r]$.

Using these notation, we rephrase $\mathcal{V}$'s tests in Step (8). It receives $(p^i)_{i\in I}$, and runs three tests:

1. **Consistency Check:** $\forall i \in I,\ \xi_i = v_i$.
2. **Checksum Check:** $\forall j \in [q]$, the χ_j^{th} coordinates of C_I indeed equals the parity of the values claimed for the χ_j^{th} column of A_I, that is: $C_I[\chi_j] = \bigoplus_{i\in I} \theta^i_j$.
3. **Paths Check:** $\forall i \in I,\ p^i$ is a valid path w.r.t. h, y_i and q.

With these in mind, we turn to the proof. For the soundness condition of the full protocol, fix $x_1, \ldots, x_k$ and assume $\exists i \in [k]$ such that $\mathcal{L}(x_i) = \emptyset$, i.e., x_i is not in the language and thus has no NP-witness. Suppose that there exists a polynomial time strategy $\mathcal{P}^*$ such that after interacting with it, the probability that $\mathcal{V}$ accepts is s, and let A denote this event.

Take B to be the following event, where we use the terms "hash root", "root" and "commitment" interchangeably:

$$\Pr[B] = \Pr\left[\exists! i^* \in I \text{ s.t. } \left(\mathcal{L}(x_{i^*}) = \emptyset\right) \vee \left(y_{i^*} \text{ is not the correct hash root of } \hat{w}_{i^*}\right)\right].$$

We call such an index a *bad* index, which means that B is the event that there exists a single bad index in I. We take $I' = I \setminus \{i^*\}$ to be the set of indices that are not *bad*, assuming B. If B does not happen, we trivially define I' to be an empty set.

Moreover, we define the following event E: the prover's answers in the Code-Switching executions to inputs in I' are consistent with the true correct witnesses. That is, the event E implies that $\forall i \in I', j \in [q],\ \theta^i_j = \mathcal{R}(w_i)[\chi_j]$. The following claim shows that E holds with high probability. Recall that $negl(n_{out}) = negl(n)$.

Claim 4.4. $\Pr\left[A \wedge B \wedge \neg E\right] \leq negl(n) + 1/n$.

Proof. First, recall that $\mathcal{L}$ is a UP language. This means that for each $i \in I'$ there exists a unique NP-witness w_i for x_i, and thus $\hat{w}_i = \mathsf{LDE}(w_i)$ is also uniquely defined. Moreover, assuming B, for any $i \in I'$ it holds that y_i is the correct root of $\hat{w}_i$. Getting back to Definition 3.15 and recalling that $\mathcal{P}^*$ passes the paths check of Step (8c), we get that if $\mathcal{P}^*$ implicitly declares the string $\hat{w}_i$ (which is defined *before* the hash functions are chosen) and sends the correct hash root y_i as the commitment to $\hat{w}_i$, then, for the index q and the path p^i:

$$\begin{aligned}
&\Pr[p^i \text{ is a } \textit{valid path from } q \textit{ to } y_i \text{ and } \xi_i \neq \hat{w}_i[r]] \leq \\
&\Pr[p^i \text{ is a } \textit{valid path from } q \textit{ to } y_i \text{ and} \\
&\qquad p^i_{\ell+1} \neq \hat{w}_i[(q-1)n_{in}] \ldots \hat{w}_i[q \cdot n_{in} - 1]] = negl(n),
\end{aligned}$$

by the targeted-collision-resistance of the commitment scheme (see Proposition 3.16). The probability is over the choice of the functions, since w.l.o.g. the prover is deterministic (as it is non-uniform). We comment that in the current setting, $\mathcal{P}^*$ receives which index to open (that is, q), although

Proposition 3.16 achieves something stronger: even if $\mathcal{P}^*$ chooses the index by itself, it cannot find a collision with probability better than $negl(n)$.

Next, assume that $\forall i \in I', \xi_i = \hat{w}_i[r]$. Since $\mathcal{P}^*$ passes the test of Step (8a), we get that $\forall i \in I', v_i = \hat{w}_i[r]$. Namely, $\forall i \in I'$, $\mathcal{P}^*$ answers honestly to the $\mathsf{flat\text{-}GKR}$ executions preformed in Step (6c), for any coordinate r chosen at random by the verifier during the execution. This implies that, unless $\mathcal{P}^*$ breaks the soundness guarantee of $\mathsf{flat\text{-}GKR}$, it outputs $(\chi_j, \theta_j^i)_{j\in[q]}$ that are all correct claims about $\mathcal{R}(w_i)$, that is, $\forall i \in I', j \in [q]$, $\theta_j^i = \mathcal{R}(w_i)[\chi_j]$.

The proposition follows by recalling that $k \leq \mathsf{poly}(n)$, and a union bound over all $i \in I'$ while taking $p(n) = k \cdot n$ in Theorem 3.23. □

Next, assuming that B and E hold, we prove that the prover's answers to the $i^{*\text{th}}$ execution of the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$ are determined non-adaptively according to a fixed string before the protocol begins. Namely, we show that for any random coordinate χ, the value θ^{i^*} of the output claim (χ, θ^{i^*}) is fixed in advance.

Proposition 4.5. *Take $\pi = C_I \bigoplus_{i\in I'} \mathcal{R}(w_i)$, and assume that the event $(A \wedge B \wedge E)$ holds. Then, with all but a $1/n$ probability, $\pi = \mathcal{R}(w')$ for some string $w' \in \{0,1\}^M$, and $\mathcal{P}^*$ uses w' as holographic input in the i^{*th} execution of the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$.*

Proof. Recall that, assuming B, the set I' is the set of indices that are not *bad*: $I' = I \setminus \{i^*\}$. Since $\mathcal{P}^*$ passes the checksum check in Step (8b), as we assume that A happens (namely, that the verifier does not reject), we get for any $\chi \in \{0,1\}^{\log(M')}$:

$$C_I[\chi] = \bigoplus_{i\in I} \theta^i = \left(\bigoplus_{i\in I'} \theta^i\right) \oplus \theta^{i^*} = \left(\bigoplus_{i\in I'} \mathcal{R}(w_i)[\chi]\right) \oplus \theta^{i^*},$$

from the assumption that E holds. Then:

$$\theta^{i^*} = \left(\bigoplus_{i\in I'} \mathcal{R}(w_i)[\chi]\right) \oplus C_I[\chi].$$

We define

$$\pi = \left(\bigoplus_{i\in I'} \mathcal{R}(w_i)\right) \oplus C_I.$$

Since $\mathcal{P}^*$ passes the test of Step (5), C_I is a codeword of $\mathcal{R}$ with all but a $1/n$ probability (here, we took $\varepsilon = 1/n$ in Lemma 3.27). Since all of $(\mathcal{R}(w_i))_{i\in I'}$ are codewords and the code is linear (see Theorem 3.26), we get that π is also a codeword. Namely, there exists a string $w' \in \{0,1\}^M$ such that $\pi = \mathcal{R}(w')$. This means that $\mathcal{P}^*$ uses a fixed string w' as holographic input in the $i^{*\text{th}}$ execution of the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$. □

Our next goal is to show that (r', v'_{i^*}) is a false claim w.r.t. w', unless the prover breaks the soundness property of $\mathsf{flat\text{-}GKR}$ in the $i^{*\text{th}}$ execution preformed in Step (6a). We stress that this step's purpose is only to catch a cheating on i^*; these $\mathsf{flat\text{-}GKR}$ executions for $i \in I'$ do not affect the protocol's soundness, and that is why we do not ask from the prover to open any of the commitments on r'. Recall that B implies that either $\mathcal{L}(x_{i^*}) = \emptyset$, or that y_{i^*} is not the correct root of $\hat{w}_{i^*}$, and we split into cases.

- If $\mathcal{L}(x_{i^*}) = \emptyset$, it is clear that w' is not a correct witness for x_{i^*} (because there is no such witness). Then, (r, v_{i^*}) is a false claim w.r.t. w', unless $\mathcal{P}^*$ breaks the soundness property of $\mathsf{flat\text{-}GKR}$ in Step (6a).

- If $\mathcal{L}(x_{i^*}) \neq \emptyset$, then, by the assumption, y_{i^*} is not the correct root of $\hat{w}_{i^*}$. Since $\mathcal{L}(x_{i^*}) \neq \emptyset$, the string w_{i^*} is well-defined as the unique NP-witness for x_{i^*}.

  - On the one hand, if $w' = w_{i^*}$, then y_{i^*} is not the correct root for w', which means that $((x_{i^*}, y_{i^*}, h), w') \notin \mathcal{L}'$. Then, (r, v_{i^*}) is a false claim w.r.t. w', unless $\mathcal{P}^*$ breaks the soundness property of $\mathsf{flat\text{-}GKR}$.
  - On the other hand, if $w' \neq w_{i^*}$, then w' is not the unique correct witness for x_{i^*}, which leads us back to the first case.

Altogether, we get that

$$\Pr\left[\mathsf{LDE}(w')[r'] \neq v'_{i^*}\right] \geq 1 - 1/n,$$

according to Theorem 3.23.

Proposition 4.5 allows us to use the soundness guarantee of the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$ w.r.t. the explicit input (r', v'_i) and the holographic input w': if (r', v'_i) is a false claim about $\hat{w}'$, then at least one of the claims $(\chi_j, \theta_j^{i^*})_{j\in[q]}$ is false, unless $\mathcal{P}^*$ breaks the soundness property of the $\mathsf{HIP}_{\mathcal{R}}$.

Assuming this event does not hold, take j^* to denote the false claim, namely, $\mathcal{R}(w')[\chi_{j^*}] \neq \theta_{j^*}^{i^*}$. Recalling that in Proposition 4.5 we showed that $\theta_{j^*}^{i^*} = \pi[\chi_{j^*}] = \mathcal{R}(w')[\chi_{j^*}]$, we reach a contradiction. This implies that $\Pr(A \wedge B \wedge E)$ is bounded by the sum of soundness errors of the $\mathsf{flat\text{-}GKR}$ HIA, of the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$, and of the local-testing procedure. Each of these is at most $1/n$.

Finally, using Proposition 3.1 for finding $\Pr[\neg B]$ (where we take T to be the actual set of *bad* indices in $[k]$, determined by $\mathcal{P}^*$'s strategy), we conclude that:

$$\begin{aligned} s = \Pr[A] &\leq \Pr[A \wedge B \wedge E] + \Pr[A \wedge B \wedge \neg E] + \Pr[\neg B] \\ &\leq 4/n + negl(n) + 1 - 1/30 \log k. \end{aligned}$$

Recalling that $k \leq \mathsf{poly}(n)$, we get that $s \leq 1 - 1/40 \log k$, since $negl(n)$ and $1/n$ are smaller than $(1/1000 \log k)$. Getting back to the full protocol, using Remark 3.21 with $\gamma = O(\log k)$ repetitions implies that the overall soundness error is at most

$$(1 - 1/40 \log k)^{O(\log k)} + negl(n) \leq 1/e + negl(n) \leq 1/2.$$

Complexity. Let $\delta \in (0, 1)$ and consider $\mathbb{H}, \mathbb{F}$ as defined in Section 4. By Proposition 3.19, $\mathcal{L}_{\mathsf{hashroot}}$ is computable by a $\mathsf{Logspace}$-uniform polynomial size circuit of depth $O(1/\delta) \cdot \mathsf{poly}(n_{in}) = n^{O(\delta)}$. Thus, the language $\mathcal{L}'$ is computable in depth $\max(n^{O(\delta)}, D(n))$ and polynomial size, by a circuit that outputs the conjunction of the circuit computing the hash root and the circuit computing $\mathcal{L}$.

Notice that $\mathcal{L}_{\mathcal{R}}$ is computable by an NC circuit, and that the length of the explicit input to the executions in Step (6a) is longer than to the ones in Step (6c). Thus, we only consider the executions of Step (6a).

The number of rounds, dominated by the round complexity of $\mathsf{flat\text{-}GKR}$, is $O(1/\delta^3)$. The communication complexity of the protocol is dominated by three steps: sending the checksum C_I, running

the $d \leq k$ invocations of the flat-GKR protocol (in Step (6a)) and running the Code-Switching protocol. Since the communication complexity of each of the latter is $O\left(\mathsf{polylog}(n) \cdot n^{O(\delta)}\right)$, and of flat-GKR is $\max(n^{O(\delta)}, D(n)) \cdot n^{O(\delta)} = D(n) \cdot n^{O(\delta)}$, this overall yields

$$\mathsf{cc} = O(M) + k \cdot D(n) \cdot n^{O(\delta)}.$$

As for the verification time, first, notice that we invoked Lemma 3.27 with a constant proximity parameter (in particular, the code's relative distance is $\left(\frac{\gamma}{O(1/\delta)}\right)^{O(1/\delta)}$, so any proximity parameter α smaller than that would work). This means that the query complexity of the tester is $O\left(n^{2\delta} \cdot \log n\right)$, since we took the code to have constant rate $1-\gamma$ and set $\varepsilon = 1/n$. Moreover, the verification time of the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$ is $n^{O(\delta)}$. Hence, the "heaviest" computation run by the verifier is executing the $d \leq k$ invocations of flat-GKR, which takes $k \cdot (D(n)+n) \cdot n^{O(\delta)}$ time. The prover runs in time $\mathsf{poly}(M,n) = \mathsf{poly}(n)$ given the witnesses $(w_1, \ldots, w_k)$, and the complexities of the full protocol follow by recalling that we preform $O(\log k)$ parallel repetitions. □

5 Interactive Arguments for Bounded-Space Computations

We denote by $\mathsf{DTISP}(T,S)$ the class of all languages accepted by a Turing machine in time $T = T(n)$ and space $S = S(n)$. Our main result is a new construction of *doubly-efficient* and *constant-round* interactive arguments for languages that are computable by bounded-space Turing machines:

Theorem 5.1 (Interactive Arguments for Bounded-Space)**.** *Assume one-way functions exist. Let $T = T(n)$ and $S = S(n)$ such that $n \leq T \leq \mathsf{exp}(n)$ and $\log(T) \leq S \leq \mathsf{poly}(n)$.*

Let $\mathcal{L} \in \mathsf{DTISP}(T,S)$ and let $\delta \in (0,1)$. Then, $\mathcal{L}$ has a public-coin interactive argument with perfect completeness and negligible soundness error. The number of rounds is $O(1/\delta^4)$. The communication complexity is $O\left(T^{O(\delta)} \cdot S^2\right)$. The prover runs in time $\mathsf{poly}(T,S)$ time, and the verifier runs in time
$O\left(n^{1+O(\delta)} + T^{O(\delta)} \cdot S^2\right)$.

If the verifier is given query access to a low-degree extension of the input, then its running time is reduced to $O\left(T^{O(\delta)} \cdot S^2\right)$.

Theorem 1.2 is derived by plugging in $T = \mathsf{poly}(n)$, and taking δ to be a small enough constant multiple of the desired σ, so that the $n^{O(\delta)}$ term in the verification time and the communication complexity ends up being n^{σ}.

We comment that our focus is on running time as the critical resource, as in past work (e.g. the [RRR16] protocol). We do not attempt to bound the verifier's space, and it might grow quadratically. Bounding the verifier's space usage is a fascinating question; linear space is a natural goal, but one could also ask whether it is possible to obtain logarithmic space.

Organization of this section. In Section 5.1, we prove the main new technical tool of this work, *batch verification* of PCIAs. This protocol takes a PCIA for the language $\mathcal{L}_t^{\mathcal{M}}$ (see Definition 3.34) and generates an efficient PCIA for checking membership of multiple inputs in $\mathcal{L}_t^{\mathcal{M}}$, albeit with a high query complexity. We use it in Section 5.2 in order to achieve a batch verification protocol of PCIAs for $\mathcal{L}_t^{\mathcal{M}}$ that also maintains low query complexity, called the *augmentation protocol*. Then, in Section 5.3, we use the augmentation protocol to prove Theorem 5.11, which is a PCIA for any bounded-space computation. Finally, we derive Theorem 5.1 by transforming the PCIA to an IA. Appendix B contains technical proofs deferred from this section.

5.1 Batch Verification of PCIAs

In this section we show how to batch verify PCIAs for computations of length $t = t(n)$, which is the main component in our interactive arguments for bounded-space computations (see Section 2.3 for an overview of the construction).

