Relaxed Locally Correctable Codes in Computationally Bounded Channels

Jeremiah Blocki[*] Venkata Gandikota[†] Elena Grigorescu[‡] Samson Zhou[§]

April 28, 2018

Abstract

Error-correcting codes that admit *local* decoding and correcting algorithms have been the focus of much recent research due to their numerous theoretical and practical applications. The goal is to obtain the best possible tradeoffs between the number of queries the algorithm makes to its oracle (the *locality* of the task), and the amount of redundancy in the encoding (the *information rate*).

In Hamming's classical adversarial channel model, the current tradeoffs are dramatic, allowing either small locality, but superpolynomial blocklength, or small blocklength, but high locality. However, in the computationally bounded, adversarial channel model, proposed by Lipton (STACS 1994), constructions of locally decodable codes suddenly exhibit small locality and small blocklength. The first such constructions are due to Ostrovsky, Pandey and Sahai (ICALP 2007) who build private locally decodable codes under the assumption that one-way functions exist, and in the setting where the sender and receiver share a private key.

We study variants of locally decodable and locally correctable codes in computationally bounded, adversarial channels, under the much weaker assumption that collision-resistant hash functions exist, and with no public-key or private-key cryptographic setup. Specifically, we provide constructions of *relaxed locally correctable* and *relaxed locally decodable codes* over the binary alphabet, with constant information rate, and poly-logarithmic locality. Our constructions compare favorably with existing schemes built under much stronger cryptographic assumptions, and with their classical analogues in the computationally unbounded, Hamming channel.

Our constructions crucially employ *collision-resistant hash functions* and *local expander graphs*, extending ideas from recent cryptographic constructions of memory-hard functions.

1 Introduction

Classically, an error-correcting code is a tuple $(\mathsf{Enc}, \mathsf{Dec})$ of encoding and decoding algorithms employed by a sender to encode messages, and by a receiver to decode them, after potential corruption by a noisy channel during transmission. Specifically, the sender encodes a *message* m of k symbols

[*]Department of Computer Science, Purdue University, West Lafayette, IN. Email: jblocki@purdue.edu.

[†]Department of Computer Science, Johns Hopkins University, Baltimore, MD. Email: gv@jhu.edu.

[‡]Department of Computer Science, Purdue University, West Lafayette, IN. Research supported by NSF CCF-1649515. Email: elena-g@purdue.edu.

[§]Department of Computer Science, Purdue University, West Lafayette, IN. Research supported in part by NSF CCF-1649515. Email: samsonzhou@gmail.com.

from an alphabet Σ into a *codeword* c of block-length n consisting of symbols over the same alphabet, via $\mathsf{Enc} : \Sigma^k \to \Sigma^n$. The receiver uses $\mathsf{Dec} : \Sigma^n \to \Sigma^k$ to recover the message from a received word $w \in \Sigma^n$, a corrupted version of some $\mathsf{Enc}(m)$. Codes over the binary alphabet $\Sigma = \{0, 1\}$ are preferred in practice. The quantities of interest in designing classical codes are the *information rate*, defined as k/n, and the *error rate*, which is the tolerable fraction of errors in the received word. Codes with both large information rate and large error rate are most desirable.

In modern uses of error-correcting codes, one may only need to recover small portions of the message, such as a single bit. In such settings, the decoder may not need to read the entire received word $w \in \Sigma^n$, but only read a few bits of it. Given an index $i \in [n]$ and oracle access to w, a local decoder must make only $q = o(n)$ queries into w, and output the bit m_i. Codes that admit such fast decoders are called *locally decodable codes* (LDCs) [KT00, STV99]. The parameter q is called the *locality* of the decoder. A related notion is that of *locally correctable codes* (LCCs). LCCs are codes for which the local decoder with oracle access to w must output bits of the codeword c, instead of bits of the message m. LDCs and LCCs have widespread applications in many areas of theoretical computer science, including private information retrieval, probabilistically checkable proofs, self-correction, fault-tolerant circuits, hardness amplification, and data structures (e.g., [BFLS91, LFKN92, BLR93, BK95, CKGS98, CGdW13, ALRW17]). See surveys [Tre04, Gas04]. However, constructions of such codes suffer from apparently irreconcilable tension between locality and rate: existing codes with constant locality have slightly subexponential blocklength [Yek08, Efr12, DGY11], and codes of linear blocklength have slightly subpolynomial query complexity [KMRS17]. See surveys by Yekhanin [Yek12] and by Kopparty and Saraf [KS16].

