Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 115 (2007)

Combinatorial Construction of Locally Testable Codes*

Or Meir†

May 10, 2010

Abstract

An error correcting code is said to be *locally testable* if there is a test that checks whether a given string is a codeword, or rather far from the code, by reading only a constant number of symbols of the string. While the best known construction of LTCs by Ben-Sasson and Sudan (STOC 2005) and Dinur (J. ACM 54(3)) achieves very efficient parameters, it relies heavily on algebraic tools and on PCP machinery. In this work we present a new and arguably simpler construction of LTCs that is purely combinatorial, does not rely on PCP machinery and matches the parameters of the best known construction. However, unlike the latter construction, our construction is not entirely explicit.

1 Introduction

An error correcting code is said to be *locally testable* if there is a test that checks whether a given string is a codeword, or far from the code, by reading only a constant number of symbols of the string. Somewhat more precisely, a code is locally testable if there exists an algorithm, called the verifier, that when given oracle access to a given string, makes a constant number of queries to the oracle, accepts if the string is a codeword and rejects with high probability if it is far from the code. Codes with related features were implicitly constructed for the first time as part of the efforts to prove the celebrated PCP theorem [A94, S95, RS96, FS95], but since then the notion of locally testable codes has been recognized as interesting by its own right (see, e.g., [GS02, BGHSV06, BS05, D07]).

1.1 A general perspective

The PCP theorem [AS98, ALMSS98] is one of the major achievements of complexity theory. A PCP (Probabilistically Checkable Proof) is a proof that allows checking the validity of a claim by reading only a constant number of symbols of the proof. The PCP theorem asserts

*This work has appeared in SICOMP 39(2). Preliminary versions of this paper have appeared in STOC 08 and as ECCC TR07-115. This research was partially supported by the Israel Science Foundation (grant No. 460/05 and No. 1041/08).

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100 Israel. Email: or.meir@weizmann.ac.il

ISSN 1433-8092

the existence of PCPs of polynomial length for any claim that can be stated as membership in an $\mathcal{NP}$ set. The theorem has found many applications, most notably in establishing lower bounds for approximation algorithms.

The discovery of PCPs of polynomial length, being remarkable by its own right, raises the natural question of how long should a proof be to enjoy local testability. Having shorter locally testable proofs also affects the various applications of PCPs. This consideration motivates the direct study of local testability, and the amount of redundancy it requires. LTCs are a natural tool for such a study.

One motivation for studying LTCs stems from the structure of most PCP constructions. Usually, a PCP system for a given claim is constructed by encoding the proofs of the claim using a special syntax. This syntax allows checking that a string is a valid proof using a constant number of queries, provided that the string follows that syntax.

In order for such syntax to be useful, it usually has to have two additional properties: First, the verification procedure should be able to reject strings that are far from following the syntax using only a constant number of queries. Second, the syntax should have error-correction capabilities, in order to allow the verification procedure to function when it is given access to strings that are close to following the syntax, while not following it entirely. Locally Testable Codes are the most simple objects that possess the both of these properties, and we can therefore hope that a better understanding of LTCs will lead to better constructions of PCPs. Furthermore, we expect constructions of LTCs to be much simpler than constructions of PCPs, since LTCs are not required to allow proof checking.

Another motivation for the study of LTCs is that LTCs can be viewed as the "combinatorial counterparts" of PCPs: While PCPs are "complexity theoretic" objects that are locally testable, LTCs are combinatorial objects that are locally testable. Since combinatorial objects tend to be simpler than complexity theoretic ones, we again may expect the construction of LTCs to be simpler than the construction of PCPs.

Previous work LTCs were discussed in the early PCP and program checking literature (see, e.g., [A94, S95, RS96, FS95]) and they were first systematically studied by Goldreich and Sudan [GS02]. The construction of LTCs that achieves the smallest amount of redundancy was given by Ben-Sasson and Sudan [BS05]. Their construction yields a code that encodes k bits of information into $k \cdot \mathrm{poly}(\log k)$ bits. However, the verifier of their construction only rejects strings that are far from the code with probability of $1/\mathrm{poly}(\log k)$. This limitation was waived later by Dinur [D07] who, by applying her gap amplification technique to the construction of [BS05], improved the rejection probability to a constant, while maintaining the block length of $k \cdot \mathrm{poly}(\log k)$. For a survey of the previous constructions of LTCs, we refer the reader to [G05].

1.2 Our result

Our work was motivated by two considerations: The first consideration is refers to a strange phenomena regarding the previous constructions of LTCs. That is, when taking the view of Section 1.1, one expects LTCs to be weaker objects than PCPs, and to have simpler and weaker constructions. In contrast, previous constructions of LTCs yielded LTCs that were as strong as PCPs, in the sense that they either used PCPs as a building block, or directly

implied constructions of PCPs (e.g. [GS02, BS05]). It seems natural to ask whether our original intuition is wrong, and LTCs are indeed as strong as PCPs, or is this intuition correct, and LTCs are weaker than PCPs.

The other consideration refers to the fact that the previous constructions of LTCs are very algebraic. In particular, the construction of [BS05] uses a very heavy algebraic machinery, even compared to the algebraic machinery common in the PCP literature, and its analysis is quite complicated. While those algebraic techniques proved very useful in the context of PCPs, they seem to give little intuition for why the resulting constructions work. A more inituitive construction, of a combinatorial nature, would have been preferred.

In this paper we give a construction that addresses both those considerations: First, our construction seems to confirm the intuition that LTCs are weaker than PCPs, as our construction does not use PCPs as a building block and does not seem to directly imply a construction of PCPs. Furthermore, our construction is purely combinatorial, and does not make use of algebraic tools. Finally, our construction matches the parameters of the best known construction of [BS05] and [D07].

Explicitness Usually, one wants constructions of codes to be explicit. That is, there should be an efficient algorithm that generates the encoding function of the code for infinitely many message lengths. The construction of [BS05] and [D07] achieves this notion of explicitness. Our construction, however, only manages to achieve a weaker notion of explicitness. That is, we only have a *probabilistic* algorithm that generates the encoding function of the code (for infinitely many message lengths), and this algorithm may err with some negligible probability. In case that the algorithm errs, the encoding function it generates might not constitute a good code, or might constitute a code that is not locally testable.

We stress that even a completely non-explicit construction of LTCs that achieves good parameters would have been valuable. The reason is that, while in the case of many combinatorial objects (such as expander graphs and extractors), one can use a simple counting argument to give a very good non-explicit construction, this is not the case for LTCs. That is, in the case of LTCs, a simple counting argument does not show the existence of LTCs, regardless of the parameters. In this regard we mention that Kaufman and Sudan [KS07] have recently showed that random linear codes with very poor rate are locally testable, while using a very sophisticated analysis.

We also mention that the construction of [GS02] achieves exactly the same notion of explicitness as our construction, though the error probability of their probabilistic algorithm is better than ours.

Remark 1.1. As pointed out by the referees, the term "combinatorial" may not be the best term to distinguish our techniques from the algebraic machinery used in previous constructions. In particular, one may claim that the mere fact that we construct linear codes implies a use of linear algebra, which can also be considered as "algebraic". Indeed, the term "algorithmic" might have been more appropriate than "combinatorial", as we feel that the crucial difference between our construction and previous construction is the "algorithmic mindset" rather than the choice of mathematical techniques. However, the use of the term "combinatorial" to distinguish algorithmic techniques from algebraic ones has already become quite common (see, e.g., [RVW00, GS02, DR06]) and we thus adhere to this convention.

Remark 1.2. We would like to mention that the referees have disagreed about the foregoing motivations for our work. The referees have suggested that the motivation for our construction should be the fact that our construction does not use low degree polynomials, while previous constructions of LTCs used low degree polynomials extensively. Needless to say, the motivations for a line of research are a subjective matter.

1.3 Our techniques

Our construction consists of two main steps, which are analogous to the constructions of [BS05] and [D07]: In the first step we give a construction that achieves block length of $k \cdot \mathrm{poly}(\log k)$, query complexity of $\mathrm{poly}(\log k)$ and rejection probability of $1/\mathrm{poly}(\log k)$. In the second step we reduce the query complexity to a constant and apply the gap amplification technique of [D07] to amplify the rejection probability to a constant. Below we give a rough sketch of the techniques used in the first step of our construction, while the second step follows [D07] quite closely.

Remark 1.3. In this section we make extensive use of coding theory terminology. The reader is referred to Section 2.1 for an overview of this terminology.

Codes with Proofs We begin our construction by defining a notion we call "Code with Proof" (CWP), which is a special case of the notion of PCP of Proximity (PCPP) of [BGHSV06, DR06]. The notion of CWP is a generalization of the notion of LTC in which, in addition to the tested string, the verifier is given oracle access to a "proof string". The proof string can be thought as given by an untrusted prover that tries to convince the verifier to accept the string as a codeword. Intuitively, constructing a CWP should be easier than constructing a LTC, because we can use the proof string in our favor, while LTC can be seen as the special case of CWP where the proof string is empty. A construction of a CWP with short codewords and short proofs can then be transformed to a short LTC with similar parameters using a known reduction (see, e.g., [GS02, Sec. 5] and [BGHSV06, Sec. 4.1]). Thus, we can focus on constructing a CWP with good parameters.

From a coding-theoretic viewpoint, the notion of CWP may be viewed as follows. Intuitively, given a LTC of block length n, one can partition its n coordinates to two types: coordinates that contribute to the relative distance of the code, and coordinates that contribute to the local testability of the code. Most of the known constructions of LTCs do not pay attention to this separation, and treats all the coordinates in the same manner. In our construction, on the other hand, we wish to treat the two types of coordinates differently, and the notion of CWP is the means for making this separation explicit. By treating differently coordinates of different types, we can perform some of the operations in our construction more efficiently, and this is the key to obtaining the parameters that we get.

Remark 1.4. While CWP is indeed a special case of a PCPP, and while PCPPs were used in previous constructions of LTCs (see, e.g., [GS02, BGHSV06, BS05]) we believe that our use of CWPs is fundamentally different and has novelty. For further discussion see titled paragraph toward the end of Section 2.5.

An iterative construction Our construction is an iterative one, and is similar in nature to the zig-zag construction of expander graphs by Reingold et al. [RVW00]. The starting point of our construction is a code of small message length, which is trivially a CWP. We then increase the message length of this CWP iteratively. In every iteration, the parameters of the CWP change as follows:

1. The message length is squared.
2. The rate and the relative distance of the CWP remain intact.
3. The ratio of the message length to the *proof length* of the CWP decreases by a constant factor.
4. The query complexity of the verifier increases by a constant factor.
5. The rejection probability of the verifier decreases by a constant factor.

After $O(\log \log k)$ such iterations, we obtain a CWP of message length k, rate $\Omega(1)$, relative distance $\Omega(1)$, proof length $k \cdot \text{poly}(\log k)$, query complexity $\text{poly}(\log k)$ and rejection probability $1/\text{poly}(\log k)$. Such a CWP translates into an LTC that has the required parameters. It remains to describe the way a single iteration works.

1.3.1 The structure of a single iteration

A single iteration consists of applying to the CWP three basic operations, each aimed at improving or maintaining some other parameters of the CWP. We describe those operations below. In order to describe those operations, we consider their effect on a CWP that has message length k, block length n, rate $R = \frac{k}{n}$, relative distance δ and proof length m. Furthermore, we assume that the code is linear.

Tensor Product In order to square the message length of the code, we use a classical operation on codes called the tensor product. The tensor product of a code C with itself, denoted C^2, is the code whose codewords are the $n \times n$ matrices all of whose rows and columns are codewords of C. It is well known that if C is a linear code, then C^2 is a linear code with message length k^2, block length n^2, rate R^2 and relative distance δ^2.

If C is locally testable, then a natural test for C^2 consists of choosing a random row or a random column of the matrix and testing whether it is a codeword of C. This test extends to the case that C is a CWP, in which case the proof of a codeword of C^2 will consist of the proofs that prove that each row and column of the matrix is a codeword of C. Note that this implies that C^2 has proof length $O(mn)$.

Unfortunately, this "natural test" does not necessarily work in the general case. However, a variant of this test, which makes an assumption regarding the structure of C, does work, and we will have to do some work in order to use it. It is important to note that in this variant too, C^2 has proof length $O(mn)$.

The tensor product operation squares the message length, as required. We would have liked to use the tensor product operation to construct CWPs by repeatedly applying tensor product to a code of small message length (in fact, a similar construction is analyzed in

[BS04]), but the problem is that the tensor product operation squares the rate and the relative distance. Thus, if one starts with a code of constant message length and applies tensor product to it for $\log \log k$ times, then the result will be a code with block length $\text{poly}(k)$ and relative distance $1/\text{poly}(k)$, which are very poor parameters. The two other operations are aimed at improving the rate and the relative distance, respectively.

Random Projection The first problem with the tensor product operation we address is that it squares the rate. In order to improve the rate, we use the random projection operation. This operation consists of choosing a random subset of the coordinates of the code and moving them from the codeword to the proof string. This operation can increase the rate back to R, while increasing the proof length only by a constant factor. It is not hard to prove that this operation roughly maintains the relative distance and local testability of the code.

But why does it help? At first glance, it is not clear why moving coordinates from one place to another should be beneficial. The crucial observation is that the tensor product has a *better effect* on the proof length than on the block length. While the tensor product squares the block length n, it increases the proof length m only by a factor of $O(n)$, which in our case will be much smaller than m. In particular, if we keep n linear in k, the ratio $\frac{k}{m}$ would decrease by only a constant factor in each iteration, which would give us what we wanted.

From the coding-theoretic viewpoint mentioned before, this is exactly where the separation between coordinates that contribute to the relative distance to coordinates that contribute to the local testability is useful: While the tensor product squares the number of coordinates that contribute to the relative distance, which is n, the tensor product only multiplies the number of coordinates that contribute to the local testability of a factor of $O(n)$.

The idea of using a random projection to control the loss of the rate caused by the tensor product is the main novelty of our work. The random projection is also the only reason that our construction is not explicit in the usual sense.

Distance Amplification We still need to make up for the decrease of the relative distance caused by the tensor product. In order to do so, we amplify the relative distance of the code using known techniques (see, e.g., [ABNNR92]). This operation decreases the rate of the code by a constant factor, but this decrease can be compensated by the random projection.

The main difficulty is showing that the distance amplification preserves the local testability of the CWP and the additional structure that is needed for the tensor product to work. In order to demonstrate this we show that applying the distance amplification procedure to a code can be viewed as encoding the code by few simple linear codes that are locally testable and enjoy some weak form of local decodability. Using these properties, we are able to show that the verifier of the original CWP can be emulated by a verifier for the amplified CWP.

By applying the three operations one after the other, we get an iteration that has the required effect on the parameters of the CWP.

1.4 Organization of this paper

In Section 2, we review the relevant background and state the results of this work formally. In Section 3 we give an overview of the construction. We then provide the full details in Sections 4, 5 and 6. Finally, in Section 7, we discuss an interesting connection between our construction and the previous construction of [BS05], as well as possible variants of our construction and some open problems that remain.

2 Preliminaries

For any $n \in \mathbb{N}$, we denote $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. Given any string x over any alphabet, we denote its i-th symbol by x_i and its length by $|x|$. Furthermore, for any string x of length n and for any set $S \subseteq [n]$ of indices $i_1 < i_2 < \ldots < i_{|S|}$ we denote by $x_{|S}$ the projection of x to S, that is, $x_{|S} \stackrel{\text{def}}{=} x_{i_1} x_{i_2} \ldots x_{i_s}$.

2.1 Error Correcting Codes

Let Σ be a finite alphabet. A code C is a one-to-one function from Σ^k to Σ^n, where k and n are called the code's message length and block length, respectively. The rate of the code is defined to be $R_C \stackrel{\text{def}}{=} \frac{k}{n}$. We will sometimes identify C with its image $C(\Sigma^k)$. Specifically, we will write $c \in C$ to indicate the fact that there exists $x \in \Sigma^k$ such that $c = C(x)$. In such case, we also say that c is a codeword of C.

For any two strings $x, y \in \Sigma^n$, the relative Hamming distance between x and y is the fraction of coordinates on which they differ, and is denoted by $\delta(x, y) \stackrel{\text{def}}{=} |\{x_i \neq y_i : i \in [n]\}| / n$. The relative distance of a code C is the minimal relative distance between two different codewords of C, and is denoted by $\delta_C \stackrel{\text{def}}{=} \min_{c_1 \neq c_2 \in C} \{\delta(c_1, c_2)\}$. For a string $x \in \Sigma^n$, we denote by $\delta_C(x)$ the minimal relative distance from x to the nearest codeword of C, that is, $\delta_C(x) \stackrel{\text{def}}{=} \min_{c \in C} \delta(x, c)$. If a string x satisfies $\delta_C(x) \leq \tau$, we say that it is τ-close to C, otherwise we say that it is τ-far from C.

Codes with different message and codeword alphabets It is also possible to define codes that encode strings over one alphabet to strings over another alphabet. All of the above definitions carry through, except for the rate of the code, which is defined as follows: Let Σ and Γ denote finite alphabets, and let $C : \Sigma^k \to \Gamma^n$ denote a code. Then the rate of C is defined to be $R_C \stackrel{\text{def}}{=} \frac{k \log |\Sigma|}{n \log |\Gamma|}$.

Infinite families of codes An infinite family of codes $C = \{C_k\}$ is a sequence of codes such that the code C_k has message length k. The block length $n(k)$, rate $R(k)$ and relative distance $\delta(k)$ of such a family are functions of k such that C_k has block length $n(k)$, rate $R(k)$ and relative distance $\delta(k)$. Throughout this paper we will often work with infinite families of codes, and refer to them simply as "codes". For example, we will say that a code C has block length k^2 and mean that for every k, the code C_k in the family C has block length k^2.

2.1.1 Linear codes

Suppose that $\Sigma = \mathbb{F}$ for some finite field $\mathbb{F}$. In such case we say that $C : \mathbb{F}^k \to \mathbb{F}^n$ is a linear code if C is a linear function.

Suppose that $C : \mathbb{F}^k \to \mathbb{F}^n$ is a linear code. Then $C\left(\mathbb{F}^k\right)$ is a linear subspace of $\mathbb{F}^n$, and thus for every two codewords $c_1, c_2 \in C$ and scalars $a, b \in \mathbb{F}$, the vector $a \cdot c_1 + b \cdot c_2$ is also a codeword. Furthermore, there exists a $k \times n$ matrix G, called the generating matrix of C, that satisfies $C(x) = x \cdot G$ for every row vector $x \in \mathbb{F}^k$.

For any string $x \in \mathbb{F}^n$, the weight of x is the fraction of non-zero coordinates of x, and is denoted by $\mathrm{wt}(x) \stackrel{\text{def}}{=} \delta(x, 0)$. Two immediate conclusions of the foregoing facts are that if C is a linear code, then the zero vector is a codeword of C, and the relative distance of C is equal to $\min_{0 \neq c \in C} \{\mathrm{wt}(c)\}$.

Consider now the case where C is a code over the alphabet $\mathbb{F}^t$ for some natural number t. We say that C is an $\mathbb{F}$-linear code if for every two strings $x, y \in \left(\mathbb{F}^t\right)^k$ and scalars $a, b \in \mathbb{F}$ it holds that $C(a \cdot x + b \cdot y) = a \cdot C(x) + b \cdot C(y)$, where the scalar multiplication is defined by viewing x,y,$C(x)$ and $C(y)$ as vectors in $\mathbb{F}^{kt}$ and $\mathbb{F}^{nt}$. Note that a code that is linear over $\mathbb{F}^t$ is necessarily $\mathbb{F}$-linear, but the converse does not necessarily hold.

2.1.2 Concatenation of codes

We turn to describe the code concatenation technique, which is commonly used in coding theory for reducing the alphabet size of codes. Let Σ and Γ denote finite alphabets, where we think of $|\Gamma|$ as being much larger than $|\Sigma|$. Let $C_1 : \Sigma^k \to \Gamma^n$ and $C_2 : \Gamma \to \Sigma^\ell$ denote codes. The concatenation of C_1 and C_2, denoted $C_1 \diamond C_2 : \Sigma^k \to \Sigma^{n\ell}$, is defined as follows: To encode a message $x \in \Sigma^k$ with $C_1 \diamond C_2$, it is first encoded by C_1, and then every symbol of the result is encoded by C_2. Formally, we define

$$C_1 \diamond C_2(x) \stackrel{\text{def}}{=} C_2\left(C_1\left(x\right)_1\right) C_2\left(C_1\left(x\right)_2\right) \ldots C_2\left(C_1\left(x\right)_n\right)$$

We refer to C_1 as the outer code and to C_2 as the inner code. It is not hard to see that $R_{C_1 \diamond C_2} = R_{C_1} \cdot R_{C_2}$ and that $\delta_{C_1 \diamond C_2} = \delta_{C_1} \cdot \delta_{C_2}$. Furthermore, if $C_1 : \mathbb{F}^k \to \left(\mathbb{F}^t\right)^n$ is an $\mathbb{F}$-linear code and $C_2 : \mathbb{F}^t \to \mathbb{F}^s$ is a linear code then their concatenation $C_1 \diamond C_2$ is a linear code over $\mathbb{F}$.

2.2 Non-standard issues regarding codes

For the rest of this paper, let $\mathbb{F}$ denote a large finite field of some fixed size (say, $|\mathbb{F}| = 64$). Unless stated explicitly otherwise, all our codes will be over the alphabet $\mathbb{F}$, and will also be linear codes. The reason for using codes over $\mathbb{F}$, rather than binary codes, is that one of the theorems we use requires the codes to have a very large relative distance (i.e., slightly more than $\sqrt[4]{7/8} \approx 0.967$), and such large relative distance can only be achieved using a sufficiently large alphabet.

Definition 2.1. Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a code and let $S \subseteq [n]$. We denote by $C_{|S}$ the function from $\mathbb{F}^k$ to $\mathbb{F}^{|S|}$ that is defined by

$$\forall x \in \mathbb{F}^k : \; C_{|S}(x) = C\left(x\right)_{|S}$$

The function $C_{|S}$ is called the projection of C to S. Note that $C_{|S}$ is not necessarily a code, since it is not necessarily one to one.

2.3 Probabilistic Circuits

A probabilistic circuit that tosses r coins is a circuit that is given, in addition to its input, a string of r bits that is chosen uniformly at random. The additional random string is referred to as the coin tosses of the circuit, and the output of the circuit is a distribution over its coin tosses.

In this paper we will use probabilistic oracle circuits. Note that the use of probabilistic circuits is not common, since a probabilistic circuit can usually be transformed into a deterministic circuit. However, we are interested in probabilistic oracle circuits that make very few queries to their oracle, and these have no deterministic counterparts.

2.4 Locally Testable Codes

A code is locally testable if it is possible to test whether a given string is a codeword, or far from being a codeword, by reading only a small number of its symbols. We now give a formal and quantitative definition of this intuitive notion.

Definition 2.2. A code $C : \mathbb{F}^k \to \mathbb{F}^n$ is said to be (q, τ, ε)-Locally Testable if there exists a probabilistic oracle circuit V that satisfies the following requirements:

1. The oracle is a string over $\mathbb{F}$.
2. V makes at most q non-adaptive queries to its oracle.
3. For every codeword $c \in C$, it holds that $\Pr[V^c \text{ accepts}] = 1$.
4. For every string $w \in \mathbb{F}^n$ that is τ-far from C, it holds that $\Pr[V^w \text{ rejects}] \geq \varepsilon$.

The circuit V is called the verifier of C, the parameter q is called the query complexity of C, the parameter τ is called the distance threshold of C and the parameter ε is called the rejection probability of C.

Remark 2.3. The common definition of LTCs uses Turing machines instead of circuits. We chose to use a more general definition that allows circuits in order to handle the case where the code C is not explicit.

Remark 2.4. Note that we chose to limit the verifier to non-adaptive queries. It is also possible to define LTCs with respect to adaptive queries. However, any verifier that makes q adaptive queries can be emulated by a verifier that makes $|\mathbb{F}|^q$ non-adaptive queries. Furthermore, in the context of linear codes, adaptive verifiers are no more powerful than non-adaptive verifiers (i.e., they have the same query complexity [BHR05])

Our result can now be stated as follows:

Theorem 2.5. *There exists an infinite family of locally testable codes* $\{C_k\}_k$ *such that* C_k *has block length* $k \cdot \mathrm{poly}(\log k)$*, relative distance* $\Omega(1)$*, query complexity* $O(1)$*, arbitrarily small constant distance threshold* $\tau > 0$*, and rejection probability* $\Omega(1)$*. Furthermore, the codes in the family are linear and there exists a probabilistic algorithm that on input* k*, runs in time* $\mathrm{poly}(k)$ *and outputs with probability* $1 - \exp(-\mathrm{poly}(\log k))$ *the generating matrix and verifier circuit of* C_k*.*

We refer the reader to Theorem 6.29 for a more detailed statement of the LTCs that we construct.

