Locally Testable Codes and PCPs of Almost-Linear Length*

Oded Goldreich†
Department of Computer Science
Weizmann Institute of Science
Rehovot, Israel.
oded.goldreich@weizmann.ac.il

Madhu Sudan‡
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139.
madhu@mit.edu

February 2, 2006

Abstract

We initiate a systematic study of locally testable codes; that is, error-correcting codes that admit very efficient membership tests. Specifically, these are codes accompanied with tests that make a constant number of (random) queries into any given word and reject non-codewords with probability proportional to their distance from the code.

Locally testable codes are believed to be the combinatorial core of PCPs. However, the relation is less immediate than commonly believed. Nevertheless, we show that certain PCP systems can be modified to yield locally testable codes. On the other hand, we adapt techniques that we develop for the construction of the latter to yield new PCPs. Our main results are locally testable codes and PCPs of almost-linear length. Specifically, we prove the existence of the following constructs:

- Locally testable binary (linear) codes in which k information bits are encoded by a codeword of length $k \cdot \exp(\tilde{O}(\sqrt{\log k}))$. This improves over previous results that either yield codewords of exponential length or obtained almost quadratic length codewords for sufficiently large non-binary alphabet.
- PCP systems of almost-linear length for SAT. The length of the proof is $n \cdot \exp(\tilde{O}(\sqrt{\log n}))$ and verification in performed by a constant number (i.e., 19) of queries, as opposed to previous results that used proof length $n^{1+O(1/q)}$ for verification by q queries.

The novel techniques in use include a random projection of certain codewords and PCP-oracles that preserves local-testability, an adaptation of PCP constructions to obtain "linear PCP-oracles" for proving conjunctions of linear conditions, and design of PCPs with some new soundness properties.

Keywords: Error-Correcting Codes, PCP, randomized reductions, low-degree tests, codeword tests, the Probabilistic Method

*An extended abstract of this work has appeared in the *43rd FOCS*, 2002. A preliminary full version [24] has appeared on ECCC.

†Supported by the MINERVA Foundation, Germany.

‡Supported in part by NSF Awards CCR 9875511, CCR 9912342, and MIT-NTT Award MIT 2001-04.

Contents

1 Introduction

Locally testable codes are (good) error-correcting codes that admit very efficient codeword tests. Specifically, the testing procedure makes only a *constant number* of (random) queries, and should reject non-codewords with probability proportional to their distance from the code.

Locally testable codes are related to Probabilistically Checkable Proofs (PCPs, cf. [2, 3, 5, 19]) and to Property Testing (cf. [22, 32]). Specifically, locally testable codes can be thought of as a combinatorial counterparts of the complexity theoretic notion of PCPs, and in fact the use of codes with related features is implicit in known PCP constructions. Local testability of codes is also a special case of property testing, and indeed the first collection of properties that were shown to be testable also yield constructions of locally testable codes [14].

Locally testable codes were introduced in passing, by Friedl and Sudan [21] and Rubinfeld and Sudan [32]. However, despite the central role of locally testable codes in complexity theoretic and algorithmic research, they have received little explicit attention so far. The primary goal of this work is to initiate a systematic study of locally testable codes. In particular, we focus on the construction of locally testable codes over a binary alphabet and on the development of techniques to reduce the alphabet size of locally testable codes. Studying the length of locally testable codes, we obtain for the first time (even for non-binary alphabets), codes of almost-linear length.

Some well-known examples: To motivate some of the parameters of concern, we start by considering some "trivial codes" that are easily testable. For example, the "code" that contains all strings of a given length is trivially testable (by accepting each string without even looking at it). It is also easy to test the "code" that consists of a single codeword (e.g., given an arbitrary string w, pick a random index i and verify that w and the single codeword agree at the the i-th coordinate). Thus, the concept of locally testable codes is interesting mainly in the case of "good" codes; that is, codes that have "many" codewords that are pairwise at "large" distance from each other.

One non-trivial code allowing efficient testing is the Hadamard code: the codewords are linear functions represented by their values on all possible evaluation points. The number of codewords in Hadamard codes grows with the length of the code, and the pairwise distance between codewords is half of the length of the code. So this code does not admit trivial tests as above. It turns out that in this case codeword testing amounts to linearity testing [14], and this can be performed efficiently, though the analysis is quite non-trivial.

The drawback of the Hadamard code is that k bits of information are encoded using a codeword of length 2^k. (The k information bits represent the k coefficients of a linear function $\{0,1\}^k \to \{0,1\}$, and bits in the codeword correspond to all possible evaluation points.)

A basic question: The question addressed in this work is whether one can hope for a better relation between the number of information bits, denoted k, and the length of the codeword, denoted n. Specifically, *can n be polynomial or even linear in k?* For a sufficiently large *non-binary* alphabet, Friedl and Sudan [21] showed that n can be made nearly quadratic in k. The main contribution of this paper is the demonstration of the existence of locally testable codes in which n is *almost-linear* in k (i.e., $n = k^{1+o(1)}$), *even for the binary alphabet.*

In Section 2.1 we provide precise definition of locally testable codes and state our main results regarding them. But before doing so, we discuss the relation between locally testable codes and three other notions (i.e., PCP, property testing and locally decodable codes).

1.1 Relation to PCP

As mentioned earlier, locally testable codes are closely related to Probabilistically Checkable Proofs (PCPs). Recall that a PCP system is defined by a (probabilistic) verifier that is given a pair of strings – a purported theorem (assertion) and a claimed proof (evidence) – such that if the theorem is true, then there exists a proof such that the verifier accepts; and if the assertion is not true then no evidence causes the verifier to accept (with high probability). Furthermore, PCP verifiers achieve their goals by making only a small number of queries to the proof, which is given as an oracle. The PCP Theorem [2, 3] shows how to construct PCP verifiers that make only a constant number of queries to the proof-oracle.

PCPs achieve their strong features by implicitly relying on objects related to locally testable codes. Indeed the construction of codes over large alphabets that are testable via a small (yet not necessarily constant) number of queries lies at the heart of many PCPs. It is a common belief, among PCP enthusiasts, that the PCP Theorem [2, 3] already provides (binary) locally testable codes. This belief relates to a stronger property of the proof of the PCP theorem which actually provides a transformation from standard witnesses for, say SAT, to PCP-proof-oracles, such that transformed strings are accepted with probability one by the PCP verifier. When applied to an instance of SAT that is a tautology, the map typically induces a good error-correcting code mapping k information bits to codewords of length $\mathrm{poly}(k)$ (or almost linear in k, when using [30]), which are pairwise quite far from each other. The common belief is that the PCP-verifier also yields a codeword test. However, this is not quite true: typically, the analysis only guarantee that each passing oracle can be "decoded" to a corresponding NP-witness, but encoding the decoded NP-witness does not necessarily yield a string that is close to the oracle. In particular, this allows for oracles that are accepted with high probability to be far from any valid codeword. Furthermore, it is not necessarily the case that only codewords pass the test with probability one. For example, part of the proof-oracle (in typical PCPs) is supposed to encode an m-variate polynomial of *individual degree* d, yet the (standard) PCP-verifier will also accept the encoding of any m-variate polynomial of *total degree* $m \cdot d$ (and the "decoding" procedure will work in this case too).

We conclude that the known constructions of PCPs as such do not yield locally testable codes. However, we show that many known PCP constructions can be *modified* to yield good codes with efficient codeword tests. We stress that these modifications are non-trivial and furthermore are *unnatural* in the context of PCP. Yet, they do yield coding results of the type we seek (e.g., see Theorem 2.3).

On the other hand, a technique that emerges naturally in the context of our study of efficient codeword tests yields improved results on the length of efficient PCPs. Specifically, we obtain (constant-query) PCP systems that utilize oracles that are shorter than known before (see Theorem 2.5).

1.2 Relation to Property Testing

Property testing is the study of highly efficient approximation algorithms (tests) for determining whether an input is close to satisfying a fixed property. Specifically, for a property (Boolean function) Π, a test may query an oracle x at few positions and accept if $\Pi(x)$ is true, and reject with high probability if $\Pi(\tilde{x})$ is not true for every $\tilde{x}$ that is "close" to x. Property testing was defined in [32] (where the focus was on algebraic properties) and studied systematically in [22] (where the focus was on combinatorial properties).

Viewed from the perspective of property testing, the tester of a local testable code is a tester for the property of being a member of the code, where the notion of "closeness" is based on Hamming distance. Furthermore, in the coding setting, it is especially natural that one is not interested in exactly deciding whether or not the input is a codeword, but rather in the "approximate" distance of the input from the code (i.e., whether it is a codeword or far from any codeword). Thus, locally testable codes are especially well-connected to the theme of property testing. Indeed the first property tests in the literature (e.g., linearity tests [14], low-degree tests [6, 5, 32, 21]) can be interpreted as yielding some forms of locally testable codes. More recent works on algebraic testing [7, 1] highlight the connections to codes more explicitly. Our work also uses the results and techniques developed in the context of low-degree testing. However, by focusing on the codes explicitly, we highlight some missing connections. In particular, most of the prior work focussed on codes over large alphabets and did not show how to go from testable codes over large alphabets to codes over small alphabets. In this work we address such issues explicitly and resolve them to derive our main results. Furthermore, we focus on codes that can be tested by making a constant number of queries.

1.3 Relation to Locally Decodable Codes

A task that is somewhat complementary to the task investigated in this paper, is the task of local decoding. That is, we refer to the project of constructing codes that have very efficient (sub-linear time) implicit decoding algorithms. Specifically, given oracle access to a string that is close to some unknown codeword, the decoding procedure should recover any desired bit of the corresponding message while making, say, a *constant number of queries* to the input oracle. Codes that have such decoding algorithms are called locally decodable codes. While local testability and local decodability appear related, no general theorems linking the two tasks are known. In fact, gaps in the performance of known constructions for the two tasks suggest that local decodability is "harder" to achieve than local testability. Our results confirm this intuition:

- We show the existence of almost-linear (i.e., $n = k^{1+o(1)}$) length (binary) codes having codeword tests that make a *constant number of queries*. In contrast, it was shown that locally decodable codes cannot have almost-linear length [27]: that is, if q queries are used for recovery then $n = \Omega(k^{1+(1/(q-1))})$.

- For a (large) alphabet that can be viewed as vector space over some field F, we show almost-linear length F-linear codes having codeword tests that make only *two* queries. In contrast, it was shown that F-linear codes that allow for local decodability by *two* queries require exponential length [23].

  Specifically, an F-linear code over the alphabet $\Sigma = F^\ell$ is a linear space over F (but not necessarily over F^ℓ). In our codes (which support two-query tests) it holds that $\ell = \exp(\sqrt{\log k})$ and $|F| = O(\ell)$, while $n < k^{1+(\log k)^{-0.4999}} = k^{1+o(1)}$. In contrast, the lower-bound on n (for two-query decoding) established in [23] assert that $n > \exp(\Omega(k - (\ell \cdot \ell')^2))$ in case $F = \mathrm{GF}(2^{\ell'})$, which yields $n > \exp(\Omega(k))$ for the relevant values of $\ell = \exp(\sqrt{\log k}) = k^{o(1)}$ and $\ell' = \log O(\ell)$.

1.4 Organization and previous versions

Section 2 provides a formal treatment of locally testable codes and PCPs. It also contains a (formal) statement of our main results as well as a high-level discussion of our main techniques (Section 2.3). In Section 3 we present direct and self-contained constructions of locally testable codes (albeit not achieving the best results). We stress that these constructions make no reference to PCP, although they do use low-degree tests. Sections 1-3 occupy less than a third of the length of the paper.

Our best constructions of locally testable codes are presented in Section 5, where we adapt standard PCP constructions and combine them with the construction presented in Section 3.2. This section takes about half of the length of the paper. In Section 4, we adapt some of the ideas presented in Section 3.2 in order to derive improved PCPs. We stress that Sections 4 and 5 can be read independently of one another, whereas they both depend on Section 3.2.

Subsequent works and open problems are discussed in Section 6. In particular, we mention that the subsequent works of Ben-Sasson *et al.* [10], Ben-Sasson and Sudan [12], and Dinur [16] do not provide strong codeword tests (but rather only weak ones).

The current version differs from our preliminary report [24] in several aspects, the most important ones are discussed next.

- In Section 2.1, we present two definitions of locally-testable codes, whereas only the weaker one has appeared in [24]. Furthermore, in order to obtain locally-testable codes under the stronger definition, we use a different analysis of the constructions presented in Section 3.2.

- Section 5 has been extensively revised, while narrowing the scope of some of the secondary results (e.g., the two composition theorems (i.e., Theorems 5.13 and 5.16)). These modifications do not effect our main results.

In addition, the presentation has been expanded and high-level overviews (most notably Sections 2.3 and 5.3.1) were added.

2 Formal Setting

Throughout this work, all oracle machines (i.e., codeword testers and PCP verifiers) are non-adaptive; that is, they determine their queries based solely on their input and random choices. This is in contrast to adaptive oracle machines that may determine their queries based on answers obtained to prior queries. Since our focus is on positive results, this only makes our results stronger.

Throughout this work, all logarithms are to base 2, and for a natural number n we denote $[n] \stackrel{\text{def}}{=} \{1, ..., n\}$. We often use an arbitrary finite set, other than $[n]$, as an index set to some sequence. For any finite set S, we denote by $\langle e_i : i \in S \rangle$ the sequence of e_i's, where the order in the sequence is induced by an (often unspecified) total order of the set S.

2.1 Codes

We consider codes mapping a sequence of k input symbols into a sequence of $n \geq k$ symbols over the same alphabet, denoted Σ, which may (but need not) be the binary alphabet. Such a generic code is denoted by $\mathcal{C} : \Sigma^k \to \Sigma^n$, and the elements of $\{\mathcal{C}(a) : a \in \Sigma^k\} \subseteq \Sigma^n$ are called codewords (of $\mathcal{C}$). Sometimes, it will be convenient to view such codes as maps $\mathcal{C} : \Sigma^k \times [n] \to \Sigma$.

Throughout this paper, *the integers k and n are to be thought of as parameters*, and Σ may depend on them. Thus, we actually discuss infinite families of codes (which are associated with infinite sets of possible k's), and whenever we say that some quantity of the code is a constant we mean that this quantity is constant for the entire family (of codes). In particular, the rate of a code is the functional dependence of n on k, which we wish to be almost-linear. Typically, we seek to have Σ as small as possible, desire that $|\Sigma|$ be a constant (i.e., does not depend on k), and are most content when $\Sigma = \{0, 1\}$ (i.e., a binary code).

The distance between n-long sequences over Σ is defined in the natural manner; that is, for $u, v \in \Sigma^n$, the distance $\Delta(u, v)$ is defined as the number of locations on which u and v differ (i.e.,

$\Delta(u,v) \stackrel{\rm def}{=} |\{i : u_i \neq v_i\}|$, where $u = u_1 \cdots u_n \in \Sigma^n$ and $v = v_1 \cdots v_n \in \Sigma^n$). The relative distance between u and v, denoted $\delta(u,v)$, is the ratio $\Delta(u,v)/n$. To avoid technical difficulties, we define the distance between sequences of different length to equal the length of the longer sequence.

The distance of a code $\mathcal{C} : \Sigma^k \to \Sigma^n$ is the minimum distance between its codewords; that is, $\min_{a \neq b}\{\Delta(\mathcal{C}(a), \mathcal{C}(b))\}$. *Throughout this work, we focus on codes of "large distance"*; specifically, codes $\mathcal{C} : \Sigma^k \to \Sigma^n$ of distance $\Omega(n)$.

The distance of $w \in \Sigma^n$ from a code $\mathcal{C} : \Sigma^k \to \Sigma^n$, denoted $\Delta_{\mathcal{C}}(w)$, is the minimum distance between w and the codewords; that is, $\Delta_{\mathcal{C}}(w) \stackrel{\rm def}{=} \min_a\{\Delta(w, \mathcal{C}(a))\}$. An interesting case is of non-codewords that are "relatively far from the code", which may mean that their distance from the code is greater than (say) a third of the distance of the code.

We will sometimes say that $w \in \Sigma^n$ is ϵ-far from v (resp., from the code $\mathcal{C}$), meaning that $\Delta(w,v) \geq \epsilon \cdot n$ (resp., $\Delta_{\mathcal{C}}(w) \geq \epsilon \cdot n$). Similarly, we say that w is ϵ-close from v (resp., from $\mathcal{C}$) if $\Delta(w,v) \leq \epsilon \cdot n$ (resp., $\Delta_{\mathcal{C}}(w) \leq \epsilon \cdot n$). Note that we have allowed w to be both ϵ-far and ϵ-close to v (resp., $\mathcal{C}$) in case its relative distance to v (resp., $\mathcal{C}$) is *exactly* ϵ.

2.1.1 Codeword tests: weak and strong versions

Loosely speaking, by a codeword test (for the code $\mathcal{C} : \Sigma^k \to \Sigma^n$) we mean a randomized (non-adaptive) oracle machine, called a tester, that is given oracle access to $w \in \Sigma^n$ (viewed as a function $w : [n] \to \Sigma$). The tester is required to (always) accept every codeword and reject with (relatively) high probability every oracle that is "far" from the code. Indeed, since our focus is on positive results, we use a strict formulation in which the tester is required to accept each codeword with probability 1. (This corresponds to "perfect completeness" in the PCP setting.)

The following two definitions differ by what is required from the tester in case the oracle is not a codeword. The weaker definition (which is the one that appears in our preliminary report [24]) requires that for every $w \in \Sigma^n$, given oracle access to w, the tester rejects with probability $\Omega(\Delta_{\mathcal{C}}(w)/n) - o(1)$. An alternative formulation (of the same notion) is that, for some function $f(n) = o(n)$, every $w \in \Sigma^n$ that is at distance greater than $f(n)$ from $\mathcal{C}$ is rejected with probability $\Omega(\Delta_{\mathcal{C}}(w)/n)$. Either way, this definition (i.e., Definition 2.1) effectively requires nothing with respect to non-codewords that are relatively close to the code (i.e., are $(f(n)/n)$-close to $\mathcal{C}$). A stronger and smoother definition (i.e., Definition 2.2) requires that *every* non-codeword w is rejected with probability $\Omega(\Delta_{\mathcal{C}}(w)/n)$.

Definition 2.1 (codeword tests, weak definition): *A randomized* (non-adaptive) *oracle machine M is called a* weak codeword test for $\mathcal{C} : \Sigma^k \to \Sigma^n$ *if it satisfies the following two conditions:*

1. Accepting codewords: *For any $a \in \Sigma^k$, given oracle access to $w = \mathcal{C}(a)$, machine M accepts with probability 1. That is,* $\Pr[M^{\mathcal{C}(a)}(k,n,\Sigma) = 1] = 1$*, for any $a \in \Sigma^k$.*

2. Rejection of non-codeword: *For some constant $c > 0$ and function $f(n) = o(n)$, for every $w \in \Sigma^n$, given oracle access to w, machine M rejects with probability at least $(c \cdot \Delta_{\mathcal{C}}(w) - f(n))/n$. That is,* $\Pr[M^w(k,n,\Sigma) \neq 1] \geq (c \cdot \Delta_{\mathcal{C}}(w) - f(n))/n$*, for any $w \in \Sigma^n$.*

We say that the code $\mathcal{C} : \Sigma^k \to \Sigma^n$ is weakly locally testable *if it has a weak codeword test that makes a* constant number of queries.

Definition 2.2 (codeword tests, strong definition): *A randomized* (non-adaptive) *oracle machine M is called a* strong codeword test for $\mathcal{C} : \Sigma^k \to \Sigma^n$ (or just a codeword test for $\mathcal{C} : \Sigma^k \to \Sigma^n$) *if it satisfies the following two conditions:*

1. Accepting codewords: *As in Definition 2.1, for any $a \in \Sigma^k$, given oracle access to $w = \mathcal{C}(a)$, machine M accepts with probability 1.*

2. Rejection of non-codeword: *For some constant $c > 0$ and for every $w \in \Sigma^n$, given oracle access to $w \in \Sigma^n$, machine M rejects with probability at least $c \cdot \Delta_{\mathcal{C}}(w)/n$.*

   That is, $\Pr[M^w(k,n,\Sigma) \neq 1] \ \geq \ c \cdot \Delta_{\mathcal{C}}(w)/n$, *for any $w \in \Sigma^n$.*

We say that the code $\mathcal{C} : \Sigma^k \to \Sigma^n$ is locally testable *if it has a strong codeword test that makes a* constant number of queries.

Our constructions satisfy the stronger definition (i.e., Definition 2.2), but we consider the weaker definition (i.e., Definition 2.1) to be of sufficient interest to warrant presentation here. Furthermore, in two cases (i.e., in the proof of Claim 3.5.2 and in Section 5.1), we find it instructive to establish the weak definition before turning to the strong one.

We comment that one may consider various natural variants on the two definitions. For example, in both cases, we have required that the rejection probability grows linearly with the distance of the oracle from the code. More generally, one may consider requiring a slower (e.g., polynomial) growth rate. Another example is relaxing our requirement that every codeword is accepted with probability 1. More generally, one may allow codewords to be rejected with some small probability. (Note that this relaxation (w.r.t codewords) may be odd if coupled with the stronger definition regarding non-codewords (i.e., the one in Definition 2.2).)

Relation to property testing: Codeword tests are indeed a special type of property testers (as defined in [32, 22]). However, in the "property testing" literature one typically prefers providing the tester with a *distance parameter* and requiring that the tester rejects all objects that are that far from the property with probability at least 2/3 (rather than with probability proportional to their distance). In such a case, the query complexity is measured as a function of the distance parameter and is constant only when the latter parameter is a constant fraction of the maximum possible distance. Strong codeword testers yield property testers with complexity that is inversely proportional to the distance parameter, whereas the complexity of testers derived from *weak codeword tests* is "well behaved" only for large values of the distance parameter.

2.1.2 Our main results

Our main result regarding codes is the following

Theorem 2.3 (locally testable binary codes of $k^{1+o(1)}$ length): *For infinitely many k's, there exist locally testable codes with binary alphabet such that $n = \exp(\tilde{O}(\sqrt{\log k})) \cdot k = k^{1+o(1)}$. Furthermore, these codes are linear and have distance $\Omega(n)$.*

Theorem 2.3 (as well as Part 2 of Theorem 2.4) vastly improves over the Hadamard code (in which $n = 2^k$), which is the only locally testable *binary* code previously known. Theorem 2.3 is proven by combining Part 1 of the following Theorem 2.4 with non-standard modifications of standard PCP constructions. We emphasize the fact that Theorem 2.4, which is weaker than Theorem 2.3, is proven without relying on any PCP construction.

Theorem 2.4 (weaker results proved by direct/self-contained constructions):

1. *For infinitely many k's, there exist locally testable codes with non-binary alphabet Σ such that $n = \exp(\tilde{O}(\sqrt{\log k})) \cdot k = k^{1+o(1)}$ and $\log|\Sigma| = \exp(\tilde{O}(\sqrt{\log k})) = k^{o(1)}$. Furthermore, the tester makes* two *queries and the code is F-linear*[1]*, where $\Sigma = F^\ell$.*

2. *For every $c > 1$ and infinitely many k's, there exist locally testable codes over* binary *alphabet such that $n < k^c$. Furthermore, the code is linear.*

In both cases, the codes have distance $\Omega(n)$.

Part 1 improves over the work of Friedl and Sudan [21], which only yields $n = k^{2+o(1)}$.

The set of k's for which Theorems 2.3 and 2.4 hold is reasonable dense; in all cases, if k is in the set then the next integer in the set is smaller than $k^{1+o(1)}$. Specifically, in Part 1 (resp., Part 2) of Theorem 2.4, if k is in the set then the next integer in the set is smaller than $\exp((\log k)^{0.51}) \cdot k$ (resp., $O(\mathrm{poly}(\log k) \cdot k)$).

Caveat: Both Theorems 2.3 and 2.4 are proven via the probabilistic method, and thus do not yield an explicit construction. Such a construction has been found subsequently by Ben-Sasson, Sudan, Vadhan and Wigderson [13]. (See further discussion in Section 6.)

Comment: The result of Theorem 2.3 holds also when using testers that make *three* queries. On the other hand, (good) *binary* codes cannot be tested using two queries (cf. [11]).

2.2 PCP: Standard definitions and new results

Following [8], we consider PCP systems for promise problems (cf. [18]). (Recall that a promise problem is a pair of non-intersecting subsets of $\{0,1\}^*$, which do not necessarily cover $\{0,1\}^*$.) A probabilistic checkable proof (PCP) system for a promise problem $\Pi = (\Pi_{\mathrm{YES}}, \Pi_{\mathrm{NO}})$ is a probabilistic polynomial-time (non-adaptive) oracle machine (called verifier), denoted V, satisfying

- *Completeness*: For every $x \in \Pi_{\mathrm{YES}}$ there exists an oracle π_x such that V, on input x and access to oracle π_x, always accepts x.
- *Soundness*: For every $x \in \Pi_{\mathrm{NO}}$ and every oracle π, machine V, on input x and access to oracle π, rejects x with probability at least $\frac{1}{2}$.

  Actually, we will allow the soundness error to be a constant that is arbitrary close to $\frac{1}{2}$.

As usual, we focus on PCP systems with *logarithmic randomness complexity* and *constant query complexity*. This means that, without loss of generality, the length of the oracle is polynomial in the length of the input. However, we aim at PCP systems that utilize oracles that are of almost-linear length. Our main result regarding such PCP systems is the following

Theorem 2.5 *There exists an almost-linear time randomized reduction of SAT to a promise problem that has a 19-query PCP system that utilizes oracles of length $\exp(\tilde{O}(\sqrt{\log n})) \cdot n = n^{1+o(1)}$, where n is the length of the input. Furthermore, the reduction maps k-bit inputs to n-bit inputs such that $n = \exp(\tilde{O}(\sqrt{\log k})) \cdot k = k^{1+o(1)}$.*

This should be compared to the PCP system for SAT of Polishchuk and Spielman [30] that when utilizing oracles of length $n^{1+\epsilon}$ makes $O(1/\epsilon)$ queries. In contrast, our PCP system utilizing oracles of length $n^{1+o(1)}$ while making 19 queries.

[1]A code over the alphabet $\Sigma = F^\ell$ is called F-linear code if its codewords form a linear space over F (but not necessarily over F^ℓ).

Caveat: Theorem 2.5 does not yield a PCP for SAT, but rather a PCP for a promise problem to which SAT can be reduced (via a randomized reduction that runs in almost-linear time). (The reduction merely augments the input by a random string of an adequate length; thus allowing the application of a probabilistic argument analogous to the one underlying the proof of Part 1 of Theorem 2.4.) A PCP for SAT itself has been found subsequently by Ben-Sasson, Sudan, Vadhan and Wigderson [13].

2.3 Our techniques

In this section we highlight some of the techniques used in the paper.

Random projection of codes and PCPs. We derive locally-testable codes (resp., PCPs) of shorter length by showing that a random projection of the original codewords (resp., proofs) on a smaller number of coordinates maintains the testability of the original construct. In Section 3.2 this process is applied to a specific code, in Section 4.1 it is applied to any two-prover one-round proof system, and in Section 4.2 it is applied to certain three-prover proof systems. In retrospect, one may say that in all cases we show that in *certain* multi-prover proof systems one may randomly "trim" all the provers to the "size" of the smallest one, where the size of a prover is defined as the length of (the explicit description of) its strategy (which, in turn, is exponential in the length of the queries that the prover answers).

Extending the paradigm of code-concatenation to codeword testing. The notion of concatenated codes was introduced by Forney [20] (in the 1960's) as a technique for reducing the alphabet size of codes. Our constructions of locally-testable codes extend this technique by showing that in *certain* cases codeword testers for the "outer" and "inner" codes yield a codeword tester for the concatenated code. Specifically, we refer to cases where the inner code allows direct access to the values checked by the tester of the outer code, and furthermore that this direct access supports self-correction (cf. [14]). Two examples appear in Sections 3.3 and 3.4, respectively. We also explore the related composition of locally-testable codes with (inner-verifier) PCPs; see Section 5.3.

Developing a theory of PCPs for linear assertions. When composing a locally-testable code with an inner-verifier, we may obtain a locally-testable code over a smaller alphabet, but will this code preserve the linearity of the original code? This seems to require that the inner-verifier uses proof-oracles that are linear in the main input, which seems plausible when the assertion itself is linear (i.e., asserts that the input resides in some linear subspace). The suitable definitions and composition results are developed in Section 5.3, while in Section 5.4 we show that known PCP constructions can be modified to maintain the linearity of the assertions.

Two notions of strong soundness for PCP. When composing a locally-testable code with an inner-verifier, we may preserve the strong testability of the original codeword test if the inner-verifier satisfies two (adequate) "strong" soundness conditions. The first condition requires the rejection of "non-canonical" proofs, whereas the second condition requires the rejection of non-proofs with probability proportional to their distance from a valid proof. We believe that these notions may be of independent interest, and refer the reader to Section 5.3.1 for a general presentation of these notions.

We comment that unexpected technical problems arise when composing such PCPs with themselves (resp., with locally-testable codes): the issue is the preservation of strong soundness (rather than standard soundness) by the composition. This issue is addressed in Section 5.3.4 (resp., Section 5.3.3).

3 Direct Constructions of Short Locally Testable Codes

In this section, we prove Theorem 2.4. In particular, for every $c > 1$, we present locally testable codes that map k bits of information to codewords of length k^c. These codes are presented in a direct and self-contained manner (without using any general PCPs). Although we do *not* use any variant of the PCP Theorem, our constructions are somewhat related to known PCP constructions in the sense that we use constructs and analyses that appear, at least implicitly, in the "PCP literature" (e.g., in [2, 3]). Specifically, we will use results regarding "low-degree tests" that were proven for deriving the PCP Theorem [2, 3]. We stress that we do not use the more complex ingredients of the proof of the PCP Theorem; that is, we neither use the (complex) parallelization procedure nor the "proof-composition" paradigm of [3, 2]. We note that the proof-composition paradigm is more complex than the classical notion of concatenated codes [20] used below.

We start by describing (in Section 3.1) a code over a large alphabet, which we refer to as the FS/RS code. This code, which is a direct interpretation of low-degree tests, was proposed by Friedl and Sudan [21] and Rubinfeld and Sudan [32]. The length of codewords (in this code) turns out to be nearly quadratic in the length of the encoded information (even when using the best possible analysis of low-degree tests). To reduce the length of the code to being nearly linear, we introduce (in Section 3.2) a "random projection" technique. This establishes Part 1 of Theorem 2.4 (which refers to codes over large alphabets, and will be used to establish Theorem 2.3). In Sections 3.3 and 3.4 we apply the "code concatenation" technique to reduce the alphabet size of the codes, while preserving local testability. Specifically, in Section 3.3 we obtain locally testable codes over a much smaller (albeit non-binary) alphabet, whereas in Section 3.4 we obtain a binary code, thus establishing Part 2 of Theorem 2.4.

3.1 The Basic Code (FS/RS-Code)

The FS/RS code is based on low-degree multi-variate polynomials over finite fields. We thus start with the relevant preliminaries. Let F be a finite field, and m, d be integer parameters such that $m \leq d < |F|$. Denote by $P_{m,d}$ the set of m-variate polynomials of *total* degree d over F. We represent each $p \in P_{m,d}$ by the list of its $\binom{m+d}{m}$ coefficients; thus,

$$|P_{m,d}| = |F|^{\binom{m+d}{m}} < |F|^{O(d/m)^m} \tag{1}$$

where the inequality holds because $m \leq d$ and $\binom{2d}{m} < (2d)^m/(m!) = O(d/m)^m$.

Denote by L_m the set of lines over F^m, where each line is defined by two points $a, b \in F^m$; that is, for $a = (a_1, ..., a_m)$ and $b = (b_1, ..., b_m)$, the line $\ell_{a,b}$ consists of the set of $|F|$ points $\{\ell_{a,b}(t) \stackrel{\text{def}}{=} ((a_1 + tb_1), ..., (a_m + tb_m)) : t \in F\}$.

The code. We consider a code $\mathcal{C} : P_{m,d} \to \Sigma^{|L_m|}$, where $\Sigma = F^{d+1}$; that is, $\mathcal{C}$ assigns each $p \in P_{m,d}$ a ($|L_m|$-long) sequence of Σ-values. For every $p \in P_{m,d}$, the codeword $\mathcal{C}(p)$ is a sequence of $|L_m|$ univariate polynomials, each of degree d, such that the element in the sequence associated with $\ell \in L_m$ is the univariate polynomial that represents the values of the polynomial $p : F^m \to F$ on the line ℓ. We view L_m as the set of indices (or coordinates) in any $w \in \Sigma^{|L_m|}$; that is, we view w as a function from L_m to Σ. Thus, for any $\ell \in L_m$, we denote by $w(\ell)$ the symbol in w having index ℓ. Viewing the code $\mathcal{C}$ as a mapping $\mathcal{C} : P_{m,d} \times L_m \to \Sigma$ such that $\mathcal{C}(p, \cdot)$ is the encoding (or codeword) of $p \in P_{m,d}$, we have that for every $\ell_{a,b} \in L_m$ the univariate polynomial $q_{a,b} = \mathcal{C}(p, \ell_{a,b})$ satisfies $q_{a,b}(z) = p(\ell_{a,b}(z))$, where $p(\ell_{a_1,...,a_m,b_1,...,b_m}(z)) = p((a_1 + b_1 z), ..., (a_m + b_m z))$. Note that, indeed, if p has total degree d then, for every $\ell_{a,b} \in L_m$, the univariate polynomial $q_{a,b} = \mathcal{C}(p, \ell_{a,b})$ has degree at most d.

Parameters. To evaluate the basic parameters of the code $\mathcal{C}$, let us consider it as mapping $\Sigma^k \to \Sigma^n$, where indeed $n = |L_m| = |F|^{2m}$ and $k = \log |P_{m,d}|/\log |\Sigma|$. Note that

$$k \;=\; \frac{\log |P_{m,d}|}{\log |\Sigma|} \;=\; \frac{\binom{m+d}{d} \log |F|}{(d+1) \log |F|} \;=\; \frac{\binom{m+d}{m}}{d+1} \tag{2}$$

which, for $m \ll d$, is approximated by $(d/m)^m/d \approx (d/m)^m$. Using $|F| = \mathrm{poly}(d)$, we have $n = |F|^{2m} = \mathrm{poly}(d^m)$, and so $k \approx (d/m)^m$ is polynomially related to $n = |F|^{2m}$ (provided, say, that $m < \sqrt{d}$). Note that the code has large distance, because the different $\mathcal{C}(p)$'s tend to disagree on most lines.

The Codeword Test: The test consists of selecting two random lines that share a random point, and checking that the univariate polynomials associated with these lines yield the same value for the shared point. That is, to check whether $w : L_m \to \Sigma$ is a codeword, we select a random point $r \in F^m$, and two random lines ℓ', ℓ'' going through r (i.e., $\ell'(t') = r$ and $\ell''(t'') = r$ for some $t', t'' \in F$), obtain the answer polynomials q' and q'' (i.e., $q' = w(\ell')$ and $q'' = w(\ell'')$) and check whether they agree on the shared point (i.e., whether $q'(t') = q''(t'')$). This test is essentially the one analyzed in [2], where it is shown that (for $|F| = \mathrm{poly}(d)$) if the oracle is ϵ-far from the code then this fact is detected with probability $\Omega(\epsilon)$.

We comment that in [2] the test is described in terms of two oracles: a point oracle $f : F^m \to F$ (viewed as the primary or "real" input) and a line oracle $g : L_m \to F^{d+1}$ (viewed as an auxiliary or additional oracle). Indeed, we will also revert to this view in our analysis. Unfortunately, using oracles having different range will complicate the code-concatenation (presented in Section 3.3), and this is the reason that we maintain explicitly only the line-oracle (and refer to the point-oracle only in the analysis). Note that a line-oracle can be used to define a corresponding point-oracle in several natural ways. For example, we may consider the (random) value given to each point by a random line passing through this point, or consider the value given to each point by a *canonical* line passing through this point.

3.2 Random projection of the FS/RS-Code

Our aim in this section is to tighten the relationship between k and n in locally testable codes. Starting with the FS/RS-Code, in order to get the best possible relation between n and k, one needs to use an analysis (of the low-degree test) that allows for $|F|$ to be as small as possible when compared to d. Based on the analysis of [30], it was shown in [21] that it suffices to use $|F| = \Theta(d)$. However, even with this (best possible) analysis, we are still left with n that is quadratic in $|F|^m$, whereas $k = o(d^m) = o(|F|^m)$. This quadratic blowup comes from the fact that the number of lines (over F^m) is quadratic in the number of points, which in turn upper-bounds the number of coefficients of a (generic) m-variate polynomial (over F). Thus, to obtain n almost-linear in k, we must use a different code.

Overview of our construction: Our main idea here is to project the FS/RS code to a randomly chosen subset of the coordinates. Thus, our code is essentially just a projection of the FS/RS code to a random subset of lines over F^m. This subset will have size that is almost-linear in $|F|^m$, and consequently the code will have almost-linear length. We note that, with overwhelmingly high probability (over the choices of this random subset), approximately the same number of selected lines pass through each point of F^m. It is also easy to see that, with overwhelmingly high probability, the resulting code maintains the distance properties of the basic FS/RS-Code. Most of this subsection will be devoted to proving that the resulting code also maintains the local testability properties of the FS/RS-Code.

The projected code: In what follows, we will fix positive integers m, d and a field F. We will assume $\log\log d \le m \le d$ and $|F| = \Theta(d)$. Our code will be over the alphabet $\Sigma = F^{d+1}$ corresponding to the vector space of univariate polynomials of degree at most d over F. For the sake of concreteness, we will assume that the univariate polynomial $p(x) = \sum_{i=0}^{d} c_i x^i$ is represented by the vector $\langle c_0, \ldots, c_d \rangle$. Let $L = L_m$ denote the collection of all lines in F^m. For a (multi-)set $R \subseteq L$, we define the code $\mathcal{C}^R : P_{m,d} \to \Sigma^R$ such that, for every $p \in P_{m,d}$ and $\ell \in R'$, the ℓ-th symbol in the encoding $\mathcal{C}^R(p)$ is the polynomial obtained by restricting p to the line ℓ. In the following definition, we view the code as a mapping from $P_{m,d} \times R$ to Σ.

Construction 3.1 *Let F be a finite field, $m \le d$ be integers, and $\Sigma = F^{d+1}$. We define $\mathcal{C}^R$: $P_{m,d} \times R \to \Sigma$ such that, for every $p \in P_{m,d}$ and $\ell \in R \subseteq L_m$, it holds that $\mathcal{C}(p, \ell)$ is the univariate polynomial that represents the values of the m-variant polynomial p on the line ℓ. That is, for every $e \in F$, the polynomial $\mathcal{C}(p, \ell)$ evaluated at e yields the value $p(\ell(e))$.*

Thus, our encoding is simply a projection of the FS/RS code to the coordinates in R, where R is an arbitrary subset of L.

In what follows, we will show that if R is chosen uniformly at random (with replication from L) and $|R| = \Theta(m|F|^m \log|F|)$, then the code $\mathcal{C}^R$ is locally testable. (To shorten our sentences we will simply say "R is chosen randomly" and mean that the elements of the multi-set R are chosen uniformly at random from L.) We next describe the parameters of the code, and then describe the codeword test.

The basic parameters: We consider the information length k, the block length n and relative distance of the code. To compare k with n, let us consider the code $\mathcal{C}^R$ as a mapping $\Sigma^k \to \Sigma^n$, where $n = |R| = O(m|F|^m \log|F|)$ and $k = \log|P_{m,d}|/\log|\Sigma|$ (as in Eq. (2)). Then, $k = \Theta(d/m)^m/d = \Theta(d)^{m-1}/m^m$ and, for $|F| = O(d)$, we have $n = O(m|F|^m \log|F|) = \tilde{O}(O(d)^m)$. In this case $\log|\Sigma| = \log|F|^{d+1} = \tilde{O}(d)$. We highlight two possible settings of the parameters:

1. Using $d = m^m$, we get $k = \Omega(d)^{m-2} = m^{m^2-2m-o(m)}$ and $n = \tilde{O}(O(d)^m) = m^{m^2+o(m)}$, which yields
$$n = \exp(\tilde{O}(\sqrt{\log k})) \cdot k \text{ and } \log|\Sigma| = \exp(\tilde{O}(\sqrt{\log k})). \tag{3}$$
2. For any constant $e > 1$, letting $d = m^e$, we get $k = \Omega(m^e)^{m-1}/m^m = m^{(e-1-o(1))m}$ and $n = \tilde{O}(O(d)^m) = m^{(e+o(1))m}$, which yields
$$n = k^{(e+o(1))/(e-1)} \text{ and } \log|\Sigma| < (\log k)^e. \tag{4}$$

We next show that when $|F| = \Omega(d)$ and R is a randomly chosen set of size $\Omega(m|F|^m \log |F|)$, the code has constant relative distance, with overwhelmingly high probability. This can be proven by upper-bounding the probability that the distance between any two codewords is too small. However, it is somewhat less cumbersome to first prove that the code is "linear" in an adequate sense (as defined below), and next to upper-bound the probability that any (non-zero) codeword has too small weight. (Furthermore, for sake of later use, we need to establish this linearity property anyhow.)

F-linearity. We say that a subset $C \subseteq \Sigma^n$, where $\Sigma = F^{d+1}$, is F-linear if C is a linear subspace of $(F^{d+1})^n$ when viewed as a vector space over F. In other words, for every $x, y \in C$ and $\alpha, \beta \in F$, it is the case that $\alpha x + \beta y \in C$, where $\alpha x = (\alpha x_1, \ldots, \alpha x_n)$ for $x = (x_1, \ldots, x_n) \in (F^{d+1})^n$ and αx_i denotes the usual scalar-vector product.

Proposition 3.2 *For every R, the code $\mathcal{C}^R$ is F-linear. That is, for every $p', p'' \in P_{m,d}$ and $\alpha, \beta \in F$, it holds that $\alpha\mathcal{C}^R(p', \cdot) + \beta\mathcal{C}^R(p'', \cdot)$ equals $\mathcal{C}^R(q, \cdot)$, for some $q \in P_{m,d}$.*

Proof: Letting $p(\ell)$ denote the univariate polynomial representing the values of the polynomial p when restricted to the line ℓ, we have $\mathcal{C}^R(p', \ell) = p'(\ell)$ and $\mathcal{C}^R(p'', \ell) = p''(\ell)$. Thus, for every $\ell \in R$, it holds that

$$\mathcal{C}^R(\alpha p' + \beta p'', \ell) \;=\; (\alpha p' + \beta p'')(\ell) \;=\; \alpha p'(\ell) + \beta p''(\ell) \;=\; \alpha\mathcal{C}^R(p', \ell) + \beta\mathcal{C}^R(p'', \ell)$$

where the second equality follows from the fact that $(\alpha p' + \beta p'')(x) = \alpha p'(x) + \beta p''(x)$ for every $x \in F^m$. Hence, $\mathcal{C}^R(\alpha p' + \beta p'', \cdot) = \alpha\mathcal{C}^R(p', \cdot) + \beta\mathcal{C}^R(p'', \cdot)$, and the proposition follows (indeed, with $q = \alpha p' + \beta p''$). ■

The relative distance of $\mathcal{C}^R$. We now turn back to analyze the relative distance of $\mathcal{C}^R$.

Proposition 3.3 *With probability $1 - o(1)$, for a randomly chosen R, the code $\mathcal{C}^R$ has relative distance at least $\delta = \Omega(1 - d/|F|) > 0$.*

We mention that the error probability in this proposition is exponentially vanishing (as a function of $|F|^m$).

Proof: Intuitively, the code $\mathcal{C}^L$ has relative distance at least $1 - d/|F|$, and so projection on a random subset of coordinates should leave it with relative distance at least $\delta = \Omega(1 - d/|F|)$. Below, we formally prove this assertion for $\delta = \frac{1}{2} \cdot (1 - d/|F|)$, but the same argument can be used to establish $\delta = c \cdot (1 - d/|F|)$, for any constant $c < 1$.

Since the code $\mathcal{C}^R$ is F-linear (see Proposition 3.2), the distance between any two different codewords is captured by the weight of some non-zero codeword. Thus, it suffices to lower-bound the weight of all non-zero codewords in $\mathcal{C}^R$. We fix a *non-zero* polynomial $p \in P_{m,d}$, and consider the corresponding codeword $\mathcal{C}^R(p)$. Our aim is to prove that the probability that $\mathcal{C}^R(p)$ has relative weight less than δ is at most $o(|P_{m,d}|^{-1})$.

We first consider $\mathcal{C}^L(p)$. By the well-known property of multivariate polynomials, we have that p evaluates to non-zero values on at least $1 - d/|F|$ fraction of the points in F^m. Extending this fact to lines, we can infer immediately that the restriction of p to a $1 - d/|F| = 2\delta$ fraction of the lines is non-zero. (This is true since one can sample a random line by picking a random point x and picking a random line through x, and if the p is non-zero at x, it must be non-zero on the line.) So in order for $\mathcal{C}^R(p)$ to have fewer than a δ fraction of non-zero coordinates, it must be that p is non-zero on fewer than a δ fraction of the lines in R. But we also have that the expected fraction of lines in R where p is non-zero, when R is chosen at random, is at least 2δ. Applying (the multiplicative) Chernoff Bound[2], we get that the probability that this fraction turns out to be less than δ when R is chosen at random, is at most $\exp(-\Omega(\delta|R|)) = o(|F|^{-|F|^m}) = o(|P_{m,d}|^{-1})$. Thus, the probability that $\mathcal{C}^R(p)$ has relative weight less than δ is at most $o(|P_{m,d}|^{-1})$. Taking the union bound over all possible polynomials p, we conclude that the probability that $\mathcal{C}^R$ has a codeword of weight less than δ is at most $o(1)$. ■

We now move to describing the codeword test.

The Codeword Test: The test for the code $\mathcal{C}^R$ is a variant of the points-vs-lines test (cf. [2]) that accesses two oracles, one giving the value of a function $f : F^m \to F$ and the other supposedly giving the restriction of f to lines in F^m. The original test picks a random point $x \in F^m$ and a random line $\ell \in L$ passing through x and verifies that $f(x)$ agrees with the supposed value of the restriction of f to the line ℓ. In implementing this test, we modify it in two ways: Firstly, we do not have the value of the restriction of f to each line, but rather only to lines in R. So we modify the above test by picking a random $\ell \in R$ that passes through x. Secondly, we do not (actually) have oracle access to the value of f on individual points, but rather the value of the restriction of f to various lines (i.e., those in R). So we use the values assigned to these lines in order to define such a point oracle. This can be done in various ways, and we used one of them.[3] Specifically, given a set of lines R, we associate to each point $x \in F^m$ some fixed line (in R), denoted ℓ_x, that passes through x. Note that we do not assume that these lines are distinct (i.e., that $\ell_x \neq \ell_y$ for $x \neq y$). Also, we do not assume that such lines exist for each point (i.e., that for every point there are lines passing thought it). Still, with overwhelmingly high probability, over the choice of R, the set R covers all points (i.e., each point resides on some line in R). This discussion leads to the following codeword test.

Construction 3.4 *Given oracle access to $w : R \to \Sigma$, which is supposedly a codeword of $\mathcal{C}^R$, the test proceeds as follows:*

1. *Pick $x \in F^m$ uniformly at random, and let $\ell_x \in R$ be an arbitrary line that passes through x. If no such line exists, halt with output 1* (representing accept).
2. *Pick $\ell \in R$ uniformly among the lines that pass through x.*

   That is, select $\ell \in R$ with probability $m_x(\ell)/t_x$, where $m_x(\ell')$ denotes the number of occurrences of x on the line ℓ', and $t_x = \sum_{\ell' \in R} m_x(\ell')$.
3. *Query w at ℓ_x and ℓ, and denote the answers by $h_x = w(\ell_x)$ and $h = w(\ell)$.*

   (Recall that h_x and h are univariate polynomials of degree d.)

[2] The (the multiplicative) Chernoff Bound (see, e.g., [29]) is extensively used in this work. It refers to independent random variables, denoted $\zeta_1, ...\zeta_n$, where each $\zeta_i \in [0,1]$. Letting $\zeta \stackrel{\text{def}}{=} \frac{1}{n} \cdot \sum_{i=1}^n \zeta_i$ denote the average of these random variables, and $p \stackrel{\text{def}}{=} \mathbf{E}[\zeta] = \frac{1}{n}\sum_i^n \mathbf{E}[\zeta_i]$ denote its expectation, the bound asserts that, for every $\gamma \in [0,1]$, the probability that ζ is not $(1\pm\gamma)\cdot p$ is exponentially vanishing in $\Omega(\gamma^2 pn)$. That is, $\Pr[|\zeta - p| > \gamma p] < 2\exp\left(-\gamma^2 pn/3\right)$.