Ben-Sasson *et al.* [BGH+06] propose the notion of *relaxed locally decodable codes* (RLDCs) that remedies the dramatic tradeoffs of classical LDCs. In this notion the decoding algorithm is allowed to output $\perp$ sometimes, to signal that it does not know the correct value; however, it should not output an incorrect value too often. More formally, given $i \in [k]$ and oracle access to the received word w assumed to be relative close to some codeword $c = \mathsf{Enc}(m) \in \Sigma^n$, the local decoder (1) outputs m_i if $w = c$; (2) outputs either m_i or $\perp$ with probability $2/3$, otherwise; (3) the set of indices i such that the decoder outputs m_i (the correct value) with probability $2/3$ is of size $> \rho \cdot k$ for some constant $\rho > 0$. The relaxed definition allows them to achieve RLDCs with constant query complexity and blocklength $n = k^{1+\epsilon}$.

Recently, Gur *et al.* [GRR18] introduce the analogous notion of *relaxed locally correctable codes* (RLCCs). In particular, upon receiving a word $w \in \Sigma^n$ assumed to be close to some codeword c, the decoder: (1) outputs c_i if $w = c$; (2) outputs either c_i or $\perp$ with probability $2/3$, otherwise; and (3) the set of indices i such that the decoder outputs c_i with probability $2/3$ is of size $\rho \cdot k$, for some $\rho > 0$. In fact, [GRR18] omits condition (3) in their definition, since the first two conditions imply the 3rd, for codes with constant locality that can withstand a constant fraction of error [BGH+06]. The reduction from [BGH+06], however, does not maintain the asymptotic error rate, and in particular, in the non-constant query complexity regime, the error rate becomes subconstant. Since our results work in the $\omega(1)$-query regime, we will build codes that achieve the 3rd condition as well (for constant error rate). The results in [GRR18] obtain significantly better parameters for RLCCs than for classical LCCs; namely, they construct RLCCs with constant query complexity, polynomial block length, and constant error rate, and RLCCs with quasipolynomial query complexity, linear blocklength (constant rate), with the caveat that the error rate is subconstant. These results immediately extend to RLDCs, since their codes are *systematic*, meaning that the initial part of the encoding consists of the message itself.

Computationally bounded, adversarial channels In this work we study RLDC*s* and RLCC*s* in the more restricted, yet natural, *computationally bounded adversarial channel*, introduced by Lipton [Lip94]. All the above constructions of local codes assume a channel that may introduce a bounded number of adversarial errors, and the channel has as much time as it needs to decide what positions to corrupt. This setting corresponds to Hamming's error model, in which codes should be resilient to any possible error pattern, as long as the number of corruptions is bounded. Hence, codes built for the Hamming channel are safe for data transmission, but the drastic requirements lead to coding limitations. By contrast, in his foundational work, Shannon proposes a weaker, probabilistic channel in which each symbol is corrupted independently, with some fixed probability.

In [Lip94], Lipton argues that many reasonable channels stand somewhere in between these two extremes, and one may assume that in reality adversaries are computationally bounded, and can be modeled as *polynomial time probabilistic* (PPT) algorithms. Variants of this model have been initially studied for classical error-correcting codes [Lip94, DGL04, MPSW05, GS16] to show better error rate capabilities than in the Hamming model.

Ostrovsky *et al.* [OPS07] introduce the study of "private" locally decodable codes against computationally bounded channels, and build codes with constant information and error rates over the binary alphabet, which can correct every message bit using only a small (superconstant) number of queries to the corrupted word. Their results assume the existence of one-way functions, and require that the encoding and decoding algorithms share a secret key that is not known to the channel. Hemenway and Ostrovky [HO08] and Hemenway *et al.* [HOSW11] construct public-key LDC*s* assuming the existence of Φ-hiding schemes [CMS99] and of IND-CPA secure cryptosystems, respectively.

By contrast, our constructions of RLDC*s* and RLCC*s* do not require the sender and receiver to exchange cryptographic keys. Instead our constructions are based on the existence of collision-resilient hash functions, a standard cryptographic assumption. Because the parameters of a collision-resistant hash function are public, *any* party (sender/receiver/attacker) is able to compute it.

2 Our Contributions

We start by defining the version of relaxed locally correctable codes that is relevant for our model. We remark that our codes are systematic as well, and therefore the results also apply to the RLDC*s* analogue.

Since these codes interact with an adversarial channel, the strength of the codes is not only measured in its error correction and locality capabilities (as is the case for RLCC*s*/RLDC*s* in the Hamming channel), but also in the security they provide against the channel. We present these codes while describing how they interact with the channel, in order to make the analogy with the classical setting. We use the notation Enc and Dec to denote encoding and decoding algorithms.