The goal of the batch verification protocol is to transform a PCIA for $\mathcal{L}_t^{\mathcal{M}}$ (see Definition 3.34), denoted $(\mathcal{P}_t, \mathcal{V}_t)$, into an efficient PCIA for $\mathcal{L}_{k \cdot t}^{\mathcal{M}}$. In fact, we will require two properties from the PCIA for $\mathcal{L}_t^{\mathcal{M}}$: that the prover is encoded (see Definition 3.32), and that the *honest* prover is μ-low-depth (see Definition 3.35), where we set μ to be as small as the security parameter (that is, n^{δ}, see next). As mentioned in the technical overview, we could have constructed an abstracted version of a more general batch verification protocol (in particular, for PCIAs with a low-depth prover). However, since it leads to a large technical overhead that makes the protocol less clear, we decided to present the restricted case of batch verification of a PCIA for the language $\mathcal{L}_t^{\mathcal{M}}$.

Parameter Setting for Section 5. Let $n \in \mathbb{N}$ and $\delta \in (0,1)$. Take $\mathbb{H} \subseteq \mathbb{F}$ to be (ensembles of) extension fields of $\mathbb{GF}[2]$, such that $|\mathbb{H}| = \Theta(n^{\delta})$ and $|\mathbb{F}| = \Theta(n^{2\delta} \cdot \log n)$.

As done in the UP batching protocol (see Section 4), we use a UOWHF family $\mathcal{H} : \{0,1\}^{n_{in}} \to \{0,1\}^{n_{out}}$, and set the security parameter to be $\kappa = n_{out}$, where we define

$$n_{in} = n^{2\delta}, \ n_{out} = n^{\delta}.$$

Thus, the UOWHF family we use is secure against polynomial time adversaries, and the time it takes to compute each function is $\mathsf{poly}(n_{in}) = n^{O(\delta)}$. Moreover, as shown in Section 4, the depth of the UOWHF tree in Construction 3.13 is $\ell = O(1/\delta)$ whenever we hash (i.e., use as leaves) strings of polynomial length, which is always the case. Note that if OWFs exist, then there exist UOWHFs satisfying these properties.

Before stating the batch verification lemma, we briefly go over its parameters: k is the number of protocols to be batched, δ is a parameter controlling the complexities of the protocol, and ε, q, r, a, b, $\mathcal{P}\mathsf{time}$, $\mathcal{V}\mathsf{time}$ are, respectively, the soundness error, the query complexity, the number of rounds, the length of prover messages, the length of verifier messages, the running time of the (honest) prover and the running time of the verifier.

Lemma 5.2 (Batch Verification for PCIAs w.r.t. encoded provers)**.** *Assume one-way functions exist, and let $\delta \in (0,1)$ be a constant. Let $\mathbb{H} \subseteq \mathbb{F}$ be (ensembles of) extension fields of $\mathbb{GF}[2]$, where $|\mathbb{H}| = \Theta(n^{\delta})$ and $|\mathbb{F}| = \Theta(n^{2\delta} \cdot \log n)$.*

Let $\mathcal{M}$ be a Turing machine that uses space at most $S = S(n)$, and let $t = t(n)$ be a time bound. Suppose that $\mathcal{L}_t^{\mathcal{M}}$ has an ε-sound $(q, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIA $(\mathcal{P}_t, \mathcal{V}_t)$ w.r.t. encoded provers and with input-oblivious queries. Let $k = k(n) \leq \mathsf{poly}(n)$ and assume that

- $q \leq \min\left(\mathcal{V}\mathsf{time}, k \cdot n^{O(\delta)}\right)$;
- $a \geq \max\left((k \cdot S)^2 \cdot \mathsf{polylog}(n), k^2 \cdot n^{O(\delta)}, (k \cdot q)^2 \cdot \mathsf{polylog}(n)\right)$;
- $r \cdot b \leq k \cdot n^{O(\delta)}$.

Then, there exists an ε_{B}-sound $(q_{\mathsf{B}}, r_{\mathsf{B}}, a_{\mathsf{B}}, b_{\mathsf{B}}, \mathcal{P}\mathsf{time}_{\mathsf{B}}, \mathcal{V}\mathsf{time}_{\mathsf{B}})$-PCIA $(\mathcal{P}_{\mathsf{batch}}, \mathcal{V}_{\mathsf{batch}})$ for the language $\mathcal{L}_{k \cdot t}$ w.r.t. encoded provers and with input-oblivious queries, with the following parameters:

- $\varepsilon_{\mathsf{B}} = 1 - (1 - \varepsilon)/30 \log k + O(1/\log n)$.

- $q_{\mathsf{B}} = (2k+1) \cdot q + n^{O(\delta)}$.
- $r_{\mathsf{B}} = r + O(1/\delta^3)$.
- $a_{\mathsf{B}} = a$.
- $b_{\mathsf{B}} = k \cdot n^{O(\delta)}$.
- $\mathcal{P}\mathsf{time}_{\mathsf{B}} = \mathsf{poly}(k, n^{\delta}, t, S) + k \cdot \mathcal{P}\mathsf{time}$.
- $\mathcal{V}\mathsf{time}_{\mathsf{B}} = \mathcal{V}\mathsf{time} \cdot \mathsf{poly}(k, n^{\delta})$.

Furthermore, if $\mathcal{P}_t$ is n^{δ}-low-depth, then $\mathcal{P}_{\mathsf{batch}}$ is n^{δ}-low-depth as well.

Most importantly, note that the length of $\mathcal{P}_{\mathsf{batch}}$'s messages does not grow, rather than the trivial $a \cdot k$ (which can be obtained by simply running the k protocols), and that the round complexity only grows additively by a constant factor.

As discussed in the overview, the high-level idea for designing the PCIA $(\mathcal{P}_{\mathsf{batch}}, \mathcal{V}_{\mathsf{batch}})$ is for $\mathcal{P}_{\mathsf{batch}}$ to first specify k evenly-spaced intermediate configurations of the Turing machine. Given these intermediate configurations, $\mathcal{V}_{\mathsf{batch}}$ wants to verify that k statements are in $\mathcal{L}_t^{\mathcal{M}}$, where the i^{th} statement refers to a computation of length t between two Turing machine configurations.

Let $\mathcal{M}$ be a Turing machine that uses space at most $S = S(n)$. For convenience, we use the notation $\mathcal{L}_t \stackrel{\text{def}}{=} \mathcal{L}_t^{\mathcal{M}}$ and $\mathcal{L}_{k \cdot t} \stackrel{\text{def}}{=} \mathcal{L}_{k \cdot t}^{\mathcal{M}}$. Assume that $\mathcal{L}_t$ has a $(q, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIA $(\mathcal{P}_t, \mathcal{V}_t)$ w.r.t. encoded provers, where the honest $\mathcal{P}_t$ is n^{δ}-low-depth, and with input-oblivious queries, where the parameters satisfy the requirements in the lemma statement.

We define the pair language $\mathcal{L}_{\mathcal{P}_t}$ with respect to the prescribed prover strategy $\mathcal{P}_t$, where Q is a query set of size q:

$$\big((Q, \phi, \beta_1, ..., \beta_r), (x, \mathcal{T}) \big) \in \mathcal{L}_{\mathcal{P}_t} \iff \begin{array}{l} \phi \text{ is consistent with the prescribed} \\ \text{strategy w.r.t. } Q, \beta_1, ..., \beta_r, \end{array}$$

where $(Q, \phi, \beta_1, ..., \beta_r) \in (\{0,1\}^{\log r + \log a})^q \times \mathbb{F}^q \times \{0,1\}^{r \cdot b}$ is the explicit input to the language $\mathcal{L}_{\mathcal{P}_t}$ and $(x, \mathcal{T}) \in \{0,1\}^n \times \{0,1\}^{t \cdot O(S)}$ is the implicit input. Notice that the representation of queries Q is actually the binary representation of two indices: each query $(j, t) \in [r] \times [a]$, namely, the t^{th} entry in the prover's messages in the j^{th} round, is represented by a binary string of length $(\log r + \log a)$.

Remark 5.3. *In what follows, we use the* flat-GKR *protocol of Theorem 3.23 in the context of the* PCIA $(\mathcal{P}_{\mathsf{batch}}, \mathcal{V}_{\mathsf{batch}})$*, by having the* PCIA *prover and verifier run these as a sub-protocol. When this is the case, we want to bound the verifier $\mathcal{V}_{\mathsf{batch}}$'s query complexity in the* flat-GKR *execution. We do so by simply having the* PCIA *verifier $\mathcal{V}_{\mathsf{batch}}$ explicitly query every bit of the messages sent by the prover of the* flat-GKR *protocol. This gives query complexity that is at most the communication complexity (that trivially bounds the overall length of the prover messages). Since the verifier reads every bit of every message, we get that all queries are input-oblivious.*

The same is applicable to the other sub-protocol that is used in $(\mathcal{P}_{\mathsf{batch}}, \mathcal{V}_{\mathsf{batch}})$, that is, the interactive reduction of Proposition 3.10, preformed in Step (6b).

Remark 5.4 (Padding)**.** *We always assume that the prover's messages in the protocol are padded to length* a_{B}*. Unlike in [RRR16], the verifier does not need to check the correctness of the padding (i.e., that the prover indeed padded with zeros, and not with other field elements) because this kind of deviation from the prescribed strategy will not hurt the soundness of the protocol.*

We proceed with the full description of protocol. Notice that, as commented in Remark 4.3, when running the flat-GKR protocol in Steps (6a) and (10a) the verifier uses the same random tape. Thus, the coordinates in the claims that the protocol outputs (these are r^{root} and r') are the same in all parallel runs, because they only depend on the verifier's randomness, and this does harm the completeness or soundness of the protocol.

Lastly, we comment that we use the term "checksum" as in the overview (there it was denoted *chksum*) and in Section 4, see Eq. (2). However, here, instead of parity, we use a sum where the addition is over the field $\mathbb{F}$.

PCIA $(\mathcal{P}_{\mathsf{batch}}, \mathcal{V}_{\mathsf{batch}})$ **for** $\mathcal{L}_{k\cdot t}$

Prover's Input: $x \in \{0,1\}^n$, configurations $u, v \in \{0,1\}^{O(S)}$ and a claim (ω, θ).
Verifier's Input: explicit access to (ω, θ) and implicit access to $\mathsf{LDE}(x, u, v)$.

1. $\mathcal{P}_{\mathsf{batch}}$ runs $\mathcal{M}$ starting at u for $k \cdot t$ steps. Let w_i be the configuration of $\mathcal{M}$ after i steps, $\forall i \in \{0, t, 2t, \ldots, k\cdot t\}$, and let $\mathcal{T}_i \in \{0,1\}^{t\cdot O(S)}$ be the sequence of t configurations from $w_{(i-1)\cdot t}$ to $w_{i\cdot t}$, for every $i \in [k]$. $\mathcal{P}_{\mathsf{batch}}$ sends $\mathsf{LDE}(w_t, w_{2t}, \ldots, w_{(k-1)\cdot t})$ to $\mathcal{V}_{\mathsf{batch}}$.
2. $\mathcal{P}_{\mathsf{batch}}$ and $\mathcal{V}_{\mathsf{batch}}$ decompose the claim (ω, θ) given as input into k claims $(\omega_i, \theta_i)_{i\in[k]}$ about $(\mathsf{LDE}(\mathcal{T}_i))_{i\in[k]}$, using the interactive process of Remark 3.8.
3. $\mathcal{V}_{\mathsf{batch}}$ receives[a] $\mathsf{LDE}(\tilde{w}_t, \ldots, \tilde{w}_{(k-1)\cdot t})$. Let $\tilde{w}_0 \stackrel{\text{def}}{=} u$ and $\tilde{w}_{k\cdot t} \stackrel{\text{def}}{=} v$.

   Let $\mathcal{H}$ be a UOWHF family, as per Definition 3.11. $\mathcal{V}_{\mathsf{batch}}$ samples $h_1, \ldots, h_\ell \in_R \mathcal{H}$ and sends their description h to $\mathcal{P}_{\mathsf{batch}}$.
4. $\mathcal{P}_{\mathsf{batch}}$ creates k Merkle trees as in Construction 3.13, using h and $\mathcal{T}_1, \ldots, \mathcal{T}_k$, and sends their roots $y_1, \ldots, y_k$ to $\mathcal{V}_{\mathsf{batch}}$.
5. $\mathcal{V}_{\mathsf{batch}}$ samples an integer $b \in_R \{0, ..., \lceil \log k \rceil\}$ and defines $d = \lceil k/2^b \rceil$. Then, it samples d indices $I \subseteq_R [k]$, and sends I to $\mathcal{P}_{\mathsf{batch}}$.
6. $\mathcal{P}_{\mathsf{batch}}$ and $\mathcal{V}_{\mathsf{batch}}$ run in parallel $\forall i \in I$:
   (a) the flat-GKR protocol of Theorem 3.23 w.r.t. $\mathcal{L}_{\mathsf{hashroot}}$ (see Definition 3.18), with (y_i, h) as explicit input and $\mathcal{T}_i$ as holographic input. The protocol ends with a claim $(r^{\text{root}}, v_i^{\text{root}})$ about $\mathsf{LDE}(\mathcal{T}_i)$.
   (b) the interactive process of reducing the claims $(r^{\text{root}}, v_i^{\text{root}})$ and (ω_i, θ_i) to a single one (Proposition 3.10), denoted $(\omega_i^{\mathcal{T}}, \theta_i^{\mathcal{T}})$.
   (c) the PCIA for $\mathcal{L}_t$ with inputs $\big((x, \tilde{w}_{(i-1)\cdot t}, \tilde{w}_{i\cdot t}), (\omega_i^{\mathcal{T}}, \theta_i^{\mathcal{T}})\big)$. For $j = 1, \ldots, r$:
      i. $\mathcal{V}_{\mathsf{batch}}$ sends randomness β^j (the same random coins are used for all d inputs).
      ii. $\mathcal{P}_{\mathsf{batch}}$ computes the messages $(\alpha_i^j)_{i\in I}$ that $\mathcal{P}_t$ would answer. It creates a matrix A_I^j and sends its checksum $C_I^j = \sum_{i\in I} \alpha_i^j$, as defined in Eq. (2).

7. $\mathcal{V}_{\mathsf{batch}}$ generates a query set $Q \subseteq [r] \times [a]$ for $\mathcal{V}_t$'s query phase.

8. $\mathcal{P}_{\mathsf{batch}}$ sends answers $(\phi_i : Q \to \mathbb{F})_{i \in I}$ such that $\forall (j,t) \in Q,\ \phi_i(j,t) = \alpha_i^j[t]$.

9. $\mathcal{V}_{\mathsf{batch}}$ checks that $\forall (j,t) \in Q,\ \sum_{i \in I} \phi_i(j,t) = C_I^j[t]$. Then, it checks that the answers $(\phi_i)_{i \in I}$ make $\mathcal{V}_t$ accept. If any of these tests fail, $\mathcal{V}_{\mathsf{batch}}$ rejects.

10. $\mathcal{P}_{\mathsf{batch}}$ and $\mathcal{V}_{\mathsf{batch}}$ run in parallel $\forall i \in I$:

    (a) The flat-GKR protocol of Theorem 3.23 w.r.t. $\mathcal{L}_{\mathcal{P}_t}$, with $(Q, \phi_i, \beta_1, \ldots, \beta_r)$ as explicit input and $(x, \mathcal{T}_i)$ as holographic input. It ends with a claim (r', v_i') about $\mathsf{LDE}(x, \mathcal{T}_i)$. Then, the verifier queries $\mathsf{LDE}(x)$ using Proposition 3.7 at a single coordinate, and finds (r, v_i), the residual claim about $\mathsf{LDE}(\mathcal{T}_i)$ w.r.t. (r', v_i').[b]

    (b) $\mathcal{P}_{\mathsf{batch}}$ opens the commitments at coordinate r.

    (c) $\mathcal{V}_{\mathsf{batch}}$ checks that the opening is valid and consistent with v_i, and rejects otherwise.

[a] Recall that we restrict our attention to encoded provers and so may assume that the received message *is* a low degree extension encoding of some (possibly incorrect) configurations.

[b] Given (r', v_i'), the verifier uses Remark 3.8 to compute τ_1, τ_2 and r for which $v_i' = \tau_1 \cdot \mathsf{LDE}(x)[r] + \tau_2 \cdot \mathsf{LDE}(\mathcal{T}_i)[r]$. Using $\mathsf{LDE}(x)[r]$ and v_i', it finds the residual value $\mathsf{LDE}(\mathcal{T}_i)[r]$, and defines it as v_i.

Proof of Lemma 5.2.