[3] An alternative to the fixed canonical lines used below, is to use a random line passing through the point. This defines a randomized function, but the analysis can be applied to it just as well. Indeed, this would coincide with the "two-line test" analyzed (differently) in our preliminary report [24].

4. Point-vs-Line Test: *Let $\alpha, \beta \in F$ be such that $\ell_x(\alpha) = \ell(\beta) = x$. If $h_x(\alpha) = h(\beta)$ then halt with output 1. Otherwise halt with output 0.*

   (Note that ℓ_x is effectively a query to a point oracle, whereas ℓ is indeed a query to a line oracle.)

The line ℓ_x will be called the canonical line *of x.*

Note that the codeword test makes two queries to w (i.e., for $w(\ell_x)$ and $w(\ell)$). We analyze the codeword test next.

Analysis. It is obvious that the test accepts a valid codeword with probability 1. Below we give a lower bound on the rejection probability of non-codewords. As in Proposition 3.3, the lower bound holds for almost all choices of R of size $n = \Omega(m|F|^m \log |F|)$.

Lemma 3.5 *The following holds, for some $n = O(m|F|^m \log |F|)$: For at least a $1 - o(1)$ fraction of the possible choices of R of size n, every $w \in \Sigma^n$ is rejected by the codeword test* (of Construction 3.4) *with probability $\Omega(\delta_{\mathcal{C}^R}(w))$, where $\delta_{\mathcal{C}^R}(w)$ is the relative distance of w from the code $\mathcal{C}^R$* (i.e., *$\delta_{\mathcal{C}^R}(w) = \Delta_{\mathcal{C}^R}(w)/n$).*

The above lemma improves over the probability bound $\Omega(\delta_{\mathcal{C}^R}(w)) - o(1)$ that was established in our preliminary report [24] (for a related test). We mention that the fraction of exceptional sets in Lemma 3.5 can be bounded by $|F|^{-tm}$, where $t = n/\Theta(m|F|^m \log |F|)$.

Proof: We start with an overview of the proof. We consider two cases regarding a point function, denoted $f_w : F^m \to F$, determined by (some of) the entries of w (which are univariate polynomials supposedly representing the values of some polynomial when restricted to the corresponding lines). Specifically, we refer to the function $f_w : F^m \to F$ defined by setting $f_w(x)$ according to the value assigned by w to the canonical line ($\ell_x \in R$) that passes through x. We consider two cases regarding the distance of f_w from $P_{m,d}$:

1. The first case is that this relative distance is large (e.g., larger than one third of $\delta_{\mathcal{C}^R}(w)$). This case is handled by proving that, for all but a $o(1)$ of the choices of R, the Point-vs-Line Test rejects with probability that is linearly related to the the distance of f_w from $P_{m,d}$. (Note that the claim refers only to the portion of w that is used to define f_w, and holds regardless of the rest of w.)

   The proof of this claim (Claim 3.5.2) is the most interesting part of the current analysis. It amounts to showing that, for most choices of R, the modified (Point-vs-Line) low-degree test that selects lines in R performs as well as the original low-degree test (which selects lines in L). The proof relies on the following observations:

   (a) Each possible function $f : F^m \to F$ determines an *optimal* answer (i.e., a univariate polynomial) for each possible line-query, which in turn assigns each possible line-query a "rejection value" that is merely the fraction of points on the line for which the (optimal) answer disagrees with the value of f.

   (b) The rejection probability of the original low-degree test is linearly related to the average of these rejection values, *where the average is taken over all lines.*

   (c) The modified test refers to a (random) set of line-queries, and so its rejection probability is linearly related to the average of the aforementioned rejection values, *where the average is taken over the said set.*

The punch-line is that, for a random set (of adequate size), with overwhelmingly high probability, the average of values assigned to elements in the set approximates the average of all values. The error probability is sufficiently small to allow for the application of a (*non-straightforward*) union bound on all possible w's; see Step 2 in the proof of Claim 3.5.2.

2. The second case is that f_w is relatively close to $P_{m,d}$ (e.g., f_w is $\delta_{\mathcal{C}^R}(w)/3$-close to $P_{m,d}$). Suppose that the function f_w is actually a low-degree polynomial (i.e., $f_w \in P_{m,d}$). Still, the sequence of univariate polynomial representing the values of f_w on the lines in R may be different from the sequence w. This distance is "accounted for" by the fact that, for all but a $o(1)$ of the choices of R, the Point-vs-Line Test will cause rejection with probability that is linearly related to the distance of $\mathcal{C}^R(f_w)$ from w. The claim can be extended to the general case in which f_w is only close to $P_{m,d}$; for details see Claim 3.5.3.

We comment that, in (the first case of) our analysis, the function f_w is viewed as the primary object, and w is viewed as a potential proof of the claim $f_w \in P_{m,d}$. This perspective is not natural in the context of testing whether w is a codeword, because in the latter context w is the primary object and f_w is an auxiliary object. Still, this is a legitimate mental experiment. As for the analysis itself, we note that in the first case the testing features of the low-degree test are used in a natural way (because f_w is "far" from being a low-degree polynomial). Indeed, in this case we refer to the standard analysis of low-degree tests. In contrast, in the second case, the low-degree test is invoked in a non-standard situation (i.e., f_w is "close" to being a low-degree polynomial), and a straightforward analysis shows that the test will reject when the proof oracle (i.e., the line oracle) is "far" from being correct (i.e., being the restriction of f_w to R).

Turning to the actual proof, we present some notation first. As above, we view w as a function from R to Σ. We denote by $f_w : F^m \to F$ the function defined by the values assigned to points by their canonical lines; that is, $f_w(x) = v$ if the polynomial $h_x = w(\ell_x)$ assigns the value v to x, where ℓ_x is the canonical line passing through x (i.e., if $\ell_x(\alpha) = x$ then $v = h_x(\alpha)$). Let $p_w \in P_{m,d}$ denote the m-variate degree d polynomial closest to f_w (breaking ties arbitrarily). Let $\delta(w) = \delta_{\mathcal{C}^R}(w)$ be the (relative) distance of w from the code $\mathcal{C}^R$. In accordance with the motivational discussion, we consider the following auxiliary distances:

1. $\delta_{\rm ldp}(w)$ denotes the relative distance of f_w from p_w (or equivalently from $P_{m,d}$).

2. $\delta_{\rm agr}(w)$ denotes the relative distance between the values assigned by p_w to lines in R and w itself; that is, $\delta_{\rm agr}(w) = \Pr_{\ell \in R}[p_w(\ell) \neq w_\ell]$, where (as above) $p_w(\ell_{a,b})$ denotes the univariate polynomial in $z \in F$ that represents $p_w(a + zb)$.

Using this notation, we have

$$\delta_{\rm ldp}(w) \;=\; \frac{\Delta(f_w, p_w)}{|F^m|} \;\text{ and }\; \delta_{\rm agr}(w) \;=\; \frac{\Delta(w, \mathcal{C}^R(p_w))}{|R|} \tag{5}$$

Clearly, $\Delta_{\mathcal{C}^R}(w) \le \Delta(w, \mathcal{C}^R(p_w))$, and so $\delta_{\rm agr}(w) \ge \delta(w)$. In Claim 3.5.3, we will show that the tester (of Construction 3.4) rejects w with probability at least $(\delta_{\rm agr}(w)/2) - \delta_{\rm ldp}(w)$, which establishes the lemma in case $\delta_{\rm ldp}(w) \le \delta(w)/3$. On the other hand, in Claim 3.5.2, we will show that the tester (of Construction 3.4) rejects w with probability at least $\Omega(\delta_{\rm ldp}(w))$, which will take care of the case $\delta_{\rm ldp}(w) \ge \delta(w)/3$. Thus, either way, the lemma follows.

Before proving the aforementioned claims, we establish a useful fact regarding typical sets R. Specifically, we show that they cover all points almost-uniformly (see Claim 3.5.1). In particular, such sets will contain canonical lines for all points.

A tedious comment: Throughout this work, when we talk about the number of lines (resp., selecting a random line) in R that pass through a specific point x, we actually mean the number of pairs (resp., selecting a random pair) of the form $(\ell, e) \in R \times F$ such that $\ell(e) = x$. Thus, lines that contain multiple occurrences of a point are counted multiple times and are selected with greater probability. Indeed, the only lines containing multiple occurrences of a point are the constant lines, and the reader can safely ignore them (because R is unlikely to contain more than few such lines). Still, the rest of the analysis (like Step 2 of Construction 3.4), does refer to the general case (where constant lines occur and are dealt with using the above convention).

Claim 3.5.1 *For all but at most an $o(1)$ fraction of the possible choices of R, it holds that, for each point $x \in F^m$, there are $(1 \pm 0.1) \cdot |R|/|F|^{m-1}$ lines in R that pass through x.*

We mention that the constant 0.1 is quite arbitrary, and can be replaced by any other constant $\epsilon > 0$ (while effecting the hidden constant in $|R| = O(m|F|^m \log |F|)$).

Proof: For every fixed $x \in F^m$ and $e \in F$, we consider the number of lines $\ell \in R$ satisfying $\ell(e) = x$. The expected number of such lines, for a random R, is exactly $|R|/|F|^m$. Using Chernoff Bound (see Footnote 2), we infer that the probability that the number of such lines deviates from $(1 \pm 0.1) \cdot |R|/|F|^m$ is exponentially vanishing in $|R|/|F|^m = \Theta(m \log |F|)$. Thus, by a suitable choice of the latter constant, the aforementioned probability is $o(|F|^{-(m+1)})$, and using a union bound on all possible $x \in F^m$ and $e \in F$, the claim follows. ■

For the next claim, we rephrase the Point-vs-Line test in terms of the associated functions $f : F^m \to F$ and $g : R \to \Sigma$, where in our application $f = f_w$ and $g(\ell) = w(\ell)$ (for every $\ell \in R$). The test picks $x \in F^m$ uniformly at random and $\ell \in R$ uniformly among the lines passing through x. For β such that $\ell(\beta) = x$, it verifies that $h(\beta) = f(x)$, where h is the univariate polynomial $g(\ell)$. Let $\delta_{\rm ld}(f) = \delta_{P_{m,d}}(f)$ denote the relative distance of f from $P_{m,d}$. Indeed, $\delta_{\rm ld}(f_w) = \delta_{\rm ldp}(w)$.

Claim 3.5.2 *For all but at most an $o(1)$ fraction of the possible choices of R, the following holds: For every $f : F^m \to F$ and $g : R \to \Sigma$, the probability that the Point-vs-Line Test rejects the oracle pair (f, g) is at least $\Omega(\delta_{\rm ld}(f))$.*

In particular, we may conclude that our codeword test rejects any w with probability at least $\Omega(\delta_{\rm ldp}(w))$. Note that Claim 3.5.2 does not refer to the distance of g from being a "consistent" line-oracle (let alone one that corresponds to f). Thus, Claim 3.5.2 effectively refers to all possible g's (or rather to the best possible g) that may be paired with f.

Proof: We prove the claim in two steps. First, we fix $f : F^m \to F$ and prove that for all but $\exp(-\Omega(\delta_{\rm ld}(f) \cdot |R|))$ fraction of R's, the rejection probability of the test on input f and *any* $g : R \to \Sigma$ is $\Omega(\delta_{\rm ld}(f))$. Next, we use a union bound over an *appropriate collection of functions*, to prove that no function f is rejected with probability less than $\Omega(\delta_{\rm ld}(f))$. An interesting aspect of the second step is that we analyze the performance of the test on all functions by using a union bound only on a small fraction of the possible functions.

Step 1 – overview: Following [32, 2, 3, 30, 21], we observe that for each possible function $f : F^m \to F$ there exists an optimal strategy for answering all possible line-queries such that the acceptance probability of the point-vs-line test for oracle pairs $(f, \cdot)$ is maximized. Specifically, for a fixed function f, and each line ℓ, the optimal way to answer the line-query ℓ is given by the degree d univariate polynomial that agrees with the value of f on the maximum number of points of ℓ. Thus, the *optimal strategy for fooling* the point-vs-line test, when the point-oracle equals f, depends only on f *and not on the set of lines that may serve as possible queries.* Furthermore, the rejection probability of the point-vs-line test is the average of quantities (i.e., the agreements of f with the best univariate polynomials) that f associates with each of the possible lines. The latter fact holds not only when the test operates with the set of all lines, but also when it operates with any set of lines R (as in the claim).[4] The key observation is that for a random set R, with overwhelmingly high probability, the average over R of quantities associated with lines in R approximates the average over L of the same quantities.

Step 1 – details: Fix $f : F^m \to F$ an let $\delta = \delta_{\rm ld}(f)$ denote its distance to the nearest low-degree polynomial. Let us denote by $D_\ell(f)$ the fractional disagreement of f, when restricted to line ℓ, with the best univariate polynomial (i.e., the univariate polynomial of degree d that is nearest to $f|_\ell$ (i.e., f restricted to ℓ)). That is,

$$D_\ell(f) \stackrel{\rm def}{=} \min_{p\in P_{1,d}} \{\Pr_{e\in F}[f(\ell(e)) \neq p(e)]\}. \tag{6}$$

Indeed, a polynomial p achieving the minimum in Eq. (6) is an optimal answer to the line-query ℓ. Note that, on input oracles f and g, the rejection probability of the standard point-vs-line test (which refers to all possible lines), denoted $p_L(f,g)$, is lower-bounded by the average of the $D_\ell(f)$'s over all $\ell \in L$ (with equality holding if, for every line $\ell \in L$, it holds that $g(\ell)$ is a polynomial with maximal agreement with $f|_\ell$). A similar observation holds for the Point-vs-Line Test that refers to the set of lines R, except that now the average is taken over the lines in R. Actually, the average is weighted according to the probability that the test inspects the different lines (because a line is selected by uniformly selecting a point and then selecting a random line that passes through this point). Thus, the rejection probability of the Point-vs-Line Test that refers to the set R, denoted $p_R(f,g)$, is lower-bounded by the weighted average of the corresponding $D_\ell(f)$'s. Denoting the Point-vs-Line Test that selects lines in R by $\mathcal{T}_R$, we state the above fact for future reference:

$$p_R(f,g) \geq \sum_{\ell\in R} \Pr[\ell \text{ is selected by } \mathcal{T}_R] \cdot D_\ell(f). \tag{7}$$

Indeed, $p_L(f,g) \geq \sum_{\ell\in L} |L|^{-1} \cdot D_\ell(f)$ follows as a special case. Using the best-known analysis of the standard low-degree test (in particular, using [21, Thm. 7] to support the case that $|F| = O(d)$), we obtain that[5]

$$p_L(f,g) \geq \tau(f) \stackrel{\rm def}{=} |L|^{-1} \cdot \sum_{\ell\in L} D_\ell(f) = \Omega(\delta)\,. \tag{8}$$

Actually, we only care about the second inequality (i.e., $\tau(f) = \Omega(\delta)$, where $\delta = \delta_{\rm ld}(f)$). Now, when R is chosen at random (as a set of n lines from L), the expected value of

$$\tau_R(f) \stackrel{\rm def}{=} |R|^{-1} \cdot \sum_{\ell\in R} D_\ell(f) \tag{9}$$

[4] In the latter case, the average is taken according to the distribution on R that is induced by the test. Note that this distribution is not necessarily uniform over R.

[5] The inequality $|L|^{-1} \cdot \sum_{\ell\in L} D_\ell(f) = \Omega(\delta_{\rm ld}(f))$ is only implicit in most prior works, but it can also be inferred from the results that are stated explicitly in them. Specifically, these works only refer to the rejection probability of the standard test (for the best possible g), showing that $\min_{g:L\to P_{1,d}}\{p_L(f,g)\} = \Omega(\delta_{\rm ld}(f))$. (For example, [21, Thm. 7] asserts that $p_L(f,g) \geq \min(1/9, \delta_{\rm ld}(f)/2)$ for every f and g, provided $|F| = \Omega(d)$.) However, by the above discussion it is clear that, for the optimal line oracle, the rejection probability of the standard point-vs-line test equals the average of the $D_\ell(f)$'s; that is, for some $g_{\rm opt}$, which depends on f, it holds that $p_L(f, g_{\rm opt}) = |L|^{-1} \cdot \sum_{\ell\in L} D_\ell(f)$).

equals $\mathbf{E}_{\ell\in L}[D_\ell(f)] = \tau(f)$. By Chernoff Bound (see Footnote 2), we have that the probability that R is such that $\tau_R(f) < \tau(f)/2$ is exponentially small in $\delta|R|$. That is, for a random set R of n lines, it holds that

$$(\forall f) \qquad \Pr_R[\tau_R(f) < \tau(f)/2] \; < \; \exp\left(-\Omega(\delta_{\mathrm{ld}}(f)\cdot|R|)\right). \tag{10}$$

In the following two paragraphs we assume that R is such that $\tau_R(f) \geq \tau(f)/2$.

Let us assume that R covers all points uniformly; that is, each point resides on the same number of lines in R (where several appearances on the same line are counted several times). This implies that our test selects lines uniformly in R. Then, the rejection probability of our test (i.e., the point-vs-line test for lines uniformly selected in R), when applied to f and any g, is lower-bounded by the (unweighted) average of the $D_\ell(f)$'s over the lines in R (rather than over the set of all lines, L). It follows that $p_R(f,g) \geq \tau_R(f) \geq \tau(f)/2 = \Omega(\delta_{\mathrm{ld}}(f))$. (Recall that $p_R(f,g)$ denotes the rejection probability of the test that selects lines in R.)

In the previous paragraph we have assumed that R covers all points uniformly (i.e., each point resides on the same number of lines in R). In general, this may not be the case. Yet, with very high probability, a random set R covers *all* points in an *almost uniform* manner, and this "almost uniformity" suffices for extending the above analysis. Specifically, we first note that, with overwhelmingly high probability, each point in F^m resides on $(1\pm 0.1)\cdot|R|/|F|^{m-1}$ lines (see Claim 3.5.1). Next observe that in the above analysis we assumed that the test selects lines uniformly in R, whereas our test selects lines in R by selecting uniformly a point and then selecting a random line passing through this point. However, as formally shown in the next paragraph, for R as above (i.e., that covers all points "almost uniformly"), the distribution induced on the selected lines assigns each line in R a probability of $(1\pm 0.1)^{-1}/|R|$. Thus, the rejection probability may be skewed by a factor of $(1\pm 0.1)^{-1} = (1\pm 0.2)$ from the value $|R|^{-1}\cdot\sum_{\ell\in R} D_\ell(f) = \tau_R(f)$, which is analyzed above. We get $p_R(f,g) \geq 0.8\cdot\tau_R(f) \geq 0.4\cdot\tau(f) = \Omega(\delta_{\mathrm{ld}}(f))$. Using only the second inequality (which holds whenever R covers all points "almost uniformly") and referring to Claim 3.5.1, we state the following fact for future reference.

$$\Pr_R[(\forall f,g) \quad p_R(f,g) \geq 0.8\cdot\tau_R(f)] = 1 - o(1). \tag{11}$$

It is left to analyze the distribution induced on lines selected from a fixed R (by the aforementioned process), when R covers all points "almost uniformly". Recall that, for a point x, we denote by $m_x(\ell)$ the number of occurrences of x on the line ℓ, and by $t_x = \sum_{\ell\in R} m_x(\ell)$. Then, the probability that the *non-constant* line $\ell = (x_1, ..., x_{|F|}) \in R$ is selected equals

$$\sum_{i=1}^{|F|} \Pr[x_i \text{ is selected}]\cdot\frac{m_{x_i}(\ell)}{t_{x_i}} \;=\; |F|\cdot\frac{1}{|F|^m}\cdot\frac{1}{(1\pm 0.1)\cdot|R|/|F|^{m-1}}$$

which equals $(1\pm 0.1)^{-1}\cdot|R|^{-1}$ as claimed. Similarly, a *constant* line $\ell = (x, ..., x) \in R$ is selected with probability $\frac{1}{|F|^m}\cdot\frac{|F|}{(1\pm 0.1)\cdot|R|/|F|^{m-1}}$, which also satisfies the claim.

Step 2 – overview: Recall that we have bounded (in Eq. (10)) the fraction of R's for which $\tau_R(f) \geq \tau(f)/2$ does not hold for (any) *fixed* f. Our current *goal* is to show that, for most R's, it is the case that $\tau_R(f) \geq \tau(f)/2$ holds *for every* f. This suffices to complete the proof of the current claim, because we have shown in Step 1 (see Eq. (8) and Eq. (11), respectively) that $\tau(f) = \Omega(\delta_{\mathrm{ld}}(f))$ holds for all f and that (for most choices of R) it holds that $p_R(f,g) = \Omega(\tau_R(f))$ for every pair (f,g). The natural approach towards meeting our goal is taking a union bound over all f's that are δ-far from $P_{m,d}$ in order to upper bound the fraction of R's such that there exists a function

f that is δ-far from $P_{m,d}$ for which $\tau_R(f) < \tau(f)/2$. The problem is that the number of such functions is certainly greater than $|P_{m,d}| > \exp(\Omega(d/m)^m)$, whereas (for a random R) we only have $\Pr_R[\tau_R(f) < \tau(f)/2] < \exp(-\Omega(\delta|R|))$ (and in fact $\Pr_R[\tau_R(f) < \tau(f)/2] > \exp(-O(\delta|R|)))$. This is not a problem in case δ is any positive constant (or more generally if $\delta|R| > H_2(\delta)\cdot|F|^m + O(d/m)^m$), which in turn suffices to establish *weak* testability (as per Definition 2.1),[6] but we wish to handle the general case (in order to establish *strong* testability as per Definition 2.2). Thus, we cluster these functions according to the low-degree function that is closest to them, and show that it is enough to analyze one cluster (e.g., the one of the zero polynomial). The validity of the latter observation relies on properties of the set $P_{m,d}$ that imply that $D_\ell(f) = D_\ell(f+p)$ holds for every function f, polynomial $p \in P_{m,d}$ and line ℓ. The benefit in the said observation is that we need only consider the functions that are closest to some fixed polynomial and are δ-far from it (rather than all functions δ-far from $P_{m,d}$). Thus, we get an upper-bound of $|F|^{\delta|F|^m} \cdot \binom{|F|^m}{\delta|F|^m} \cdot \exp(-\Omega(\delta|R|)$, which is negligible (because $|R| \gg |F|^m \log|F|^m$).

Step 2 – details: For any fixed $\delta_0 > 0$, we start by considering the functions that are at relative distance exactly δ_0 from the zero polynomial. The number of such functions is at most

$$(|F|-1)^{\delta_0|F|^m} \cdot \binom{|F|^m}{\delta_0|F|^m} \;<\; (|F|^{m+1})^{\delta_0|F|^m} \;=\; \exp(\delta_0 \cdot (m+1)|F|^m \log|F|)\,. \tag{12}$$

On the other hand, by Eq. (10), for any function f, it holds that $\Pr_R[\tau_R(f) < \tau(f)/2] = \exp(-\Omega(\delta_{\rm ld}(f)\cdot|R|))$, and if this function is closest to the zero polynomial (i.e., $\Delta(f,0) = \Delta_{P_{m,d}}(f)$) then $\delta_{\rm ld}(f) = \delta_0$. Thus, using $|R| = c\cdot|F|^m\log|F|^m$ (for an adequate constant c), the probability (over the choices of R) that there *exists* a function f that is closest to the zero polynomial and is at relative distance exactly δ from it such that $\tau_R(f) < \tau(f)/2$ is upper-bounded by

$$\exp(\delta\cdot(m+1)|F|^m\log|F|)\cdot\exp(-\Omega(\delta\cdot|R|)) \;=\; \exp(-2\delta|F|^m\log|F|^m) \;<\; o(|F|^{-m})\,,$$

where the last inequality uses $\delta \geq 1/|F|^m$. Summing over all (the $|F|^m$) possible values of δ, we see that the probability over R, that there exists a function f that is closest to the zero polynomial (among all polynomials in $P_{m,d}$) such that $\tau_R(f) < \tau(f)/2$ is $o(1)$. Thus, we have

$$\Pr_R[\text{for every } f \text{ s.t. } \Delta(f,0) = \Delta_{P_{m,d}}(f) \text{ it holds that } \tau_R(f) \geq \tau(f)/2] = 1 - o(1) \tag{13}$$

To conclude the argument, we use properties of the set $P_{m,d}$. Specifically, suppose that R is such that for every function f' that is closest to the zero polynomial it holds that $\tau_R(f') \geq \tau(f')/2$. Now, consider an arbitrary function f and let $p \in P_{m,d}$ be the polynomial closest to f. Then, the function $f' = f - p$ is closest to the zero polynomial, and we claim that $\tau(f') = \tau(f)$ and $\tau_R(f') = \tau_R(f)$. These claims follow from the fact that, for every function f and every polynomial $p \in P_{m,d}$ and for every line ℓ, it holds that $D_\ell(f) = D_\ell(f+p)$ (although the polynomials selected to achieve the maximum agreement with f and $f+p$, over the line ℓ, may be different). Indeed, if q is used to achieve the maximum agreement with f over the line ℓ then $q + (p|_\ell)$ achieves the maximum agreement with $f+p$, where $p|_\ell$ is the univariate polynomial obtained by restricting the polynomial p to the line ℓ. Thus, for every function f and $p \in P_{m,d}$ that is closest to f, it holds that $\tau_R(f) = \tau_R(f-p)$ and $\tau(f-p) = \tau(f)$. Using Eq. (13), we get

$$\Pr_R[\tau_R(f) \geq \tau(f)/2 \text{ for every } f] = 1 - o(1)\,. \tag{14}$$

[6] Weak testability is all that was established in our preliminary report [24], and the stronger analysis that follows is new.

Combining Eq. (11) and Eq. (14), we get

$$\Pr_R[\forall (f,g)\;\; p_R(f,g) \geq 0.8\tau_R(f) \geq 0.4\tau(f)] = 1 - o(1).$$

Recalling Eq. (8), which asserts $\tau(f) = \Omega(\delta_{\rm ld}(f))$ for every f, the claim follows. ■

The last claim, which also relates to the Point-vs-Line Test, is also phrased in terms of the associated functions $f : F^m \to F$ and $g : R \to \Sigma$, where in our application $f = f_w$ and $g(\ell) = w(\ell)$ (for every $\ell \in R$). (When applied outside the context of this work, one should note that $\mathcal{C}^R(p)$ is the sequence of univariate polynomials representing the restriction of the polynomial p to all lines in R.)

Claim 3.5.3 *Let R be such that, for each point $x \in F^m$, there are $(1 \pm 0.1) \cdot |R|/|F|^{m-1}$ lines in R that pass through x. Then, for every $f : F^m \to F$ and $g : R \to \Sigma$, the probability that the Point-vs-Line Test rejects the oracle pair (f,g) is at least*

$$\frac{1}{2} \cdot \frac{\Delta(g, \mathcal{C}^R(p))}{|R|} - \frac{\Delta(f,p)}{|F^m|},$$

where p is the polynomial in $P_{m,d}$ that is closest to f.

Claim 3.5.3 will be applied to pairs (f_w, w), in which case $\Delta(w, \mathcal{C}^R(p_w)) = \delta_{\rm agr}(w) \cdot |R|$ and $\Delta(f_w, p_w) = \delta_{\rm ldp}(w) \cdot |F^m|$ (recalling that p_w is the polynomial closest to f_w). Consequently, we will infer that the codeword test reject any w with probability at least $(\delta_{\rm agr}(w)/2) - \delta_{\rm ldp}(w)$. Needless to say, Claim 3.5.3 will be invoked only in case $\delta_{\rm ldp}(w) < \delta_{\rm agr}(w)/2$.

Proof: We will first consider what happens when the test is invoked with oracle access to the pair (p,g), rather than to the pair (f,g). The claim will follow by observing that the test queries the point oracle on a single uniformly distributed point, and so replacing p by f may reduce the rejection probability by at most the relative distance between f and p.

As in the proof of Claim 3.5.2, we start by assuming that R covers all points uniformly (i.e., each point resides on the same number of lines in R). In this case, the test selects lines uniformly in R. Thus, with probability $\delta \stackrel{\rm def}{=} \Delta(g, \mathcal{C}^R(p))/|R|$, the test selects a line ℓ such that $h \stackrel{\rm def}{=} g(\ell)$ does not agree with p on ℓ. Now, since both h and $p|_\ell$ (i.e., the values of p restricted to the line ℓ) are degree d univariate polynomials (and since they disagree), they disagree on at least $|F| - d > 2|F|/3$ of the points on ℓ. Thus, the test will reject the oracle pair (p,g) with probability at least $(2/3) \cdot \delta$.

However, in general, R may not cover all points uniformly. Yet, the claim's hypothesis by which R covers all points "almost uniformly" suffices for extending the above analysis. Specifically (as shown in the proof of Claim 3.5.2), in this case each line is selected (by the test) with probability $(1 \pm 0.1)^{-1}/|R|$, and so the test rejects the oracle pair (p,g) with probability at least $0.8 \cdot (2\delta/3) > \delta/2$.

So far we have analyzed the behavior of the test with respect to the oracle pair (p,g), whereas we need to analyze the behavior with respect to the oracle pair (f,g). Recalling the test makes a single uniformly distributed query to the point oracle, it follows that test rejects the oracle pair (f,g) with probability at least $(\delta/2) - (\Delta(f,p)/|F|^m)$. The claim follows. ■

Completing the proof of Lemma 3.5: We call the set R good if it satisfies the conclusions of Claims 3.5.1 and 3.5.2. Thus, these claims assert that $1 - o(1)$ fraction of the possible choices of R are good, and we are going to fix such a good R for the rest of the discussion. Considering any $w \in \Sigma^n$, recall that $\delta_{\rm agr}(w) \geq \delta(w) \stackrel{\rm def}{=} \Delta_{\mathcal{C}^R}(w)/n$ and $\delta_{\rm ld}(f_w) = \delta_{\rm ldp}(w)$. If $\delta_{\rm ld}(f_w) \geq \delta(w)/3$ then invoking Claim 3.5.2 (with $f = f_w$ and $g = w$) we are done, because (for a good R) the test rejects with probability $\Omega(\delta_{\rm ld}(f))$, which in this case is $\Omega(\delta(w))$. Otherwise (i.e., $\delta_{\rm ld}(f_w) < \delta(w)/3$), invoking Claim 3.5.3 (with $f = f_w$ and $g = w$), we conclude that (for a good R) the test rejects with probability $(\delta_{\rm agr}(w)/2) - \delta_{\rm ld}(f_w) > \delta(w)/6$, because $\delta_{\rm agr}(w) \geq \delta(w)$. The lemma follows. ■

Remark 3.6 In continuation to Footnote 3, we note that the proof of Lemma 3.5 holds for any choice of a line ℓ_x that passes through x, including a probabilistic choice. In particular, Lemma 3.5 holds also in the case that Construction 3.4 is modified such that ℓ_x is selected uniformly among all lines that passes through x; that is, ℓ_x is selected identically to the way ℓ is selected in Step 2, which means that we select independently and uniformly two lines that pass through the random point x. The important fact about this modification is that both lines (i.e., the queries of the tester) are almost uniformly distributed in R, provided that R covers all points almost uniformly (which we assume and establish anyhow – see Claim 3.5.1). Specifically, each line in R is selected (as a query) with probability $(1 \pm 0.2)/|R|$. The constant 0.2 is rather arbitrary, and by using $|R| = O(\epsilon^{-2}m|F|^m \log|F|)$, we can ensure that each line in R is selected (as a query) with probability $(1 \pm \epsilon)/|R|$.

Corollary: Part 1 of Theorem 2.4. By the above, with probability $1 - o(1)$ over the choice of R, the code $\mathcal{C}^R : \Sigma^k \to \Sigma^n$ has relative constant distance and is locally-testable (using two queries). Furthermore, by Proposition 3.2, the code is F-linear where $\Sigma = F^{d+1}$. Using the first parameter-setting (i.e., $d = m^m$), we establish Part 1 of Theorem 2.4 (see Eq. (3)). In particular we establish that *for infinitely many k's, there exist two-query testable codes of constant relative distance over a non-binary alphabet Σ such that $n = \exp(\tilde{O}(\sqrt{\log k})) \cdot k = k^{1+o(1)}$ and* $\log|\Sigma| = \exp(\tilde{O}(\sqrt{\log k})) = k^{o(1)}$.

Remark 3.7 The above code $\mathcal{C}^R : \Sigma^k \to \Sigma^n$, where $\Sigma = F^{d+1}$, can be constructed only for specific values of k; that is, those given in Eq. (2) as a function of the parameters m and d. Furthermore, using $d = m^m$, we get a construction of any k that satisfies $k = k_1(m) \stackrel{\rm def}{=} \binom{m+m^m}{m}/(m^m+1) \approx m^{m^2-1}$ for some m. In this case $k_1(m+1) \approx \exp(\sqrt{\log k_1(m)}) \cdot k_1(m)$. Using $d = m^e$ for some $e > 1$, we get a construction of any k that satisfies $k = k_2(m) \stackrel{\rm def}{=} \binom{m+m^e}{m}/(m^e+1) \approx m^{(e-1)m}$ for some m. In this case $k_2(m+1) < (\log k_1(m))^e \cdot k_2(m)$.

3.3 Decreasing the alphabet size

The code $\mathcal{C}^R$ presented in Construction 3.1 uses quite a big alphabet (i.e., $\Sigma = F^{d+1}$, where $|F| = \Theta(d)$). Our aim, in this subsection, is to maintain the local-testability properties of $\mathcal{C}^R$ while using a smaller alphabet (i.e., F rather than F^{d+1}). This is achieved by concatenating $\mathcal{C}^R$ (which encodes information by a sequence of n univariate polynomials over F, each of degree d) with the following inner-code $\mathcal{C}'$ that maps F^{d+1} to $F^{n'}$, where n' is sub-exponential in $k' \stackrel{\rm def}{=} d+1$.

The inner-code: For a (suitable) constant integer d', let $k' = h^{d'}$. As a warm-up, consider the special case of $d' = 2$. In this case, the code $\mathcal{C}'$ maps bilinear forms in x_i's and y_i's (with coefficients $\langle c_{i,j} : i,j \in [h]\rangle$) to the values of these forms under all possible assignments. That is, $\mathcal{C}' : F^{h^2} \to F^{|F|^{2h}}$ maps the sequence of coefficients $\langle c_{i,j} : i,j \in [h]\rangle$ to the sequence of values $\langle v_{a_1,...,a_h,b_1,...,b_h} : a_1,...,a_h,b_1,...,b_h \in F\rangle$ where $v_{a_1,...,a_h,b_1,...,b_h} = \sum_{i,j\in[h]} c_{i,j} \cdot a_i b_j$. Viewing $\mathcal{C}'$ as a mapping from $F^{h^2} \times F^{2h}$ to F, we have $\mathcal{C}'((c_{1,1},...,c_{h,h}),(a_1,...,a_h,b_1,...,b_h)) = \sum_{i,j\in[h]} c_{i,j} \cdot a_i b_j$. In general (i.e., for an arbitrary integer $d' \geq 1$), the inner-code $\mathcal{C}'$:

$F^{k'} \to F^{n'}$ maps d'-multilinear forms in the variables sets $\{z_i^{(1)} : i \in [h]\}, ..., \{z_i^{(d')} : i \in [h]\}$ to the values of these d'-multilinear forms under all possible assignments to these $d'h$ variables. That is, $\mathcal{C}'$ maps the sequence of coefficients $\langle c_{i_1,...,i_{d'}} : i_1, ..., i_{d'} \in [h]\rangle$ to the sequence of values $\langle v_{a_1^{(1)},...,a_h^{(1)},...,a_1^{(d')},...,a_h^{(d')}} : a_1^{(1)}, ..., a_h^{(1)}, ..., a_1^{(d')}, ..., a_h^{(d')} \in F\rangle$ where $v_{a_1^{(1)},...,a_h^{(1)},...,a_1^{(d')},...,a_h^{(d')}} = \sum_{i_1,...,i_{d'} \in [h]} c_{i_1,...,i_{d'}} \cdot \prod_{j=1}^{d'} a_{i_j}^{(j)}$. Viewing $\mathcal{C}'$ as a mapping from $F^{h^{d'}} \times F^{d'h}$ to F, we have

$$\mathcal{C}'((c_{1,...,1}, ..., c_{h,...,h}), (a_1^{(1)}, ..., a_h^{(1)}, ..., a_1^{(d')}, ..., a_h^{(d')})) = \sum_{i_1,...,i_{d'} \in [h]} c_{i_1,...,i_{d'}} \cdot \prod_{j=1}^{d'} a_{i_j}^{(j)} \qquad (15)$$

Thus, ($k' = h^{d'}$ and) $n' = |F|^{d'h} = \exp(d' \cdot (k')^{1/d'} \cdot \log|F|)$. Using $|F| = O(k')$ and $d' = O(1)$, we have $n' = \exp(\tilde{O}(k')^{1/d'})$. Note that the inner-code has relative distance $(1 - (d'/|F|)) > 3/4$, assuming $|F| > 4d'$.

Testing the inner-code: A valid codeword (viewed as a function from $F^{d'h}$ to F) is a multilinear function (in the variable sets $\{z_i^{(1)} : i \in [h]\}, ..., \{z_i^{(d')} : i \in [h]\}$); that is, for each j, a valid codeword is linear in the variables $z_i^{(j)}$'s. Thus, testing whether $w : F^{d'h} \to F$ belongs to the inner-code reduces to d' linearity checks. Specifically, for each j, we randomly select $\overline{r} = (r_1^{(1)}, ..., r_h^{(1)}, ..., r_1^{(d')}, ..., r_h^{(d')}) \in F^{d'h}$, $s^{(j)} = (s_1^{(j)}, ..., s_h^{(j)}) \in F^h$ and $e \in F$, and check whether or not $w(\overline{r}', r^{(j)}, \overline{r}'') + e \cdot w(\overline{r}', s^{(j)}, \overline{r}'') = w(\overline{r}', r^{(j)} + e \cdot s^{(j)}, \overline{r}'')$, where $\overline{r}' = (r_1^{(1)}, ..., r_h^{(1)}, ..., r_1^{(j-1)}, ..., r_h^{(j-1)})$, $r^{(j)} = (r_1^{(j)}, ..., r_h^{(j)})$, and $\overline{r}'' = (r_1^{(j+1)}, ..., r_h^{(j+1)}, ..., r_1^{(d')}, ..., r_h^{(d')})$. In addition, we also let the test employ a total low-degree test (to verify that the codeword is a multi-variate polynomial of total-degree d').[7] The total-low-degree test uses $d' + 2$ queries, and so our codeword test uses $3d' + (d' + 2) = O(d')$ queries. For sake of clarity, we provide an explicit statement of the resulting test.

Construction 3.8 *Given oracle access to $w : F^{d'h} \to F$, which is supposedly a codeword of $\mathcal{C}'$, the test proceeds as follows:*

The linearity tests: *For $j = 1, ..., d'$, we test whether w is linear in the j^{th} block of variables. That is, for uniformly selected $\overline{r}' = (r_1^{(1)}, ..., r_h^{(1)}, ..., r_1^{(j-1)}, ..., r_h^{(j-1)}) \in F^{(j-1)h}$ and $\overline{r}'' = (r_1^{(j+1)}, ..., r_h^{(j+1)}, ..., r_1^{(d')}, ..., r_h^{(d')}) \in F^{(d'-j)h}$, we test whether the resulting function $w_{\overline{r}',\overline{r}''}(z_1, ..., z_h) \stackrel{\text{def}}{=} w(\overline{r}', z_1, ..., z_h, \overline{r}'')$ is linear* (in $z_1, ..., z_h$).

The linearity of $w_{\overline{r}',\overline{r}''} : F^h \to F$ is tested using a BLR-type test [14], specifically the Extended Linearity Test of Kiwi [28, P. 10]. We select uniformly $r'_1, ..., r'_h, s'_1, ..., s'_h, e, f \in F$, and accept if and only if $e \cdot w_{\overline{r}',\overline{r}''}(r'_1, ..., r'_h) + f \cdot w_{\overline{r}',\overline{r}''}(s'_1, ..., s'_h) = w_{\overline{r}',\overline{r}''}(e \cdot r'_1 + f \cdot s'_1, ..., e \cdot r'_h + f \cdot s'_h)$.

The (auxiliary) low-degree test: *Following the low-degree test of [2], we select uniformly $\overline{a} = (a_1^{(1)}, ..., a_h^{(1)}, ..., a_1^{(d')}, ..., a_h^{(d')}) \in F^{d'h}$ and $\overline{b} = (b_1^{(1)}, ..., b_h^{(1)}, ..., b_1^{(d')}, ..., b_h^{(d')}) \in F^{d'h}$ and test whether there exists a univariate polynomial p of degree d' such that $w(\overline{a} + e \cdot \overline{b})$ agrees with $p(e)$ on every $e \in F$, where $\overline{a} + e \cdot \overline{b} = (a_1^{(1)} + eb_1^{(1)}, ..., a_h^{(1)} + eb_h^{(1)}, ..., a_1^{(d')} + eb_1^{(d')}, ..., a_h^{(d')} + eb_h^{(d')})$. Specifically, for fixed and distinct $\eta_0, ..., \eta_{d'} \in F$, we check whether the univariate degree d' polynomial $p(\zeta)$ defined such that $p(\eta_i) = w(\overline{a} + \eta_i \cdot \overline{b})$, for $i = 0, 1, .., d'$, agrees with*

[7] We believe that the codeword test operates well also without employing the total-degree test, but the augmented codeword test is certainly easier to analyze.

$w(\overline{a}+\zeta\cdot\overline{b})$ on a random point; that is, we uniformly select $\eta \in F$ and accept if and only if $p(\eta) = w(\overline{a}+\eta\cdot\overline{b})$.

We accept if and only if all $d'+1$ tests accept.

We note that the interpolation condition used for low-degree testing is linear in the recovered values. Thus, Construction 3.8 checks $d'+1$ linear conditions, where each of the first d' conditions involves three values of w, and the remaining condition involves $d'+2$ such values. Clearly, Construction 3.8 accepts every codeword of $\mathcal{C}'$ with probability 1. The following lemma provides a lower bound on the rejection probability of non-codewords.

Lemma 3.9 *Every $w' \in F^{n'}$ is rejected by the codeword test of Construction 3.8 with probability $\Omega(\delta_{\mathcal{C}'}(w'))$, where $\delta_{\mathcal{C}'}(w')$ is the relative distance of w' from the code $\mathcal{C}'$* (i.e., $\delta_{\mathcal{C}'}(w') = \Delta_{\mathcal{C}'}(w')/n'$).

Proof: Let $\delta = \delta_{\mathcal{C}'}(w')$, and let δ' denote the relative distance of w' (viewed as a function $w' : F^{d'h} \to F$) from the set of $d'h$-variate polynomials of total degree d'. (Indeed, $\delta' \leq \delta$, because every codeword (i.e., a d'-multilinear function) is a polynomial of total degree d'.) If $\delta' \geq \min(\delta, 0.4)$ then w is rejected with probability $\Omega(\delta')$ by the total-degree test (cf., e.g., [2, Lem. 7.2.1.4]), and the lemma follows (because $\delta' \geq \delta$). Specifically, the analysis in [2] shows that the restriction of w on a random line is expected to be $\Omega(\delta')$-far from any univarite polynomial polynomial of degree d', which in particular applies to the polynomial obtained by interpolation based on the points $\eta_0, ..., \eta_{d'}$.

Otherwise (i.e., $\delta' < \min(\delta, 0.4)$), let p' denote the degree d' polynomial closest to w'. By the case hypothesis (i.e., $\delta' < \delta$) this p' must be non-linear in some block of variables (otherwise $\delta = \delta_{\mathcal{C}'}(w') \leq \Delta(w', p')/n' = \delta'$); that is, for some j, the polynomial p' is non-linear in $\{z_i^{(j)} : i \in [h]\}$. We claim that, with probability at least $1-(d'/|F|) > 0.9$, this non-linearity is preserved when assigning *random* values to the variables of *all the other blocks*; that is, *for a random* $\overline{r} = (\overline{r}^{(1)}, ..., \overline{r}^{(d')}) \in (F^h)^{d'}$, *with probability at least* 0.9, *the polynomial* $p'_{\overline{r}}(\overline{z}^{(j)}) \stackrel{\rm def}{=} p'(\overline{r}^{(1)}, ..., \overline{r}^{(j-1)}, \overline{z}^{(j)}, \overline{r}^{(j+1)}, ..., \overline{r}^{(d')})$ *is not linear in* $\overline{z}^{(j)} = (z_1^{(j)}, ..., z_h^{(j)})$. The claim is proved by writing p' as the sum of monomials in $\overline{z}^{(j)}$ with coefficients being functions of the other variables. Consider any non-linear monomial in $\overline{z}^{(j)}$ having a non-zero coefficient. This non-zero coefficient is a polynomial of degree at most $d'-2$ in the other variables (because p' has total degree d' and the said monomial is non-linear in $\overline{z}^{(j)}$). Then, by the Schwarz–Zippel Lemma, with probability at least $1-((d'-2)/|F|)$, a random assignment $\overline{r}$ to the other variables will yield a non-zero value, and thus this (non-linear) monomial in $\overline{z}^{(j)}$ will appear in $p'_{\overline{r}}$ (with a non-zero coefficient).

Furthermore, in this case, the non-linear polynomial $p'_{\overline{r}}$ (which also has degree at most d') is at distance $1-(d'/|F|) > 0.9$ from any linear function (in $\overline{z}^{(j)}$). Thus, for a random $\overline{r}$, the *expected* relative distance between $p'_{\overline{r}}$ and the set of linear functions is greater than $0.9 \cdot 0.9 > 0.8$. On the other hand, the *expected* relative distance between the residual w' and p' (i.e., between $w'_{\overline{r}}$ and $p'_{\overline{r}}$) *under the random assignment* $\overline{r}$ is $\delta' < 0.4$ (where the inequality is due to the case hypothesis). Thus, under such random assignment, the expected fractional distance of the residual w' (i.e., $w'_{\overline{r}}$) from the set of linear functions (in $\{z_i^{(j)} : i \in [h]\}$) is greater than $0.8 - 0.4 = 0.4$. It follows that w' is rejected with constant probability by the $j^{\rm th}$ linearity test (because, with probability at least 0.2, the residual w' is at least 0.2-far from being linear, and so is rejected with constant probability [28, Lemma 4.4]). ■

The concatenated-code: We apply the code-concatenation paradigm (cf. [20]) to the codes $\mathcal{C} = \mathcal{C}^R$ (of Construction 3.1) and $\mathcal{C}'$ (of Eq. (15)). The concatenated-code obtained by composing the outer-code $\mathcal{C} : \Sigma^k \to \Sigma^n$ with the inner-code $\mathcal{C}' : F^{k'} \to F^{n'}$, where $\Sigma = F^{d+1} = F^{k'}$, maps $(x_1, ..., x_k)$ to $(\mathcal{C}'(y_1), ..., \mathcal{C}'(y_n))$, where $(y_1, ..., y_n) \stackrel{\text{def}}{=} \mathcal{C}(x_1, ..., x_k)$. In other words, for $x \in \Sigma^k \equiv F^{k \cdot k'}$, we have:

$$\text{concatenated-code}(x) = \mathcal{C}'(\mathcal{C}(x,1)), ..., \mathcal{C}'(\mathcal{C}(x,n)) \tag{16}$$

where here we view $\mathcal{C}$ as a mapping from $\Sigma^k \times [n]$ to Σ. Thus, the concatenated-code maps $k \cdot k'$-long sequences over F to $n \cdot n'$-long sequences over F. Furthermore, since both $\mathcal{C}$ and $\mathcal{C}'$ are F-linear, the concatenated-code is F-linear; that is, for each i, each F-symbol in the sequence $\mathcal{C}'(y_i)$ is a linear combination of the F-symbols in $y_i = \mathcal{C}(x,i) \in F^{k'}$, which in turn are linear combinations of the F-symbols in $(x_1, ..., x_k) \in F^{k \cdot k'}$.