Definition 2.1 *A* local code *is a tuple* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *of probabilistic algorithms such that*

- $\mathsf{Gen}(1^\lambda)$ *takes as input security parameter* λ *and generates a public seed* $s \in \{0,1\}^*$. *This public seed* s *is* fixed *once and for all.*
- Enc *takes as input the public seed* s *and a message* $x \in \Sigma^k$ *and outputs a codeword* $c = \mathsf{Enc}(s, x)$ *with* $c \in \Sigma^n$.

- Dec *takes as input the public seed* s*, an index* $i \in [n]$*, and is given oracle access to a word* $w \in \Sigma^n$*.* $\mathsf{Dec}^w(s,i)$ *outputs a symbol* $b \in \Sigma$ *(which is supposed to be the value of the closest codeword to* w *at position* i*)*

We say that the (information) rate *of the code* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* k/n*. We say that the code is* efficient *if* $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$ *are all probabilistic polynomial time (PPT) algorithms.*

Definition 2.2 *A* computational adversarial channel $\mathcal{A}$ *with error rate* τ *is an algorithm that interacts with a local code* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *of rate* k/n *in rounds, as follows. In each round of the execution, given a security parameter* λ*,*

(1) Generate $s \leftarrow \mathsf{Gen}(1^\lambda)$*;* s *is public, so* Enc*,* Dec*, and* $\mathcal{A}$ *have access to* s

(2) The channel $\mathcal{A}$ *on input* s *hands a message* x *to the sender.*

(3) The sender computes $c = \mathsf{Enc}(s,x)$ *and hands it back to the channel (in fact the channel can compute* c *without this interaction).*

(4) The channel $\mathcal{A}$ *corrupts at most* τn *entries of* c *to obtain a word* $w \in \Sigma^n$*;* w *is given to the receiver's* Dec *with query access, together with a challenge index* $i \in [n]$

(5) The receiver outputs $b \leftarrow \mathsf{Dec}^w(s,i)$*.*

(6) We define $\mathcal{A}(s)$*'s* probability of fooling Dec *on this round to be* $p_{\mathcal{A},s} = \Pr[b \notin \{\perp, c_i\}]$*, where the probability is taken only over the randomness of the* $\mathsf{Dec}^w(s,i)$*. We say that* $\mathcal{A}(s)$ *is* γ*-successful* at fooling Dec *if* $p_{\mathcal{A},s} > \gamma$*. We say that* $\mathcal{A}(s)$ *is* ρ*-successful* at limiting Dec *if* $|\mathsf{Good}_{\mathcal{A},s}| < \rho \cdot n$*, where* $\mathsf{Good}_{\mathcal{A},s} \subseteq [n]$ *is the set of indices* j *such that* $\Pr[\mathsf{Dec}^w(s,j) = c_j] > \frac{2}{3}$*. We use* $\mathsf{Fool}_{\mathcal{A},s}(\gamma,\tau,\lambda)$ *(resp.* $\mathsf{Limit}_{\mathcal{A},s}(\rho,\tau,\lambda)$*) to denote the event that the attacker was* γ*-successful at fooling* Dec *(resp.* ρ*-successful at limiting* Dec*) on this round.*

We now define our secure RLCC codes against computational adversarial channels.

Definition 2.3 ((Computational) Relaxed Locally Correctable Codes (CRLCC)) *A local code* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a* $(q,\tau,\rho,\gamma,\mu(\cdot))$*-CRLCC* against a class $\mathbb{A}$ *of adversaries if* Dec^w *makes at most* q *queries to* w *and satisfies the following:*

(1) For all public seeds s *if* $w \leftarrow \mathsf{Enc}(s,x)$ *then* $\mathsf{Dec}^w(s,i)$ *outputs* $b = (\mathsf{Enc}(s,x))_i$*.*

(2) For all $\mathcal{A} \in \mathbb{A}$ *we have* $\Pr[\mathsf{Fool}_{\mathcal{A},s}(\gamma,\tau,\lambda)] \leq \mu(\lambda)$*, where the randomness is taken over the selection of* $s \leftarrow \mathsf{Gen}(1^\lambda)$ *as well as* $\mathcal{A}$*'s random coins.*

(3) For all $\mathcal{A} \in \mathbb{A}$ *we have* $\Pr[\mathsf{Limit}_{\mathcal{A},s}(\rho,\tau,\lambda)] \leq \mu(\lambda)$*, where the randomness is taken over the selection of* $s \leftarrow \mathsf{Gen}(1^\lambda)$ *as well as* $\mathcal{A}$*'s random coins.*