Completeness. Let $x \in \{0,1\}^n$, $u, v \in \{0,1\}^{O(S)}$ and (ω, θ). First, note that the prescribed prover sends $\mathsf{LDE}(\tilde{w}_t, \ldots, \tilde{w}_{(k-1)\cdot t})$ such that $\tilde{w}_{i \cdot t} = w_{i \cdot t}$ for every $i \in [k-1]$, where $w_{i \cdot t}$ denotes the configuration of the Turing machine $\mathcal{M}$ after $i \cdot t$ steps (on input x and when starting at configuration u). We denote $u = w_0 = \tilde{w}_0$ and $v = \tilde{w}_{k \cdot t}$. By construction and the perfect completeness of Remark 3.8, Theorem 3.23, and Proposition 3.10, it holds that $\forall i \in [k],\ \left(x, \tilde{w}_{(i-1)\cdot t}, \tilde{w}_{i \cdot t}, (\omega_i^{\mathcal{T}}, \theta_i^{\mathcal{T}})\right) \in \mathcal{L}_t$. Thus, the verifier $\mathcal{V}_t$ accepts in all of its executions by the perfect completeness of $(\mathcal{P}_t, \mathcal{V}_t)$.

To justify the usage of the flat-GKR protocol in Step (10a), we use the next claim, that shows that the language $\mathcal{L}_{\mathcal{P}_t}$ satisfies the required properties of Theorem 3.23 and, moreover, is computable by $n^{O(\delta)}$-depth circuits. The proof is deferred to Appendix B.2.

Proposition 5.5. *Let $\mathbb{F}$ be an extension field of $\mathbb{GF}[2]$. Assume that the prescribed prover strategy $\mathcal{P}_t$ of the base protocol $(\mathcal{P}_t, \mathcal{V}_t)$ that $\mathcal{P}_{\mathsf{batch}}$ runs in Step (6c) is n^δ-low-depth. Then, the language $\mathcal{L}_{\mathcal{P}_t}$ is computable by* Logspace*-uniform arithmetic circuits over $\mathbb{F}$, with fan-in 2, of depth $n^{O(\delta)}$ and size* poly(n).

By the perfect completeness of the flat-GKR protocol, the verifier outputs a correct claim in Step (10a), and the prover opens the commitments consistently with it and passes all of the tests preformed in Step (10).

Moreover, recall that the honest $\mathcal{P}_{\mathsf{batch}}$ should preserve the property of a low-depth prover (see Definition 3.35). The following proposition captures this, and its proof is also deferred to Appendix B.3.

Proposition 5.6 ($\mathcal{P}_{\mathsf{batch}}$ is low-depth)**.** *Assume that the prover strategy $\mathcal{P}_t$ of the base protocol $(\mathcal{P}_t, \mathcal{V}_t)$ that $\mathcal{P}_{\mathsf{batch}}$ runs in Step (6c) is n^δ-low-depth. Then, $\mathcal{P}_{\mathsf{batch}}$ is also n^δ-low-depth.*

Lastly, notice that the queries in Q are input-oblivious (see Definition 3.31), since we assumed that $(\mathcal{P}_t, \mathcal{V}_t)$ is input-oblivious. The queries in Step (10c), checking consistency between the values of the openings and v_i (the value of the output claim of the $\mathsf{flat\text{-}GKR}$ execution performed in Step (10a)), are also input-oblivious. By Remark 5.3, the queries induced by transforming the IAs used as sub-protocols to PCIAa are all input-oblivious queries, and it follows that $\mathcal{V}_{\mathsf{batch}}$ only makes input-oblivious queries.

Soundness. Let $\mathcal{P}^*_{\mathsf{batch}}$ be an encoded cheating prover. Let $x \in \{0,1\}^n$, $u, v \in \{0,1\}^{O(S)}$ and (ω, θ) such that $(x, u, v, (\omega, \theta)) \notin \mathcal{L}_{k \cdot t}$. This means that at the beginning of the protocol $\mathcal{P}^*_{\mathsf{batch}}$ sends a message[17] $\mathsf{LDE}(\tilde{w}_t, \ldots, \tilde{w}_{(k-1)\cdot t})$ such that there exists $i^* \in [k-1]$ for which $\tilde{w}_{i^* \cdot t} \neq w_{i^* \cdot t}$ (namely, that $\tilde{w}_{i^* \cdot t}$ is not the correct configuration of the (deterministic) Turing machine $\mathcal{M}$ after $i^* \cdot t$ steps), *or* that u and v are consistent (namely, that $\mathcal{M}$ moves from configuration u to configuration v in exactly t steps, and $\mathcal{T}_{u,v}$ denotes this computation), but $\mathsf{LDE}(\mathcal{T}_{u,v})[\omega] \neq \theta$. We denote the former by $\tilde{w}_{(i^*-1)\cdot t} \overset{t}{\not\rightarrow} \tilde{w}_{i^* \cdot t}$. Recalling that $\tilde{w}_0 \overset{\text{def}}{=} u$ (and that $w_0 = u$) and that $\tilde{w}_{k \cdot t} \overset{\text{def}}{=} v$, notice that this includes the case that the prover cheats only on the first or the last transitions.

We use the notation $\mathcal{T}_i \in \{0,1\}^{t \cdot O(S)}$ for the sequence of t configurations starting at $\tilde{w}_{(i-1)\cdot t}$, for every $i \in [k]$. We say that $\tilde{y}_i$ is not correct w.r.t. a tableau $\mathcal{T}_i$ if $\tilde{y}_i \neq y_i$, where y_i is the correct commitment (i.e., the hash root of the UOWHF tree) to $\mathcal{T}_i$.

Suppose that after interacting with $\mathcal{P}^*_{\mathsf{batch}}$, the probability that $\mathcal{V}_{\mathsf{batch}}$ does not reject is s, and let A denote this event. Take B to be the following event:

$$\exists! i^* \in I \text{ s.t. } \left(\tilde{w}_{(i^*-1)\cdot t} \overset{t}{\not\rightarrow} \tilde{w}_{i^* \cdot t}\right) \vee \left(\tilde{y}_{i^*} \text{ is not correct w.r.t. } \mathcal{T}_{i^*}\right) \vee \left(\mathsf{LDE}(\mathcal{T}_{i^*})[\omega_{i^*}] \neq \theta_{i^*}\right).$$

We call such an index a *bad* index, which means that B is the event that there exists a single bad index in I. We take $I' = I \setminus \{i^*\}$ to be the set of indices that are not *bad*, assuming B. If B does not happen, we trivially define I' to be an empty set.

We define the following event E: the prover's answers in executions $i \in I'$ of the PCIA for $\mathcal{L}_t$ are according to the prescribed strategy $\mathcal{P}_t$. Namely, E implies that

$$\forall i \in I',\ \forall (j,t) \in Q : \phi_i(j,t) = \alpha_i^j[t].$$

The following proposition shows that E holds w.h.p. First, we prove that, unless the prover breaks the commitment scheme, then its openings to inputs in I' in Step (10c) are consistent with the tableaux. Then, we show that, unless the prover breaks the soundness property of $\mathsf{flat\text{-}GKR}$, then consistency with the tableaux implies consistency with the prescribed strategy.

Proposition 5.7. $\Pr[A \wedge B \wedge \neg E] \leq k \cdot negl(n) + 1/n$.

Proof. We start with additional notations for Steps (10b) and (10c):

- In Step (10b), where $\mathcal{P}^*_{\mathsf{batch}}$ opens the commitments at coordinate r, it computes the leaf index $q = \left\lceil \frac{r}{n_{in}} \right\rceil$, sends it and the path $p^i = \left(p_1^i, ..., p_{\ell+1}^i\right)$ that corresponds to q.
- In Step (10c), $\mathcal{V}_{\mathsf{batch}}$ receives the answers and checks that the paths are valid openings (Definition 3.14) w.r.t. h, y_i and q.

[17] Here we use the fact that $\mathcal{P}^*_{\mathsf{batch}}$ is an encoded prover and so the message that it sends must be a low degree extension encoding of some (possibly incorrect) configurations.

- Then, in order to check consistency with v_i, the verifier checks that $p^i_{\ell+1}[r'] = v_i$, where $r' = r \pmod{n_{in}}$, i.e., r' is the relative index of r in the q^{th} leaf.

We proceed with the proof. Assuming B, for any $i \in I'$ it holds that y_i is the correct commitment of $\mathsf{LDE}(\mathcal{T}_i)$. We use the targeted-collision-resistance of the commitment scheme (see Proposition 3.16) in order to show that for any $i \in I'$,

$$\Pr\left[p^i_{\ell+1}[r'] \neq \mathsf{LDE}(\mathcal{T}_i)[r] \text{ and } \mathcal{V} \text{ accepts}\right] = negl(n),$$

while recalling that $negl(n_{out}) = negl(n)$. Then, by a union bound, we get that

$$\Pr\left[\exists i \in I' \text{ s.t. } p^i_{\ell+1}[r'] \neq \mathsf{LDE}(\mathcal{T}_i)[r] \text{ and } \mathcal{V} \text{ accepts}\right] \leq |I'| \cdot negl(n) \leq k \cdot negl(n).$$

First, recall that $\mathcal{M}$ is a deterministic Turing machine. Thus, for each $i \in I'$, the tableau $\mathsf{LDE}(\mathcal{T}_i)$ is also uniquely defined (as the t consecutive configurations of $\mathcal{M}$ after $\tilde{w}_{(i-1)\cdot t}$). Getting back to the targeted-collision-resistance property of the scheme (Definition 3.15) and recalling that $\mathcal{P}^*_{\mathsf{batch}}$ passes the paths check of Step (10c), we get that if $\mathcal{P}^*_{\mathsf{batch}}$ implicitly declares the string $\mathsf{LDE}(\mathcal{T}_i)$, which is defined *before* the hash functions are chosen, and sends the correct hash root y_i as the commitment to $\mathsf{LDE}(\mathcal{T}_i)$, then, for the index q and the path p^i:

$$\begin{aligned}&\Pr[p^i \text{ is a } \textit{valid path from } q \textit{ to } y_i \text{ and } p^i_{\ell+1}[r'] \neq \mathsf{LDE}(\mathcal{T}_i)[r]] \leq \\ &\Pr[p^i \text{ is a } \textit{valid path from } q \textit{ to } y_i \text{ and} \\ &\qquad p^i_{\ell+1} \neq \mathsf{LDE}(\mathcal{T}_i)[(q-1)n_{in}] \ldots \mathsf{LDE}(w_i)[q \cdot n_{in} - 1]] = negl(n).\end{aligned}$$

The probability is over the choice of the UOWHFs, since w.l.o.g. $\mathcal{P}^*_{\mathsf{batch}}$ is deterministic (recall that it is non-uniform, so its optimal random coins can always be fixed non-uniformly).

Assuming that $\mathcal{P}^*_{\mathsf{batch}}$ does not break the security of the commitment, i.e., that for any $i \in I'$, it holds that $p^i_{\ell+1}[r'] = \mathsf{LDE}(\mathcal{T}_i)[r]$, then, since it also passes the test of Step (10c), we get that $\mathsf{LDE}(\mathcal{T}_i)[r] = v_i$. In other words, (r_i, v_i) is a correct claim about $\mathsf{LDE}(\mathcal{T}_i)$. Hence, (r, v'_i) is a correct claim about $\mathsf{LDE}(x, \mathcal{T}_i)$, and this yields that ϕ_i is a sequence of correct answers about the prover's messages in the i^{th} execution of the PCIA for $\mathcal{L}_t$, w.r.t. the query set Q and the random coins $\beta_1, \ldots, \beta_r$. Namely, ϕ_i is consistent with the prescribed strategy $\alpha^1_i, \ldots, \alpha^r_i$, that is:

$$\forall (j,t) \in Q : \phi_i(j,t) = \alpha^j_i[t],$$

where recall that $\alpha^j_i[t]$ is the t^{th} entry in the prover's messages in the j^{th} round (according to the prescribed strategy). The proposition follows by a union bound over all $i \in I'$, and taking $p(n) = k \cdot n$ in Theorem 3.23 (recall that $k \leq \mathsf{poly}(n)$). □

Next, assuming that B and E hold, we prove that ϕ_{i^*} (the prover's answers to the queries of the verifier in the $i^{*\text{th}}$ execution of the PCIA) is determined non-adaptively according to a fixed string before the protocol begins.

Proposition 5.8. *Take $\alpha^j = C^j_I - \sum_{i \in I'} \alpha^j_i$ for any $j \in [r]$, and assume that $(A \wedge B \wedge E)$ holds. Then, $\mathcal{P}^*_{\mathsf{batch}}$ uses $(\alpha^j)_{j \in [r]}$ as answers to the i^{*th} execution of $(\mathcal{P}_t, \mathcal{V}_t)$ in Step (6c), and α^j is an LDE codeword for any $j \in [r]$.*

Proof. Recall that, assuming B, the set I' is the set of indices that are not *bad*: $I' = I \setminus \{i^*\}$. Since $\mathcal{P}^*_{\mathsf{batch}}$ passes the checksum check in Step (8b), as we assume that A happens (namely, that the verifier does not reject), we get that:

$$\forall (j,t) \in Q,\ C_I^j[t] = \sum_{i \in I} \phi_i(j,t) = \sum_{i \in I'} \phi_i(j,t) + \phi_{i^*}(j,t) = \sum_{i \in I'} \alpha_i^j[t] + \phi_{i^*}(j,t),$$

where the last equality follows by assuming E. Thus,

$$\forall (j,t) \in Q,\ \phi_{i^*}(j,t) = C_I^j[t] - \sum_{i \in I'} \alpha_i^j[t].$$

In other words, for any $(j,t) \in [r] \times [a]$ that $\mathcal{V}_{\mathsf{batch}}$ chooses at random, the value $\phi_{i^*}(j,t)$ that the prover sends is determined before the execution begins: it is equal to $\alpha[(j,t)]$, where we define $\alpha = \alpha^1 \circ \cdots \circ \alpha^r$.

To conclude, notice that any α^j is a codeword of the LDE encoding, since the code (which consists of low-degree polynomials) is closed under addition and scalar multiplication, and we know that $C_I^j, (\alpha_i^j)_{i \in I'}$ are all codewords as they are sent by the encoded prover $\mathcal{P}^*_{\mathsf{batch}}$.

□

Having established the non-adaptivity of the prover in the $i^{*\text{th}}$ execution of $(\mathcal{P}_t, \mathcal{V}_t)$, we turn to prove that the input to the this execution is not in the language $\mathcal{L}_t$. Take F to denote the event that in Step (6b), the prover breaks the soundness of the interactive reduction of Proposition 3.10, which happens with probability at most $O(|\mathbb{H}|)/|\mathbb{F}|$ (we use the claim with $\varepsilon = 1$). We recall the definition of the event B and split $(B \wedge \neg F)$ into three cases.

- $\tilde{w}_{(i^*-1)\cdot t} \overset{t}{\not\rightsquigarrow} \tilde{w}_{i^* \cdot t}$. In this case, it is clear that the input is not in $\mathcal{L}_t$.
- $\tilde{y}_{i^*}$ is not the correct hash root of $\mathcal{T}_{i^*}$. In this case, the $\mathsf{flat\text{-}GKR}$ execution in Step (6a) outputs a false claim $(r^{\text{root}}, v_i^{\text{root}})$. Then, the interactive reduction performed in Step (6b) outputs a false claim $(\omega_i^{\mathcal{T}}, \theta_i^{\mathcal{T}})$, since we assumed that $\neg F$ holds, and the input is not in $\mathcal{L}_t$.
- $\mathsf{LDE}(\mathcal{T}_{i^*})[\omega_{i^*}] \neq \theta_{i^*}$. Then, once again, $\neg F$ implies that $(\omega_i^{\mathcal{T}}, \theta_i^{\mathcal{T}})$ is a false claim about $\mathcal{T}_{i^*}$ and the input is not in $\mathcal{L}_t$.

Overall, the input to the $i^{*\text{th}}$ execution is not in the language. Next, we show that this implies that $\mathcal{P}^*_{\mathsf{batch}}$ breaks the soundness property of the PCIA.

Claim 5.9. $\Pr[A \wedge B \wedge E \wedge \neg F] \leq \varepsilon/30 \log k$.

Proof. Assume B holds. We reduce the cheating strategy $\mathcal{P}^*_{\mathsf{batch}}$ for a cheating strategy $\mathcal{P}^*_t$ for the PCIA. The prover $\mathcal{P}^*_t$ works as follows:

1. $\mathcal{P}^*_t$ interacts with $\mathcal{P}^*_{\mathsf{batch}}$ until it reaches Step (6c).[18] Upon receiving $\mathsf{LDE}(\tilde{w}_t, \ldots, \tilde{w}_{(k-1)\cdot t})$, the prover $\mathcal{P}^*_t$ finds the unique index $i^* \in I$ such that $\left(x, \tilde{w}_{(i^*-1)\cdot t}, \tilde{w}_{i^* \cdot t}, (\omega_{i^*}^{\mathcal{T}}, \theta_{i^*}^{\mathcal{T}})\right) \notin \mathcal{L}_t$. Notice that i^* is unique since B holds, and that $\mathcal{P}^*_t$ can find i^* in polynomial time by checking membership in $\mathcal{L}_t$ for all $i \in I$.