Testing the concatenated-code: Loosely speaking, in order to test the concatenated code, we first test whether its n blocks are codewords of the inner-code, and next use "self-correction" (cf. [14]) on these blocks to emulate the testing of the outer-code. Specifically, the tester for the concatenated code first selects at random (as the tester of the outer-code) two intersecting lines ℓ' and ℓ'', and applies the inner-code tester (of Construction 3.8) to the inner-encoding of the polynomials associated with these two lines (by the outer-code). Next, to emulate the actual check of the outer-code test (of Construction 3.4), the current tester needs to obtain the values of these two polynomials at some elements of F (which are determined by the outer test). Suppose that we need the value of q' (a univariate polynomial of degree $d = h^{d'} - 1$ over F) at $t \in F$, and that q' is encoded by the inner-code. Recall that q' is represented as a sequence of coefficients $(q'_0, ..., q'_d)$. For sake of the inner-code, this sequence may be viewed as indexed by d'-tuples over $[h]$ such that the index $(i_1, ..., i_{d'}) \in [h]^{d'}$ corresponds to $\sum_{j=1}^{d'} (i_j - 1) \cdot h^{j-1} \in \{0, 1, ..., d\}$; that is, $q'_{i_1,...,i_{d'}}$ is the coefficient of the $\sum_{j=1}^{d'} (i_j - 1) \cdot h^{j-1}$-th power. Thus (under this convention), $q'(z) = \sum_{i_1,...,i_{d'} \in [h]} q'_{i_1,...,i_{d'}} \cdot z^{\sum_{j=1}^{d'} (i_j - 1) \cdot h^{j-1}}$, which in turn (using Eq. (15)) yields the following key observation:

$$q'(t) \;=\; \mathcal{C}'(\langle q'_{i_1,...,i_{d'}} : i_1, ..., i_{d'} \in [h] \rangle, (t^0, ..., t^{h-1}, t^0, ..., t^{(h-1)h}, ..., t^0, ..., t^{(h-1)h^{d'-1}})). \tag{17}$$

That is, $q'(t)$ resides in the entry of $\mathcal{C}'(\langle q'_{i_1,...,i_{d'}} : i_1, ..., i_{d'} \rangle)$ that is indexed by $\bar{t} \in F^{d'h}$, where the i^{th} entry in $\bar{t}$ is $t^{(i-1 \bmod h) \cdot h^{\lfloor (i-1)/h \rfloor}}$. But since this specific entry of the inner-code may be corrupted (in a noisy codeword), we recover it by self-correction based on few *random* positions in the codeword. Specifically, self-correction of the desired entry is performed via polynomial interpolation, and requires only $d' + 1$ queries (where each query is uniformly distributed). This discussion leads to the following test, where we are assuming (for simplicity) that the lines in R cover all points of F^m.

Construction 3.10 *Given oracle access to $w : R \times [h^{d'}] \to F$, which is supposedly a codeword of the concatenated-code* (of $\mathcal{C}^R$ and $\mathcal{C}'$), *the test proceeds as follows:*

1. *As in Construction 3.4, we pick $x \in F^m$ uniformly at random, and let $\ell_x \in R$ be an arbitrary line that passes through x. Pick $\ell \in R$ uniformly among the lines that pass through x.*

2. Testing the inner-code: *Apply Construction 3.8 to the residual oracles $w(\ell_x, \cdot)$ and $w(\ell, \cdot)$.*

3. Emulating the Point-vs-Line Test (i.e., the remaining steps of Construction 3.4): *Let $\alpha, \beta \in F$ be such that $\ell_x(\alpha) = \ell(\beta) = x$. Let q_x and q be the polynomials encoded* (possibly with noise) *in $w(\ell_x, \cdot)$ and $w(\ell, \cdot)$, respectively. We obtain the values of $q_x(\alpha)$ and $q(\beta)$, via self-correction, and check whether these two values are equal.*

   Self-correction *of $q(\beta)$ is performed as follows. Setting*

$$\overline{a} = (1, ..., \beta^{h-1}, 1..., \beta^{(h-1)h}, 1..., \beta^{(h-1)h^2}, ..., 1, ..., \beta^{(h-1)h^{d'-1}})$$

   we select uniformly $\overline{b} = (b_1^{(1)}, ..., b_h^{(1)}, ..., b_1^{(d')}, ..., b_h^{(d')}) \in F^{d'h}$, obtain the values $w(\ell, \overline{a} + \eta \cdot \overline{b})$ for $d'+1$ distinct non-zero values $\eta \in F$, and compute the desired value by polynomial interpolation. That is, for fixed and distinct $\eta_1, ..., \eta_{d'} \in F \setminus \{0\}$, we determine the univariate polynomial p of degree d' satisfying $p(\eta_i) = w(\ell, \overline{a} + \eta_i \cdot \overline{b})$ for $i = 1, .., d'+1$, and take $p(0)$ as the value of $q(\beta)$. Note that, by Eq. (17), $q(\beta)$ equals $\mathcal{C}'(\langle q_{i_1,...,i_{d'}} : i_1, ..., i_{d'} \in [h]\rangle, \overline{a})$, and that we have accessed a slightly noise version of $\mathcal{C}'(\langle q_{i_1,...,i_{d'}} : i_1, ..., i_{d'} \in [h]\rangle)$ at $d'+1$ uniformly distributed positions. Self-correction of $q_x(\alpha)$ is performed analogously.

We accept if and only if both invocation of Construction 3.8 as well as the emulated Point-vs-Line Test accept.

Note that the tester performs $2 \cdot (4d' + 2) + 2 \cdot (d' + 1) = O(d')$ queries. Furthermore, its checks amount to checking several linear conditions regarding the retrieved values. (The latter fact follows from the linearity of the check performed by Construction 3.8 and the linearity of the interpolation performed in Step 3.) Clearly, Construction 3.10 accepts each codeword with probability 1, but lower-bounding the rejection probability of non-codewords does require a detailed analysis (which is provided next). The point is to prove that the "composition of tests" (for the concatenation-code) does work as one would have expected.

Lemma 3.11 *Let F and $\mathcal{C} = \mathcal{C}^R$ be as in Construction 3.1, and $\mathcal{C}' : F^{k'} \to F^{n'}$ be as in Eq.* (15), *where $k' = h^{d'}$ and $n' = |F|^{d'h}$. Then, for all but a $o(1)$ fraction of the possible choices of R, every $w \in F^{nn'}$ is rejected by the concatenated-code tester* (of Construction 3.10) *with probability that is linearly related to the distance of w from the concatenated-code* (of Eq. (16))*.*

Proof: We fix any set R satisfying the claims of Lemma 3.5 and Claim 3.5.1. That is, the code $\mathcal{C}^R$ is locally testable (via Construction 3.4), and R covers all points almost-uniformly. (Recall that indeed all but a $o(1)$ fraction of the possible choices of R can be used here.) For this fixed R, we analyze the performance of Construction 3.10 with respect to the corresponding concatenated-code (of Eq. (16)).

Fixing any $w = (w_1, ..., w_n) \in (F^{n'})^n$, let us denote by δ the relative distance of w from the concatenated-code, and let $\delta_i \stackrel{\text{def}}{=} \Delta_{\mathcal{C}'}(w_i)/n'$ denote the relative distance of w_i from the inner-code $\mathcal{C}'$. Throughout this proof (unless stated differently), distances refers to sequences over F.

Recall that each of the two lines selected by the outer-code tester (i.e., the tester of Construction 3.4) is not uniformly distributed in $[n] \equiv R$. It is rather the case that the first line is the canonical line associated with a uniformly selected point, whereas the second line is a selected uniformly among the lines (in R) that passes through this point. Let us denote by p_i and q_i the corresponding distribution on lines; that is, p_i (resp., q_i) denotes the probability that the i-th line in R is selected as a canonical line (resp., a random line) for a uniformly selected point. Note that, by the almost uniformity condition, it holds that $q_i = 1/(1 \pm 0.1)n$ for every $i \in [n]$. For a constant $c > 1$ (to be determined), we consider the following two cases:

Case 1: either $\sum_{i=1}^{n} p_i\delta_i > \delta/c$ or $\sum_{i=1}^{n} q_i\delta_i > \delta/c$. In this case, at least one of the two blocks (i.e., either w_{ℓ_x} or w_ℓ) probed by the outer-code tester is at expected relative distance at least δ/c from the inner-code. Thus, in this case, the inner-code tester (of Construction 3.8, as analyzed in Lemma 3.9) invoked in Step 2 rejects with probability $\Omega(\delta/c)$, which is $\Omega(\delta)$ because c is a constant.

Case 2: both $\sum_{i=1}^{n} p_i\delta_i \leq \delta/c$ and $\sum_{i=1}^{n} q_i\delta_i \leq \delta/c$. In particular, using $q_i = 1/(1 \pm 0.1)n$, it follows that $\frac{1}{n}\sum_{i=1}^{n}\delta_i < 2\delta/c$. Denoting the closest corresponding $\mathcal{C}'$-codewords by c_i's (i.e., $\Delta(w_i, c_i) = \delta_i \cdot n'$), we let d_i denote the decoding of c_i (and of w_i). Then, denoting the concatenated code by $\mathcal{CC}$ (and viewing $d_i \in \Sigma = F^{d+1}$ as a single symbol but $c_i = \mathcal{C}'(d_i)$ and w_i as n'-long sequences (over F)), we have

$$\begin{aligned}\frac{\Delta_{\mathcal{C}}(d_1, ..., d_n)}{n} &\geq \frac{\Delta_{\mathcal{CC}}(\mathcal{C}'(d_1), ..., \mathcal{C}'(d_n))}{nn'}\\ &\geq \frac{\Delta_{\mathcal{CC}}(w_1, ..., w_n)}{nn'} - \frac{\Delta((w_1, ..., w_n), (c_1, ..., c_n))}{nn'}\\ &= \delta - \frac{1}{n}\sum_{i=1}^{n}\delta_i\end{aligned}$$

which is greater than $\delta-(2\delta/c) > \delta/2$ (using $\sum_{i=1}^{n}\delta_i/n < 2\delta/c$ and assuming $c \geq 4$). Thus, for some *constant* $c' > 0$ (determined in Lemma 3.5), the outer-code test rejects $(d_1, ..., d_n)$ with probability at least $c' \cdot \delta/2$. (We will set $c = 16(d'+1)/c' > 4$.) The question is what happens when the concatenated-code tester (given access to $(w_1, ..., w_n)$) emulates the outer-code test.

Recall that, in the current case, *both* the indices probed by the outer-code tester correspond to w_i's that are at expected relative distance at most δ/c from the inner-code, where each expectation is taken over the distribution of the corresponding index. Thus, for each of the two indices, with probability at most $2(d'+1)\delta/c$, the randomly selected index corresponds to a block w_i that is at relative distance greater than $p \stackrel{\text{def}}{=} 1/2(d'+1)$ from the inner-code. It follows that, with probability at least $1 - 2 \cdot (2(d'+1)\delta/c)$, *both* indices probed by the outer-code tester correspond to w_i's that are at relative distance at most p from the inner-code. In this case, with probability at least $(1-(d'+1)\cdot p)^2 = 1/4$, all actual (random) probes made in Step 3 to the inner-code are to locations in which the corresponding w_i and $c_i = \mathcal{C}'(d_i)$ agree, and thus both the self-corrected values (computed by our test) will match the corresponding d_i's. Note that if the above two events occur then our tester correctly emulates the outer-code tester. Thus, our tester rejects if the following three events occur:

1. The outer-code tester would have rejected the two answers (i.e., the two d_i's).
2. The two probed indices correspond to w_i's that are at relative distance at most $1/2(d'+1)$ from the inner-code (and in particular from the corresponding $\mathcal{C}'(d_i)$'s, which are the $\mathcal{C}'$-codewords closest to them).
3. The self-corrected values match the corresponding d_i's.

By the above, Event 1 occurs with probability at least $c'\delta/2$, and Event 2 fails with probability at most $4(d'+1)\delta/c = c'\delta/4$ (by setting $c = 16(d'+1)/c'$). Thus, our tester rejects w with probability at least $((c'\delta/2) - (c'\delta/4)) \cdot (1/4) = \Omega(\delta)$, where the 1/4 is due to the probability that Event 3 occurs (conditioned on Events 1 and 2 occuring).

Thus, in both cases, any word that is at relative distance δ from the concatenated-code is rejected with probability $\Omega(\delta)$. The lemma follows. ∎

Other properties: Recall that the concatenated code, mapping $F^{kk'}$ to $F^{nn'}$ is linear (over F). Furthermore, the codeword test is a conjunction of $O(d')$ linear tests. Alternatively, we may perform one of these linear tests, selected at random (with equal probability). The relative distance of the concatenated code is the product of the relative distances of the outer and inner codes, and thus is a constant. Regarding the parameters of the concatenated code, suppose that in the outer-code we use the setting $d = m^e$ (for any constant $e > 1$), and that in the inner-code we use $d' = 2e$. Then, we obtain a code that maps $F^{kk'}$ to $F^{nn'}$, where $n < k^{(e+o(1))/(e-1)}$ and $k' = d + 1 < |F| < (\log k)^e$ (both by Eq. (4)), and $n' = \exp(\tilde{O}(d^{1/d'}))$ (see Eq. (15)) which in turn equals $\exp(\tilde{O}((\log k)^{e/d'})) = \exp(\tilde{O}(\sqrt{\log k})) = k^{o(1)}$. Thus,

$$nn' = (kk')^{(e+o(1))/(e-1)} \text{ and } |F| < (\log k)^e. \tag{18}$$

For usage in the next subsection, we only care that the alphabet size (i.e., $|F|$) is $k^{o(1)}$, while the rate is good (i.e., $nn' \approx (kk')^{e/(e-1)}$).

Remark 3.12 The code $\mathcal{C}' : F^{k'} \to F^{n'}$ can be constructed only for specific values of k'; that is, $k' = h^{d'}$ for some integers h and d'. Thus, fixing any constant integer d', we obtain codes for every d'-th (integer) power. Recall that when using this code together with $\mathcal{C} : \Sigma^k \to \Sigma^n$, where $\Sigma = F^{d+1}$, to derive the concatenated code we must set $k' = d + 1$. Actually, we go the other way around: Starting with any h, we set $k' = h^{d'}$ and $d = k' - 1$, and determine k as a function of d (and the parameter $m < d$) according to Eq. (2).

3.4 Obtaining a binary locally-testable code

Our last step is to derive a binary code. This is done by concatenating the code presented in Section 3.3 with the Hadamard code, while assuming that $F = \mathrm{GF}(2^{k''})$. That is, the Hadamard code is used to encode elements of F by binary sequences of length $n'' \stackrel{\text{def}}{=} 2^{k''}$. (Recall that the Hadamard encoding of a string $s \in \{0,1\}^\ell$ is given by the sequence all 2^ℓ partial sums (mod 2) of the bits of s.)

To test the newly concatenated code, we combine the obvious testing procedure for the Hadamard code with the fact that all that we need to check for the current outer-code are (a constant number of) linear (in F) conditions involving a constant number of F-entries. (Recall that Construction 3.10 only checks linear constraints, and that we are going to set d' to be a constant.) Now, instead of checking such a linear condition over F, we check that the corresponding equality holds for a random sum of the bits in the representation of the elements of F (using the hypothesis that $F = \mathrm{GF}(2^{k''})$). Specifically, suppose that we need to check whether $\sum_{i=1}^t \alpha_i a_i = 0$ (in F), for some known $\alpha_1, ..., \alpha_t \in F$ and oracle answers denoted by $a_1, ..., a_t \in F$. Then, we uniformly select $r \in \mathrm{GF}(2^{k''})$, and check whether $\mathrm{IP}_2(r, \sum_{i=1}^t \alpha_i a_i) \equiv 0 \bmod 2$ holds, where $\mathrm{IP}_2(u, v)$ denotes the inner-product modulo 2 of (the $\mathrm{GF}(2^{k''})$ elements) u and v (viewed as k''-bit long vectors). The latter check is performed by relying on the following two facts:

Fact 1: $\mathrm{IP}_2(r, \sum_{i=1}^t \alpha_i a_i) \equiv \sum_{i=1}^t \mathrm{IP}_2(r, \alpha_i a_i) \bmod 2$.
This fact holds because $\mathrm{IP}_2(r_1 \cdots r_{k''}, s_1 \cdots s_{k''}) = \sum_{j=1}^{k''} r_j s_j$.

Fact 2: Each $\mathrm{IP}_2(r, \alpha_i a_i)$ can be obtained by making a single query (which is determined by r and α_i) to the Hadamard coding of a_i, because $\mathrm{IP}_2(r, \alpha_i a_i)$ is merely a linear combination of the bits of a_i with coefficients depending on α_i and r (i.e., $\mathrm{IP}_2(r, \alpha_i a_i) = \mathrm{IP}_2(f(r, \alpha_i), a_i)$, where f is determined by the irreducible polynomial representing the field $\mathrm{GF}(2^{k''})$).

This fact holds because each bit of $\alpha_i a_i \in \mathrm{GF}(2^{k''})$ is a linear combination of the bits of a_i with coefficients depending on α_i, and $\mathrm{IP}_2(r,v)$ is a linear combination of the bits of v with coefficients depending on r.

We now turn to the actual construction of the final (binary) code. Recall that we wish to apply the code-concatenation paradigm to the code presented in Section 3.3 and the suitable Hadamard code. Specifically, let $\mathcal{C}^{\mathrm{out}} : F^{kk'} \to F^{nn'}$ denote the former code, and let $\mathcal{C}'' : \{0,1\}^{k''} \to \{0,1\}^{n''}$ denote the suitable Hadamard code, where $F = \mathrm{GF}(2^{k''}) \equiv \{0,1\}^{k''}$ and $[n''] \equiv \{0,1\}^{k''}$. (The parameter d' that determines the rate of $\mathcal{C}^{\mathrm{out}}$ as well as the query complexity of its codeword tester will be set to a constant.) Then, concatenating these two codes, we obtain a code that maps $(x_1,...,x_{kk'}) \in (\{0,1\}^{k''})^{kk'}$ to $(\mathcal{C}''(y_1),...,\mathcal{C}''(y_{nn'}))$, where $(y_1,...,y_{nn'}) = \mathcal{C}^{\mathrm{out}}(x_1,...,x_{kk'})$. In other words, for $x \in F^{kk'} \equiv \{0,1\}^{kk'\cdot k''}$, we have:

$$\text{concatenated-code}(x) = \mathcal{C}''(\mathcal{C}^{\mathrm{out}}(x,1)),....,\mathcal{C}''(\mathcal{C}^{\mathrm{out}}(x,nn')) \tag{19}$$

where $\mathcal{C}''(y) = \langle \mathrm{IP}_2(y,p) : p \in \{0,1\}^{|y|} \rangle$.

Loosely speaking, in order to test the concatenated code, we first test (random instances of) the inner-code (i.e., $\mathcal{C}''$), and next use "self-correction" (cf. [14]) on the latter to emulate the testing of the outer-code (i.e., $\mathcal{C}^{\mathrm{out}}$). Setting d' to a constant, the query complexity of the codeword tester of $\mathcal{C}^{\mathrm{out}}$ is a constant, denoted q (because $q = O(d')$). Recall that the codeword tester of $\mathcal{C}^{\mathrm{out}} : F^{kk'} \to F^{nn'}$ (i.e., Construction 3.10) checks a constant number of linear conditions, each depending on a constant number of positions (i.e., F-symbols). By uniformly selecting one of these conditions, we obtain a tester, denoted $\mathcal{T}$, that randomly selects q positions in the tested word and checks a single linear condition regarding the F-symbols in these positions. (Indeed, the non-codeword detection probability probability of $\mathcal{T}$ may be q times smaller than that of Construction 3.10.) Thus, the tester of the (new) concatenated code invokes $\mathcal{T}$ to determine q random locations $i_1,...,i_q \in [nn']$ and a linear condition $(\alpha_1,...,\alpha_q) \in F^q$ to be checked (on the corresponding answers). The (new) tester next checks whether the corresponding q blocks in the tested ($nn'n''$-bit long) string are codewords of $\mathcal{C}''$. Finally, the tester emulates the check $\sum_{j=1}^q \alpha_j d_{i_j} = 0$ of $\mathcal{T}$, where d_{i_j} is the $\mathcal{C}''$-decoding the i_j^{th} block of the tested string. This emulation is performed via self-correction, to be discussed next.

Recall that, rather than checking $\sum_{j=1}^q \alpha_j d_{i_j} = 0$, we are going to check $\mathrm{IP}_2(r, \sum_{j=1}^q \alpha_j d_{i_j}) = 0$, for a uniformly selected $r \in \{0,1\}^{k''}$. Furthermore, by Fact 1, rather then checking $\mathrm{IP}_2(r, \sum_{j=1}^q \alpha_j d_{i_j}) = 0$, we may check $\sum_{j=1}^q \mathrm{IP}_2(r, \alpha_j d_{i_j}) = 0$. To this end, we should obtain $\mathrm{IP}_2(r, \alpha_j d_{i_j})$, for r and α_j that are known to us. As stated in Fact 2, the desired bit can be expressed as a linear combination (with coefficients depending only on r and α_j) of the bits of d_{i_j}. That is, $\mathrm{IP}_2(r, \alpha_j d_{i_j}) = \mathrm{IP}_2(r_j, d_{i_j})$, where r_j is determined by r and α_j (i.e., $r_j = f(r, \alpha_i)$, where f depends on the representation of $\mathrm{GF}(2^{k''})$). Recall that $\mathrm{IP}_2(r_j, d_{i_j}) = \mathcal{C}''(d_{i_j}, r_j)$. However, since we may not have a valid codeword of d_{i_j}, we obtain the corresponding entry via self-correction of the $i_j{}^{\mathrm{th}}$ block of the tested string. That is, we obtain a good guess for $\mathcal{C}''(d_{i_j}, r_j)$, by taking the exclusive-or of positions $r_j \oplus s_j$ and s_j in that block, for a uniformly selected $s_j \in \{0,1\}^{k''} \equiv [n'']$. This discussion leads to the following test, where we add Step 4 to ease the analysis.

Construction 3.13 *The tester is given oracle access to $w = (w_1,....,w_{nn'})$, where each $w_i = w_{i,1} \cdots w_{i,n''} \in \{0,1\}^{n''}$, and proceeds as follows:*

1. *The tester selects the locations $i_1,...,i_q \in [nn']$ and the linear condition $(\alpha_1,...,\alpha_q) \in F^q$ to be checked by the codeword tester of $\mathcal{C}^{\mathrm{out}}$. That is, these choices are determined by invoking $\mathcal{T}$.*

2. *For $j = 1, ..., q$, the tester checks that w_{i_j} is a codeword of $\mathcal{C}''$. For each j, this is done by uniformly selecting $r, s \in \{0,1\}^{k''}$, and checking whether $w_{i_j,r} + w_{i_j,s} = w_{i_j,r\oplus s}$.*

3. *The tester emulates the check $\sum_{j=1}^{q} \alpha_j d_{i_j} = 0$ of the tester for $\mathcal{C}^{\rm out}$, where d_{i_j} is the $\mathcal{C}''$-decoding of w_{i_j} and the arithmetic is over $F = \mathrm{GF}(2^{k''})$. This is done by uniformly selecting $r \in \{0,1\}^{k''}$, and checking that $\mathrm{IP}_2(r, \sum_{j=1}^{q} \alpha_j d_{i_j}) = 0$. Actually, we test the equivalent condition $\sum_{j=1}^{q} \mathrm{IP}_2(r, \alpha_j d_{i_j}) = 0$, where the values $\mathrm{IP}_2(r, \alpha_j d_{i_j})$, for $j = 1, ..., q$, are obtained via self-correction as follows.*

   For the uniformly selected $r \in \{0,1\}^{k''}$, we determine $r_1, ..., r_q$ based on r and $\alpha_1, ..., \alpha_q$ (where the α_j's are as determined in Step 1). *That is, for each j, we determine $r_j = f(r, \alpha_j)$ such that $\mathrm{IP}_2(r_j, x) = \mathrm{IP}_2(r, \alpha_j x)$ holds for any value of $x \in \mathrm{GF}(2^{k''})$. Next, we select uniformly $s_1, ..., s_q \in \{0,1\}^{k''}$, and check that $\sum_{j=1}^{q} (w_{i_j,r_j\oplus s_j} - w_{i_j,s_j}) = 0$.*

4. *The tester selects uniformly $i_0 \in [nn']$, and checks that w_{i_0} is a codeword of $\mathcal{C}''$* (by uniformly selecting $r, s \in \{0,1\}^{k''}$, and checking whether $w_{i_0,r} + w_{i_0,s} = w_{i_0,r\oplus s}$).

We output 1 if and only if all $q + 2$ checks are satisfied.

Our tester makes $3(q+1) + 2q$ to the code, where q is a constant. It is clear that this tester accepts any valid codeword (because $\mathcal{C}''(y, r \oplus s) = \mathcal{C}''(y, r) + \mathcal{C}''(y, s)$ for every $y, r, s \in \{0,1\}^{k''}$). The analysis of the rejection probability of non-codewords can be carried out analogously to Lemma 3.11.

Lemma 3.14 *For $F = \mathrm{GF}(2^{k''})$, let $\mathcal{C}^{\rm out} : F^{kk'} \to F^{nn'}$ be as in Eq.* (16), *and $\mathcal{C}'' : \{0,1\}^{k''} \to \{0,1\}^{n''}$ be the Hadamard code, where $n'' = 2^{k''}$. Then, every $w \in \{0,1\}^{nn'n''}$ is rejected by the concatenated-code tester* (of Construction 3.13) *with probability that is linearly related to the distance of w from the concatenated-code* (of Eq. (19)).

Proof: Recall that $\mathcal{C}^{\rm out}$ is locally testable (by Lemma 3.11), and furthermore that the test $\mathcal{T}$, which makes a constant number of queries, rejects every non-codeword with probability that is linearly related to its distance from $\mathcal{C}^{\rm out}$.

Fixing any $w = (w_1, ..., w_{nn'}) \in (\{0,1\}^{n''})^{nn'}$, let us denote by δ the relative distance of w from the concatenated-code, and let $\delta_i \stackrel{\rm def}{=} \Delta_{\mathcal{C}''}(w_i)/n''$ denote the relative distance of w_i from the inner-code $\mathcal{C}''$. Throughout this proof (unless stated differently), distances refers to binary sequences.

As in the proof of Lemma 3.11, we distinguish between two cases according to the average distance of the w_i's from valid codewords of $\mathcal{C}''$. However, rather than relying on the specifics of $\mathcal{T}$, we use a more generic approach here, while relying on the added test performed in Step 4. Specifically, let us denote by p_i the probability that a random query of $\mathcal{T}$ probes the $i^{\rm th}$ location, where $i \in [nn']$ (and we refer to a uniformly selected query among the q random queries made by $\mathcal{T}$). For a constant $c > 1$ (to be determined), we consider the following two cases:

Case 1: either $\sum_{i=1}^{nn'} p_i \delta_i > \delta/c$ or $\sum_{i=1}^{nn'} \delta_i/nn' > \delta/c$. In this case, at least one of the blocks (i.e., either w_{i_j} for some $j \in [q]$ or w_{i_0}) probed by the outer-code tester (in either Step 2 or Step 4, respectively) is at expected relative distance at least δ/c from the inner-code. Thus, in this case, the inner-code tester (which is the extensively analyzed BLR-test [14]) rejects with probability $\Omega(\delta/c) = \Omega(\delta)$.

Case 2: both $\sum_{i=1}^{nn'} p_i \delta_i \leq \delta/c$ and $\sum_{i=1}^{nn'} \delta_i/nn' \leq \delta/c$. Denoting the closest corresponding $\mathcal{C}''$-codewords by c_i's (i.e., $\Delta(w_i, c_i) = \delta_i \cdot n''$), we let d_i denote the decoding of c_i (and of

w_i). Thus, the relative distance of $(d_1,...,d_{nn'})$ from the outer-code (when viewing each d_i as a single symbol) is at least $\delta - (\delta/c) > \delta/2$, provided that $c > 4$, where the δ/c term accounts for the average relative distance between the c_i's and the w_i's. That is, for every $x \in F^{kk'}$ the fraction of i's such that $d_i \neq \mathcal{C}^{\rm out}(x,i)$ is at least $\delta/2$. It follows that, for some *constant* $c' > 0$ (determined in Lemma 3.11), the outer-code test $\mathcal{T}$ rejects $(d_1,...,d_{nn'})$ with probability at least $c' \cdot \delta/2$. (We will set $c = 12q^2/c' > 4$.) The question is what happens when the concatenated-code tester (given access to w) emulates $\mathcal{T}$. The answer is that our tester rejects if the following four events all occur.

1. The outer-code tester $\mathcal{T}$ would have rejected the q answers (i.e., the relevant d_i's). That is, $\sum_{j=1}^{q} \alpha_j d_{i_j} \neq 0$ (over $\mathrm{GF}(2^{k''})$), for the adequate α_j's.
   Recall that this event occurs with probability at least $c'\delta/2$.
2. The q probed indices correspond to w_i's that are at relative distance at most $1/3q$ from the inner-code (and in particular from the corresponding $\mathcal{C}''(d_i)$'s).
   To see that this event occurs with probability at least $1 - 3q^2\delta/c$, let b_j denotes the probability for the bad (complementary) (sub-)event in which the $j^{\rm th}$ query is made to a block w_i that is $1/3q$-far from the corresponding codeword $\mathcal{C}''(d_i)$. Then, $\sum_{i=1}^{nn'} \delta_i/nn' \geq \frac{1}{q} \cdot \sum_{j=1}^{q} b_j \cdot (1/3q)$, which using the case hypothesis implies that $\sum_{j=1}^{q} b_j \leq 3q^2 \cdot \delta/c$, as claimed.
3. The self-corrected values match the corresponding d_i's.
   Given Event 2, the current event occurs with probability at least $(1 - 2\cdot(1/3q))^q > 1/3$, because self-correction succeeds whenever all actual (random) probes to the inner-code are to locations in which the corresponding w_i and $c_i = \mathcal{C}''(d_i)$ agree.
4. The string $r \in \{0,1\}^{k''}$, selected uniformly in Step 3, is such that $\sum_{j=1}^{q} \mathrm{IP}_2(r, \alpha_j d_{i_j}) \neq 0$,
   Given Event 1, the current event occurs with probability $1/2$.

Note that the first three events are analogous to events considered in the proof of Lemma 3.11, whereas the last event is introduced because we do not emulate the actual check of $\mathcal{T}$ (but rather a randomized version of it). Setting $c = 12q^2/c'$, we infer that all four events occur with probability at least $((c'\delta/2) - (3q^2\delta/c)) \cdot (1/3) \cdot (1/2) = c'\delta/24 = \Omega(\delta)$.

Thus, in both cases, any word that is at relative distance δ from the concatenated-code is rejected with probability $\Omega(\delta)$. The lemma follows. ■

Corollary: Part 2 of Theorem 2.4. For any desired constant $e > 1$, we use the parameter setting $d = m^e$ and $d' = 2e$ in the construction of the code $\mathcal{C}^{\rm out}$. As summarized in Eq. (18), this yields a code $\mathcal{C}^{\rm out} : F^{kk'} \to F^{nn'}$, where $nn' < (kk')^{(e+o(1))/(e-1)}$ and $|F| < (\log k)^e$. Recall that we compose $\mathcal{C}^{\rm out}$ with $\mathcal{C}'' : \{0,1\}^{k''} \to \{0,1\}^{n''}$, where $\{0,1\}^{k''}$ is associated with $F = \mathrm{GF}(2^{k''})$. Thus, our final code maps $\{0,1\}^{kk'k''}$ to $\{0,1\}^{nn'n''}$, where $n'' = 2^{k''} = |F| = \mathrm{poly}(\log k) = k^{o(1)}$, and so $nn'n'' < (kk'k'')^{(e+o(1))/(e-1)}$. Also note that the final code is linear and has linear distance. Thus, we have established Part 2 of Theorem 2.4.

Remark 3.15 The performance of the final codeword tester (of Construction 3.13) depends on the parameter $e > 1$, which determines the rate of the final code (i.e., the relation between $nn'n''$ and $kk'k''$). The query complexity of the tester is linear in e, and the rejection probability of non-codewords depends is inversely proportional to $\mathrm{poly}(e)$ (i.e., a string that is δ-far from the code is rejected with probability $\Omega(\delta/e^4)$). The rejection probability of non-codewords can be improved, but we doubt that one get get below $\Omega(\delta/e^2)$ without introducing significantly different ideas.

Remark 3.16 In continuation to Remarks 3.7 and 3.12, we comment that the final binary code can be constructed only for specific values of k, k' and k''. Fixing any integer $e > 1$, the aforementioned code can be constructed for any integer h, while setting $k' = h^e$, $k'' = \log O(k')$ and $k \approx (m^{e-1})^m$, where $m = (h^e - 1)^{1/e} \approx h$. Thus, $K \stackrel{\text{def}}{=} kk'k'' \approx h^{(e-1)h} \cdot h^e \cdot \log h^e \approx h^{(e-1)h}$. The ratio between consecutive admissible values of K is given by $\frac{(h+1)^{(e-1)(h+1)}}{h^{(e-1)h}} = O(h)^{e-1} < (\log K)^{e-1}$, and so the admissible successor of K is smaller than $(\log K)^{e-1} \cdot K$.

4 PCPs of Nearly-Linear Length

In this section we give a probabilistic construction of nearly-linear sized PCPs for SAT. More formally, we reduce SAT in *almost-linear probabilistic time* to a promise problem, and show that this problem has a PCP of randomness complexity $(1 + o(1)) \log n$ (on inputs of length n) and constant query complexity. Furthermore, this PCP has perfect completeness, soundness arbitrarily close to $\frac{1}{2}$, and its query complexity is a small explicit constant. Specifically, with 19 (bit) queries we obtain randomness complexity $\log_2 n + \tilde{O}(\sqrt{\log n})$. Recall that actually we care about the proof length (i.e., the length of the PCP oracle), which is $2^r \cdot q$, where r and q are the randomness and query complexities of the PCP. Our PCPs improve over the parameters of the PCPs constructed by Polishchuk and Spielman [30], and are obtained by applying the "random projection" method (introduced in Section 3) to certain *constant-prover one-round proof systems*, which are crucial ingredients in the constructions of PCPs. Specifically, we apply this technique to (a variant of) the three-prover one-round proof system of Harsha and Sudan [25].

Random projection of proof systems. Typically, constant-prover one-round proof systems use provers of very different sizes. Indeed, this is the case with the proof system of Harsha and Sudan [25]. By applying the "random projection" method to the latter proof system, we obtain an equivalent system in which all provers have size roughly equal to the size of the smallest prover in the original scheme. At this point, we reduce the randomness complexity to be logarithmic in the size of the provers (i.e., and thus logarithmic in the size of the smallest original prover). (The latter step is rather straightforward.) Starting with the system of [25], we obtain a constant-prover one-round proof system of randomness complexity $(1+o(1)) \log n$ (on inputs of length n). However, the query complexity of the resulting system is not constant, although it is small, but the standard proof composition paradigm (combined with known PCPs) comes to our rescue.

Recall that typical PCP constructions are obtained by using the technique of proof composition introduced by Arora and Safra [3]. In this technique, an "outer verifier", typically a verifier for a constant-prover one-round proof system, is composed with an "inner verifier" to get a new PCP verifier. The new verifier essentially inherits the randomness complexity of the outer verifier and the query complexity of the inner verifier. Since our goal is to reduce the randomness complexity of the composed verifier, we achieve this objective by reducing the randomness complexity of the outer verifier.

Organization. As stated above, our key step is to reduce the sizes of the provers (in certain constant-prover one-round proof system). As a warm-up (in Section 4.1), we first show that the random projection method can be applied to any 2-prover one-round proof system, resulting in an equivalent proof system in which both provers have size roughly equal to the size of the smallest prover in the original scheme.

Next, in Section 4.2, we show how to apply the random projection to the verifier of a specific 3-prover one-round proof system used by Harsha and Sudan [25]. Their verifier is a variant of the one constructed by Raz and Safra [31] (see also, Arora and Sudan [4]), which are, in turn, variants of a verifier constructed by Arora *et al.* [2]. All these verifiers share the common property of working with provers of vastly different sizes. We manage to reduce the sizes of all the provers to the size of the smallest one, and consequently reduce the randomness of the verifier to $(1+o(1))\log n$ (where n is the input length). We stress that the "random size reduction" step is not generic, but rather relies on properties of the proof of soundness in, say, [25], which are abstracted below. Applying known composition lemmas (i.e., those developed in [25]) to this gives us the desired short PCP constructions.

4.1 Two-prover verifiers and random sampling

We start by defining a 2-prover 1-round proof system as a combinatorial game between a verifier and two provers. Below, Ω denotes the space of verifier's coins, q_i denotes its strategy of forming queries to the i-th prover, and P_i denotes a strategy for answering these queries (where we refer to the residual strategy for a fixed common input, which is omitted from the notation).

Definition 4.1 *For finite sets Q_1, Q_2, Ω, and A, a (Q_1, Q_2, Ω, A)-*2IP verifier *V is given by functions $q_1 : \Omega \to Q_1$ and $q_2 : \Omega \to Q_2$ and* $\text{Verdict} : \Omega \times A \times A \to \{0,1\}$. *The value of V, denote $w(V)$, is the maximum, over all functions $P_1 : Q_1 \to A$ and $P_2 : Q_2 \to A$ of the quantity*

$$w_{P_1,P_2}(V) \stackrel{\text{def}}{=} \mathop{\mathbf{E}}_{r\in\Omega}\left[\text{Verdict}(r, P_1(q_1(r)), P_2(q_2(r)))\right]. \tag{20}$$

where r is uniformly distributed in Ω. A 2IP verifier V is said to be uniform *if, for each $i \in \{1,2\}$, the function $q_i : \Omega \to Q_i$ is $|\Omega|/|Q_i|$-to-one. The* size of prover *P_i is defined as $|Q_i|$.*

Focusing on the case $|Q_2| \gg |Q_1|$, we define a "sampled" 2IP verifier. In accordance with the preliminary motivational discussion, we use a two-stage sampling: first, we sample the queries to the bigger prover (i.e., $S \subset Q_2$), and then we sample the set of (relevant) coin tosses (i.e., $T \subset \Omega$). Thus, the first step corresponds to a random projection of the second prover's strategy on a subset of the possible queries. Note that the first stage results in a proof system in which the second prover has size $|S|$ (rather than $|Q_2|$), and that we should restrict the space of the resulting verifier's coins such that their image under q_2 equals S.

Definition 4.2 *Given a (Q_1, Q_2, Ω, A)-2IP verifier V and set $S \subseteq Q_2$, let*

$$\Omega_S = \{r \in \Omega : q_2(r) \in S\}. \tag{21}$$

For $T \subseteq \Omega_S$, the (S,T)-sampled 2IP verifier, *denoted $V|_{S,T}$, is a (Q_1, S, T, A)-2IP verifier given by functions $q_1' : T \to Q_1$, $q_2' : T \to S$, and* $\text{Verdict}' : T \times A \times A \to \{0,1\}$ *obtained by restricting q_1, q_2 and* Verdict *to T.*

In the following lemma we show that a sufficiently large randomly sampled set S from Q_2 is very likely to approximately preserve the value of a verifier. Furthermore, the value continues to be preserved approximately if we pick T to be a sufficiently large random subset of Ω_S.

Lemma 4.3 *There exist absolute constants c_1, c_2 such that the following holds for every Q_1, Q_2, Ω, A, ϵ and $\gamma > 0$. Let V be an (Q_1, Q_2, Ω, A)-uniform 2IP verifier.*

Completeness: *For any S and T, the (S,T)-sampled verifier preserves the perfect completeness of V. That is, if $\omega(V)=1$ then, for every $S \subseteq Q_2$ and $T \subseteq \Omega_S$, it holds that $\omega(V|_{S,T})=1$.*

Soundness: *For sufficiently large S and T, a random (S,T)-sampled verifier preserves the soundness of V up-to a constant factor. Specifically, let $N_1 = \frac{c_1}{\epsilon} \cdot \left(|Q_1| \log |A| + \log \frac{1}{\gamma}\right)$ and $N_2 = \frac{c_2}{\epsilon} \cdot \left(N_1 \log |A| + \log \frac{1}{\gamma}\right)$, and suppose that S is a uniformly selected multi-set of size N_1 of Q_2, and T is a uniformly selected multi-set of size N_2 of Ω_S. Then, for $\omega(V) \leq \epsilon$, with probability at least $1-\gamma$, it holds that $\omega(V|_{S,T}) \leq 2\epsilon$.*

Note that the reduction in the randomness complexity (i.e., obtaining $N_2 = \tilde{O}(|Q_1|)$) relies on the shrinking of the second prover to size $N_1 = \tilde{O}(|Q_1|)$. Without shrinking the second prover, we would obtain $N_2 = \tilde{O}(|Q_2|)$, which is typically useless (because, typically, $|\Omega| = \tilde{O}(|Q_2|)$).

Proof: We focus on the soundness condition, and assume that $\omega(V) \leq \epsilon$. The proof is partitioned into two parts. First we show that a random choice of S is unlikely to increase the value of the game to above $\frac{3}{2} \cdot \epsilon$. Next, assuming that S satisfies the latter condition, we show that a random choice of T is unlikely to increase the value of the game above 2ϵ. The second part of the proof is really a standard argument, which has been observed before in the context of PCPs (e.g., in [8]). We thus focus on the first part, which abstracts the idea of the random projection from Section 3.

Our aim is to bound the value $\omega(V|_{S,\Omega_S})$, for a randomly chosen S. Fix any prover strategy $P_1 : Q_1 \to A$ for the first prover. Now, note that an optimal strategy, denoted P_2^*, for the second prover answer each question $q_2 \in Q_2$ by an answer that maximizes the acceptance probability with respect to the fixed P_1 (i.e., an optimal answer is a string a_2 that maximizes $\mathbf{E}_{r \in \Omega | q_2(r)=q_2}[\mathrm{Verdict}(r, P_1(q_1(r)), a_2)]$). We stress that this assertion holds both for the original 2IP verifier V as well as for any (S,Ω_S)-sampled verifier.[8] For every question $q_2 \in Q_2$, let ϵ_{q_2} denote the acceptance probability of the verifier V given that the second question is q_2 (i.e., $\epsilon_{q_2} = \mathbf{E}_{r \in \Omega | q_2(r)=q_2}[\mathrm{Verdict}(r, P_1(q_1(r)), P_2^*(q_2))]$). By (uniformity and) the definition of ϵ_{q_2}, we have $\mathbf{E}_{q_2 \in Q_2}[\epsilon_{q_2}] = \mathbf{E}_{r \in \Omega}[\epsilon_{q_2(r)}] \leq \epsilon$. The quantity of interest to us is $\mathbf{E}_{r \in \Omega_S}[\epsilon_{q_2(r)}]$, which by uniformity equals $\mathbf{E}_{q_2 \in S}[\epsilon_{q_2}]$. A straightforward application of Chernoff Bound (see Footnote 2) shows that the probability that this quantity exceeds $\frac{3}{2} \cdot \epsilon$ is exponentially vanishing in ϵN_1. Taking the union bound over all possible P_1's, we infer that the probability that there exists a P_1, P_2 such that $\mathbf{E}_{r \in \Omega_S}[\mathrm{Verdict}(r, P_1(q_1(r)), P_2(q_2(r)))] > \frac{3}{2} \cdot \epsilon$ is at most $\exp(-\epsilon N_1) \cdot |A|^{|Q_1|}$. Thus, using $N_1 = \frac{c_1}{\epsilon}\left(|Q_1| \log |A| + \log \frac{1}{\gamma}\right)$ for some absolute constant c_1, it follows that $\omega(V|_{S,\Omega_S}) \leq \frac{3}{2} \cdot \epsilon$ with probability at least $1 - \frac{\gamma}{2}$ (over the choices of S). The lemma follows.[9] ∎

4.2 Improved 3-prover proof system for NP

We now define the more general notion of a constant-prover one-round interactive proof system (MIP). We actually extend the standard definition from languages to promise problems (cf. [18] and [8]).

[8] But, the assertion does not hold for most (S,T)-sampled verifiers.

[9] Indeed, we have ignored the effect of sampling Ω_S; that is, the relation of $\omega(V|_{S,\Omega_S})$ and $\omega(V|_{S,T})$, for a random $T \subseteq \Omega_S$ of size N_2. As stated above, this part is standard. Fixing any S such that $\omega(V|_{S,\Omega_S}) \leq \frac{3}{2} \cdot \epsilon$, we assume without loss of generality that $\omega(V|_{S,\Omega_S}) \geq \epsilon$. First, we fix any choice of $P_1 : Q_1 \to A$ and $P_2 : S \to A$, and applying Chernoff Bound (again) we infer that the probability that the restrictions of Ω_S to T lead to acceptance with probability greater than $\frac{4}{3} \cdot \omega(V|_{S,\Omega_S})$ is $\exp(-\Omega(\epsilon N_2))$. Taking the union bound over all choices of P_1 and P_2, we infer that $\omega(V|_{S,T}) > \frac{4}{3} \cdot \omega(V|_{S,\Omega_S})$ with probability at most $\exp(-\Omega(\epsilon N_2)) \cdot |A|^{|Q_1|+|S|}$. Thus, using $N_2 = \frac{c_2}{\epsilon}(|S| \log |A| + \log(1/\gamma))$, we conclude that $\omega(V|_{S,T}) \leq \frac{4}{3} \cdot \omega(V|_{S,\Omega_S}) \leq 2\epsilon$ with probability at least $1 - \frac{\gamma}{2}$ (over the choices of T).

Definition 4.4 *For positive reals c, s, integer p and functions $r, a : \mathsf{Z}^+ \rightarrow \mathsf{Z}^+$, we say that a promise problem $\Pi = (\Pi_{\rm YES}, \Pi_{\rm NO})$ is in* $\mathrm{MIP}_{c,s}[p, r, a]$ (or, Π has a p-prover one-round proof system with randomness r and answer length a) *if there exists a probabilistic polynomial-time verifier V interacting with p provers $P_1, \ldots, P_p$ such that*

Operation: *On input x of length n, the verifier tosses $r(n)$ coins, generates queries $q_1, \ldots, q_p$ to provers $P_1, ..., P_p$, obtain the corresponding answers $a_1, \ldots, a_p \in \{0,1\}^{a(n)}$, and outputs a Boolean verdict that is a function of x, its randomness and the answers $a_1, \ldots, a_p$.*

Completeness: *If $x \in \Pi_{\rm YES}$ then there exist strategies $P_1, \ldots, P_p$ such that V accepts their response with probability at least c.*

If $c = 1$ then we say that V has perfect completeness.

Soundness: *If $x \in \Pi_{\rm NO}$ then for every sequence of prover strategies $P_1, \ldots, P_p$, machine V accepts their response with probability at most s, which is called the* soundness error.

If for every choice of verifier's coins, its queries to P_i reside in a set Q_i, then we say that prover P_i has size $|Q_i|$.

Recall that a language L is captured by the promise problem $(L, \{0,1\}^* \setminus L)$.

Harsha and Sudan [25] presented a randomness efficient 3-prover one-round proof system for SAT, with answer length $\mathrm{poly}(\log n)$. Specifically, their proof system has randomness complexity $(3+\epsilon)\log_2 n$, where $\epsilon > 0$ is an arbitrary constant and n denotes the length of the input. Here we reduce the randomness required by their verifier to $(1+o(1))\log n$. Actually, we do not reduce the randomness complexity of the proof system for SAT, but rather present a randomized reduction of SAT to a problem for which we obtain a 3-prover one-round proof system with answer length $\mathrm{poly}(\log n)$ and randomness complexity $(1+o(1))\log n$. It is, of course, crucial that our reduction does not increase the length of the instance by too much. To capture this condition, we present a quantified notion of length preserving reductions.

Definition 4.5 *For a function $\ell : \mathsf{Z}^+ \rightarrow \mathsf{Z}^+$, a reduction is ℓ*-length preserving *if it maps instances of length n to instances of length at most $\ell(n)$.*

Our key technical result is summarized as follows.

Theorem 4.6 (Random Projection of certain MIPs): *Let $m, \ell : \mathsf{Z}^+ \rightarrow \mathsf{Z}^+$ be functions satisfying $\ell(n) = \Omega(m(n)^{\Omega(m(n))} n^{1+\Omega(1/m(n))})$ and $m(n) \geq 2$. Then, for any constant $\epsilon > 0$, SAT reduces in probabilistic polynomial time, under ℓ-length preserving reductions, to a promise problem Π in $\mathrm{MIP}_{1,\epsilon}[3, r, a]$, where $r(n) = (1 + 1/m(n))\log n + O(m(n)\log m(n))$ and $a(n) = m(n)^{O(1)} n^{O(1/m(n))}$.*

We comment that the reduction actually runs in time ℓ. Before proving Theorem 4.6, let us see a special case of it obtained by setting $m(n) = \sqrt{\log n}$ (which is an approximately optimal choice).