The codes of Theorem 2.5 are over the alphabet $\mathbb{F}$. Using concatenation with any binary code, one can get binary LTCs with roughly the same parameters. In particular, since $\mathbb{F}$ is of constant size, the concatenation will increase the query complexity of the code only by a constant factor. If one wants to get *linear* binary LTCs, one only needs to choose $\mathbb{F}$ to be an extension field of GF(2).

2.5 Codes with Proofs

We now define the notion of "Code With Proof" (CWP). Intuitively, a CWP is a generalization of LTC, in which the verifier is given, in addition to oracle access to the tested string, an oracle access to a "proof string", which is supposed to "prove" that the tested string is indeed a codeword. Formally, a CWP is defined as follows.

Definition 2.6. A code $C : \mathbb{F}^k \to \mathbb{F}^n$ is said to be a (q, ε)-Code With Proof (CWP) if there exists a probabilistic oracle circuit V that satisfies the following requirements:

1. The oracle is a string over $\mathbb{F}$.
2. V makes at most q non-adaptive queries to its oracle.
3. For every codeword $c \in C$ there exists a string π_c over $\mathbb{F}$ such that $\Pr[V^{c,\pi_c} \text{ accepts}] = 1$. We refer to π_c as a proof string of c with respect to V.
4. For every string $w \in \mathbb{F}^n$ and every string π over $\mathbb{F}$, we have that $\Pr[V^{w,\pi} \text{ rejects}] \geq \varepsilon \cdot \delta_C(w)$. We refer to w as the "tested string" and to π as the "proof string".

If V tosses at most r coins, then C is said to be a (q, ε, r)-CWP.

The circuit V is called the verifier of C, the parameter q is called the query complexity of C, the parameter ε is called the rejection ratio of C, and the parameter r is called the randomness complexity of C. Another measure that is of interest is the length of the proof strings: We say that C has proof length m if for every codeword $c \in C$ there exists a proof string π_c of c whose length is at most m. In such case we define the proof rate of C to be the ratio k/m. Furthermore, we will sometimes refer to the rate of C as the code rate, in order to distinguish it from the proof rate.

We turn to discuss few important features of Definition 2.6.

The randomness complexity Note that Definition 2.6 keeps track of the randomness complexity of the verifier, instead of its proof length, which may seem more natural. However, knowing the randomness complexity of the CWP will be important at some points of the construction. Moreover, in some parts of the construction it is more comfortable to track the randomness complexity of the CWP than tracking its proof length. Note that we can assume without loss of generality that the proof length of the CWP is bounded by $2^r \cdot q$.

CWPs imply LTCs Every CWP of block length n and proof length m can be transformed to a LTC of block length $O(n+m)$ using a known reduction (see Section 6.2 for more details). Thus, we construct the LTCs of Theorem 2.5 by constructing the following CWPs:

Theorem 2.7. *There exists an infinite family of CWPs* $\{C_k\}_k$ *such that* C_k *has block length* $O(k)$*, relative distance* $\Omega(1)$*, query complexity* $O(1)$*, rejection ratio* $\Omega(1)$ *and randomness complexity* $\log k + O(\log\log k)$*. Furthermore, the codes in the family are linear and there exists a probabilistic algorithm that on input* k*, runs in time* $\mathrm{poly}(k)$ *and outputs with probability* $1 - \exp(-\mathrm{poly}(\log k))$ *the generating matrix and verifier circuit of* C_k*.*

We refer the reader to Theorem 6.28 for a more detailed statement of the CWPs we construct.

A stronger soundness requirement Note that the soundness requirement of CWPs (Definition 2.6, Requirement 4) is stronger than the corresponding requirement of LTCs (Definition 2.2, Requirement 4). In particular, the definition of LTCs requires the verifier to reject only strings that are far from the code, while the definition of CWPs requires the verifier to reject *any* non-codeword with adequate probability. This difference between the soundness requirements is an artifact of the transformation of CWPs to LTCs, which loses the stronger soundness property. We mention that LTCs/PCPPs satisfying the stronger soundness requirement are known as "strong LTCs/PCPPs" (see, e.g., [BS05, Definition 2.7]) .

CWPs and PCPPs A reader who is familiar with the PCP literature will note that the notion of CWP is a special case of the notion of PCP of Proximity (PCPP), introduced in [BGHSV06] and [DR06]. Specifically, CWPs are good codes coupled with PCPPs that are able to prove membership in those codes. While PCPPs were used in previous works to construct LTCs, our use of CWPs is fundamentally different for two reasons:

1. Our definition of CWP couples the code with the PCPP used to prove memebership in the code, while previous works treat the code and the PCPP as two separate objects. This coupling of the code and the PCPP makes it possible to define transformations that act on CWPs and could not be defined when the codes and PCPPs are separated. In particular, our main new transformation, namely the random projection of CWPs (see Sections 3.2.2 and 4.2) is such a transformation.
   Another way to view the benefit of the coupling of codes and PCPPs is as follows. The previous works construct PCPPs for codes by first fixing the code and then constructing a PCPP for this code. In contrast, our work constructs the code and the PCPP simultaneously, which allows adapting the construction of the code to the construction of the PCPP.

2. Most of the previous works that use PCPPs (with the sole exception of [BS05]) construct LTCs by first constructing PCPPs for an $\mathcal{NP}$-complete set, and then deriving LTCs from those PCPPs. On the other hand, CWPs are PCPPs for codes, which are sets in $\mathcal{P}$. Hence, CWPs are weaker than the PCPPs that are usually constructed, which corresponds with our goal of constructing LTCs without PCP machinery. This fact also suggests that CWPs should be simpler and easier to construct.

A coding theoretic perspective As mentioned in the introduction, one may take the following "coding-theoretic" perspective at CWPs. Any construction of an LTC should achieve the following properties: The code should have good relative distance, and it should be locally testable. Thus, each coordinate of the code should contribute to at least one of those goals, and one can partition the coordinates of the code into those that contribute to the relative distance and those that contribute to the local testability. It turns out that it is sometimes useful to treat those two kinds of coordinates differently.

The notion of CWP allows us to make this partition of the coordinates explicit. Specifically, given a CWP C that has a codeword c and a corresponding proof π_c, we can think of the string $c\pi_c$ as being the codeword of some LTC C', where the coordinates of c are the "relative distance" coordinates of C' and the coordinates of π_c are the "local testability" coordinates of C'.

3 Overview of the construction

In this section, we give an overview of our main construction. This construction yields CWPs that almost achieve the parameters of Theorem 2.7, but have query complexity $\mathrm{poly}(\log k)$ and rejection ratio $1/\mathrm{poly}(\log k)$. In order to obtain CWPs of constant query complexity and constant rejection ratio, as stated in Theorem 2.7, we apply a known query reduction technique and the gap amplification technique of Dinur [D07] (see Sections 6.3 and 6.4 for details). We then transform the latter CWPs to LTCs with good parameters using a standard transformation (see Section 6.2).

For simplicity, in this section we measure the proof length of the CWP rather than its randomness complexity. Actually, instead of considering the proof length of the CWP, it will be more convenient to consider its *proof rate*. Similarly, it will be more convenient to consider the rate of the CWP than considering its block length. Table 1 summarizes the parameters of the CWPs that we construct in this section.

3.1 The Tensor Product operation and a simple construction of LTCs

We begin by introducing the Tensor Product operation on codes. Given a code C of message length k and block length n, the tensor product of C with itself, denoted C^2, is a code of message length k^2 and block length n^2 defined as follows. To encode a message $x \in \mathbb{F}^{k^2}$, viewed as a $k \times k$ matrix, we first encode every row of x with C. Let y be the resulting $k \times n$

Parameter name	Parameter value
Message length	k
Rate	$\Omega(1)$
Relative distance	$\Omega(1)$
Proof rate	$1/\operatorname{poly}(\log k)$
Query complexity	$\operatorname{poly}(\log k)$
Rejection ratio	$1/\operatorname{poly}(\log k)$

Table 1: The parameters of the main construction of CWPs

matrix. We then encode every column of y with C, and define the resulting $n \times n$ matrix to be $C^2(x)$. Note that if C has rate R, then C^2 has rate R^2. It is also easy to prove that if C has relative distance δ then the relative distance of C^2 is δ^2 (see [S01, Lecture 6]).

The tensor product is one of the simplest ways to obtain a code of large message length from a code of small message length. Since a code with constant message length is trivially locally testable, we can hope to construct locally testable codes with large message length by repeatedly applying tensor product to a code with constant message length. In order to implement this idea, we need to show that if C is a locally testable code, then C^2 is locally testable with related parameters.

One can show, using the linearity of C, that the codewords of C^2 are exactly the $n \times n$ matrices all of whose rows and columns are codewords of C (see [S01, Lecture 6]). If C is locally testable, then this fact suggests the following natural verifier for C^2: Given an $n \times n$ matrix to be tested, the verifier chooses either a random row or a column of the matrix, and invokes the verifier of C to test whether this row/column is a codeword of C.

Does this natural verifier "work"? It turns out that while in some important cases the answer is "yes" (see [PS94, DSW06]), in the general case, the answer is "no" (see [V05, CR05, GM07a]). However, we assume for now that that the answer is "yes" also in the general case. Specifically, we assume the following:

Simplifying Assumption for LTCs: There exists a constant $\alpha \leq 1$ such that given a (q, τ, ε)-locally testable code C, the code C^2 is $(q, \tau, \alpha \cdot \varepsilon)$-locally testable with respect to the verifier that chooses a random row/column and tests it.

In Section 3.3, we describe how the simplifying assumption can be removed by using a result of Ben-Sasson and Sudan [BS04] and making an assumption regarding the structure of the code C.

Using the simplifying assumption, we can give a simple construction of locally testable codes: Suppose we wish to construct a locally testable code with message length k. For some constant k_0, let C_0 be a code with message length k_0. The code C_0 is trivially locally testable with query complexity k_0, distance threshold $1/k_0$ and rejection probability 1. We now define a sequence of codes $\{C_i\}_i$ by setting $C_{i+1} = C_i^2$. It is easy to see that $C_{\log \log_{k_0} k}$ has message length k, and that it is locally testable with query complexity k_0, distance threshold $1/k_0$, and rejection probability $1/\operatorname{poly}(\log k)$.

This construction of locally testable codes is very simple, but the codes it yields are very poor: The rate and relative distance of $C_{\log \log_{k_0} k}$ are $1/\operatorname{poly}(k)$. In the next subsection we show that additional ideas may improve both parameters.

3.2 A simplified construction of CWPs with good parameters

Recall that our final goal is to construct LTCs with rate $1/\operatorname{poly}(\log k)$ and relative distance $\Omega(1)$. The simple construction of Section 3.1 falls short of achieving those parameters, and only achieves rate and relative distance of $1/\operatorname{poly}(k)$. Recall that this construction is working in iterations, where the i-th iteration applies tensor product to the LTC C_{i-1} and creates the LTC C_i. Observe that the reason that this construction has such poor rate and relative distance is that in every iteration the rate and relative distance are *square*d. If we can change the way a single iteration works so that in every iteration the rate will decrease only by a constant factor and the relative distance will be maintained, then we will get LTCs with the parameters we want. We now describe how this idea can be implemented in the context of CWPs.

We design a transformation that transforms a CWP C into a CWP C' with the following effect on the parameters:

Parameter name	C	C'
Message length	k	k^2
Rate	R	R
Relative distance	δ	δ
Proof rate	P	$\gamma \cdot P$ for a constant $\gamma < 1$
Query complexity	q	$\beta \cdot q$ for a constant $\beta > 1$
Rejection ratio	ε	$\alpha \cdot \varepsilon$ for a constant $\alpha < 1$

Table 2: The effect of a single iteration

Having designed a transformation as stated in Table 2, we construct CWPs with the parameters stated in Table 1 by starting with a trivial CWP of constant message length and iteratively applying the transformation for $O(\log \log k)$ iterations. The transformation consists of three basic operations:

1. Tensor Product - In order to square the message length, we use the tensor product operation. We extend the tensor product operation to the setting of CWPs in Section 3.2.1.

2. Random Projection - This operation increases the rate. See Section 3.2.2 for details.

3. Distance Amplification - This operation increases the relative distance. See Section 3.2.3 for details.

By applying the basic operations one after the other, we obtain the transformation stated in Table 2. In the rest of this subsection, we describe each of the operations in detail.

3.2.1 Tensor product of CWPs

Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a CWP with a verifier V. We would like the code C^2 to be CWP with related parameters. In order to verify C^2, we wish to use the natural verifier that given an $n \times n$ matrix, emulates V on a random row/column. But, in contrast to the case of LTCs, in the case of CWPs the verifier V needs to be given oracle access to a proof string in addition to the row/column it tests. We therefore define the proof string of a codeword c of C^2, that is viewed as an $n \times n$ matrix, to be the collection of the proof strings that correspond to each row and column of c (as a codeword of C).

We define the verifier V' of C^2 as follows: Given oracle access to a tested string w and a proof string π, the verifier V' views w as an $n \times n$ matrix and interprets π as a collection of strings $\pi_1, \ldots, \pi_{2n}$ such that every string π_i corresponds to some row or column of w. The verifier V' then chooses a random row/column and emulates the verifier V with oracle access to this row/column and to its corresponding proof string π_i. The verifier V' accepts if and only if the emulation of V accepts.

Clearly, if w is a codeword of C^2 and every string π_i is a proof string of the corresponding row/column, then the verifier V' accepts. In order to deal with the rejection ratio, we extend our simplifying assumption as follows:

Simplifying Assumption for CWPs: There exists a constant $\alpha \leq 1$ such that given a (q, ε)-CWP C, the code C^2 is a $(q, \alpha \cdot \varepsilon)$-CWP with respect to the verifier and proof strings described above.

As before, if C has rate R and relative distance δ, then C^2 has rate R^2 and relative distance δ^2. A new feature of the tensor product for CWPs is the proof rate: If C has block length n and proof length m, then C^2 has proof length of $2n \cdot m$. This means that if C has proof rate P, then C^2 has proof rate $\frac{1}{2} \cdot R \cdot P$. A crucial point now becomes visible - the tensor product does not square P, but rather multiplies it by R.

3.2.2 Random Projection

In this subsection we describe the random projection operation, which we use to improve the rate of the CWPs. Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a CWP. The most straightforward way to increase the rate of C is to project C to a subset $S \subseteq [n]$. This poses two problems:

1. The function $C_{|S}$ does not necessarily have a good relative distance. Moreover, $C_{|S}$ may not be one-to-one.

2. The function $C_{|S}$ may not be a CWP. In particular, we can no longer use the verifier of C, because this verifier may query one of the coordinates that were projected out.

The first issue is solved by choosing the subset $S \subseteq [n]$ *uniformly at random*. It can be shown that if S is a sufficiently large random set, then with high probability the function $C_{|S}$ is a code that maintains the relative distance of C up to a constant factor. The second issue is solved by providing the verifier of C with the "missing coordinates" in the proof string. That is, for every $c \in C$, we define the proof string of the codeword $c_{|S} \in C_{|S}$ to contain $c_{|[n]\setminus S}$ in addition to the proof string of c. We refer to the operation of choosing a random subset of

Parameter Name	Initial CWP	After Tensor Product	After Random Projection
Message length	k	k^2	k^2
Block Length	n	n^2	$k^2/R = k \cdot n$
Proof Length	$m \geq n$	$2n \cdot m$	$2n \cdot m + n^2 - k \cdot n < 4n \cdot m$
Rate	R	R^2	R
Proof rate	$P \leq R$	$\frac{1}{2} \cdot R \cdot P$	$\frac{k^2}{2n \cdot m + n^2 - k \cdot n} > \frac{1}{4} \cdot R \cdot P$

Table 3: The effect of a single iteration on the parameters of a CWP whose block length is shorter than its proof length.

the coordinates, projecting the code to this subset, and moving the other coordinates to the proof part as Random Projection.

At first glance, including the missing coordinates in the proof part seems weird. Our goal in projecting out those coordinates was reducing the redundancy of C. If we just move those coordinates from the codeword part to the proof part, it seems that we do not gain anything in terms of redundancy.

One of the most important observations of this work is that decreasing the block length of a CWP at the expense of increasing its proof length is *beneficial*. The reason is that while the tensor product operation squares the block length, it only multiplies the proof length by a factor that depends on the block length. Thus, decreasing the block length of a CWP C improves the proof length of C^2. This effect turns out to be very significant, and therefore moving coordinates from the codeword part to the proof part, while not giving an "immediate" gain in the redundancy (in C), is beneficial in the "long run" (in C^2).

To see it, suppose that a single iteration of our construction consisted of applying a tensor product to the CWP, and then applying random projection to the CWP to increase its rate back to what it was before the tensor product. Consider the effect of applying such iterations to a CWP of code rate R and proof rate P. Assume that $P \leq R$, and note that in such case the random projection decreases the proof rate by a factor of at most 2. After the first iteration, the CWP will have code rate R and proof rate $\frac{1}{4} \cdot R \cdot P$, because the tensor product multiplies the proof rate by $\frac{1}{2} \cdot R$ and the random projection multiplies the proof rate by $\frac{1}{2}$ (since we assume $P \leq R$). After the second iteration, the CWP will have code rate R and proof rate $\left(\frac{1}{4} \cdot R\right)^2 \cdot P$. In general, after i iterations the CWP will have code rate R and proof rate $\left(\frac{1}{4} \cdot R\right)^i \cdot P$. It is now easy to see the benefit of the random projection by comparing this construction with the construction of Section 3.1, in which after i iterations the CWP had code rate of R^{2^i}. Table 3 summarizes this example.

We conclude that, by using random projection to maintain the code rate the same in all iterations, we can make the proof rate decrease by only a constant factor in each iteration, as stated in Table 2.

Remark 3.1. Once we use the operation of Random Projection, our construction ceases to be explicit in the usual sense, because the set of projected coordinates was selected at random. Note that the only reason we need the set of coordinates to be random is that we need the projection to preserve the relative distance of the code. Thus, if we could deterministically find a set of coordinates such that if we project the code to this set, the

relative distance is preserved up to a constant factor, then our construction would have been explicit.

3.2.3 Distance Amplification

In this subsection we describe the distance amplification operation, which we use to improve the relative distance of the CWPs. Specifically, we present a distance amplification procedure that can increase the relative distance of a code from any constant to any constant that is smaller than $\frac{|\mathbb{F}|-1}{|\mathbb{F}|}$, while decreasing the rate by only a constant factor. We comment that similar distance amplification procedures are known in coding theory literature for quite some time (see, e.g., [ABNNR92]). Our contribution is merely observing that these procedures preserve local testability.

Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a code with relative distance δ. We wish to improve the relative distance of C while not decreasing its rate by too much. In order to do so, we take the following view on the relative distance of C: Consider the "experiment" that given a non-zero codeword $c \in C$, chooses a random coordinate $i \in [n]$ and "succeeds" if c_i is non-zero. Since the relative distance of C is the minimal weight of a non-zero codeword of C, the probability that the above experiment succeeds is δ. Furthermore, the experiment tosses only $\log n$ coins.

Observe that improving the relative distance of C while incurring only a small loss to the rate of C is analogous to increasing the success probability of the above experiment while not tossing too many additional coins. One possible way of improving the success probability in a randomness efficient manner is taking random walks on expander graph. Let G be a d-regular expander on n vertices with relative second eigenvalue λ. We identify the vertices of G with the coordinates of c. The probability that a random walk of length t of G hits a coordinate i such that $c_i \neq 0$ is at least $1-(1-\delta+\lambda)^t$. Furthermore, such a random walk tosses only $\log n + t \log d$ coins.

We adapt the above method to the problem of improving the relative distance of C. Define a code $C' : \mathbb{F}^k \to \left(\mathbb{F}^{t+1}\right)^{d^t n}$ as follows (Note that the codewords of C' are over the alphabet $\mathbb{F}^{t+1}$): Let $x \in \mathbb{F}^k$ be a message. We identify every coordinate of $C(x)$ with a vertex of G and every coordinate of $C'(x)$ with a walk of length t on G. Now, for every walk $(i_0, \ldots, i_t) \in [n]^t$ of length t on G we define the coordinate of $C'(x)$ that corresponds to $(i_0, \ldots, i_t)$ to be

$$C'(x)_{(i_0,\ldots,i_t)} \stackrel{\text{def}}{=} \left(C(x)_{i_0}, \ldots, C(x)_{i_t}\right) \in \mathbb{F}^{t+1} \tag{1}$$

By the discussion above, the relative distance of C' is at least $1-(1-\delta+\lambda)^t$. We finish the distance amplification procedure with reducing the alphabet of C' back to $\mathbb{F}$ by concatenating C' with some good inner code over the alphabet $\mathbb{F}$. Let $\mathrm{DistAmp}(C)$ denote the result of the concatenation. By choosing t to be a sufficiently large constant and by choosing an inner code of sufficiently large relative distance, we can amplify the relative distance of C to any constant less than $\frac{|\mathbb{F}|-1}{|\mathbb{F}|}$, while decreasing the rate of C only by a constant factor which depends on t and on the inner code.

The local testability of $\mathrm{DistAmp}(C)$ In order to use the distance amplification procedure, we need to show that if C is a CWP then so is $\mathrm{DistAmp}(C)$. We sketch the argument

below.

Assume that C is a CWP. We begin by observing that C' is a CWP: The reason is that the transformation from $C(x)$ to $C'(x)$ that was described in Equation 1 can be viewed as applying a certain repetition code to c. Thus, verifying that a string is a codeword of C' amounts to testing a repetition code and emulating the verifier of C.

Now, to show that $\mathrm{DistAmp}(C)$ is a CWP, observe that a verifier for $\mathrm{DistAmp}(C)$ can emulate the verifier of C' as follows: Whenever the verifier of C' queries a coordinate i, the verifier of $\mathrm{DistAmp}(C)$ reads the supposed encoding of the coordinate i, checks that it is a legal codeword of the inner code, and uses it to answer the query of the verifier of C'.

Note that the verifier of $\mathrm{DistAmp}(C)$ uses more queries than the verifier of C. Specifically, every time we apply distance amplification to a CWP, its query complexity increases by a constant factor. However, we can afford this increase, since it still matches the parameters stated in Table 2.

Remark 3.2. The distance amplification procedure described above is only one way out of many to improve the relative distance of a code. A similar way was described in [ABNNR92], who used neighborhoods of vertices in a bipartite expander instead of random walks on a non-bipartite expander. Furthermore, as explained above, the intuition of the distance amplification comes from amplifying hitting probabilities, and therefore any randomness efficient hitter can be used for distance amplification. For example, one could take neighborhoods of vertices in a disperser instead of a bipartite expander.

3.2.4 The construction so far

Using the ideas described in this subsection, we can now present a simplified construction of CWPs with good parameters. Suppose we wish to construct a CWP of message length k, rate R and relative distance δ. For some constant k_0, let $C_0 : \mathbb{F}^{k_0} \to \mathbb{F}^{n_0}$ be a code with rate R and relative distance δ. The code C_0 is trivially a CWP with query complexity n_0, rejection ratio 1 and proof length 0. We define a sequence of CWPs $\{C_i\}_i$, where the CWP C_{i+1} is obtained from from C_i as follows:

1. Apply tensor product to C_i and define C_i^{TP} to be C_i^2.
2. Apply random projection to C_i^{TP} to improve its rate to R. Let C_i^{RP} denote the result.
3. Apply distance amplification to C_i^{RP} to improve its relative distance back to δ. Set C_{i+1} to be the result.

It is not hard to see that $C_{\log\log_{k_0} k}$ has message length k, proof rate and rejection ratio $1/\operatorname{poly}(\log k)$, query complexity $\operatorname{poly}(\log k)$, constant rate and constant relative distance, as stated in Table 1.

3.2.5 A coding-theoretic perspective

The foregoing construction relies on the observation that the tensor product of CWPs has a better effect on the proof length than on the block length. As mentioned in the introduction, one can view this observation from a coding-theoretic perspective. We discuss this perspective on the intuitive level.

Let C be an LTC that has message length k and relative distance δ, and suppose we wish to square the message length of C. The straightforward way to construct an LTC of message length k^2 from C is as follows: Given a message of length k^2, partition it to k blocks of length k, and encode each of them with C. The code that results from this method has message length k^2, as we wanted. This code also has the same rate as C, and it can be easily shown that this code is locally testable. The problem is that this code will have relative distance δ/k; that is, this method decreases the relative distance by a factor of k, which we can not possibly afford in our construction.

Indeed, the main reason we use the tensor product operation in our construction is because we want to increase the message length of C without decreasing its relative distance *by too much*. However, as noted in Section 3.1, using the tensor product operation creates a new problem, as this operation squares the rate of the code.

We now argue that by taking a closer look at C, we can actually make the tensor product more efficient. Let n' denote the block length of C. As an LTC, C should achieve two goals: it should have good relative distance, and it should be locally testable. Thus, each of the n' coordinates of C should contribute to achieving at least one of those goals - or otherwise this coordinate is redundant and can be dropped. Suppose that among the n' coordinates, there are n coordinates that contribute to the relative distance of C, and m coordinates that contribute to the local testability of C. The crucial point is that since the tensor product operation is aimed at maintaining the relative distance of C, we can apply the tensor product operation only to the n "relative distance" coordinates of C, and treat the m "local testability" coordinates in a different and more efficient way.