[18] Recall that $\mathcal{P}^*_t$, as well as any other cheating prover strategy considered throughout this work, is non-uniform. Thus, w.l.o.g., we may assume that it has access to randomness, by fixing non-uniformly the optimal random coins.

2. Then, $\mathcal{P}_t^*$ starts interacting with the verifier $\mathcal{V}_t$. For every $j \in [r]$:
   (a) $\mathcal{V}_t$ sends randomness β^j.
   (b) $\mathcal{P}_t^*$ sends β^j to $\mathcal{P}^*_{\mathsf{batch}}$.
   (c) $\mathcal{P}^*_{\mathsf{batch}}$ answers to $\mathcal{P}_t^*$ with a checksum C_I^j.
   (d) $\mathcal{P}_t^*$ defines $\alpha^j = C_I^j - \sum_{i \in I'} \alpha_i^j$, and sends α^j as an answer to $\mathcal{V}_t$.

Proposition 5.8 implies that when $\mathcal{P}^*_{\mathsf{batch}}$ uses $(\alpha^j)_{j\in[r]}$, then $\mathcal{V}_{\mathsf{batch}}$ accepts, which in turn implies that $\mathcal{V}_t$ also accepts. Moreover, it implies that $(\alpha^j)_{j\in[r]}$ are all codewords of the LDE encoding. Hence, soundness against encoded provers applies.

We conclude that, assuming that B holds, the event $(A \wedge E \wedge \neg F)$ implies that the prover $\mathcal{P}_t$ breaks the soundness of the PCIP for $\mathcal{L}_t$, thus

$$\Pr\left[A \wedge E \wedge \neg F \mid B\right] = \varepsilon.$$

The claim follows using Proposition 3.1 (justified by the fact that indeed there must exists at least one bad index). □

All in all, recalling that $|\mathbb{F}| = \Theta(n^{2\delta} \cdot \log n)$, $|\mathbb{H}| = \Theta(n^\delta)$, $k \leq \mathsf{poly}(n)$ and using Propositions 3.10 and 3.1, we get that

$$\begin{aligned} s = \Pr[A] &\leq \Pr[A \wedge B \wedge E \wedge \neg F] + \Pr[F] + \Pr[A \wedge B \wedge \neg E] + \Pr[\neg B] \\ &\leq \varepsilon/30 \log k + O(|\mathbb{H}|)/|\mathbb{F}| + k \cdot negl(n) + 1/n + 1 - 1/30 \log k \\ &\leq 1 - (1-\varepsilon)/30 \log k + O(1/\log n). \end{aligned}$$

Complexity. Recall that $|x| = n$, where x is the original input to $\mathcal{L}_t$, and that the base protocol is a $(q, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-$\mathsf{PCIA}$. Notice that the circuits on which we run the $\mathsf{flat\text{-}GKR}$ protocol in Steps (6a) and (10a) are of depth $O(1/\delta) \cdot \mathsf{poly}(n_{in}) = n^{O(\delta)}$, justified by Propositions 3.19 and 5.5, and that the length of the explicit input to each execution is at most $n^{O(\delta)}$, by the assumptions required in the theorem statement. The complexities of $(\mathcal{P}_{\mathsf{batch}}, \mathcal{V}_{\mathsf{batch}})$ are as follows:

- **Soundness error**: $1 - (1-\varepsilon)/30 \log k + O(1/\log n)$, as argued above.
- **Query complexity**: In Step (9), $\mathcal{V}_{\mathsf{batch}}$ makes $q \cdot (d+1)$ queries, the first q for each of the $d \leq k$ base PCIAs, and another one for the checksum. In Step (10c), it makes another query $(\forall i \in I)$ for checking consistency between the value of the opening and v_i. We use Remark 5.3 and get that transforming the $\mathsf{flat\text{-}GKR}$ protocol (Theorem 3.23) to a PCIA adds $n^{O(\delta)}$ queries, and the procedure of Step (6b) adds $n^{O(\delta)}$ queries as well. Overall, $q_{\mathsf{B}} \leq (k+1) \cdot q + k + n^{O(\delta)} \leq (2k+1) \cdot q + n^{O(\delta)}$.
- **Round complexity**: $r_{\mathsf{B}} = r + O(1/\delta^3)$, dominated by $\mathsf{flat\text{-}GKR}$'s rounds.
- **Prover's message length**: Since $\mathcal{P}_{\mathsf{batch}}$ is *encoded*, all of its messages are of length $|\mathbb{F}|^m$ for $m = O(1/\delta)$, thus for any message w, the length of its encoding is $|\mathsf{LDE}(w)| = |w|^2 \cdot \mathsf{polylog}(n)$. Sending the hash roots and opening the commitments are dominated by $\mathsf{flat\text{-}GKR}$'s message length, which is $n^{O(\delta)}$ by Theorem 3.23. The messages in Step (2) have length at most $k \cdot \log |\mathbb{F}| \leq k \cdot n^{O(\delta)}$. The length of the checksums is a: we do not pay with a multiplicative

factor on the prover's messages length in the base protocol, namely, we do not carry a $k \cdot a$ term, because the prover only sends the checksum. Notice that these messages are already encoded, as $\mathcal{P}_t$ is encoded. The length of the answers sent in Step (8) is at most $k \cdot q$, and so applying the second condition required in the theorem statement, we conclude that $a_{\mathsf{B}} = a$.

- **Verifier's message length**: We stress that $\mathcal{V}_{\mathsf{batch}}$ uses the same random coins $\beta^1, \ldots, \beta^r$ for all $d \leq k$ protocols. The maximal verifier's message length in each of the flat-GKR executions is $n^{O(\delta)}$. The maximal verifier's message length in Proposition 3.10 is $k \cdot O(m \cdot |\mathbb{H}| \cdot \log |\mathbb{F}|) \leq k \cdot n^{O(\delta)}$. The queries are of length $q \cdot (\log r + \log a) \leq q \cdot \log n$. Recalling that $O(q) \leq k \cdot n^{O(\delta)}$ and that we assume that $r \cdot b \leq k \cdot n^{O(\delta)}$, we conclude that $b_{\mathsf{B}} = k \cdot n^{O(\delta)}$.

- **Prover's running time**: $\mathcal{P}\mathsf{time}_{\mathsf{B}}$ is dominated by the flat-GKR executions in Step (10a) and the base protocol executions in Step (6c). The former can be done in time $\mathsf{poly}(\log |\mathbb{F}|, n^{\delta}, t \cdot S, q)$, and since $q \leq k \cdot n^{O(\delta)}$, this yields $\mathcal{P}\mathsf{time}_{\mathsf{B}} = \mathsf{poly}(k, n^{\delta}, t, S) + k \cdot \mathcal{P}\mathsf{time}$. Note that Proposition 3.6 implies that computing the LDE of the k configurations, where each configuration is of size $O(S)$, can be done in time $(k \cdot S)^{3+2\delta} \cdot \mathsf{polylog}(n) \leq \mathsf{poly}(k, n^{\delta}, t, S)$.

- **Verifier's running time**: Similarly to the previous item, the verification time is dominated by two steps: the flat-GKR executions in Step (10a) and the base protocol executions in Step (6c). As for the former, the verifier runs for $k \cdot \left(n^{O(\delta)} + (q \cdot \log |\mathbb{F}| + r \cdot b) \cdot n^{O(\delta)}\right)$ time. Due to the assumptions that $q \leq \mathcal{V}\mathsf{time}$ and that $r \cdot b \leq k \cdot n^{O(\delta)}$, we get that the overall running time of the verifier is bounded by $\mathcal{V}\mathsf{time}_{\mathsf{B}} = \mathcal{V}\mathsf{time} \cdot \mathsf{poly}(k, n^{\delta})$.

This completes the proof of Lemma 5.2. □

5.2 Augmentation Protocol

As described in the technical overview (Section 2), our construction is based on iterative and interleaved applications of the Batch Verification protocol (Lemma 5.2), our main technical tool, and [RRR16]'s Query Reduction protocol (Lemma 3.36). Using these two, we show an efficient transformation from a holographic PCIA for $\mathcal{L}_t$ (see Definition 3.34) to a holographic PCIA for $\mathcal{L}_{k \cdot t}$. We call it the Augmentation protocol since it "augments" the length of computations, but we stress that the conceptual step is the same as in the batch verification protocol: the only difference between them is that the augmentation protocol achieves low query complexity.

Lemma 5.10 (Augmentation Lemma)**.** *Assume one-way functions exist, and let $\delta \in (0, 1)$ be a constant. Let $\mathbb{H} \subseteq \mathbb{F}$ be (ensembles of) extension fields of $\mathbb{GF}[2]$, where $|\mathbb{H}| = \Theta(n^{\delta})$ and $|\mathbb{F}| = \Theta(n^{2\delta} \cdot \log n)$. Let $k = k(n) \leq \mathsf{poly}(n)$ be a parameter, and define $\sigma = \delta^3$.*

Suppose that $\mathcal{L}_t^{\mathcal{M}}$ (Definition 3.34) has an ε-sound $(q, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIA $(\mathcal{P}_t, \mathcal{V}_t)$ w.r.t. encoded provers and with input-oblivious queries. Assume that

- $q \leq \min\left(\mathcal{V}\mathsf{time}, k \cdot n^{O(\delta)}\right)$;
- $a \geq \max\left((k \cdot S)^2 \cdot \mathsf{polylog}(n), k^2 \cdot n^{O(\delta)}, (k \cdot q)^2 \cdot \mathsf{polylog}(n)\right)$;
- $r \cdot b \leq k \cdot n^{O(\delta)}$;
- $|\mathbb{H}| \leq (k \cdot q)^{\sigma}$.[19]

[19] We comment that the assumption that Lemma 3.36 (the query reduction protocol) requires is $|\mathbb{H}| \leq \min(k \cdot q, k \cdot (n^{O(\delta)} + \mathcal{V}\mathsf{time}))^{\sigma}$, however, it follows from this assumption when combining it with the first one.

Then, there exists an $\varepsilon_{\mathsf{aug}}$*-sound* $(q_{\mathsf{aug}}, r_{\mathsf{aug}}, a_{\mathsf{aug}}, b_{\mathsf{aug}}, \mathcal{P}\mathsf{time}_{\mathsf{aug}}, \mathcal{V}\mathsf{time}_{\mathsf{aug}})$-PCIA $(\mathcal{P}_{k\cdot t}, \mathcal{V}_{k\cdot t})$ *for the language* $\mathcal{L}_{k\cdot t}$ *w.r.t. encoded provers and with input-oblivious queries, with the following parameters:*

- $\varepsilon_{\mathsf{aug}} = 1 - (1-\varepsilon)/30 \log k + O(1/\log n)$.
- $q_{\mathsf{aug}} = (k \cdot n^{\delta} \cdot \mathcal{V}\mathsf{time})^{\sigma} + O\left((r + 1/\sigma) \cdot \log n\right)$.
- $r_{\mathsf{aug}} = r + O(1/\sigma)$.
- $a_{\mathsf{aug}} = \max\Big(a, \mathsf{poly}\big(r, 1/\sigma, k, n^{\delta}, \mathcal{V}\mathsf{time}\big)\Big)$.
- $b_{\mathsf{aug}} = k \cdot n^{O(\delta)}$.
- $\mathcal{P}\mathsf{time}_{\mathsf{aug}} = \mathsf{poly}(k, n^{\delta}, t, S) + k \cdot \mathcal{P}\mathsf{time} + \mathsf{poly}\Big(k, n^{\delta}, r, 1/\sigma, \mathcal{V}\mathsf{time}, b\Big)$.
- $\mathcal{V}\mathsf{time}_{\mathsf{aug}} = (k \cdot n^{\delta} \cdot \mathcal{V}\mathsf{time})^{\sigma} + \mathsf{poly}\Big(r, 1/\sigma, k, n^{\delta}\Big)$.

Furthermore, if $\mathcal{P}_t$ *is* n^{δ}*-low-depth, then* $\mathcal{P}_{k\cdot t}$ *is* n^{δ}*-low-depth as well.*

Notice that there is only one free parameter, δ, that fixes σ once it is set. Still, we state the complexities in terms of δ and σ, to ease the application of Lemma 3.36 and to simplify notation.

As discussed in the overview, we use our batch verification lemma for $\mathcal{L}_t$ to obtain an efficient PCIA $(\mathcal{P}_{\mathsf{B}}, \mathcal{V}_{\mathsf{B}})$ for verifying all k statements (the B abbreviates *Batched*). The latter PCIA has relatively many queries, so rather than running it directly, we first apply the query reduction transformation on it to derive a query-efficient PCIA $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ (where BQ stands for *Batched* and *Query Reduced*) and then have the verifier and prover emulate $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$.

The PCIA for $\mathcal{L}_{k\cdot t}$, denoted by $(\mathcal{P}_{k\cdot t}, \mathcal{V}_{k\cdot t})$, is presented next.

PCIA $(\mathcal{P}_{k\cdot t}, \mathcal{V}_{k\cdot t})$ **for** $\mathcal{L}_{k\cdot t}$

Prover's Input: $x \in \{0,1\}^n$, configurations $u, v \in \{0,1\}^{O(S)}$ and a claim (ω, θ).
Verifier's Input: explicit access to (ω, θ) and implicit access to $\mathsf{LDE}(x, u, v)$.

- Let $(\mathcal{P}_{\mathsf{B}}, \mathcal{V}_{\mathsf{B}})$ be the batched PCIA for the language $\mathcal{L}_{k\cdot t}$, obtained by applying Lemma 5.2 to the language $\mathcal{L}_t$ (which has the PCIA $(\mathcal{P}_t, \mathcal{V}_t)$).
- Let $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ be the query reduced PCIA (also for the language $\mathcal{L}_{k\cdot t}$), obtained by applying Lemma 3.36 to the PCIA $(\mathcal{P}_{\mathsf{B}}, \mathcal{V}_{\mathsf{B}})$, w.r.t. parameter $\sigma' = \sigma/c$, where $c \geq 1$ is some sufficiently large constant.
- The prover $\mathcal{P}_{k\cdot t}$ and verifier $\mathcal{V}_{k\cdot t}$ run $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ on input $(x, u, v, (\omega, \theta))$, where $\mathcal{V}_{k\cdot t}$ uses the procedure in Proposition 3.7 to answer $\mathcal{V}_{\mathsf{BQ}}$'s input queries. If $\mathcal{V}_{\mathsf{BQ}}$ accepts then $\mathcal{V}_{k\cdot t}$ accepts and otherwise it rejects.

Completeness. Completeness follows immediately from the prefect completeness of the protocols of Lemmas 3.36 and 5.2, and the input-oblivious property follows from these lemmas as well. The n^{δ}-low-depth property of the honest prover (Definition 3.35) follows from Lemmas 3.36 and 5.2.

We stress that the depth of the circuit computing $\mathcal{P}_{k\cdot t}$'s messages only increases by a multiplicative $O(\log n)$ factor, due to Proposition 5.6, thus a constant number of iterative applications of the augmentation protocol (performed in the next section) still keep the prover n^{δ}-low-depth.

Soundness. Let $\mathcal{P}_{k\cdot t}$ be an encoded cheating prover. Let $x \in \{0,1\}^n$, $u, v \in \{0,1\}^{O(S)}$ and (ω, θ) such that $((x,u,v),(\omega,\theta)) \notin \mathcal{L}_{k\cdot t}$. Then, $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ is run on an input that does not belong to $\mathcal{L}_{k\cdot t}$ and so, using Lemmas 3.36 and 5.2, the probability that $\mathcal{V}_{k\cdot t}$ accepts after interacting with $\mathcal{P}_{k\cdot t}$ is at most the sum of errors of the batching protocol and the query reduction protocol (and it is computed next).

Complexity. Let $\delta \in (0,1)$ and recall that $\sigma \stackrel{\text{def}}{=} \delta^3$. First, note that the parameters satisfy the requirements of Lemmas 3.36 and 5.2 with parameters k and σ, by the assumptions made in this theorem (i.e., Lemma 5.10), and by the fact that $\log(\max(n_{\mathsf{B}}, r_{\mathsf{B}}, a_{\mathsf{B}}, b_{\mathsf{B}}, \mathcal{P}\mathsf{time}_{\mathsf{B}}, \mathcal{V}\mathsf{time}_{\mathsf{B}})) \leq |\mathbb{H}|$ always holds by the choice of $|\mathbb{H}|$, where $n_{\mathsf{B}} = n + O(k \cdot S)$. Further, $|\mathbb{H}| \leq \min(k \cdot q, k \cdot (n^{O(\delta)} + \mathcal{V}\mathsf{time}))^{\sigma'}$ implies $|\mathbb{H}| \leq \min(q_{\mathsf{B}}, \mathcal{V}\mathsf{time}_{\mathsf{B}})^{\sigma'}$.