Corollary 4.7 *For every $\mu > 0$, SAT reduces in probabilistic polynomial time, under ℓ-length preserving reductions, to a promise problem Π in $\mathrm{MIP}_{1,\mu}[3, r, a]$, where $\ell(n) = n^{1+O((\log\log n)/\sqrt{\log n})}$, $r(n) = (1 + O((\log\log n)/\sqrt{\log n})) \cdot \log_2 n$ and $a(n) = 2^{O(\sqrt{\log n})}$.*

In Section 4.3, we show how to apply state-of-the-art proof composition to the aforementioned MIPs in order to derive our main result (i.e., a PCP with similar randomness complexity using a *constant* number of queries).

Overview and organization of the proof of Theorem 4.6. The rest of the Section 4.2 is devoted to proving Theorem 4.6. We start with an overview of the proof, which modifies the proof of [25], improving the latter in two points. The proof of [25] first reduces SAT to a parameterized problem, called GapPCS, under $\ell'(n)$-length preserving reductions for $\ell'(n) = n^{1+\gamma}$ for any $\gamma > 0$. Then they give a 3-prover MIP proof system for the reduced instance of GapPCS, where the verifier tosses $(3+\gamma)\log \ell'(n)$ random coins.

Our first improvement shows that the reduction of [25] actually yields a stronger reduction than stated there, in two ways. First we note that their proof allows for smaller values of $\ell'(n)$ than stated there, allowing in particular for the parameters we need; that is, we get $\ell'(n) = \ell(n)$, where ℓ is as in Theorem 4.6. Furthermore, we notice that their result gives rise to instances from a restricted class, for which slightly more efficient proof systems can be designed. In particular, we can reduce the size of the smallest prover in their MIP system to $\ell(n)$ (as opposed to their result which gives a prover of size $\ell'(n)^{1+\gamma}$ for arbitrarily small γ). These improvements are stated formally in Appendix A (see Lemmas A.3 and A.4, yielding Theorem A.5).

The second improvement is more critical to our purposes. Here we improve the randomness complexity of the MIP verifier of [25], by applying a random projection to it. In order to allow for a clean presentation of this improvement, we first abstract the verifier of [25] (or rather the one obtained from Theorem A.5). This is done in Section 4.2.1. We then show how to transform such a verifier into one with $(1 + o(1)) \log n$ randomness. This transformation comes in three stages, described in Sections 4.2.2-4.2.4. The key stage (undertaken in Section 4.2.3) is a shrinking of the sizes of all provers to roughly ℓ.

4.2.1 Abstracting the verifier of Theorem A.5

The verifier (underlying the proof) of Theorem A.5 interacts with three provers, which we denote P, P_1, and P_2. We let Q, Q_1, and Q_2 denote the question space of the three provers, respectively. Similarly, A, A_1, and A_2 denote the corresponding spaces of (prover) answers; that is, $P : Q \to A$ (resp., $P_1 : Q_1 \to A_1$ and $P_2 : Q_2 \to A_2$). We denote by $V_x(r, a, a_1, a_2)$ the acceptance predicate of the verifier on input $x \in \{0,1\}^n$, where r denotes the verifier's coins, and a (resp., a_1, a_2) the answer of prover P (resp., P_1, P_2). (Note: The value of V_x is 1 if the verifier accepts.) We will usually drop the subscript x unless needed. Let us denote by $q(r)$, (resp. $q_1(r)$, $q_2(r)$) the verifier's query to P (resp., P_1, P_2) on random string $r \in \Omega$, where Ω denotes the space of verifier's coins. We note that the following properties hold for the 3-prover proof system given by Theorem A.5 (cf. Section A.3).

1. Sampleability: The verifier only tosses $O(\log n)$ coins (i.e., $\Omega = \{0,1\}^{O(\log n)}$). Thus, it is feasible to sample from various specified subsets of the space of all possible coin outcomes. For example, given $S_1 \subseteq Q_1$, we can uniformly select in poly(n)-time a sequence of coins r such that $q_1(r) \in S_1$.

2. Uniformity: The verifier's queries to prover P (resp. P_1 and P_2) are uniformly distributed over Q (resp. over Q_1 and Q_2); that is, q is $|\Omega|/|Q|$-to-1 (resp. q_i is $|\Omega|/|Q_i|$-to-1).

3. Decomposability: The acceptance-predicate V decomposes in the sense that for some predicates V_1 and V_2 it holds that $V(r, a, a_1, a_2) = V_1(r, a, a_1) \wedge V_2(r, a, a_2)$, for all r, a, a_1, a_2. Furthermore, for any constant $\epsilon > 0$ (as in Theorem A.5), if x is a NO-instance then *for every* possible P strategy, *there exists* a subset $Q' = Q'_P \subseteq Q$ such that for every P_1 and P_2 the

following two conditions holds

$$\Pr_{r\in\Omega}[q(r) \in Q' \wedge V_1(r, P(q(r)), P_1(q_1(r)))] \;<\; \frac{\epsilon}{2} \tag{22}$$

$$\Pr_{r\in\Omega}[q(r) \notin Q' \wedge V_2(r, P(q(r)), P_2(q_2(r)))] \;<\; \frac{\epsilon}{2} \tag{23}$$

where V_1 and V_2 are the decomposition of $V = V_x$.

Indeed, $\Pr_{r\in\Omega}[V(r, P(q(r)), P_1(q_1(r)), P_2(q_2(r))) = 1] < \epsilon$ follows, but the current property says something much stronger.

The Decomposition Property plays a central role in the rest of our argument. Intuitively, it allows us to reduce the three-prover case to the two-prover case (treated in Section 4.1).

4.2.2 The 3-prover MIP: Stage I

The current stage is merely a preparation towards the next stage, which is the crucial one in our construction. The preparation consists of modifying the verifier of Theorem A.5 such that its queries to provers P_1 and P_2 are "independent" (given the query to the prover P). That is, we define a new verifier, denoted W, that behaves as follows:

Construction 4.8 (Verifier W) *On input x, let $V = V_x$ be the* (original) *verifier's predicate and let V_1 and V_2 be as given in the Decomposability Property.*

1. *Pick $q \in Q$ uniformly and pick coins r_1 and r_2 uniformly and independently from the set $\Omega_q \stackrel{\rm def}{=} \{r \in \Omega : q(r) = q\}$.*
2. *Make queries q* (which indeed equals $q(r_1) = q(r_2)$), *$q_1 = q_1(r_1)$ and $q_2 = q_2(r_2)$, to P, P_1 and P_2, respectively. Let $a = P(q)$, $a_1 = P_1(q_1)$ and $a_2 = P_2(q_2)$ denote the answers received.*
3. *Accept if and only if $V_1(r_1, a, a_1) \wedge V_2(r_2, a, a_2)$.*

In Step 1, we use the Sampleability Property (with respect to a specific set of r's). The analysis of W relies on the Uniformity Property, and more fundamentally on the Decomposition Property. We note that Construction 4.8 merely motivates the construction in Stage II, and thus the analysis of Construction 4.8 (captured by Proposition 4.9) is not used in the rest of the paper (although it provides a good warm-up).

Proposition 4.9 *Verifier W has perfect completeness and soundness at most ϵ.*

Proof: The completeness is obvious, and so we focus on the soundness. Fix a NO-instance x and any choice of provers P, P_1 and P_2. By the Decomposition Property, the probability that W accepts is given by

$$\Pr_{q\in Q,\; r_1,r_2\in\Omega_q}[EV_1(r_1) \wedge EV_2(r_2)] \tag{24}$$

where $EV_1(r_1) \stackrel{\rm def}{=} V_1(r_1, P(q), P_1(q_1(r_1)))$ and $EV_2(r_2) \stackrel{\rm def}{=} V_2(r_2, P(q), P_2(q_2(r_2)))$. Note that $q = q(r_1) = q(r_2)$, where (q and) r_1, r_2 are selected as above. Thus, EV_i only depends on r_i, and the shorthand above is legitimate. Letting $Q' = Q'_P$ be the subset of Q as given by the Decomposition Property of the MIP, we upper-bound Eq. (24) by

$$\begin{aligned} & \Pr_{q\in Q,\ r_1,r_2\in\Omega_q}[q \in Q' \wedge EV_1(r_1) \wedge EV_2(r_2)] + \Pr_{q\in Q,\ r_1,r_2\in\Omega_q}[q \notin Q' \wedge EV_1(r_1) \wedge EV_2(r_2)] \\ \leq & \Pr_{q\in Q,\ r_1\in\Omega_q}[q \in Q' \wedge EV_1(r_1)] + \Pr_{q\in Q,\ r_2\in\Omega_q}[q \notin Q' \wedge EV_2(r_2)] \end{aligned} \tag{25}$$

By the Uniformity Property, the process of selecting r_1 (resp., r_2) in Eq. (25) is equivalent to selecting it uniformly in Ω (and setting $q = q(r_i)$). We thus upper bound (25) by

$$\Pr_{r_1\in\Omega}[q(r_1) \in Q' \wedge EV_1(r_1)] + \Pr_{r_2\in\Omega}[q(r_2) \notin Q' \wedge EV_2(r_2)].$$

Using the Decomposition Property, each of these two terms is bounded by $\epsilon/2$ and thus their sum is upper-bounded by ϵ. ■

4.2.3 The 3-prover MIP: Stage II

In the next stage, which is the crucial one in our construction, we reduce the size of the provers P_1 and P_2 by a random projection. Specifically, we reduce the size of P_i from $|Q_i|$ to $|S_i|$.

Construction 4.10 (The projected W) *For sets $S_1 \subseteq Q_1$ and $S_2 \subseteq Q_2$, we define the (S_1, S_2)-* restricted verifier*, denoted W_{S_1,S_2}, as follows: Again, on input x, let $V = V_x$ be the verifier's predicate and let V_1 and V_2 be as given in the Decomposability Property.*

1. *Pick $q \in Q$ uniformly and pick coins r_1 and r_2 uniformly and independently from the sets $\Omega^{(1)}_{q,S_1} \stackrel{\text{def}}{=} \{r\in\Omega : q(r)=q \wedge q_1(r)\in S_1\}$ and $\Omega^{(2)}_{q,S_2} \stackrel{\text{def}}{=} \{r\in\Omega : q(r)=q \wedge q_2(r)\in S_2\}$, respectively. If either of the sets is empty, then the verifier simply accepts.*

2. *Make queries $q = q(r_1) = q(r_2)$, $q_1 = q_1(r_1)$ and $q_2 = q_2(r_2)$, to P, P_1 and P_2, respectively. Let $a = P(q)$, $a_1 = P_1(q_1)$ and $a_2 = P_2(q_2)$ denote the answers received.*

3. *Accept if and only if $V_1(r_1, a, a_1) \wedge V_2(r_2, a, a_2)$.*

Again, in the construction, we use the sampleability of various subsets of the verifier coins, whereas we will rely on the Uniformity and Decomposability Properties for the analysis. As in Construction 4.8, it is clear that the verifier W_{S_1,S_2} has perfect completeness (for every S_1 and S_2). We now bound the soundness of this verifier, for most choices of sufficiently large sets S_1 and S_2:

Lemma 4.11 *For randomly chosen sets S_1 and S_2, each of size $N \stackrel{\text{def}}{=} O(|Q|\cdot\max\{\log|A|, \log|Q|\})$, with probability at least $5/6$, the soundness error of the verifier W_{S_1,S_2} is at most 4ϵ.*

Proof: We start with some notation. Recall that Ω denotes the space of random strings of the verifier V (of Section 4.2.1). For $i \in \{1, 2\}$ and a fixed set S_i, let W_i denote the distribution on Ω induced by picking uniformly a query $q \in Q$, then picking r_i uniformly from the set $\Omega^{(i)}_{q,S_i}$, and outputting r_i. Note that the verifier W_{S_1,S_2} picks r_1 (resp., r_2) according to distribution W_1 (resp., W_2), where r_1 and r_2 depend on the same random $q \in Q$. Similarly, let U_i denote the distribution on Ω induced by picking a random string r_i uniformly from the set $\cup_{q\in Q}\Omega^{(i)}_{q,S_i}$; that is, U_i is the uniform distribution on $\{r\in\Omega : q_i(r)\in S_i\}$. Note that both W_i and U_i depend on S_i, but to avoid cumbersome notation we did not make this dependence explicit. Still, at times, we use $\mathcal{W}_i(S_i)$

(resp., $\mathcal{U}_i(S_i)$) to denote the distribution W_i (resp., U_i) that is defined as above based on the set S_i.

We use the notation $r \leftarrow D$ to denote that r is picked according to distribution D. In our analysis, we will show that, for a (sufficiently large) random S_i, the distributions U_i and W_i are statistically close, where as usual the statistical difference between U_i and W_i is defined as $\max_{T\subseteq\Omega}\{\Pr_{r_i\leftarrow U_i}[r_i\in T]-\Pr_{r_i\leftarrow W_i}[r_i\in T]\}$. We will then show that a modified verifier W'_{S_1,S_2} that picks r_1 and r_2 independently from the distributions U_1 and U_2, respectively, has low soundness error. We stress that in contrast to W'_{S_1,S_2}, the verifier W_{S_1,S_2} selects r_1 and r_2 (from distributions W_1 and W_2) such that the r_1 and r_2 are not independent (but rather depend on the same $q\in Q$). Still, as in the proof of Proposition 4.9, the Decomposition Property (of Section 4.2.1) allows for the analysis to go through.

The above informal description is made rigorous by considering the following bad events, over the probability space defined by the random choices of S_1 and S_2:

BE1: The statistical difference between $\mathcal{U}_1(S_1)$ and $\mathcal{W}_1(S_1)$ is more than ϵ.

BE2: The statistical difference between $\mathcal{U}_2(S_2)$ and $\mathcal{W}_2(S_2)$ is more than ϵ.

BE3: There exist P and P_1 such that for $Q'=Q'_P$ (as in Decomposition Property) the condition of Eq. (22) is strongly violated when selecting r_1 according to $\mathcal{U}_1(S_1)$ (rather than uniformly in Ω); that is,

$$\Pr_{r_1\leftarrow\mathcal{U}_1(S_1)}\left[(q(r_1)\in Q')\wedge V_1(r_1,P(q(r_1)),P_1(q_1(r_1)))\right]>\epsilon.$$

BE4: There exist P and P_2 such that for $Q'=Q'_P$ the condition of Eq. (23) is strongly violated when selecting r_2 according to $\mathcal{U}_2(S_2)$; that is,

$$\Pr_{r_2\leftarrow\mathcal{U}_2(S_2)}\left[(q(r_2)\in Q')\wedge V_2(r_2,P(q(r_2)),P_2(q_2(r_1)))\right]>\epsilon.$$

Below we will bound the probability of these bad events, when S_1 and S_2 are chosen at random. But first we show that if none of the bad events occur, then the verifier W_{S_1,S_2} has small soundness error.

Claim 4.11.1 *If for sets S_1 and S_2 none of the four bad event occurs then the soundness error of W_{S_1,S_2} is at most 4ϵ.*

Proof: Let $(r_1,r_2)\leftarrow W_{S_1,S_2}$ denote a random choice of the pair (r_1,r_2) as chosen by the verifier W_{S_1,S_2}. Fix proofs P,P_1,P_2 and let $Q'=Q'_P$ (and V_1,V_2) be as in the Decomposition Property. Then,

$$\begin{aligned}
&\Pr_{(r_1,r_2)\leftarrow W_{S_1,S_2}}[V_1(r_1,P(q(r_1)),P_1(q_1(r_1)))\wedge V_2(r_2,P(q(r_2)),P_2(q_2(r_2)))]\\
&\quad\le \Pr_{r_1\leftarrow\mathcal{W}_1(S_1)}[(q(r_1)\in Q')\wedge V_1(r_1,P(q(r_1)),P_1(q_1(r_1)))]\\
&\qquad\quad+\Pr_{r_2\leftarrow\mathcal{W}_2(S_2)}[(q(r_2)\notin Q')\wedge V_2(r_2,P(q(r_2)),P_2(q_2(r_2)))]\\
&\quad\le \Pr_{r_1\leftarrow\mathcal{U}_1(S_1)}[(q(r_1)\in Q')\wedge V_1(r_1,P(q(r_1)),P_1(q_1(r_1)))]+\epsilon \qquad [\neg\mathrm{BE1}\text{ and }\neg\mathrm{BE2}]\\
&\qquad\quad+\Pr_{r_2\leftarrow\mathcal{U}_2(S_2)}[(q(r_2)\notin Q')\wedge V_2(r_2,P(q(r_2)),P_2(q_2(r_2)))]+\epsilon\\
&\quad\le 2\epsilon+2\epsilon \qquad [\neg\mathrm{BE3}\text{ and }\neg\mathrm{BE4}]
\end{aligned}$$

where the first inequality uses manipulation as in the proof of Proposition 4.9 (cf. Eq. (25)). ■

We now turn to upper-bound the probability of the bad events.

Claim 4.11.2 *The probability of event* BE1 (resp., BE2) *is at most* 1/24.

Proof: To estimate the statistical difference between $U_i = \mathcal{U}_i(S_i)$ and $W_i = \mathcal{W}_i(S_i)$, we take a closer look at the distribution U_i. We note that sampling r_i according to U_i is equivalent to selecting $r'_i \leftarrow U_i$ (i.e., r'_i is selected uniformly in $\{r : q_i(r) \in S_i\}$), setting $q = q(r'_i)$, and picking r_i uniformly from the set $\{r : (q(r) = q) \wedge (q_i(r) \in S_i)\} = \Omega^{(i)}_{q,S_i}$. In contrast, in the distribution W_i, the output is selected uniformly in $\Omega^{(i)}_{q,S_i}$, where q is selected uniformly in Q. Thus, the statistical difference between U_i and W_i is due to the statistical difference in the distributions induced on $q = q(r_i)$, which in turn equals

$$\frac{1}{2} \cdot \sum_{q \in Q} \left| \Pr_{r_i \leftarrow \mathcal{U}_i(S_i)}[q(r_i) = q] - \Pr_{r_i \leftarrow \mathcal{W}_i(S_i)}[q(r_i) = q] \right| \; = \; \frac{1}{2} \cdot \sum_{q \in Q} \left| \Pr_{r_i \leftarrow \mathcal{U}_i(S_i)}[q(r_i) = q] - \frac{1}{|Q|} \right|.$$

To bound this sum, we bound the contribution of each of its terms (for a random S_i of size N). Fixing an arbitrary $q \in Q$, we consider the random variable

$$\zeta_q = \zeta_q(S_i) \stackrel{\rm def}{=} \Pr_{r \leftarrow \mathcal{U}_i(S_i)}[q(r) = q] = \frac{|\{r : (q(r) = q) \wedge (q_i(r) \in S_i)\}|}{|\{r : q_i(r) \in S_i\}|}$$

(as a function of the random choice of S_i of size N). Using the Uniformity Property, we infer that the denumenator equals $N \cdot \frac{|\Omega|}{|Q_i|}$, and the expected value of the numerator equals $|\{r : q(r) = q\}| \cdot \frac{N}{|Q_i|} = \frac{|\Omega|}{|Q|} \cdot \frac{N}{|Q_i|}$. Thus, $\mathbf{E}[\zeta_q] = 1/|Q|$. A simple application of Chernoff Bound (see Footnote 2) shows that, with probability at least $\exp(-\Omega(\epsilon^2 \cdot N/|Q|))$, this random variable is $(1 \pm \epsilon)/|Q|$. Thus, for $N = c \cdot |Q| \log |Q|$ (where $c = O(1/\epsilon^2)$), the probability that $\Pr_{r \leftarrow U_i}[q(r) = q]$ is not in $[(1 \pm \epsilon)/|Q|]$ is at most $|Q|^{-1}/24$. By the union bound, the probability that such a q exists is at most $1/24$, and if no such q exists then the statistical difference is bounded by at most ϵ. ■

Claim 4.11.3 *The probability of event* BE3 (resp., BE4) *is at most* 1/24.

Proof: We will bound the probability of the event BE3. The analysis for BE4 is identical. Both proofs are similar to the proof of Lemma 4.3 (i.e., projection in the two-prover case). Indeed, our interest in the Decomposition Property is motivated by the fact that it allows for a reduction of the three-prover case to the two-prover case. This reduction culminates in the current proof, which refer only to the communication with two provers (i.e., P and P_i).

Fix P and let $Q' = Q'_P$ be the set as given by the Decomposition Property (of Section 4.2.1). We will show that for a randomly selected subset $S_1 \subset Q_1$ of size N the following holds

$$\Pr_{S_1}\left[\exists P_1 \text{ s.t. } \Pr_{r_1 \leftarrow \mathcal{U}_1(S_1)}[(q(r_1) \in Q') \wedge V_1(r_1, P(q(r_1)), P_1(q_1(r_1)))] \geq 2\epsilon\right] \; \leq \; \frac{1}{24} \cdot |A|^{-|Q|} \qquad (26)$$

The claim will follow by a union bound over the $|A|^{|Q|}$ possible choices of P.

Note that, for each fixed P (and thus fixed $Q' = Q'_P$), there is an optimal prover $P_1 = P_1^*$ that maximizes the quantity $\epsilon'_{q_1} \stackrel{\rm def}{=} \Pr_{r \in \Omega | q_1(r) = q_1}[(q(r) \in Q') \wedge V_1(r, P(q(r)), P_1(q_1))]$, for every $q_1 \in Q_1$. Furthermore, by (the Uniformity Property and) the Decomposition Property (see Eq. (22)), it holds that $\mathbf{E}_{q_1 \in Q_1}[\epsilon'_{q_1}] = \mathbf{E}_{r \in \Omega}[\epsilon'_{q_1(r)}] < \epsilon/2$. For simplicity, assume that the expectation is at least $\epsilon/3$ (by possibly augmenting the event that defines ϵ'_{q_1}). Applying Chernoff Bound (see Footnote 2), we get that the probability that when we pick N elements from Q_1, uniformly and independently, their average is more than ϵ (or even more than twice the expectation) is at most $\exp(-\Omega(\epsilon N))$. Thus if $N \geq c \cdot |Q| \log |A|$ for some large enough constant c, then this probability is at most $\frac{1}{24}|A|^{-|Q|}$ as claimed in Eq. (26). The current claim follows. ■

Combining Claims 4.11.2 and 4.11.3, we conclude that for random sets S_1 and S_2, with probability at most 4/24 a bad event occurs; that is, with probability at least 5/6 none of the four BEi's occurs. Invoking Claims 4.11.1, the lemma follows. ■

4.2.4 The 3-prover MIP: Stage III

Having reduced the sizes of the three prover strategies, it is straightforward to reduce the amount of randomness used by the verifier. Below we describe a reduced randomness verifier $W_{S_1,S_2,T}$, where $S_i \subseteq Q_i$ for $i = 1, 2$, and $T \subseteq \Omega$.

Construction 4.12 (The final verifier $W_{S_1,S_2,T}$): *For sets $S_1 \subseteq Q_1$ and $S_2 \subseteq Q_2$, and*

$$T \subseteq \{(r_1, r_2) : (q(r_1) = q(r_2)) \wedge (q_i(r_i) \in S_i, \forall i \in \{1, 2\})\}, \tag{27}$$

*we define the (S_1, S_2, T)-*restricted verifier*, denoted W_{S_1,S_2}, as follows: Again, on input x, let $V = V_x$, V_1 and V_2 be as given in Construction 4.10.*

1. *Pick $(r_1, r_2) \in T$ uniformly at random.*
2. *Make queries $q = q(r_1) = q(r_2)$, $q_1 = q_1(r_1)$ and $q_2 = q_2(r_2)$, to P, P_1 and P_2, respectively. Let $a = P(q)$, $a_1 = P_1(q_1)$ and $a_2 = P_2(q_2)$ denote the answers received.*
3. *Accept if and only if $V_1(r_1, a, a_1) \wedge V_2(r_2, a, a_2)$.*

Here, again, we use sampleability of subsets of the verifier coins, It is obvious that the verifier uses $\log_2 |T|$ random bits, and that perfect completeness is preserved (for any S_1, S_2 and T). It is also easy to see that a sufficiently large random set T yields $W_{S_1,S_2,T}$ of low soundness error; that is:

Lemma 4.13 *Let $s \stackrel{\text{def}}{=} O(|Q| \max\{\log |A|, \log |Q|\})$ and $t \stackrel{\text{def}}{=} O(|Q|(\log |A|) + s \cdot ((\log |A_1|) + (\log |A_2|))$. Suppose that S_1 and S_2 are uniformly selected s-subsets of Q_1 and Q_2, and that T is a uniformly selected t-subset satisfying Eq. (27). Then, with probability at least $\frac{2}{3}$, the verifier $W_{S_1,S_2,T}$ has soundness error at most 5ϵ.*

Proof: By Lemma 4.11, with probability 5/6, the verifier W_{S_1,S_2} has soundness error at most 4ϵ. Using the Uniformity Property, W_{S_1,S_2} can be seen as selecting (r_1, r_2) uniformly in the set on the r.h.s of Eq. (27), and setting $q = q(r_1) = q(r_2)$. It is quite straightforward[10] to show that for a random T, with probability at least 5/6, the resulting $W_{S_1,S_2,T}$ has soundness error at most 5ϵ. The lemma follows. ■

Using Lemma 4.13, we now prove Theorem 4.6.

Proof [of Theorem 4.6]: Fix $\epsilon' = \epsilon/5$. Let V be the 3-prover verifier for SAT as obtained from Theorem A.5. In particular, V has perfect completeness and soundness ϵ'. The size of the

[10]The proof proceeds along the outline provided in Footnote 9 (to the proof of Lemma 4.3). First, fixing any choice of strategies $P : Q \to A$, $P_1 : S_1 \to A_1$ and $P_2 : S_2 \to A_2$, we consider the event that $W_{S_1,S_2,T}$ accepts with probability greater than 5ϵ (when interacting with these strategies). Using Chernoff Bound (again), we see that for a random T this event occurs with probability at most $\exp(-\Omega(\epsilon t))$. Taking the union bound over all possible strategies (i.e., choices of P, P_1 and P_2), we infer that $W_{S_1,S_2,T}$ fails with respect to some choice of strategies with probability at most $|A|^{|Q|}|A_1|^{|S_1|}|A_2|^{|S_2|} \cdot \exp(-\Omega(\epsilon t)) < 1/6$ (by the setting of t).

smallest prover is $\ell'(n) = m(n)^{O(m(n))} \cdot n^{1+O(1/m(n))}$, the answer length is bounded by $a'(n) = m(n)^{O(1)} \cdot n^{O(1/m(n))}$, and V satisfies the properties listed in Section 4.2.1.

For sets S_1, S_2, T, let $W_{S_1,S_2,T}$ be the verifier obtained by modifying V as described in the current section (see Constructions 4.8, 4.10 and 4.12). Consider the promise problem Π whose instances are tuples (ϕ, S_1, S_2, T) where an instance is a YES-instance if $W_{S_1,S_2,T}$ accepts ϕ with probability one, and the instance is a NO-instance if $W_{S_1,S_2,T}$ accepts with probability at most ϵ. We note that an instance of Π of size $N > n$ has a 3-prover proof system using at most $\log_2 N$ random coins, having answer length $a'(n) < a'(N)$, perfect completeness and soundness error $7\epsilon' = \epsilon$ (since $W_{S_1,S_2,T}$ is such a verifier).

Now, consider the reduction that maps an instance ϕ (of length n) of SAT to the instance (ϕ, S_1, S_2, T), where $S_1 \subset Q_1$ and $S_2 \subset Q_2$ are random subsets of queries of V of size $s'(n) = \ell'(n) \cdot a'(n)$ and T is a random subset of size $t'(n) = s'(n) \cdot a'(n)$ of the random strings used by the verifier W_{S_1,S_2} (see Constructions 4.10 and 4.12). This reduction always maps satisfiable instances of SAT to YES-instances of Π and, by Lemma 4.13, with probability at least $\frac{2}{3}$, it maps unsatisfiable instances of SAT to NO-instances of Π. Finally, note that

$$|(\phi, S_1, S_2, T)| = \tilde{O}(t'(n)) = \tilde{O}(\ell'(n) \cdot a'(n)^2) = m(n)^{O(m(n))} \cdot n^{1+O(1/m(n))} = \ell(n) \,.$$

The theorem follows. ■

4.3 Reducing the answer size and obtaining PCPs

Applying state-of-the-art composition lemmas to the MIP constructed in Section 4.2 gives our final results quite easily. In particular, we use the following lemmas.

Lemma 4.14 (cf. [4] or [9, 31]): *For every $\epsilon > 0$ and integer p, there exists $\delta > 0$ such that for every $r, a : \mathsf{Z}^+ \to \mathsf{Z}^+$,*

$$\mathrm{MIP}_{1,\delta}[p, r, a] \subseteq \mathrm{MIP}_{1,\epsilon}[p + 3, r + O(\log a), \mathrm{poly}(\log a)].$$

Starting with the MIP of Theorem 4.6, we apply Lemma 4.14 repeatedly till the answer lengths become $\mathrm{poly}(\log\log\log n)$. Then, to terminate the recursion, we use the following result of [25].

Lemma 4.15 [25, Lem. 2.6]: *For every $\epsilon > 0$ and integer p, there exists $\gamma > 0$ such that for every $r, a : \mathsf{Z}^+ \to \mathsf{Z}^+$,*

$$\mathrm{MIP}_{1,\gamma}[p, r, a] \subseteq \mathrm{PCP}_{1,\frac{1}{2}+\epsilon}[r + O(2^{pa}), p + 7],$$

where $\mathrm{PCP}_{1,s}[r, q]$ denotes the set of promise problems having a PCP verifier of perfect completeness, soundness error s, randomness complexity r and query complexity q.

Recall that in case of PCP the query complexity is measured in bits. Combining the above lemmas with the nearly-linear 3-prover systems obtained in Section 4.2, we obtain:

Theorem 4.16 (Our main PCP result): *For every $\epsilon > 0$, SAT reduces probabilistically, under $n^{1+O((\log\log n)/\sqrt{\log n})}$-length preserving reductions to a promise problem $\Pi \in \mathrm{PCP}_{1,\frac{1}{2}+\epsilon}[(1 + O((\log\log n)/\sqrt{\log n})) \cdot \log n, 19]$. Furthermore, the reduction runs in time ℓ.*

Proof: We start with Corollary 4.7 and apply Lemma 4.14 thrice, obtaining a 12-prover MIP system with answer lengths $\text{poly}(\log\log\log n)$. Specifically, we start with a 3-prover MIP of randomness complexity $r_0(n) = \log_2 n + (\log\log n)\cdot\sqrt{\log n}$ and answer length $a_0(n) = \exp(O(\sqrt{\log n}))$, and after i iterations we get a $(3+3i)$-prover MIP of randomness complexity $r_i(n) = r_{i-1}(n) + O(\log a_{i-1}(n))$ and answer length $a_i(n) = \text{poly}(\log a_{i-1}(n))$. Thus, $a_i(n) = \text{poly}(\log^{(i)} n)$ and $r_i(n) = r_0(n) + O(\log a_0(n)) = \log_2 n + O((\log\log n)\cdot\sqrt{\log n})$.

Next, applying Lemma 4.15 gives the desired 19-query PCP. Specifically, this $(12+7)$-query PCP has randomness complexity $r_3(n) + \text{poly}(2^{a_3(n)}) < r_3(n) + 2^{(\log\log n)/2}$, which is upper-bounded by $\log_2 n + O((\log\log n)\cdot\sqrt{\log n})$. The furthermore clause follows by recalling that Corollary 4.7 (which is merely an instantiation of Theorem 4.6) is proven using a reduction that appends $\ell(n)$ random bits to the original instance (see proof of Theorem 4.6). ■

Corollary: Theorem 2.5. Theorem 4.16 implies Theorem 2.5, because the (effective) length of the oracle used by a $\text{PCP}[r,q]$ system is at most $2^r \cdot q$.

5 Shorter Locally Testable Codes from PCPs

In this section we strengthen the results of Section 3 by presenting locally-testable binary codes of nearly-linear length (i.e., $n = k^{1+o(1)}$ rather than $n = k^{1+\epsilon}$, for any constant $\epsilon > 0$, as in Section 3.4). We do so by starting with the random projection of the FS/RS-code from Section 3.2, and applying PCP techniques to reduce the alphabet size (rather than following the paradigm of concatenated codes as done in the rest of Section 3). Specifically, in addition to encoding individual alphabet symbols via codewords of a binary code, we also augment the new codewords with small PCPs that allow to emulate the local-tests of the original codeword tester. Using an off-the-shelve PCP (e.g., the one of [2]) this yields a *weak* locally testable code (i.e., one satisfying Definition 2.1); for details see Section 5.1. As we explain in Section 5.2, using an off-the-shelve PCP fails to provide a locally testable code (i.e., one satisfying Definition 2.2), and some modifications are required in order to redeem this state of affairs (as well as in order to obtain a linear code). Most of the current section is devoted to implementing these modifications. Still, the easy derivation of the weak testability result (in Section 5.1) serves as a good warm-up.

Organization: After presenting (in Section 5.1) the *weak* codeword testing result, and discussing (in Section 5.2) the difficulties encountered when trying to obtain a *strong* codeword testing result, we turn to establish the latter (i.e., prove Theorem 2.3). We start by developing (in Section 5.3) a framework for PCPs with extra properties that are useful to our goal of using these PCPs in the construction of locally testable codes. We call the reader's attention to §5.3.1, which provides a wider perspective that may be of independent interest. We then construct such PCPs (by modifying known constructions in Section 5.4), and combine all the ingredients to establish Theorem 2.3 (in Section 5.5). Finally, in Section 5.6, we consider the actual randomness and query complexities of codeword testers, and show that logarithmic randomness and three queries suffices (for establishing our main results).

5.1 Easy derivation of a weak testability result

We start with the locally-testable code $\mathcal{C}^R : \Sigma^k \to \Sigma^n$, where $n = |R|$, presented in Section 3.2. Recall that codewords in $\mathcal{C}^R$ assigns to each line $\ell \in R$ a univariate polynomial of low-degree (represented as a Σ-symbol, where $\Sigma = F^{d+1}$). We refer to the codeword test of Construction 3.4, which works by selecting a pair of intersecting lines and checking that the two polynomials assigned to these lines agree on the value of their point of intersection. We wish to convert $\mathcal{C}^R$, which is a code over a large alphabet, into to a binary code that is locally-testable and preserves the distance and rate of $\mathcal{C}^R$.

The basic idea is to augment $\mathcal{C}^R$ with small PCPs, each corresponding to a pair of intersecting lines that can be selected by the $\mathcal{C}^R$-tester, such that each PCP asserts that the corresponding two polynomials (i.e., the two polynomials residing at the locations associated with these two lines) agree on the value of the point of intersection. Each such PCP has length polynomial in the length of its assertion, which in turn has length $2 \cdot \log|\Sigma|$, and can be verified using a constant number of queries (see, e.g., [2]). Assuming that R covers all points almost uniformly (see Claim 3.5.1), we note that the number of pair of intersecting lines that can be selected by the $\mathcal{C}^R$-tester (of Construction 3.4) is approximately $|R| \cdot |F|$, where $|F| = O(\log|\Sigma|)$. Thus, the total length of the proofs that we need to add to the code is at most a $\text{poly}(\log|\Sigma|)$ factor larger than n, which is fine under an adequate choice of parameters (discussed below). Essentially, the tester for the new code will emulate the old codeword tester by invoking the PCP verifier, which in turn accesses only a constant number of bits in the adequate proof.

The main problem with the above description is that the PCP verifier needs to be given (explicitly) the assertion it verifies, whereas we are only willing to read a constant number of bits (both of the assertion and the corresponding proof). Still, all standard PCP constructs (e.g., [19, 2]) can be extended to yield similar results in case one is charged for oracle access to both the input and the proof-oracle, provided that the input is presented in a suitable error-correcting format. Actually, this property is stated explicitly in [5][11], and is always referred to when using the PCP as an "inner-verifier" (in the setting of PCP composition). Furthermore, these (inner) PCPs can also handle an input that is presented by a constant number of encoding of substrings that cover the entire input. Indeed, we are using the PCP here as an inner-verifier (but compose it with a codeword-tester rather than with an outer-verifier). Lastly, we should replace each symbol in the $\mathcal{C}^R$-codeword by its encoding under a *suitable (for the inner-verifier)* code $\mathcal{C}' : \Sigma \to \{0,1\}^{\text{poly}(\log|\Sigma|)}$ of *linear distance*. This allows to verify that two substrings provide the encoding of two (low-degree) polynomials that agree on a certain point, by making a constant number of (bit) queries. (Needless to say, it is only guaranteed that the verifier rejects with high probability when the two substrings are far from having the desired property, which suffices for our purposes.)

A last issue regarding the code construction is that we should apply a suitable number of repetitions to the resulting n-sequence (of $\mathcal{C}'$-codewords) such that its length dominates the length of the added PCPs (denoted L). Recall that the number of PCPs equals the size of the ("effective") probability space of the codeword tester of $\mathcal{C}^R$ (given in Construction 3.4)[12], which in turn equals $|R| \cdot |F| = |F| \cdot n$. The size of each proof is polynomial in the length of the assertion, which in turn consists of two $\mathcal{C}'$-codewords, each of length $n' \stackrel{\text{def}}{=} \text{poly}(\log|\Sigma|)$, where $\Sigma = F^{d+1}$ and $d < |F|$. Thus, the total length of the added PCPs is approximately

$$L \stackrel{\text{def}}{=} (|F| \cdot n) \cdot \text{poly}(2n') \;=\; \text{poly}(|F|) \cdot n \;=\; n^{1+O(1/m)}, \tag{28}$$

because $n = |R| > |F|^m$ and $\log|\Sigma| = (d+1)\log|F| = \tilde{O}(|F|)$ (using $d < |F|$). Since the length of

[11]In fact, the presentation of Babai *et al.* [5] is in these terms, as captured by their notion of a holographic proof. We mention that the recently introduced notion of a PCP of Proximity [10] (a.k.a Assignment Tester [17]) generalizes holographic proofs by omitting the reference to the encoding (of inputs via a good error-correcting code).

[12]Recall that this tester uniformly selects a point in F^m and a line in R going through this point. The effective probability (relevant for the following construction) is the number of possible choices of such (point and line) pairs, which equals $|R| \cdot |F|$.

each PCP is greater than n', it follows that L is bigger than $n \cdot n'$, and so repetitions are indeed needed to make the (concatenated) code dominate the length of the final code. On the other hand, L is not too large (i.e., $L = n^{1+O(1/m)}$), and so the repetition will not effect the rate of the code by too much. This yield the following construction.

Construction 5.1 *For a suitable number of repetitions t, the resulting code maps $x = (x_1, ..., x_k) \in \Sigma^k$ to $((\mathcal{C}'(y_1), ..., \mathcal{C}'(y_n))^t, \pi_1, ..., \pi_r)$, where $(y_1, ..., y_n) = \mathcal{C}^R(x_1, ..., x_k)$ and π_i is a PCP* (to be further discussed) *that refers to the i^{th} possible choice of a pair of lines examined by the $\mathcal{C}^R$-tester, and r denotes the number of such possible choices. Specifically, we set $t = \omega(L/nn')$; e.g., $t = (L/nn') \cdot \log n$. As for the π_i's they are PCPs that establish that the corresponding $\mathcal{C}'$-codewords in the first block of nn' bits in the new codeword encode Σ-symbols that would have been accepted by the codeword test of $\mathcal{C}^R$. In particular, these PCPs establish that the corresponding n'-bit long strings are $\mathcal{C}'$-codewords. Indeed, for $i = (x, \ell)$, the proof π_i refers to the lines ℓ_x and ℓ, where ℓ_x is the canonical line of $x \in F^m$ and ℓ is a random line* (in R) *that passes through x (i.e., one of approximately $|R|/|F|^{m-1}$ possibilities). This proof* (i.e., π_i) *asserts that the two n'-bit long strings in locations corresponding to ℓ_x and ℓ are $\mathcal{C}'$-codewords that encode two polynomials, denoted h_x and h, that satisfy $h_x(\alpha) = h(\beta)$, where α and β are determined by $i = (x, \ell)$ such that $\ell_x(\alpha) = \ell(\beta) = x$. In the sequel, we will identify the index of these PCPs with the corresponding pair of lines* (i.e., $i \equiv (\ell_x, \ell)$).

By our choice of t, the distance of the new code (of Construction 5.1) is determined by the distance of $\mathcal{C}^R$ (and the distance of $\mathcal{C}'$). The block-length of the new code is $N \stackrel{\text{def}}{=} (1 + \log n) \cdot L$, where (by Eq. (28)) $L = n^{1+O(1/m)}$. Using $d = m^m$, we have $m > (\sqrt{\log k})/(\log\log k)$ (by Eq. (2)). Furthermore, by Eq. (3), we have $n = \exp(\tilde{O}(\sqrt{\log k})) \cdot k$ and $\log |\Sigma| = \exp(\tilde{O}(\sqrt{\log k}))$. Thus, we have $N = \tilde{O}(L) = n^{1+O(1/m)}$. Note that

$$n^{1+O(1/m)} = (\exp(\tilde{O}(\sqrt{\log k})) \cdot k)^{1+O((\log\log k)/(\sqrt{\log k}))} = \exp(\tilde{O}(\sqrt{\log k})) \cdot k.$$

Thus, the code of Construction 5.1 maps $K = k \cdot \log_2 |\Sigma| > k$ bits to N-bit long codewords, where $N = \exp(\tilde{O}(\sqrt{\log K})) \cdot K$.

The tester for the code of Construction 5.1 emulates the testing of $\mathcal{C}^R$ by inspecting the PCP that refers to the selected pair of lines. In addition, it also tests (at random) that the first t blocks (of length nn' each) are identical. A specific implementation of this scheme follows.

Construction 5.2 (weak codeword tester for Construction 5.1): *When testing $w = (w_1, ..., w_{tn}, w_{tn+1}, ..., w_{tn+r})$, where $w_i : [n'] \to \{0,1\}$ for $i = 1, ..., tn$ and $w_i : [L/r] \to \{0,1\}$ for $i = tn+1, ..., tn+r$, proceed as follows.*

1. *Invoke the $\mathcal{C}^R$-tester in order to select a random pair of intersecting lines (ℓ_1, ℓ_2). That is, (ℓ_1, ℓ_2) is distributed as in Step 1 of Construction 3.4.*
2. *Invoke the PCP-verifier providing it with oracle access to the input-oracles w_{ℓ_1} and w_{ℓ_2} and the proof-oracle w_{tn+i}, where $i \equiv (\ell_1, \ell_2)$. If the verifier reject then halt and reject, otherwise continue.*
3. *Check that $w_{jn+\ell} = w_\ell$, for uniformly selected $j \in [t-1]$ and $\ell \in [n]$, by comparing $w_{jn+\ell}(i) = w_\ell(i)$ for a randomly selected $i \in [n']$. If equality holds then accept.*

Clearly, Construction 5.2 makes a constant number of queries and accepts every codeword of Construction 5.1. We thus turn to analyze this test's performance on non-codewords. The key point in the following (relatively easy) analysis is that if a sequence is far from the new code then most of the distance must be due to the tnn'-bit long prefix of the N-bit sequence, because $L = N - tnn' < N/\log n$. That is, if $w = (w_1, ..., w_{tn}, w_{tn+1}, ..., w_{tn+r})$ is δ-far from the code, then $(w_1, ..., w_{tn})$ must be δ'-far from the code (denoted E' in Lemma 5.3) that consists of the tnn'-bit long prefix of the code described in Construction 5.1, where $\delta' > \delta - (1/\log n)$. Thus, for any constant $\delta > o(1)$ or even any $\delta > 2/\log n$, we may focus on analyzing the case that the tnn'-bit long prefix of w is $\delta/2$-far from the residual code (obtained by ignoring the PCP part of Construction 5.1). We undertake this task next.

Lemma 5.3 *Let E' be the code obtained by projecting the code described in Construction 5.1 on the first tnn' coordinates; that is, $E'(x)$ is the tnn'-bit long prefix of the encoding of x by Construction 5.1. Suppose that $w' = (w_1, ..., w_{tn}) \in (\{0,1\}^{n'})^{tn}$ is δ'-far from E'. Then, when given oracle access to $(w_1, ..., w_{tn}, w_{tn+1}, ..., w_{tn+r})$, the tester of Construction 5.2 rejects with probability $\Omega(\delta')$, regardless of the values of $w_{tn+1}, ..., w_{tn+r} \in \{0,1\}^{L/r}$.*

It follows that if w is δ-far from the code of Construction 5.1, then w is rejected by Construction 5.2 with probability $\Omega(\delta - (1/\log n))$.

Proof: Let us denote by ϵ the *average* (relative) distance of $(w_1, ..., w_n)$ from $(w_{jn+1}, ..., w_{jn+1})$, for a random $j \in [t-1]$. Let E be the code obtained by taking the first nn' bits of the code E'; that is, the bits corresponding to $(w_1, ..., w_n)$. We observe (see Proposition 5.5 at the end of this section) that either $\epsilon \geq \delta'/2$ or $(w_1, ..., w_n)$ is $(\delta'/2)$-far from the code E. Noting that the first case is detected with probability ϵ by the ("repetition") test of Step 3, we focus on the second case and consider what happens when invoking the PCP verifier.

Bearing in mind that $(w_1, ..., w_n)$ is $(\delta'/2)$-far from E, let us denote by ϵ_i the (relative) distance of w_i from $\mathcal{C}'$. We distinguish two cases, regarding the average of the ϵ_i's:

Case 1: If $\sum_{i=1}^{n} \epsilon_i/n > \delta'/4$ then the PCP verifier will reject with probability $\Omega(\delta')$. The reason is that the second query of the $\mathcal{C}^R$-tester (i.e., the random line passing through a random point) is almost uniformly distributed, and so the PCP verifier will be invoked on a pair of input-oracles such that on the average the second input-oracle is $(\delta'/8)$-far from the code $\mathcal{C}'$, where the average is taken over this (slightly skewed) choice of the second line (which is the choice used in Construction 5.2). In such a case, the PCP verifier will reject with probability $\Omega(\delta'/8)$.

Case 2: If $\sum_{i=1}^{n} \epsilon_i/n \leq \delta'/4$ then we consider the $\mathcal{C}'$-codewords, denoted by c_i's, that are closest to these w_i's. In the current case, $(c_1, ..., c_n)$ is $(\delta'/4)$-far from E, because $(w_1, ..., w_n)$ is $(\delta'/2)$-far from E. Let d_i be the $\mathcal{C}'$-decoding of c_i (i.e., $c_i = \mathcal{C}'(d_i)$). Then, $(d_1, ..., d_n)$ is $(\delta'/4)$-far from the code $\mathcal{C}^R$, and would have been rejected by the $\mathcal{C}^R$-tester with probability $p \stackrel{\text{def}}{=} \Omega(\delta'/4)$.