More concretely, we can transform C into a code C' of message length k^2 as follows: A codeword of C' consists of three parts. The first part is an $n \times n$ matrix, that has the property that each row and each column of the matrix form the n "relative distance" coordinates of some codeword of C. The second part of the codeword of C' contains for each row of the matrix the m "local testability" coordinates of the codeword of C to which the row corresponds. The third part of the codeword of $C'$is the same as the second part with respect to the columns of the matrix. Thus, C' has block length $n^2 + 2nm$, which is considerably better then the block length of C^2 (which is $(n+m)^2$) if m is significantly larger than n.

This transformation from C to C' is not without problems, and indeed, our construction uses somewhat more complicated ideas. However, this transformation might serve as a coding-theoretic intuition for why our construction works. In particular, this transformation demonstrates the importance of maintaining the separation between the "relative distance" coordinates of an LTC to its "local testability" coordinates. Indeed, this is the reason we use the notion of CWP in our construction: This notion provides the formal way of separating the two kinds of coordinates, where the "proof strings" correspond to the "local testability" coordinates of the LTC, and the "codewords" corresponds to the "relative distance" coordinates.

3.3 Removing the simplifying assumption and the Full Construction

A crucial issue, of course, is removing the simplifying assumption (of Section 3.2.1). Recall that this assumption is that for every CWP C, the code C^2 is a CWP with related parameters. This assumption is *not true* in general. However, we can use a special case (presented in [BS04]) for which the assumption is true in order to make our construction work. The result of [BS04] says, roughly, that if a CWP C is of the form $C = C_s^2$ for some code C_s, then C^2 is a CWP with related parameters (Note that C_s is not necessarily a CWP). We note that the result of [BS04] uses a slightly more sophisticated verifier for C^2 than the row/column verifier we used before. We describe this verifier in Section 3.3.1.

We say that a code C is of a square form if there exists a code C_s such that $C = C_s^2$. In order to use the result of [BS04], we need to make sure that every CWP to which we apply tensor product is of a square form. To do so, we will maintain this form as an invariant throughout the iterations of our construction: We will start with an initial CWP that is of a square form, and then show that every iteration preserves the square form. In order to show that a single iteration preserves the square form, we will need to modify the random projection and distance amplification operations such that they preserve the square form (while the tensor product trivially preserves the square form). In Section 3.3.2, we describe the required modification of the random projection and the distance amplification operations.

3.3.1 The Axis Parallel Planes Test

Let C_s be a code with block length n_s, and let $C = C_s^2$ be a CWP with block length $n = n_s^2$. Recall that the codewords of C^2 are the matrices such that each of their rows and columns is a codeword of C. We now view the code C^2 as follows: C^2 has block length n_s^4, and we view the strings of length n_s^4 as 4-dimensional hypercubes with coordinates in $[n_s]^4$. Taking this view, one can show that a string $w \in \mathbb{F}^{n_s^4}$ is a codeword of C^2 if and only if every restriction of w to an axis parallel plane of the hypercube is a codeword of C (see Section 4.1.1 for details).

The latter characterization of the codewords of C^2 suggests the following "planes verifier" for C^2: Given a tested string w, the verifier views w as a 4-dimensional hypercube, chooses a random axis parallel plane of w and verifies that it is a codeword of C (by emulating the verifier of C). It turns out that this verifier works well. Using a result of [BS04], we show that if C is a CWP with rejection ratio ε, then C^2 with respect to the planes verifier is a CWP with rejection ratio $2^{-32} \cdot \varepsilon$ (see Section 4.1 for details). Note that in order for the "planes verifier" to work, it needs to be given access to the proof strings of all the axis parallel planes. Thus, the proof string of a codeword c of C^2 will consist of the proof strings that prove that each axis parallel plane of c is a codeword of C. It can be shown that if C has block length $n = n_s^2$ and proof length m, then C^2 has proof length $\binom{4}{2} \cdot n \cdot m = 6 \cdot n \cdot m$, rather than $2 \cdot n \cdot m$ as in Section 3.2.1. However, the extra factor of 3 is immaterial.

Remark 3.3. The result of [BS04] only holds if C_s has a sufficiently large relative distance (i.e., greater than $\sqrt[4]{\frac{7}{8}}$). Thus, we will have to make sure that the relative distance of C_s is that large. Such relative distance can be achieved using the distance amplification. The

need to have codes of such high relative distance is the reason why we need to work with codes over some sufficiently large finite field $\mathbb{F}$ instead of binary codes.

3.3.2 Preserving the Square Form

In this subsection we describe how we modify the random projection and distance amplification operations such that they preserve the square form of the CWP. Let us focus for now on the random projection operation. Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a CWP of a square form, that is, $C = C_s^2$ for some code $C_s : \mathbb{F}^{k_s} \to \mathbb{F}^{n_s}$. Suppose we want to improve the rate of C. If we project C to some random subset of coordinates $S \subseteq [n]$, it is unlikely that that $C_{|S}$ will be of a square form, since the probability that the subset S will have the form of a square is very low.

The solution to the problem is to apply a random projection to C_s instead of C. That is, in order to improve the rate of C, we choose a set $T \subseteq [n_s]$ uniformly at random and take $\left(C_{s|T}\right)^2$ to be our new CWP. Clearly, $\left(C_{s|T}\right)^2$ is of a square form and has a better rate than $C = C_s^2$. Furthermore, $\left(C_{s|T}\right)^2$ is indeed a CWP, since $\left(C_{s|T}\right)^2 = C_{|T\times T}$.

We use the same solution for the distance amplification operation: In order to improve the relative distance of C, we take $(\text{DistAmp}(C_s))^2$ to be our new CWP. The code $(\text{DistAmp}(C_s))^2$ is of a square form and has a better relative distance than C. However, it is not clear why should $(\text{DistAmp}(C_s))^2$ be a CWP. The way of showing that $(\text{DistAmp}(C_s))^2$ is a CWP is another novelty of this paper. Below we give a brief summary of this proof (see Section 5 for details).

We begin by observing that applying distance amplification to a codeword of C_s is a linear operation - that is, there is a linear function A such that for every message $x_s \in \mathbb{F}^{k_s}$ it holds that

$$A\left(C_s(x_s)\right) = \left(\text{DistAmp}(C_s)\right)(x_s)$$

We proceed by viewing $(\text{DistAmp}(C_s))^2$ as follows. In order to encode a message $x \in \mathbb{F}^k$ with $(\text{DistAmp}(C_s))^2$, view x as $k_s \times k_s$ matrix and perform the following steps:

1. Encode every row of x with C_s. Denote the result by x^1.
2. Apply A to every row of x^1. Denote the result by x^2.
3. Encode every column of x^2 with C_s. Denote the result by x^3.
4. Apply A to every column of x^3. Set $\left(\text{DistAmp}(C_s)\right)^2(x)$ to be the result.

We claim that using the linearity of C_s and A, we can switch the order of Steps 2 and 3. After switching the order, the encoding procedure of $(\text{DistAmp}(C_s))^2$ becomes:

1. Encode x with $C = C_s^2$. Denote the result by x^1.
2. Apply A to every row of x^1. Denote the result by x^2.
3. Apply A to every column of x^2. Set $\left(\text{DistAmp}(C_s)\right)^2(x)$ to be the result.

Thus, $(\text{DistAmp}(C_s))^2(x)$ is obtained by applying A many times to parts of $C(x)$. We next show that a verifier of $(\text{DistAmp}(C_s))^2$ can test a candidate codeword by emulating the verifier of C. While we could prove the latter claim directly, such a proof would have been quite complicated. Instead, we develop a general framework that allows us to give a cleaner formulation of this argument.

A general framework In order to prove that $(\text{DistAmp}(C_s))^2$ is a CWP, we develop a general framework for proving such claims. This framework allows us to give a simple and elegant proof of the argument sketched in this subsection, and may be of independent interest. We first define the composition of two codes to be the composition of their encoding functions, and define a non-standard notion of "repetition codes". We then examine the structure of $(\text{DistAmp}(C_s))^2$ and observe that it can be obtained by composing C with repetition codes and permuting its coordinates. Next, we identify some simple properties such that composing a code having those properties with a CWP yields a CWP. Finally, we show that repetition codes have the aforementioned properties and conclude that $(\text{DistAmp}(C_s))^2$ is a CWP.

For a simple example of how a code can be obtained using repetitions and permutations, consider the code C' of Section 3.2.3 (see Equation 1), and view its codewords as strings over the alphabet $\mathbb{F}$ (rather than $\mathbb{F}^{t+1}$). Observe that the codewords of C' can be obtained by duplicating the symbols of the codewords of C and permuting their coordinates. Although the latter example may seem trivial, the framework we develop in Section 5 allows expressing more complicated codes (e.g., $\text{DistAmp}(C)$ and C^2) as results of composing C with repetition codes.

3.3.3 The full construction

Suppose we wish to construct a CWP of message length k. As before, we start with a code C_0 of message length k_0, rate R and relative distance δ, and then define a sequence of codes $\{C_i\}_i$. But, this time we do *not* require that each C_i will be a CWP, but rather that each C_i^2 will be a CWP. In other words, we think of the i-th iteration as starting with the CWP C_i^2 and producing the CWP C_{i+1}^2. The code C_{i+1} is obtained from C_i as follows:

1. Let $C_i^{\text{TP}} \stackrel{\text{def}}{=} C_i^2$. The code $\left(C_i^{\text{TP}}\right)^2$ is a CWP with respect to the planes verifier (of Section 3.3.1).

2. Let C_i^{RP} be the result of applying a random projection to C_i^{TP}. The code $\left(C_i^{\text{RP}}\right)^2$ is a CWP since $\left(C_i^{\text{TP}}\right)^2$ is a CWP (see discussion at the beginning of Section 3.3.2).

3. Set C_{i+1} to be the result of applying the distance amplification to C_i^{RP}. The code C_{i+1}^2 is a CWP since $\left(C_i^{\text{RP}}\right)^2$ is a CWP (see Section 3.3.2).

We now have that $\left(C_{\log\log_{k_0} k-1}\right)^2$ is a CWP with message length k and with the parameters stated in Table 1.

Organization of the rest of this paper In Section 4, we analyze the three basic operations on CWPs described above, namely, the tensor product, random projection and distance amplification. In Section 5, we show that if a code $C = C_s^2$ is a CWP then $(\mathrm{DistAmp}(C_s))^2$ is a CWP with related parameters. The full construction (as outlined above) is then analyzed in Section 6. In addition, in Section 6 we also show how to improve the construction such that the CWPs have constant query complexity and constant rejection ratio, and derive LTCs with good parameters from those CWPs.

We stress that in the rest of this paper we measure the randomness complexity of the CWPs rather than their proof length. Thus, our goal is constructing CWPs that have randomness complexity $\log k + O(\log\log k)$, which corresponds to having proof length $k\,\mathrm{poly}\,(\log k)$.

4 Operations on CWPs

In this section we give formal statements and proofs of the properties of the tensor product, random projection and distance amplification operations.

4.1 Tensor Product

In this subsection we analyze the Tensor Product operation, which we use in order to increase the message length of our CWPs. Specifically, we prove the following result:

Theorem 4.1 (Tensor Product)**.** *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a code of rate R and relative distance $\delta \geq \sqrt[4]{7/8} + \frac{1}{n}$, such that C^2 is a (q, ε, r)-CWP. Then $C^4 \stackrel{\text{def}}{=} (C^2)^2$ is a (q, ε', r')-CWP for $\varepsilon' = 2^{-32} \cdot \varepsilon$ and*

$$r' = r + \log\left[\binom{4}{2} n^2\right] = r + \log\left[6 \cdot \frac{k^2}{R^2}\right] = r + \log k^2 + O(\log\frac{1}{R})$$

Remark 4.2. Note that C^2 has message length k^2, that C^4 has message length k^4 and that $(r' - \log k^4) = (r - \log k^2) + \log\frac{6}{R^2}$. That is, the tensor product operation increases the difference between the randomness complexity of the CWP and the logarithm of its message length by an additive term of $\log\frac{6}{R^2}$. This is analogous to multiplying the proof rate by a factor of $\frac{R^2}{6}$, as described in Section 3.3.1.

Let C be as in the theorem. Before we prove Theorem 4.1, we first discuss the structure of $C^4 \stackrel{\text{def}}{=} (C^2)^2$. We then define the proof strings and verifier of C^4 and analyze them.

4.1.1 The structure of C^4

Note that the code C^4 has block length n^4. We identify the set of coordinates $[n^4]$ with the 4-dimensional hypercube $[n]^4$. We say that a subset $L \subseteq [n]^4$ is an axis parallel line of the hypercube if there exist $i_1 \neq i_2 \neq i_3 \in [4]$ and $a_1, a_2, a_3 \in [n]$ such that $L = \left\{x \in [n]^4 : x_{i_1} = a_1, x_{i_2} = a_2, x_{i_3} = a_3\right\}$. If $i_4 \in [4] \setminus \{i_1, i_2, i_3\}$, we say that L is parallel to the i_4-th axis. Similarly, we say that a subset $P \subseteq [n]^4$ is an axis parallel plane of the hypercube if there exist $i_1 \neq i_2 \in [4]$ and $a_1, a_2 \in [n]$ such that $P = \left\{x \in [n]^4 : x_{i_1} = a_1, x_{i_2} = a_2\right\}$. If $i_3 \in [4] \setminus \{i_1, i_2\}$, we say that P is parallel to the i_3-th axis.

Observation Let c be a codeword of $C^4 = (C^2)^2$, and recall that when c is viewed as a $n^2 \times n^2$ matrix, all the rows and columns of c are codewords of C^2. Observe that when taking the view of the 4-dimensional hypercube, every row (respectively, column) of the matrix c corresponds to an axis parallel plane that is parallel to the third and fourth axis (respectively, the first and second axis).

We can now prove the following facts.

Fact 4.3. *Let $c \in C^4$. Then for every axis parallel line $L \subseteq [n]^4$ it holds that $c_{|L} \in C$.*

Proof We prove the fact only for lines that are parallel to the first axis, and the proof for the other axes is similar. By the definition of L, there exist $a_2, a_3, a_4 \in [n]$ such that $L = \left\{x \in [n]^4 : x_2 = a_2, x_3 = a_2, x_4 = a_4\right\}$. Let $P = \left\{x \in [n]^4 : x_3 = a_3, x_4 = a_4\right\}$ and note that P is an axis parallel plane that contains L and is parallel to the first and second axis. Now, observe that if we view c as an $n^2 \times n^2$ matrix, then the coordinates in P form a column of c, and therefore $c_{|P} \in C^2$. Furthermore, if we view $c_{|P}$ as an $n \times n$ matrix, then the coordinates in L form a column of $c_{|P}$, and are thus a codeword of C. ■

The verifier of C^4 is motivated by the following fact.

Fact 4.4. *A string $w \in \mathbb{F}^{n^4}$ is a codeword of C^4 if and only if for every axis parallel plane P it holds that $w_{|P} \in C^2$.*

Proof For one direction, let $w \in \mathbb{F}^{n^4}$ be a string such that for every axis parallel plane P it holds that $w_{|P} \in C^2$. View w as an $n^2 \times n^2$ matrix and note that each row of w can be viewed as a restriction of w to an axis parallel plane. Therefore, every row of w is a codeword of C^2. Similarly, every column of w is a codeword of C^2. It follows that $w \in C^4 = (C^2)^2$.

For the other direction, let $c \in C^4$ and let $P \subseteq [n]^4$ be an arbitrary axis parallel plane. View $c_{|P}$ as an $n \times n$ matrix. Observe that the rows and columns of $c_{|P}$ are axis parallel lines of the hypercube and therefore by Fact 4.3 they are codewords of C. It follows that $c_{|P} \in C^2$, as required. ■

4.1.2 The proof strings and verifier of C^4

The proof strings of C^4 Let $c \in C^4$. The proof string π_c of c consists of a collection of proof strings of C^2: For every axis parallel plane P, the proof string π_c includes a proof string $\pi_{c_{|P}}$ that proves that $c_{|P}$ is a codeword of C^2. That is, the proof string for $c \in C^4$ consists of $\binom{4}{2} \cdot n^2$ proof strings for codewords of C^2.

The verifier of C^4 Let V denote the verifier of C^2. The verifier V' of C^4 is defined as follows. Given oracle access to a tested string w and proof string π, the verifier V' first partitions π into parts such that each part corresponds to some axis parallel plane. The part of π that corresponds to the axis parallel plane P, denoted $\pi_{|P}$, is assumed to contain a proof that $w_{|P} \in C^2$. The verifier V' then chooses an axis parallel plane P uniformly at random and invokes V, with oracle access to $w_{|P}$ and $\pi_{|P}$. The verifier V' accepts if and only if V accepts.

Remark 4.5. We stress the difference between the planes verifier used in this section and the row/column verifier used in the simplified construction of Section 3.2. The row/column verifier views a tested string $w \in \mathbb{F}^{n^4}$as a $n^2 \times n^2$ matrix and tests that random row/column is a codeword of C^2. Taking the hypercubes view, every row/column corresponds to some axis parallel plane, but some axis parallel planes do not correspond to any row or column of the $n^2 \times n^2$ matrix. Thus, the planes verifier uses more possible tests than the row/column verifier, and this fact is crucial to the proof of Theorem 4.6 due to [BS04] that is stated below.

4.1.3 The parameters of C^4

The query complexity of V' equals that of V. As for the randomness complexity of V', the verifier V' tosses $\log\left[\binom{4}{2}n^2\right]$ coins in order to select a random axis parallel plane, and then tosses r coins in order to emulate V. It follows that the randomness complexity of V' is $r + \log\left[\binom{4}{2}n^2\right]$, as claimed in Theorem 4.1.

We turn to analyze to rejection ratio of V'. In order to do it, we need a theorem of [BS04]. Let $\mathcal{P}$ denote the set of axis parallel planes in $[n]^4$. For any $w \in \mathbb{F}^{n^4}$, we denote

$$\rho(w) \stackrel{\text{def}}{=} \frac{1}{|\mathcal{P}|}\sum_{P\in\mathcal{P}} \delta_{C^2}(w_{|P})$$

That is, $\rho(w)$ is the average relative distance of $w_{|P}$ to C^2, taken over all axis parallel planes $P \in \mathcal{P}$. We can now state the result we need:

Theorem 4.6 ([BS04, Lemma 4.2]). *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a code with relative distance δ such that $\left(\delta - \frac{1}{n}\right)^4 \geq \frac{7}{8}$. Then, for every string $w \in \mathbb{F}^{n^4}$, it holds that $\rho(w) \geq 2^{-32} \cdot \delta_{C^4}(w)$.*

Note that the hypothesis of Theorem 4.1 matches the distance requirement of Theorem 4.6. Now, for every string $w \in \mathbb{F}^{n^4}$ and every proof string π we have that

$$\begin{aligned}
\Pr\left[V' \text{ rejects } w \text{ and } \pi\right] &= \frac{1}{|\mathcal{P}|}\sum_{P\in\mathcal{P}} \Pr\left[V' \text{ rejects } w_{|P} \text{ and } \pi_{|P}\right] \\
&\geq \frac{1}{|\mathcal{P}|}\sum_{P\in\mathcal{P}} \varepsilon \cdot \delta_{C^2}(w_{|P}) \\
&= \varepsilon \cdot \rho(w) \\
&\geq 2^{-32} \cdot \varepsilon \cdot \delta_{C^4}(w)
\end{aligned}$$

where the first inequality is due to the rejection ratio of V and the second inequality is due to Theorem 4.6. This proves the rejection ratio of V', and Theorem 4.1 follows.

4.2 Random Projection

In this subsection we analyze the Random Projection operation - the projection of a code to a random subset of its coordinates. Recall that we use this operation in order to increase the rate of our codes. We prove the following result:

Theorem 4.7 (Random Projection)**.** *For every $\delta > 0$ there exists a constant R_δ such that the following holds. Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a code with rate $R \leq R_\delta$ and relative distance δ. Then, with probability at least $1 - \exp(-\Omega(k))$ over the uniform selection of a set $S \subseteq [n]$ of size $\frac{1}{R_\delta}k$, the function $C_{|S}$ is a code with rate R_δ and relative distance $\frac{1}{2}\delta$. In such a case, if C^2 is a (q, ε, r)-CWP, then $(C_{|S})^2$ is a (q, ε', r)-CWP for $\varepsilon' = \frac{R^2}{R_\delta^2} \cdot \varepsilon$.*

In Section 4.2.1 we show how to choose R_δ and prove that, with probability $1-\exp(-\Omega(k))$, the function $C_{|S}$ is a code with relative distance $\frac{1}{2}\delta$. In Section 4.2.2 we prove that $(C_{|S})^2$ is a CWP.

Remark 4.8. Theorem 4.7 states that $C_{|S}$ has relative distance $\frac{1}{2}\delta$. Actually, we could prove that $C_{|S}$ has relative distance $\alpha \cdot \delta$ for any $\alpha < 1$. Such a change would have affected the way the constant R_δ is chosen and the constants in the Big-Ω notation of the success probability.

4.2.1 The relative distance of $C_{|S}$

The proof that $C_{|S}$ has relative distance $\frac{1}{2}\delta$ uses a standard probabilistic argument. Fix a non-zero codeword $c \in C$, and let $S \subseteq [n]$ be a uniformly chosen set of size $\frac{1}{R_\delta}k$ for a constant R_δ that will be determined later. The relative weight of c is at least δ, and therefore the *expected* relative weight of $c_{|S}$ is at least δ. Applying the Chernoff Bound, it follows that the probability that the relative weight of $c_{|S}$ is less than $\frac{1}{2}\delta$ is at most $2\exp\left(-\frac{1}{4} \cdot \delta^2 \cdot |S|\right)$. By taking a union bound over all the codewords of C, the probability that there exists a non-zero codeword $c \in C$ such that $c_{|S}$ has relative weight less than $\frac{1}{2} \cdot \delta$ is bounded by

$$|\mathbb{F}|^k \cdot 2\exp\left(-\frac{\delta^2}{4} \cdot |S|\right) = 2\exp\left(\left(\ln|\mathbb{F}| - \frac{\delta^2}{4R_\delta}\right) \cdot k\right)$$

Now, by choosing R_δ small enough such that $\delta^2/(4 \cdot R_\delta) > \ln|\mathbb{F}|$, we get that the probability that the relative distance of $C_{|S}$ is less than $\delta/2$ is bounded by $2\exp(-\Omega(k))$, as required.

4.2.2 $(C_{|S})^2$ is a CWP

Let $S \subseteq [n]$ be such that $C_{|S}$ is one-to-one. Below we prove that if C^2 is a CWP then so is $(C_{|S})^2$. Assume that C^2 is a (q, ε, r)-CWP. We define the proof strings and verifier of $(C_{|S})^2$ and analyze them, using the following observation.

Observation Let $S^2 \stackrel{\text{def}}{=} S \times S$. Then $(C_{|S})^2 = C^2_{|S^2}$.

The proof strings of $(C_{|S})^2$ For every codeword $c \in C^2$ with proof string π_c, the proof string of the corresponding codeword $c_{|S^2} \in (C_{|S})^2$ consists of $c_{[n^2]\setminus S^2}$ and π_c.

The verifier of $(C_{|S})^2$ Let V denote the verifier of C^2. The verifier V' of $(C_{|S})^2$ is almost identical to V, except that whenever V queries the tested string at a coordinate in $[n^2]\setminus S^2$, the verifier V' queries the corresponding coordinate in the proof string.

The parameters of $(C_{|S})^2$ Clearly, V' maintains the query complexity and randomness complexity of V. As for the rejection ratio, suppose we have a string $w \in \mathbb{F}^{|S|^2}$ that is α-far from $(C_{|S})^2$. This means that w must disagree with any codeword of $(C_{|S})^2$ on at least $\alpha \cdot |S|^2$ coordinates, which implies that every string $u \in \mathbb{F}^{n^2}$ that satisfies $u_{|S^2} = w$ must disagree with any codeword of C^2 on at least $\alpha \cdot |S|^2$ coordinates. Therefore, for every such u it holds that $\delta_{C^2}(u) \geq \frac{|S|^2}{n^2} \cdot \alpha$, and thus V' rejects w with probability at least $\varepsilon \cdot \frac{|S|^2}{n^2} \cdot \alpha$. Using $n = k/R$ and $|S| = k/R_\delta$, it follows that the rejection ratio of $(C_{|S})^2$ is at least $\varepsilon' = \frac{|S|^2}{n^2} \cdot \varepsilon = \frac{R^2}{R_\delta^2} \cdot \varepsilon$, as claimed in Theorem 4.7.