We take σ' such that $\mathsf{poly}(k, n^{\delta}, q, \mathcal{V}\mathsf{time})^{\sigma'} = (k, n^{\delta}, q, \mathcal{V}\mathsf{time})^{\sigma}$. Notice that this is possible, since the unspecified $\mathsf{poly}(\cdot)$ comes from the query reduction procedure and is independent of n and δ. Moreover, recall that $\log|\mathbb{F}| = O(\log n)$ and that $k \leq \mathsf{poly}(n)$.

1.
$$\begin{aligned}\varepsilon_{\mathsf{aug}} &= \varepsilon_{\mathsf{B}} + O\left((|\mathbb{H}| \cdot m)^2\right)/|\mathbb{F}| \\ &= 1 - (1-\varepsilon)/30 \log k + O(1/\log n) + O(|\mathbb{H}|^2)/\left(|\mathbb{H}|^2 \cdot \log n\right) \\ &\leq 1 - (1-\varepsilon)/30 \log k + O(1/\log n).\end{aligned}$$

2.
$$\begin{aligned}q_{\mathsf{aug}} &= \mathsf{poly}(\mathcal{V}\mathsf{time}_{\mathsf{B}})^{\sigma'} + O(r_{\mathsf{B}} \cdot \log|\mathbb{F}|) \\ &= \mathsf{poly}(\mathcal{V}\mathsf{time} \cdot \mathsf{poly}(k, n^{\delta}))^{\sigma'} + O((r + O(1/\delta^3)) \cdot \log|\mathbb{F}|) \\ &= \mathsf{poly}(k, n^{\delta}, \mathcal{V}\mathsf{time})^{\sigma'} + O\left((r + 1/\sigma) \cdot \log n\right) \\ &\leq (k \cdot n^{\delta} \cdot \mathcal{V}\mathsf{time})^{\sigma} + O\left((r + 1/\sigma) \cdot \log n\right).\end{aligned}$$

3.
$$r_{\mathsf{aug}} = r_{\mathsf{B}} + O(1/\sigma') = r + O(1/\delta^3) + O(1/\sigma') = r + O(1/\sigma).$$

4.
$$\begin{aligned}a_{\mathsf{aug}} &= \max\left(a_{\mathsf{B}}, \mathsf{poly}(r_{\mathsf{B}}, b_{\mathsf{B}}, \mathcal{V}\mathsf{time}_{\mathsf{B}}, |\mathbb{H}|)\right) \\ &= \max\left(a, \mathsf{poly}\big((r + O(1/\delta^3)), k \cdot n^{O(\delta)}, \mathcal{V}\mathsf{time} \cdot \mathsf{poly}(k, n^{\delta}), |\mathbb{H}|\big)\right) \\ &\leq \max\left(a, \mathsf{poly}\big(r, 1/\sigma, k, n^{\delta}, \mathcal{V}\mathsf{time}\big)\right).\end{aligned}$$

5.

$$\begin{aligned} b_{\mathsf{aug}} &= \max\left(b_{\mathsf{B}}, O(|\mathbb{H}| \cdot \log|\mathbb{F}|)\right) \\ &= \max\left(b, k \cdot n^{O(\delta)}, O(|\mathbb{H}| \cdot \log|\mathbb{F}|)\right) = k \cdot n^{O(\delta)}. \end{aligned}$$

6.

$$\begin{aligned} \mathcal{P}\mathsf{time}_{\mathsf{aug}} &= \mathcal{P}\mathsf{time}_{\mathsf{B}} + \mathsf{poly}(q_{\mathsf{B}}, r_{\mathsf{B}}, b_{\mathsf{B}}, \mathcal{V}\mathsf{time}_{\mathsf{B}}) \\ &= \mathsf{poly}(k, n^{\delta}, t, S) + k \cdot \mathcal{P}\mathsf{time} + \\ &\quad \mathsf{poly}\Big((2k+1) \cdot q + n^{O(\delta)}, r + O(1/\delta^3), k \cdot n^{O(\delta)}, \mathcal{V}\mathsf{time} \cdot \mathsf{poly}(k, n^{\delta})\Big) \\ &\leq \mathsf{poly}(k, n^{\delta}, t, S) + k \cdot \mathcal{P}\mathsf{time} + \mathsf{poly}\Big(k, n^{\delta}, r, 1/\sigma, \mathcal{V}\mathsf{time}, b\Big). \end{aligned}$$

7.

$$\begin{aligned} \mathcal{V}\mathsf{time}_{\mathsf{aug}} &= \mathsf{poly}(\mathcal{V}\mathsf{time}_{\mathsf{B}})^{\sigma'} + \mathsf{poly}(b_{\mathsf{B}}, r_{\mathsf{B}}, |\mathbb{H}|) \\ &= \mathsf{poly}(\mathcal{V}\mathsf{time} \cdot \mathsf{poly}(k, n^{\delta}))^{\sigma'} + \mathsf{poly}\Big(k \cdot n^{O(\delta)}, (r + O(1/\delta^3)), |\mathbb{H}|\Big) \\ &= (k \cdot n^{\delta} \cdot \mathcal{V}\mathsf{time})^{\sigma} + \mathsf{poly}\left(r, 1/\sigma, k, n^{\delta}\right). \end{aligned}$$

5.3 PCIA for Bounded-Space Computations

In this section we construct an efficient PCIAs w.r.t. encoded provers for bounded-space computations (Theorem 5.11). Our main result, which is efficient interactive arguments for bounded-space computations (Theorem 5.1) follows from Theorem 5.11 by using the transformation between PCIA and interactive arguments (Proposition 3.33).

Theorem 5.11 (PCIA for Bounded-Space w.r.t. Encoded Provers)**.** *Assume one-way functions exist and let $T = T(n)$ and $S = S(n)$ such that $n \leq T \leq \mathsf{exp}(n)$ and $\log(T) \leq S \leq \mathsf{poly}(n)$. Let $\delta \in (0,1)$, and let $\mathbb{H} \subseteq \mathbb{F}$ be (ensembles of) extension fields of $\mathbb{GF}[2]$, where $|\mathbb{H}| = \Theta(n^{\delta})$ and $|\mathbb{F}| = \Theta(n^{2\delta} \cdot \log n)$.*

Then, for every $\mathcal{L} \in \mathsf{DTISP}(T, S)$ there exists an $(q, r, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIA w.r.t. encoded provers, with input-oblivious queries and the following parameters:

- $\varepsilon = 1 - 1/\mathsf{polylog}(T)$.
- $q = O\left(T^{O(\delta^3)}\right)$.
- $r = O(1/\delta^4)$.
- $a = O\left(T^{O(\delta)} \cdot S^2\right)$.
- $b = T^{O(\delta)}$.
- $\mathcal{P}\mathsf{time} = \mathsf{poly}(T, S)$
- $\mathcal{V}\mathsf{time} = O\left(T^{O(\delta)}\right)$.

Proof. Fix T and S as above and let $\mathcal{M}$ be a time T and space S Turing machine for $\mathcal{L}$. We assume that $\mathcal{M}$ is an *oblivious* Turing machine, while noting that any time-T and space-S Turing machine can be simulated by a time $T' = O(T \cdot S)$ and space $S' = O(S)$ *oblivious*[20] Turing machine (and we shall account for this additional overhead at the end of the proof). Recall that a configuration $w \in \{0,1\}^{O(S)}$ of a Turing machine includes the contents of all work tapes, the current time step, the positions of the work and input heads and the current internal state. Let $w_{\mathsf{start}} \in \{0,1\}^{O(S)}$ be the machine $\mathcal{M}$'s initial configuration. We assume w.l.o.g. that $\mathcal{M}$ has a *unique* accepting configuration w_{end}. Furthermore, we assume w.l.o.g. that the configurations w_{start} and w_{end} are (fixed) strings such that individual points in their respective low degree extensions $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w_{\mathsf{start}})$ and $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w_{\mathsf{end}})$ can be evaluated in $\mathsf{poly}(\mathbb{H})$ time.[21]

Let $\delta \in (0,1)$, fix $k \stackrel{\text{def}}{=} T^{\delta}$ and recall Definition 3.34 for $t \leq T$. We will show how to directly construct a PCIA for $\mathcal{L}_1$ (i.e., $t = 1$). Then, using iterative applications of Lemma 5.10 we will obtain holographic PCIAs for $\mathcal{L}_{(k^i)}$ for larger and larger values of $i \leq \log_k(T)$, until ultimately, when $i = \log_k(T)$, we obtain a PCIA for the language $\mathcal{L}_T$. A holographic PCIA for the language $\mathcal{L}$, as in the theorem's statement, follows by running the PCIA for $\mathcal{L}_T$ on implicit input $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x, w_{\mathsf{start}}, w_{\mathsf{end}})$, where input queries are emulated using queries to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$, $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w_{\mathsf{start}})$ and $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w_{\mathsf{end}})$, as shown in Proposition 3.7.

The "base" PCIA for $\mathcal{L}_1$. Let us recall the definition of the language $\mathcal{L}_1$:

$$\mathcal{L}_1 = \left\{ (x,u,v,(\omega,\theta)) : \begin{array}{c} \text{On input } x \in \{0,1\}^n, \mathcal{M} \text{ moves from configuration} \\ u \in \{0,1\}^{O(S)} \text{ to configuration } v \in \{0,1\}^{O(S)} \\ \text{and } \mathsf{LDE}_{\mathbb{F},\mathbb{H}}(u,v)[\omega] = \theta \end{array} \right\}.$$

A PCIA for $\mathcal{L}_1$ essentially verifies single steps of the machine and a claim about their low degree extension. Recall that prover has full access to the input, whereas the verifier has explicit access to (ω,θ) and implicit access to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x,u,v)$, since this is a holographic PCIA for a pair language (see Section 3.6). We remark that no interaction with a prover is necessary for this PCIA: the verifier can check membership in $\mathcal{L}_1$ by itself, but for technical convenience (specifically, to facilitate the application of Lemma 5.10) we will introduce some "dummy" interaction.

We state the following proposition in terms of $|\mathbb{H}|$ and $|\mathbb{F}|$ to ease its application in what follows, although it holds with respect to the fields stated in Theorem 5.11: $|\mathbb{H}| = \Theta(n^{\delta})$ and $|\mathbb{F}| = \Theta(n^{2\delta} \cdot \log n)$

Proposition 5.12. *The language $\mathcal{L}_1$ has a $(q_0, r_0, a_0, b_0, \mathcal{P}\mathsf{time}_0, \mathcal{V}\mathsf{time}_0)$-PCIA w.r.t. encoded provers, with n^{δ}-low-depth honest prover and with input-oblivious queries, with the following parameters:*

- $\varepsilon_0 = O(|\mathbb{H}|)/|\mathbb{F}|$.

[20] An oblivious Turing machine is a machine whose input and work tape head *positions* are a fixed function of the current time step (and in particular do not depend on the input). Furthermore, given a time step $t \in [T]$, the head positions in time t can be computed in $\mathsf{polylog}(T)$ time. We remark that more efficient simulation is known [PF79].

[21] We can assume w.l.o.g. that both the initial configuration w_{start} and (unique) accepting configuration w_{end} are zero on all but $O(\log(T))$ bits (at fixed locations) and the values of the non-zero bits can be computed by a Turing machine in $O(\log(T))$ time. Each point in their respective low degree extension is therefore a linear combination (over the field $\mathbb{F}$) of these $O(\log(T))$ bits. The coefficients of this linear combination (and therefore also its sum) can be computed in time $\mathsf{poly}(|\mathbb{H}|, \log_{|\mathbb{H}|}(S)) = \mathsf{poly}(\mathbb{H})$, see Section 3.1.

- $q_0 = \mathsf{poly}(|\mathbb{H}|)$.
- $r_0 = 1$.
- $a_0 = S^2 \cdot \mathsf{poly}(k, |\mathbb{H}|)$.
- $b_0 = k \cdot \mathsf{poly}(|\mathbb{H}|)$.
- $\mathcal{P}\mathsf{time}_0 = \mathsf{poly}\left(k, |\mathbb{H}|, S\right)$.
- $\mathcal{V}\mathsf{time}_0 = \mathsf{poly}(|\mathbb{H}|)$.

Proof. Let $\mathsf{LDE} = \mathsf{LDE}_{\mathbb{F},\mathbb{H}}$. The verifier first reads the time step t_u that appears as part of the configuration u. Since $\mathcal{M}$ is *oblivious*, given t_u it can determine in time $\mathsf{polylog}(T)$ a set $\Omega \subseteq [O(S)]$, of size $|\Omega| = O(\log(T))$, of coordinates on which u and v may differ (i.e., the time step, the position of the heads, the current state and the $O(1)$ locations in the work tapes on which the heads are currently positioned). The verifier queries the coordinates in Ω and checks that they were updated correctly (i.e., in accordance with the machine's specification and with the input x). For all other coordinates, it checks that they are equal in u and v. Since the low degree extension is a linear code, $\mathsf{LDE}(u) - \mathsf{LDE}(v) = \mathsf{LDE}(u - v)$ and so it only needs to check that $\mathsf{LDE}(u - v)$ is zero on all points outside Ω. The latter can be done by running the procedure in Proposition 3.9, in $\mathsf{poly}(|\mathbb{H}|)$ time, with an error probability of at most $\frac{O(|\mathbb{H}| \cdot \log_{|\mathbb{H}|}(S))}{|\mathbb{F}|} = \frac{O(|\mathbb{H}|)}{|\mathbb{F}|}$, as we assumed that $S \leq \mathsf{poly}(n)$.

In order to verify the claim (ω, θ) about $\mathsf{LDE}(u, v)$, the verifier makes three queries to its implicit input (using Proposition 3.7) and finds $\mathsf{LDE}(u, v)[\omega]$ in time $\mathsf{poly}(|\mathbb{H}|)$. If the result is different from θ, it rejects.

For technical convenience, since we want a PCIA that satisfies the requirements of Lemma 5.10 and the recurrence relations of Proposition 5.13, we also have the prover send a single "dummy" message of length $\mathsf{poly}(k, |\mathbb{H}|, S)$: the low degree extension of the all-zeros string, and the verifier a message of length $k \cdot \mathsf{poly}(|\mathbb{H}|)$: the all-zeros string. Observe that the honest prover is (trivially) n^δ-low-depth, as computing the low degree extension of the all-zeros string can be done by a depth-1 circuit that outputs a sequence of zeros. □

We proceed to prove to the inductive step.

Proposition 5.13. *Assume one-way functions exist and let $\delta \in (0, 1)$ be a constant. Take $\sigma = \delta^3$. Then, for every $k = k(n) \leq \mathsf{poly}(n)$ and $1 \leq i \leq O(1/\sigma)$, the language $\mathcal{L}_{(k^i)}$ has an ε_i-sound $(q_i, r_i, a_i, b_i, \mathcal{P}\mathsf{time}_i, \mathcal{V}\mathsf{time}_i)$-$\mathsf{PCIA}$ $(\mathcal{P}_i, \mathcal{V}_i)$ w.r.t. encoded provers, with n^δ-low-depth honest prover and with input-oblivious queries, with the following parameters:*

1. $\varepsilon_i = 1 - (1/30 \log k)^i \cdot (1 - \varepsilon_0) + O(i/\log n)$.
2. $q_i = (\mathcal{V}\mathsf{time}_0)^{\sigma^i} \cdot \mathsf{poly}\big(i/\sigma, k, n^\delta\big)^{i \cdot \sigma} + O\left((r_0 + i/\sigma) \cdot \log n\right)$.
3. $r_i = r_0 + i \cdot O(1/\sigma)$.
4. $a_i = \max\left(a_0, \mathsf{poly}\big(1/\sigma, k, n^\delta, r_0, \mathcal{V}\mathsf{time}_0^{\sigma^i}\big)\right)$.
5. $b_i = k \cdot n^{O(\delta)}$.

6. $\mathcal{P}\mathsf{time}_i = k^i \cdot \Big(\mathcal{P}\mathsf{time}_0 + i \cdot \mathsf{poly}\big(k, n^\delta, r_0, 1/\sigma, (\mathcal{V}\mathsf{time}_0)^\sigma\big)\Big) + i \cdot \mathsf{poly}(n^\delta, k^i, S)$.

7. $\mathcal{V}\mathsf{time}_i = (\mathcal{V}\mathsf{time}_0)^{\sigma^i} \cdot \mathsf{poly}\Big(i/\sigma, k, n^\delta\Big)^{1+i\cdot\sigma}$.