Let us call a pair of lines (ℓ_1, ℓ_2) good if the $\mathcal{C}^R$-tester would have rejected the values d_{ℓ_1} and d_{ℓ_2}. By the above, with probability p, the $\mathcal{C}^R$-tester selects a good pair of lines. On the other hand, for a good pair of lines (ℓ_1, ℓ_2), when given access to the input-oracles c_{ℓ_1} and c_{ℓ_2} (and any proof-oracle), the PCP verifier rejects with constant probability. We need, however, to consider what happens when the PCP verifier is given access to the input-oracles w_{ℓ_1} and w_{ℓ_2} (and the proof-oracle $w_{tn+(\ell_1,\ell_2)}$), when (ℓ_1, ℓ_2) is a good pair. In the rest of this proof we show that, for a good (ℓ_1, ℓ_2), the PCP verifier rejects the input-oracles w_{ℓ_1} and w_{ℓ_2} with constant probability. This happens regardless of whether or not (w_{ℓ_1}, w_{ℓ_2}) is close to (c_{ℓ_1}, c_{ℓ_2}). Letting $\delta_{\mathcal{C}'}$ denote the constant relative distance of $\mathcal{C}'$, we consider two sub-cases:

1. If both w_{ℓ_i}'s are $(\delta_{\mathcal{C}'}/4)$-close to the corresponding c_{ℓ_i}'s then (w_{ℓ_1}, w_{ℓ_2}) is $(\delta_{\mathcal{C}'}/4)$-far from any pair of *acceptable* strings, and the PCP verifier rejects the input-oracles w_{ℓ_1} and w_{ℓ_2} with constant probability (i.e., $\Omega(\delta_{\mathcal{C}'}/4) = \Omega(1)$).
   The reason that (w_{ℓ_1}, w_{ℓ_2}) is $(\delta_{\mathcal{C}'}/4)$-far from any acceptable pair of strings is due to the fact that the latter are pairs of codewords and the code has relative distance $\delta_{\mathcal{C}'}$. Specifically, if (c'_1, c'_2) is a pair of acceptable codewords, then $(c'_1, c'_2) \neq (c_{\ell_1}, c_{\ell_2})$ and
$$\begin{aligned}\Delta((w_{\ell_1}, w_{\ell_2}), (c'_1, c'_2)) &\geq \Delta((c_{\ell_1}, c_{\ell_2}), (c'_1, c'_2)) - \Delta((w_{\ell_1}, w_{\ell_2}), (c_{\ell_1}, c_{\ell_2})) \\ &\geq \delta_{\mathcal{C}'} \cdot n' - 2 \cdot \frac{\delta_{\mathcal{C}'}}{4} \cdot n'\end{aligned}$$
   which equals $(\delta_{\mathcal{C}'}/4) \cdot 2n'$.
2. Otherwise (i.e., some w_{ℓ_i} is $(\delta_{\mathcal{C}'}/4)$-far from the corresponding c_{ℓ_i}, which by definition is the codeword closest to w_{ℓ_i}), one of the input-oracles is $\delta_{\mathcal{C}'}/4$-far from being a codeword, and again the PCP verifier rejects with constant probability.

We conclude that, for a good pair (ℓ_1, ℓ_2), when given access to the input-oracles w_{ℓ_1} and w_{ℓ_2}, the PCP verifier rejects with constant probability (regardless of the contents of the proof-oracle). Recalling that a good pair is selected with probability $p = \Omega(\delta')$, it follows that in this case (i.e., Case 2) the PCP verifier rejects with probability $\Omega(\delta')$.

The lemma follows. ■

Combining all the above, we obtain:

Theorem 5.4 (weak version of Theorem 2.3): *For infinitely many K's, there exist* weak *locally-testable binary codes of length $N = \exp(\tilde{O}(\sqrt{\log K})) \cdot K = K^{1+o(1)}$ and constant relative distance.*

In contrast to Theorem 2.3, the codes asserted in Theorem 5.4 only have *weak* codeword tests (i.e., tests satisfying Definition 2.1). Furthermore, these codes are *not necessarily linear.*

Digression on distances in repetition codes. In the proof of Lemma 5.3, we noted above that the distance of a string from a code obtained by repeating some basic code can be attributed (in half) either to the distance of the first block from the basic code or to the distance of the other blocks from the first block. Here we state a more general result that suggests that, for any probability distribution $(p_1, ..., p_t)$, we may test a "repetition of some basic code" by selecting the i^{th} block with probability p_i and checking whether this block is in the basic code and whether this block equals a uniformly selected block.

Proposition 5.5 *Let Ω be a finite set and $\delta : \Omega \times \Omega \to \mathsf{R}$ be any non-negative function that satisfies the triangle inequality* (i.e., $\delta(x,z) \leq \delta(x,y) + \delta(y,z)\ \forall x,y,z \in \Omega$). *For any $S \subseteq \Omega$, define $\delta_S(x) \stackrel{\text{def}}{=} \min_{y \in S}\{\delta(x,y)\}$. Fixing any t, define $\overline{\delta}((x_1, ..., x_t), (y_1, ..., y_t)) = \sum_{i=1}^{t} \delta(x_i, y_i)/t$ and $\mathsf{R}(S) = \{x^t : x \in S\}$. Also, for any $T \subseteq \Omega^t$ and $\overline{x} \in \Omega^t$, let $\overline{\delta}_T(\overline{x}) \stackrel{\text{def}}{=} \min_{\overline{y} \in T}\{\overline{\delta}(\overline{x}, \overline{y})\}$. Then, for any probability distribution $(p_1, ..., p_t)$ on $[t]$, and for every $\overline{x} = (x_1, ..., x_t) \in \Omega^t$, it holds that*
$$\sum_{i \in [t]} p_i \cdot \delta_S(x_i) + \sum_{i \in [t]} p_i \cdot \sum_{j \in [t]} \frac{\delta(x_j, x_i)}{t} \geq \overline{\delta}_{\mathsf{R}(S)}(\overline{x}).$$

Proof: We first establish the claim for the special case in which $p_1 = 1$ and $p_2 = \cdots = p_t = 0$. We do so by using the triangle inequality (and the definitions of $\overline{\delta}$ and $\mathsf{R}(S)$), and observing that

$$\begin{aligned}\overline{\delta}_{\mathsf{R}(S)}(\overline{x}) &\leq \overline{\delta}(\overline{x}, x_1^t) + \overline{\delta}_{\mathsf{R}(S)}(x_1^t) \\ &= \sum_{j\in[t]} \frac{\delta(x_j, x_1)}{t} + \delta_S(x_1).\end{aligned}$$

Clearly, this generalizes to any i (i.e., using x_i instead of x_1), and taking the weighted average (weighted by the general p_i's), the proposition follows. ■

5.2 Problems with an easy derivation of the strong testability result

Before turning to the actual constructions, we explain why merely plugging-in a standard (inner-verifier) PCP will not work (for strong codeword testability). We start with the most severe problem, and then turn to additional ones.

Non-canonical encoding: As discussed in Section 1.1, the soundness property of standard PCPs does not guarantee that only the "canonical" proof (obtained by the designated construction) is accepted with high probability. The standard soundness property only guarantees that false assertions are rejected with high probability (no matter which proof-oracle is used). Furthermore, typical PCPs tend to accept also non-canonical proofs. This is due to a gap between the canonical oracles (used in the completeness condition) that encodes information as polynomials of specific *individual degree*, and the verification procedure that only refers to the *total degree* of the polynomial.[13] This problem was avoided in Section 5.1 by discarding non-codewords that are close to the code and making the PCPs themselves a small part of the codeword. Thus, the non-canonical PCPs by themselves could not make the sequence too far from the code, and so nothing is required when we use the weak definition of codeword testing. However, when we seek to achieve the stronger definition, this problem becomes relevant (and cannot be avoided).

An additional potential problem is that, per definition, PCPs do not necessarily provide "strong soundness" (i.e., reject a proof that is ϵ-far from being correct with probability $\Omega(\epsilon)$). Although some known PCPs (e.g., [2]) have this added property, other (e.g., [26]) don't.

Linearity: We wish the resulting code to be linear, and it is not clear whether this property holds when composing a linear code with a standard inner-verifier. Since we start with an F-linear code (and an F-linear codeword test), there is hope that the proof-oracle added to the concatenated code will also be linear (over GF(2), provided that F is an extension field of GF(2)). Indeed, with small modifications of standard constructions, this is the case.

Other technical problems: Other problems arise in translating some of the standard "complexity-theoretic tricks" that are used in all PCP constructions. For example, PCP constructions are typically described in terms of a dense collection of input lengths (e.g., the input length must fit $|H|^m$ for some suitable sizes of $|H|$ and m (i.e., $m = \Theta(|H|/\log|H|)$), and are

[13] In basic constructions of codes, this is not a real problem because we can define the code to be the collection of all polynomials of some total degree as opposed to containing only polynomials satisfying some individual degree bound. However, when using such a code as the inner code in composition, we cannot adopt the latter solution because we only know how to construct adequate inner-verifiers for inputs encoded as polynomials of individually-bounded degree (rather than bounded total degree).

extended to arbitrary lengths by padding (of the input). In our context, such padding, depending on how it is done, either permits multiple encodings (of the same information), or forces us to check for additional conditions on the input (e.g., that certain bits of the input are zeroes). Other complications arise when one attempts to deal with "auxiliary variables" that are introduced in a process analogous to the standard reduction of verification of an arbitrary computation to the satisfiability of a 3CNF expression.

This forces us to re-work the entire PCP theory, while focusing on "strongly rejecting" non-canonical proofs and on obtaining "linear PCP oracles" when asked to verify homogeneous linear conditions on the input. By strongly rejecting non-canonical proofs, we mean that any string should be rejected with probability proportional to its distance from the canonical proof (which is indeed analogously to the definition of strong codeword tester). We comment that for the purposes of constructing short locally testable codes, it suffices to construct verifiers verifying systems of homogeneous linear equations and this is all we will do (although we could verify affine equations equally easily). In what follows, whenever we refer to a linear system, we mean a conjunction of homogeneous linear constraints.

5.3 Inner verifiers for linear systems: Definition and composition

We use PCP techniques to transform linear locally testable codes over a large alphabet into locally testable codes over a smaller alphabet. Specifically, we adapt the construction of inner-verifiers such that using them to test linear conditions on the input-oracles can be done while utilizing a proof-oracle that is obtained by a linear transformation of the input-oracles. Furthermore, the constructions are adapted to overcome the other difficulties mentioned in Section 5.2 (most importantly the issue of non-canonical proofs).

The basic ingredient of our transformations is the notion of an inner verifier for linear codes. Since the definition is quite technical, we consider it useful to start with a wider perspective on the various ingredients of this definition. We consider this perspective, provided in §5.3.1, to be of independent interest. The actual definition of an inner verifier for linear codes and its various composition properties are presented in §5.3.2-5.3.4.

5.3.1 A wider perspective

Two basic extensions of the standard definition of soundness (for PCP systems) were mentioned in Section 5.2: The first is a requirement to reject "non-canonical" proofs, where a canonical proof is one specified in the completeness condition. The second extension is a requirement for strong soundness, which means the rejection of non-valid proofs with probability that is proportional to their distance from a valid proof. In the following definition we incorporate both requirements, while considering strings over arbitrary alphabets (rather than binary strings).

Definition 5.6 (Strong PCP): *A* standard verifier, *denoted V, is a probabilistic polynomial-time oracle machine. On input $x \in \Sigma^*$, we only consider oracles of length $\ell(|x|)$, where $\ell : \mathsf{N} \to \mathsf{N}$ satisfies $\ell(n) \leq \exp(\text{poly}(n))$. A* prover strategy, *denoted P, is a function that maps* YES-*instances to adequate proof-oracles. In particular, $|P(x)| = \ell(|x|)$. We say that V is a* strong PCP *for the promise problem Π if it satisfies the following two conditions:*

- Completeness (w.r.t P)*: For every* YES-*instance $x \in \Sigma^*$* (of Π)*, on input x and access to oracle $P(x)$, the verifier always accepts x. That is,* $\Pr[V^{P(x)}(x) = 1] = 1$.

  The string $P(x)$ is called the canonical proof for *x.*

- Strong soundness (w.r.t canonical proofs): *For every $x \in \Sigma^*$ and $\pi \in \Sigma^{\ell(|x|)}$, the following holds:*

  1. *If x is a* NO*-instance* (of Π) *then $P(x) = \lambda$, and every π is said to be 1-far from λ.*
  2. *If π is δ-far from $P(x)$ then, on input x and access to oracle π, the verifier rejects with probability $\Omega(\delta)$. That is,* $\Pr[V^\pi(x) \neq 1] = \Omega(\Delta(\pi, P(x)))/|\pi|$, *for every x and π.*

Standard soundness follows by combining the two parts of the strong soundness condition. We comment that strong soundness per se (i.e., with respect to any valid proof) can be defined by letting $P(x)$ be the set of all ("absolutely") valid proofs (i.e., $P(x) = \{\pi \in \Sigma^{\ell(|x|)} : \Pr[V^\pi(x) = 1] = 1\}$). That is, strong soundness (w.r.t any valid proof) says that for any YES-instance x and every $\pi \in \Sigma^{\ell(|x|)}$, the rejection probability of $V^\pi(x) = 1$ is proportional to the distance of π from the set of all proofs that are accepted with probability 1 (i.e., $\Pr[V^\pi(x) = 1] = \Omega(\delta_x(\pi))$, where $\delta_x(\pi)$ is the minimum of $\Delta(\pi, \pi')/|\pi|$ taken over all π' satisfying $\Pr[V^{\pi'}(x) = 1] = 1$, and $\delta_x(\pi) = 1$ if no such proof exists (i.e., x is a NO-instance)). It seems that, in the context of PCP, strong soundness w.r.t any valid proof is a more natural notion than strong soundness w.r.t canonical proofs.[14] Things change, when one wishes to use PCP in the construction of locally testable codes. Strong soundness (w.r.t canonical or arbitrary valid proofs) extends naturally to PCPs of Proximity (PCPP, as defined recently in [10, 17]):

Definition 5.7 (Strong PCPP): *A* proximity verifier*, denoted V, is a probabilistic polynomial-time oracle machine that is given access to two oracles, an* input-oracle *$x : [n] \to \Sigma$ and a* proof-oracle *$\pi : [\ell(n)] \to \Sigma$, where n is V's only explicitly given input, and ℓ is as in Definition 5.6. A prover strategy, denoted P, is defined as in Definition 5.6. We say that V is a* strong PCPP *for the promise problem Π if it satisfies the following two conditions:*

- Completeness (w.r.t P): *For every* YES*-instance $x \in \Sigma^*$, on input $1^{|x|}$ and access to the oracles x and $P(x)$, the verifier always accepts x. That is,* $\Pr[V^{x,P(x)}(1^{|x|}) = 1] = 1$. *Again, $P(x)$ is called the* canonical proof for *x.*

- Strong soundness (w.r.t canonical proofs): *For every $x \in \Sigma^*$ and $\pi \in \Sigma^{\ell(|x|)}$, on input $1^{|x|}$ and access to the oracles x and π, the verifier rejects with probability $\Omega(\delta(x))$, where*

$$\delta(x) \stackrel{\text{def}}{=} \min_{x',\pi'} \left\{ \max\left(\frac{\Delta(x, x')}{|x|} \; ; \; \frac{\Delta(P(x), \pi')}{\ell(|x|)} \right) \right\} \tag{29}$$

  and, as in Definition 5.6, for any NO*-instance x we define $P(x) = \lambda$, and say that any π is 1-far from λ. Alternatively, $\delta(x)$ can be defined as the minimum of* $\max(\frac{\Delta(x,x')}{|x|} \; ; \; \frac{\Delta(P(x),\pi')}{\ell(|x|)})$ *taken over all* YES*-instances x' and every π'.*

We mention that the above formulation benefits from [10, 17], which has appeared after the preliminary publication of the current work. In the current work, we follow the older tradition (rooted in [5]) of considering only the special case in which the YES-instances of Π are encodings, under some good error correcting code E, of YES-instances in some other set S. That is, the YES-instances of Π are $\{E(x) : x \in S\}$, and the NO-instances of Π are all strings that are far from the YES-instances

[14] Indeed, standard PCP constructions tend to satisfy strong soundness w.r.t any valid proof. Furthermore, some of the valid proofs correspond to the "encoding" of different NP-witnesses, whereas others arise from the gap between individual degree bound and total degree bound (discussed in Section 1.1 and 5.2).

of Π. (We stress that the notion of a PCPP (let alone Definition 5.7) is not used in the rest of this work, except for a few clarifying comments.)

The definition presented in §5.3.2 incorporates all the above themes, while adding two additional themes. Firstly, we refer to a situation (which arises naturally in proof composition a la [3, 2]) in which the verifier is given access to $p > 1$ input-oracles rather than to one. (These oracles are supposed to contain the encoding of strings whose concatenation yields a YES-instance of another language.) Secondly, we refer to PCPPs that check linear relations, while utilizing verifiers that only conduct linear tests (on the retrieved oracle answers) and having canonical proofs that are linear transformations of the (actual) input.

5.3.2 The actual definition

One basic ingredient of our constructions is the notion of an inner-verifier for linear codes. These inner-verifiers are actually strong PCPPs (as in Definition 5.7) for assertions regarding linear conditions on the input-oracles. This means that their definition is quite complex: it refers to *strong soundness w.r.t canonical proofs* as well as to a formalism regarding encoding of inputs. In addition, the following definition refers to a formalism for expressing (conjunctions of) linear conditions.

The "linear inner PCP systems" defined below have quite a few parameters, where the main ones specify the field F, the number of input-oracles q and the set F^b of possible symbols that they encode, and the number of queries p made by the inner verifier and the set F^a of possible answers to these queries. That is, each of the q input-oracles is supposed to encode an element of F^b as a sequence over F^a, where typically $a \ll b$. Thus, an $(F,(q,b) \to (p,a))$ linear inner-PCP system is the main ingredient in a transformation of an F-linear code over an alphabet $\Sigma = F^b$ *that is testable by q queries*, into an F-linear code (of a typically longer length) over an alphabet $\Gamma = F^a$ *that is testable by p queries*, where typically $a \ll b$ but $p > q$. Informally, the inner-verifier allows to emulate a local test in the given code over Σ, by providing an encoding (over Γ) of each symbol in the original codeword as well as auxiliary proofs (regarding the satisfiability of homogeneous linear conditions) that can be verified based on a constant number of queries. That is, given a locally testable code $\mathcal{C}_0 : \Sigma^{K_0} \times [N_0] \to \Sigma$, we consider the mapping of $x \in \Sigma^{K_0}$ to $(E(\mathcal{C}_0(x,1)), ..., E(\mathcal{C}_0(x,N_0)))$, where $E : \Sigma \to \Gamma^n$ is the aforementioned encoding. Then an $(F,(q,b) \to (p,a))$ inner-PCP system should allow to transform a q-query codeword tester of $\mathcal{C}_0$ (which makes F-linear checks) into a p-query codeword tester of the code resulting from appending adequate (inner-verifier) proofs to the aforementioned mapping (i.e., the concatenated code of $\mathcal{C}_0$ and E). In addition, we wish these auxiliary proofs to be obtained by F-linear transformations of x.

We start by presenting the basic syntax of such linear inner PCP systems, which depend on a formalism for expressing (conjunctions of) linear conditions. We observe that verifying that a vector satisfies a conjunction of (homogeneous) linear conditions is equivalent to verifying that it lies in some linear subspace (i.e., the space of vectors that satisfy these conditions). For integer d and field F, we let $\mathcal{L}_{F,d}$ denote the set of all linear subspaces of F^d. We will represent such a subspace $L \in \mathcal{L}_{F,d}$ by a matrix $M \in F^{d \times d}$ such that $L = \{x \in F^d : Mx = \vec{0}\}$. According to convenience, we will sometimes say that *a vector lies in L* and sometimes say that *it satisfies the conditions L*.

Definition 5.8 (the mechanics of linear inner verification): *For a field F, positive integers q, b, p, a and $\delta \in (0,1)$, an $(F,(q,b) \to (p,a),\delta)$-*linear inner system *consists of a triple (E,P,V) such that*

1. *$E : F^b \to (F^a)^n$ is an F-linear code of minimum distance at least δn over the alphabet F^a. We call E the* encoding function.

2. $P : \mathcal{L}_{F,qb} \times (F^b)^q \to (F^a)^\ell$ *is called the* proving function. *For every* $L \in \mathcal{L}_{F,qb}$, *the mapping* $x \mapsto P(L,x)$ *is required to be* F-linear.

3. V *is an oracle machine, called the* verifier, *that gets as input* $L \in \mathcal{L}_{F,qb}$ *and* (coins) $\omega \in \{0,1\}^r$ *and has oracle access to* $q+1$ *vectors over* F^a, *denoted* $X_1, \ldots, X_q : [n] \to F^a$ *and* $X_{q+1} : [\ell] \to F^a$. *That is, a query* $j \in [n]$ *to oracle* $i \in [q]$ *is answered by* $X_i(j)$, *and a query* $j \in [\ell]$ *to oracle* $q+1$ *is answered by* $X_{q+1}(j)$. *It is required that* V *satisfies the following two conditions:*

   Query complexity: *For every* $L \in \mathcal{L}_{F,qb}$ *and* $\omega \in \{0,1\}^r$, *machine* V *makes a total of exactly* p *oracle calls to the oracles* $X_1, \ldots, X_{q+1}$.

   Linearity of verdict: *For every* L *and* ω, *the acceptance condition of* V *is a conjunction of* F-*linear constraints on the responses to the queries. That is, based on* L *and* ω, *machine* V *determines some* $L' \in \mathcal{L}_{F,pa}$ *and accepts if and only if* $(\alpha_1, ..., \alpha_p) \in L'$, *where* α_i *is the answer obtained for the* j^{th} *query.*

   The vectors $X_1, \ldots, X_q$ *are called the* input-oracles *and the vector* X_{q+1} *is called the* proof-oracle.

Such a system is said to use r *coins, encodings of length* n *and proofs of length* ℓ.

Indeed, the requirement that V makes exactly p queries (rather than at most p queries) is made for technical convenience (and can be easily met by making dummy queries if necessary). Definition 5.8 makes no reference to the quality of the decisions made by the verifier. This is the subject of the next definition.

Definition 5.9 (linear inner verification – perfect completeness and strong soundness): *A system* (E, P, V) *as in Definition 5.8 is called* γ-good *if it satisfies the following two conditions:*

Completeness: *If the first* q *oracles encode a* q-*tuple of vectors over* F^b *that satisfies* L *and if* $X_{q+1} = P(L, x_1, \ldots, x_q)$ *then* V *always accepts.*

That is, for every $x_1, \ldots, x_q \in F^b$ *and* $L \in \mathcal{L}_{F,qb}$ *such that* $(x_1, \ldots, x_q) \in L$, *and for every* $\omega \in \{0,1\}^r$, *it holds that* $V^{E(x_1),\ldots,E(x_q),P(L,x_1,\ldots,x_q)}(L,\omega) = 1$.

Strong Soundness: *If the first* q *oracles are far from encoding any* q-*tuple of vectors over* F^b *that satisfies* L *then* V *rejects with significant probability, no matter which* X_{q+1} *is used. Furthermore, if the first* q *oracles are close to encoding some* q-*tuple that satisfies* L *but* X_{q+1} *is far from the corresponding unique proof determined by* P *then* V *rejects with significant probability. Actually, in both cases, we require that the rejection probability be proportional to the relevant relative distance, where* γ *is the constant of proportionality.*

That is, for every $L \in \mathcal{L}_{F,qb}$, *every* $X_1, \ldots, X_q : [n] \to F^a$ *and* $X_{q+1} : [\ell] \to F^a$, *it holds that*

$$\Pr_\omega[V^{X_1,\ldots,X_q,X_{q+1}}(L,\omega) \neq 1] \geq \gamma \cdot \delta_L(X_1, \ldots, X_q, X_{q+1})$$

where for $\overline{X} = (X_1, \ldots, X_q, X_{q+1})$,

$$\delta_L(\overline{X}) = \min_{(x_1,\ldots,x_q)\in L} \left\{ \max\left(\max_{i\in[q]} \left\{ \frac{\Delta(X_i, E(x_i))}{n} \right\} ; \frac{\Delta(X_{q+1}, P(L, x_1, \ldots, x_q))}{\ell} \right) \right\} \tag{30}$$

The quantity $\delta_L(X_1, \ldots, X_q, X_{q+1})$ *will be called the* deviation of $(X_1, \ldots, X_q, X_{q+1})$.

*In such a case we say that (E,P,V) is an $(F,(q,b) \to (p,a),\delta,\gamma)$-*linear inner proof system*, abbreviated as $(F,(q,b) \to (p,a),\delta,\gamma)$-*LIPS.

We comment that there is a redundancy in the linearity requirements made in Definition 5.8. Specifically, we have required E, P and V (or rather its acceptance condition) to be F-linear. However, under the completeness and soundness conditions of Definition 5.9, the linearity of E and V implies the linearity of P, and the linearity of E and P implies without loss of generality the linearity of V.[15]

Typically, we aim at having n,ℓ and 2^r be small functions of b (i.e., polynomial or even almost-linear in b), whereas p may grow as a function of q (which is typically a constant). Note that Definition 5.9 is designed to suit our applications. Firstly, the strong notion of soundness, which refers also to "non-canonical" proofs of valid statements, fits our aim of obtaining a code that is locally testable (because it guarantees rejection of sequences that are not obtained by the transformation induced by (the encoding function and) the proving function). Indeed, this augmentation of soundness is non-standard (and arguably even unnatural) in the context of PCP. Secondly, the strong notion of soundness allows also to reject with adequate probability inputs that are close to the code (or alleged proofs that are close to the canonical ones), and thus support the strong definition of codeword testing (i.e., Definition 2.2). Finally, Definition 5.9 only handles the verification of linear conditions, and does so while using proofs that are linear transformation of the input. Indeed, this fits our aim of transforming F-linear codes over a large alphabet (i.e., the alphabet F^b) to F-linear codes over a smaller alphabet (i.e., F^a).

5.3.3 Obtaining locally testable codes

The utility of linear inner proof systems (LIPSes) in constructing locally-testable codes is demonstrated by two of the following results (i.e., Proposition 5.10 and Theorem 5.13). In Proposition 5.10 we show that any LIPS yields a locally-testable code, where the distance is provided by the encoding function of the LIPS. In Theorem 5.13 we compose a locally testable code over a large alphabet with a LIPS to obtain a locally testable code over a smaller alphabet. Proposition 5.10 merely serves as a warm-up towards the Theorem 5.13, which is the result actually used in the rest of our work.

To simplify the exposition, we are going to confine ourselves to $(\cdot,\cdot,\cdot,\gamma)$-LIPSes with $\gamma \leq 1$. Indeed, for any $\gamma' < \gamma$, any $(\cdot,\cdot,\cdot,\gamma)$-LIPS constitute a $(\cdots,\gamma')$-LIPS. Furthermore, this is typically the case anyhow (because the deviation parameter may equal 1, or at least be very close to 1).

Proposition 5.10 *Suppose that $a < b$ divides b, and $\gamma \in (0,1]$. Then an $(F,(1,b) \to (p,a),\delta,\gamma)$-LIPS implies the existence of an F-linear locally-testable code of relative distance at least $\delta/2$, over the alphabet $\Gamma = F^a$, mapping $F^b = \Gamma^{b/a}$ to Γ^M for $M < 2(n+\ell)$, where n and ℓ are the corresponding lengths of the encoding and the proof used by the LIPS. Specifically, the code is testable with p queries, and the tester rejects a word that is ϵ-far from the code with probability at*

[15] To see that the linearity of E and V implies the linearity of P, note that the combination of perfect completeness and strong soundness means that the set $S \stackrel{\rm def}{=} \{(E(x_1),\ldots,E(x_q),P(L,x_1,\ldots,x_q)) : (x_1,..,x_q) \in L\}$ equals the set of $q+1$-tuples $(X_1,...,X_q,X_{q+1})$ that pass all possible checks of V. Since all the latter checks are F-linear, it follows that the set S is an F-linear subspace. Using the fact that E is F-linear, it follows that $\{(x_1,..,x_q,P(L,x_1,..,x_q)) : (x_1,..,x_q) \in L\}$ is an F-linear subspace, and hence $(x_1,..,x_q) \mapsto P(L,x_1,..,x_q)$ is F-linear. To see that the linearity of E and P implies the linearity of V, we refer to [11, Prop. A.1] which implies that when testing membership in a linear subspace by a one-sided error tester (i.e., perfect completeness), without loss of generality, the tester may make only linear checks.

least $(\gamma/4)\cdot\epsilon$*. Furthermore, the tester tosses* $1+\max(r_V,\log M)$ *coins, where* r_V *is the number of coins tossed by the inner-verifier of the above LIPS.*

The point is that Proposition 5.10 establishes a locally-testable code while only relying on a standard error-correcting code (i.e., the encoding $E:\Gamma^{b/a}\to\Gamma^n$ that is part of the LIPS).

Proof: Let (E,P,V) be the $(F,(1,b)\to(p,a),\delta,\gamma)$-LIPS, where $E:F^b\to(F^a)^n$ and $P:\mathcal{L}_{F,b}\times F^b\to F^\ell$. We let $t=\lceil\ell/n\rceil$, where $n\ll\ell$ is typically the case. Under this setting, $tn\geq\ell$ and $tn+\ell<2\ell+n$. We construct a locally testable code $\mathcal{C}:\Gamma^{b/a}\to\Gamma^{tn+\ell}$, where $\Gamma^{b/a}\cong F^b$, such that the encoding of x equals the sequence $\mathcal{C}(x)=(E(x)^t,P(L,x))$, where $L=F^b$ (i.e., L is satisfied by every vector) and $E(x)$ is replicated t times. Thus, at least half of the length of $\mathcal{C}(x)$ is taken by replications of $E(x)$, and so the relative distance of $\mathcal{C}$ is at least $\delta/2$, because E has relative distance δ. Indeed $\mathcal{C}$ has block-length $M=tn+\ell<2(\ell+n)$.

To test a potential codeword $(X_1,\ldots,X_t,Y)$, where $X_i:[n]\to\Gamma$ and $Y:[\ell]\to\Gamma$, we perform at random one out of two kinds of tests: With probability $\frac{1}{2}$ we test that the t strings X_i's are replications of X_1. We do so by picking at random $i\in[t]$ and $j\in[n]$, and testing that $X_1(j)=X_i(j)$. With the remaining probability we pick a random test as per the verifier $V(F^b,\cdot)$, and emulate V's execution. In particular, we answer V's queries to its (single) input-oracle by querying our oracle X_1, and answer V's queries to its proof-oracle by querying our oracle Y. Note that although we set no condition on the vector encoded by the input-oracle (i.e., every b-ary vector over F satisfies the conditions $L=F^b$), the verifier needs to verify that the input-oracle is a codeword of E, which is what we need in order to provide a codeword test for $\mathcal{C}$.

The above tester has randomness complexity $1+\max(r_V,\log M)$, and always accepts any codeword of $\mathcal{C}$. We need to show that words at distance ϵ from the code $\mathcal{C}$ are rejected with probability $\Omega(\gamma\cdot\epsilon)$. Analogously to the proof of Proposition 5.5 , we have

$$\Delta_{\mathcal{C}}((X_1,...,X_t,Y))\;\leq\;\Delta((X_1,...,X_t,Y),(X_1^t,Y))+\Delta_{\mathcal{C}}((X_1^t,Y)).$$

Thus, if $(X_1,...,X_t,Y)$ is ϵ-far from $\mathcal{C}$ then either $(X_1,...,X_t,Y)$ is $(\epsilon/2)$-far from (X_1^t,Y) or (X_1^t,Y) is $(\epsilon/2)$-far from $\mathcal{C}$. In the *first case* we have $\Delta((X_1,...,X_t,Y),(X_1^t,Y))\geq(\epsilon/2)\cdot(tn+\ell)$, and so $\sum_{i=1}^t\Delta(X_1,X_i)/t\geq(\epsilon/2)\cdot(n+(\ell/t))>(\epsilon/2)\cdot n$. Thus the new tester rejects with probability at least $(1/2)\cdot(\epsilon/2)\geq\gamma\cdot\epsilon/4$, by virtue of the replication test (and $\gamma\leq1$). In the *second case*, we have $\Delta_{\mathcal{C}}((X_1^t,Y))\geq(\epsilon/2)\cdot(tn+\ell)$, and so $\Delta((X_1^t,Y),(E(x)^t,P(F^b,x))\geq(\epsilon/2)\cdot(tn+\ell)$ for every $x\in F^b$. Thus, for every x, either X_1 is $(\epsilon/2)$-far from $E(x)$ or Y is $(\epsilon/2)$-far from $P(F^b,x)$, which means that the deviation of (X_1,Y) (as defined in Eq. (30)) is at least $\epsilon/2$, because here the deviation is the minimum taken over all $x\in F^b$ of the maximum between $\Delta(X_1,E(x))/n$ and $\Delta(Y,P(F^b,x))/\ell$. It follows that (in this case), the inner-verifier V rejects with probability at least $p\stackrel{\rm def}{=}\gamma\cdot\epsilon/2$, and thus our codeword test rejects with probability at least $p/2=\gamma\cdot\epsilon/4$. ■

Remark 5.11 We wish to highlight an interesting fact regarding the code constructed in the proof of Proposition 5.10. Unlike in Section 5.1, the replication of the basic codeword (conducted in the construction) does not help the analysis of the new codeword test (but rather complicate it by the need to analyze the replication test). That is, the test presented in the proof of Proposition 5.10 is a strong codeword test (for $\mathcal{C}$), regardless of the choice of the parameter t (which governs the number of replications). The sole role of replication is to guarantee that the resulting code has constant relative distance. This requires setting $t=\Omega(\ell/n)$ (or alternatively relying on distance properties of the proving function). On the other hand, the bigger t the worse rate we get for the resulting code, and thus we pick $t=O(\ell/n)$. This remark applies also to Theorem 5.13.

Composing locally testable codes and LIPSes. The following theorem (i.e., Theorem 5.13) will be used to compose locally testable codes over large alphabets with suitable linear inner proof systems, obtaining locally testable codes over smaller alphabets. Specifically, given a q-query testable F-linear code over the alphabet $\Sigma = F^b$, we wish to construct a (F-linear) locally-testable code over a smaller alphabet $\Gamma = F^a$, by using a suitable LIPS. The latter includes an adequate encoding of F^a by $(F^a)^n$, and its verifier will be used to emulate the local conditions checked by the codeword test of the original code. (Recall that, using the F-linearity of $\mathcal{C}$, we may assume without loss of generality (cf. [11, Prop. A.1]) that the codeword tester makes only F-linear checks.) These conditions are subspaces of $F^{q \cdot b}$, and so we need a $(F, (q, b) \to (\cdot, a), \cdot, \cdot)$-LIPS in order to verify them. Regarding the unspecified parameters of the abovementioned $(F, (q, b) \to (p, a), \delta, \gamma)$-LIPS, we wish p to be as small as possible and γ, δ be as large as possible. The construction will be similar to the one used for deriving the weak testability result in Section 5.1. Thus, in addition to the above, we wish the randomness complexity of the codeword tester and the (encoding and) proof length of the LIPS to be as small as possible.

Although the aforementioned composition (captured by Theorem 5.13) is very natural, we were only able to establish its validity in case the locally testable code is testable by a procedure that makes (almost) uniformly distributed queries. We note that the tester presented in Section 3.2 has a version that satisfies this property (see Remark 3.6). This motivates the following definition.

Definition 5.12 (codeword testers with almost uniform queries): *For $\alpha \in (0, 1]$, a probabilistic oracle machine is said to make α-*uniform queries *if when given access to an oracle of length N, a random query in a random execution equals any fixed $i \in [N]$ with probability at least α/N and at most α^{-1}/N. That is, for every $i \in [N]$, we denote by $p_i^{(j)}$ the probability that, in a random invocation, the j^{th} query of the q-query tester is to location i, and require that*

$$\frac{\alpha}{N} \leq \frac{1}{q} \cdot \sum_{j=1}^{q} p_i^{(j)} \leq \frac{\alpha^{-1}}{N} \tag{31}$$

Theorem 5.13 (composing an outer code with an inner-verifier): *Consider integers $a < b$ such that a divides b, a finite field F, $\Sigma = F^b$ and $\alpha, \beta, \gamma, \delta \in (0, 1]$. Suppose that the following two constructs exist:*

1. *A locally testable F-linear code $\mathcal{C} : \Sigma^K \to \Sigma^N$ of relative distance at least $\delta_{\mathcal{C}}$, having a codeword test that makes q queries that are α-uniform, and uses r coins. Furthermore, suppose that this tester rejects ϵ-far sequences with probability at least $\beta \cdot \epsilon$.*
2. *A $(F, (q, b) \to (p, a), \delta, \gamma)$-linear inner proof system, (E, P, V), where $E : F^b \to (F^a)^n$, $P : \mathcal{L}_{F, q \cdot b} \times (F^b)^q \to (F^a)^\ell$, and V tosses r_V coins.*

Then, there exists an F-linear locally-testable code of relative distance at least $\delta \cdot \delta_{\mathcal{C}}/2$, over the alphabet $\Gamma = F^a$, mapping $\Sigma^K \equiv \Gamma^{b \cdot K/a}$ to Γ^M, for $M < 2 \cdot (Nn + 2^r \ell)$. Furthermore, this code can be tested by making p queries and tossing $1 + \max(r + r_V, \log M)$ coins such that ϵ-far sequences are rejected with probability at least $(\alpha\beta\gamma\delta^2/16q) \cdot \epsilon$.

Typically, $r_V > 2 + \log \ell$ and $2^r \ell > Nn$, which implies that $\log M < r + 2 + \log \ell < r + r_V$. We comment that the resulting codeword test does not necessarily make almost uniform queries; we will redeem this state of affairs at a later point (in Theorem 5.15).

Proof: The new code consists of two parts (which are properly balanced). The first part is obtained by encoding each Σ-symbol of the codeword of $\mathcal{C}$ by the code E, whereas the second part is obtained by providing proofs (testable by the inner-verifier) for the validity of each of the 2^r possible checks that may be performed by the codeword test. Specifically, let us denote by $i_{\omega,j}$ the j^{th} query that the $\mathcal{C}$-tester makes when using coins ω, and let L_ω be the linear condition verified on these coins. (Recall that, using the F-linearity of $\mathcal{C}$, we may assume without loss of generality (cf. [11, Prop. A.1]) that the codeword tester makes only F-linear checks.) Let $t = \lceil 2^r \cdot \ell/Nn \rceil$ and note that $M \stackrel{\text{def}}{=} t \cdot Nn + 2^r\ell$ satisfies $M \leq 2tNn$ and $M < 2 \cdot (Nn + 2^r\ell)$. Then, viewing $\mathcal{C}$ as $\mathcal{C} : \Sigma^K \times [N] \to \Sigma$ and recalling that $\Sigma = F^b \equiv \Gamma^{b/a}$, the string $x \in \Sigma^K$ is encoded by the sequence $(\mathcal{C}'(x)^t, P'(x))$, where

$$\mathcal{C}'(x) \stackrel{\text{def}}{=} (E(\mathcal{C}(x,1)), \ldots, E(\mathcal{C}(x,N))) \tag{32}$$

$$P'(x) \stackrel{\text{def}}{=} \langle P(L_\omega, \mathcal{C}(x, i_{\omega,1}), \ldots, \mathcal{C}(x, i_{\omega,q})) : \omega \in \{0,1\}^r \rangle \tag{33}$$

Let us denote this encoding by $\mathcal{C}''$; that is, $\mathcal{C}''(x) = (\mathcal{C}'(x)^t, P'(x))$. Note that $\mathcal{C}'' : \Gamma^{bK/a} \to \Gamma^M$, and that $\mathcal{C}''$ has distance at least $t \cdot \delta_{\mathcal{C}} n \cdot \delta N$, which means a relative distance of at least $\delta\delta_{\mathcal{C}} \cdot tNn/M \geq \delta\delta_{\mathcal{C}}/2$ (because $M \leq 2tNn$).

Testing the code $\mathcal{C}''$ is essentially done by emulating the codeword test of $\mathcal{C}$. That is, to test a potential codeword $(X_1, ..., X_{tN}; Y_{0^r}, ..., Y_{1^r})$, where $X_i : [n] \to \Gamma$ and $Y_\omega : [\ell] \to \Gamma$, we select uniformly $\omega \in \{0,1\}^r$, determine the corresponding linear condition $(i_{\omega,1}, ..., i_{\omega,q}, L_\omega)$ that would have been checked by the $\mathcal{C}$-tester, and invoke the inner-verifier V on input L_ω while providing V with oracle access to $X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}$ and Y_ω. Note that $i_{\omega,1}, ..., i_{\omega,q} \in [N]$, and that V tosses additional coins, denoted $\omega' \in \{0,1\}^{r_V}$. As in the proof of Proposition 5.10, this is done with probability 1/2, and otherwise we check the correctness of the replication (by randomly selecting $i \in [t]$, $j_1 \in [N]$ and $j_2 \in [n]$ and comparing $X_{j_1}(j_2)$ and $X_{(i-1)t+j_1}(j_2)$). Let us denote the resulting procedure by $\mathcal{T}''$.

The above procedure $\mathcal{T}''$ has randomness complexity $1+\max(r+r_V, \log M)$, makes $\max(p,2) = p$ queries (because $q \geq 1$ implies $p \geq 2$), and always accepts any codeword of $\mathcal{C}''$. Although $\mathcal{T}''$ looks very appealing, it may *not* satisfies the requirements (of a codeword tester) in case the $\mathcal{C}$-tester does *not* make almost uniform queries. Nevertheless, we will show that $\mathcal{T}''$ is indeed a $\mathcal{C}''$-tester, provided that the $\mathcal{C}$-tester makes almost uniform queries (as guaranteed by the theorem's hypothesis). Before doing so, we discuss the reason for this technical condition.

On the necessity of almost uniform queries. Consider, for example, the case in which $\mathcal{C}(x) = (0, \mathcal{C}_0(x))$, where $\mathcal{C}_0$ is a locally testable code with tester $\mathcal{T}_0$, and suppose that the first query of the $\mathcal{C}$-tester is always to the first position in the sequence (i.e., the position that is supposed to be identically 0) but the $\mathcal{C}$-tester usually ignores the answer (and with probability $1/N$ checks that the answer equals 0). (The other queries of the $\mathcal{C}$-tester emulate $\mathcal{T}_0$.) Further suppose that the proving function of the LIPS sets the first $\ell/2$ symbols of the (ℓ-symbol long) proof to 0, and that the inner-verifier always compares a random symbol in its first (n-symbol long) input-oracle (which is supposed to encode the answer to the $\mathcal{C}$-tester's first query and hence is supposed to be $E(0) = 0^n$) to a random symbol in the first half of its proof-oracle. (In addition, the inner-verifier emulates some "normal" inner-verifier using the same input-oracles and the second half of its proof-oracle.) Then, the corresponding code $\mathcal{C}''$ has non-codewords that are very far from the code, where the difference is concentrated almost only in the "proof-part", but these non-codewords are rejected by $\mathcal{T}''$ with negligible probability. For example, consider the non-codeword $((1, \mathcal{C}_0(x))^t, \langle 1^{\ell/2}\pi_\omega : \omega \in \{0,1\}^r \rangle)$, where $0^{\ell/2}\pi_\omega$ is the canonical proof associated with coins ω and input x (i.e., $P'(x) = \langle 0^{\ell/2}\pi_\omega : \omega \in \{0,1\}^r \rangle$). This sequence is 1/4-far from $\mathcal{C}''$ but is rejected by $\mathcal{T}''$ only if it emulates the checking of the first bit of $\mathcal{C}$, which happens with probability $1/N = o(1)$. The above discussion establishes the necessity of the upper-bound on $\sum_{j=1}^{q} p_i^{(j)}$ provided in Eq. (31).

The lower-bound provided by Eq. (31) is inessential, because an alternative one follows by the fact that the tester must query each location with sufficiently high probability (in order to reject non-codewords that are corrupted only at that location).

Overview of the rest of the proof. To evaluate the rejection probability of $\mathcal{T}''$, we consider any $(X;Y) = (X_1, ..., X_{tN}; Y_{0^r}, ..., Y_{1^r})$ that is ϵ-far from $\mathcal{C}''$, where throughout the proof (unless said differently), all distances refer to sequences over Γ. Our aim is to prove that $\mathcal{T}''$ rejects this sequence with probability that is proportional to ϵ, and thus establishing that $\mathcal{T}''$ is a $\mathcal{C}''$-tester. The proof combines elements from the proof of Lemma 5.3 and Proposition 5.10. As in the proof of Proposition 5.10, we focus our attention on the case that $((X_1, ..., X_N)^t; Y_{0^r}, ..., Y_{1^r})$ is $\epsilon/2$-far from $\mathcal{C}''$, because the other case is handled by the replication test. We consider three (remaining) cases:

1. The sequence $(X_1, ..., X_N)$ is relatively far from a sequence of E-codewords.
2. The sequence of E-codewords closest to $(X_1, ..., X_N)$ is relatively far from the code $\mathcal{C}'$.
3. The sequence $(X_1, ..., X_N)$ is relatively close to a sequence of E-codewords, which in turn is relatively close to the code $\mathcal{C}'$. (In this case, the distance of $((X_1, ..., X_N)^t; Y)$ from $\mathcal{C}''$ is due to Y.)

Each of these cases will be handled by a corresponding claim. The first two cases correspond to the two cases considered in the proof of Lemma 5.3, whereas the third case was not relevant there.

Analogously to the proof of Lemma 5.3, we denote by ϵ_i the (relative) distance of X_i from E; that is, $\epsilon_i = \Delta_E(X_i)/n$.

Claim 5.13.1 (Case 1 – using $\epsilon' = \epsilon/4$): *If $\sum_{i=1}^N \epsilon_i/N > \epsilon'$ then the inner-verifier rejects with probability at least $\alpha\gamma \cdot \epsilon'$.*

Proof: Things would have been very easy if at least one of the queries made by $\mathcal{C}$-tester was uniformly distributed. In such a case, one of the input-oracles accessed by the inner-verifier would be at expected distance ϵ' from the code, and the inner-verifier would reject with probability at least $\gamma \cdot \beta\epsilon'$. Unfortunately, the aforementioned condition does not necessarily hold. Surely, we could modify the $\mathcal{C}$-tester to satisfy this condition, by adding a uniformly distributed query, but here we take an alternative route by recalling that (by the hypothesis that the tester makes "almost uniform" queries) the queries cover each possible location with sufficiently high probability.[16] Specifically, recall that $\sum_{j=1}^q p_i^{(j)} \geq \alpha q/N$, where $p_i^{(j)}$ denotes the probability that the $j^{\rm th}$ query of the $\mathcal{C}$-tester is to location i. Now, consider the oracles $X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}$ accessed by the inner-verifier, where $\omega \in \{0,1\}^r$ is uniformly selected by our tester. By the above,

$$\mathop{\mathbf{E}}_{\omega}\left[\sum_{j=1}^{q} \frac{\Delta_E(X_{i_{\omega,j}})}{n}\right] = \sum_{j=1}^{q}\sum_{i=1}^{N} p_i^{(j)} \cdot \epsilon_i \geq \frac{\alpha q}{N} \cdot \sum_{i=1}^{N} \epsilon_i > \alpha q \cdot \epsilon'$$

[16]Here we use the lower-bound on $\sum_{j=1}^q p_i^{(j)}$ provided by Eq. (31). Alternatively, we could prove that a different lower-bound follows by the fact that the tester must reject non-codewords with adequate probability. Specifically, let us denote by p_i the probability that, on a random invocation, *at least one* of the $\mathcal{C}$-tester's queries is to location i. Clearly, $p_i \leq \sum_{j=1}^q p_i^{(j)}$. On the other hand, we claim that $p_i \geq \beta/N$. The reason is that $\mathcal{C}$-tester must accept 0^N with probability 1, and reject $0^{i-1}10^{N-i}$ with probability at least β/N, but it cannot possibly distinguish the two cases unless it probes the $i^{\rm th}$ location. Thus, $\sum_{j=1}^q p_i^{(j)} \geq \beta/N$, for every $i \in [N]$.

which implies that, for uniformly distributed $\omega \in \{0,1\}^r$ and $j \in [q]$, the oracle $X_{i_{\omega,j}}$ is $\alpha\epsilon'$-far from E. Thus, the excepted deviation of $(X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}; Y_\omega)$ is at least $\alpha\epsilon'$ and the inner-verifier rejects with probability at least $\gamma \cdot \alpha\epsilon'$. ■

We are going to consider the E-codewords, denoted by c_i's, that are closest to these X_i's (i.e., $\Delta(X_i, c_i) = \epsilon_i n$). Let $\overline{c} = (c_1, ..., c_N)$. Let d_i be the E-decoding of c_i (i.e., $c_i = E(d_i)$), and $\overline{d} = (d_1, ..., d_N)$. Assuming that Case 1 does not hold, we have $(\overline{c}^t; Y)$ is $(\epsilon/4)$-far from $\mathcal{C}''$, because $\overline{c}^t$ is $(\epsilon/4)$-close to $(X_1, ..., X_N)$ and $((X_1, ..., X_N)^t; Y)$ is $(\epsilon/2)$-far from $\mathcal{C}''$. However, unlike in the proof of Lemma 5.3, the sequence $\overline{c}$ is not necessarily far from $\mathcal{C}'$, because the distance between $(\overline{c}^t, Y)$ and $\mathcal{C}''$ may be due to Y. Thus, there are two additional cases to consider.