4.3 Distance Amplification

In this subsection we define and analyze the Distance Amplification operation, which we use in order to increase the relative distance of our codes. For motivation to the following definitions, we refer the reader to the discussion in Section 3.2.3. We prove the following theorem:

Theorem 4.9 (Distance Amplification)**.** *For all constants $\delta < \delta_0 < \frac{|\mathbb{F}|-1}{|\mathbb{F}|}$ there exists a transformation on codes $\mathrm{DistAmp}_{\delta \to \delta_0}$ that acts as follows: Given a code C with rate R and relative distance δ, the code $\mathrm{DistAmp}_{\delta \to \delta_0}(C)$ has relative distance δ_0 and rate $\Omega(R)$. Furthermore, if C^2 is a (q, ε, r)-CWP then $\left(\mathrm{DistAmp}_{\delta \to \delta_0}(C)\right)^2$ is a (q', ε', r')-CWP for $q' = O(q)$, $\varepsilon' = \Omega(\varepsilon)$ and $r' = \max\{r, \log n^2\} + O(1)$. The constants inside all the Big-O/Ω notations depend only on δ and δ_0.*

Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a code with rate R and relative distance δ. Let G be a d-regular expander on n vertices with normalized second eigenvalue $\lambda < \delta$. We identify the vertices of G with the set of coordinates $[n]$. Let t be an arbitrary natural number. We first define the following code $\mathrm{Walk}_{G,t}(C) : \mathbb{F}^k \to (\mathbb{F}^{t+1})^{d^t n}$:

Definition 4.10. For any message $x \in \mathbb{F}^k$, the codeword $\mathrm{Walk}_{G,t}(C)(x) \in (\mathbb{F}^{t+1})^{d^t n}$ is defined as follows. We identify the $d^t n$ coordinates of $\mathrm{Walk}_{G,t}(C)(x)$ with the walks of length t on G. For every walk $\bar{i} = (i_0, \ldots, i_t)$ on G (where $i_0, \ldots, i_t \in [n]$ are viewed both as vertices of G and as coordinates of $C(x)$) we define

$$\mathrm{Walk}_{G,t}(C)(x)_{\bar{i}} = \left(C(x)_{i_0}, C(x)_{i_1}, \ldots, C(x)_{i_t}\right) \in \mathbb{F}^{t+1}$$

$\mathrm{Walk}_{G,t}(C)$ maps messages of length $k = R \cdot n$ over $\mathbb{F}$ to messages of length $d^t n$ over $\mathbb{F}^{t+1}$, and therefore has rate of $\frac{1}{d^t(t+1)} \cdot R$. Using standard properties of expanders (see, e.g., [HLW06]), it can be seen that $\mathrm{Walk}_{G,t}(C)$ has relative distance $1 - (1 - \delta + \lambda)^t$. Note that $\mathrm{Walk}_{G,t}(C)$ is a $\mathbb{F}$-linear code.

Let I be a code over $\mathbb{F}$ with message length $t + 1$, rate R_I and relative distance δ_I. We use concatenation with the inner code I to decrease the alphabet from $\mathbb{F}^{t+1}$ back to $\mathbb{F}$.

Definition 4.11. The code $\mathrm{DistAmp}_{G,t,I}(C)$ is defined to be the concatenation of $\mathrm{Walk}_{G,t}(C)$ with the *inner* code I.

Clearly, $\mathrm{DistAmp}_{G,t,I}(C)$ has relative distance $\delta_I \cdot \left[1-(1-\delta+\lambda)^t\right]$ and rate $R_I \cdot \frac{1}{d^t(t+1)} \cdot R$. Furthermore, $\mathrm{DistAmp}_{G,t,I}(C)$ is a linear code. In Section 5 we prove the following theorem:

Theorem 4.12. *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a code such that C^2 is a (q,ε,r)-CWP, let G be a d-regular graph on n vertices, let t be any constant integer and let $I : \mathbb{F}^{t+1} \to \mathbb{F}^{n_I}$ be a code. Then $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$ is a (q',ε',r')-CWP where $q' = O(q)$, $\varepsilon' = \Omega(\varepsilon)$ and $r' = \max\{r, \log n^2\} + O(1)$, and where the constants in the Big-O/Ω notations depend only on d,t and n_I.*

We are now ready to prove Theorem 4.9.

Proof of Theorem 4.9 Let $\delta < \delta_0 < \frac{|\mathbb{F}|-1}{|\mathbb{F}|}$ be two constants. Let λ be some constant strictly smaller than δ and let G be any explicitly constructable constant degree expander with degree d and normalized second eigenvalue at most λ. Let I be a code over $\mathbb{F}$ with message length $t+1$, rate R_I and relative distance $\delta_I > \delta_0$ - Note that such a code must exist, since one can set I to be the $|\mathbb{F}|$-ary Hadamard code (see Definition 6.19), which has relative distance $\frac{|\mathbb{F}|-1}{|\mathbb{F}|} > \delta_0$, or alternatively use the Zybalov Bound (see Appendix A, Theorem A.4). Let t be the minimal integer such that $\delta_I \cdot \left[1-(1-\delta+\lambda)^t\right] \geq \delta_0$. Observe that t is a constant, since $1-\delta+\lambda$ is a positive constant that is strictly less than 1 and since δ_I is a constant larger than δ_0.

Now, let C be any code with rate R and relative distance δ. Define $\mathrm{DistAmp}_{\delta\to\delta_0}(C) \stackrel{\text{def}}{=} \mathrm{DistAmp}_{G,t,I}(C)$. Then $\mathrm{DistAmp}_{\delta\to\delta_0}(C)$ has relative distance δ_0 and rate $R_I \cdot \frac{1}{d^t(t+1)} \cdot R$. In particular, note that d and t depend only on δ and δ_0, so $\mathrm{DistAmp}_{\delta\to\delta_0}(C)$ has rate $\Omega(R)$ where the constants in the Big-Ω notation depend only on δ and δ_0, as required. The fact that $\left(\mathrm{DistAmp}_{\delta\to\delta_0}(C)\right)^2$ is a CWP with the required parameters follows immediately from Theorem 4.12. ■

5 Distance amplification preserves local testability of the square

In this section we prove Theorem 4.12.

Theorem (Theorem 4.12, restated)**.** *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a code such that C^2 is a (q,ε,r)-CWP, let G be a d-regular graph on n vertices, let t be any constant integer and let $I : \mathbb{F}^{t+1} \to \mathbb{F}^{n_I}$ be a code. Then $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$ is a (q',ε',r')-CWP where $q' = O(q)$, $\varepsilon' = \Omega(\varepsilon)$ and $r' = \max\{r, \log n^2\} + O(1)$, and where the constants in the Big-O/Ω notations depend only on d,t and n_I.*

Our proof strategy is based on examining the structure of $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$ and observing that it can be obtained from C^2 by "composing" it with certain "repeated codes" (see Section 5.1). We then show that composing a CWP with a repeated code yields a CWP (see Section 5.2) and conclude that $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$ is a CWP (see Section 5.3).

5.1 The structure of $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$

5.1.1 Repetitions, Compositions and Permutations of Codes

In this subsection we define structural properties of codes that will allow us to describe in a more convenient form the structure of $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$.

Definition 5.1. Let $C_1 : \mathbb{F}^k \to \mathbb{F}^n$ be a code. We define the ℓ-repeated code $C_1^{[\ell]} : \mathbb{F}^{\ell k} \to \mathbb{F}^{\ell n}$ as the code that results from partitioning the message to ℓ blocks of length k and encoding each of them by C_1. That is, $C_1^{[\ell]}$ encodes each message $x \in \mathbb{F}^{\ell k}$ by

$$C_1^{[\ell]}(x) = C_1(x_1 \dots x_k)\, C_1(x_{k+1} \dots x_{2k}) \dots C_1\left(x_{(\ell-1)k+1} \dots x_{\ell k}\right)$$

Remark 5.2. We stress that repeated codes are a different notion from the common notion of "repetition codes", which are codes that encode a message by repeating it a number of times.

Definition 5.3. Let $C_1 : \mathbb{F}^k \to \mathbb{F}^n$ and $C_2 : \mathbb{F}^n \to \mathbb{F}^{n'}$ be codes. We define the composition $C_2 \circ C_1$ to be the composition of C_1 and C_2 as functions. That is, $C_2 \circ C_1$ is the code that encodes a message by encoding it with C_1 and then encoding the result with C_2.

Definition 5.4. Let σ be a permutation over $[k]$. We define the permutation code $P_\sigma : \mathbb{F}^k \to \mathbb{F}^k$ as the "code" that permutes the coordinates of its message according to σ. That is, for every $x \in \mathbb{F}^k$ we have that

$$P_\sigma(x) = x_{\sigma(1)} x_{\sigma(2)} \dots x_{\sigma(k)}$$

We now demonstrate two useful connections between the forgoing notions and the tensor product operation. In particular, the following proposition will allow us to simplify the analysis of tensor products of compositions of codes.

Proposition 5.5. *Let $C_1 : \mathbb{F}^k \to \mathbb{F}^n$, $C_2 : \mathbb{F}^n \to \mathbb{F}^{n'}$ be codes. Then $(C_2 \circ C_1)^2 = C_2^2 \circ C_1^2$.*

Proof Note that both $(C_2 \circ C_1)^2$ and $C_2^2 \circ C_1^2$ have message length k^2. We show that for every message $M \in \mathbb{F}^{k^2}$, viewed as a $k \times k$ matrix over $\mathbb{F}$, it holds that $C_2^2(C_1^2(M)) = (C_2 \circ C_1)^2(M)$. Let G_1 and G_2 denote the generating matrices of C_1 and C_2, respectively. Note that $C_1^2(M) = G_1^T \cdot M \cdot G_1$, and similarly that

$$C_2^2(C_1^2(M)) = G_2^T \cdot \left(G_1^T M G_1\right) \cdot G_2$$

Furthermore, the generating matrix of $C_2 \circ C_1$ is $G_1 G_2$ and so $(C_2 \circ C_1)^2(M) = (G_1 G_2)^T \cdot M \cdot (G_1 G_2)$. Thus

$$C_2^2(C_1^2(M)) = G_2^T \cdot \left(G_1^T \cdot M \cdot G_1\right) \cdot G_2 = (G_1 \cdot G_2)^T \cdot M \cdot (G_1 \cdot G_2) = (C_2 \circ C_1)^2(M)$$

as required. ■

Recall that the code C^2 encodes a $k \times k$ matrix by first encoding each of its rows with C, and then encoding each of the resulting columns with C. The next proposition states this fact in terms of compositions, repeated codes, and permutation codes. The proposition

will be used to simplify the presentation of tensor product codes. For this proposition, we assume that an $m \times \ell$ matrix over $\mathbb{F}$ is represented as a vector in $\mathbb{F}^{m \cdot \ell}$ by the concatenation of its rows, that is, the first ℓ elements of the vector are the matrix first row, the next ℓ elements are the second row, and so on.

Proposition 5.6. *Let $C_1 : \mathbb{F}^k \to \mathbb{F}^n$ be a code. There exist permutations σ_1 and σ_2 over $[kn]$ and $[n^2]$ respectively such that $C_1^2 = P_{\sigma_2} \circ C_1^{[n]} \circ P_{\sigma_1} \circ C_1^{[k]}$.*

Proof Let σ_1 be a permutation on $[kn]$ such that given a vector in $\mathbb{F}^{kn}$, which represents a $k \times n$ matrix M, permutes the coordinates of the vector such that it represents the transpose matrix M^T. Let σ_2 be a similar permutation over $[n^2]$ for $n \times n$ matrices. Let M be a $k \times k$ matrix represented as a vector in $\mathbb{F}^{k^2}$. Then $C_1^{[k]}(M)$ is the result of encoding each of the rows of M with C_1, and $\left(P_{\sigma_2} \circ C_1^{[n]} \circ P_{\sigma_1}\right)\left(C_1^{[k]}(M)\right)$ is the result of encoding the columns of $C_1^{[k]}(M)$ with C_1. It follows that $C_1^2(M) = \left(P_{\sigma_2} \circ C_1^{[n]} \circ P_{\sigma_1} \circ C^{[k]}\right)(M)$. ■

Remark 5.7. Note that Lemma 5.6 suggests a simple algorithm for computing the generating matrix of C_1^2. Specifically, given the generating matrix of C, we can compute the generating matrix of C_1^2, by first computing the generating matrices of $C_1^{[k]}$, P_{σ_1}, $C_1^{[n]}$ and P_{σ_2}, and then outputting their product.

5.1.2 Obtaining $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$ from C^2

Let C, G, t and I be as in Theorem 4.12. Let us view the construction of $\mathrm{DistAmp}_{G,t,I}(C)$ in a different way than the way it was defined in Section 4.3. Specifically, the encoding of a message $x \in \mathbb{F}^k$ using $\mathrm{DistAmp}_{G,t,I}(C)$ can be done by the following steps:

1. Encode $x \in \mathbb{F}^k$ with C, resulting in a codeword $c^1 \in \mathbb{F}^n$.

2. Duplicate each symbol of c^1 to create $(t+1)d^t$ consecutive copies of the symbol, and denote the result by $c^2 \in \mathbb{F}^{(t+1)d^t n}$. In other words, if we denote by $R : \mathbb{F} \to \mathbb{F}^{(t+1)d^t}$ the repetition code that duplicates a single symbol $(t+1)d^t$ times, then we have that $c^2 = R^{[n]}(c^1)$. The number $(t+1)d^t$ was chosen because it is the number of times a vertex of G appears in walks of length t over G.

3. Permute the coordinates of c^2 such that every $t+1$ consecutive non-overlapping symbols form a walk of length t on G, and denote the result by $c^3 \in \mathbb{F}^{(t+1)d^t n}$.
   More specifically, define $c_3 = P_\sigma(c^2)$, where σ is a permutation over $[(t+1)d^t n]$ such that for every walk $\bar{i} = (i_0, \ldots, i_t)$ on G (where $i_0, \ldots, i_t \in [n]$) there exists $0 \leq j < d^t n$ such that
   $$c^3_{(t+1)\cdot j+1} c^3_{(t+1)\cdot j+2} \cdots c^3_{(t+1)\cdot j+(t+1)} = c^1_{i_0} c^1_{i_1} \cdots c^1_{i_t}$$

4. Finally, encode every $t+1$ consecutive symbols of c^3 with $I : \mathbb{F}^{t+1} \to \mathbb{F}^{n_I}$, and set $\mathrm{DistAmp}_{G,t,I}(C)(x)$ to be the result. That is, $\mathrm{DistAmp}_{G,t,I}(C)(x) = I^{[d^t n]}(x) \in \mathbb{F}^{n_I \cdot d^t n}$.

Thus, $\mathrm{DistAmp}_{G,t,I}(C)$ can be obtained as $I^{\left[d^t n\right]} \circ P_\sigma \circ R^{[n]} \circ C$. We turn to examine $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$. By Proposition 5.5, it follows that we can obtain $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$ from C^2 by

$$\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2 = \left(I^{\left[d^t n\right]}\right)^2 \circ P_\sigma^2 \circ \left(R^{[n]}\right)^2 \circ C^2 \tag{2}$$

Note that P_σ^2 is also a permutation code. Using Proposition 5.6, we can remove the tensor products $\left(R^{[n]}\right)^2$ and $\left(I^{\left[d^t n\right]}\right)^2$ from Equality (2), thereby obtaining codes that are easier to analyze. Specifically, applying Proposition 5.6, there exist permutations $\phi_1, \phi_2, \phi_3, \phi_4$ such that

$$\begin{aligned}\left(R^{[n]}\right)^2 &= P_{\phi_2} \circ \left(R^{[n]}\right)^{\left[(t+1)d^t n\right]} \circ P_{\phi_1} \circ \left(R^{[n]}\right)^{[n]} \\ &= P_{\phi_2} \circ R^{\left[(t+1)d^t n^2\right]} \circ P_{\phi_1} \circ R^{[n^2]}\end{aligned}$$

$$\begin{aligned}\left(I^{\left[d^t n\right]}\right)^2 &= P_{\phi_4} \circ \left(I^{\left[d^t n\right]}\right)^{\left[n_I \cdot d^t n\right]} \circ P_{\phi_3} \circ \left(I^{\left[d^t n\right]}\right)^{\left[(t+1)d^t n\right]} \\ &= P_{\phi_4} \circ I^{\left[n_I \cdot \left(d^t n\right)^2\right]} \circ P_{\phi_3} \circ I^{\left[(t+1)\left(d^t n\right)^2\right]}\end{aligned}$$

By substituting the above equalities in Equality (2), we obtain the following important characterization of $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$:

Proposition 5.8. *There exist permutations $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ such that*

$$\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2 = P_{\sigma_4} \circ I^{\left[n_I\left(d^t n\right)^2\right]} \circ P_{\sigma_3} \circ I^{\left[(t+1)\left(d^t n\right)^2\right]} \circ P_{\sigma_2} \circ R^{\left[(t+1)d^t n^2\right]} \circ P_{\sigma_1} \circ R^{[n^2]} \circ C^2$$

5.2 Composition preserves local testability

Proposition 5.8 implies that in order to prove that $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$ is a CWP, it suffices to study the preservation of local testability under composition with repeated codes. Our goal in this section is showing that the composition with repeated codes preserves local testability, that is, the composition of a CWP with a repeated code results in a CWP. In Section 5.2.1 we identify properties of codes that are sufficient for the preservation of local testability under composition. In Section 5.2.2, we show that repeated codes enjoy the aforementioned properties and conclude that the composition with repeated codes preserves local testability.

5.2.1 Properties sufficient to make composition preserve local testability

Definition 5.9. A code $C_1 : \mathbb{F}^k \to \mathbb{F}^n$ is said to be λ-proximity preserving if for every two strings $x, y \in \mathbb{F}^k$ we have that

$$\delta\left(C_1(x), C_1(y)\right) \leq \lambda \cdot \delta\left(x, y\right)$$

The property of "proximity preservation" is non-standard in coding theory, since it conflicts with having good relative distance. In particular, a λ-proximity preserving code can have relative distance at most λ/n. Nevertheless, proximity preserving codes will be useful in our analysis. For example, note that permutation codes are 1-proximity preserving.

Recall that in the definitions of LTCs and CWPs (Definitions 2.2 and 2.6), the rejection property of CWPs was stronger than the one of LTCs. In this section we will make use of LTCs that enjoy the strong rejection property of CWPs. Such LTCs are usually referred to as "strong LTCs" (see, e.g., [GS02, Section 2.1]), and are a special case of CWPs in which the proof string is always empty. We give a more detailed definition of strong LTCs below.

Definition 5.10. A code $C_1 : \mathbb{F}^k \to \mathbb{F}^n$ is said to be (q, ε, r)-strong locally testable if there exists a probabilistic oracle circuit V such that

1. V makes at most q non-adaptive queries to its oracle and tosses at most r coins.
2. For every codeword $c \in C_1$, we have that $\Pr[V^c \text{ accepts}] = 1$.
3. For every string $w \in \mathbb{F}^n$, we have that $\Pr[V^w \text{ rejects}] \geq \varepsilon \cdot \delta_C(w)$.

Recall that in order to argue that if C is a CWP then so is $\mathrm{DistAmp}_{G,t,I}(C)$ (in Section 3.2.3), we claimed that the verifier of $\mathrm{DistAmp}_{G,t,I}(C)$ can emulate the verifier of C. We now define the exact property we use in the emulation of verifiers. We comment that this is a *non-standard* notion.

Definition 5.11. A code $C_1 : \mathbb{F}^k \to \mathbb{F}^n$ is said to be (q, ε, r)-locally accessible if there exists a probabilistic oracle circuit A such that

1. A receives as input an index $i \in [k]$. A either outputs a symbol or rejects.
2. A makes at most q non-adaptive queries to its oracle and tosses at most r coins.
3. For every message $x \in \mathbb{F}^k$ and every $i \in [k]$, we have that $\Pr\left[A^{C_1(x)}(i) = x_i\right] = 1$.
4. For every string $w \in \mathbb{F}^n$, every $i \in [k]$ and every message x such that $\delta(w, C_1(x)) = \delta_C(w)$, we have that

$$\Pr[A^w(i) \neq x_i \text{ and } A^w(i) \text{ has not rejected}] \leq 1 - \varepsilon$$

   That is, A may reject with any probability, but if it did not reject, then the probability that it outputs a wrong symbol is small (where "wrong" means different than x_i).

We refer to A as the accessor of C_1, and to q, ε and r as the query complexity, success probability and randomness complexity of C_1, respectively.

Locally Accessible Codes versus Locally Decodable Codes The notion of Locally Accessible Codes (LACs) is a very weak variant of the notion of Locally Decodable Codes (LDCs) formalized in Katz and Trevisan [KT00]. The main difference between LACs and LDCs is that the definition of an LAC does not require the code to have a good relative distance, while a good relative distance is implicit in the definition of LDCs. This implies that an LAC may have no error correction capability - for example, it is easy to see that the identity map is an LAC. This difference makes LACs a seemingly weaker notion than LDCs, and indeed they are much easier to construct.

We do note, however, that LACs are stronger than LDCs in one sense: The accessor of an LAC is required to have a low error probability when given access to *any* oracle, while the decoder of an LDC is required to have a low error probability *only* when given access to oracles that are close to the code. LACs can achieve this stronger property by using their ability to reject whenever they are given access to an oracle that is too corrupted. The latter feature is related to the notion of LTCs, and indeed, it can be shown that a code that is both a strong LTC and a LDC is also a LAC.

We mention that the definition of LACs is reminiscent of the definition of relaxed LDCs due to [BGHSV06]. However, relaxed LDCs, just as LDCs, always have good relative distance, and should therefore be stronger than LACs. In particular, the decoder of a relaxed LDC is only allowed to reject a constant fraction of the indices $i \in [k]$, while the decoder of an LAC may reject every index with probability 1.

We are now ready to prove the following "composition theorem":

Theorem 5.12. *Let $C_1 : \mathbb{F}^k \to \mathbb{F}^n$ be a $(q_1, \varepsilon_1, r_1)$-CWP. Let $C_2 : \mathbb{F}^n \to \mathbb{F}^{n'}$ be a λ-proximity preserving, $\left(q_2^T, \varepsilon_2^T, r_2^T\right)$-strong locally testable and $\left(q_2^A, \varepsilon_2^A, r_2^A\right)$-locally accessible code. Then $C_2 \circ C_1$ is a CWP with the following parameters:*

1. *Query complexity* $\max\left\{q_1 \cdot q_2^A, q_2^T\right\}$.
2. *Rejection ratio* $\min\left\{\frac{1}{4\lambda}\left(\varepsilon_2^A\right)^{q_1} \cdot \varepsilon_1, \frac{1}{4}\varepsilon_2^T\right\}$.
3. *Randomness complexity* $1 + \max\left\{r_1 + q_1 \cdot r_2^A, r_2^T\right\}$.

Proof We define the proof strings and verifier of $C_2 \circ C_1$ and prove they satisfy the requirements of Theorem 5.12.

- **The proof strings of $C_2 \circ C_1$:** The proof strings of $C_2 \circ C_1$ are the same as those of C_1. That is, for every message $x \in \mathbb{F}^{k_1}$, the proof $\pi_{(C_2 \circ C_1)(x)}$ of $(C_2 \circ C_1)(x)$ equals the proof $\pi_{C_1(x)}$ of $C_1(x)$.
- **The verifier of $C_2 \circ C_1$:** Let V_1 and V_2 be the verifiers of C_1 and C_2, and let A be the accessor of C_2. We define the verifier V of $C_2 \circ C_1$ as follows. Suppose V is given oracle access to a tested string $w \in \mathbb{F}^{n'}$ and a proof string π. With probability $\frac{1}{2}$, the verifier V invokes V_2^w to verify the membership of w in C_2. With probability $\frac{1}{2}$, the verifier V emulates V_1. Whenever V_1 queries a symbol of the tested string (say, the i-th symbol), V invokes $A^w(i)$. If $A^w(i)$ rejects, then V rejects, and otherwise V answers the query of V_1 with the output of $A^w(i)$. Whenever V_1 queries a symbol of the proof, V simply queries this symbol from π and feed it to V_1. Finally, V accepts if and only if V_1 accepts.

The query complexity and randomness complexity of $C_2 \circ C_1$ are obvious from the description of V . We turn to analyze its rejection ratio. The idea of the analysis is that if a string $w \in \mathbb{F}^{n'}$ is far from $C_2 \circ C_1$, then w is either far from C_2, or close to C_2. In the first case, the invocation of V_2 "catches" w and thus V rejects with high probability. In the second case, we decode w by C_2, thereby obtaining a string $x \in \mathbb{F}^n$, and use the fact that C_2 is proximity

preserving to show that x is far from C_1. We then claim that the emulation of V_1 "catches" x and therefore V rejects with high probability. Details follow.

Let $w \in \mathbb{F}^{n'}$. Suppose that $\delta_{C_2}(w) > \frac{1}{2} \cdot \delta_{C_2 \circ C_1}(w)$. In this case, with probability $\frac{1}{2}$, the verifier V invokes V_2^w, and then V_2^w rejects with probability at least $\varepsilon_2^T \cdot \delta_{C_2}(w) > \varepsilon_2^T \cdot \frac{1}{2} \cdot \delta_{C_2 \circ C_1}(w)$. Thus, in this case, V rejects with probability at least $\frac{1}{4} \cdot \varepsilon_2^T \cdot \delta_{C_2 \circ C_1}(w)$.