Proof. The proof is by induction. The base case $i = 0$ follows directly from Proposition 5.12. For the induction step, let $1 \leq i \leq O(1/\sigma)$. Assume inductively that $\mathcal{L}_{(k^{i-1})}$ has an ε_{i-1}-sound $(q_{i-1}, r_{i-1}, a_{i-1}, b_{i-1}, \mathcal{P}\mathsf{time}_{i-1}, \mathcal{V}\mathsf{time}_{i-1})$-PCIA $(\mathcal{P}_{i-1}, \mathcal{V}_{i-1})$ w.r.t. encoded provers, n^δ-low-depth honest prover and with input-oblivious queries, where the parameters are as in the proposition's statement.

We next show that the parameters of this PCIA satisfy the requirements of Lemma 5.10 with parameters k and δ. Recall the choice of $|\mathbb{H}|, |\mathbb{F}|$ as stated in Theorem 5.11. Indeed it holds that:

- $q_{i-1} \leq \min\big(\mathcal{V}\mathsf{time}_{i-1}, k \cdot n^{O(\delta)}\big)$:
  1. $q_{i-1} \leq \mathcal{V}\mathsf{time}_{i-1}$: Follows from their definitions as long as $\mathsf{poly}\big((i-1)/\sigma, k, n^\delta\big) \geq (r_0 + (i-1)/\sigma) \cdot \log n$, which is indeed the case since $n^\delta \geq \log n$ and $r_0 = 1$ due to Proposition 5.12.
  2. $q_{i-1} \leq k \cdot n^{O(\delta)}$: Recall that
  $$q_{i-1} = (\mathcal{V}\mathsf{time}_0)^{\sigma^{i-1}} \cdot \mathsf{poly}\big((i-1)/\sigma, k, n^\delta\big)^{(i-1)\cdot\sigma} + O\left((r_0 + (i-1)/\sigma) \cdot \log n\right),$$
  and note that the polynomial that sets the term $\mathsf{poly}\big((i-1)/\sigma, k, n^\delta\big)^{(i-1)\cdot\sigma}$ is independent of σ and δ, since it originates in the query reduction procedure (see Lemma 3.36). Thus, it is possible to define $i \leq O(1/\sigma)$ such that $\mathsf{poly}\big((i-1)/\sigma, k, n^\delta\big)^{(i-1)\cdot\sigma} \leq k \cdot n^{O(\delta)}$. Observing that $O((r_0 + (i-1)/\sigma) \cdot \log n) \leq n^{O(\delta)}$, since $i = O(1/\sigma)$, and that $(\mathcal{V}\mathsf{time}_0)^{\sigma^{i-1}} \leq n^{O(\delta)}$, the inequality follows.
- $a_{i-1} \geq \max\big((k \cdot S)^2 \cdot \mathsf{polylog}(n), k^2 \cdot n^{O(\delta)}, (k \cdot q_{i-1})^2 \cdot \mathsf{polylog}(n)\big)$:
  1. $a_{i-1} \geq (k \cdot S)^2 \cdot \mathsf{polylog}(n)$: Notice that $a_{i-1} \geq a_0 = S^2 \cdot \mathsf{poly}(k, |\mathbb{H}|)$, due to Proposition 5.12.
  2. $a_{i-1} \geq k^2 \cdot n^{O(\delta)}$: Once again, it follows from the fact that $a_{i-1} \geq a_0 = S^2 \cdot \mathsf{poly}(k, |\mathbb{H}|)$, and that $|\mathbb{H}| = n^\delta$.
  3. $a_{i-1} \geq (k \cdot q_{i-1})^2 \cdot \mathsf{polylog}(n)$: It holds that
  $$\begin{aligned} &a_{i-1} \geq \mathsf{poly}\big(1/\sigma, k, n^\delta, r_0, \mathcal{V}\mathsf{time}_0^{\sigma^{i-1}}\big) \geq \\ &\Big(k \cdot (\mathcal{V}\mathsf{time}_0)^{\sigma^{i-1}} \cdot \mathsf{poly}\big((i-1)/\sigma, k, n^\delta\big)^{(i-1)\cdot\sigma} \\ &+ O\left((r_0 + (i-1)/\sigma) \cdot \log n\right)\Big)^2 \cdot \mathsf{polylog}(n). \end{aligned}$$
- $r_{i-1} \cdot b_{i-1} \leq k \cdot n^{O(\delta)}$: Indeed, $r_0 + (i-1) \cdot O(1/\sigma) \cdot k \cdot n^{O(\delta)} \leq k \cdot n^{O(\delta)}$, using Proposition 5.12 (note that the constant in the $n^{O(\delta)}$ term grows, but only by a negligible additive factor).
- $|\mathbb{H}| \leq (k \cdot q_{i-1})^\sigma$: Follows directly from taking a sufficiently small constant multiple of σ (i.e., resetting $\sigma = \Theta(\delta^3)$).

By applying Lemma 5.10 with parameters as above and recalling that $t = k^i$, we conclude that the language $\mathcal{L}_{(k^i)}$ has an ε_i-sound $(q_i, r_i, a_i, b_i, \mathcal{P}\mathsf{time}_i, \mathcal{V}\mathsf{time}_i)$-PCIA w.r.t. encoded provers and with input-oblivious queries, and the following parameters.

1.
$$\begin{aligned}\varepsilon_i &= 1 - (1-\varepsilon_{i-1})/30\log k + O(1/\log n)\\ &= 1 - (1/30\log k)^i \cdot (1-\varepsilon_0) + O(i/\log n).\end{aligned}$$

2.
$$\begin{aligned}q_i &= (k \cdot n^\delta \cdot \mathcal{V}\mathsf{time}_{i-1})^\sigma + O\left((r_{i-1} + 1/\sigma)\cdot \log n\right)\\ &= \left(k \cdot n^\delta \cdot (\mathcal{V}\mathsf{time}_0)^{\sigma^{i-1}} \cdot \mathsf{poly}\big((i-1)/\sigma, k, n^\delta\big)^{\sum_{j=0}^{i-1}\sigma^j}\right)^\sigma\\ &+ O\left((r_0 + (i-1)/\sigma)\cdot \log n\right)\\ &\le (\mathcal{V}\mathsf{time}_0)^{\sigma^i} \cdot \mathsf{poly}\big(i/\sigma, k, n^\delta\big)^{i\cdot\sigma} + O\left((r_0 + i/\sigma)\cdot \log n\right).\end{aligned}$$

3.
$$r_i = r_{i-1} + O(1/\sigma) = r_0 + (i-1)\cdot O(1/\sigma) + O(1/\sigma) = r_0 + i\cdot O(1/\sigma).$$

4.
$$\begin{aligned}a_i &= \max\left(a_{i-1}, \mathsf{poly}\big(r_{i-1}, 1/\sigma, \max\left(b_{i-1}, k\cdot n^{O(\delta)}\right), k, n^\delta, \mathcal{V}\mathsf{time}_{i-1}\big)\right)\\ &= \max\left(a_0, \mathsf{poly}\big(1/\sigma, k, n^\delta, (\mathcal{V}\mathsf{time}_0)^{\sigma^i} \cdot \mathsf{poly}\Big(i/\sigma, k, n^\delta\Big)^{\sum_{j=0}^{i}\sigma^j}\big)\right)\\ &= \max\left(a_0, \mathsf{poly}\big(1/\sigma, k, n^\delta, r_0, \mathcal{V}\mathsf{time}_0^{\sigma^i}\big)\right).\end{aligned}$$

5.
$$b_i = k\cdot n^{O(\delta)}.$$

6.
$$\begin{aligned}\mathcal{P}\mathsf{time}_i &= \mathsf{poly}(k, n^\delta, k^i, S) + k\cdot\mathcal{P}\mathsf{time}_{i-1} + \mathsf{poly}\Big(k, n^\delta, r_{i-1}, 1/\sigma, \mathcal{V}\mathsf{time}_{i-1}, b_{i-1}\Big)\\ &\le \mathsf{poly}(n^\delta, k^i, S) + k\cdot\mathcal{P}\mathsf{time}_{i-1} + \mathsf{poly}\Big(k, n^\delta, r_0 + (i-1)\cdot O(1/\sigma), 1/\sigma,\\ &\quad (\mathcal{V}\mathsf{time}_0)^{\sigma^{(i-1)}} \cdot \mathsf{poly}\Big((i-1)/\sigma, k, n^\delta\Big)^{\sum_{j=0}^{i-1}\sigma^j}, k\cdot n^{O(\delta)}\Big)\\ &\le k^i\cdot\Big(\mathcal{P}\mathsf{time}_0 + i\cdot\mathsf{poly}\big(k, n^\delta, r_0, 1/\sigma, (\mathcal{V}\mathsf{time}_0)^\sigma\big)\Big) + i\cdot\mathsf{poly}(n^\delta, k^i, S).\end{aligned}$$

7.
$$\begin{aligned}\mathcal{V}\mathsf{time}_i &= (k\cdot n^\delta\cdot\mathcal{V}\mathsf{time}_{i-1})^\sigma + \mathsf{poly}\Big(r_{i-1}, 1/\sigma, k\cdot n^{O(\delta)}\Big)\\ &\le (k\cdot n^\delta\cdot\mathcal{V}\mathsf{time}_{i-1})^\sigma + \mathsf{poly}\Big(i/\sigma, k, n^\delta\Big)\\ &\le (\mathcal{V}\mathsf{time}_{i-1})^\sigma\cdot\mathsf{poly}\Big(i/\sigma, k, n^\delta\Big) \le (\mathcal{V}\mathsf{time}_0)^{\sigma^i}\cdot\mathsf{poly}\Big(i/\sigma, k, n^\delta\Big)^{1+i\cdot\sigma}.\end{aligned}$$

The prover's n^δ-low-depth property follows from Lemma 5.10: as mentioned there, the depth of the circuit computing the prover's messages only increases by a multiplicative $O(\log n)$ factor, thus the n^δ-low-depth property is preserved as long as $i = O(1)$, which indeed holds by the assumption. This completes the proof of Proposition 5.13. □

Let us get back to the proof of Theorem 5.11. Let $\delta \in (0,1)$ and $\sigma \stackrel{\text{def}}{=} \delta^3$, and recall that $|\mathbb{H}| = \Theta(n^\delta)$. Assume that $(\mathcal{V}\mathsf{time}_0)^{\sigma^{1/\delta}} = (\mathcal{V}\mathsf{time}_0)^{\delta^{3/\delta}} \leq |\mathbb{H}|^{\delta^2}$.[22] We obtain from Proposition 5.13 (notice that we take $i = 1/\delta$, so $i = O(1/\sigma)$ as required) that the language $\mathcal{L}_{(k^{(1/\delta)})} = \mathcal{L}_T$ has an $\varepsilon_{(1/\delta)}$-sound $(q_{(1/\delta)}, r_{(1/\delta)}, a_{(1/\delta)}, b_{(1/\delta)}, \mathcal{P}\mathsf{time}_{(1/\delta)}, \mathcal{V}\mathsf{time}_{(1/\delta)})$-PCIA w.r.t. encoded provers, with n^δ-low-depth honest prover and with input-oblivious queries, with the following parameters:

1.
$$\begin{aligned}\varepsilon_{1/\delta} &= 1 - (1/30\log k)^i \cdot (1-\varepsilon_0) + O(i/\log n)\\ &= 1 - (1/30\log k)^{1/\delta} \cdot (1 - O(|\mathbb{H}|/|\mathbb{F}|)) + O(1/(\delta\cdot\log n))\\ &\leq 1 - 1/\mathsf{polylog}(T).\end{aligned}$$

2.
$$\begin{aligned}q_{1/\delta} &= (\mathcal{V}\mathsf{time}_0)^{\sigma^{1/\delta}} \cdot \mathsf{poly}\big(1/(\delta\cdot\sigma), k, n^\delta\big)^{\sigma/\delta} + O\left((r_0 + 1/(\delta\cdot\sigma))\cdot\log n\right)\\ &\leq |\mathbb{H}|^{\delta^2} \cdot \mathsf{poly}\big(1/(\delta\cdot\sigma), k, n^\delta\big)^{\delta^2} + O\left(\log n/(\delta\cdot\sigma)\right) \leq O\left(T^{O(\delta^3)}\right).\end{aligned}$$

3.
$$r_{1/\delta} = r_0 + 1/\delta \cdot O(1/\sigma) = 1 + O\left(1/(\delta\cdot\sigma)\right) = O(1/\delta^4).$$

4.
$$\begin{aligned}a_{1/\delta} &= \max\left(a_0, \mathsf{poly}\big(1/\sigma, k, n^\delta, r_0, \mathcal{V}\mathsf{time}_0^{\sigma^{1/\delta}}\big)\right)\\ &\leq \max\left(S^2\cdot\mathsf{poly}(k,|\mathbb{H}|), \mathsf{poly}\big(1/\sigma, k, n^\delta, r_0, |\mathbb{H}|\big)\right) \leq S^2\cdot\mathsf{poly}(k, n^\delta)\\ &\leq O\left(T^{O(\delta)}\cdot S^2\right).\end{aligned}$$

5.
$$b_{1/\delta} = k\cdot n^{O(\delta)} = T^{O(\delta)}.$$

6.
$$\begin{aligned}\mathcal{P}\mathsf{time}_{1/\delta} &= k^{1/\delta}\cdot\left(\mathcal{P}\mathsf{time}_0 + 1/\delta\cdot\mathsf{poly}\big(k, n^\delta, r_0, 1/\sigma, (\mathcal{V}\mathsf{time}_0)^\sigma\big)\right)\\ &\quad + 1/\delta\cdot\mathsf{poly}(n^\delta, k^{1/\delta}, S)\\ &\leq T\cdot\left(\mathsf{poly}\left(k, S, n^\delta\right) + \mathsf{poly}\big(k, n^\delta, 1/\delta, |\mathbb{H}|\big)\right) + 1/\delta\cdot\mathsf{poly}(n^\delta, T, S)\\ &\leq T\cdot\mathsf{poly}\left(k, S\right) + \mathsf{poly}(T, S) \leq T^{1+O(\delta)}\cdot\mathsf{poly}(S) + \mathsf{poly}(T,S)\\ &= \mathsf{poly}(T,S).\end{aligned}$$

[22] Let ν such that $\mathcal{V}\mathsf{time}_0 = \mathsf{poly}(|\mathbb{H}|) \leq |\mathbb{H}|^\nu$. If δ does not satisfy the inequality $\delta^{3/\delta} \leq \delta^2/\nu$, we reset δ to be sufficiently small. Notice that ν is independent of δ (see Proposition 3.9, where $\mathcal{V}\mathsf{time}_0$ is set).

7.

$$\begin{aligned}\mathcal{V}\mathsf{time}_{1/\delta} &= (\mathcal{V}\mathsf{time}_0)^{\sigma^{1/\delta}} \cdot \mathsf{poly}\left(1/(\delta\cdot\sigma), k, n^{\delta}\right)^{1+\sigma/\delta} \leq |\mathbb{H}|^{\delta^2} \cdot \mathsf{poly}\left(k, n^{\delta}\right)^{1+\delta^2} \\ &\leq T^{O(\delta)}.\end{aligned}$$

Theorem 5.11 follows by transforming the foregoing PCIA for $\mathcal{L}_T$ into a PCIA for $\mathcal{L}$ as explained above (i.e., by emulating queries to the input $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x, w_{\mathsf{start}}, w_{\mathsf{end}})$ using Proposition 3.7). This transformation does not affect the n^{δ}-low-depth property of the prover.

To complete the proof, recall that we assumed that $\mathcal{M}$ is an *oblivious* Turing machine. In case it is not, we can trivially make it oblivious while increasing its time complexity to $T' = O(T \cdot S)$ and its space complexity to $S' = O(S)$ and the theorem follows. □

Applying Proposition 3.33 to the PCIA of Theorem 5.11 with error parameter $\xi = 1/\mathsf{polylog}(T)$, and then applying parallel repetition as per Theorem 3.21 with $s = 1 - 1/\mathsf{polylog}(T)$ and $\gamma = \mathsf{polylog}(T)$ to reduce the soundness error to be negligible, we obtain Theorem 5.1. The repetitions increase the communication complexity, the verifier's running time and the prover's running time by a $\mathsf{polylog}(T)$ factor. Notice that transforming the PCIA to an IA (Proposition 3.33) increases the verification time by an additive factor of

$$O\left(n^{\delta}\cdot\log n\cdot\log\left(T^{O(\delta)}/\mathsf{polylog}(T)\right)\right)\cdot O\left(T^{O(\delta^3)}\right) = O(n^{\delta})\cdot T^{O(\delta^2)} \leq O\left(T^{O(\delta)}\right),$$

and, in the non-holographic case, by another additive factor of $n^{1+O(\delta)}$ for computing a single coordinate in the low degree extension of the input x. Theorem 5.1 follows.