Claim 5.13.2 (Case 2 – using $\epsilon'' = \alpha\delta\epsilon/4q$): *If $\overline{c}$ is ϵ''-far from $\mathcal{C}'$ then the inner-verifier rejects with probability at least $(\beta\gamma\delta/2) \cdot \epsilon''$.*

The condition $\sum_{i=1}^{N} \epsilon_i/N \le \epsilon'$ is omitted from this claim, because the claim holds regardless of this condition. The following proof is analogous to the treatment of Case 2 in the proof of Lemma 5.3.

Proof: By the claim's hypothesis, for every $x \in \Gamma^{bK/a}$, it holds that the set $D_x \stackrel{\text{def}}{=} \{i \in [N] : c_i \neq E(\mathcal{C}(x,i))\}$ has cardinality at least $\epsilon'' \cdot N$, because $\epsilon'' \cdot nN \le \Delta(\overline{c}, \mathcal{C}'(x)) \le |D_x| \cdot n$. Noting that $D_x = \{i \in [N] : d_i \neq \mathcal{C}(x,i)\}$, it follows that $\overline{d}$ is ϵ''-far from the code $\mathcal{C}$, when both are viewed as sequences over Σ. Thus, $\overline{d}$ would have been rejected by the $\mathcal{C}$-tester with probability at least $p \stackrel{\text{def}}{=} \beta \cdot \epsilon''$.

Let us call a choice of coins ω for the $\mathcal{C}$-tester good if the $\mathcal{C}$-tester would have rejected the values $(d_{i_{\omega,1}}, ..., d_{i_{\omega,q}})$; that is, $(d_{i_{\omega,1}}, ..., d_{i_{\omega,q}}) \notin L_\omega$. By the above, with probability p, the $\mathcal{C}$-tester selects good coins. On the other hand, for good coins ω, when given input L_ω and access to the input-oracles $c_{i_{\omega,1}}, ..., c_{i_{\omega,q}}$ (and any proof-oracle), the inner-verifier V rejects with constant probability (i.e., probability at least $\gamma \cdot \delta$). We need, however, to consider what happens when V is given access to the input-oracles $X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}$ (and the proof-oracle Y_ω), for a good ω.

We next show that, for a good ω, the inner-verifier V rejects the input-oracles $(X_{i_{\omega,1}}, ..., X_{i_{\omega,q}})$ with constant probability (regardless of Y_ω). This happens regardless of whether or not $X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}$ is close to $(c_{i_{\omega,1}}, ..., c_{i_{\omega,q}})$. We consider two cases:

1. Suppose that, for every $j \in [q]$, the oracle $X_{i_{\omega,j}}$ is $(\delta/2)$-close to $c_{i_{\omega,j}}$. Then, for every *acceptable* q-tuple $(a_1, ..., a_q)$ (i.e., $(a_1, ..., a_q) \in \{(E(z_1), ..., E(z_q)) : (z_1, ..., z_q) \in L_\omega\}$), there exists a j such that the oracle $X_{i_{\omega,j}}$ is $(\delta/2)$-far from a_j. The reason being that for every acceptable $(a_1, ..., a_q)$ there exists a j such that $a_j \neq c_{i_{\omega,j}}$, while on the other hand a_j must also be an E-codeword. Thus,
$$\frac{\Delta(X_{i_{\omega,j}}, a_j)}{n} \ge \frac{\Delta(c_{i_{\omega,j}}, a_j)}{n} - \frac{\Delta(X_{i_{\omega,j}}, c_{i_{\omega,j}})}{n} \ge \delta - \frac{\delta}{2}.$$
It follows that the deviation of $(X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}; Y_\omega)$ is at least $\delta/2$, and V rejects it with probability at least $\gamma \cdot \delta/2$.

2. Otherwise (i.e., some $X_{i_{\omega,j}}$ is $(\delta/2)$-far from the corresponding $c_{i_{\omega,j}}$), one of the input-oracles is $\delta/2$-far from being an E-codeword (because the $c_{i_{\omega,j}}$'s are the E-codewords closest to the $X_{i_{\omega,j}}$'s). Again, the deviation of $(X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}; Y_\omega)$ is at least $\delta/2$, and V rejects it with probability at least $\gamma \cdot \delta/2$.

We conclude that, for a good ω, when given access to $(X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}; Y_\omega)$, the inner-verifier V rejects with probability at least $\gamma \cdot \delta/2$. Recalling that a good ω is selected with probability at least $p = \beta \cdot \epsilon''$, it follows that V rejects with probability at least $\beta\epsilon'' \cdot \gamma\delta/2$. ■

Claim 5.13.3 (Case 3): *If $\sum_{i=1}^N \epsilon_i/N \le \epsilon/4$ and $\overline{c}$ is $(\alpha\delta\epsilon/4q)$-close to $\mathcal{C}'$ then the inner-verifier rejects with probability at least $(\gamma\delta/8)\cdot\epsilon$.*

Proof: Referring to the second hypothesis, let $x \in \Gamma^{bK/a}$ be such that $\overline{c}$ is $(\alpha\delta\epsilon/4q)$-close to $\mathcal{C}'(x)$, when both are viewed as nN-long sequences over Γ. Since both $\overline{c}$ and $\mathcal{C}'(x)$ are N-sequences of E-codewords, these sequences may differ on at most a $(\alpha\delta\epsilon/4q)/\delta$ fraction of these codewords; i.e., $|\{i : c_i \neq E(\mathcal{C}(x,i))\}| \le (\alpha\epsilon/4q)N$.

Combining the two hypotheses, it follows that $(X_1, ..., X_N)$ is $((\epsilon/4)+(\alpha\delta\epsilon/4q))$-close to $\mathcal{C}'(x)$, and thus Y is $(\epsilon/2)$-far from $\mathcal{P}'(x)$. (Note that for the last implication we only use the fact that $(X_1, ..., X_N)$ is $(\epsilon/2)$-close to $\mathcal{C}'(x)$ whereas $((X_1, ..., X_N)^t, Y)$ is $(\epsilon/2)$-far from $\mathcal{C}''(x)$.) For future usage, let us restate the fact that Y is $(\epsilon/2)$-far from $\mathcal{P}'(x)$ as follows

$$\mathop{\mathbf{E}}_{\omega}\left[\frac{\Delta(Y_\omega, P(L_\omega, \mathcal{C}(x, i_{\omega,1}), ..., \mathcal{C}(x, i_{\omega,q})))}{\ell}\right] \ge \frac{\epsilon}{2}. \tag{34}$$

We first analyze what happens when our procedure $\mathcal{T}''$ is given oracle access to $(\overline{c}^t, Y)$. Recalling that $|\{i : c_i \neq E(\mathcal{C}(x,i))\}| \le (\alpha\epsilon/4q)N$ and using the hypothesis that an average query of the $\mathcal{C}$-tester hits each location with probability at most α^{-1}/N, it follows that $\Pr_{\omega\in\{0,1\}^r, j\in[q]}[c_{i_{\omega,j}} \neq E(\mathcal{C}(x, i_{\omega,j}))] \le \alpha^{-1}\cdot(\alpha\epsilon/4q) = \epsilon/4q$. Thus, for a uniformly chose ω, with probability at least $1 - q\cdot\epsilon/4q$, the sequences $(c_{i_{\omega,1}}, ..., c_{i_{\omega,q}})$ and $(E(\mathcal{C}(x, i_{\omega,1})), ..., E(\mathcal{C}(x, i_{\omega,q})))$ are identical. Let us denote the set of these choices by G. Then,

$$G = \{\omega : (\forall j)\ d_{i_{\omega,j}} = \mathcal{C}(x, i_{\omega,j})\}, \text{ and } \Pr_\omega[\omega \in G] \ge 1 - (\epsilon/4). \tag{35}$$

We observe that, for any $\omega \in G$, the deviation of $((c_{i_{\omega,1}}, ..., c_{i_{\omega,q}}); Y_\omega)$ is lower-bounded by the minimum between $\Delta(Y_\omega, P(L_\omega, d_{i_{\omega,1}}, ..., d_{i_{\omega,q}}))/\ell$ and δ, where the first term is due to changing Y_ω to fit the $d_{i_{\omega,j}}$'s and the second term is due to changing at least one of the $c_{i_{\omega,j}}$'s so to obtain some other (acceptable) sequence of codewords. The minimum of the two terms is obviously lower-bounded by their product; that is, the deviation of $((c_{i_{\omega,1}}, ..., c_{i_{\omega,q}}); Y_\omega)$ is at least

$$\frac{\Delta(Y_\omega, P(L_\omega, d_{i_{\omega,1}}, ..., d_{i_{\omega,q}}))}{\ell}\cdot\delta. \tag{36}$$

Note that this lower-bound is in terms of the distance of Y_ω from the proof computed for the $d_{i_{\omega,j}}$'s, whereas Eq. (34) refers to the distance from the proof computed for the $\mathcal{C}(x, i_{\omega,j})$'s. Yet, recalling that the $d_{i_{\omega,j}}$'s equal the $\mathcal{C}(x, i_{\omega,j})$'s with probability at least $1-(\epsilon/4)$, (and using Eq. (34)) we have

$$\mathop{\mathbf{E}}_{\omega}\left[\frac{\Delta(Y_\omega, P(L_\omega, d_{i_{\omega,1}}, ..., d_{i_{\omega,q}}))}{\ell}\right] \ \ge\ \frac{\epsilon}{2} - \frac{\epsilon}{4} \ =\ \frac{\epsilon}{4}.. \tag{37}$$

Using Eq. (36), it follows that the expected deviation of $((c_{i_{\omega,1}}, ..., c_{i_{\omega,q}}); Y_\omega)$, when the expectation is taken uniformly over $\omega \in \{0,1\}^r$, is at least $\delta\epsilon/4$ (and so V rejects $(\overline{c}^t, Y)$ with probability at least $\gamma\delta\epsilon/4$).

However, we need to estimate the deviation of $((X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}); Y_\omega)$, which we do next (in a way analogous to Case 2). For $\omega \in G$, we lower-bound the deviation of $((X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}); Y_\omega)$ by considering two cases (as in the proof of Claim 5.13.2):

1. Suppose that, for every $j \in [q]$, the oracle $X_{i_{\omega,j}}$ is $(\delta/2)$-close to $c_{i_{\omega,j}}$. Recall that $(c_{i_{\omega,1}}, ..., c_{i_{\omega,q}})$ equals $(E(\mathcal{C}(x, i_{\omega,1})), ..., E(\mathcal{C}(x, i_{\omega,q})))$, or equivalently $(d_{i_{\omega,1}}, ..., d_{i_{\omega,q}})$ equals $(\mathcal{C}(x, i_{\omega,1}), ..., \mathcal{C}(x, i_{\omega,q}))$. Thus, the deviation of $((X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}); Y_\omega)$ is lower-bounded by

the minimum between $\Delta(Y_\omega, P(L_\omega, \mathcal{C}(x, i_{\omega,1}), ..., \mathcal{C}(x, i_{\omega,q})))/\ell$ and $\delta - (\delta/2)$, where the first term is due to changing Y_ω to fit the $\mathcal{C}(x, i_{\omega,j})$'s and the second term is due to changing at least one of the $X_{i_{\omega,j}}$'s so to obtain some other (acceptable) sequence of codewords. As before, we lower-bound the deviation by the product of these terms, yielding half the value of Eq. (36).

2. Otherwise (i.e., some $X_{i_{\omega,j}}$ is $(\delta/2)$-far from the corresponding $c_{i_{\omega,j}}$), one of the input-oracles is $\delta/2$-far from being an E-codeword (because the $c_{i_{\omega,j}}$'s are the E-codewords closest to the $X_{i_{\omega,j}}$'s). As in the proof of Claim 5.13.2, in this case, the deviation of $((X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}); Y_\omega)$ is at least $\delta/2$.

We conclude that, for $\omega \in G$, the deviation of $((X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}); Y_\omega)$ is at least half the value of Eq. (36). Using Eq. (37), we lower-bound the expected deviation of $((X_{i_{\omega,1}}, ..., X_{i_{\omega,q}}); Y_\omega)$, where the expectation is taken uniformly over $\omega \in \{0,1\}^r$, by $(\delta/2) \cdot (\epsilon/4)$. It follows that the inner-verifier V rejects with probability at least $\gamma \cdot \delta\epsilon/8$. ■

Combining Claims 5.13.1–5.13.3, while setting $\epsilon' = \epsilon/4$ and $\epsilon'' = \alpha\delta\epsilon/4q$, it follows that the inner-verifier rejects with probability at least

$$\min\left(\alpha\gamma\epsilon'; \frac{\beta\gamma\delta\epsilon''}{2}; \frac{\gamma\delta\epsilon}{8}\right) = \min\left(\frac{\alpha\gamma}{4}; \frac{\alpha\beta\gamma\delta^2}{8q}; \frac{\gamma\delta}{8}\right) \cdot \epsilon = \frac{\alpha\beta\gamma\delta^2}{8q} \cdot \epsilon .$$

Recalling that the inner-verifier is invoked with probability 1/2 (and otherwise the repetition test is invoked with much better corresponding performance), the theorem follows. ■

Preserving the almost-uniform queries property. Note that the proof of Theorem 5.13 yields a codeword tester that does not make almost uniform queries. We wish to redeem this state of affairs, both for the sake of elegancy and for future use in Section 5.6. This requires a minor modification of the construction presented in the proof of Theorem 5.13 as well as using a LIPS that makes almost uniform queries (as defined next). We stress that our main results (e.g., Theorem 2.3) do not use the following Theorem 5.15, but we will need the following definition in any case (i.e., also in case we do not use Theorem 5.15).

Definition 5.14 (LIPS with almost uniform queries): *For $\alpha \in (0,1]$, a verifier* (of an $(F, (q,b) \to (p,a), \cdot, \cdot)$-LIPS) *is said to make α-*uniform queries to its oracles *if for each of its oracles and each location in that oracle the probability that a random query to that oracle equals that location is proportional to the oracle's length, where the constant of proportion is in $[\alpha, \alpha^{-1}]$. That is, for every $i \in [q+1]$, we denote by $p_j(i)$ the probability that a random query to the j^{th} oracle is to location i. We require that, for every $j \in [q]$ and $i \in [n]$, it holds that $\alpha/n \leq p_j(i) \leq \alpha^{-1}/n$, and that, for every $i \in [\ell]$, it holds that $\alpha/\ell \leq p_{p+1}(i) \leq \alpha^{-1}/\ell$, where n and ℓ are the length of the encoding and proof, respectively.*

Note that the definition only requires almost uniformity of the queries made to each individual oracle, and nothing is required regarding the proportion of queries made to the different oracles.

Theorem 5.15 (Theorem 5.13, revisited): *Suppose we are given a locally testable code and a LIPS as in the hypothesis of Theorem 5.13. Furthermore, suppose that the LIPS makes α_V-uniform queries to its oracles and that $2^r\ell > 2Nn$. Then, there exists an F-linear locally-testable code as in the conclusion of Theorem 5.13. Furthermore, for $t \stackrel{\text{def}}{=} \lceil 2^r\ell/Nn \rceil$, this code can be tested by making*

2p queries that are $((1-t^{-1})\cdot\alpha\alpha_V)$*-uniform and tossing* $1+(\log_2 t)+\max(r+r_V,\log M)$ *coins such that* ϵ*-far sequences are rejected with probability at least* $(\alpha\beta\gamma\delta^2/16q)\cdot\epsilon$*, where* $M = tNn + 2^r\ell < 2^{r+1}\ell$*.*

Theorem 5.15 provides a codeword tester that makes almost uniform queries (whereas the codeword tester provided by Theorem 5.13 does not have the feature). This is done at the expense of doubling the query complexity (i.e., from p to $2p$), and adding a term of $\log_2 t$ to the randomness complexity. Typically, $r_V > 1+\log\ell$ and so $\log_2 t < r+r_V-\log N$ (and $\log_2 M < r+1+\log\ell < r+r_V$), where r (resp., r_v) is the randomness complexity of the original codeword tester (resp., the LIPS verifier) and N is the length of the original code. In this case, the randomness complexity grows from $r+r_V+1$ to less than $2\cdot(r+r_V+1)-\log_2 N$. We note that in our applications $r+r_V = (1+o(1))\cdot\log_2 N$, and so the increase in the randomness complexity merely doubles the $o(1)$ term.

Proof: We use the same code $\mathcal{C}''$ as constructed in the proof of Theorem 5.13, and slightly modify the codeword tester presented there. Instead of emulating the $\mathcal{C}$-tester using the first N encodings (in the tested word), we use the i^{th} block of such N encodings, for a uniformly chosen $i\in[t]$. The replication test is modified accordingly (i.e., we compare this block to the i'-th block, for a uniformly chosen $i'\in[t]$). In addition, we add dummy queries such that the resulting tester makes an equal number of queries to each of the two parts of the tested word (i.e., the tNn-long prefix and the suffix). Each of these dummy queries is uniformly distributed in the corresponding part (and the answer is ignored by the tester). The purpose of these modification is to obtain a codeword tester that makes almost uniform queries.

Analogously to Remark 3.6 (see also the proof of Proposition 5.5), the (soundness) analysis presented in the proof of Theorem 5.13 remains valid. Thus, we focus on the syntactic conditions.

Clearly, it suffices to use p dummy queries, and thus the query complexity of the new tester is $2p$. By choosing a careful implementation (which recycles randomness in order to implement the dummy queries[17]), the randomness complexity increases only by a $\log_2 t$ term (for selecting the index $i\in[t]$ as mentioned above).

It remains to analyze the uniformity of the queries made by the tester. The dummy queries guarantee that each of the two parts of the tested word be probed the same number of times, and furthermore that the distribution of the dummy queries does not skew the distribution in each of the two parts. Since the two parts are of almost the same length (i.e., upto a factor of about $1-t^{-1}$), it suffices to analyze the distribution in each part.

For the first part (i.e., the tNn-long prefix), we combine the hypothesis that the $\mathcal{C}$-tester makes α-uniform queries with the fact that we use a random copy of $\mathcal{C}'$. This means that the q input-oracles that we select (for the inner verifier) are almost uniformly distributed among the tN encodings (of length n each). Using the hypothesis that the LIPS makes α_V-uniform queries to each of its oracles (and thus to each of its input-oracle), it follows that the queries made to the first part are $\alpha\alpha_V$-uniform.

For the second part (i.e., the proof part), we combine the fact that the tester selects a uniformly distributed proof (i.e., $\omega\in\{0,1\}^r$ is uniformly distributed) with the hypothesis that the LIPS makes α_V-uniform queries to each of its oracles (and thus to its proof-oracle). It follows that the queries made to the second part are α_V-uniform. The theorem follows. ■

[17]Note that the number of coins exceeds $\log_2 M$, and thus we may re-use these coins to select a random position for the dummy queries. It does not matter that all these dummy positions will be identical, nor that they are correlated with the other queries.

5.3.4 Composing inner verifiers

Subsection 5.3.3 (e.g., Theorem 5.13) refers to the composition of an outer *code* with an inner-verifier yielding a new *code*. In contrast, the following theorem refers to composing two inner-verifiers yielding a new inner-verifier. Indeed, we could have worked only with Theorem 5.13 (or actually with Theorem 5.15), but it seems more convenient to (have and) work with both types of composition theorems.[18] As in the case of Theorem 5.13, we need to assume that the outer construct (in this case the outer LIPS) makes almost uniform queries; the reader is thus referred to Definition 5.14. We comment that the resulting LIPS does not necessarily make almost uniform queries; we will redeem this state of affairs at a later point (in Theorem 5.17).

Theorem 5.16 (composition of linear inner-verifiers): *Consider a finite field F, real numbers $\alpha_1, \gamma_1, \gamma_2, \delta_1, \delta_2 \in (0,1]$, and integers $b > b' > b''$ such that b'' divides b', which divides b. Suppose that there exist a $(F,(p,b) \to (p',b'),\delta_1,\gamma_1)$-LIPS that makes α_1-uniform queries to its oracles and a $(F,(p',b') \to (p'',b''),\delta_2,\gamma_2)$-LIPS. Then, there exists a $(F,(p,b) \to (p'',b''),\delta_1\delta_2,\gamma)$-LIPS, where $\gamma = \alpha_1\gamma_1\gamma_2\delta_2/8p'$. Furthermore, if the i^{th} original LIPS uses r_i coins, encoding length n_i and proof length ℓ_i, then the resulting LIPS uses $r_1 + r_2$ coins, encoding length $n_1 \cdot n_2$ and proof length $\ell_1 \cdot n_2 + 2^{r_1} \cdot \ell_2$.*

Proof: We start with the construction, which is analogous to the one used in the proof of Theorem 5.13 (except that no replication is needed here). The basic idea is to start from the $(F,(p,b) \to (p',b'),\delta_1,\gamma_1)$-LIPS, which uses p input-oracles that are supposed to be encoded using a function $E_1 : F^b \to (F^{b'})^{n_1}$ and a proving function with range $(F^{b'})^{\ell_1}$, and encode each of the $F^{b'}$ symbols using a function $E_2 : F^{b'} \to (F^{b''})^{n_2}$ (i.e., the encoding function of the second LIPS). This yields p new input-oracles and a part of the new proof-oracle. In addition, we use the proving function of the second LIPS to produce auxiliary proofs for each of the possible coin tosses of the first (i.e., outer) verifier. The concatenation of these auxiliary proofs yields the second part of the new proof-oracle. The new verifier will check the execution of the first (i.e., outer) verifier by invoking the second verifier and giving it access to the suitable oracles, which are blocks in the oracles to which the new verifier is given access. Specifically, given a $(F,(p,b) \to (p',b'),\delta_1,\gamma_1)$-LIPS, denoted (E_1,P_1,V_1), and a $(F,(p',b') \to (p'',b''),\delta_2,\gamma_2)$-LIPS, denoted (E_2,P_2,V_2), we define their composition, denoted (E,P,V), as follows:

- The encoding function $E : F^b \to (F^{b''})^{n_1 \cdot n_2}$ is the concatenation of the encoding functions $E_1 : F^b \to (F^{b'})^{n_1}$ and $E_2 : F^{b'} \to (F^{b''})^{n_2}$. That is, for $x \in (F^{b''})^{b/b''} \equiv (F^{b'})^{b/b'} \equiv F^b$, we have $E(x) = (E_2(y_1),\ldots,E_2(y_{n_1}))$, where $(y_1,\ldots,y_{n_1}) \stackrel{\text{def}}{=} E_1(x)$.
- The proving function $P = (P^{(1)},P^{(2)})$ operates as follows: Given $L \in \mathcal{L}_{F,pb}$ and $x_1,\ldots,x_p \in F^b$, the first part of the proof (i.e., $P^{(1)}(L,x_1,\ldots,x_p)$) is the symbol-by-symbol encoding under E_2 of $P_1(L,x_1,\ldots,x_p) \in (F^{b'})^{\ell_1}$. That is, $P^{(1)}(L,x_1,\ldots,x_p) = (E_2(y_1),\ldots,E_2(y_{\ell_1}))$, where $(y_1,\ldots,y_{\ell_1}) \stackrel{\text{def}}{=} P_1(L,x_1,\ldots,x_p)$.

  The second part of the proof (i.e., $P^{(2)}(L,x_1,\ldots,x_p)$) consists of 2^{r_1} blocks corresponding to each of the 2^{r_1} possible checks of V_1. For each $\omega_1 \in \{0,1\}^{r_1}$, the block corresponding to ω_1

[18] An analogous comment applies to the construction of PCP systems. That is, it suffices to have a composition theorem that refers to using a standard PCP as an outer verifier and composes it with an inner-verifier (as done in [3, 2] and most subsequent works). However, it is useful to consider also the composition of two inner-verifiers (i.e., the composition of PCPPs [10] or assignment testers [17]). We note that the following composition result predates [10, 17].

in $P^{(2)}(L, x_1, \ldots, x_p)$ is the ℓ_2-long sequence $P_2(L_{\omega_1}, z_{\omega_1,1}, \ldots, z_{\omega_1,p'})$, where $z_{\omega_1,1}, \ldots, z_{\omega_1,p'}$ denote the p' symbols (i.e., $F^{b'}$-symbols) of $E_1(x_1), \ldots, E_1(x_p)$ and $P_1(L, x_1, \ldots, x_p)$ that are inspected by $V_1(L, \omega_1)$, and L_{ω_1} is the conjunction of F-linear conditions checked by V_1. That is, if the j^{th} query of $V_1(L, \omega_1)$ is to location ℓ_j of its i_j^{th} input-oracle (resp., of its proof-oracle), then $z_{\omega_1,j}$ equals the ℓ_j^{th} symbol in $E_1(x_{i_j})$ (resp., in $P_1(L, x_1, \ldots, x_p)$).

Note that the proof length is $\ell_1 \cdot n_2 + 2^{r_1} \cdot \ell_2$, where the first (resp., second) term corresponds to $P^{(1)}$ (resp., $P^{(2)}$).

- The verifier V is given $L \in \mathcal{L}_{F,pb}$ as well as oracle access to p input-oracles, denoted $X_1, \ldots, X_p$, and to a proof-oracle, denoted $\Pi = (\Pi^{(1)}, \Pi^{(2)})$. The input-oracle $X_i : [n_1 n_2] \to F^{b''}$ is viewed as consisting of n_1 blocks, each of length n_2, and $\Pi^{(1)} : [\ell_1 n_2] \to F^{b''}$ is viewed as consisting of ℓ_1 such blocks. We also view $\Pi^{(2)} : [2^{r_1} \ell_2] \to F^{b''}$ as $\langle \Pi^{(2)}_{\omega_1} : \omega_1 \in \{0,1\}^{r_1} \rangle$, where $\Pi^{(2)}_{\omega_1} : [\ell_2] \to F^{b''}$.

  Note that $X_1, \ldots, X_p$ and $\Pi^{(1)}$ are supposed to be the encodings, under E_2, of corresponding oracles $X'_1, \ldots, X'_p$ and Π' that are of the format expected by V_1. Intuitively, V checks the claim that V_1 would have accepted these $X'_1, \ldots, X'_p$ and Π'. The verifier V does so by selecting a check for V_1, and using V_2 to verify the corresponding check, while utilizing a proof that is part of $\Pi^{(2)}$.

  Specifically, on input $L \in \mathcal{L}_{F,pb}$ and coins $(\omega_1, \omega_2) \in \{0,1\}^{r_1+r_2}$, the verifier $V(L, (\omega_1, \omega_2))$ operates as follows:

  1. It determines the queries $q_{\omega_1,1}, \ldots, q_{\omega_1,p'}$ that $V_1(L, \omega_1)$ makes into its $p+1$ oracles (i.e., to $X'_1, \ldots, X'_p$ and Π') on randomness ω_1, and the conjunction of linear conditions L'_{ω_1} that $V_1(L, \omega_1)$ needs to verify on the p' responses.

     Note that each of these p' queries is actually a pair indicating an oracle and a position in it; that is, $q_{\omega_1,j} = (i_{\omega_1,j}, q'_{\omega_1,j})$, where $i_{\omega_1,j} \in [p+1]$ and $q'_{\omega_1,j} \in [k]$ such that $k = n_1$ if $i_{\omega_1,j} \in [p]$ and $k = \ell_1$ otherwise.
  2. Next, V invokes $V_2(L'_{\omega_1}, \omega_2)$ providing it with oracle access to the input-oracles as determined by $q_{\omega_1,1}, \ldots, q_{\omega_1,p'}$ and to the proof-oracle that is the block of $\Pi^{(2)}$ that corresponds to ω_1. Specifically, for every $j = 1, ..., p''$, the j^{th} input-oracle of V_2, denoted X''_j, is defined to equal the $q'_{\omega_1,j}$-th n_2-long block of $X_{i_{\omega_1,j}}$ if $i_{\omega_1,j} \in [p]$ and the $q'_{\omega_1,j}$-th n_2-long block of $\Pi^{(1)}$ otherwise (i.e., $X''_j(i) = X_{i_{\omega_1,j}}((i_{\omega_1,j} - 1) \cdot n_1 + i)$ if $i_{\omega_1,j} \in [p]$ and $X''_j(i) = \Pi^{(1)}(p \cdot n_1 + i)$ otherwise). The proof-oracle of V_2, denoted Π'', is defined to equal $\mathcal{P}^{(2)}_{\omega_1}$.
  3. The verifier V accepts if and only V_2 accepts.

Aside from the (strong) soundness requirement, it is clear that the resulting LIPS satisfies all other requirements. To evaluate the rejection probability of the latter, we consider any $(X_1, ..., X_p, (\Pi^{(1)}, \Pi^{(2)}))$, where $X_i : [n_1 n_2] \to F^{b''}$, $\Pi^{(1)} : [\ell_1 n_2] \to F^{b''}$ and $\Pi^{(2)} : [2^{r_1} \ell_2] \to F^{b''}$. The analysis follows the outline of the proof of Theorem 5.13. Specifically, we consider three cases (which correspond to the three (main) cases considered in the proof of Theorem 5.13):

Case 1: Either some X_i (for $i \in [p]$) or $\Pi^{(1)}$ is relatively far from a sequence of E_2-codewords. In this case, V_2 is given access to some oracle (i.e., a block of either X_i or $\Pi^{(1)}$) that is far (on the average) from a E_2-codeword, and rejects with proportional probability. For details, see Claim 5.16.1.

Otherwise, we let $X'_1, ..., X'_p$ and Π' denote the corresponding sequence of E_2-decodings. That is, encoding the elements of X'_i (resp., Π') under E_2 yields the sequence of E_2-codewords that is closest to X_i (resp., $\Pi^{(1)}$).

Case 2: The deviation of $(X'_1, ..., X'_p, \Pi')$ with respect to (E_1, P_1, V_1) is relatively big. In this case, with proportional probability V_1 determines p' positions in $(X'_1, ..., X'_p, \Pi')$ such that their contents violates the linear condition (checked by V_1). In the latter case, V_2 is given access to a corresponding sequence of p' oracles that has a big deviation, and rejects with constant probability. Thus, V rejects with probability proportional to the deviation of $(X'_1, ..., X'_p, \Pi')$. For details, see Claim 5.16.2.

Case 3: Otherwise, $(X_1, ..., X_p, \Pi^{(1)})$ is close to the sequence of E_2-encodings of $(X'_1, ..., X'_p, \Pi')$, which in turn has a relatively small deviation with respect to (E_1, P_1, V_1). It follows that $\Pi^{(2)}$ is far from the corresponding sequence of canonical proofs, and V_2 rejects with probability proportional to the latter distance. For details, see Claim 5.16.3.

The proofs of the aforementioned claims are very similar to the proofs of the corresponding claims in the proof of Theorem 5.13. The key difference is that we refer to the deviation of sequences of oracles (as defined in Definition 5.9) rather than to distances from codewords.

Suppose that $(X_1, ..., X_p, (\Pi^{(1)}, \Pi^{(2)}))$ has deviation ϵ with respect to the linear inner proof system (E, P, V). Our aim is to show that these oracles will be rejected by V with probability proportional to ϵ. We will use the following notations:

- To simplify notations, let use denote $X_{p+1} \stackrel{\text{def}}{=} \Pi^{(1)}$. Let $k_i = n_1$ if $i \in [p]$ and $k_{p+1} = \ell_1$.
- For every $i \in [p+1]$, let $X_i = (w_{i,1}, ..., w_{i,k_i})$, where $w_{i,j} \in (F^{b''})^{n_2}$, and $\epsilon_{i,j} = \Delta_{E_2}(w_{i,j})/n_2$ for $j \in [k_i]$.

That is, $\epsilon_{i,j}$ is the relative distance of the j^{th} block (of length n_2) in X_i from an E_2-codeword.

Claim 5.16.1 (Case 1 – using $\epsilon' = \epsilon/4$): *If for some $i \in [p+1]$ it holds that $\sum_{j=1}^{k_i} \epsilon_{i,j}/k_i > \epsilon'$ then V rejects with probability at least $\alpha_1\gamma_2 \cdot \epsilon'$.*

Proof: Recalling that V_1 makes α_1-uniform queries to each of its oracles, we note that the queries of V_1 to its i^{th} oracle correspond to n_1-long blocks in X_i that are at *expected* (relative) distance at least $\alpha_1 \cdot \epsilon'$ from E_2. Thus, V_2 is given access to p' oracles that have an expected deviation of at least $\alpha_1\epsilon'$, and so V_2 rejects with probability at least $\gamma_2 \cdot \alpha_1\epsilon'$, where the probability is taken over the random choices of ω_1 and ω_2. The claim follows. ■

By Claim 5.16.1, we may focus on the case that $\sum_{j=1}^{k_i} \epsilon_{i,j}/k_i \leq \epsilon'$ (for every $i \in [p+1]$). We are going to consider the E_2-codewords, denoted by $c_{i,j}$'s, that are closest to the $w_{i,j}$'s (i.e., $\Delta(w_{i,j}, c_{i,j}) = \epsilon_{i,j}n_2$). Let $d_{i,j}$ be the E_2-decoding of $c_{i,j}$ (i.e., $c_{i,j} = E_2(d_{i,j})$), and $X'_i = (d_{i,1}, ..., d_{i,k_1})$.

Claim 5.16.2 (Case 2 – using $\epsilon'' = \alpha_1\epsilon/4p'$): *If the deviation of $(X'_1, ..., X'_{p+1})$ with respect to (E_1, P_1, V_1) is at least ϵ'' then V rejects with probability at least $(\gamma_1\gamma_2\delta_2/2) \cdot \epsilon''$.*

The condition $\sum_{j=1}^{k_i} \epsilon_{i,j}/k_i \leq \epsilon'$ (for every i) is omitted from Claim 5.16.2, because this claim holds regardless of this condition.

Proof: By the claim's hypothesis (and the strong soundness of V_1), the verifier V_1 would have been rejected $(X'_1, ..., X'_{p+1})$ with probability at least $p \stackrel{\rm def}{=} \gamma_1 \cdot \epsilon''$. Let us call a choice of coins ω_1 for V_1 good if V_1 would have rejected the values $(d_{q_{\omega_1,1}}, ..., d_{q_{\omega_1,p'}})$; that is, $(d_{q_{\omega_1,1}}, ..., d_{q_{\omega_1,p'}}) \notin L'_{\omega_1}$. (Recall that $q_{\omega_1,j} = (i_{\omega_1,j}, q'_{\omega_1,j})$, where $i_{\omega_1,j} \in [p+1]$ and $q'_{\omega_1,j} \in [k_{i_{\omega_1,j}}]$, and that $d_{q_{\omega_1,j}}$ is the $q'_{\omega_1,j}$-th symbol in $X'_{i_{\omega_1,j}}$.) By the above, with probability p, a uniformly selected choice of $\omega_1 \in \{0,1\}^{r_1}$ is good. On the other hand, for good ω_1, when given input L'_{ω_1} and access to the input-oracles $(c_{q_{\omega_1,1}}, ..., c_{q_{\omega_1,p'}}) = (E_2(d_{q_{\omega_1,1}}), ..., E_2(d_{q_{\omega_1,p'}}))$ (and any proof-oracle), the verifier V_2 rejects with constant probability (i.e., probability at least $\gamma_2 \cdot \delta_2$). We need, however, to consider what happens when V_2 is given access to the input-oracles $X_{q_{\omega_1,1}}, ..., X_{q_{\omega_1,p'}}$ (and the proof-oracle $\Pi^{(2)}_{\omega_1}$), for a good ω_1. We next show that, for a good ω_1, the verifier V_2 rejects the input-oracles $\overline{X}_{\omega_1} \stackrel{\rm def}{=} (X_{q_{\omega_1,1}}, ..., X_{q_{\omega_1,p'}})$ with constant probability (regardless of $\Pi^{(2)}_{\omega_1}$). This happens regardless of whether or not $\overline{X}_{\omega_1}$ is close to $(c_{q_{\omega_1,1}}, ..., c_{q_{\omega_1,p'}})$. We consider two cases:

1. Suppose that, for every $j \in [p']$, the oracle $X_{q_{\omega_1,j}}$ is $(\delta_2/2)$-close to $c_{q_{\omega_1,j}}$. Then, for every *acceptable* p'-tuple (i.e., $(a_1, ..., a_{p'}) \in \{(E_2(z_1), ..., E_2(z_{p'})) : (z_1, ..., z_{p'}) \in L'_{\omega_1}\}$), there exists a j such that the oracle $X_{q_{\omega_1,j}}$ is $(\delta_2/2)$-far from the $j^{\rm th}$ element in the acceptable sequence (i.e., from a_j). The reason being that for every acceptable $(a_1, ..., a_{p'})$ there exists a j such that $a_j \neq c_{q_{\omega_1,j}}$, while on the other hand a_j must also be an E_2-codeword. It follows that the deviation of $(\overline{X}_{\omega_1}; \Pi^{(2)}_{\omega_1})$ with respect to (E_2, P_2, V_2) is at least $\delta_2/2$, and V_2 rejects it with probability at least $\gamma_2 \cdot \delta_2/2$.

2. Otherwise (i.e., some $X_{q_{\omega_1,j}}$ is $(\delta_2/2)$-far from the corresponding $c_{q_{\omega_1,j}}$), one of the input-oracles is $\delta/2$-far from being an E_2-codeword (because the $c_{q_{\omega_1,j}}$'s are the E_2-codewords closest to the $X_{q_{\omega_1,j}}$'s). Again, the deviation of $(\overline{X}_{\omega_1}; \Pi^{(2)}_{\omega_1})$ is at least $\delta_2/2$, and V_2 rejects it with probability at least $\gamma_2 \cdot \delta_2/2$.

We conclude that, for a good ω_1, when given access to $(\overline{X}_{\omega_1}; \Pi^{(2)}_{\omega_1})$ the verifier V_2 rejects with probability at least $\gamma_2 \cdot \delta_2/2$. Recalling that a good ω_1 is selected with probability at least $p = \gamma_1 \epsilon''$, it follows that V rejects with probability at least $\gamma_1 \epsilon'' \cdot \gamma_2 \delta_2/2$. ■

Claim 5.16.3 (Case 3): *If for every $i \in [p+1]$ it holds that $\sum_{j=1}^{k_i} \epsilon_{i,j}/k_i \leq \epsilon/4$ and the deviation of $(X'_1, ..., X'_{p+1})$ with respect to (E_1, P_1, V_1) is at most $(\alpha_1 \epsilon/4p')$ then V rejects with probability at least $(\gamma_2 \delta_2/8) \cdot \epsilon$.*

Proof: Referring to the second hypothesis, let $(x_1, ..., x_p) \in L$ (where $L \in \mathcal{L}_{F,pb}$) be such that for every $i \in [p]$ the input-oracle X'_i is $(\alpha\epsilon/4p')$-close to $(y_{i,1}, ..., y_{i,n_1}) \stackrel{\rm def}{=} E_1(x_i)$ and X'_{p+1} is $(\alpha\epsilon/4p')$-close to $(y_{p+1,1}, ..., y_{p+1,\ell_1}) \stackrel{\rm def}{=} P_1(L, x_1, ..., x_p)$, when all these objects are viewed as sequences over $F^{b'}$. It follows that the E_2-encodings of these objects (i.e., $(c_{i,1}, ..., c_{i,k_i})$ and $(E_2(y_{i,1}), ..., E_2(y_{i,k_i}))$) differ on at most a $(\alpha\epsilon/4p')$ fraction of these E_2-codewords; that is, for every i, we have $|\{j : c_{i,j} \neq E_2(y_{i,j})\}| \leq (\alpha\epsilon/4p') \cdot k_i$.

Combining the two hypotheses, it follows that each X_i is $((\epsilon/4) + (\alpha_1\epsilon/4p'))$-close to $(E_2(y_{i,1}), ..., E_2(y_{i,k_i}))$, and thus $Y \stackrel{\rm def}{=} \Pi^{(2)}$ is $(\epsilon/2)$-far from $P^{(2)}(L, x_1, ..., x_p)$. (Note that for the last implication we only use the fact that each X_i is $(\epsilon/2)$-close to $(y_{i,1}, ..., y_{i,k_i})$, whereas the deviation of $(X_1, ..., X_p; (X_{p+1}, Y))$ with respect to (E, R, V) is ϵ.) For future usage, let us restate the fact that $Y = \langle Y_{\omega_1} : \omega_1 \in \{0,1\}^{r_1} \rangle$ is $(\epsilon/2)$-far from $P^{(2)}(L, x_1, ..., x_p) =$

$\langle P_2(L'_{\omega_1}, y_{q_{\omega_1,1}}, ..., y_{q_{\omega_1,p'}}) : \omega_1 \in \{0,1\}^{r_1} \rangle$ as follows

$$\mathop{\mathbf{E}}_{\omega}\left[\frac{\Delta(Y_{\omega_1}, P_2(L'_{\omega_1}, y_{q_{\omega_1,1}}, ..., y_{q_{\omega_1,p'}}))}{\ell_2}\right] \geq \frac{\epsilon}{2}\,. \tag{38}$$

We first analyze what happens when V is given oracle access to $(\overline{c}_1, ..., \overline{c}_p; (\overline{c}_{p+1}, Y))$, where $\overline{c}_i \stackrel{\rm def}{=} (c_{i,1}, ..., c_{i,k_i})$. Recalling that $|\{j : c_{i,j} \neq E_2(y_{i,j})\}| \leq (\alpha\epsilon/4p') \cdot k_i$ and using the hypothesis that V_1 makes α_1-uniform queries to each of its oracles, it follows that $\Pr_{\omega_1 \in \{0,1\}^{r_1}, j \in [p']}[c_{q_{\omega_1,j}} \neq E_2(y_{q_{\omega_1,j}})] \leq \alpha_1^{-1} \cdot (\alpha_1\epsilon/4p') = \epsilon/4p'$. Thus, for a uniformly chose ω_1, with probability at least $1 - p' \cdot \epsilon/4p'$, the sequences $(c_{q_{\omega_1,1}}, ..., c_{q_{\omega_1,p'}})$ and $(E_2(y_{q_{\omega_1,1}}), ..., E_2(y_{q_{\omega_1,p'}}))$ are identical. Let us denote the set of these choices by G. Then,

$$G = \{\omega_1 : (\forall j)\ d_{q_{\omega_1,j}} = y_{q_{\omega_1,j}}\}, \text{ and } \Pr_{\omega_1}[\omega_1 \in G] \geq 1 - (\epsilon/4). \tag{39}$$

We observe that, for any $\omega_1 \in G$, the deviation of $((c_{q_{\omega_1,1}}, ..., c_{q_{\omega_1,p'}}); Y_{\omega_1})$ with respect to (E_2, P_2, V_2) is lower-bounded by the minimum between $\Delta(Y_{\omega_1}, P_2(L'_{\omega_1}, d_{q_{\omega_1,1}}, ..., d_{q_{\omega_1,p'}}))/\ell_2$ and δ_2, where the first term is due to changing Y_{ω_1} to fit the $d_{q_{\omega_1,j}}$'s and the second term is due to changing at least one of the $c_{q_{\omega_1,j}}$'s so to obtain some other (acceptable) sequence of codewords. The minimum of the two terms is obviously lower-bounded by their product; that is, the deviation of $((c_{q_{\omega_1,1}}, ..., c_{q_{\omega_1,p'}}); Y_{\omega_1})$ is at least

$$\frac{\Delta(Y_{\omega_1}, P_2(L'_{\omega_1}, d_{q_{\omega_1,1}}, ..., d_{i_{\omega_1,p'}}))}{\ell_2} \cdot \delta_2\,. \tag{40}$$

Note that this lower-bound is in terms of the distance of Y_{ω_1} from the proof computed for the $d_{q_{\omega_1,j}}$'s, whereas Eq. (38) refers to the distance from the proof computed for the $y_{q_{\omega_1,j}}$'s. Yet, recalling that the $d_{q_{\omega,j}}$'s equal the $y_{q_{\omega_1,j}}$'s with probability at least $1 - (\epsilon/4)$, (and using Eq. (38)) we have

$$\mathop{\mathbf{E}}_{\omega_1}\left[\frac{\Delta(Y_{\omega_1}, P_2(L'_{\omega_1}, d_{q_{\omega_1,1}}, ..., d_{i_{\omega_1,p'}}))}{\ell_2}\right] \;\geq\; \frac{\epsilon}{4}\,.. \tag{41}$$

Using Eq. (40), it follows that the expected deviation of $((c_{q_{\omega_1,1}}, ..., c_{q_{\omega_1,p'}}); Y_{\omega_1})$, when the expectation is taken uniformly over $\omega_1 \in \{0,1\}^{r_1}$, is at least $\delta_2\epsilon/4$ (and so V rejects $(\overline{c}_1, ..., \overline{c}_p, (\overline{c}_{p+1}, Y))$ with probability at least $\gamma_2\delta_2\epsilon/4$).

However, we need to estimate the deviation of $(\overline{X}_{\omega_1}; Y_{\omega_1})$, where $\overline{X}_{\omega_1} = (X_{q_{\omega_1,1}}, ..., X_{q_{\omega_1,p'}})$, which we do next. For $\omega_1 \in G$, we lower-bound the deviation of $(\overline{X}_{\omega_1}; Y_{\omega_1})$ by considering two cases (as in the proof of Claim 5.16.2):

1. Suppose that, for every $j \in [p']$, the oracle $X_{q_{\omega_1,j}}$ is $(\delta_2/2)$-close to $c_{q_{\omega_1,j}}$. Then, the deviation of $((X_{i_{\omega_1,1}}, ..., X_{i_{\omega_1,p'}}); Y_{\omega_1})$ is lower-bounded by the minimum between $\Delta(Y_{\omega_1}, P_2(L'_{\omega_1}, d_{q_{\omega_1,1}}, ..., d_{q_{\omega_1,p'}}))/\ell_2$ and $\delta_2 - (\delta_2/2)$, where the first term is due to changing Y_{ω_1} to fit the $d_{q_{\omega_1,j}}$'s and the second term is due to changing at least one of the $X_{q_{\omega_1,j}}$'s so to obtain some other (acceptable) sequence of codewords. As before, we lower-bound the deviation by the product of these terms, yielding half the value of Eq. (40).

2. Otherwise (i.e., some $X_{q_{\omega_1,j}}$ is $(\delta_2/2)$-far from the corresponding $c_{q_{\omega_1,j}}$), one of the input-oracles is $\delta/2$-far from being an E_2-codeword. As in the proof of Claim 5.16.2, in this case, the deviation of $(\overline{X}_{\omega_1}, Y_{\omega_1})$ is at least $\delta_2/2$.

We conclude that, for $\omega_1 \in G$, the deviation of $(\overline{X}_{\omega_1}, Y_{\omega_1})$ is at least half the value of Eq. (40). Using Eq. (41), we lower-bound the expected deviation of $(\overline{X}_{\omega_1}, Y_{\omega_1})$ by $(\delta_2/2) \cdot (\epsilon/4)$. It follows that the inner-verifier V rejects with probability at least $\gamma_2 \cdot \delta_2\epsilon/8$. ■

Combining Claims 5.16.1–5.16.3, while setting $\epsilon' = \epsilon/4$ and $\epsilon'' = \alpha_1\epsilon/4p'$, it follows that the inner-verifier rejects with probability at least

$$\min\left(\alpha_1\gamma_2\epsilon'; \frac{\gamma_1\gamma_2\delta_2\epsilon''}{2}; \frac{\gamma_2\delta_2\epsilon}{8}\right) = \min\left(\frac{\alpha_1\gamma_2}{4}; \frac{\alpha_1\gamma_1\gamma_2\delta_2}{8p'}; \frac{\gamma_2\delta_2}{8}\right) \cdot \epsilon = \frac{\alpha_1\gamma_1\gamma_2\delta_2}{8p'} \cdot \epsilon\,,$$

and the theorem follows. ■

Preserving the almost-uniform queries property. Note that the proof of Theorem 5.16 yields an inner verifier that does not necessarily make almost uniform queries to its oracles. We wish to redeem this state of affairs, both for the sake of elegancy and for future use in Section 5.6. This requires a minor modification of the construction presented in the proof of Theorem 5.16 as well as using an "inner" system (i.e., (E_2, P_2, V_2)) that makes almost uniform queries to its oracles. We also assume that each of the composed verifier makes the same number of queries to each of its input-oracles, which is hereafter referred to as regularity. We stress that our main results (e.g., Theorem 2.3) do not use the following Theorem 5.17.