On the other hand, suppose that $\delta_{C_2}(w) \leq \frac{1}{2}\delta_{C_2 \circ C_1}(w)$. Let $x \in \mathbb{F}^n$ be such that $\delta_{C_2}(w) = \delta\left(w, C_2(x)\right)$. Then, by the triangle inequality, it holds that

$$\delta_{C_2 \circ C_1}(w) \leq \delta\left(w, C_2(x)\right) + \delta_{C_2 \circ C_1}\left(C_2(x)\right) = \delta_{C_2}(w) + \delta_{C_2 \circ C_1}\left(C_2(x)\right) \leq \frac{1}{2}\delta_{C_2 \circ C_1}(w) + \delta_{C_2 \circ C_1}\left(C_2(x)\right)$$

and therefore

$$\delta_{C_2 \circ C_1}\left(C_2(x)\right) \geq \frac{1}{2}\delta_{C_2 \circ C_1}(w)$$

Let $c \in C_1$ be the codeword of C_1 closest to x. Since C_2 is λ-proximity preserving, we have that

$$\delta_{C_1}(x) = \delta\left(x, c\right) \geq \frac{1}{\lambda} \cdot \delta\left(C_2(x), C_2(c)\right) \geq \frac{1}{\lambda} \cdot \delta_{C_2 \circ C_1}\left(C_2(x)\right) \geq \frac{1}{2\lambda} \cdot \delta_{C_2 \circ C_1}\ (w)$$

Recall that with probability $\frac{1}{2}$, the verifier V emulates V_1. In this case, with probability at least $\left(\varepsilon_2^A\right)^{q_1}$ each of the invocations of A either outputs the correct symbol of x or rejects. Recall that V_1 rejects x with probability at least $\varepsilon_1 \cdot \delta_{C_1}(x) \geq \frac{\varepsilon_1}{2\lambda} \cdot \delta_{C_2 \circ C_1}(w)$. Therefore, in this case, V rejects w with probability at least $\frac{1}{2} \cdot \left(\varepsilon_2^A\right)^{q_1} \cdot \frac{\varepsilon_1}{2\lambda} \cdot \delta_{C_2 \circ C_1}(w)$. We conclude that regardless of which case holds, the rejection probability of V is as required. ■

Remark 5.13. Note that the fact that C_2 is proximity preserving is crucial to the analysis of the second case above. If C_2 was not proximity preserving, it could have been the case that a string $x \in \mathbb{F}^n$ is very close to C_1, but $C_2(x)$ is far from $C_2 \circ C_1$. In such a case, neither the invocation of V_2 nor the emulation of V_1 could "catch" $C_2(x)$, and thus the verifier V would have failed on $C_2(x)$.

Remark 5.14. Note that permutations codes trivially match the conditions of Theorem 5.12 with respect to C_2, that is, any permutation code is trivially proximity preserving, locally testable and locally accessible. In Section 5.2.2 we will see that repeated codes too match the conditions of Theorem 5.12.

5.2.2 Composition with repeated codes preserve local testability

In this subsection we show that repeated codes enjoy the properties defined above, and conclude that composition with repeated codes preserves local testability.

Fact 5.15. *Let C_1 be a λ-proximity preserving code. Then, for every natural number ℓ, the code $C_1^{[\ell]}$ is λ-proximity preserving.*

Fact 5.16. *Let $C_1 : \mathbb{F}^k \to \mathbb{F}^n$ be a (q, ε, r)-strong locally testable code. Then, for every natural number ℓ, the code $C_1^{[\ell]}$ is $(q, \varepsilon, r + \log \ell)$-strong locally testable.*

Proof We define a verifier V for the code $C_1^{[\ell]}$. Recall that a string $w \in \mathbb{F}^{\ell n}$ is a codeword of $C_1^{[\ell]}$ if and only if consists of ℓ consecutive blocks of length n such that every block is a codeword of C_1. When given oracle access to a string w, the verifier V chooses a random block of w and invokes the verifier of C_1 to test that this block is a legal codeword of C_1. The verifier V accepts if and only if the verifier of C_1 accepts. Clearly, V has query complexity q and randomness complexity $r + \log \ell$. The analysis of the rejection probability of V follows directly from the fact that the relative distance of a string $w \in \mathbb{F}^{\ell n}$ from $C_1^{[\ell]}$ is the average of the relative distances of the blocks of w from C_1. ■

Fact 5.17. *Let $C_1 : \mathbb{F}^k \to \mathbb{F}^n$ be a (q, ε, r)-locally accessible code. Then, for every natural number ℓ, the code $C_1^{[\ell]}$ is (q, ε, r)-locally accessible.*

Proof We define the accessor A for the code $C_1^{[\ell]}$. Recall that the encoding function of $C_1^{[\ell]}$ receives a message $m \in \mathbb{F}^{\ell k}$, partitions it to ℓ blocks of length k and encodes each of those blocks with C_1. When given an input i and oracle access to a string w, the accessor A computes the block of the message to which the i-th coordinate belongs. Then, A invokes the accessor of C_1 to decode the i-th coordinate from this block. If the accessor of C_1 rejects, then A rejects. Otherwise, A outputs the output of the accessor of C_1. ■

Corollary 5.18. *Let $C_1 : \mathbb{F}^k \to \mathbb{F}^n$ be a (q, ε, r)-CWP, let ℓ be a natural number that divides n and let $C_2 : \mathbb{F}^{n/\ell} \to \mathbb{F}^{n'/\ell}$ be a code. Then $C_2^{[\ell]} \circ C_1$ is a CWP with query complexity $(n'/\ell) \cdot q$, rejection ratio $\frac{1}{4(n'/\ell)} \cdot \varepsilon$ and randomness complexity $\max\{r, \log n\} + 1$.*

Proof The code C_2 is trivially n'/ℓ-proximity preserving. It is also trivially locally testable using the verifier that reads the entire tested string and accepts if and only if it is a codeword of C_2, and is trivially locally accessible using the accessor that reads the entire oracle, rejects if the oracle is not a codeword of C_2, and otherwise decodes the required coordinate and outputs it. The corollary now follows directly by applying Lemmas 5.15, 5.16 and 5.17 to $C_2^{[\ell]}$, and by applying Theorem 5.12 to $C_2^{[\ell]} \circ C_1$ with $\lambda = n'/\ell$ and $\varepsilon_2^T = \varepsilon_2^A = 1$. ■

Remark 5.19. We stress that while the query complexity and rejection ratio of $C_2^{[\ell]} \circ C_1$ depend on the block lengths of C_1 and C_2, they do not depend on ℓ.

5.3 Proof of Theorem 4.12

We are now ready to prove Theorem 4.12:

Theorem (Theorem 4.12, restated)**.** *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a code such that C^2 is a (q, ε, r)-CWP, let G be a d-regular graph on n vertices, let t be any constant integer and let $I : \mathbb{F}^{t+1} \to \mathbb{F}^{n_I}$ be a code. Then $\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2$ is a (q', ε', r')-CWP where $q' = O(q)$, $\varepsilon' = \Omega(\varepsilon)$ and $r' = \max\{r, \log n^2\} + O(1)$, and where the constants in the Big-O/Ω notations depend only on d,t and n_I.*

Proof Recall that by Proposition 5.8

$$\left(\mathrm{DistAmp}_{G,t,I}(C)\right)^2 = P_{\sigma_4} \circ I^{\left[n_I\left(d^t n\right)^2\right]} \circ P_{\sigma_3} \circ I^{\left[(t+1)\left(d^t n\right)^2\right]} \circ P_{\sigma_2} \circ R^{\left[(t+1)d^t n^2\right]} \circ P_{\sigma_1} \circ R^{[n^2]} \circ C^2$$

Theorem 4.12 now follows by applying Corollary 5.18 to each of the compositions with $R^{[\ell]}$ and $I^{[\ell]}$, and by observing that composing a CWP with a permutation code results in a CWP with the same parameters. ■

6 Wrapping everything together

In this section we use the theorems proved in Section 4 to obtain constructions of CWPs and LTCs. In Section 6.1 we give a formal proof of the main construction of CWPs that was outlined in Section 3. In Section 6.2 we show how LTCs can derived from the aforementioned CWPs. The CWPs of the main construction have super-constant query complexity and sub-constant rejection ratio, and in Sections 6.3 and 6.4 we show how they can be modified to have constant query complexity and constant rejection ratio. This modification immediately translates to LTCs with better query complexity and rejection probability. In Section 6.5 we discuss the size of the verifier circuits of our CWPs and LTCs. The strongest results we can prove are stated in Theorems 6.28 and 6.29.

6.1 The main construction

In this subsection we present our main construction of CWPs. Those CWPs have constant rate and relative distance, randomness complexity $\log k + O(\log\log k)$, query complexity $\operatorname{poly}(\log k)$ and rejection ratio $1/\operatorname{poly}(\log k)$. Recall that this construction is an iterative construction, that is, we start from a CWP with a small message length, and then increase its message length in an iterative process. We now state and prove the effect of a single iteration on a CWP.

Theorem 6.1 (Single Iteration)**.** *There exist constants R_0 and $\delta_0 < \frac{|\mathbb{F}|-1}{|\mathbb{F}|}$ such that for all constants $R \leq R_0$ there exists a randomized polynomial time procedure that works as follows: The procedure is given as input the generating matrix of a linear code $C : \mathbb{F}^k \to \mathbb{F}^n$ (for any $k, n \geq 1$) and a verifier circuit V with respect to which C^2 is a CWP. The procedure is only guaranteed to work provided that C has rate R and relative distance δ_0. In such a case, the procedure outputs, with probability at least $1 - \exp(-\Omega(k))$, the generating matrix of a code C' and a circuit V' such that the following holds:*

1. *The code C' has message length k^2, rate R_0 and relative distance δ_0.*
2. *If C^2 is a (q, ε, r)-CWP with respect to V, then $(C')^2$ is a $(O(q), \Omega(\varepsilon), r')$-CWP with respect to V', where $r' \stackrel{\text{def}}{=} \max\{r, \log k^2\} + \log k^2 + O(1)$ and where the constants in the Big-O/Ω notations depend only on R.*

Remark 6.2. Note that C^2 and $(C')^2$ have message lengths k^2 and k^4, respectively, and that if $r \geq \log k^2$ then $(r' - \log k^4) = (r - \log k^2) + O(1)$. In other words, applying the foregoing procedure increases the difference between the randomness complexity of the CWP and the logarithm of its message length by a constant term. In the terminology of proof lengths, this is analogous to decreasing the proof rate of the CWP by a constant factor.

Proof Let δ_0 be any constant strictly greater than $\sqrt[4]{7/8}$. The constant R_0 will be determined below. The procedure works as follows:

1. Let $C_{\text{TP}} = C^2$. The code C_{TP} has message length k^2, rate R^2 and relative distance δ_0^2. Without loss of generality, we assume that $\delta_0 \geq \sqrt[4]{7/8} + \frac{1}{n}$, since otherwise n is bounded by a constant and it is easy to prove the required result. By the Tensor Product

Theorem (Theorem 4.1), it follows that $\left(C_{\mathrm{TP}}\right)^2$ is a CWP with query complexity q, rejection ratio $\varepsilon_{TP} \stackrel{\text{def}}{=} \Omega(\varepsilon)$ (where the constant in the Big-Ω notation is absolute) and randomness complexity $r_{TP} \stackrel{\text{def}}{=} r + \log k^2 + O(1)$ (where the constant in the Big-O notation depend only on R). The procedure computes the generating matrix of C_{TP} and the verifier circuit of $\left(C_{\mathrm{TP}}\right)^2$, as defined in the proof of Theorem 4.1. Recall that the generating matrix of C_{TP} can be efficiently computed using, for example, the algorithm described in Remark 5.7.

2. Let $R_{\delta_0^2}$ be the constant whose existence is guaranteed by the Random Projection Theorem (Theorem 4.7). The procedure chooses a set $S \subseteq [n^2]$ of size $k^2/R_{\delta_0^2}$ uniformly at random. Let $C_{\mathrm{RP}} = \left(C_{\mathrm{TP}}\right)_{|S}$. By the Random Projection Theorem (Theorem 4.7), with probability $1 - \exp\left(-\Omega(k)\right)$, the function C_{RP} is a code with message length k^2, rate $R_{\delta_0^2}$ and relative distance $\frac{1}{2}\delta_0^2$. In such case, the code $\left(C_{\mathrm{RP}}\right)^2$ is a CWP with query complexity q, rejection ratio $\varepsilon_{\mathrm{RP}} \stackrel{\text{def}}{=} \Omega(\varepsilon_{\mathrm{TP}})$ (where the constant in the Big-Ω notation depends only on R and δ_0) and randomness complexity r_{TP}. The procedure computes the generating matrix of C_{RP} and the verifier circuit of $\left(C_{\mathrm{RP}}\right)^2$, as defined in the proof of Theorem 4.7.

3. Set $C' = \mathrm{DistAmp}_{\frac{1}{2}\delta_0^2 \to \delta_0}\left(C_{\mathrm{RP}}\right)$. By the Distance Amplification Theorem (Theorem 4.9), C' has message length k^2 and relative distance δ_0. Furthermore, C' has a rate that depends only on δ_0 and $R_{\delta_0^2}$ - set R_0 to be this rate and note that R_0 is a constant that depends only on δ_0, since $R_{\delta_0^2}$ is a constant that depends only on δ_0. Moreover, $(C')^2$ is a CWP with query complexity $q' = O(q)$, rejection ratio $\varepsilon' = \Omega(\varepsilon_{\mathrm{RP}})$ and randomness complexity $r' = \max\left\{r_{\mathrm{TP}}, \log|S|^2\right\} + O(1)$ (where the constants in the Big-O/Ω notations depend only on δ_0). The procedure computes the generating matrix of C' and the verifier circuit V' of $(C')^2$, as defined in the proof of Theorem 4.9.

The procedure outputs the generating matrix of C' and the circuit V'. It is easy to verify that this procedure satisfies the requirements of the theorem. In particular, note that $q' = O(q)$, that $\varepsilon' = \Omega(\varepsilon_{\mathrm{RP}}) = \Omega(\varepsilon_{\mathrm{TP}}) = \Omega(\varepsilon)$ where the constant in the Big-Ω depends only on R and that

$$\begin{aligned} r' &= \max\left\{r_{\mathrm{TP}}, \log|S|^2\right\} + O(1) \\ &= \max\left\{r_{\mathrm{TP}}, \log\left(k^4/R_{\delta_0^2}^2\right)\right\} + O(1) \\ &= \max\left\{r_{\mathrm{TP}}, \log k^4\right\} + O(1) \\ &= \max\left\{r + \log k^2, \log k^4\right\} + O(1) \\ &= \max\left\{r, \log k^2\right\} + \log k^2 + O(1) \end{aligned}$$

where the constant in the Big-O notation depends only on R. ■

We can now use Theorem 6.1 to give our main construction of CWPs. The construction is similar to the construction discussed in Section 3, except for one difference: Since the success probability of the procedure of Theorem 6.1 is $1 - \exp\left(-\Omega(k)\right)$, we wish to apply it

to CWPs with message length as large as possible. Thus, instead of using a code of constant message length as the starting point of the construction, we use a code with message length $\text{poly}(\log k)$. The trivial verifier for such a code has query complexity $\text{poly}(\log k)$, but this is not a problem since we are allowed to have query complexity $\text{poly}(\log k)$ anyhow.

Theorem 6.3 (Main Construction)**.** *For every constant $c > 0$ the following holds: There exists an infinite family of CWPs $\{C_k\}_k$ such that C_k has block length $O(k)$, relative distance $\Omega(1)$, query complexity $\text{poly}(\log k)$, rejection ratio $1/\text{poly}(\log k)$, and randomness complexity $\log k + O(\log\log k)$. Furthermore, the codes in the family are linear and there exists a probabilistic algorithm that on input k, runs in time $\text{poly}(k)$ and outputs with probability $1 - \exp(-\Omega(\log^c k))$ the generating matrix and verifier circuit of C_k.*

Proof Let δ_0 and R_0 be the constants from Theorem 6.1. Let $C^{(0)}$ be an arbitrary linear code with message length $k_0 \stackrel{\text{def}}{=} \log^c k$, a constant rate of at most R_0 and relative distance δ_0 (Such a code can be obtained, for example, from the Zybalov bound, see Appendix A, Theorem A.4). Let $V^{(0)}$ be the trivial verifier for $\left(C^{(0)}\right)^2$ that when given oracle access to a tested string and a proof string, reads the tested string entirely and accepts if and only if it is a legal codeword of $\left(C^{(0)}\right)^2$. Clearly, $\left(C^{(0)}\right)^2$ with respect to this verifier is a CWP with query complexity k_0, rejection ratio 1 and randomness complexity 0.

On input k, the probabilistic algorithm acts as follows: The algorithm first constructs the generating matrix of $C^{(0)}$ and the verifier circuit $V^{(0)}$ of $\left(C^{(0)}\right)^2$. Then, for every $0 \leq i \leq \log\log_{k_0} k - 1$, the algorithm runs the procedure of Theorem 6.1 on the generating matrix of $C^{(i)}$ and the verifier $V^{(i)}$, and sets $C^{(i+1)}$ and $V^{(i+1)}$ to be the output of the procedure. Finally, the algorithm outputs the generating matrix of $\left(C^{(\log\log_{k_0} k-1)}\right)^2$ and the verifier $V_{\log\log_{k_0} k-1}$. It can be seen that if all the invocations of the procedure succeed, in the sense that they output the generating matrix and verifier circuit stated in Theorem 6.1, then the algorithm outputs the generating matrix and verifier circuit stated in Theorem 6.3. In particular:

1. In every iteration, the rejection ratio is multiplied by some constant factor $\alpha < 1$. Therefore, after $\log\log_{k_0} k$ iterations, the CWP will have rejection ratio $\alpha^{\log\log_{k_0} k} = 1/\text{poly}(\log k)$.

2. In every iteration, the query complexity is multiplied by some constant factor $\beta > 1$. Therefore, after $\log\log_{k_0} k$ iterations, the CWP will have the query complexity $\beta^{\log\log_{k_0} k} \cdot \text{poly}(\log k) = \text{poly}(\log k)$.

3. In every iteration, the difference between the randomness complexity of the CWP and its message length increases by a constant term $\gamma > 0$. Therefore, after $\log\log_{k_0} k$ iterations, the CWP will have randomness complexity $\log k + \gamma \cdot \log\log_{k_0} k = \log k + O(\log\log k)$.

It remains to analyze the probability that all the invocations of the procedure succeed. By the union bound, the probability that one of the invocations fails is bounded by

$$\sum_{i=0}^{\log\log_{k_0} k} \exp\left(-\Omega(k_0^{2^i})\right) \leq \sum_{i=0}^{\infty} \exp\left(-i \cdot \Omega(k_0)\right) \leq \exp\left(-\Omega(k_0)\right)$$

The algorithm therefore succeeds with probability $1 - \exp(-\Omega(k_0)) = 1 - \exp(-\Omega(\log^c k))$, as required. ■

Remark 6.4. The reader should note the importance of maintaining a constant rate in every iteration of the algorithm of Theorem 6.3. Let k_i denote the message length of $C^{(i)}$ for every i. If we allowed the rate to drop to a sub-constant function, then the procedure of Theorem 6.1 (and in particular, the tensor product operation) would have increased the randomness complexity of $C^{(i)}$ by more than $\log k_i^2 + O(1)$, and after $\log \log k$ iterations we would have ended up with randomness complexity of much more than $\log k + O(\log \log k)$. This is the reason why we apply the random projection operation.

Remark 6.5. The reader should note that, while increasing the constant c in Theorem 6.3 improves the success probability of the algorithm, this improvement is not "for free": The higher we set c to be, the higher the query complexity of C will be.

The density of the message lengths So far we have ignored the possible message lengths that can be achieved by our construction. Observe that there are values of k for which no CWP C_k can be constructed using the proof of Theorem 6.3 (e.g., large prime values of k). However, this construction can clearly achieve every k of the form 2^{2^i} for some integer i, so a more precise statement of the above theorem would be that there exist an infinite family $\{C_{k_i}\}_{i=0}^{\infty}$ for $k_i = 2^{2^i}$. In particular, we have that $k_{i+1} = k_i^2$. It turns out that we can do better, namely, construct a family $\{C_{k_i}\}_{i=0}^{\infty}$ for a sequence that satisfies $k_{i+1} = (1 + o(1)) \cdot k_i$, by choosing the message length k_0 of the initial CWP in a more clever way. Details follow. The current construction of Theorem 6.3 can be viewed as follows: Given a natural number i, we construct a CWP with message length 2^{2^i} by starting from a CWP of initial message length 2^{ci} and then applying $i - \log(ci)$ iterations of Theorem 6.1. Instead, we can consider a construction that given two natural numbers i and $1 \leq j \leq 2^{2ci} - 2^{ci}$, starts from a CWP of initial message length $2^{ci} + j$ and then applies $i - \log(ci)$ iterations of Theorem 6.1. This yields a CWP with message length $k_{i,j} \stackrel{\text{def}}{=} (2^{ci} + j)^{2^i/(ci)}$. A straightforward calculation shows that if one sorts the values $k_{i,j}$ in lexicographical order of (i, j), then the quotient between two successive values of $k_{i,j}$ is $1 + 2^{-(c-1)\cdot i}$.

6.2 From CWPs to LTCs

We now derive LTCs from the CWPs of Theorem 6.3. Since we want our LTCs to be *linear*, we first need to define another property of CWPs.

Definition 6.6. A CWP $C : \mathbb{F}^k \to \mathbb{F}^n$ with proof length m and verifier V is said to have linear proofs if and only if there exists a $k \times m$ matrix P such that for every $x \in \mathbb{F}^k$ it holds that $x \cdot P$ is a proof string of $C(x)$. We refer to P as the proof matrix of C.

It is easy to prove that the CWPs of the main construction (Theorem 6.3) have linear proofs, by noting that the initial CWP of the construction has linear proofs (since it has no proofs) and that the linearity of the proofs is preserved in every iteration. Furthermore, the algorithm of Theorem 6.3 can be modified to output the proof matrix of the CWP, in addition to its generating matrix and verifier. We can now use the following transformation

from CWPs to LTCs, which was developed in prior works (see, e.g., [GS02, Sec. 5] and [BGHSV06, Sec. 4.1]). We include the analysis of the transformation for the sake of self-containment.

Theorem 6.7. *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a (q, ε, r)-CWP with relative distance δ, and let $\tau > 0$ be an arbitrarily small constant. Then there exists locally testable code C' with message length k, block length $O(n + q \cdot 2^r)$, relative distance $\Omega(\delta)$, query complexity $\max\{q, 2\}$, distance threshold τ and rejection probability $\Omega(\varepsilon)$, where the constants in the Big-O/Ω notation depend on τ. Furthermore, if C has linear proofs then C' is linear, and there exists a polynomial time algorithm that given the generating matrix, verifier circuit and proof matrix of C, outputs the generating matrix and verifier circuit of C'.*

Proof Idea The most straightforward way to construct C' from C is defining $C'(x) = C(x)\pi_{C(x)}$, for every message $x \in \mathbb{F}^k$, where $\pi_{C(x)}$ is a proof string of $C(x)$. However, if the proof length of C is much larger than its block length, this construction has two problems:

1. We are not guaranteed that the proof strings have good relative distance. That is, it is possible that there are distinct codewords $c_1, c_2 \in C$ such that $\pi_{c_1} = \pi_{c_2}$. In such a case, the codewords $c_1\pi_{c_1}$ and $c_2\pi_{c_2}$ of C' may be very close, and so C' may have a very low relative distance.

2. We are not guaranteed that errors in the proof string are detected by the verifier. That is, consider the case where the verifier of C' is given oracle access to a string $c\pi'$, where $c \in C$ and π' is very far from π_c. In such a case, $c\pi'$ may be very far from C', but the verifier of C is not guaranteed to reject $c\pi'$ at all.

The solution to both problems is to use many copies of the codeword such that their length dominates the length of the resulting codeword. That is, we define

$$C'(x) = \underbrace{C(x) \ldots C(x)}_{\ell} \pi_{C(x)} \tag{3}$$

such that $\ell \cdot |C(x)| \gg \left|\pi_{C(x)}\right|$. Defining C' this way ensures that any harm caused by the proof strings is absorbed by the codewords. For example, note that even if $\pi_{c_1} = \pi_{c_2}$ for some distinct codewords $c_1, c_2 \in C$, the corresponding codewords $c_1 \ldots c_1\pi_{c_1}$ and $c_2 \ldots c_2\pi_{c_2}$ of C' are guaranteed to be very far, because those codewords differ on a lot of coordinates in the $c_1 \ldots c_1$ and $c_2 \ldots c_2$ part, and this part dominates the π_{c_1} and π_{c_2} part.