Acknowledgments

We are most grateful to Ron Rothblum for his brilliant suggestion to use a high-rate tensor code in the Batch Verification for UP protocol (Section 4).

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819702), from DARPA-TA1 under grant no. HR001119S0076, and from the Simons Foundation Collaboration on the Theory of Algorithmic Fairness.

References

[AR23] Noga Amit and Guy N. Rothblum. Constant-round arguments from one-way functions. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 1537–1544, New York, NY, USA, 2023. Association for Computing Machinery.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. Cryptology ePrint Archive, Report 2016/116, 2016. http://eprint.iacr.org/.

[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, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 21–31, 1991.

[BGG+88] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, pages 37–56, 1988.

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

[BIN97] Mihir Bellare, Russell Impagliazzo, and Moni Naor. Does parallel repetition lower the error in computationally sound protocols. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 374 – 383, 11 1997.

[BKP18] Nir Bitansky, Yael Tauman Kalai, and Omer Paneth. Multi-collision resistance: a paradigm for keyless hash functions. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 671–684. ACM, 2018.

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Snargs for P from LWE. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 68–79. IEEE, 2021.

[CP15] Kai-Min Chung and Rafael Pass. Tight parallel repetition theorems for public-coin arguments using KL-divergence. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 229–246. Springer, 2015.

[Dam89] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989.

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.*, 67(4):205–214, 1998.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[Gol08] Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008.

[GR15] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 133–142, 2015.

[GR17] Tom Gur and Ron D. Rothblum. A hierarchy theorem for interactive proofs of proximity. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 39:1–39:43, 2017.

[GR20] Oded Goldreich and Guy N. Rothblum. Constant-round interactive proof systems for AC0[2] and NC1. In Oded Goldreich, editor, *Computational Complexity and Property Testing - On the Interplay Between Randomness and Computation*, volume 12050 of *Lecture Notes in Computer Science*, pages 326–351. Springer, 2020.

[GS92] Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Inf. Process. Lett.*, 43(4):169–174, 1992.

[GS06] Oded Goldreich and Madhu Sudan. Locally testable codes and PCPs of almost-linear length. *J. ACM*, 53(4):558–655, 2006.

[GVW02a] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Comput. Complex.*, 11(1-2):1–53, 2002.

[GVW02b] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, 11(1-2):1–53, 2002.

[Hai09] Iftach Haitner. A parallel repetition theorem for any interactive argument. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 241–250, 2009.

[Ish20a] Yuval Ishai. Zero-knowledge proofs from information-theoretic proof systems - part i. Available at https://zkproof.org/2020/08/12/information-theoretic-proof-systems/, 2020.

[Ish20b] Yuval Ishai. Zero-knowledge proofs from information-theoretic proof systems - part ii. Available at https://zkproof.org/2020/10/15/information-theoretic-proof-systems-part-ii/, 2020.

[IY87] Russell Impagliazzo and Moti Yung. Direct minimum-knowledge computations. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 1987.

[Jus72] Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Trans. Inf. Theory*, 18:652–656, 1972.

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

[KK05] Jonathan Katz and Chiu-Yuen Koo. On constructing universal one-way hash functions from arbitrary one-way functions, 2005. jkatz@cs.umd.edu 13051 received 16 Sep 2005, last revised 24 Sep 2005.

[KNY18] Ilan Komargodski, Moni Naor, and Eylon Yogev. Collision resistant hashing for paranoids: Dealing with multiple collisions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 162–194. Springer, 2018.

[KRR22] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: The power of no-signaling proofs. *J. ACM*, 69(1):1:1–1:82, 2022.

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

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

[Mer89] Ralph C. Merkle. One way hash functions and DES. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1989.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *FOCS*, pages 436–453, 1994.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 33–43. ACM, 1989.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 387–394. ACM, 1990.

[RR19] Noga Ron-Zewi and Ron Rothblum. Local proofs approaching the witness length. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:127, 2019.

[RR20] Guy N. Rothblum and Ron D. Rothblum. Batch verification and proofs of proximity with polylog overhead. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 108–138. Springer, 2020.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 49–62, 2016.

[RRR18] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Efficient batch verification for UP. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, pages 22:1–22:23, 2018.

[RV22] Ron D. Rothblum and Prashant Nalini Vasudevan. Collision-resistance from multi-collision-resistance. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part III*, volume 13509 of *Lecture Notes in Computer Science*, pages 503–529. Springer, 2022.

[RVW13] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *STOC*, pages 793–802, 2013.

[Sim98] Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, volume 1403 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 1998.

[Sud95] Madhu Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*, volume 1001 of *Lecture Notes in Computer Science*. Springer, 1995.

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. Available at https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html, 2022.

[Wol65] Jack K. Wolf. On codes derivable from the tensor product of check matrices. *IEEE Trans. Information Theory*, 11(2):281–284, 1965.

A Code Switching for Tensor Codes: Definitions and Proofs

In this appendix, for the sake of completeness, we provide the details of the encoding $\mathcal{R}$ and the Code-Switching Lemma from Section 3.5, which are taken almost verbatim from [RR19]. We always consider a constructible field ensemble $\mathbb{F} = (\mathbb{F}_n)_n$ that extends $\mathbb{GF}[2]$, and let $\gamma \in (0, 1)$ be a rate parameter, which we think of as an arbitrary constant.

A.1 The Encoding

We describe the structure and properties of the encoding defined in Theorem 3.26, that we denote by $\mathcal{R}$. Whereas we do not prove its existence, since it is already proved in RR's work, we briefly discuss its structure and show that it satisfies one additional property.

Theorem 2.3 in RR's work proves the existence of a systematic binary linear code, that they denote as R (not to be confused with our $\mathcal{R}$), with messages of length $M^{1/2t}$ and rate $1 - \frac{\gamma}{2t}$. Then,

they consider its second-tensor, denoted $R^{\otimes 2} : \{0,1\}^{M^{1/t}} \to \{0,1\}^{\gamma \cdot M^{1/t}}$ where $\gamma = (1 - \frac{\gamma}{2t})^{-1}$. They take $R^{\otimes 2}$ to be the code C used in the Sumcheck for tensor codes, Lemma 7.1 there and Lemma 3.28 here (see Section A.2). Hence, the code on which the Sumcheck for tensor code applies is the t-wise tensor product of $R^{\otimes 2}$, which is also the $(2t)$-wise tensor product of R, is $R^{\otimes 2t}$. This is the encoding that we use throughout this work, i.e., $\mathcal{R} \stackrel{\text{def}}{=} R^{\otimes 2t}$. Its relative distance is $\left(\frac{\gamma}{t}\right)^{O(t)}$. From the properties of R, it follows that $\mathcal{R}$ is a binary linear code, with message length M, that is also locally-testable. The last property stated in Theorem 3.26 is that, for a constant t, $\mathcal{R}$ is encodable in NC. Since RR do not prove this, we do so in the following proposition. We assume familiarity only with the Reed-Solomon encoding.

Proposition A.1 ($\mathcal{R}$ encodable in NC)**.** *Let $M = M(n)$ and $t \in \mathbb{N}$ be a constant. The encoding $\mathcal{R} \stackrel{def}{=} R^{\otimes 2t}$ as defined above w.r.t. to message length M is encodable in* Logspace*-uniform* NC*.*

Proof. The construction of $\mathcal{R}$ is a high-rate variant of Justesen's code [Jus72], and its full construction can be found in [RR19, Appendix A]. To prove that it is also encodable in Logspace-uniform NC, we first define the code R mentioned above.

The encoding R is obtained by encoding the symbols of a Reed-Solomon code with the Wozencraft Ensemble. Assume R operates on messages of length k (eventually, we take $k = M^{1/2t}$). Given a string $x \in \{0,1\}^k$, we view any consecutive $\log |\mathbb{F}|$ coordinates of x as a field element in the natural way, and apply a Reed-Solomon code with message length $k/\log |\mathbb{F}|$ and codeword length $n' = (k/\log |\mathbb{F}|) \cdot (1 + \gamma/3)$. Let y denote this encoded string, so $y \in \mathbb{F}^{n'}$. For any $i \in [n']$, the i^{th} coordinate of y, denoted y_i, is encoded by the Wozencraft Ensemble in the following manner: $y_i \mapsto (y_i, \sigma(\alpha_i \cdot y_i))$, where α_i is the i^{th} element of $\mathbb{F}$, and $\sigma : \mathbb{F} \to \{0,1\}^s$ is the first $s = \lfloor \log k \cdot \frac{\gamma}{3} \rfloor$ bits in the binary representation of its input.

Observe that σ is computable in depth $O(\log |\mathbb{F}|)$, and that computing the Reed-Solomon w.r.t. message length $k/\log |\mathbb{F}|$ can be done in depth $\mathsf{polylog}(k, |\mathbb{F}|)$ by exponentiation via repeated squaring. Since k and $|\mathbb{F}|$ are always polynomial in n, the circuit computing R is an NC circuit. Logspace-uniformity follows by construction.

Since any application of a tensor product raises the depth of the circuit by a multiplicative factor of at most 2, the $2t$-tensor of R, namely, $\mathcal{R}$, is encodable by a $2^{2t} \cdot \mathsf{polylog}(n)$-depth circuit. Recalling that t is a constant, the proposition follows. □

A.2 Proof of the Code-Switching Lemma

Proof. We show how to derive Lemma 3.28, to which we refer as the "Code-Switching $\mathsf{HIP}_{\mathcal{R}}$", from Lemma 7.1 of [RR19], called a "Sumcheck for tensor codes".

Let us first briefly recall the setup and the goal of the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$. The holographic input is a string $w \in \{0,1\}^M$, and the explicit input is a claim (r, v) about $\mathsf{LDE}(w)$ that $\mathsf{LDE}(w)[r] = v$. The output is a set of claims $(\chi_j, \theta_j)_{j \in [q]}$ about a *different* encoding, defined in Theorem 3.26 (see also Section A.1) and denoted as $\mathcal{R}$. The claims are that $\forall j \in [q], \mathcal{R}(w)[\chi_j] = \theta_j$. The $\mathsf{HIP}_{\mathcal{R}}$ reduces the input claim to the output claims, such that if it is correct, then all claims will be correct, whereas if it is false then at least one of the output claims will also be false.

The Sumcheck for tensor codes is a generalization of the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$ in two aspects. First, the input can be any linear claim (as long as its coefficients satisfy a "rank 1 tensor" property, see Definition A.2 below), and not necessarily a claim about the low degree extension of the input. The second aspect is analogous: the encoding used for the output claims can be more general than

the specific choice of $\mathcal{R}$. In their work, [RR19] state the protocol for families of codes that satisfy some requirements, and then prove that $\mathcal{R}$ satisfies these.

We comment that the Sumcheck for tensor codes approach is inspired by Meir's [Mei13] combinatorial proof of the $\mathsf{IP} = \mathsf{PSPACE}$ theorem. However, we do not focus on this protocol, and refer the reader to the work of Ron-Zewi and Rothblum [RR19] for the full details.

For the proof of the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$, we start with defining the property that is required from the input claim. Then, we show that the low degree extension satisfies it. Next, we set the parameters of the Sumcheck for tensor codes such they work with the Code-Switching $\mathsf{HIP}_{\mathcal{R}}$ setting. Lastly, we mention a single change needed for our alternative statement.

To define a "rank 1 tensor", we introduce some notation. For an integer t and a sequence of t vectors $\lambda_1, \ldots, \lambda_t \in \mathbb{F}^{M^{1/t}}$, we let $\lambda_1 \otimes \ldots \otimes \lambda_t \in \mathbb{F}^M$ denote the vector that satisfies

$$\forall i_1, i_2, \ldots, i_t \in [M^{1/t}],\ (\lambda_1 \otimes \ldots \otimes \lambda_t)_{i_1, i_2, \ldots, i_t} = (\lambda_1)_{i_1} \cdot (\lambda_2)_{i_2} \cdot \ldots \cdot (\lambda_t)_{i_t}.$$

Notice that, here and in what follows, we take $\lambda_i = \lambda[i]$, i.e., the i^{th} coordinate of the vector λ.

Definition A.2 (Rank 1 tensors of dimension t)**.** *Let $\mathbb{F} = (\mathbb{F}_n)_n$ be a constructible field ensemble that extends $\mathbb{GF}[2]$. A vector $\lambda \in \mathbb{F}^M$ is a* Rank 1 Tensor of Dimension t *if there exist t vectors $\lambda_1, \ldots, \lambda_t \in \mathbb{F}^{M^{1/t}}$ such that $\lambda = \lambda_1 \otimes \ldots \otimes \lambda_t$.*

For example, for $t = 2$, consider an $\sqrt{M} \times \sqrt{M}$ matrix of rank 1. Taking λ to be the vector in $\mathbb{F}^M$ that represents the matrix (in the natural way), we get that λ satisfies the definition.

Proposition A.3. *Let $\mathbb{H} \subseteq \mathbb{F}$ be (ensembles of) extension fields of $\mathbb{GF}[2]$, such that $\mathbb{H} \subseteq \mathbb{F}$. Let $\mathsf{LDE} = \mathsf{LDE}_{\mathbb{F},\mathbb{H}}$ be the low degree extension encoding. Let $w \in \{0,1\}^M$ be an input and define $m = \log_{|\mathbb{H}|}(M)$. Then, for any integer t that divides m, and any claim $(r, v) \in \mathbb{F}^m \times \mathbb{F}$, there exists a sequence $\lambda_1, \ldots, \lambda_t \in \mathbb{F}^{M^{1/t}}$, such that*

$$\mathsf{LDE}(w)[r] = v \iff \langle \lambda_1 \otimes \ldots \otimes \lambda_t, w \rangle = v.$$

Furthermore, each λ_i for $i \in [t]$ is computable in time $\mathsf{poly}(|\mathbb{H}|, m, \log|\mathbb{F}|)$.

Proof. First, we define $\lambda \in \mathbb{F}^M$ such that $\mathsf{LDE}(w)[r] = v \iff \langle \lambda, w \rangle = v$. Then, we show that λ is a rank 1 tensor of dimension t (i.e., satisfies Definition A.2). Recall the individual degree $|\mathbb{H}| - 1$ polynomial $\hat{\tau} : \mathbb{F}^m \times \mathbb{F}^m \to \mathbb{F}$ defined in Eq. (1):

$$\hat{\tau}(x, r) \stackrel{\text{def}}{=} \prod_{i \in [m]} \prod_{h \in \mathbb{H} \setminus \{0\}} \frac{r_i - x_i - h}{h},$$

and the definition of the low degree extension of w at coordinate r:

$$\mathsf{LDE}(w)[r] = \sum_{x \in \mathbb{H}^m} \hat{\tau}(x, r) \cdot w[x],$$

where $w[x]$ is the x^{th} coordinate of $w \in \mathbb{H}^m$. We define $\lambda \in |\mathbb{F}|^M$ as the vector of these coefficients:

$$\forall x \in |\mathbb{H}|^m,\ \lambda[x] \stackrel{\text{def}}{=} \hat{\tau}(x, r),$$

where recall that we identify $|\mathbb{H}|^m$ with M.

The fact that λ has a rank 1 tensor of dimension m structure follows from the fact that the low degree extension is itself a tensor code. In particular, define the binomial $\hat{I} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ as

$$\hat{I}(x, x') \stackrel{\text{def}}{=} \prod_{h \in \mathbb{H} \setminus \{0\}} \frac{x' - x - h}{h}.$$

For every $i \in [m]$, we fix r_i and define $\lambda_i \in \mathbb{F}^{|\mathbb{H}|}$ such that

$$\forall x \in |\mathbb{H}|^m, \ \lambda_i[x] \stackrel{\text{def}}{=} \hat{I}(x, r_i).$$

Notice that $\forall i_1, i_2, \ldots, i_m \in [|\mathbb{H}|]$:

$$\begin{aligned}
&(\lambda_1 \otimes \ldots \otimes \lambda_m)_{i_1, i_2, \ldots, i_m} \\
&= \hat{I}(i_1, r_1) \cdot \hat{I}(i_2, r_2) \cdot \ldots \cdot \hat{I}(i_m, r_m) \\
&= \prod_{h \in \mathbb{H} \setminus \{0\}} \frac{r_1 - i_1 - h}{h} \cdot \prod_{h \in \mathbb{H} \setminus \{0\}} \frac{r_2 - i_2 - h}{h} \cdot \ldots \cdot \prod_{h \in \mathbb{H} \setminus \{0\}} \frac{r_m - i_m - h}{h} \\
&= \prod_{j \in [m]} \prod_{h \in \mathbb{H} \setminus \{0\}} \frac{r_j - i_j - h}{h} \\
&= \hat{\tau}\big((i_1, i_2, \ldots, i_m), (r_1, r_2, \ldots, r_m)\big) = (\lambda)_{i_1, i_2, \ldots, i_m}.
\end{aligned}$$

Hence, $\lambda = \lambda_1 \otimes \ldots \otimes \lambda_m$ as required. To conclude, note that $\lambda_1, \ldots, \lambda_m$ are all computable in time $\mathsf{poly}(|\mathbb{H}|, m, \log|\mathbb{F}|)$ by Proposition 3.6. For any t that divides m, finding the sequence $\lambda_1, \ldots, \lambda_t$ is possible (within this time bound) by [RR19, Fact 6.5]. □

Our next goal is setting the parameters for the Sumcheck for tensor codes.