Theorem 5.17 (Theorem 5.16, revisited): *Let (E_1, P_1, V_1) and (E_2, P_2, V_2) be two LIPSes as in the hypothesis of Theorem 5.16, and $\gamma = \alpha_1\gamma_1\gamma_2\delta_2/8p'$ as there. Furthermore, suppose that V_2 makes α_2-uniform queries to its oracles and that each of the two verifiers makes the same number of queries to each of its input-oracles. Then, for any $\epsilon \in (0,1)$, there exists a $(F, (p,b) \to (p'', b''), \delta_1\delta_2, \gamma/2)$-LIPS that makes $(1-\epsilon)\alpha_1\alpha_2$-uniform queries to its oracles and makes the same number of queries to each of its input-oracles. Furthermore, if the i^{th} given LIPS uses r_i coins, encoding length n_i and proof length ℓ_i then, assuming that $n_i < \ell_i < 2^{r_i}$, the resulting inner verifier V uses $1 + r_1 + r_2 + \log(2^{r_1}\ell_2/\ell_1 n_2) + \log(p'p''/\epsilon)$ coins, encoding length $n_1 \cdot n_2$ and proof length $(p'p''/\epsilon)\cdot(\ell_1 \cdot n_2 + 2^{r_1} \cdot \ell_2)$.*

We comment that the hypothesis that a verifier makes the same number of queries to each of its input-oracles is quite natural. We note that in comparison to Theorem 5.16, the proof-oracle of V is only $p'p''/\epsilon$ times longer. In our applications, we don't care about constant factors in the randomness complexity of the inner-verifier, and thus it is worthwhile to note that the randomness complexity of V is at most $2 \cdot (r_1 + r_2) + \log(p'p''/\epsilon)$, whereas in Theorem 5.16 it was $r_1 + r_2$.

Proof: We use almost the same construction as in the proof of Theorem 5.16. The only modification is that we replicate the two different parts of the resulting proof-oracle an adequate number of times. The purpose of this replication is to guarantee that uniform queries to each of the two parts yield uniform queries to the resulting proof-oracle. Needless to say, we need to check the validity of the replication by an adequate replication test.

For parameters t_1 and t_2 to be determined later, we let the proof Π (constructed by the proving function P) consist of t_1 copies of $\Pi^{(1)} = P^{(1)}(L, x_1, ..., x_p)$ followed by t_2 copies of $\Pi^{(2)} = P^{(1)}(L, x_1, ..., x_p)$. The corresponding verifier V tests these replications (by comparing two random locations) with probability $1/2$, and otherwise acts as before when using *randomly selected copies* of $\Pi^{(1)}$ and $\Pi^{(2)}$. As in the proof of Theorem 5.15, the rejection probability of the resulting verifier is maintained (upto a factor of $1/2$). Thus, the (completeness and) strong soundness holds for any choice of the parameters t_1 and t_2. Turning to analyze the distribution of queries made by V, we consider three types of queries:

1. Queries made by V to one of its p input-oracles. Recall that these queries are determined by the queries that V_1 makes to its own p input-oracles, and the queries made by V_2 to the input-oracles determined by the former queries. Using the uniformity conditions of these two verifiers (and the regularity of V_2's queries), we conclude that V makes $\alpha_1\alpha_2$-uniform queries to its input-oracles. Furthermore, if both V_1 and V_2 are regular (i.e., make the same number of queries to each of the input-oracles) then so is V.
2. Queries made by V to the first part of its proof-oracle. These queries are determined by the queries that V_1 makes to its proof-oracle and the queries made by V_2 to the input-oracles determined by the former queries. Similarly to the previous item, we conclude that V makes $\alpha_1\alpha_2$-uniform queries to the first part of its proof-oracle.
3. Queries made by V to the second part of its proof-oracle. These queries are determined by the uniformly chosen coins $\omega_1 \in \{0,1\}^{r_1}$ and the queries made by V_2 to its proof-oracle. Thus, V makes α_2-uniform queries to the second part of its proof-oracle.

The issue at hand is the proportion between the number of queries that V makes to each of the two parts of its proof-oracle. Suppose that, on the average, V_1 (resp., V_2) makes $p_1 \le p'$ (resp., $p_2 \le p''$) queries to its proof-oracle. Then, on the average, V makes $p_1 \cdot (p'' - p_2)/p'$ queries to the first part of its proof-oracle, and p_2 queries to the second part. Thus, we should replicate the two parts of the proof-oracle so to fit these proportions; that is, we should have

$$\frac{t_1 \cdot \ell_1 n_2}{t_2 \cdot 2^{r_1}\ell_2} \approx \frac{p_1 \cdot (p'' - p_2)}{p' \cdot p_2} \tag{42}$$

Recalling that $\ell_1 n_2 < 2^{r_1}\ell_2$, it suffices to have $t_2 \in [p'p''/\epsilon]$ in order to obtain in Eq. (42) an approximation up to factor $(1 \pm \epsilon)$. Furthermore, $t_1 \cdot \ell_1 n_2 + t_2 \cdot 2^{r_1}\ell_2$ need not be greater than $(p'p''/\epsilon) \cdot (\ell_1 n_2 + 2^{r_1}\ell_2)$. The added randomness (required for selecting random copies) is thus bounded by $\log_2 t_1 + \log_2 t_2 = \log_2(p'p''2^{r_1}\ell_2/\ell_1 n_2 \epsilon)$. (Note that the replication test itself can be implemented using $\log_2((p'p''/\epsilon) \cdot (\ell_1 n_2 + 2^{r_1}\ell_2))$ coins, which in turn is upper-bounded by $r_1 + \log \ell_2 + \log_2(p'p''/\epsilon) < r_1 + r_2 + \log_2(p'p''/\epsilon)$.) The theorem follows. ■

5.4 Linear inner verifiers: Two constructions

Throughout the rest of this section, $F_2 \stackrel{\text{def}}{=} \mathrm{GF}(2)$. We present two LIPSes, one based on the Hadamard encoding function, and the other based on the Reed-Muller encoding function. The first LIPS is a straightforward adaptation of the "inner-most" verifier of Arora *et al.* [2], whereas the second LIPS is obtained by a careful adaptation of the "outer" verifier of [2].

5.4.1 LIPS based on the Hadamard encoding function

We start by presenting a linear inner verifier that corresponds to the inner-most verifier of Arora *et al.* [2]. Things are only simpler in our context, since we only need to prove (and verify) linear conditions (and so we do not need the table of quadratic forms used in the original work). Thus, these F_2-linear conditions, which refer to p elements of F_2^k, may be easily verified by accessing the Hadamard encoding of these p elements.

We comment that one possible implementation of the aforementioned idea amounts to testing each of these p encodings (via a 3-query codeword test), and checking a random F_2-linear condition by self-correction (requiring 2 queries to each input-oracle). Indeed, this implementation requires no proof-oracle, but seems to require at least $2p$ queries (whereas a straightforward implementation uses $5p$ queries). The alternative implementation presented below makes only $p+O(1)$ queries and is closer in spirit to the inner-most verifier of Arora *et al.* [2] (esp., as interpreted in [25, Lem. 2.6]).

Proposition 5.18 *For every pair of integers p and k, there exists a $(F_2,(p,k) \to (p+5,1),\frac{1}{2},\frac{1}{8})$-LIPS. Furthermore, the length of the encoding is 2^k, the length of the proof is 2^{pk}, and the randomness in use equals $3pk+p$. Moreover, the verifier makes uniformly distributed queries to each of its oracles, and makes exactly one query to each of the p input-oracles.*

Proof: The encoding function $E : F_2^k \to F_2^{2^k}$ is just the Hadamard encoding (having relative distance $\frac{1}{2}$). The proving function $P(L,x_1,\ldots,x_p) \in F_2^{2^{pk}}$ is also the Hadamard encoding, this time of the vector $(x_1,\ldots,x_p)$. (Indeed, $P(L,x_1,\ldots,x_p) = E(x_1\cdots x_p)$ is oblivious of L.) The verifier V is given a linear subspace L, in the form of a matrix $M \in F_2^{pk\times pk}$, and access to input-oracles $X_1,\ldots,X_p : F_2^k \to F_2$ and a proof-oracle $\Pi : F_2^{pk} \to F_2$. It operates as follows:

1. Selects uniformly $r = (r_1,...,r_p) \in F_2^{pk}$ and $s = (s_1,...,s_p) \in F_2^{pk}$, and checks that $\sum_{i=1}^p X_i(r_i) = \Pi(r)$ and $\Pi(r)+\Pi(s) = \Pi(r\oplus s)$.
2. Selects a random linear combination v of the constraints of L (i.e., picks a random vector $w \in F_2^{pk}$ and sets $v = w\cdot L$), and verifies that $\Pi(r) = \Pi(r+v)$.
3. Selects uniformly $\sigma_1,...,\sigma_p \in F_2$ and checks that $\sum_{i=1}^p \sigma_i \cdot X_i(r_i) = \Pi(s\oplus r') - \Pi(s)$, where $r' = (r'_1,...,r'_p)$ such that $r'_i = r_i$ if $\sigma_i = 1$ and $r'_i = 0^k$ otherwise.

We note that self-correction (cf. [14]) is performed only on the proof-oracle, whereas each input-oracle is queried at a single (random) point. Furthermore, a couple of queries to the proof-oracle are being re-used (yielding a saving of two queries). We observe that all complexities are as stated in the proposition, and claim that (strong) soundness follows by the standard analysis. Still, since strong soundness was not analyzed explicitly before, we provide a detailed analysis next. Suppose that $(X_1,...,X_p;\Pi)$ has deviation ϵ with respect to (E,P,V). For $\epsilon' = \min(\epsilon/2,1/8)$, we consider the following possible sources of the value of the deviation.

Case 1: The proof-oracle Π is ϵ'-far from the code E. In this case, the linearity test applied to Π in Step 1 guarantees that V rejects with probability at least ϵ' (cf. [7]).

Thus, we may assume for the rest of the analysis that Π is ϵ'-close to $E(x_1\cdots x_p)$, for some $x_1,...,x_p \in F_2^k$. We fix these x_i-s for the rest of the analysis. Viewing $E : F_2^\ell \to F_2^{2^\ell}$ as $E : F_2^\ell \times F_2^\ell \to F_2$, we note that $E(x_1\cdots x_p, r_1\cdots r_p) = \sum_{i=1}^p E(x_i,r_i)$, and so

$$\Pr_r\left[\Pi(r) = \sum_{i=1}^p E(x_i,r_i)\right] \;\ge\; 1-\epsilon' \,. \tag{43}$$

where $r = (r_1,...,r_p)$ is uniformly distributed in F_2^{pk}.

Case 2: $\Pr_r[\Pi(r) \neq \sum_{i=1}^p X_i(r_i)] \ge \epsilon'$. In this case, the other test in Step 1 rejects with probability at least ϵ'.

Thus, we may assume that $\Pr_r[\Pi(r) = \sum_{i=1}^p X_i(r_i)] \ge 1-\epsilon'$. Combining this with Eq. (43), we have

$$\Pr_r\left[\sum_{i=1}^p X_i(r_i) = \sum_{i=1}^p E(x_i,r_i)\right] \;\ge\; 1-2\epsilon' \,. \tag{44}$$

It follows that for every $j \in [p]$ there exists $c = (c_1,...,c_{j-1},c_{j+1},...,c_p) \in F_2^{(p-1)k}$ such that $\Pr_{r_j}[X_j(r_j) = E(x_j,r_j) + b_{j,c}] \ge 1-2\epsilon'$, where $b_{j,c} \stackrel{\rm def}{=} \sum_{i\neq j}(E(x_i,c_i) - X_i(c_i))$.

Case 3: For some j, the input-oracle X_j is not $2\epsilon'$-close to $E(x_i)$. Recalling that X_j is $2\epsilon'$-close to $E(x_i) \oplus b^{2^k}$, where $b \stackrel{\text{def}}{=} b_{j,c} \in F_2$ is as defined above, we show that in this case (i.e., when $b = 1$) the test in Step 3 rejects with constant probability.

We first note that, for any $r_i, s_i \in F_2^k$ and $\sigma_i \in F_2$, it holds that $\sigma_i \cdot E(x_i, r_i) = E(x_i, r_i') = E(x_i, s_i \oplus r_i') - E(x_i, s_i)$, where r_i' is as defined in Step 3. Thus, for random $r, s \in F_2^{pk}$ and $\sigma = (\sigma_1, ..., \sigma_p) \in F_2^p$, and for r' as defined in Step 3,

$$\begin{aligned} & \Pr_{r,s,\sigma}\left[\Pi(s \oplus r') - \Pi(s) = \sum_{i=1}^{p} \sigma_i \cdot E(x_i, r_i)\right] \\ = \ & \Pr_{r,s,\sigma}\left[\Pi(s \oplus r') - \Pi(s) = \sum_{i=1}^{p} E(x_i, s_i \oplus r_i') - \sum_{i=1}^{p} E(x_i, s_i)\right] \\ \geq \ & 1 - 2\epsilon' , \end{aligned}$$

where the inequality is due to Eq. (43). This means that Step 3 essentially checks whether $\sum_{i=1}^p \sigma_i \cdot X_i(r_i)$ equals $\sum_{i=1}^p \sigma_i \cdot E(x_i, r_i)$, or equivalently whether $\sigma_j \cdot (X_j(r_j) - E(x_j, r_j))$ equals $\sum_{i \neq j} \sigma_i \cdot (E(x_i, r_i) - X_i(r_i))$. On the other hand, for each possible choice of $b' \in F_2$, it holds that

$$\begin{aligned} \Pr_{r_j,\sigma_j}\left[\sigma_j \cdot (X_j(r_j) - E(x_j, r_j)) \neq b'\right] & \geq \Pr_{r_j}\left[X_j(r_j) - E(x_j, r_j) = 1\right] \cdot \Pr_{\sigma_j}\left[\sigma_j \neq b'\right] \\ & \geq (1 - 2\epsilon') \cdot \frac{1}{2} , \end{aligned}$$

where the second inequality is due to the case's hypothesis. Using a random choice of $r_1, ..., r_{j-1}, r_{j+1}, ..., r_p \in F_2^k$ and $\sigma_1, ..., \sigma_{j-1}, \sigma_{j+1}, ..., \sigma_p \in F_2$, and setting $b' = \sum_{i \neq j} \sigma_i \cdot (E(x_i, r_i) - X_i(r_i))$, it follows that, in the current case, Step 3 rejects with probability at least $((1 - 2\epsilon')/2) - 2\epsilon' = (1/2) - 3\epsilon' \geq 1/8$. Specifically:

$$\begin{aligned} & \Pr_{r,s,\sigma_1,...,\sigma_p}\left[\sum_{i=1}^{p} \sigma_i \cdot X_i(r_i) \neq \Pi(s \oplus r') - \Pi(s)\right] \\ \geq \ & \Pr_{r_1,...,r_p,\sigma_1 \cdots \sigma_p}\left[\sigma_j \cdot (X_j(r_j) - E(x_j, r_j)) \neq \sum_{i \neq j} \sigma_i \cdot (E(x_i, r_i) - X_i(r_i))\right] \\ & - \Pr_{r,s,\sigma_1,...,\sigma_p}\left[\Pi(s \oplus r') - \Pi(s) \neq \sum_{i=1}^{p} \sigma_i \cdot E(x_i, r_i)\right] \\ \geq \ & \frac{1 - 2\epsilon'}{2} - 2\epsilon' = \frac{1}{2} - 3\epsilon' . \end{aligned}$$

Case 4: $x \stackrel{\text{def}}{=} (x_1, ..., x_p) \notin L$. Recall that Step 2 ensures that encodings of vectors not in L are rejected with probability 1/2. It follows that, in this case

$$\Pr_{r,w}[\Pi(r \oplus wL) - \Pi(r) = E(x, wL) \neq 0] \geq (1 - 2\epsilon')/2 ,$$

because $\Pr_r[\Pi(r \oplus v) - \Pi(r) = E(x, r \oplus v) - E(x, r)] \geq 1 - 2\epsilon'$ for every v, and $\Pr_w[E(x, wL) \neq 0] = 1/2$ for $x \notin L$.

Thus, in each case, V rejects with probability at least $\min(\epsilon', 1/8) \geq \min(\epsilon/2, 1/8) \geq \epsilon/8$. On the other hand, one of these cases must occurs, because otherwise $(X_1, ..., X_p; \Pi)$ has deviation less than ϵ (in contradiction to the hypothesis). ■

5.4.2 LIPS based on the Reed-Muller encoding function

The main result in this subsection is an adaptation of the intermediate inner-verifier of Arora *et al.* [2, Sec. 7]. Recall that the latter uses significantly shorter encoding and proofs (and less randomness) than the simpler Hadamard-based verifier, but verification is based on (a constant number of) *non-boolean* answers.

Theorem 5.19 *There exists a $\gamma > 0$ such that for every pair of integers p and $k > 2^p$, there exists a $(F_2, (p,k) \to (p+4, \mathrm{poly}(\log pk)), \frac{1}{2}, \gamma)$-LIPS. Furthermore, the lengths of the encoding and the proof are $\mathrm{poly}(pk)$, and the randomness in use equals $O(\log pk)$. Moreover, the verifier makes $(1-k^{-1})$-uniformly distributed queries to each of its oracles, and makes exactly one query to each of the p input-oracles.*

Our construction is a modification of the inner-verifier presented by Arora *et al.* [2]; we refer specifically to the proof of Theorem 2.1.9 presented in [2, Sec. 7.5], as interpreted in [25]. We thus start by providing an overview of this proof and discuss the main issues that need to be addressed in adapting it to a proof of Theorem 5.19.

Overview of the proof of [2, Thm. 2.1.9]. We use the formalism of [25] to interpret the main steps in the proof of [2]. As a first step in their proof, Arora *et al.* [2] reduce SAT to a GapPCS problem (see Definition A.1 in Appendix A). Then, using a low-total-degree test, they give a 3-prover 1-round proof system for the latter problem. Finally they observe that the proof system with slight modifications also works for proving properties of inputs presented as oracles that encode strings that when concatenated yield the input. Let us review the completeness and soundness condition of the reduction (used in the first step). Recall that an instance of GapPCS consists of a sequence of algebraic constraints on the values of a function $g : F^m \to F$. Each constraint is dependent on the value of g at only poly-logarithmically many inputs. The goal is to find a low-degree polynomial g that satisfies all (or many) constraints. Actually, the reduction consists of a pair of algorithms A and B, where A reduces instances of SAT to instances of GapPCS, and B transforms pairs (ϕ, τ) to polynomials g such that if τ satisfies the formula ϕ then g satisfies all constraints of $A(\phi)$. The properties of the reduction are as follows:

Completeness: If τ is an assignment satisfying ϕ then $g = B(\phi, \tau)$ is a polynomial of total degree d that satisfies all constraints of $A(\phi)$.

Soundness: If ϕ is not satisfiable, then no polynomial of total degree d satisfies more than an ϵ fraction of the constraints of $A(\phi)$.

Since the soundness condition only focuses on degree d polynomials (and does not refer to arbitrary functions), constructing such a reduction turns out to be easier than constructing a full-fledged PCP. On the other hand, by combining this reduction with a low-degree test it is easy to extend the soundness to all functions.

One would hope to use the above reduction directly to get a LIPS by setting ϕ to be some formula enforcing the linear conditions L. But as noted earlier, several problems come up: First, B is not a linear map, but this is fixed easily. The more serious issue is that the soundness condition permits the existence of low-degree functions that satisfy all constraints but are not even close to $B(\phi, \tau)$ for any τ. Indeed, in standard reductions the only functions in the range of B are polynomials of individual degree d/m in each variable, but this is not something that the low-degree test checks (nor can this be checked directly by a constant number of queries). Thus, to apply the low-degree test and protocol of [2], we *augment the reduction* (from SAT to GapPCS) itself such that it satisfies the following stronger soundness condition (which corresponds to rejection of non-canonical proofs (cf. Section 5.3.1)).

Modified Soundness: If g is a polynomial of total degree d that is not in the range of $B(\phi, \cdot)$ then g does not satisfy more than an ϵ fraction of the constraints of $A(\phi)$.

We note that in our setting τ is provided by the input-oracles[19] (whereas the linear constraints are given as an explicit input), and so the modified soundness refers to this τ (i.e., we require that if the degree d polynomial g differs from $B(\phi,\tau)$ then g does not satisfy more than an ϵ fraction of the constraints of $A(\phi)$). We comment that strong soundness (as defined in Section 5.3.1) will follow by combining this modified soundness with a low-degree test.

To obtain the modified soundness condition, we need to delve further into the reduction of [2] (including the corresponding transformation B). Suppose that their reduction produces a GapPCS instance on m variate polynomials. Then, the corresponding solution $g = B(\phi,\tau)$ satisfies the following additional conditions:

1. The m-variate polynomial $g = B(\phi,\tau)$ has the form $g(i,\vec{x}) = g_i(\vec{x})$, for $i \in [m']$, where the g_i's are polynomials (of varying degrees) in $m-1$ variables. Furthermore, g is a polynomial of degree $m'-1 < d$ in the first variable.

2. There exists a sequence of integers $\langle m_i \rangle_{i\in[m']}$ such that the polynomial g_i only depends on the first $m_i \le m-1$ variables.

3. For every $i \in [m']$ there exists a sequence of integers $\langle d_{i,j} \rangle_{j\in[m-1]}$ such that g_i has a degree bound of $d_{i,j} \le (d-m'+1)/(m-1)$ in its j^{th} variable.

4. The polynomial g must evaluate to zero on some subset of the points (due to padding of the actual input to adequate length).

5. Finally, over some subset of the points, g evaluates to either 0 or 1.

   (Note that this condition is not trivial because we will not be working with F_2 but some extension field K of F_2. In fact over the extension field, these constraints are not even linear. However, these conditions turn out to be F_2-linear.)

In what follows we will, in effect, be augmenting the reduction from SAT to GapPCS so as to include all constraints of the above form. This will force the GapPCS problem to only have satisfying assignments of the form $g = B(\phi,\tau)$ and thus salvage the reduction.

Actuality, we will be considering satisfying assignments that are presented as a concatenation of several pieces that are individually encoded (in corresponding input-oracles), and the constraints of the system we build will be verifying that the "concatenation" of the various pieces is a satisfying assignment. Furthermore, we will only by looking at systems of linear equations and not at general satisfiability.

The actual construction (i.e., proof of Theorem 5.19): Recall that we need to describe the three ingredients in the LIPS: the encoding function $E : F_2^k \to (F_2^{k'})^n$, the proving function $P : F_2^{pk} \to (F_2^{k'})^N$, and the verifier (oracle machine) V. As stated above, we do so by adapting

[19] Indeed, as hinted in previous subsections, the terminology of assignment testers [17] (or PCPPs [10]) is perfectly tailored to express what is going on.

known constructions. (In particular, whenever we refer to a step as being "standard", such a step is performed explicitly in [25].) We start by developing the machinery for the encoding function and the proving function. We do so by transforming the question of satisfaction of a system of linear equations into a sequence of consistency relationships among polynomials and using this sequence to describe the encoding and proving function. For the rest of the discussion, we fix a linear space $L \in \mathcal{L}_{F_2,pk}$ and vectors $x_1,\ldots,x_p$ such that $(x_1,\ldots,x_p) \in L$.

Obtaining a width-3 linear system. Our first step corresponds to the reduction of SAT (or $\mathcal{NP}$) to 3SAT, which is taken for granted in the standard setting. Here we reduce the linear conditions to ones that refer to three variables each (i.e., width-3 linear constraints). As in the standard case, this is done by introducing auxiliary variables.

To convert L into a conjunction of width-3 linear constraints, we introduce a vector, denoted x_{p+1}, of at most $n = (pk)^2$ auxiliary variables, and transform L into a linear space L' of width 3-constraints such that $(x_1,\ldots,x_p) \in L$ if and only if there exists x_{p+1} such that $(x_1,\ldots,x_{p+1}) \in L'$. (Indeed, each linear condition in $t \leq pk$ variables is replaced by $t-2$ width 3-constraints using $t-3$ new auxiliary variables.) Furthermore, for each $(x_1,\ldots,x_p) \in L$ there exists a unique x_{p+1} such that $(x_1,\ldots,x_{p+1}) \in L'$.

For sake of simplicity, we will *assume in the sequel that* $x_1,...,x_{p+1}$ *are all inputs*, although x_{p+1} is actually not an input but rather (only) part of the proof. Thus, it is important to note here that the bits of x_{p+1} are (uniquely determined as) linear combinations of the bits of $x_1,...,x_p$. Indeed, one may think of the current step as a reduction (while noting that this reduction is a linear transformation).

Note that $L' \in \mathcal{L}_{F_2,pk+n}$, because $|x_i| = k$ if $i \leq p$ whereas $|x_{p+1}| = n \gg k$. We will take care of the latter discrepancy in the next step.

Input representation: Low-degree extensions and dealing with padding. The step of taking a low-degree extension is standard, but we need to deal with the padding (of inputs) that it creates (as well as with the padding required to eliminate the discrepancy in the input lengths, created in the previous step). That is, we have to augment the linear system to verify that the padded parts of the input are indeed all-zero.

For $h = \lceil \log n \rceil$ and $m = \lceil \log n / \log\log n \rceil$ (so that $h^m \geq n$), we pick a field $\mathrm{K} = \{\eta_0 = 0, \eta_1 = 1, \ldots, \eta_{|\mathrm{K}|-1}\}$ of size poly(h) that extends F_2 (i.e., $\mathrm{K} = \mathrm{GF}(2^{O(\log h)})$), and a subset $H = \{\eta_0,\ldots,\eta_{h-1}\}$ of K. Next, we let $x_i' = x_i 0^{h^m - |x_i|}$ (i.e., we pad x_i with enough zeroes so that its length is exactly h^m). Now, we let L'' be the F_2-linear constraints indicating that the padded parts of x_i' are zero, and $(x_1',\ldots,x_{p+1}')$ correspond to the padding of $(x_1,\ldots,x_{p+1}) \in L'$.

Finally, as usual, we view x_i' as a function from $H^m \to \{0,1\}$ and let $f_1,\ldots,f_{p+1} : \mathrm{K}^m \to \mathrm{K}$ be m-variate polynomials of degree $h-1$ in each of the m variables that extend the functions described by $x_1',\ldots,x_{p+1}'$.

(We mention that the encoding function E will essentially map x_i to the table of all values of the function f_i.)

Concatenating the p pieces (standard): We let $f : \mathrm{K}^{m+1} \to \mathrm{K}$ be the function given by $f(\eta_i,\cdots) = f_i(\cdots)$ for $i \in \{1,\ldots,p+1\}$ such that f is a polynomial of degree p in its first variable.

Low-degree extension of L'' (standard): Note that L'' imposes a linear constraints on the values of f, where each constraint depends on at most three values of f. Thus, each constraint has the generic form $\alpha_1 f(z_1) + \alpha_2 f(z_2) + \alpha_3 f(z_3)$, for some $\alpha_1, \alpha_2, \alpha_3 \in \{0,1\}$ and $z_1, z_2, z_3 \in H_{p,m} \stackrel{\text{def}}{=} \{\eta_1, \ldots, \eta_{p+1}\} \times H^m$. We view L'' as a function $L'' : \{0,1\}^3 \times H_{p,m}^3 \to \{0,1\}$ such that $L''(\alpha_1, \alpha_2, \alpha_3, z_1, z_2, z_3) = 1$ if the constraint $\alpha_1 f(z_1) + \alpha_2 f(z_2) + \alpha_3 f(z_3)$ is imposed by L'', and extend it to $\hat{L}'' : \mathrm{K}^{3(m+1)+3} \to \mathrm{K}$ that is linear in the first three variables, has degree p in other three variables and degree $h-1$ in all other $3m$ variables. Thus, using $h > p$, the polynomial $\hat{L}''$ has individual degree $h-1$.

We comment that the current step does not rely on L'' being a linear subspace (but rather on it being a system of width-3 equations). The linearity of L'' (or rather of the generic conditions $\alpha_1 f(z_1) + \alpha_2 f(z_2) + \alpha_3 f(z_3)$) will be used in the next step (i.e., in rule $(\mathcal{R}_0)$).

Verifying satisfiability of L'' via sequence of polynomials. This step corresponds to the "sum check" in [2] (which is one of the two procedures in the original inner-verifier, the other being a low-degree test). The current presentation follows [25].

The current step is standard except for rule $(\mathcal{R}_0)$ below, which capitalizes on the linearity of the condition being checked. That is, in the standard presentation g_1 is the product of three values of g_0 (corresponding to an OR of three Boolean values), whereas here it is their sum (corresponding to a width-3 *linear* constraint). In addition, rule $(\mathcal{R}_0)$ includes an extra check that some elements being considered are in $\{0,1\}$.

Let $m' = 4m + 8$. We define a sequence of polynomials $g_0, \ldots, g_{m'+1} : \mathrm{K}^{m'} \to \mathrm{K}$, where g_0 is essentially f, and each g_i is related to g_{i-1} (i.e., g_1 is related to g_0 by an F_2-linear relationship, and g_i is related to g_{i-1} by a K-linear relationship). The motivation behind these polynomials is the following: The function g_1 is defined such that the condition $(x_1, \ldots, x_{p+1}) \in L'$ is equivalent to the condition $g_1(\vec{u}) = 0$ for every $\vec{u} \in H^{m'}$. The polynomials g_i gradually expand the set of points on which the function vanishes from $H^{m'}$ to $\mathrm{K}^{m'}$; specifically, g_{i+1} should vanish on $\mathrm{K}^i \times H^{m'-i}$. Indeed, rule $(\mathcal{R}_i)$ implies that g_i vanishes on $\mathrm{K}^{i-1} \times H^{m'-i+1}$ if and only if g_{i+1} vanishes on $\mathrm{K}^i \times H^{m'-i}$. Thus, finally we have $(x_1, \ldots, x_{p+1}) \in L'$ if and only if $g_{m'+1} \equiv 0$.

For α_i's and u_i's from K and z_i's from K^{m+1}, we require that

$$g_0(z_1, \ldots, z_4, \alpha_1, \ldots, \alpha_4) = f(z_1) \tag{45}$$

Whereas Eq. (45) seems at this stage as merely a notational convention, it actually imposes a condition that will have to be checked. It is more evident that the following conditions impose relations between the various polynomials. As stated above, these relations deviate from the standard ones only in the next rule $(\mathcal{R}_0)$.

$$\begin{aligned}(\mathcal{R}_0): \qquad g_1(z_1, \ldots, z_4, \alpha_1, \ldots, \alpha_4) = \hat{L}''(\alpha_1, \alpha_2, \alpha_3, z_1, z_2, z_3) \cdot \sum_{i=1}^{3} \alpha_i \cdot g_0(z_i\vec{0}) \\ + \alpha_4 \cdot (g_0(z_4\vec{0})^2 - g_0(z_4\vec{0})).\end{aligned}$$

We call the reader attention to the fact that the main term in $(\mathcal{R}_0)$ is linear in the three (typically different) values of g_0, whereas in the standard construction this term is the produce of three such values. In contrast, the secondary term in $(\mathcal{R}_0)$ involves a power of a single value of g_0 (i.e., it includes $g_0(z_4\vec{0})^2$), which is not K-linear but is F_2-linear. The latter fact is based on the fact that the map $\beta \mapsto \beta^2$ is an F_2-linear map over fields of characteristic two. We mention that the secondary term in $(\mathcal{R}_0)$ is meant to verify that for every $z_4 \in H^m$ the value of $g_0(z_4\vec{0})$ is in $\{0,1\}$ (bearing in mind that we will require g_1 to vanish on $H^{m'}$). This verification is "optional" in standard PCPs, in the sense that it is not needed for soundness, but is occasionally thrown in because it serve the intuition (and do not involve much extra work). In contrast, in our case this verification is necessary to enforce the strong soundness condition (i.e., to rule out the possibility that the input-oracles are not valid encodings).

The standard relations are, for $i = 1, \ldots, m'$ (and u_j's in K):

$$(\mathcal{R}_i): \; g_{i+1}(u_1, \ldots, u_{i-1}, u_i, u_{i+1}, \ldots, u_{4m+8}) \; = \; \sum_{j=0}^{h-1} u_i^j \cdot g_i(u_1, \ldots, u_{i-1}, \eta_j, u_{i+1}, \ldots, u_{4m+8}).$$

Merging the different polynomials into a single polynomial g (standard): Let $g : \mathrm{K}^{m'+1} \to \mathrm{K}$ be the function given by $g(\eta_i, z) = g_i(z)$ for $i \in \{0, \ldots, m'+1\}$ such that g is a polynomial of degree $m'+1$ in the first variable (i.e., i). Using $h > m' > p$, we have that g is a polynomial of individual degree at most $2h$, because g_0 has individual degree h, the polynomial g_1 has individual degree $2h$, and each g_{i+1} has individual degree h in its first i variables and individual degree $2h$ in the other variables. Thus, g has total degree at most $d = 2m'h$.

Lines and curves over g (standard): Let $g|_{\text{lines}} : \mathrm{K}^{2(m'+1)} \to \mathrm{K}^{d+1}$ be the function describing the total degree d polynomial $g : \mathrm{K}^{m'+1} \to \mathrm{K}$ restricted to lines; that is, for a line $\ell \in \mathrm{K}^{2(m'+1)}$ the value of $g|_{\text{lines}}(\ell)$ is a univariate degree d polynomial representing the values of g on ℓ. Let $w = 2(m'+1)h$ and $k'' = wd + 1$ and let $g|_{\text{curves}} : \mathcal{C} \to \mathrm{K}^{k''}$ be the restriction of g to some subset $\mathcal{C}$ of degree w curves, where *$\mathcal{C}$ is the set of all the curves that arise in the computation of the verifier described below.* That is, a curve $C \in \mathcal{C}$ is a function $C = (C_1, \ldots, C_m) : \mathrm{K} \to \mathrm{K}^{m'+1}$, where each C_i is a univariate polynomial of degree w, and $g|_{\text{curves}}(C)$ is the univariate degree wd polynomial that represents the value of g on the curve C (i.e., on the set of points $\{C(e) : e \in \mathrm{K}\}$). (Indeed, a line is a curve of degree 1.)

The encoding and proving functions: Finally, we get to define the encoding and proving functions. This step is standard, but we highlight a few non-standard aspects of it.

The encoding function $E(x_i)$ is the table of values of the function $f_i' : \mathrm{K}^m \to \mathrm{K}^{k''}$, where $f_i'(x) = (f_i(x), 0^{k''-1})$; i.e., elements of K are being written as vectors from $\mathrm{K}^{k''}$. Recall that f_i is a low-degree extension of (the padded version of) x_i. Thus, each of the values of f_i (i.e., the value of f_i at each point) is a F_2-linear combination of the values of (the bits in) x_i. This is due to the fact that polynomial extrapolation is a linear operation (on the function's values).

The proving function $P(L'', x_1', \ldots, x_{p+1}') = P_0(L, x_1, \ldots, x_p)$ consists of the triple of functions $(g', g|_{\text{lines}}', g|_{\text{curves}})$, where $g' : \mathrm{K}^{m'+1} \to \mathrm{K}^{k''}$ and $g|_{\text{lines}}' : \mathrm{K}^{2(m'+1)} \to \mathrm{K}^{k''}$ are the functions g and $g|_{\text{lines}}$ with their range being mapped, by padding, into $\mathrm{K}^{k''}$; that is, $g'(x) = (g(x), 0^{k''-1})$ and $g|_{\text{lines}}'(\ell) = (g|_{\text{lines}}(\ell), 0^{k''-(d+1)})$. Note that $P(L'', x_1', \ldots, x_{p+1}')$ refers to the proving function of the reduced instance (obtained in the reduction to width-3 constraints), whereas $P_0(L, x_1, \ldots, x_p)$ refers to the proving function of the original instance. Recall that x_{p+1}' (as well as the other x_i'-s) are F_2-linear combinations of the original x_i-s. We highlight the fact that the values of g are linear in the values of the g_i's, which in turn are linear in g_1, which in turn are F_2-linear in g_0 (and hence in the f_i's). Also, the values of $g|_{\text{lines}}$ and $g|_{\text{curves}}$ are linear in the values of g.

We note that the encoding is a sequence of length $|\mathrm{K}|^m = \mathrm{poly}(h)^m = \mathrm{poly}(n) = \mathrm{poly}(pk)$ over the alphabet $\mathrm{K}^{k''}$, and its relative distance is at least $1-(h^2/|\mathrm{K}|) > 1/2$. The proof length (i.e., $|\mathrm{K}|^{m'+1}$) is polynomial in the encoding length (because $m' = O(m)$).

To motivate the description of the verifier V, we note that the verifier, which essentially has access to the input-oracles $f_1, \ldots, f_{p+1}$ and to the proof-oracle $(g, g|_{\mathrm{lines}}, g|_{\mathrm{curves}})$, needs to verify the following conditions:

1. The function g is a polynomial of degree at most d, the function $g|_{\mathrm{lines}}$ is the restriction of g to lines, and $g|_{\mathrm{curves}}$ is the restriction of g to curves.

2. The degree of g in its first variable is at most $m'+1$.

3. For $i \in \{1, \ldots, m'+1\}$, the function $g_i : \mathrm{K}^{m'} \to \mathrm{K}$ given by $g_i(z) = g(\eta_i, z)$ is computed correctly from g_{i-1} by an application of the rule $(\mathcal{R}_{i-1})$.

4. The function $g_{m'+1}$ is identically zero.

5. The function g_0 is a polynomial of degree 0 in all but its first $m+1$ variables.

6. The function $f : \mathrm{K}^{m+1} \to \mathrm{K}$ given by $f(x) = g_0(x, 0^{m'-(m+1)})$ is a polynomial of degree at most p in its first variable and degree at most $h-1$ in each of the remaining m variables.

7. The function f satisfies $f(\eta_i, x) = f_i(x)$ for every $i \in \{1, \ldots, p+1\}$ and $x \in \mathrm{K}^m$.

Working one's way upwards, one can see that $P_0(L, x_1, \ldots, x_p)$ is the only function that satisfies all the above conditions. In particular, Conditions (5)-(7) force g_0 to uniquely represent the f_i's, Conditions (1)-(4) guarantee that the f_i's are the encoding of inputs that satisfy L, and Conditions (1)-(2) also force the uniqueness of the three parts of the proof-oracle. (We comment that $g|_{\mathrm{lines}}$ and $g|_{\mathrm{curves}}$ are included in the proof-oracle (merely) in order to allow the verification of the aforementioned conditions using very few queries.)

Indeed, it is time to describe the verifier's actions. The aim is to emulate a large number of checks (i.e., random verification of all the above conditions) by using only $p+4$ oracle calls, and still incur only a constant error probability. Specifically, ignoring Condition (1) for a moment, a random test of Condition (2) requires $m'+2$ points in the domain of g, Condition (3) involves $m'+1$ equalities (which refer to $m'+1$ different parts of g and each (but one) of these equalities refers to h values), Condition (5) involves $m'-m$ equalities (one per each suitable variable in g_0) and Condition (7) involves p equalities, each referring to a different function f_i. Following [2], all these different conditions will be checked by retrieving the corresponding (random) g-values from a suitable curve in $g|_{\mathrm{curves}}$, and obtaining the f_i-values from the corresponding oracles. Finally, Condition (1) will be tested by comparing the value of g at a random point to the values of $g|_{\mathrm{lines}}$ and $g|_{\mathrm{curves}}$ on random lines and curves that pass through this point. The comparison to g and $g|_{\mathrm{lines}}$ (which is the well known low-degree test) will also establish the claim that g has low-degree. Details follow.

The verifier first picks one random test (to be emulated) per each of the equalities corresponding to the Conditions (2)–(7) above. Specifically, in order to emulate the testing of Conditions (2), (5) and (6), it picks random axis parallel lines (one per each of the relevant variables) and picks $O(h)$ arbitrary points on these $\mathrm{K}^{m'+1}$-lines with the intention of inspecting the value of g' at these points. (We stress that the verifier does not query g' at these points, but rather only determines these points at this stage.) Similarly, in order to emulate the testing of Conditions (3), (4) and (7), it picks random points from the domain of the corresponding g_i's and f. Having chosen these points, it picks one totally random point in $K^{m'}$. All in all this amounts to determining $w = O(mh)$ points in the domain of g'. The verifier then determines a degree w curve, denoted $C : K \to K^{m'+1}$, that passes through these w points. Finally (in order to check Condition (1)), it picks a random point α on this curve and a random line ℓ through the point α.

Overall, the above random choices can be implemented by picking a constant number of random points in $K^{m'+1}$ and recycling randomness among the various tests (see details in [2] on [25]). Thus, the randomness complexity of the verifier is $O(m' \log |K|) = O(m \log h) = O(\log n) = O(\log pk)$. At this point, we may also bound the size of of the set of curves used by the verifier (i.e., $\mathcal{C}$) by $\text{poly}(pk)$. This bounds the size of $g|_{\text{curves}}$ and thus the length of the entire proof (by $\text{poly}(pk)$).

We finally get to the actual queries of the verifier. It queries the proof-oracle for the values of $g'(\alpha)$, $g|'_{\text{lines}}(\ell)$ and $g|_{\text{curves}}(C)$. It verifies that $g'(\alpha)$ is actually in K and that $g|'_{\text{lines}}(\ell)$ is in K^{d+1} (as opposed to K^k). It then verifies that the three responses agree at α, thus checking Condition (1). Finally, it verifies the values of g' on the test points for tests (2)-(7), as provided (or "claimed") by $g|_{\text{curves}}(C)$, are consistent with the Conditions (2)-(7). In particular, verifying Condition (7) requires a single probe into each of the input-oracles. (Once again the responses to these probes are elements of K^k and the verifier checks that the responses are in K padded with 0's.)

This concludes the description of the verifier. We stress that this description is identical to the one in [2] (as interpreted in [25]), except for two aspects. Firstly, the curve sub-oracle provides the value of g on some additional points in order to support the additional checks in Conditions (2), (5) and (6). Indeed, these conditions were added here in order to enforce the modified soundness condition (which implies strong soundness). Secondly, Conditions (1)-(7) refer to the functions $f_1, ..., f_{p+1}$, g and $g|_{\text{lines}}$, whereas the verifier actually has access to padded versions of these functions (i.e., $f'_1, ..., f'_{p+1}$, g' and $g|'_{\text{lines}}$) and verifiers the correctness of the padding. Indeed, the "0-padding verifications" are only intended to guarantee the modified notion of soundness (and are not needed for the standard notion of soundness). Omitting all these extra tests, would get us back to the interpretation of [2] as provided in [25].

In total, the verifier makes only $(p+1)+3$ queries. Furthermore, the single query made to each of the $p+1$ input-oracles is uniformly distributed and the three queries made to the proof-oracle are each uniformly distributed in the corresponding part of the proof-oracle. (We will address the issue of making almost-uniform queries to the proof-oracle, as a single entity, at the end of the proof.) The answers received by V are from $K^{k''}$ and thus the answer length equals $k'' \log_2 |K|$, which is $\text{poly}(\log(pk))$ as required (using $k'' \log_2 |K| = O(wd \cdot \log h)$ and $d < w = O(mh) < (\log n)^2 = O(\log(pk))^2)$. Finally, note that all checks by the verifier are actually K-linear, except for the satisfaction of rule $(\mathcal{R}_0)$, which is only F_2-linear.

The (strong) soundness of the above verifier is established, as usual, assuming $|K| \geq \text{poly}(h)$. In particular, if the function $g : K^{m'+1} \to K$ (obtained by ignoring the last $k'' - 1$ coordinates of the function g') is not 0.01-close to some polynomial $\hat{g}$ of total degree d then the (point-versus-line) low-degree test will reject with constant probability. Thus, we may assume that g' is 0.01-close to such a $\hat{g}$. Standard soundness follows by the standard argument, but actually the same argument also establishes strong soundness. Intuitively, the low-degree test also guarantees that g' is rejected with probability proportional to its distance from $\hat{g}$. Furthermore, a disagreement of either $g|'_{\text{lines}}$ or $g|_{\text{curves}}$ with $\hat{g}$ is detected with proportional probability by the test that checks Condition (1). Similarly, disagreement between f'_i and $\hat{g}$ is detected with proportional probability by the test that checks Condition (7). Finally, if any of the Conditions (2)-(6) is violated (when applied to $\hat{g}$), then the verifier rejects with constant probability (also when accessing g rather than $\hat{g}$). Following is a more detailed analysis.

We consider an arbitrary $(X_1,...,X_p,X_{p+1};\Pi)$, where $X_i : \mathrm{K}^m \to \mathrm{K}^{k''}$ and $\Pi = (g', g|'_{\mathrm{lines}}, g|_{\mathrm{curves}})$ such that $g' : \mathrm{K}^{m'+1} \to \mathrm{K}^{k''}$, $g|'_{\mathrm{lines}} : \mathrm{K}^{2(m'+1)} \to \mathrm{K}^{k''}$ and $g|_{\mathrm{curves}} : \mathcal{C} \to \mathrm{K}^{k''}$. We denote by ϵ the deviation of $(X_1,...,X_p,X_{p+1};\Pi)$ with respect to (E,P,V). Our aim is to show that V rejects $(X_1,...,X_p,X_{p+1};\Pi)$ with probability $\Omega(\epsilon)$. For $\epsilon' = \epsilon/5 \le 1/5$, we consider the following possible sources of the value of the deviation.

Case 1: Either $\Pr_z[g'(z) \notin K \times 0^{k''-1}] \ge \epsilon'$ or $\Pr_\ell[g|'_{\mathrm{lines}}(\ell) \notin K^d \times 0^{k''-(d+1)}] \ge \epsilon'$. In this case, by virtue of the 0-padding verification, V rejects with probability at least ϵ'.

Thus, we assume in the rest of the analysis that, for some functions $g : \mathrm{K}^{m'+1} \to \mathrm{K}$ and $g|_{\mathrm{lines}} : \mathrm{K}^{2(m'+1)} \to \mathrm{K}^{d+1}$, it holds that $\Pr_z[g'(z) = (g(z), 0^{k''-1})] > 1-\epsilon'$ and $\Pr_\ell[g|'_{\mathrm{lines}}(\ell) = (g|_{\mathrm{lines}}(\ell), 0^{k''-(d+1)})] > 1-\epsilon'$.

Case 2: The function g defined above is ϵ'-far from being a degree d polynomial. In this case, by virtue of the point-versus-line test included in Condition (1), the verifier rejects with probability $\Omega(\epsilon')$ [2, Lem. 7.2.1.4]. (Here we use $|\mathrm{K}| = \mathrm{poly}(d)$. The constant in the Ω is unspecified in [2], but explicit bounds are known now. E.g., [4, Thm. 16] lower bounds the rejection probability by $\frac{2}{3}\epsilon'$.)

Thus, we assume in the rest of the analysis that the function g is ϵ'-close to a degree d polynomial, denoted $\hat{g}$.

Case 3: $\Pr_\ell[\exists e \in \mathrm{K}\ g|_{\mathrm{lines}}(\ell)(e) \neq \hat{g}(\ell(e))] \ge 4\epsilon'$, where $g|_{\mathrm{lines}}(\ell)(e)$ denotes the value of the univariate polynomial $g|_{\mathrm{lines}}(\ell)$ at e. Note that if $g|_{\mathrm{lines}}(\ell)(e) \neq \hat{g}(\ell(e))$ for some $e \in \mathrm{K}$ then the two different (degree d) univariate polynomials $g|_{\mathrm{lines}}(\ell)$ and $\hat{g}(\ell)$ must disagree on at least $|\mathrm{K}| - d > |\mathrm{K}|/2$ of the points on the line ℓ. Thus, in this case, $\Pr_{\ell,e}[g|_{\mathrm{lines}}(\ell)(e) \neq \hat{g}(\ell(e))] \ge 4\epsilon'/2$. Noting that $\ell(e)$ is uniformly distributed in $\mathrm{K}^{m'+1}$, it follows that $\Pr_{\ell,e}[g|_{\mathrm{lines}}(\ell)(e) \neq g(\ell(e))] \ge 2\epsilon' - \epsilon'$, which means that (again by virtue of the point-versus-line test) V will reject with probability at least ϵ'.

Case 4: $\Pr_{C\in\mathcal{C}}[\exists e \in \mathrm{K}\ g|_{\mathrm{curves}}(C)(e) \neq \hat{g}(C(e))] \ge 4\epsilon'$, where $g|_{\mathrm{curves}}(C)(e)$ denotes the value of the univariate polynomial $g|_{\mathrm{curves}}(C)$ at e. Again, using the degree bound (i.e., $wd = O(d^2)$) of these two univariate polynomials (and $|K| > wd/2$), it follows that $\Pr_{C\in\mathcal{C},e}[g|_{\mathrm{curves}}(C)(e) \neq \hat{g}(C(e))] \ge 4\epsilon'/2$, and $\Pr_{C\in\mathcal{C},e}[g|_{\mathrm{curves}}(C)(e) \neq g(C(e))] \ge 2\epsilon' - \epsilon'$, because $C(e)$ is uniformly distributed in $\mathrm{K}^{m'+1}$. Thus, in this case (by virtue of the point-versus-curve test), V will reject with probability at least ϵ'.