Proof of Theorem 6.7 Without loss of generality, assume that all the proof strings of C are of length $m = 2^r \cdot q$, and let $\ell = (\lceil 2/\tau \rceil - 1) \cdot \lceil m/n \rceil$. For every $c \in C$, fix some specific proof string of c and denote it by π_c. For every message $x \in \mathbb{F}^k$, we define $C'(x)$ to be the concatenation of ℓ copies of $C(x)$ and one copy of $\pi_{C(x)}$, as in Equation 3. The code C' has block length $\ell n + m = O(n + q \cdot 2^r)$. Furthermore, C' has relative distance of at least $(1 - \tau/2) \cdot \delta = \Omega(\delta)$, since for every two messages $x \neq y \in \mathbb{F}^k$, each copy of $C(x)$ in $C'(x)$ differs from the corresponding copy of $C(y)$ in $C'(y)$ on δ-fraction of their coordinates, and those copies contribute at least $(1 - \tau/2)$-fraction of the coordinates of $C'(x)$ and $C'(y)$. It

should also be clear that if C has linear proofs then C' is linear, provided that we fixed $\pi_{C(x)} = x \cdot P$ where P is the proof matrix of C.

We turn to define the verifier V' of C'. Let V denote the verifier of C. When given oracle access to a tested string $w \in \mathbb{F}^{\ell n+m}$ the verifier V' views w as composed of ℓ strings $w^1, \ldots, w^\ell$ of length n and another string π of length m. With probability $\frac{1}{2}$, the verifier V' emulates $V^{w^1,\pi}$ and accepts if and only if $V^{w^1,\pi}$ accepts. Otherwise, V' chooses $i \in [n]$ and $j \in [\ell]$ uniformly at random and accepts if and only if $w_i^1 = w_i^j$.

The query complexity of C' is clearly $\max\{q, 2\}$. We show that V' rejects strings that are τ-far from C' with probability at least $\frac{1}{8} \cdot \tau \cdot \varepsilon$. Let $w \in \mathbb{F}^{\ell n+m}$ be a string that is τ-far from C', and let $w^1, \ldots, w^\ell$ and π be as in the previous paragraph. Let $c \in C$ be a codeword of C such that $\delta_C(w^1) = \delta(w^1, c)$. Let c' be the codeword of C' that consists of ℓ copies of c and of one copy of π_c. We know that w is τ-far from C' so in particular $\delta(w, c') \geq \tau$. Since the coordinates of π form at most $\tau/2$-fraction of all the coordinates of w, the relative distance between the concatenation of $w^1, \ldots, w^\ell$ and the concatenation of ℓ copies of c must be at least $\frac{\tau}{2}$. This implies that $\mathbb{E}_{j \in [\ell]}[\delta(w^j, c)] \geq \tau/2$.

Suppose that V' is given oracle access to w. If $\delta_C(w^1) \geq \frac{1}{4} \cdot \tau$, then the emulation of $V^{w^1,\pi}$ rejects with probability at least $\frac{1}{4} \cdot \tau \cdot \varepsilon$, and therefore V' rejects with probability at least $\frac{1}{8} \cdot \tau \cdot \varepsilon = \Omega(\varepsilon)$, as required. On other hand, suppose that $\delta_C(w^1) = \delta(c, w^1) < \frac{1}{4} \cdot \tau$. Then, by the triangle inequality, it holds that

$$\mathbb{E}_{j \in [\ell]}\left[\delta(w^j, w^1)\right] \geq \mathbb{E}_{j \in [\ell]}\left[\delta(w^j, c)\right] - \mathbb{E}_{j \in [\ell]}\left[\delta(w^1, c)\right] > \frac{1}{4} \cdot \tau$$

and therefore

$$\Pr_{i \in [n], j \in [\ell]}\left[w_i^1 \neq w_i^j\right] > \frac{1}{4} \cdot \tau$$

Now, with probability $\frac{1}{2}$, the verifier V' chooses $i \in [n]$ and $j \in [\ell]$ uniformly at random and checks that $w_i^1 = w_i^j$. It follows that V' rejects with probability at least $\frac{1}{2} \cdot \frac{1}{4} \cdot \tau \geq \frac{1}{8} \cdot \tau \cdot \varepsilon$, as required. ■

By applying Theorem 6.7 to the CWPs of the main construction (Theorem 6.3), we obtain the following construction of LTCs:

Theorem 6.8. *There exists a constant $\delta > 0$ such that for every two constants $c > 0$ and $\tau > 0$ the following holds: There exists an infinite family of LTCs $\{C_k\}_k$ such that C_k has block length $k \cdot \mathrm{poly}(\log k)$, relative distance δ, query complexity $\mathrm{poly}(\log k)$, distance threshold τ, and rejection probability $1/\mathrm{poly}(\log k)$. Furthermore, the codes in the family are linear and there exists a probabilistic algorithm that on input k, runs in time $\mathrm{poly}(k)$ and outputs with probability $1 - \exp(-\Omega(\log^c k))$ the generating matrix and verifier circuit of C_k.*

Remark 6.9. Note that the proof of Theorem 6.7 also works for sub-constant values of τ, in which case we have block length $O((n + q \cdot 2^r)/\tau)$ and rejection probability $\Omega(\tau \cdot \varepsilon)$, while all the other parameters remain as in Theorem 6.7.

6.3 Obtaining CWPs with constant query complexity

The CWPs and LTCs we obtained in Theorems 6.3 and 6.8 have query complexity poly $(\log k)$. In this subsection we show that the query complexity of our CWPs and LTCs can be reduced to a constant. We begin with proving a general query reduction theorem for CWPs. In order to prove the theorem, we will need the following definition.

Definition 6.10. Let C be a (q, ε, r)-CWP with a verifier V. We say that V is a linear verifier if when given coin tosses $\omega \in \{0,1\}^r$ and oracle access to any oracle, the verifier V acts as follows: The verifier V first chooses a sequence of vectors $v_1, \ldots, v_\ell \in \mathbb{F}^q$ that depend only on ω. Then, V queries the tested string and the proof string on q locations that depend only on ω. Let $a \in \mathbb{F}^q$ be the vector that contains the answers to those queries. The verifier V accepts if and only if $\langle a, v_i \rangle = 0$ for all $1 \leq i \leq \ell$.

We can now state and prove the following well-known query reduction theorem. The proof of the theorem is included for the sake of self-containment.

Theorem 6.11 (Query Reduction)**.** *Let C be a (q, ε, r)-CWP that has a* linear *verifier V. Then there exists a* linear *verifier V' with respect to which C is a CWP with query complexity* 3*, rejection ratio $\varepsilon/\operatorname{poly}(q)$, and randomness complexity $r + O(\log q)$. Furthermore, V' can be computed in polynomial time from V. Finally, if C has linear proofs with respect to V then it also has linear proofs with respect to V', and the proof matrix of C with respect to V' can be computed in polynomial time from the verifier V, the generating matrix and proof matrix of C with respect to V.*

The proof of Theorem 6.11 is based on a transformation of systems of linear equations to systems in which every equations contains only three variables. For example, consider the following equation:

$$X_1 + X_2 + X_3 + X_4 + X_5 + X_6 = 0$$

By adding auxiliary variables Y_1, Y_2, Y_3, one can create the following "equivalent" system of linear equations:

$$\begin{aligned} X_1 + X_2 - Y_1 &= 0 \\ X_3 + X_4 - Y_2 &= 0 \\ X_5 + X_6 - Y_3 &= 0 \\ Y_1 + Y_2 + Y_3 &= 0 \end{aligned}$$

By generalizing this idea, one can prove the following proposition:

Proposition 6.12. *Given a system A of at most ℓ linear equations over variables $X_1, \ldots, X_\ell$, we can construct a new system A' of linear equations that has the following properties:*

- *A' is a system of linear equations over the variables $X_1, \ldots, X_\ell$ and also over new auxiliary variables $Y_1, \ldots, Y_{\ell'}$ for $\ell' = \operatorname{poly}(\ell)$.*
- *A' contains at most $\operatorname{poly}(\ell)$ equations.*
- *Every equation of A' contains at most three variables.*

- $(x_1, \ldots, x_\ell) \in \mathbb{F}^\ell$ *is a solution to the system* A *if and only if there exists a* $(y_1, \ldots, y_{\ell'}) \in \mathbb{F}^{\ell'}$ *such that* $(x_1, \ldots, x_\ell, y_1, \ldots, y_{\ell'})$ *is a solution the system* A'*. Furthermore, for every* $1 \leq i \leq \ell'$*, the value* y_i *is a linear combination of the values* $x_1, \ldots, x_\ell$*.*

Proof of Theorem 6.11 Let C and V be as in Theorem 6.11. This means that, on every given sequence of coin tosses, V chooses some system of linear equations and checks that the answers to its queries form a solution to this system. For every sequence of coin tosses $\omega \in \{0,1\}^r$, let A_ω denote the system that V chooses given ω. Note that without loss of generality, we can assume that A_ω contains at most q equations, since A_ω is a system of linear equations over q variables. Let A'_ω denote result of applying Proposition 6.12 to A_ω, and let $Y_{\omega,1}, \ldots, Y_{\omega,q'}$ (for $q' = \text{poly}(q)$) denote the corresponding auxiliary variables. We turn to define the verifier V' and its corresponding proof strings:

- **The proof strings of** V'**:** Let $c \in C$ be a codeword and let π_c be the proof string of c with respect to V. For every $\omega \in \{0,1\}^r$ let $(x_{\omega,1}, \ldots, x_{\omega,q}) \in \mathbb{F}^q$ denote the answers that V gets to its queries when given coin tosses ω and oracle access to c and π_c. We know that $(x_{\omega,1}, \ldots, x_{\omega,q})$ is a solution to the system A_ω, and therefore by Proposition 6.12 there exists a vector $(y_{\omega,1}, \ldots, y_{\omega,q'}) \in \mathbb{F}^{q'}$ such that $(x_{\omega,1}, \ldots, x_{\omega,q}, y_{\omega,1}, \ldots, y_{\omega,q'})$ is a solution the system A'_ω. We now define the proof string of c with respect to V' to consist of π_c and of the tuple $(y_{\omega,1}, \ldots, y_{\omega,q'}) \in \mathbb{F}^{q'}$ for every $\omega \in \{0,1\}^r$. Since for every $\omega \in \{0,1\}^r$ and every $1 \leq i \leq q'$ the value $y_{\omega,i}$ is a linear combination of the values $x_{\omega,1}, \ldots, x_{\omega,q}$, it follows that C has linear proofs with respect to V'.

- **The verifier** V'**:** Given oracle access to a tested string w and a proof string π, the verifier V' first tosses a sequence $\omega \in \{0,1\}^r$ of coins and emulates V to find A_ω. The verifier V' then computes the system A'_ω. Finally, V' chooses a single equation of A'_ω uniformly at random and checks that it is satisfied by w and π. Note that since the chosen equation of A'_ω contains at most three variables, the latter check can be done using at most three queries. Furthermore, since A'_ω contains at most $\text{poly}(q)$ equations, it follows see that V' has rejection ratio $\varepsilon/\text{poly}(q)$ and randomness complexity $r + O(\log q)$.

It follows that C with respect to V' is a CWP with the parameters as claimed. ■

We turn to apply Theorem 6.11 to the CWPs of the main construction (Theorem 6.3). We note that the CWPs of the main construction have linear verifiers, since the initial CWP of the construction has a linear verifier and the linearity of the verifier is preserved in every iteration. We obtain the following construction of CWPs with constant query complexity:

Theorem 6.13. *Same as Theorem 6.3, but with constant query complexity.*

Using Theorem 6.7, we obtain LTCs with constant query complexity:

Theorem 6.14. *Same as Theorem 6.8, but with constant query complexity.*

The role of linear verifiers We note that it is possible to reduce the query complexity of CWPs without assuming that the verifier is linear, using roughly the following argument: Suppose we want to reduce the query complexity of a CWP C that has a verifier V. Assume first that C is over binary alphabet rather than over the alphabet $\mathbb{F}$. Then, for every sequence of coin tosses $\omega \in \{0,1\}^r$, we compute a SAT formula Φ_ω whose variables correspond to the queries of V on coin tosses ω, such that V would have accepted the answers to its queries if and only if those answers satisfied Φ_ω. We then transform Φ_ω to a 3-SAT formula Φ'_ω by adding auxiliary variables. Finally, we define a new verifier V' that chooses a random clause of Φ'_ω and checks that the clause is satisfied. Obviously, C with respect to V' has query complexity 3. Now, if C is a CWP over the alphabet $\mathbb{F}$ (and not over a binary alphabet), we can represent every symbol of $\mathbb{F}$ by some binary string and proceed as before.

The problem with the this argument is that the CWPs that result from this transformation may not have linear proofs. Thus, the assumption that the CWP has a linear verifier, while not necessary to do query reduction, is needed in order to to ensure that the resulting CWP will have linear proofs.

Remark 6.15. We comment that the linearity of the verifier of the main construction (Theorems 6.3 and 6.8) can also be deduced from a result of Ben-Sasson et al. [BHR05] showing that every linear locally testable code has a linear verifier. However, using such a strong result is unnecessary, since it is easy to give a direct proof of the linearity of our verifier.

6.4 Obtaining CWPs with constant rejection ratio

The CWPs we obtained in Theorems 6.3 and 6.13 have rejection ratio of only $\Omega\left(1/\operatorname{poly}\left(\log k\right)\right)$, and thus translated into LTCs with rejection probability $\Omega\left(1/\operatorname{poly}\left(\log k\right)\right)$. In this subsection we use the gap amplification technique of [D07] to obtain CWPs with constant rejection ratio while maintaining the constant query complexity and the randomness complexity of $\log k + O(\log\log k)$. Such CWPs translate into LTCs with constant rejection probability that maintain the block length of $k \cdot \operatorname{poly}\left(\log k\right)$.

Basically, the gap amplification theorem of [D07] provides a transformation that increases the rejection ratio of a CWP by a constant factor while increasing its randomness complexity by a constant additive term. By applying this transformation to our CWPs for $O(\log\log k)$ times, we get CWPs of constant rejection ratio, while maintaining the randomness complexity of $\log k + O(\log\log k)$ (though the constant inside the Big-O notation will be a larger one). Implementing this idea requires handling two issues:

1. Since the gap amplification theorem of [D07] was proved in a somewhat different setting, there are few minor technical issues that need to be handled before we can apply it to our CWPs. Those issues are handled in Section 6.4.1.

2. The gap amplification theorem of [D07] does not preserve the linearity of the proof part of the CWPs. In order to obtain CWPs of constant rejection ratio that have *linear proofs*, we need to modify the proof of [D07]. This is done in Sections 6.4.2 and 6.4.3. We mention that this modification of [D07] was observed independently by Ben-Sasson et. al. [BHLM07].

We note that this subsection is less detailed than the rest of this paper, since writing the full details would have required us to rewrite large parts of [D07]. It is not difficult, however, to fill-up the missing details, based on [D07].

6.4.1 Obtaining CWPs with non-linear proofs and constant rejection ratio

In order to state the gap amplification theorem of [D07], we first extend the definition of CWPs to allow CWPs whose proof strings are over a different alphabet than their codewords.

Definition 6.16. We say that a CWP C has proof alphabet Σ if its proof strings are over the alphabet Σ (which may or may not be $\mathbb{F}$). The verifier of such a CWP is given oracle access to two oracles: The oracle of the tested string, which is over the alphabet $\mathbb{F}$, and the oracle of the proof string, which is over the alphabet Σ.

The gap amplification theorem of [D07] we need is stated and proved for PCPPs (a.k.a Assignment Testers). Nevertheless, the following *variant* of this theorem is implicit in [D07]:

Theorem 6.17 (implicit in [D07, Theorem 9.1][1])**.** *There exists a constant $\varepsilon_0 > 0$ such that for every natural number e the following holds: Let C be a CWP with a verifier V, rejection ratio ε, randomness complexity r, query complexity 2 and proof alphabet $\mathbb{F}^e$. Then, there exists a verifier V' with respect to which C is a CWP with rejection ratio at least $\min\{2\varepsilon, \varepsilon_0\}$, randomness complexity $r + O(1)$, query complexity 2 and proof alphabet $\mathbb{F}^e$. Furthermore, there exists a polynomial time algorithm that when given as input the verifier V, outputs the verifier V'.*

We comment that in order to derive Theorem 6.17 from the proof of [D07, Theorem 9.1], one should pay attention to the following issues:

1. Theorem 9.1 of [D07] is stated for PCPPs of $\mathcal{NP}$-complete sets. However, the proof of [D07, Theorem 9.1] does not rely on the fact that the input verifier is a verifier for a PCPP of an $\mathcal{NP}$-complete set. In particular, the algorithm claimed in Theorem 6.17 is the same as the algorithm used in the proof of [D07, Theorem 9.1], and it can be shown that if we feed this algorithm with a verifier of a CWP C (rather than a verifier of an $\mathcal{NP}$-complete set), then the output will be a verifier of C.

2. Theorem 9.1 of [D07] is originally stated only for PCPPs of sets of *binary* strings, while we want to use it for codes over the alphabet $\mathbb{F}$. However, the proof of [D07, Theorem 9.1] works also for sets of strings over $\mathbb{F}$.

We would like to apply Theorem 6.17 to our CWPs. However, Theorem 6.17 is only stated for CWPs with query complexity 2, while the CWPs we constructed in Theorem 6.13 have query complexity 3. This problem is solved using the following well-known technique (see, e.g., [**?**]):

[1] There seems to be a gap in the proof of [D07, Theorem 8.1]. However, this gap can be filled, see [GM07b] for details.

Lemma 6.18. *Let C be a (q, ε, r)-CWP with a verifier V. Then, there exists a verifier V' with respect to which C is a CWP with query complexity 2, proof alphabet $\mathbb{F}^q$, rejection ratio $\frac{1}{q} \cdot \varepsilon$ and randomness complexity $r + \log q$. Furthermore, there exists a polynomial time algorithm that, when given V as input, outputs the verifier V'.*

Proof We include the proof of this lemma for the sake of self-containment. We define the proof strings of C with respect to V' as follows. Let $c \in C$ and let π_c be its proof string with respect to V. The proof string π'_c of c with respect to V' consists of two parts:

1. The string π_c, viewed as a string over $\mathbb{F}^q$ using some simple embedding of $\mathbb{F}$ into $\mathbb{F}^q$.
2. For every sequence of coin tosses $\omega \in \{0,1\}^r$, the proof string π'_c contains a symbol in $\mathbb{F}^q$ that contains the q answers that V would have received if it was given coin tosses ω and oracle access to c and π_c. Let i_ω denote the coordinate of this symbol in π'_c.

Given oracle access to a string w and proof string π', the verifier V' acts as follows. V' first tosses random coins $\omega \in \{0,1\}^r$, and queries π' at i_ω. Let a_ω denote the answer V' receives to its query, and let $i_1, \ldots, i_q$ the coordinates that V would have queried on coin tosses ω. The symbol a_ω is supposed to contain the values of w and π' on the coordinates $i_1, \ldots, i_q$. Now, V' first checks that V would have accepted on coin tosses ω and given the vector a_ω as answers to its queries. Then, V' chooses $j \in [q]$ uniformly at random and checks that the j-th element of a_ω equals to w_{i_j} (if i_j is a coordinate of the tested string) or to $(\pi')_{i_j}$ (if i_j is a coordinate of the proof string). It is not hard to see that C with respect to V' is a CWP with the claimed parameters. ■

We now obtain CWPs with constant rejection ratio as follows: First, we apply Lemma 6.18 to the CWPs of Theorem 6.13, and obtain CWPs with proof alphabet $\mathbb{F}^3$, query complexity 2 and related rejection ratio and randomness complexity. Next, we apply Theorem 6.17 to the latter CWPs for $O(\log \log k)$ times and obtain CWPs with rejection ratio $\Omega(1)$, randomness complexity $\log + O(\log \log k)$, query complexity 2 and proof alphabet $\mathbb{F}^3$. Finally, we modify the aforementioned CWPs to have proof alphabet $\mathbb{F}$ by "unbundling" every symbol in $\mathbb{F}^3$ to a sequence of 3 symbols in $\mathbb{F}$. The result is CWPs with query complexity 6, rejection ratio $\Omega(1)$, and randomness complexity $\log k + O(\log \log k)$, as desired.

Unfortunately, the CWPs constructed this way *do not have linear proofs*, since Theorem 6.17 does not maintain the linearity of the proof strings. In order to maintain the linearity of the proof strings, we have to modify the proof of [D07].

6.4.2 Review of additional known techniques

Before modifying the proof of [D07], we review some known techniques that will be useful in the proof. First, we define a generalization of the Hadamard code to arbitrary finite fields.

Definition 6.19. The $|\mathbb{F}|$-ary Hadamard code, denoted H, encodes a message $x \in \mathbb{F}^k$ by the codeword $H(x) \in \mathbb{F}^{|\mathbb{F}|^k}$ defined by

$$H(x)_i = \langle x, i \rangle \text{ for every } i \in \mathbb{F}^k$$

That is, $H(x)$ consists of the inner products of $x \in \mathbb{F}^k$ with all the vectors in $\mathbb{F}^k$.

It is not hard to see that the $|\mathbb{F}|$-ary Hadamard code has relative distance $\frac{|\mathbb{F}|-1}{|\mathbb{F}|}$. Furthermore, it is well-known that the $|\mathbb{F}|$-ary Hadamard code is strong locally testable (see Definition 5.10):

Lemma 6.20. *The $|\mathbb{F}|$-ary Hadamard code is strong locally testable with query complexity 3, rejection ratio 1/6 and randomness complexity $2(k+1)\log|\mathbb{F}|$.*

Proof The verifier of the $|\mathbb{F}|$-ary Hadamard code is defined as follows: Given oracle access to a tested string $w \in \mathbb{F}^{|\mathbb{F}|^k}$, the verifier chooses two vectors $u, v \in \mathbb{F}^k$ and two scalars $a, b \in \mathbb{F}$ uniformly at random, and checks that

$$a \cdot w_u + b \cdot w_v = w_{a \cdot u + b \cdot v}$$

The analysis of this verifier follows from a simple variant of the proof of Blum et al. [BLR93]. For completeness, we include this analysis in Appendix B. ■

The following self-correction property of the $|\mathbb{F}|$-ary Hadamard code is the key property we use in the modified proof of [D07]. Intuitively, this proposition means that given oracle access to the $|\mathbb{F}|$-ary Hadamard encoding of a string $x \in \mathbb{F}^k$, one can retrieve the inner product of x with any vector in $\mathbb{F}^k$ using only 2 queries, *even if the oracle is slightly corrupted.* We will use this property to emulate, using very few queries, verifiers that check linear conditions.

Lemma 6.21 (Self-correction of the $|\mathbb{F}|$-ary Hadamard)**.** *There exists a probabilistic polynomial time oracle machine that on input $v \in \mathbb{F}^k$ and oracle access to any $w \in \mathbb{F}^{|\mathbb{F}|^k}$ such that $\delta(w, H(x)) < \frac{1}{2}$ for some $x \in \mathbb{F}^k$, outputs $\langle x, v \rangle$ with probability at least $1 - 2 \cdot \delta(w, H(x))$. Furthermore, the machine makes at most 2 non-adaptive queries to its oracle and tosses at most $k \log|\mathbb{F}|$ coins.*

Proof On input $v \in \mathbb{F}^k$ and oracle access to w, the machine first chooses a vector $u \in \mathbb{F}^k$ uniformly at random, then queries w at the coordinates u and $v - u$, and finally outputs $w_u + w_{v-u}$.

The vectors u and $v - u$ are uniformly distributed over $\mathbb{F}^k$, and we therefore have that both $\Pr_u[w_u \neq H(x)_u]$ and $\Pr_u\left[w_{v-u} \neq H(x)_{v-u}\right]$ are upper bounded by $\delta(w, H(x))$. By the union bound, with probability at least $1 - 2 \cdot \delta(w, H(x))$ we have that both $w_u = H(x)_u$ and $w_{v-u} = H(x)_{v-u}$ hold, which implies that

$$w_u + w_{u+v} = H(x)_u + H(x)_{v-u} = \langle x, u \rangle + \langle x, v - u \rangle = \langle x, v \rangle$$

It follows that the machine outputs $\langle x, v \rangle$ with probability at least $1 - 2 \cdot \delta(w, H(x))$, as required. ■

We also use the following notation.

Notation. Let C be a CWP with a linear verifier V. Recall that V is a linear verifier if and only if on every sequence of coin tosses, it chooses some system of linear equations, and then queries its oracle and checks that the answers it gets satisfy the system. We say that V checks at most ℓ equations if and only if on every sequence of coin tosses, the system that V chooses contains at most ℓ equations.

The following Lemma allows us to reduce the number of equations that a linear verifier checks to one, while roughly preserving the parameters of the CWP.

Lemma 6.22. *Let C be a (q, ε, r)-CWP with a linear verifier V that checks at most ℓ equations. Then, V can be transformed in polynomial time to a linear verifier V' that checks at most* one *equation with respect to which C is a CWP with query complexity q, rejection ratio $\Omega(\varepsilon)$ and randomness complexity $r + O(\ell)$.*

Proof The proof strings of C with respect to V' are defined to be the same as its proof strings with respect to V. The verifier V' is defined as follows: When given oracle access to any oracle, V' first emulates V to find the system of linear equations that V checks, and chooses a linear combination of those linear equations uniformly at random. Then, V' makes the same queries as V, and checks that the answers it gets satisfy the linear combination of the equations. Clearly, V' checks at most one equation. The query complexity and randomness complexity of V' are obvious from its description, and it is not hard to prove that the V' has rejection ratio $\frac{|\mathbb{F}|-1}{|\mathbb{F}|} \cdot \varepsilon$. ■

6.4.3 Maintaining linear proofs

The proof of [D07, Theorem 9.1] consists of two main steps:

1. First, the rejection ratio of the CWP is increased at the expense of increasing the proof alphabet's size. This step is called "Graph Powering"[2].