- We take $C^{\otimes t}$ to be $\mathcal{R}$ as defined in Theorem 3.26. In Section 7.1 of their work, they prove that $\mathcal{R}$ satisfies the requirements of the Sumcheck lemma. Recall that the relative distance of $\mathcal{R}$ is $\left(\frac{\gamma}{t}\right)^{O(t)} = O\left(\gamma \cdot \delta\right)^{O(1/\delta)}$.
- We set $t = m$, where we recall that $m = \log_{|\mathbb{H}|}(M) = O(1/\delta)$. This means that the round complexity is $O(1/\delta)$, and that $M^{1/m} = |\mathbb{H}|$.
- Taking $\varepsilon = 1/n$, we get that

  $$d = O\left(\frac{\log(t/\varepsilon)}{(\gamma \cdot \delta)^{O(1/\delta)}}\right) = O(\log n),$$

  where the constant depends on γ and δ. To simplify notation, we define $q \stackrel{\text{def}}{=} d^t = d^{O(1/\delta)}$. Indeed, it holds that $q = \mathsf{polylog}(n)$, although the constant is of order $\mathsf{exp}(1/\delta)$ (notice that q does *not* affect the round complexity).
- By Proposition A.3, the validity of the input claim (r, v) is reduced to the validity of the claim that $\langle \lambda_1 \otimes \ldots \otimes \lambda_t, w \rangle = v$.

Finally, the (minor) change that we make in the Sumcheck for tensor codes is restating it as a $\mathsf{HIP}_{\mathcal{R}}$ instead of an IP. We do so by making the verifier output query locations and values that would make it accept had it actually queried the encoded input at these locations. It does make queries to the transcript, and its decision predicate ϕ, given at the form of q claims, only contains queries to the input. Lemma 3.28 follows. □

B Additional Proofs

B.1 Proof of Proposition 3.25.

Let $n \in \mathbb{N}$ and $M = M(n)$. Let $\mathbb{F}$ be an extension field of $\mathbb{GF}[2]$ where $|\mathbb{F}| = \Theta(n^{2\delta})$ for $\delta \in (0,1)$, and let $\mathcal{L}$ be a pair language with inputs of the form $(x,w) \in \{0,1\}^n \times \{0,1\}^M$. Assume that $\mathcal{L}$ is computable by Logspace-uniform circuits with fan-in 2, of depth $D = D(n,M)$ and size $\mathsf{poly}(n,M)$.

We take $(\mathcal{P},\mathcal{V})$ to denote the flat-GKR protocol (see Theorem 3.23), ρ to denote its number of rounds, and $\beta_1,\ldots,\beta_\rho$ to denote $\mathcal{V}$'s messages. In this section, we prove that $\mathcal{P}$'s messages are computable in low-depth: namely, that there exists a Logspace-uniform arithmetic circuits over $\mathbb{F}$ with fan-in 2, of depth $\mathsf{poly}(D, n^\delta)$ and size $\mathsf{poly}(n,M)$, whose input is $((x,w),\beta_1,\ldots,\beta_\rho)$ and output is a sequence of $\mathcal{P}$'s messages.

Proof. We assume familiarity with the flat-GKR protocol from [AR23, Theorem 5.1] and follow the prover's steps. Note that the encoding in use is always the low degree extension encoding (see Section 3.1), and that by the choice of $\mathbb{F}$ and $\mathbb{H}$, encoding a string incurs a quadratic blowup in its length. From now on, when we use the term polynomial we mean $\mathsf{poly}(n,M)$.

- Computing the circuit $C(x,w)$. This step (trivially) takes depth D and polynomial size.
- Computing the hash tree of each of the circuit's layers. Each of these computations can be preformed "in parallel". The depth of a hash tree (computed on any polynomial-length string) is $\mathsf{poly}(\kappa)$ for a security parameter $\kappa = n^\delta$ (recall that the tree has a constant number of layers, namely, $O(1/\delta)$). Its size is at most twice the number of leaves, which is at most polynomial, multiplied by the size of the circuit computing the hash functions, which is $\mathsf{poly}(\kappa) \geq \mathsf{polylog}(n)$.
- Computing answers to the protocol preformed in Step $4(a)$. Roughly speaking, the protocol executed in this step, referred to as the "HHR" protocol, is a HIP that reduces the validity of a hash root to a claim about the encoding of the string being committed to. First, notice that computing the complete hash tree, as we already mentioned, can be done by a $\mathsf{poly}(\kappa)$-depth circuit of polynomial size. The HHR protocol operates sequentially, by reducing claims about the encoding of a *tree layer* to a claim about the encoding of the layer below it, until it is left with a claim about the leaves, which are the string that the prover commits to. Since the tree is of constant depth, we focus on computing the answers to this sub-protocol (for its full details, see [AR23, Lemma 4.6]). There, the prover's computations[23] boil down to:
  - Computing the encoding of strings of length at most polynomial (in particular, the low degree extension encoding, see Section 3.1), which can be done by an NC circuit. We comment that in one of the sub-protocols used there, they view the low degree extension encoding as the m^{th}-tensor of a Reed-Solomon code (more details about these can be found above in Appendix A). Still, since AR's work always uses a constant m, the prover's answers are computable in Logspace-uniform NC.
  - Computing arithmetic operations over $\mathbb{F}$, all are computable in Logspace-uniform NC.

  All of the above computations can be preformed in parallel.

[23] In particular, these are dominated by computing the answers to the protocols of Theorems 3.25 and 3.26 of that work, which are variant of the [GR20] and [RVW13] protocols, respectively.

- Computing answers to the protocol preformed in Step 4(b). The protocol executed in this step is [AR23, Lemma 5.2], that preforms a "single step" of the [GKR08] protocol. In high-level, the GKR protocol uses a sub-protocol that checks the consistency of two consecutive circuit-layers. It reduces a claim about the encoding of one layer to claim about the encoding of the layer that feeds it. In flat-GKR, they extract this reduction, albeit with a different parameter setting such that it runs in a constant number of rounds. Once again, the computations of the prover boil down to computing the low degree extension of strings of polynomial length, and preforming arithmetic operations over $\mathbb{F}$. All of these can be done in parallel, thus in Logspace-uniform NC and the proposition follows.

□

B.2 Proof of Proposition 5.5.

Recall that $|Q| = q$. We construct an arithmetic circuit C over $\mathbb{F}$, whose input is a string x, a tableau $\mathcal{T}$, a sequence of claims about the prover's messages ϕ, a query set Q, and a full sequence of $\mathcal{V}_t$'s random coins $\beta_1, \ldots \beta_r$. Its output is a vector $v \in \mathbb{F}^q$, such that $v = 1^q$ if and only if ϕ is correct with respect to the prescribed strategy $\mathcal{P}_t$. We show that C satisfies all the desired properties stated above, while recalling that the flat-GKR protocol extends for vector-valued arithmetic circuits as well (see Remark 3.24).

Take $C_{\mathcal{P}_t}$ to be the $n^{O(\delta)}$-depth (arithmetic) circuit promised by the fact that the prescribed prover is n^δ-low-depth (see Definition 3.35). The circuit C works as follows: First, it feeds $C_{\mathcal{P}_t}$ with $(x, \mathcal{T}, \beta_1, \ldots \beta_r)$. Denote $C_{\mathcal{P}_t}$'s output, that is, the prescribed prover's messages, by $(\alpha_1, \ldots, \alpha_r)$. In order to compare the values in $(\alpha_1, \ldots, \alpha_r)$ at locations Q with the values in ϕ, the circuit C has $|Q| = q$ addition gates located in parallel (i.e., in the same layer), such that the $(j,t)^{\text{th}}$ gate, for $(j,t) \in Q$, is wired to $\phi(j,t)$ and $\alpha^j[t]$ using multiple copies of the circuit C_{select} described next. Notice that since addition and subtraction are the same over extension fields of $\mathbb{GF}[2]$, each of these addition gates outputs 0 if and only if the two values are equal.

In order to find $\alpha^j[t]$, we construct the arithmetic circuit C_{select}, that gets a sequence of strings and an index (represented in binary), and outputs the appropriate string. Then, we will use it twice: once, with the index j, to output α_j, and then, with the index t, to output $\alpha_j[t]$. Recall that each query $(j,t) \in [r] \times [a]$ is represented by a binary string of length $(\log r + \log a)$.

We define the following circuit C_{select} that "selects" z_i, given $z_1, ..., z_k$ and an index i. The circuit is arithmetic, and we identify $0_{\mathbb{F}}, 1_{\mathbb{F}}$ with $\mathbb{GF}[2] = \{0,1\}$ (and refer to them as "bits"). As a first step we allow its fan-in to be > 2.

- Its input is $i \in \{0,1\}^{\log k}$ and $z_1, ..., z_k \in \mathbb{F}^{|z_1| \cdot k}$.
- Its first layer compares between i and all of the elements in $[k]$ (namely, the k binary strings of length $\log k$ that represent $1, \ldots, k$), using the following identity for two bits b, b': $(b = b') \equiv ((1-b) + b') \cdot (b + (1-b'))$.
- The second layer shrinks each of the k results to a single bit, by k multiplication gates of fan-in $\log k$. This bit equals 1 if and only if the two strings are equal. This implies that, at this point, only the i^{th} bit is 1, and the rest are 0.
- The third layer uses $k \cdot |z_1|$ multiplication gates of fan-in 2, that operate on the results of the third layer (each one has fan-out $|z_1|$) and the inputs $z_1, ..., z_k$. If we interpret this layer as

k sequences of length $|z_1|$, we get that only the i^{th} sequence is the string z_i, and the rest of them are $0^{|z_1|}$.

- The forth (output) layer is the sum of these k results, i.e., $|z_1|$ addition gates of fan-in k. Since a sum of a field element and a zero returns the bit, we get that the circuit's output is exactly the string z_i.

By the construction, it is evident that the circuit C_{select} is an $\#\mathsf{AC}^0_{\mathbb{F},f_{in}}$ circuit of depth 4, where $f_{in} \leq k$. Since we want to reduce its fan-in to 2, we use the standard transformation that replaces any gate with fan-in k with k gates, wired in the form of a tree. This increases the depth of the circuit by a $\log k$ factor, while increasing the size by at most a $O(k)$ factor.

We use C_{select} as described above. First, the circuit C locates "in parallel" q copies of C_{select}, such that for $i \in [q]$, the i^{th} copy gets as input the sequence of messages $(\alpha_1, \ldots, \alpha_r)$ and the first index of the i^{th} query in S. Taking (j_i, t_i) to denote this query, the output of the i^{th} copy is α^{j_i}. Next, we locate another sequence of q parallel copies of C_{select}, where the i^{th} copy gets as input α^{j_i} (here we consider $|z_1| = 1$) and the second index of the i^{th} query in S. We get that the output of the i^{th} copy is $\alpha^{j_i}[t_i]$. Notice that the depth of these layers is $\log r + \log a = O(\log n)$.

The output of C is the sequence of these q results, when we add the constant 1 to each of them. This means that C outputs 1^q iff ψ is correct. The fact that the circuit is $\mathsf{Logspace}$-uniform, fan-in 2 and of polynomial size follows directly from its description. Notice that the depth of the circuit is bounded by the sum of $C_{\mathcal{P}_t}$'s depth, the $\log k$ factor and $O(\log n)$, and the proposition follows.

B.3 Proof of Proposition 5.6.

Let $t = t(n)$ be a time bound and $x \in \{0,1\}^n$ be an input. Let $u, v \in \{0,1\}^{O(S)}$ be two configurations of the machine $\mathcal{M}$, and let $\mathcal{T}$ denote the tableau of the computation starting at u and running for t steps. Let (ω, θ) be a claim about $\mathsf{LDE}(\mathcal{T})$.

We prove that there exists a $\mathsf{Logspace}$-uniform fan-in 2, depth $n^{O(\delta)}$ and polynomial size arithmetic circuit C whose input is $x, \mathcal{T}$ and a full sequence of $\mathcal{V}_{\mathsf{batch}}$'s random coins $\beta_1^{\mathsf{B}}, \ldots, \beta_{r_{\mathsf{B}}}^{\mathsf{B}}$, and output is the sequence of message sent by $\mathcal{P}_{\mathsf{batch}}$. We carefully go over each step preformed by $\mathcal{P}_{\mathsf{batch}}$ during the batching protocol, and prove that it can be preformed by a low-depth circuit.

- **Step** (1): Since C gets the tableau as input, finding $\mathsf{LDE}(w_t, w_{2t}, \ldots, w_{(k-1)\cdot t})$ boils down to computing the low-degree extension, that can be performed by a $\mathsf{Logspace}$-uniform arithmetic AC^0 circuit (see, e.g., [AR23, Section 4.4.1]), and in particular by a $\mathsf{Logspace}$-uniform NC^1 arithmetic circuit.

- **Step** (2): When decomposing the claim (ω, θ) into k claims, the circuit C computes "in parallel" k values in the low-degree extensions of the tableaux $(\mathcal{T}_i)_{i\in[k]}$ for the k sub-computations. Following the previous item, this can be done by a $\mathsf{Logspace}$-uniform NC^1 arithmetic circuit as well.

- **Step** (4): Computing the hash roots $(y_i)_{i\in[k]}$ can be done by a $\mathsf{Logspace}$-uniform Boolean circuit with fan-in 2, of depth $\ell \cdot \mathsf{poly}(n_{in}) = n^{O(\delta)}$ and size $\mathsf{poly}(n)$, using Proposition 3.19. We can convert it to an arithmetic circuit over $\mathbb{F}$ while increasing its size and depth of the circuit by at most a constant factor.

- **Step** (6a): Applying Proposition 3.25 on the circuit mentioned in the previous item yields that C can compute $\mathcal{P}_{\mathsf{batch}}$ messages in the flat-GKR execution by a circuit that satisfies all the desired properties.

- **Step** (6c): Computing $\mathcal{P}_t$'s messages (the prover of the base protocol) can be done by a Logspace-uniform fan-in 2, depth $n^{O(\delta)}$ and polynomial size arithmetic circuit using the assumption that $\mathcal{P}_t$ is a n^{δ}-low-depth prover. Computing the checksum can be done by a Logspace-uniform arithmetic AC^0 circuit by locating in parallel d addition gates, thus by a Logspace-uniform NC^1 arithmetic circuit as well.

- **Step** (8): Here, C computes the locations to outputs from the computed answers to the base protocols, according to the query set Q (notice that the verifier's messages $\beta_1^{\mathsf{B}}, \ldots, \beta_{r_{\mathsf{B}}}^{\mathsf{B}}$ include the query set Q, not to be confused with the sequence of verifier messages in the base protocol $\beta_1, \ldots, \beta_r$). All of these can be preformed in parallel. Proposition 5.5 shows that this task is computable by a circuit of depth $\log k$ that satisfies the desired properties.

- **Step** (10a): Recall we assumed that the prover $\mathcal{P}_t$ of the base protocol $(\mathcal{P}_t, \mathcal{V}_t)$ is n^{δ}-low-depth. This means that Proposition 5.5 applies, and the circuit on which we run the flat-GKR protocol in this step is Logspace-uniform arithmetic circuit with fan-in 2, of depth $n^{O(\delta)}$ and size $\mathsf{poly}(n)$. Thus, once again we can use Proposition 3.25 and get that C can compute $\mathcal{P}_{\mathsf{batch}}$ messages in the flat-GKR execution by a circuit that satisfies all the desired properties.

- **Step** (10b): Since the openings are a sub-circuit of the full circuit computing the Merkle tree, this step is included in Step (4), where $\mathcal{P}_{\mathsf{batch}}$ computes the hash root.

With the exception of Step (8), that requires the output of Step (6c), all of these computations can be performed *in parallel*: in computing its answers to one step, the prover does not use its answers from a previous step. This means that, compared to the circuit computing $\mathcal{P}_t$'s messages, the depth of the circuit computing $\mathcal{P}_{\mathsf{batch}}$'s messages is only bigger by a multiplicative factor of $\log k = O(\log n)$, and thus remains in $n^{O(\delta)}$-depth. The proposition follows by taking C to be the composition of all of the sub-circuits described above.

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il