Thus, in the rest of the analysis, we assume that

$$\Pr_{C\in\mathcal{C}}[\forall e \in \mathrm{K}\ g|_{\mathrm{curves}}(C)(e) = \hat{g}(C(e))] \ge 1 - 4\epsilon' . \tag{46}$$

In the rest of the analysis, we will heavily rely on the fact that when the verifier needs the values of $\hat{g}$ at certain locations (for a random test of some of Conditions (2)-(7)), it obtains these values by a single random query to $g|_{\mathrm{curves}}$. Furthermore, Eq. (46) guarantees that the answers obtained from $g|_{\mathrm{curves}}$ typically match all relevant values of $\hat{g}$.

Case 5: For some i it holds that $\Pr_{z'\in\mathrm{K}^m}[\hat{g}(\eta_0, \eta_i, z', 0^{m'-(m+1)}) \neq X_i(z')] \ge 5\epsilon'$. In this case, the testing of Conditions (6)-(7), will cause rejection with probability at least $5\epsilon' - 4\epsilon'$, where the latter term is due to (Eq. (46) and) the fact that in testing these conditions we obtain the values of $\hat{g}$ by a single (random) probe to $g|_{\mathrm{curves}}$.

Thus, in the rest of the analysis, we assume that each X_i is $5\epsilon'$-close to $\hat{g}(\eta_0, \eta_i, \cdot, 0^{m'-(m+1)})$. In particular, it follows that X_i is $5\epsilon'$-close to some m-variant polynomial of total degree d, denoted f_i.

Case 6: Some f_i has individual degree greater than $h-1$ in one of its variables. In this case, the verifier rejects with constant probability by virtue of checking Condition (6). (Indeed, here we rely on the negation of Cases 4 and 5.)

Thus, in the rest of the analysis, we assume that each f_i is an m-variant polynomial of individual degree $h-1$, which encodes an h^m-long input, denoted x'_i.

Case 7: Either $(x'_1, ..., x'_{p+1}) \notin L''$ or some x'_i is not in $\{0,1\}^{h^m}$. In this case, the verifier rejects with constant probability by virtue of checking Conditions (3)-(4).

Case 8: The polynomial $\hat{g}$ does not equal $P(L'', x'_1, ..., x'_{p+1})$. Since both polynomials satisfy the same relations, this case may be due only to the individual degrees of $\hat{g}$, which are checked in Conditions (2), (5) and (6). Thus, in this case, the verifier rejects with constant probability.

Thus, in each case, the verifier rejects with probability at least $\min(\Omega(\epsilon'), \Omega(1)) = \Omega(\epsilon') = \Omega(\epsilon)$. On the other hand, one of these cases must occurs, because otherwise $(X_1, ..., X_{p+1}; \Pi)$ has deviation less than $5\epsilon' = \epsilon$ (in contradiction to the hypothesis).

This establishes the theorem, except for the extra condition that requires that the verifier makes almost-uniform to each of its oracles. Recall that the single query made to each of the input-oracles is uniformly distributed, and that each of the three queries made to the proof-oracle is uniformly distributed in the corresponding part of the proof-oracle. The problem is that these three parts do not have the same length. The solution is to modify the construction such that each part of the proof-oracle has approximately the same size. This is done by replications, and as usual a replication test will be used (i.e., with probability 1/2 and otherwise we invoke the verifier V described above while providing it with access to random copies of the corresponding parts). Since we may afford a factor k blow-up in the proof length (and randomness complexity that is logarithmic in the proof length), we can easily make the lengths equal up to a $1 \pm k^{-1}$ factor. Thus, the modified verifier makes $(1-k^{-1})$-uniform queries to each of its oracles, and the theorem follows. ■

5.5 Combining all the constructions

We are now ready to prove the main theorem of this section.

Theorem 5.20 (Theorem 2.3, restated): *For infinitely many k, there exists a locally-testable binary code of constant relative distance mapping k bits to $n \stackrel{\text{def}}{=} \exp(\tilde{O}(\sqrt{\log k})) \cdot k$ bits. Furthermore, the code is linear.*

Proof: The theorem is proved by composing the locally testable code of (Part 1 of) Theorem 2.4 with the two LIPSes constructed in Section 5.4 (i.e., in Proposition 5.18 and Theorem 5.19). Actually, we apply three composition operations, using the LIPS of Theorem 5.19 twice. The sequence of compositions can be ordered arbitrarily. For example, we may first compose the locally-testable code (LTC) with the LIPS of Theorem 5.19, obtaining a new LTC, which is composed again with the latter LIPS, and finally compose the resulting LTC with the LIPS of Proposition 5.18. This, "top-down" order requires to use the augmented composition theorems (which guarantee preservation of the almost-uniformity of the tester's queries). Wishing to use only the "vanilla" composition theorems (which do not preserve the said feature), we use instead a "bottom-up" order of compositions. This will only require that, in each of the compositions, the outer construct (which is one of the abovementioned basic constructs) makes almost-uniform queries. We start by recalling the constructs being used (going from the bottom upwards):

1. The $(F_2, (p_{\rm H}, k_{\rm H}) \to (p_{\rm H}+5, 1), \frac{1}{2}, \frac{1}{8})$-LIPS of Proposition 5.18, for any choice of $p_{\rm H}$ and $k_{\rm H}$. This (Hadamard based) LIPS uses encoding length $2^{k_{\rm H}}$, proof length $2^{p_{\rm H}k_{\rm H}}$, and randomness $3p_{\rm H}k_{\rm H}+p_{\rm H} < 4p_{\rm H}k_{\rm H}$.

2. The $(F_2, (p_{\rm RM}, k_{\rm RM}) \to (p_{\rm RM}+4, \mathrm{poly}(\log p_{\rm RM}k_{\rm RM})), \frac{1}{2}, \Omega(1))$-LIPS of Theorem 5.19, for any choice of $p_{\rm RM}$ and $k_{\rm RM}$. This (Reed-Muller based) LIPS uses encoding and proof length $\mathrm{poly}(p_{\rm RM}k_{\rm RM})$, and randomness $O(\log p_{\rm RM}k_{\rm RM})$. Moreover, the verifier makes $(1-{k_{\rm RM}}^{-1})$-uniformly distributed queries to each of its oracles.

3. The locally testable code $\Sigma^k \to \Sigma^n$ used in Section 3.2 to establish Part 1 of Theorem 2.4, where $n = \exp(\tilde{O}(\sqrt{\log k}))\cdot k$ and $\Sigma = F_2^b$ for $b = \exp(\tilde{O}(\sqrt{\log k}))$.

   Recall that this locally testable code (LTC) is F_2-linear and has constant relative distance, and that the underlying parameters in its construction are $d = m^m$ such that $n = m^{m^2+o(m)}$ and $k = m^{m^2-2m-o(m)}$ (see Eq. (3) and the parameter setting before it). Furthermore, referring to Remark 3.6, the tester makes two 0.8-uniform queries, and uses randomness complexity r such that $2^r \approx |F|^m \cdot (|F|\cdot|R|/|F|^m)^2$, where $|F| = O(d)$ and $|R| = n$. Thus, $2^r < (n/d^{m-2})\cdot n < m^{3m}\cdot n$, which in turn equals $\exp(\tilde{O}(\sqrt{\log k}))\cdot k$, since $n < m^{3m}\cdot k$ and $m < \sqrt{\log k}$.

We start by using Theorem 5.16 to compose the LIPS of Item 2 (as the outer LIPS) with the LIPS of Item 1 (as the inner LIPS), which means setting $k_{\rm H} = \mathrm{poly}(\log p_{\rm RM}k_{\rm RM})$ and $p_{\rm H} = p_{\rm RM}+4$. Setting $p_{\rm RM} = p'$ and $k_{\rm RM} = k'$, the result is a $(F, (p', k') \to (p'+9, 1), \Omega(1), \Omega(1/p'))$-LIPS, denoted S', that uses $O(\log p'k') + O(p'\cdot \mathrm{poly}(\log p'k')) = \mathrm{poly}(p'\cdot \log k')$ random coins, and encoding (and proof) length $\mathrm{poly}(p'k')\cdot \exp(\mathrm{poly}(\log p'k')) = \exp(\mathrm{poly}(\log p'k'))$.

Next, we compose the LIPS of Item 2 (as the outer LIPS) with the LIPS S' (as the inner LIPS), which means setting $k' = \mathrm{poly}(\log p_{\rm RM}k_{\rm RM})$ and $p' = p_{\rm RM}+4$. Setting $p_{\rm RM} = p''$ and $k_{\rm RM} = k''$, the result is a $(F, (p'', k'') \to (p''+13, 1), \Omega(1), \Omega(1/p'')^2)$-LIPS, denoted S'', that uses $O(\log p''k'') + \mathrm{poly}(p''\cdot \log\log k'') = O(p''\cdot \log k'')$ random coins, and encoding (and proof) length $\mathrm{poly}(p''k'')$.

Finally, using Theorem 5.13, we compose the LTC of Item 3 with the LIPS S'' (as the inner LIPS), which means setting $k'' = b = \exp(\tilde{O}(\sqrt{\log k}))$ and $p'' = 2+13$. The result is a binary linear LTC of constant relative distance having length $(\exp(\tilde{O}(\sqrt{\log k}))\cdot k)\cdot \mathrm{poly}(b)$, which equals $\exp(\tilde{O}(\sqrt{\log k}))\cdot k$. The theorem follows. ■

5.6 Additional remarks

In this section we show that certain locally testable linear codes over small alphabets can be modified such that the codeword tester makes only three queries, while essentially preserving the distance and rate of the code. Specifically, we refer to testers that make almost-uniform queries, and start by providing a version of Theorem 5.20 that satisfies this condition.

Proposition 5.21 (Theorem 5.20, revisited): *For infinitely many k, there exists a linear locally-testable binary code of relative constant distance that maps k bits to $n \stackrel{\rm def}{=} \exp(\tilde{O}(\sqrt{\log k}))\cdot k$ bits. Furthermore, for any $\alpha \in (0,1)$, the codeword tester makes α-uniform queries, and uses $\log_2 k + \tilde{O}(\sqrt{\log k})$ random coins.*

Proof: The proposition is proved by following the proof of Theorem 5.20, while using composition theorems (i.e., Theorems 5.15 and 5.17) that preserve the almost-uniformity of the queries made by the verifier (or tester). We note that Theorem 5.17 requires that the inner LIPS make the same number of queries to each of its input-oracles, and we observe that this property holds for each of the two basic LIPSes used in the proof of Theorem 5.20. Furthermore, the Hadamard-based LIPS makes uniformly distributed queries to each of its oracles (cf. Proposition 5.18). We also note that the extra overhead created by Theorems 5.15 and 5.17 (as compared to Theorems 5.13 and 5.16) is insignificant in our case. Details follow.

We first note that the almost-uniformity of the resulting LTC is essentially the product of the almost-uniformity parameters of the basic constructs, which are dominated by the 0.8-uniformity of the LTC of Remark 3.6. However, as stated in Remark 3.6, this bound is arbitrary and we may obtain $(1-\epsilon)$-uniformity for any constant $\epsilon > 0$.

We could have proved the current proposition using any order of composition, but it seems best to verify it using the same order used in the proof of Theorem 5.20. We merely verify that the extra overhead of the composition theorems used here is indeed insignificant. This is obvious for the randomness complexity of the LIPSes obtained by the first two compositions, in which Theorem 5.15 is to be used (instead of Theorem 5.16). The reason is that the randomness in Theorem 5.17 is at most twice than in Theorem 5.16, whereas in the proof of Theorem 5.20 we anyhow stated the randomness complexity of the resulting LIPSes upto a multiplicative constant. Recalling that we compose constructs that make a constant number of queries, this suffices for establishing the current proposition, except for the randomness complexity of the resulting tester.

To analyze the randomness complexity of the resulting tester, we take a closer look at the third composition (i.e., the composition of the LTC with the resulting LIPS, which uses Theorem 5.15). Note that the LTC being composed has randomness complexity $r = \log_2 k + O(\log(n/k))$, and so the composition may incur an extra term of at most $r - \log_2 k$. Furthermore, the randomness complexity of the LIPS verifier is $O(\log(n/k))$, and so the resulting tester also has randomness complexity $\log_2 k + O(\log(n/k)) = \log_2 k + \tilde{O}(\sqrt{\log k})$. ■

Reducing the randomness complexity of testers. As in the case of PCP (cf. [8, Prop. 11.2]), the randomness complexity of codeword testers can be reduced to be logarithmic in the length of the codeword. This complexity reduction is not important in case we start with Proposition 5.21, but we state it for sake of generality.

Proposition 5.22 (reducing the randomness complexity of codeword testers): *Let $\mathcal{C} : \Sigma^k \to \Sigma^n$ be a code.*

1. *Every* (weak) *codeword tester for $\mathcal{C}$ can be modified into one that has randomness complexity $\log_2 n + O(\log(1/\epsilon)) + \log\log|\Sigma|$, and maintains the same rejection probabilities up-to an additive term of ϵ, while preserving the number of queries.*

2. *If $\Sigma = F^\ell$ and $\mathcal{C}$ is F-linear then every* (strong) *codeword tester for $\mathcal{C}$ can be modified into one that has randomness complexity $\log_2 n + \log\log n + \log\log|\Sigma| + O(1)$, while preserving the number of queries.*

   The rejection probability may decrease by a constant factor. Furthermore, if the original tester made α-uniform queries then the resulting one makes $(\alpha - o(1))$-uniform queries.

Note that Part 1 may be used to obtain weak codeword testers of essentially optimal randomness complexity, whereas Part 2 is used to obtain strong codeword testers (but requires $\mathcal{C}$ to be linear).

Proof: The proof of Part 1 is straightforward (and is analogous to the easy case in Step 2 of the proof of Claim 3.5.2). Specifically, using the probabilistic method, there exists a set of $O(\epsilon^{-2}\log_2|\Sigma^n|)$ possible random-tapes for the original tester such that if the tester restricts its choices to this set then its rejection probability on *every* potential sequence is preserved up to an additive term of ϵ. The reason is that, with probability $1-\exp(-\epsilon^2 t)$, a random set of t random-tapes approximates the rejection probability for any *fixed* sequence up to ϵ, while the number of possible sequences is $|\Sigma^n|$.

The proof of Part 2 is analogous to the general case in Step 2 of the proof of Claim 3.5.2. As in the proof of Claim 3.5.2, it suffices to consider the non-codewords that have $\mathcal{C}(0^k)$ as the codeword closest to them. We first observe that, for every fixed $w \in \Sigma^n$ that is δ-far from $\mathcal{C}(0^k)$, with probability $\exp(-\Omega(\delta \cdot t))$, a random set of t random-tapes approximates the rejection probability of w up-to a constant factor. Next, we upper-bound the number of non-codewords that are at distance δn from $\mathcal{C}(0^k)$ by $(|\Sigma|-1)^{\delta n} \cdot \binom{n}{\delta n} < (|\Sigma| \cdot n)^{\delta n}$. Thus, the probability that a random set of t random-tapes approximates the rejection probability of all non-codewords (up-to a constant factor) is at least $1-\exp(-\Omega(\delta \cdot t) + \delta n \log(|\Sigma| \cdot n))$. Thus, setting $t = O(n \log(|\Sigma| \cdot n))$ and using $\delta \leq 1/n$, the main claim of Part 2 follows.

Regarding the almost-uniformity of queries, note that with probability at least $1 - n \cdot \exp(-\epsilon^2 \cdot t/n)$ (over the choices of the set of t random-tapes) the resulting tester makes $((1-\epsilon)\cdot\alpha)$-uniform queries. The proposition follows. ■

Reducing the query complexity of testers. The relevance of low randomness complexity to the project of reducing the query complexity becomes clear in the next proposition. (Note that low randomness complexity of the tester was also used in establishing Theorem 5.20.)

Proposition 5.23 *Let $\Sigma = F$ and suppose that $\mathcal{C} : \Sigma^k \to \Sigma^n$ is a locally-testable F-linear code of constant relative distance. Furthermore, suppose that, for some $\alpha \in (0,1)$, the codeword tester makes α-uniform queries and has randomness complexity $r = r(k,n)$. Then, for $n' = n + O(2^r)$, there exists an F-linear code $\mathcal{C}' : \Sigma^k \to \Sigma^{n'}$ of constant relative distance that is testable with three queries.*

Proposition 5.23 can be extended to the case $\Sigma = F^\ell$, for any constant ℓ, obtaining $n' = n + (q\ell)^2 \cdot 2^r$, where q is the query complexity of the original tester. An analogous result can be stated for non-linear codes (and proven by using the Long Code of [8], but in this case the length blows-up double-exponentially with $q \log|\Sigma|$).

Proof: The current proposition follows by composing the $\mathcal{C}$-tester, which makes $q = O(1)$ queries, with the $(F,(q,1) \to (3,1), 1, q^{-2})$-LIPS presented next, where the composition uses Theorem 5.15.

We note that the LIPS that we are going to construct is fundamentally different from the ones considered so far. It does not reduce the alphabet (but rather keeps it invariant), and it reduces the number of queries (from any q to 3) rather than increasing it. We pay however in the parameter representing the soundness feature (i.e., the proportion between the deviation and the rejection probability). Following is a description of this LIPS:

- The encoding function $E : F \to F$ is the identity function.
- Letting $L \in \mathcal{L}_{F,q}$ be represented by a q-by-q matrix over F, the proving function $P : \mathcal{L}_{F,q} \times F^q \to F^{q(q-1)}$ is as follows: For every $i_1 \in [q]$ and $i_2 \in [q-1]$, the $((i_1-1)(q-1)+i_2)^{\text{th}}$ element of $P(L, x_1, \ldots, x_q)$ equals $\sum_{j=1}^{i_2} c_{i_1,j} x_j$, where $c_{i,j}$ is the $(i,j)^{\text{th}}$ entry in the matrix representing L.

- On input $L \in \mathcal{L}_{F,q}$ and access to input-oracles $X_1, ..., X_q \in F$ (each containing a single symbol) and proof-oracle $Y : [q] \times [q-1] \to F$, the verifier V selects uniformly $i_1, i_2 \in [q]$ and proceed according to the value of i_2.
  1. For $i_2 = 1$, the verifier checks whether $c_{i_1,1} \cdot X_1$ equals $Y(i_1, 1)$.
  2. For $i_2 \in \{2, ..., q-1\}$, the verifier checks whether $Y(i_1, i_2-1) + c_{i_1,i_2} \cdot X_{i_2}$ equals $Y(i_1, i_2)$.
  3. For $i_2 = q$, the verifier checks whether $Y(i_1, q-1) + c_{i_1,q} \cdot X_q = 0$.

  The verifier accepts if and only if the relevant check passes.

Note that if $X \stackrel{\text{def}}{=} (X_1, ..., X_p) \notin L$ then (X, Y) has deviation 1, for every Y. On the other hand, in such a case, there exists an $i_1 \in [q]$ such that $\sum_{j=1}^{q} c_{i_1,j} X_j \neq 0$. For this $i_1 \in [q]$, there exists an $i_2 \in [q]$ such that the above V rejects (because otherwise $0 = Y(i_1, q-1) + c_{i_1,q} \cdot X_q = \cdots = \sum_{j=1}^{q} c_{i_1,j} X_j$). Similarly, if $(X_1, ..., X_p) \in L$ and $Y \neq P(L, X_1, ..., X_p)$, then for some $i_1, i_2 \in [q]$ the verifier rejects. The proposition follows. ■

Short 3-query testable binary codes. Using Propositions 5.21 and 5.23, we show that *our main result regarding locally testable codes* (i.e., Theorem 2.3) *holds also with a tester that make only three queries.*

Corollary 5.24 *For infinitely many k, there exists a linear binary code of relative constant distance that maps k bits to $n \stackrel{\text{def}}{=} \exp(\tilde{O}(\sqrt{\log k})) \cdot k$ bits and has a three-query codeword test.*

Perspective. Corollary 5.24 asserts that *three* queries suffice for a meaningful definition of locally-testable linear codes. This result is analogous to the three-query PCPs available for NP-sets.[20] In both cases, the constant error probability remains unspecified, and a second level project aimed at minimizing the error of three-query test arises. Another worthy project refers to the trade-off between the number of queries and the error probability, which in the context of PCP is captured by the notion of amortized query complexity. The definition of an analogous notion for locally-testable codes is less straightforward because one needs to specify which strings (i.e., at what distance from the code) should be rejected with the stated error probability. One natural choice is to consider the rejection probability of strings that are at distance $d/2$ from the code, where d is the distance of the code itself. Alternatively, one may consider the proportion between the relative distance to the code and the rejection probability.

6 Subsequent Work and Open Problems

We have presented locally testable codes and PCP schemes of almost-linear length, where $\ell : \mathbb{N} \to \mathbb{N}$ is called almost-linear if $\ell(n) = n^{1+o(1)}$. For PCP, this improved over a previous result where for each $\epsilon > 0$ a scheme of length $n^{1+\epsilon}$ was presented (with query complexity $O(1/\epsilon)$). Recall that our schemes have length $\ell(n) = \exp(\tilde{O}(\sqrt{\log n})) \cdot n$. In earlier versions of this work (e.g., [24]), we wondered whether length $\ell(n) = \text{poly}(\log n) \cdot n$ (or even linear length) can be achieved. Similarly, the number of queries in our proof system is really small, say 19, while simultaneously achieving nearly linear-sized proofs. Further reduction of this query complexity is very much feasible and it is unclear what the final limit may be. Is it possible to achieve nearly-linear (or even linear?) proofs with 3 query bits and soundness nearly 1/2?

[20] In both cases, testability by two queries is weak: see [8, Prop. 10.3] for PCPs and [11] for locally-testable codes.

Turning to more technical issues, we note that our constructions of codes and PCPs are actually randomized. In case of codes, this means that we prove the existence of certain codes (by using the probabilistic method), but we do not provide fully-explicit codes. In case of PCPs, we obtained PCPs for a problem to which SAT can be randomly reduced (rather for SAT itself). In both cases, the probabilistic method is used to determine a sample of random-tapes for a relevant test, and the probabilistic analysis shows that almost all choices of the subspace will do. A natural (de-randomization) goal, stated in our preliminary report [24], has been to provide an explicit construction of a good subspace. For example, in case of the low-degree test (which underlies our codeword tester), the goal was to provide an explicit set of $\tilde{O}(|F|^m)$ lines that can be used for this test (as the set R in the construction of Section 3.2).

In our preliminary report [24] we also suggested the following seemingly easier goal of de-randomizing the linearity test of Blum, Luby and Rubinfeld [14]. Recall that in order to test whether $f : G \to H$ is linear, one uniformly selects $(x, y) \in G \times G$ and accepts if and only if $f(x) + f(y) = f(x + y)$. Now, by the probabilistic method, there exists a set $R \subset G \times G$ of size $O(|G| \log |H|)$ such that the test works well when (x, y) is uniformly selected in R (rather than in $G \times G$).[21] The challenge suggested in [24] was to present an explicit construction of such a set R.

The latter challenge as well as the more general goal of de-randomizing all our results were recently resolved by Ben-Sasson, Sudan, Vadhan and Wigderson [13]. Specifically, they showed that for low-degree testing one may use a small set of lines that consists of all lines going in a small set of directions. They also showed that this result suffices for the derandomization of our PCP result.

Another natural question that arises in this work refers to obtaining locally-testable codes for coding $k' < k$ information symbols out of codes that apply to k information symbols. The straightforward idea of converting k'-symbol messages into k-symbol messages (via padding) and encoding the latter by the original code, preserves many properties of the code but does not necessarily preserve local-testability.[22]

Finally, we mention a few recent works that address the main question raised by our work and mentioned above (i.e., whether PCPs and codes of length $\mathrm{poly}(\log n) \cdot n$ are achievable, where n is the length of the relevant input). The first quantitative improvement over our work was obatined by Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan [10] that, for every constant $\epsilon > 0$, presented PCPs and (weak) locally testable codes of length $\exp(\log^\epsilon n) \cdot n$. Building on the work of Ben-Sasson and Sudan [12], Dinur [16] has resolved the aforementioned problem by presenting PCPs and (weak) locally testable codes of length $\mathrm{poly}(\log n) \cdot n$. Specifically, Dinur applied her "PCP amplification" technique, which is the main contribution of her work, to the PCP presented by [12]. We note, however, that these improved codes (resp., PCP constructions) do *not* achieve strong codeword testability (resp., strong soundness). Indeed, obtaining such strong constructs of length that improves on $\exp(\tilde{O}(\sqrt{\log n}) \cdot n$ is an open problem.

Acknowledgments

We are grateful to Salil Vadhan for suggesting some modifications to the construction and analysis in Section 3.2 , yielding stronger results with simpler proofs. We also wish to thank the anonymous

[21]For every $f : G \to H$, with probability $1 - \exp(-|R|)$ a random set R will be good for testing whether f is linear, and the claim follows using the union bound for all $|H|^{|G|}$ possible functions $f : G \to H$.

[22]Indeed, this difficulty (as well as other difficulties regarding the gap between PCPs and codes) disappears if one allows probabilistic coding. That is, define a code $\mathcal{C} : \Sigma^k \to \Sigma^n$ as a randomized algorithm (rather than a mapping), and state all code properties with respect to randomized codewords $\mathcal{C}(a)$'s.

referees for their helpful comments.

References

[1] N. Alon, T. Kaufman, M. Krivelevich, S. Litsyn, and D. Ron. Testing Low-Degree Polynomials over GF(2). In *Proc. RANDOM 2003*, Lecture Notes in Computer Science, vol. 2754, pages 188-199, Springer, 2003.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *JACM*, Vol. 45, pages 501–555, 1998. Preliminary version in *33rd FOCS*, 1992.

[3] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. *JACM*, Vol. 45, pages 70–122, 1998. Preliminary version in *33rd FOCS*, 1992.

[4] S. Arora and M. Sudan. Improved low degree testing and its applications. In *29th STOC*, pages 485–495, 1997.

[5] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking Computations in Polylogarithmic Time. In *23rd STOC*, pages 21–31, 1991.

[6] L. Babai, L. Fortnow, and C. Lund. Non-Deterministic Exponential Time has Two-Prover Interactive Protocols. *Computational Complexity*, 1: 3-40 (1991).

[7] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan. Linearity testing over characteristic two. *IEEE Transactions on Information Theory*, 42(6):1781-1795, November 1996.

[8] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs, and nonapproximability – towards tight results. *SIAM Journal of Computing 27*, 3 (June 1998), 804–915. Preliminary Version in *36th FOCS*, 1995.

[9] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximation. In *26th STOC*, 1994.

[10] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of proximity, Shorter PCPs and Applications to Coding. In *36th STOC*, 2004, pages 1–10. Also available as an ECCC Technical Report, TR04-021, March 2004.

[11] E. Ben-Sasson, O. Goldreich and M. Sudan. Bounds on 2-Query Codeword Testing. In the proceedings of *RANDOM'03*, Springer LNCS, Vol. 2764, pages 216–227, 2003.

[12] E. Ben-Sasson and M. Sudan. Simple PCPs with Poly-log Rate and Query Complexity. *ECCC*, TR04-060, 2004.

[13] E. Ben-Sasson, M. Sudan, S. Vadhan and A. Wigderson. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *35th STOC*, pages 612–621, 2003.

[14] M. Blum, M. Luby and R. Rubinfeld. Self-Testing/Correcting with Applications to Numerical Problems. *JCSS*, Vol. 47, No. 3, pages 549–595, 1993.

[15] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems.* SIAM Press, Philadeplhia, PA, USA, March 2001.

[16] I. Dinur. The PCP Theorem by Gap Amplification. *ECCC*, TR05-046, 2005.

[17] I. Dinur and O. Reingold. Assignment-testers: Towards a combinatorial proof of the PCP-Theorem. In *45th FOCS*, 2004, pages 155–164.

[18] S. Even, A.L. Selman, and Y. Yacobi. The Complexity of Promise Problems with Applications to Public-Key Cryptography. *Inform. and Control*, Vol. 61, pages 159–173, 1984.

[19] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating Clique is almost NP-complete. *JACM*, Vol. 43, pages 268–292, 1996. Preliminary version in *32nd FOCS*, 1991.

[20] G.D. Forney. *Concatenated Codes.* MIT Press, Cambridge, MA 1966.

[21] K. Friedl and M. Sudan. Some Improvements to Low-Degree Tests. In the *3rd Israel Symp. on Theory and Computing Systems (ISTCS)*, 1995.

[22] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *JACM*, pages 653–750, July 1998.

[23] O. Goldreich, H. Karloff, L.J. Schulman and L. Trevisan. Lower Bounds for Linear Locally Decodable Codes and Private Information Retrieval. In the *Proc. of the 17th IEEE Conference on Computational Complexity*, 2002.

[24] O. Goldreich and M. Sudan. Locally Testable Codes and PCPs of Almost-Linear Length. ECCC Report TR02-050, 2002.

[25] P. Harsha and M. Sudan. Small PCPs with Low Query Complexity. *Computational Complexity*, 9(3-4):157-201, 2000.

[26] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, Vol. 182, pages 105–142, 1999. Preliminary versions in *28th STOC* (1996) and *37th FOCS* (1996).

[27] J. Katz and L. Trevisan. On The Efficiency Of Local Decoding Procedures For Error-Correcting Codes. In the *32nd STOC*, 2000.

[28] M. Kiwi. Testing and Weight Distribution of Dual Codes. ECCC Technical Report Number TR97-010, 1997 .

[29] R. Motwani and P. Raghavan. *Randomized algorithms.* Cambridge University Press, 1995.

[30] A. Polishchuk and D.A. Spielman. Nearly-linear size holographic proofs. In *26th STOC*, pages 194–203, 1994.

[31] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *29th STOC*, 1997.

[32] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, Vol. 25 (2), pages 252–271, 1996.

Appendix A: The 3-prover system of [25], revisited

The 3-prover system of Harsha and Sudan [25] handles an NP-complete (promise) problem called GapPCS. This promise problem is revisited in Section A.1, where we also present a restricted version of it called rGapPCS. In Section A.2 we adapt the results of [25] to the variant introduced in Section A.1, while in Section A.3 we describe the high level operation of the 3-prover system of [25]. The latter section is aimed to support the claims made when abstracting this proof system in Section 4.2.1.

A.1 The Gap Polynomial-Constraint-Satisfaction Problem

We start by recalling the "Gap Polynomial Constraint Satisfaction Problem" and introducing a restricted version of this problem.

Standard CSPs. Constraint satisfaction problems (CSPs) are a natural class of optimization problems where an instance consists of t Boolean constraints $C_1, \ldots, C_t$ placed on n variables, each taking on values from some finite domain, say $\{0, \ldots, D-1\}$. Each constraint is restricted in that it may only depend on a small number, w, of variables. The goal of the optimization problem is to find an assignment to the n variables that maximizes the number of constraints that are satisfied. The complexity of the optimization task depends on the nature of constraints that may be applied, and thus each class of constraints gives rise to a different optimization problem (cf. [15]). CSPs form a rich subdomain of optimization problems that include Max-3SAT, Max-2SAT, Max-Cut, Max-3-Colorability etc., and lend themselves as targets for reductions from PCPs (i.e., PCPs with certain parameters were often reduced to CSP problems of certain types and parameters).

Algebraic CSPs. Following Harsha and Sudan [25], we consider algebraic variants of CSPs. These problems differ from the standard CSPs in certain syntactic ways. The domain of the values that a variable can assume is associated with a finite field F; the index set of the variables is associated with F^m for some integer m, rather than being the set $[n]$; and thus an assignment to the variables may be viewed naturally as a function $f : F^m \to F$. Thus, the optimization problem(s) ask for functions that satisfy as many constraints as possible. In this setting, constraints are also naturally interpreted as algebraic functions, say given by an algebraic circuit.

The interesting (non-syntactic) aspect of these problems is when we optimize over a restricted class of functions, rather than over the space of all functions. Specifically, for a given degree bound d, we consider the maximum number of constraints satisfied by degree d polynomial $f : F^m \to F$. Under this restriction on the space of solutions, it is easier to establish NP-hardness of the task of distinguishing instances where all constraints are satisfiable from instances where only a tiny fraction of the constraints are satisfiable. This motivates the "Gap Polynomial CSP", first defined by Harsha and Sudan [25].

Definition A.1 (Gap Polynomial Constraint Satisfaction (GapPCS)): *For integers k, m, s and a finite field F, an (m, k)*-ary algebraic constraint of complexity *s* over *F is a $(k+1)$-tuple $C = (A; v_1, \ldots, v_k)$, where $A : (F^m)^k \to F$ is an algebraic circuit of size s, and $v_1, \ldots, v_k \in F^m$ are variable names. For $\epsilon : \mathsf{Z}^+ \to \mathsf{R}^+$ and $m, b, q : \mathsf{Z}^+ \to \mathsf{Z}^+$, the promise problem* $\mathrm{GapPCS}_{\epsilon,m,b,q}$ *has as instances tuples $(1^n, d; C_1, \ldots, C_t)$, where $d, k \le b(n)$ are integers and $C_j = (A_j; v_{j,1}, \ldots, v_{j,k})$ is an $(m(n), k)$-ary algebraic constraint of complexity $b(n)$ over $F = \mathrm{GF}(q(n))$. The promise problem consists of the following sets of* YES *and* NO *instances.*

YES-instances: *The instance* $(1^n, d; C_1, \ldots, C_t)$ *is a* YES-*instance if there exists a polynomial* $p : F^m \to F$ *of total degree at most* d *such that, for every* j, *the constraint* C_j *is satisfied by* p; *that is,* $A_j(p(v_{j,1}), \ldots, p(v_{j,k})) = 0$, *for every* $j \in [t]$.

NO-instances: *The instance* $(1^n, d; C_1, \ldots, C_t)$ *is a* NO-*instance if, for every polynomial* p *of total degree at most* d, *at most* $\epsilon(n) \cdot t$ *constraints are satisfied* (i.e., evaluate to 0).

Note that all the varying parameters are expressed in terms of (the explicitly given) parameter n, whereas the instance length is essentially $n + \log b(n) + t \cdot b(n) \cdot (m(n) + 1) \cdot \log q(n)$.

We stress that these gap problems are shown to be NP-hard (in [25]) via a reduction that does not start from a PCP; instead the ideas underlying the PCP construction of [5, 19] are (directly) used in the reduction. Furthermore, these (algebraic) CSPs are used as the problem for which PCPs are designed (rather than as the target of reduction from certain PCPs). We comment that, so far (including our work), this approach was used to design PCPs with certain parameters per se (and not to establish "hardness of approximation" results).

Restricting the algebraic CSPs. In order to facilitate the design of PCPs, we consider a restricted version of the algebraic CSPs considered in [25]. Specifically, we consider a restriction on the class of instances, where each constraint, in addition to being restricted to apply only to k variables, is restricted to apply only to variables that lie on some "2-dimensional variety" (i.e., the names/indices of the variables that appear in a constraint must lie on such a variety). We define this notion first.

A d-dimensional variety of degree r is represented by a function $Q = (Q_1, \ldots, Q_m) : F^d \to F^m$ where each Q_i is a d-variate polynomial of degree r, and consists of the set of points $\mathcal{V}_Q \stackrel{\text{def}}{=} \{Q(x) : x \in F^d\}$. (Note that this formulation is more restrictive than the standard definitions of varieties.) A set of points is said to lie on the variety $\mathcal{V}_Q$ if this set is contained in V_Q.

In the following definition, in addition to requiring that the variables of each constraint lie on a 2-dimensional variety (of degree r), we include this variety in the description of the constraint. (This was not required in [25], because they used a canonical higher-dimensional variety, which was constructed generically from the aforementioned points and did not rely on the special structure of these points.)

Definition A.2 (restricted Gap Polynomial Constraint Satisfaction (rGapPCS)): *For integers* k, m, s, r *and a finite field* F, *a* $(2, r)$-restricted (m, k)-ary algebraic constraint of complexity s over F *is a* $(k+2)$*-tuple* $C = (A; v_1, \ldots, v_k; Q)$, *where* $A : (F^m)^k \to F$ *is an algebraic circuit of size* s, *and* $v_1, \ldots, v_k \in F^m$ *are variable names that lie on the 2-dimensional variety of degree* r *represented by* Q. *For* $\epsilon : \mathsf{Z}^+ \to \mathsf{R}^+$ *and* $r, m, b, q : \mathsf{Z}^+ \to \mathsf{Z}^+$, *the promise problem* $\mathrm{rGapPCS}_{\epsilon,r,m,b,q}$ *has as instances tuples* $(1^n, d; C_1, \ldots, C_t)$, *where* $d, k \leq b(n)$ *are integers and* $C_j = (A_j; v_{j,1}, \ldots, v_{j,k}; Q_j)$ *is a* $(2, r(n))$*-restricted* $(m(n), k)$*-ary algebraic constraint of complexity* $b(n)$ *over* $F = \mathrm{GF}(q(n))$. *The partition of these instances to* YES *and* NO *instances is as in Definition A.1.*

Again, all the varying parameters are expressed in terms of (the explicitly given) parameter n, whereas the instance length is

$$N \stackrel{\text{def}}{=} |(1^n, d; C_1, \ldots, C_t)| \leq n + \log b(n) + t \cdot (b(n) + r(n)^2) \cdot (m(n) + 1) \cdot \log q(n) \qquad (47)$$

(when ignoring the effect on length involved in encoding sequences as a single string).

A.2 The complexity of rGapPCS

The following lemma is a slight variant of Lemma 3.16 in [25]. Specifically, while [25] use the generic fact that any k points lie on a d-dimensional variety of degree $d \cdot k^{1/d}$, we note that the specific $O(m(n)b(n))$ points chosen for each constraint (in the reduction) happen to lie on a 2-dimensional variety of degree $O(m(n))$. This is because each constraint refers to $O(m(n)b(n))$ points such that each point lies on one out of $O(m(n))$ lines. Furthermore, we can construct a representation of this variety, given that we have both the points and the lines on which they lie. The following lemma simply lists conditions on the parameters that allows for *restricted* GapPCS to be NP-hard.

Lemma A.3 (slight variant of [25, Lem. 3.16]): *There exists constants c_1, c_2 and a polynomial p_1 such that for any collection of functions $\varepsilon : \mathsf{Z}^+ \to \mathsf{R}^+$ and $m, r, b, q, \ell : \mathsf{Z}^+ \to \mathsf{Z}^+$ that satisfy $b(n) \geq \log n$, $(b(n)/m(n))^{m(n)} \geq n$, $r(n) \geq c_1 m(n)$, $q(n) \geq (b(n)/\varepsilon(n)) \cdot p_1(m(n))$, and $\ell(n) \geq q(n)^{m(n)+c_2}$, it holds that SAT reduces to $\mathrm{rGapPCS}_{\varepsilon,r,m,b,q}$ under a ℓ-length preserving reduction.*

The proof of Lemma A.3 is immediate from the description in [25] and the aforementioned observation about the existence and constructibility of an adequate (2-dimensional) variety (of degree $r(n)$). On the other hand, when applying the MIP system of [25, Section 3.6] to *restricted* GapPCS instances, we get:

Lemma A.4 (implicit in [25, Sec. 3.6]): *There exists a polynomial p_2 such that if $\varepsilon : \mathsf{Z}^+ \to \mathsf{R}^+$ and $r, m, b, q : \mathsf{Z}^+ \to \mathsf{Z}^+$ satisfy $q(n) \geq p_2(r(n)) \cdot (b(n)/\varepsilon(n))$ then the promise problem $\mathrm{rGapPCS}_{\varepsilon,r,m,b,q}$ has a 3-prover MIP proof with perfect completeness, soundness $O(\varepsilon(n))$, answer length $\mathrm{poly}(b(n) + r(n)) \cdot \log q(n)$, and randomness $O(\log N) + O(m(n) \log q(n))$, where N denotes the size of the GapPCS instance and n denotes the first parameter in the instance. Furthermore, the size of the first prover is $q(n)^{m(n)}$, and its answer length is $\log q(n)$.*

When wishing to derive 3-prover MIPs for SAT by using Lemma A.4, we may use the reduction provided by Lemma A.3 for an appropriate choice of the parameters $\varepsilon, m, b, q, \ell$. Indeed, combining Lemmas A.3 and A.4, we state the following result regarding 3-prover MIPs for SAT, where we restrict attention to the case of constant $\epsilon > 0$ (and set most of the free parameters appearing in the two lemmas).

Theorem A.5 *For every constant $\epsilon > 0$ and $m : \mathsf{Z}^+ \to \mathsf{Z}^+$, let $\ell(n) = m(n)^{O(m(n))} \cdot n^{1+O(1/m(n))}$. Then SAT has a 3-prover proof system with perfect completeness, soundness ϵ, randomness $O(\log n)$, and answer length $m(n)^{O(1)} \cdot n^{O(1/m(n))}$, in which the first prover has size $O(\ell(n))$, where n denotes the length of the input.*

Proof: Assume without loss of generality that $m(n) \leq (\log n)/(\log \log n)$. (For larger $m(\cdot)$, the requirements on both the function $\ell(n)$ and the answer length become weaker.) Let p be a polynomial such that $p(t) \geq \max(p_1(t), p_2(c_1 t))$ for every $t \geq 1$, where c_1 is the constant in Lemma A.3 and p_1 and p_2 are the polynomials in Lemmas A.3 and A.4, respectively. We use the following setting of the functions b, r and q.

$$
\begin{aligned}
b(n) &= m(n) \cdot n^{1/m(n)} \\
r(n) &= c_1 \cdot m(n) \\
\varepsilon(n) &= \epsilon/O(1) \\
q(n) &= (b(n)/\varepsilon(n)) \cdot p(m(n))
\end{aligned}
$$

The reader can easily verify that this setting satisfies all relevant conditions in Lemmas A.3 and A.4. To verify the remaining condition, which refers to ℓ, note that

$$\begin{aligned} q(n)^{m(n)+c_2} &= ((m(n) \cdot n^{1/m(n)}/\varepsilon(n)) \cdot p(m(n)))^{m(n)+c_2} \\ &\leq (m(n)/\varepsilon(n))^{O(m(n))} \cdot n^{1+(c_2/m(n))} \end{aligned}$$

Using $\varepsilon(n)^{-1} \leq m(n)^{O(1)}$, we have $q(n)^{m(n)+c_2} < m(n)^{O(m(n))} \cdot n^{1+O(1/m(n))}$ and $\ell(n) \geq q(n)^{m(n)+c_2}$ follows for a suitable constant c in the setting $\ell(n) = m(n)^{c \cdot m(n)} \cdot n^{1+(c/m(n))}$. We note that $\log q(n) = O(\log m(n)) + (1/m(n)) \log n$, and recall that $m(n) \log m(n) < \log n$. Now, invoking Lemmas A.3 and A.4 (with the setting of parameters as above), we obtain a 3-prover proof system for SAT with perfect completeness, soundness ϵ, and the following parameters

- Answer length $\mathrm{poly}(b(n) + r(n)) \cdot \log q(n) = m(n)^{O(1)} \cdot n^{1/O(m(n))}$.
- Randomness $O(\log \ell(n)) + O(m(n) \log q(n)) = O(\log n)$.
- The size of the first prover oracle is $q(n)^{m(n)} < \ell(n)$.

The theorem follows. ■

A.3 The proof system of Theorem A.5

In this section we provide a high level description of the operation of the 3-prover system that underlies the proof of Theorem A.5, which in fact is the system underlying the proof of Lemma A.4. (Needless to say, a full description of this system is given in the original work of Harsha and Sudan [25].)

Recall that the problem (instance) consists of parameters n, d and a sequence of constraints $C_1, ..., C_t$. (See Definition A.2.) The field $F = \mathrm{GF}(q(n))$ is determined by n (and so are the values $m = m(n)$ and $r = r(n)$). In the 3-prover one-round system underlying the proof of Lemma A.4, the verifier expects the three provers P, P_1, P_2 to answer its queries as follows:

- P should answer according to an assignment function f that satisfies the conditions of Definition A.2. In particular, f is supposed to be a degree d polynomial in m variables over F.
- P_1 should provide the value of f when restricted to any plane in F^m, where a plane Π is defined by three points in F^m (i.e., $\Pi = \Pi_{a,b,c} = \{i \cdot a + j \cdot b + c : i, j \in F\}$, for $a, b, c \in F^m$). That is, P_1 should answer the query $\Pi = \Pi_{a,b,c}$ with the bivariate polynomial $f_\Pi = f(\Pi)$ over F, where $f_\Pi(x, y) = f(x \cdot a + y \cdot b + c)$.
- P_2 should provide the value of f when restricted to any curve (of appropriate flexibility) in F^m. Specifically, the curves are 3-dimensional varieties of degree r, given by m trivariate polynomials of degree r (over F).

The verifier operates as follows. It picks a random constraint $C_j = (A_j; v_{j,1}, ..., v_{j,k}; Q_j)$ and a random point v_0, picks a random plane Π that passes through v_0, and a random curve $\mathcal{C}$ (i.e., a 3-dimensional variety of degree r) that extends the variety represented by Q_j and passes through the point v_0. (Specifically, this curve may be the one given by $\mathcal{C}(s, t_1, t_2) = s \cdot v_0 + (1-s) \cdot Q_j(t_1, t_2)$.) It sends v_0 to P, Π to P_1, and $\mathcal{C}$ to P_2, receiving the answers $a \stackrel{\mathrm{def}}{=} P(x_0)$, $g = P_1(\Pi)$, and $h = P_2(\mathcal{C})$. The verifier accepts if and only if the following two conditions hold:

1. The function g is consistent with P's answer at v_0; that is, $g(t',t'') = a$, where $\Pi(t',t'') = v_0$.

2. The function h is consistent with P's answer at v_0 and the values of f (as provided by h) on $v_{j,1}, ..., v_{j,k}$ satisfy A_j. That is:

   (a) $h(\alpha_0) = a$, where $\mathcal{C}(\alpha_0) = v_0$.

   (b) $A_j(h(\alpha_1), ..., h(\alpha_k)) = 0$, where $\mathcal{C}(\alpha_i) = v_{j,i}$ for $i = 1, ..., k$.

Note that this verifier has logarithmic randomness complexity (i.e., it tosses $(\log t) + O(m \log q(n))$ coins, whereas its input length exceeds $t \cdot m \log q(n)$), and that each of its queries is uniformly distributed in the corresponding domain. Thus, this verifier satisfies the Sampleability and Uniformity Properties defined in Section 4.2.1. Before turning to the Decomposition Property, we note that the verifier has perfect completeness (i.e., if a good solution f exists then setting the prover strategies as suggested above makes the verifier accept with probability 1).

Soundness and Decomposition Property: Suppose that $f = P$ does not satisfy the rGapPCS instance. Consider the set of all m-variate polynomials of degree d that agree with f on at least $\epsilon/2$ of the domain. Denoting these polynomials by $p_1, ..., p_L$, we denote by S_i the set of points where f agrees with p_i (i.e., $S_i = \{x \in F^m : f(x) = p_i(x)\}$). Let $Q' = Q'_P = F^m \setminus \cup_i S_i$. We consider the following two cases (concerning whether or not the random point v_0 is in Q'):

$v_0 \in Q'$: This case is analyzed as Event 1 in the proof of [25, Claim 3.30], where it is shown that for every P_1

$$\Pr_{v_0,\Pi}[v_0 \in Q' \text{ and } (P_1(\Pi))(t',t'') = f(v_0)] < \epsilon/2$$

where $\Pi(t',t'') = v_0$.

$v_0 \in \cup_i S_i$: This case is analyzed as Events 2 and 3 in the proof of [25, Claim 3.30], where it is shown that for every P_2

$$\Pr_{v_0,j,\mathcal{C}}[v_0 \notin Q',\ (P_2(\mathcal{C}))(\alpha_0) = f(v_0) \text{ and } A_j(P_2(\mathcal{C}))(\alpha_1), ..., (P_2(\mathcal{C}))(\alpha_k)) = 0] < \epsilon/2$$

where $C_j = (A_j, v_{j,1}, ..., v_{j,k})$, $\Pi(\alpha_0) = v_0$, and $\Pi(\alpha_\ell) = v_{j,\ell}$ for $\ell = 1, ..., k$.

Combining the two cases, soundness is established. Furthermore, the above analysis satisfies the Decomposition Property.