2. Next, PCP composition is applied to the CWP in order to reduce the size of the proof alphabet.

The reason that the resulting CWP does not necessarily have linear proofs is the use of the PCP composition technique. In order to solve this problem, we observe that if the original CWP has a linear verifier, then we can avoid the use of PCP composition and reduce the proof alphabet size by concatenating the proof strings with the $|\mathbb{F}|$-ary Hadamard code. The concatenation with the $|\mathbb{F}|$-ary Hadamard preserves the linearity of the proof strings, as desired. The reason we can use concatenation with the $|\mathbb{F}|$-ary Hadamard instead of PCP composition is that we can use the self-correction property of the the $|\mathbb{F}|$-ary Hadamard (Lemma 6.21) to emulate the action of a linear verifier on symbols of the big alphabet. Details follow.

We begin by making the following definitions:

Definition 6.23. Let C be a CWP with proof alphabet $\mathbb{F}^e$. We say that C has $\mathbb{F}$-linear proofs if it has linear proofs when viewing its proof strings as vectors over $\mathbb{F}$ rather than $\mathbb{F}^e$. In such a case we say that a matrix P is the proof matrix of C if P is the proof matrix of C as per Definition 6.6 when viewing its proof strings as vectors over $\mathbb{F}$.

Definition 6.24. Let C be a CWP with a verifier V and proof alphabet $\mathbb{F}^e$. We say that V is an $\mathbb{F}$-linear verifier if it treats the answers to its queries as vectors over $\mathbb{F}$ and checks that their elements satisfy linear equations over $\mathbb{F}$. Alternatively, we can view C as a CWP with proof alphabet $\mathbb{F}$, and view V as querying e symbols over $\mathbb{F}$ whenever it needs to query a symbol in $\mathbb{F}^e$. We say that V is an $\mathbb{F}$-linear verifier if it is a linear verifier when taking the latter view.

[2] Actually, the proof of [D07] divides this step into two different steps, called "Preprocessing" and "Graph Powering". We ignore this minor technical issue.

For an example of the two latter notions, note that if apply Lemma 6.18 to a CWP C (with proof alphabet $\mathbb{F}$) that has linear proofs and a linear verifier, the result is a CWP with $\mathbb{F}$-linear proofs and $\mathbb{F}$-linear verifier.

The following lemma summarizes the properties of the Graph Powering step that are relevant to us:

Lemma 6.25 ("Graph Powering", implicit in [D07, Theorem 9.1])**.** *There exists $d \in \mathbb{N}$ such that for every $t \in \mathbb{N}$ the following holds: Let C be a CWP with verifier V, query complexity 2, proof alphabet $\mathbb{F}^e$, rejection ratio ε and randomness complexity r. Then there exists a verifier V_t with respect to which C is a CWP such that*

1. *C has rejection ratio $\Omega\left(\min\left\{\sqrt{t}\cdot\varepsilon, 1/t\right\}\right)$, proof alphabet $\mathbb{F}^{e\cdot d^{t/2}}$, randomness complexity $r + O(t)$ and query complexity 2.*
2. *If C has linear proofs with respect to V then it has $\mathbb{F}$-linear proofs with respect to V_t.*
3. *If V is a linear verifier then V_t is an $\mathbb{F}$-linear verifier.*

Furthermore, V_t can be computed in polynomial time from V. Finally, if C has linear proofs with respect to V then the proof matrix of C with respect to V_t can be computed in polynomial time from the verifier V, the generating matrix of C and the proof matrix of C with respect to V.

We mention that [D07] only deals with Item 1, but it is easy to verify that Items 2 and 3 hold for her construction. We can now prove a "linear version" of the gap amplification theorem for CWPs (Theorem 6.17), by replacing PCP composition by concatenation with the $|\mathbb{F}|$-ary Hadamard:

Theorem 6.26. *There exist constants $q_0 \geq 3$ and $\varepsilon_0 > 0$ such that the following holds: Let C be a (q_0, ε, r)-CWP that has* linear proofs *and a* linear verifier *V. Then there exists a* linear verifier *V' with respect to which C is a CWP with* linear proofs*, rejection ratio at least $\min\{2\varepsilon, \varepsilon_0\}$, randomness complexity $r + O(1)$, and query complexity q_0. Furthermore, there exists a polynomial time algorithm that, when given as input the generating matrix of C, the verifier V, and the proof matrix of C with respect to V, outputs the verifier V' and the proof matrix of C with respect to V'.*

Proof Idea In order to prove the theorem, we first apply Graph Powering to C with a large parameter t. This increases the rejection ratio of C and also increases its proof alphabet. Let $\mathbb{F}^{e'}$ and V_t denote the proof alphabet and verifier of C after the Graph Powering. The challenge is to reduce the proof alphabet of C back to $\mathbb{F}$ while not decreasing its rejection ratio by too much.

In order to do so, we concatenate the proof strings that are over the alphabet $\mathbb{F}^{e'}$ with the $|\mathbb{F}|$-ary Hadamard code. We then observe that the fact that V_t is an $\mathbb{F}$-linear verifier implies that, while an answer to a query of V_t is a symbol in $\mathbb{F}^{e'}$, the verifier V_t only needs to know a linear combination of this answer in order to decide whether to accept or not. Thus, if we have oracle access to the $|\mathbb{F}|$-ary Hadamard encoding of each of the answers to V_t's queries, or even to a slightly corrupted encoding, then we can use the self-correction of the $|\mathbb{F}|$-ary

Hadamard (Lemma 6.21) to obtain the linear combinations that V_t needs and use them to emulate the operation of V_t. We can also make sure that we are indeed given access to a (slightly corrupted) $|\mathbb{F}|$-ary Hadamard encoding, using the local testability of the $|\mathbb{F}|$-ary Hadamard (Lemma 6.20).
It follows that we can emulate the operation of V_t on the proof strings that were concatenated with the $|\mathbb{F}|$-ary Hadamard, and it can be shown that this emulation does not decrease the rejection ratio of C by too much. By choosing t to be a sufficiently large constant, we get the desired parameters. Details follow.

Proof Let q_0 and t be some large enough integers to be fixed later, and let ε_0 be some constant to be fixed later. Let C be a (q_0, ε, r)-CWP that has linear proofs and a linear verifier. We begin by reducing the query complexity of C to 2 using Lemma 6.18, and then applying the Graph Powering (Lemma 6.25) with the parameter t. Let V_t denote the resulting verifier. With respect to V_t, the CWP C has proof alphabet $\mathbb{F}^{q_0 \cdot d^t}$, query complexity 2 and rejection ratio $\Omega\left(\min\left\{\sqrt{t}\varepsilon, 1/t\right\}\right)$.

Observe that the verifier V_t is an $\mathbb{F}$-linear verifier. We can assume without loss of generality that V_t checks at most $2 \cdot q_0 \cdot d^t$ linear equations (over $\mathbb{F}$), since its queries contain at most $2 \cdot q_0 \cdot d^t$ values in $\mathbb{F}$. We modify V_t so it checks at most one linear equation (by using Lemma 6.22), and obtain a new verifier V_t'. Observe that V_t' is an $\mathbb{F}$-linear verifier and that C with respect to V_t' has $\mathbb{F}$-linear proofs.

We view the verifier V_t' as follows: Recall that C with respect to V_t' has proof alphabet $\mathbb{F}^{q_0 \cdot d^t}$. For simplicity, we assume that the answers to the queries of V_t' are always elements of $\mathbb{F}^{q_0 \cdot d^t}$, where the symbols of the tested string are embedded into $\mathbb{F}^{q_0 \cdot d^t}$ using a trivial linear embedding. On a sequence of coins tosses ω, the verifier V_t' first chooses two vectors $v_1^\omega, v_2^\omega \in \mathbb{F}^{q_0 \cdot d^t}$ that depend only on ω. The verifier V_t' then makes two queries to its oracles, receiving answers $x_1, x_2 \in \mathbb{F}^{q_0 \cdot d^t}$, and accepts if and only if $\langle v_1^\omega, x_1 \rangle + \langle v_2^\omega, x_2 \rangle = 0$.

We are now ready to define the verifier V'. We define the proof strings of V' to be the concatenation of the proof strings of V_t' with the $|\mathbb{F}|$-ary Hadamard. That is, for every codeword c whose proof string with respect to V_t' is π_t, the proof string π' of c with respect to V' is a string over $\mathbb{F}$ that consists of the $|\mathbb{F}|$-ary Hadamard encoding of every symbol of π_t (recall that every such symbol is an element of $\mathbb{F}^{q_0 d^t}$). Observe that C with respect to V' has linear proofs, since it has $\mathbb{F}$-linear proofs with respect to V_t'.

Given oracle access to a tested string w and to a proof string π', the verifier V' acts as follows:

1. The verifier V' first emulates V_t' on a sequence of coins ω. Let i_1^ω and i_2^ω be the coordinates of that V_t' queries on coins ω, and assume that both i_1^ω and i_2^ω belong to the proof string (the other cases can be handled similarly). The verifier V' finds i_1^ω and i_2^ω and also the vectors v_1^ω and v_2^ω defined above.

2. Let a_1 and a_2 denote the blocks of π' that are supposed to be $|\mathbb{F}|$-ary Hadamard encoding of the coordinates i_1^ω and i_2^ω. The verifier V' uses the local testability of the $|\mathbb{F}|$-ary Hadamard (Lemma 6.20) to check that a_1 and a_2 are close to legal codewords of the $|\mathbb{F}|$-ary Hadamard, and rejects otherwise.

3. Let $x_1, x_2 \in \mathbb{F}^{q_0 \cdot d^t}$ be such that a_1 and a_2 are close to $H(x_1)$ and $H(x_2)$ respectively. The verifier V' uses the self-correction of the $|\mathbb{F}|$-ary Hadamard (Lemma 6.21) to retrieve

the inner products $\langle v_1^\omega, x_1 \rangle$ and $\langle v_2^\omega, x_2 \rangle$ and accepts if and only if

$$\langle v_1^\omega, x_1 \rangle + \langle v_2^\omega, x_2 \rangle = 0$$

Note that the verifier V' needs to make only a constant number of queries in order to use local testing and self-correction of the $|\mathbb{F}|$-ary Hadamard, and in particular its query complexity is a constant that does not depend on t and on the original query complexity of V. We can therefore fix q_0 to be the query complexity of V'. Furthermore, it is not hard to prove that the rejection ratio of V' is at least a constant factor times the rejection ratio of V'_t, so V' has rejection ratio of at least $\Omega\left(\min\left\{\sqrt{t}\cdot\varepsilon, 1/t\right\}\right)$. Now, fix t to be a large enough constant so that V' has rejection ratio $\min\{2\varepsilon, 1/t\}$, and set ε_0 to be $1/t$. Finally, note that V' has randomness complexity $r + O(1)$ (where the constant in the Big-O notation depends on q_0 and t). It follows that C with respect to V' is a CWP with the required parameters. ■

Remark 6.27. Note that the reason we needed to reduce the number of equations that V_t checks (Lemma 6.22) is as follows: If we had not applied Lemma 6.22 to V_t, the verifier V' would have needed to check that $2 \cdot q_0 \cdot d^t$ linear equations are satisfied, and thus its query complexity would have become at least $2 \cdot q_0 \cdot d^t$. However, we need the query complexity of V' to be independent of t and of the initial query complexity.

We can now apply Theorem 6.26 to the CWPs of Theorem 6.13 for $O(\log\log k)$ times and obtain the following theorem:

Theorem 6.28. *For every constant $c > 0$ the following holds: There exists an infinite family of CWPs $\{C_k\}_k$ such that C_k has block length $O(k)$, relative distance $\Omega(1)$, query complexity $O(1)$, rejection ratio $\Omega(1)$, and randomness complexity $\log k + O(\log\log k)$. Furthermore, the codes in the family are linear and there exists a probabilistic algorithm that on input k, runs in time $\mathrm{poly}(k)$ and outputs with probability $1 - \exp(-\Omega(\log^c k))$ the generating matrix and verifier circuit of C_k.*

Using Theorem 6.7, we obtain the following LTCs:

Theorem 6.29. *There exists a constant $\delta > 0$ such that for every two constants $c > 0$ and $\tau > 0$ the following holds: There exists an infinite family of LTCs $\{C_k\}_k$ such that C_k has block length $k \cdot \mathrm{poly}(\log k)$, relative distance δ, query complexity $O(1)$, distance threshold τ, and rejection probability $\Omega(1)$. Furthermore, the codes in the family are linear and there exists a probabilistic algorithm that on input k, runs in time $\mathrm{poly}(k)$ and outputs with probability $1 - \exp(-\Omega(\log^c k))$ the generating matrix and verifier circuit of C_k.*

Remark 6.30. Note that replacing the PCP composition technique with the code concatenation technique allows us to avoid using PCP machinery in this part of our construction. Thus, changing the proof of [D07] serves our goal of constructing LTCs without using PCP machinery, in addition to preserving the linearity of the proof strings.

6.5 The efficiency of our verifiers

So far we have ignored the issue of the *efficiency* of the verifier circuits of our CWPs and LTCs. Clearly, since those circuits are produced in time poly (k), their size is at most poly (k). However, some previous constructions of LTCs have verifiers that are Turing machines that run in time poly $(\log k)$, so it is natural to ask whether the verifiers of our construction can be that efficient.

The most straightforward way to formalize the latter question is to ask whether our verifiers can be circuits of size poly $(\log k)$. The answer to this question is negative: To see why, consider our main construction of CWPs, and consider the effect of the random projection operation at the last iteration of this construction. At the last iteration, the random projection operation chooses a set S of size $\Omega(k)$, and projects the code to the set S. Clearly, in order to allow the verifier to verify membership in the projected code, we will have to hardwire the set S into our verifier, but this means that the verifier must be of size at least $\Omega(k)$.

The foregoing argument shows that we can not hope to have verifier circuits of size poly $(\log k)$. However, observe that what this argument actually shows is not that our verifiers are inefficient, but rather that they have to use a large amount of non-uniformity. Thus, we may still obtain efficient verifiers, but in order to do so, we have to give up the use of circuits as a computational model for our verifiers, and instead use a model that measures their running time separately from the amount of non-uniformity they use. A computational model that is usually used for such purposes is "machines that take advice": A machine that takes advice is a machine that when invoked on input x, may use an additional string $a_{|x|}$ that depends only on the length of x and not on x itself. It is a known fact that such machines are equivalent families of circuits. The advantage of this model is that it allows us to measure the amount of non-uniformity directly (by considering the length of $a_{|x|}$), and also allows us to analyze the running time of the machine separately from the amount of non-uniformity.

However, this model still does not solve our problem: A machine that takes advice has to run in time that lower-bounded by its advice length, since the machine has to read all of its advice tape in order to use it. In order to solve this problem and allow the running time of the machines to be smaller than the length of the advice, we change the computational model so the machines will have *oracle access* to their advice string. This way, the machines can read only the part of the advice that they need, without having to read all the advice tape.

Now, it turns out that in this model, our verifiers run in time poly $(\log k)$ due to a simple observation: In this model of verifiers, any verifier with query complexity $q(k)$, randomness complexity $r(k)$, block length $n(k)$ and proof length $m(k)$ can be modified to run in time that is polynomial in $\log n(k)$, $q(k)$, $r(k)$, and $\log m(k)$. The latter modification can be done by providing the verifier with the queries it should make, as well as with the truth-table of the predicate it should compute on the answers to its queries, in the advice string. We provide the full details of this modification, as well as a formal definition of the model, in a separate note [M08].

We conclude that in an appropriate choice of model, our verifiers run in time poly $(\log k)$, as required.

7 Discussion and open problems

7.1 Variants of our construction

Integrating the query reduction and gap amplification into the main construction We point out a possible variant of our construction. We recall the structure of our construction: Our main construction is iterative, where each iteration increases the query complexity and decreases the proof rate and rejection ratio by constant factors. By applying $O(\log \log k)$ iterations, the main construction yields a CWP with query complexity $\text{poly}(\log k)$, rejection ratio $1/\text{poly}(\log k)$ and proof rate $1/\text{poly}(\log k)$. Then, we apply query reduction and gap amplification techniques to reduce the query complexity to $O(1)$ and to increase the rejection ratio of $\Omega(1)$.

Observe that, instead of applying the query reduction and the gap amplification *after* the main construction, we could have integrated those operations into the iterations of our main construction. That is, we could have added the query reduction and the gap amplification to the iteration as two additional basic operations (in addition to the Tensor Product, Random Projection and Distance Amplification), and use them to maintain the query reduction and the rejection ratio of the CWP. In such case, each iteration would have maintained the query complexity and the rejection ratio of the CWP, and would have decreased the proof rate of the CWP by a larger constant factor, which we could still afford. After applying $O(\log \log k)$ iterations, this construction would have yielded with a CWP with query complexity $O(1)$, rejection ratio $\Omega(1)$ and proof rate $1/\text{poly}(\log k)$, like our original construction.

A simple construction of LTCs of block length $\text{poly}(k)$ We point out that using the query reduction and gap amplification techniques of Section 6 (Theorems 6.11 and 6.26) one can obtain a relatively simple construction of LTCs of block length $\text{poly}(k)$: The construction starts with any linear code C of message length k and block length $\text{poly}(k)$, and views it as a CWP with the trivial verifier. With respect to this verifier, C is a CWP with query complexity $\text{poly}(k)$, rejection ratio 1 and randomness complexity 0. The construction then applies the query reduction theorem (Theorem 6.11) to C, resulting in a CWP with query complexity 3, rejection ratio $1/\text{poly}(k)$ and randomness complexity $O(\log k)$. Finally, the construction applies the gap amplification theorem (Theorem 6.26) to C for $O(\log k)$ times, resulting in a CWP with query complexity $O(1)$, rejection ratio $\Omega(1)$ and randomness complexity $O(\log k)$. By transforming this CWP into an LTC (Theorem 6.7), we obtain an LTC with block length $\text{poly}(k)$, query complexity $O(1)$, arbitrarily small constant distance threshold, and rejection probability $\Omega(1)$.

7.2 The connection to the construction of Ben-Sasson and Sudan

The construction presented in this work was inspired by the work of Ben-Sasson and Sudan [BS05], and was obtained by trying to imitate their construction without using algebraic techniques. In order to make the connection between the two constructions more apparent, we need to present the construction of [BS05] differently than the way it is presented in their original paper. In this subsection, we describe this alternative presentation of the construction of [BS05], and discuss the similarities and differences between the their construction

and our construction. We begin the with defining the Reed-Solomon code:

Definition 7.1. Let $\mathbb{K}$ denote a finite field, let $S \subseteq \mathbb{K}$ and let $d < |S|$ denote a natural number. The Reed-Solomon code $\mathrm{RS}_{\mathbb{K},S,d} : \mathbb{K}^{d+1} \to \mathbb{K}^{|S|}$ is defined as follows: Suppose we wish to encode a message $a \in \mathbb{K}^{d+1}$ with $\mathrm{RS}_{\mathbb{K},S,d}$. We define the polynomial $P_a(X) \stackrel{\text{def}}{=} \sum_{i=0}^{d} a_i X^i$, and set the codeword $\mathrm{RS}_{\mathbb{K},S,d}(a)$ to consist of the evaluations of P_a at each of the elements of S. The relative distance of $\mathrm{RS}_{\mathbb{K},S,d}$ is $1 - \frac{d+1}{|S|}$ (see [S01, Lecture 4]).

The result of [BS05] we are interested in states that certain Reed-Solomon codes are CWPs. Specifically, the work of [BS05] proves the following theorem.

Theorem 7.2 ([BS05, Theorem 4])**.** *Let $\mathbb{K} = \mathrm{GF}(2^\ell)$ and let $L \subseteq \mathbb{K}$ be a $\mathrm{GF}(2)$-linear subspace of $\mathbb{K}$. Then for any $d < |L|$ the code $\mathrm{RS}_{\mathbb{K},L,d}$ is a CWP with query complexity $O(1)$, rejection ratio $1/\operatorname{poly}(\log|L|)$, randomness complexity $\log|L| + O(\log\log|L|)$ and proof length $|L| \cdot \operatorname{poly}(\log|L|)$.*

Note that by choosing $d = O(|L|)$ in Theorem 7.2, one gets a CWP with the same parameters as the CWPs we constructed in Theorem 6.13, albeit over an alphabet of a super-constant size (since a Reed-Solomon code of block length n must be over an alphabet of size at least n).

Much like our main construction, the construction of Theorem 7.2 is an iterative construction. The construction starts with an RS code with constant *block length* (rather than message length), and increases the block length in iterations while maintaining the other parameters. Specifically, in every iteration:

1. The block length is squared.
2. The rate, relative distance, and query complexity remain the same.
3. The proof rate and the rejection ratio are decreased by a constant factor.

Note that the effect of a single iteration of [BS05] is very similar to the effect of a single iteration of our main construction (compare the above list to Table 2 at the beginning of Section 3.2). The only important difference between the iteration of [BS05] and our iteration is that our iteration increases the query complexity by a constant factor, which is the reason that we end up with a poly-logarithmic query complexity.

The iteration of [BS05] and our iteration not only share a similar effect on the parameters, but they also have a similar structure. In particular, an iteration of [BS05] can be divided into two operations:

- Tensor Product - In this operation, the construction algorithm chooses two Reed-Solomon codes C_1 and C_2 *according to a certain algebraic rule*. The algorithm then computes the tensor product $C_1 \otimes C_2$, which is the code whose codewords are the matrices whose rows are codewords of C_1 and whose columns are codewords of C_2. Ben-Sasson and Sudan show that the code $C_1 \otimes C_2$ is a CWP with respect to the row/column verifier described in Section 3.2.1. The tensor product operation is used in order to increase the message length and block length of the CWP.

- Algebraic Projection - In this operation, the construction algorithm projects $C_1 \otimes C_2$ to a subset of its coordinates that is chosen *according to a sophisticated algebraic rule*. The coordinates that are "projected out" are moved to the proof part of the CWP. The algebraic projection operation is used to increase the rate and relative distance of the CWP back to the level they were at before the tensor product operation.
  In addition to increasing the rate and the relative distance, the algebraic projection operation has another role: Recall that the final goal is obtaining a CWP which is an RS code. However, the tensor product operation results in a CWP which is not an RS code but rather a tensor product of RS codes. The algebraic projection operation transforms the CWP $C_1 \otimes C_2$ back to a CWP which is a RS code.

We now discuss few important differences between the iteration of [BS05] and our iteration.

- Unlike the random projection operation operation that we use, the algebraic projection operation is deterministic. This is the reason why the CWPs of [BS05] are explicit while our CWPs are constructed by a randomized algorithm.

- The algebraic projection operation *increases* the relative distance, while the random projection operation *decreases* the relative distance. This is the reason why we need to use the distance amplification operation while [BS05] do not.
  Note that the distance amplification is the only operation in our iteration that increases the query complexity. Indeed, the use of distance amplification is the reason that our iteration increases the query complexity while the iteration of [BS05] does not.

- Recall that in order to show that the tensor product operation preserves the local testability (i.e., applying a tensor product to a CWP yields a CWP), we needed to use a result of [BS04]. This result of [BS04] forced us to maintain the square form throughout our construction (see Section 3.3 for details). In contrast, the analysis of [BS05] relies on a result of Polishchuk and Spielman [PS94], which says roughly that the tensor product of RS codes can be tested using the row/column verifier described in Section 3.2.1. Thus, the construction of [BS05] does not need to preserve a square form.

Remark 7.3. We note that the above presentation of the construction of [BS05] is very different from the presentation of this construction in the paper of [BS05]. In particular, the above presentation views the construction as going "bottom-up", from codes of constant block length to codes of large block length, while the paper of [BS05] views the construction as going "top-down". Furthermore, the paper of [BS05] does not divide a single iteration to two separate operations, but rather views the whole iteration as consisting of a single operation.

Remark 7.4. The above presentation omits some important technical details of the construction of [BS05]. In particular, the construction of [BS05] does not use Tensor Product, but rather a "Tensor Product with some additions". However, the code that results from the "Tensor Product with some additions" operation can still be verified using the row/column verifier described in Section 3.2.1.

7.3 Open Problems

Below we list few open problems regarding locally testable codes. The first two stem from our work, while the last two are more general and are independent of our work.

An explicit combinatorial construction of LTCs While in this work we give a combinatorial construction of LTCs, our construction is randomized and therefore not entirely explicit. Giving an explicit combinatorial construction of LTCs remains an interesting open problem.

One possible approach for giving such explicit construction is derandomizing our construction. This only requires derandomizing the random projection operation we use. In order to derandomize the random projection, one needs to design a deterministic algorithm that given a code C with block length n that has good relative distance, finds a relatively small set $S \subset [n]$ such that $C_{|S}$ has good relative distance. Designing such an algorithm requires overcoming two significant obstacles:

1. Observe that for every small set $S \subset [n]$ there exists a code C with good relative distance such that $C_{|S}$ does not have a good relative distance. For example, one can take any linear code with good relative distance and permute its coordinates such that for some specific codeword most of the non-zero coordinates do not fall in the set S. Thus, the choice of the set S must somehow depend on C. In contrast, standard derandomization techniques seem to give a set S that does not depend on C, which means that we need to do something non-standard and indeed C-dependent.

2. The problem of approximating the relative distance of a linear code is $\mathcal{NP}$-hard. Thus, even if we are given a set S, it is not clear that we can check whether $C_{|S}$ has a good relative distance. In contrast, many derandomization techniques require the ability to check whether a candidate object is good.

It is interesting to note that the construction of [BS05] overcomes both obstacles by ensuring that the code to which the projection is applied has a certain algebraic structure. The construction of [BS05] then uses the structure of the code to find a good subset of coordinates to which the code can be projected. Thus, it may be possible to derandomize the random projection operation by first changing our construction so that the code C has a certain combinatorial structure, and then using the combinatorial structure to find the set S.

Simplifying our construction Recall that in order to show that the Tensor Product operation preserves the local testability of CWPs, we had to use a result of [BS04], which required us to maintain a "square form" throughout our construction. The need to maintain the "square form" makes our construction more complicated than the simplified construction outlined in Section 3.2. It is not clear whether this complication is really required.

In particular, it is possible that simplified construction outlined in Section 3.2 yields good CWPs, but we do not know how to prove it. To be more specific, recall the simplifying assumption that was made in the construction of Section 3.2. This assumption says that if a code C is a CWP then C^2 is a CWP with respect to the row/column verifier. While it is known that this assumption does not hold for *every* CWP C (see [D07, V05, GM07a]), it is

possible that it holds for the CWPs that result from our construction. If this is indeed the case, it will be possible to replace our construction by the simpler construction outlined in Section 3.2. We suggest checking this possibility as an additional open problem.

Strong locally testable codes Recall that a locally testable code C is said to be strong if it has a verifier that rejects *any* non-codeword w with a probability that is proportional to the relative distance of w from C (see Definition 5.10 for details). Our construction does not yield strong LTCs because the transformation from CWPs to LTCs loses the strong rejection property. Thus, it remains an open problem to give a combinatorial construction of strong LTCs. It seems to us that such a construction will have to be very different from our construction, since it will not be able to use CWPs.

We mention that, unlike constructions of non-strong LTCs, the best known construction of strong LTCs achieves block length of $k^{1+\log^{-1/2+\varepsilon}}$ (for every $\varepsilon > 0$), and relies heavily on algebra and PCP machinery (see [GS02, Sections 3 and 5]). Thus, a combinatorial construction of strong LTCs with block length $k^{1+o(1)}$ (or even $\mathrm{poly}(k)$) will be very interesting. One might also consider the open problem of giving a (not necessarily combinatorial) construction of strong LTCs that have a block length $k \cdot \mathrm{poly}(\log k)$.

Shorter Locally Testable Codes Another interesting open problem is constructing LTCs whose block length is shorter than $k \cdot \mathrm{poly}(\log k)$. Alternatively, one can also try to give lower bounds for LTCs and show that any LTC must have block length of at least $k \cdot \log k$.

Acknowledgement. The author would like to thank his adviser, Oded Goldreich, who refused to co-author this paper, for introducing him to the subject and for many valuable discussions and ideas. The author would also like to thank Madhu Sudan for valuable discussions, and in particular for suggesting the query reduction technique described in Section 6.3, which simplified this work considerably.

References

[A94] S. Arora, *Probabilistic checking of proofs and hardness of approximation problems*, Ph.d. thesis, University of California at Berkeley, 1994.

[ABNNR92] N. Alon, J. Bruck, J. Naor, M. Naor, and R. Roth, *Construction of asymptotically good low rate error-correcting codes through pseudo-random graphs*, IEEE Transactions on Information Theory 38, 1992, pages 509–516.

[ALMSS98] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, *Proof verification and intractability of approximation Problems*, Journal of ACM, Volume 45(3), 1998, pages 501-555. Preliminary version in FOCS, 1992, pages 14-23.

[AS98] S. Arora and S. Safra, *Probabilistic Checkable Proofs: A new characterization of NP*, Journal of ACM volume 45(1), 1998, pages 70-122. Preliminary version in FOCS 1992, pages 2-13.

[BHLM07] E. Ben-Sasson, P. Harsha, O. Lachish and A. Matsliah, *Sound 3-query PCPPs are Long*, ECCC TR07-127, 2007.

[BHR05] E. Ben-Sasson, Prahladh Harsha, Sofya Raskhodnikova, *Some 3-CNF properties are hard to test*, SIAM Journal on Computing 35(1), 2005, pages 1-21. Preliminary version in STOC 2003, pages 345-354.

[BLR93] M. Blum, M. Luby and R. Rubinfeld, *Self Testing/Correcting with applications to numerical problems*, Journal of Computer and System Science, Volume 47(3), 1993, pages 549-595.

[BGHSV06] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan and S. Vadhan, *Robust PCPs of Proximity, shorter PCPs and applications to coding*, SIAM Journal of Computing 36(4), 2006, pages 889-974. Preliminary version in STOC 2004, pages 120-134.

[BS04] E. Ben-Sasson and M. Sudan, *Robust locally testable codes and products of codes*, Random Structurs and Algorithms 28(4), 2006, pages 387-402. Preliminary version in APPROX-RANDOM 2004, pages 286-297.

[BS05] E. Ben-Sasson and M. Sudan, *Simple PCPs with poly-log rate and query complexity*, SIAM Journal on Computing 38(2), 2008, pages 551-607. Preliminary version in STOC 2005.

[CR05] D. Coppersmith and A. Rudra, *On the robust testability of tensor products of codes*, ECCC TR05-104, 2005.

[D07] I. Dinur, *The PCP Theorem by gap amplification*, Journal of ACM 54(3), 2007. Preliminary version in STOC 2006, pages 241-250.

[DR06] I. Dinur and O. Reingold, *Assignment testers: Towards combinatorial proofs of the PCP theorem*, SIAM Journal of Computing 36(4), 2006, pages 975-1024. Preliminary version in FOCS 2004, pages 155-164.

[DSW06] I. Dinur, M. Sudan and A. Wigderson, *Robust local testability of tensor products of LDPC codes*, APPROX-RANDOM 2006, pages 304-315.

[FRS94] L. Fortnow, J. Rompel and M. Sipser. *On the power of multi-prover interactive protocols*, Theoretical Computer Science 134(2), 1994, 545-557. Preliminary version in 3rd IEEE Symp. on Structure in Complexity Theory 1988.

[FS95] K. Friedl and M. Sudan, *Some improvements to total degree tests*, Israel Symposium on Theory of Computing and Systems (ISTCS) 1995, pages 190-198.

[G05] O. Goldreich, *Short locally testable codes and proofs (Survey)*, ECCC TR05-014, 2005.

[G52] E. N. Gilbert, *A comparision of signalling alphabets*, Bell System Technical Journal 31, 1952, pages 504-522.

[GM07a] O. Goldreich and O. Meir, *The tensor product of two good codes is not necessarily locally testable*, ECCC TR07-062, 2007.

[GM07b] O. Goldreich and O. Meir, *A small gap in the gap amplification of assignment testers*, Comment 3 on ECCC TR05-46, 2007.

[GS02] O. Goldreich and M. Sudan, *Locally testable codes and PCPs of almost linear length*, Journal of ACM 53(4), 2006, pages 558-655, Preliminary version in FOCS 2002, pages 13-22.

[HLW06] S. Hoory, N. Linial and A. Wigderson, *Expander graphs and their applications*, Bulletin of AMS, 43(4), 2006, pages 439-561.

[KT00] J. Katz and L. Trevisan, *On the efficiency of local decoding procedures for error correcting codes*, STOC 2000, pages 80-86.

[KS07] T. Kaufman and M. Sudan, *Sparse random linear codes are locally decodable and testable*, FOCS 2007, pages 590-600.

[M08] O. Meir, *On the efficiency of non-uniform PCPP verifiers*, ECCC TR08-064.

[PS94] A. Polishchuk and D.A. Spielman, *Nearly-linear size holographic proofs*, STOC 1994, pages 194-203.

[RVW00] O. Reingold, S. Vadhan and A. Wigderson, *Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors*, Annals of Mathematics 155(1), 2002, pages 157-187. Preliminary version in FOCS 2000.

[RS96] R. Rubinfeld and M. Sudan, *Robust characterization of polynomials with applications to program testing*, SIAM Journal on Computing 25(2), 1996, pages 252-271.

[S95] D. Spielman, *Computationally efficienterror-correcting codes and holographic proofs*, Ph.d. thesis, MIT, 1995.

[S01] M. Sudan, *Algorithmic introduction to coding theory*, *Lecture notes*. Available from `http://theory.csail.mit.edu/~madhu/FT01/`, 2001.

[V05] P. Valiant, *The tensor product of two codes is not necessarily robustly testable*, APPROX-RANDOM 2005, pages 472-481.

[V57] R. R. Varshamov, *Estimate of the number of signals in error correcting codes*, Doklady Akadamii Nauk 117, 1957, pages 739-741.

[Z71] V. V. Zybalov, *An estimate on the complexity of constructing binary linear cascade codes*, Problems of Information Transmission 7(1), 1971, pages 3-10.

A Zybalov Bound

In this appendix, we review a special case of the Zybalov bound, which gives an explicit construction of an infinite family of codes over the alphabet $\mathbb{F}$ that has relative distance arbitrarily close to $\frac{|\mathbb{F}|-1}{|\mathbb{F}|}$ and a constant rate. For more details, the reader is referred to

[S01, Lectures 5 and 6]. The construction of the Zybalov bound starts with two codes, the Reed-Solomon code and the Gilbet-Varshamov bound, and shows that, while each of those codes falls short of achieving the desired properties, their concatenation does achieve them. We begin with recalling the definition of the Reed-Solomon code, defined in Section 7:

Definition (Definition 7.1, restated)**.** Let $\mathbb{K}$ denote a finite field, let $S \subseteq \mathbb{K}$ and let $d < |S|$ denote a natural number. The Reed-Solomon code $\mathrm{RS}_{\mathbb{K},S,d} : \mathbb{K}^{d+1} \to \mathbb{K}^{|S|}$ is defined as follows: Suppose we wish to encode a message $a \in \mathbb{K}^{d+1}$ with $\mathrm{RS}_{\mathbb{K},S,d}$. We define the polynomial $P_a(X) \stackrel{\text{def}}{=} \sum_{i=0}^{d} a_i X^i$, and set the codeword $\mathrm{RS}_{\mathbb{K},S,d}(a)$ to consist of the evaluations of P_a at each of the elements of S. The relative distance of $\mathrm{RS}_{\mathbb{K},S,d}$ is $1 - \frac{d+1}{|S|}$ (see [S01, Lecture 4]).

The Reed-Solomon code constitutes an explicit infinite family of codes that has constant rate and relative distance, as we desire. The problem is that this family has alphabet of non-constant size, and in particular much larger than $\mathbb{F}$. To see it, observe that if a Reed-Solomon code $\mathrm{RS}_{\mathbb{K},S,d}$ has block length n, then its alphabet must be of size $|\mathbb{K}| \geq |S| = n$.

We turn to introduce the Gilbert-Varshamov codes. Let $q \stackrel{\text{def}}{=} |\mathbb{F}|$. We use the following definitions:

Definition A.1. The q-ary entropy function $H_q : (0,1) \to [0,1]$ is defined as follows:

$$H_q(p) = p \cdot \log_q \frac{1}{p} + (1-p) \cdot \log_q \frac{1}{1-p}$$

Definition A.2. Let $r \leq n$ be natural numbers and let $v \in \mathbb{F}^n$. The q-ary Hamming ball with center v and radius r is defined to be

$$B(v,r) = \{u \in \mathbb{F}^n : \delta(u,v) \leq r/n\}$$

Observe that $|B(v,r)| = \sum_{i=0}^{r} \binom{r}{i}(q-1)^i$. Using Stirling's formula, one can show that when n goes to infinity we have that

$$\log_q |B(v,r)| \approx \left(H_q(r/n) + (r/n) \cdot \log_q(q-1)\right) \cdot n$$

.

The Gilbert-Varshamov bound uses the probabilistic method to give a non-explicit construction of an infinite family of codes $\{\mathrm{GV}_k\}_k$ over the alphabet $\mathbb{F}$ that has constant rate and relative distance.

Theorem A.3 (Gilbert-Varshamov Bound, due to [G52, V57])**.** *For any constant $\delta \in \left(0, \frac{q-1}{q}\right)$ and for any $\varepsilon > 0$ there exists an infinite family of linear codes $\{\mathrm{GV}_k\}_k$ over the alphabet $\mathbb{F}$ that has relative distance δ and rate $R = 1 - H_q(\delta) - \delta \cdot \log_q(q-1) - \varepsilon$.*

Proof Let $\delta \in \left(0, \frac{q-1}{q}\right)$ and let $R = 1 - H_q(\delta) - \delta \cdot \log_q(q-1) - \varepsilon$. Fix some large enough message length k and let $n = k/R$. Let G be a $k \times n$ matrix over $\mathbb{F}$ chosen uniformly at random and define GV_k to be the code generated by G, that is, $\mathrm{GV}_k(x) = x \cdot G$ for every $x \in \mathbb{F}^k$. We show that GV_k has relative distance δ with non-zero probability, and this will imply the theorem.

Fix some message $x \in \mathbb{F}^k$. It is easy to see that $\mathrm{GV}(x) = x \cdot G$ is uniformly distributed in $\mathbb{F}^n$. The probability over the choice of G that $C(x)$ has weight less than δ equals the probability that $C(x) \in B(0, \delta n)$ and is therefore at most

$$\frac{|B(0,\delta n)|}{q^n} \approx q^{\left(H_q(\delta)+\delta\cdot\log_q(q-1)-1\right)\cdot n} = q^{-(R+\varepsilon)\cdot n}$$

By taking union bound over all possible messages $x \in \mathbb{F}^k$, we get that the probability that GV_k does not have relative distance δ is at most

$$q^{k-(R+\varepsilon)\cdot n} = q^{R\cdot n-(R+\varepsilon)\cdot n} = q^{-\varepsilon n} < 1$$

It follows that GV_k has relative distance δ with non-zero probability, as required. ■

The problem with the codes of the Gilbet-Varshamov Bound is, of course, that they are non-explicit. However, observe that a generating matrix of a Gilbert-Varshamov code can be computed in exponential time, by going over all possible matrices. The Zybalov bound uses this observation to construct a code of message length k as follows: First, we compute the generating matrix of an RS code of message length $\frac{k}{\log_q O(k)}$ over alphabet of size $O(k)$. Then, we compute the generating matrix of a GV code of message length $\log_q O(k)$, and note that this can be done in time polynomial in k. Finally, we concatenate the two codes, taking the RS code to be the outer code and the GV code to be the inner code. This results in the following family of codes:

Theorem A.4 (Zybalov Bound, due to [Z71])**.** *For any constant $\delta \in \left(0, \frac{q-1}{q}\right)$ and for every $\varepsilon > 0$ there exists an infinite family of linear codes $\{Z_k\}_k$ over the alphabet $\mathbb{F}$ that has relative distance δ and rate*

$$R = \max_{\delta<\delta_2<\frac{|\mathbb{F}|-1}{|\mathbb{F}|}} \left\{ (1 - H_q(\delta_2) - \varepsilon)\left(1 - \frac{\delta}{\delta_2}\right)\right\}$$

Furthermore, there exists an algorithm that on input k, runs in time $\mathrm{poly}(k)$ *and outputs the generating matrix of Z_k.*

B Local testability of the $|\mathbb{F}|$-ary Hadamard

We recall the definition of the $|\mathbb{F}|$-ary Hadamard code:

Definition (Definition 6.19, restated)**.** The $|\mathbb{F}|$-ary Hadamard code, denoted H, encodes a messages $x \in \mathbb{F}^k$ by the codeword $H(x) \in \mathbb{F}^{|\mathbb{F}|^k}$ defined by

$$H(x)_i = \langle x, i\rangle \text{ for every } i \in \mathbb{F}^k$$

That is, $H(x)$ consists of the inner products of $x \in \mathbb{F}^k$ with all the vectors in $\mathbb{F}^k$.

In this appendix we prove Lemma 6.20, which stated that the $|\mathbb{F}|$-ary Hadamard code is strong locally testable (see Definition 5.10).

Lemma (Lemma 6.20, restated)**.** *The $|\mathbb{F}|$-ary Hadamard code is strong locally testable with query complexity 3, rejection ratio 1/6 and randomness complexity $2(k+1)\log|\mathbb{F}|$.*

The proof presented here is a variant of the analysis of [BLR93]. Fix some message length k. For convenience, we view strings in $\mathbb{F}^{|\mathbb{F}|^k}$ as functions from $\mathbb{F}^k$ to $\mathbb{F}$, and note that taking this view, the codewords of the $|\mathbb{F}|$-ary Hadamard are exactly the linear functions from $\mathbb{F}^k$ to $\mathbb{F}$.

We recall the verifier V of the $|\mathbb{F}|$-ary Hadamard defined in Section 6.4.2: Given oracle access to a function $f : \mathbb{F}^k \to \mathbb{F}$, the verifier V chooses two vectors $x, y \in \mathbb{F}^k$ and two scalars $a, b \in \mathbb{F}$ uniformly at random, and checks that

$$a \cdot f(x) + b \cdot f(y) = f(a \cdot x + b \cdot y)$$

The query complexity and the randomness complexity of this verifier are obvious from its definition. We turn to analyze its rejection ratio. Let $f : \mathbb{F}^k \to \mathbb{F}$ be a function such that V rejects with probability $\varepsilon < \frac{1}{6}$ when given oracle access to f. We show that $\delta_H(f) \leq 2 \cdot \varepsilon$, and this will imply the required result.

In order to prove that $\delta_H(f) \leq 2 \cdot \varepsilon$, we define a linear function $\phi : \mathbb{F}^k \to \mathbb{F}$ such that $\delta(f, \phi) \leq 2 \cdot \varepsilon$. For every two vectors $x, y \in \mathbb{F}^k$ and scalars $a, b \in \mathbb{F}$ define the vote of y, a and b regarding the value of f at x by

$$\phi_{y;a,b}(x) = a^{-1} \cdot (f(a \cdot x + b \cdot y) - f(b \cdot y))$$

We define ϕ by defining $\phi(x)$ to be the corresponding plurality vote, that is, $\phi(x)$ is defined to be the value v that maximizes the probability $\Pr_{y \in \mathbb{F}^k, a,b \in \mathbb{F}}[\phi_{y;a,b}(x) = v]$. Note that indeed $\delta(f, \phi) \leq 2 \cdot \varepsilon$, since:

$$\begin{aligned}
\varepsilon &= \Pr\left[V^f \text{ rejects}\right] \\
&= \Pr_{x,y \in \mathbb{F}^k, a,b \in \mathbb{F}}[a \cdot f(x) + b \cdot f(y) \neq f(a \cdot x + b \cdot y)] \\
&= \Pr_{x,y \in \mathbb{F}^k, a,b \in \mathbb{F}}[f(x) \neq \phi_{y;a,b}(x)] \\
&\geq \Pr_{x,y \in \mathbb{F}^k, a,b \in \mathbb{F}}[f(x) \neq \phi_{y;a,b}(x) \wedge f(x) \neq \phi(x)] \\
&= \Pr_{x,y \in \mathbb{F}^k, a,b \in \mathbb{F}}[f(x) \neq \phi_{y;a,b}(x) | f(x) \neq \phi(x)] \cdot \Pr[f(x) \neq \phi(x)] \\
&= \Pr_{x,y \in \mathbb{F}^k, a,b \in \mathbb{F}}[f(x) \neq \phi_{y;a,b}(x) | f(x) \neq \phi(x)] \cdot \delta \\
&\geq \frac{1}{2} \cdot \delta
\end{aligned}$$

Where the last inequality follows from the fact that for every $x \in \mathbb{F}^k$, if $\Pr[f(x) = \phi_{y;a,b}(x)] > \frac{1}{2}$, then by definition of $\phi(x)$ we must have $\phi(x) = f(x)$.

It remains to prove that ϕ is a linear function. We begin with proving that for every $x \in \mathbb{F}^k$ it holds that $\Pr_{y \in \mathbb{F}^k, a,b \in \mathbb{F}}[\phi_{y;a,b}(x) = \phi(x)] \geq 1 - 2 \cdot \varepsilon$. Fix $x \in \mathbb{F}^k$, and call a pair of triplets $((y_1, a_1, b_1), (y_2, a_2, b_2)) \in (\mathbb{F}^k \times \mathbb{F} \times \mathbb{F})^2$ good if

$$a_1^{-1} \cdot f(b_1 \cdot y_1) - a_2^{-1} \cdot f(b_2 \cdot y_2) = f(a_1^{-1} \cdot b_1 \cdot y_1 - a_2^{-1} \cdot b_2 \cdot y_2) \tag{4}$$

$$a_1^{-1} \cdot f(a_1 \cdot x + b_1 \cdot y_1) - a_2^{-1} \cdot f(a_2 \cdot x + b_2 \cdot y_2) = f(a_1^{-1} \cdot b_1 \cdot y_1 - a_2^{-1} \cdot b_2 \cdot y_2) \quad (5)$$

and observe that for every such a good pair of triplets is it holds that $\phi_{y_1;a_1,b_1}(x) = \phi_{y_2;a_2,b_2}(x)$.

Now, note that for a uniformly distributed pair of triplets, each of Equations 4 and 5 holds with probability at least $1 - \varepsilon$. Therefore, by applying the union bound, a uniformly distributed pair of triplets is good with probability at least $1-2\cdot\varepsilon$. By an averaging argument, there exists a triplet (y_1, a_1, b_1) such that for a uniformly distributed triplet (y_2, a_2, b_2) the pair $((y_1, a_1, b_1), (y_2, a_2, b_2)) \in (\mathbb{F}^k \times \mathbb{F} \times \mathbb{F})^2$ is good with probability at least $1-2\cdot\varepsilon$. It follows that for a uniformly chosen $y_2 \in \mathbb{F}^k$ and $a_2, b_2 \in \mathbb{F}$ it holds that $\Pr[\phi_{y_1;a_1,b_1}(x) = \phi_{y_2;a_2,b_2}(x)] \geq 1-2\cdot\varepsilon > \frac{2}{3}$, and therefore it holds that $\phi(x) = \phi_{y_1;a_1,b_1}(x)$ and $\Pr_{y\in\mathbb{F}^k,a,b\in\mathbb{F}}[\phi_{y;a,b}(x) = \phi(x)] \geq 1 - 2 \cdot \varepsilon$, as we wanted.

Finally, let $x, y \in \mathbb{F}^k$ and let $a, b \in \mathbb{F}$. We prove that $\phi(a \cdot x + b \cdot y) = a \cdot \phi(x) + b \cdot \phi(y)$. We prove it by showing, using the probabilistic method, that there exist $z \in \mathbb{F}^k$ and $c, d \in \mathbb{F}$ such that

$$a \cdot \phi(x) = c^{-1}(\phi(c \cdot a \cdot x + d \cdot z) - \phi(d \cdot z)) \quad (6)$$

$$b \cdot \phi(y) = c^{-1}(\phi(d \cdot z) - \phi(d \cdot z - c \cdot b \cdot y)) \quad (7)$$

$$\phi(a \cdot x + b \cdot y) = c^{-1}(\phi(c \cdot a \cdot x + d \cdot z) - \phi(d \cdot z - c \cdot b \cdot y)) \quad (8)$$

Observe that if such z, c and d exist then indeed $\phi(a \cdot x + b \cdot y) = a \cdot \phi(x) + b \cdot \phi(y)$, since the sum of right hand sides of Equations 6 and 7 equals the right hand side of Equation 8. To show that such z, c and d exist, observe that by substituting $z' = z - d^{-1} \cdot c \cdot b \cdot y$, $a' = c \cdot a$ and $b' = c \cdot b$, the above equations are equivalent to

$$\phi(x) = \phi_{z;a',d}(x)$$

$$\phi(y) = \phi_{z';b',d}(y)$$

$$\phi(a \cdot x + b \cdot y) = \phi_{z';c,d}(y)$$

Furthermore, observe that if z, c and d are chosen uniformly at random, then z', a' and b' are uniformly distributed. Therefore, for each of the three latter equations, the probability that it holds for a uniformly chosen z, c and d is at least $1 - 2 \cdot \varepsilon$. By the union bound, the probability that all the three equations hold is at least $1 - 3 \cdot 2 \cdot \varepsilon > 1 - 3 \cdot 2 \cdot \frac{1}{6} > 0$. This implies there exists at least one choice of z, c and d satisfying equations 6, 7 and 8, and therefore we get that $\phi(a \cdot x + b \cdot y) = a \cdot \phi(x) + b \cdot \phi(y)$, as required.

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