When $\mu(\lambda) = 0$ *and* $\mathbb{A}$ *is the set of all (computationally unbounded) channels we say that the code is a* (q,τ,ρ,γ)*-RLCC. When* $\mu(\cdot)$ *is a negligible function* and $\mathbb{A}$ *is restricted to the set of all probabilistic polynomial time (PPT) attackers we say that the code is a* (q,τ,ρ,γ)*-CRLCC (computational relaxed locally correctable code).*

We say that a code that satisfies conditions 1 and 2 is a Weak CRLCC*, while a code satisfying conditions 1, 2 and 3 is a* Strong CRLCC *code.*

We construct Weak and Strong CRLCCs against PPT adversaries, under the assumption that *Collision-Resistant Hash Functions* (CRHF) exist. Briefly, a CRHF function is a pair (GenH, H) of probabilistic polynomial time (PPT) algorithms, where GenH takes as input a security parameter 1^λ and outputs a public seed $s \in \{0,1\}^*$; the function $H : \{0,1\}^* \times \Sigma^* \to \Sigma^{\ell(\lambda)}$, takes as input the seed s and a long enough input that is hashed into a string of length $\ell(\lambda)$. We note that H is deterministic upon fixing s. The value $\ell(\lambda)$ is the *length* of the hash function. (GenH, H) is said to be collision-resistant if for all PPT adversaries that take as input the seed s generated by $\mathsf{Gen}(1^\lambda)$, the probability that they produce a collision pair (x, x'), i.e. such that $H(s, x) = H(s, x')$ and $x \neq x'$, is negligible in 1^λ.

Theorem 2.4 *Assuming the existence of a collision-resistant hash function* (GenH, H) *with length* $\ell(\lambda)$*, there exist constants* $0 < \tau, \rho, \gamma < 1$ *and a negligible function* μ*, such that there exists a constant rate* $(\ell(\lambda) \cdot \mathsf{polylog}\, n, \tau, \rho, \gamma, \mu(\cdot))$*-Weak CRLCC of blocklength* n *over the binary alphabet. In particular, if* $\ell(\lambda) = \mathsf{polylog}\, \lambda$ *and* $\lambda \in \Theta(n)$ *then the code is a* $(\mathsf{polylog}\, n, \tau, \rho, \gamma, \mu(\cdot))$*-Weak CRLCC.*

We note that in the above constructions the codes withstand a constant error rate. The classical RLCC*s* of [GRR18] achieve $(\log n)^{\mathcal{O}(\log \log n)}$ query complexity, constant information rate, but subconstant error rate, in the Hamming channel.

In order to obtain Strong CRLCC we need further technical insights, that builds upon the Weak CRLCCs. Our Strong CRLCCs have the same parameters, with only a $\mathsf{polylog}\, n$ loss in query complexity.

Theorem 2.5 *Assuming the existence of a collision-resistant hash function* (GenH, H) *with length* $\ell(\lambda)$*, there exist constants* $0 < \tau, \rho, \gamma < 1$ *and a negligible function* μ*, such that there exists a constant rate* $(\mathsf{polylog}\, n, \tau, \rho, \gamma, \mu(\cdot))$*-Strong CRLCC of blocklength* n *over the binary alphabet. In particular, if* $\ell(\lambda) = \mathsf{polylog}\, \lambda$ *and* $\lambda \in \Theta(n)$ *then the code is a* $(\mathsf{polylog}\, n, \tau, \rho, \gamma, \mu(\cdot))$*-Strong CRLCC.*

Our constructions are systematic, so they immediately imply the existence of CRLDCs with the same parameters.

2.1 Technical Ingredients

At a technical level our construction uses two main building blocks: local expander graphs and collision resistant hash functions.

Local Expander Graphs Intuitively, given a graph $G = (V, E)$ and distinguished subsets $A, B \subseteq V$ of nodes such that A and B are disjoint and $|A| = |B|$ we say that the pair (A, B) contains a δ-expander if for all $X \subseteq A$ and $Y \subseteq B$ with $|X| > \delta|A|$ and $|Y| > \delta|B|$ there is an edge connecting X and Y *i.e.*, $(X \times Y) \cap E \neq \emptyset$. A δ-local expander is a directed acyclic graph (DAG) G with n nodes $V(G) = \{1, \ldots, n\}$ with the property that for *any* radius $r > 0$ and *any* node $v \geq 2r$ the sets $A = \{v - 2r + 1, \ldots, v - r\}$ and $B = \{v - r + 1, \ldots, v\}$ contain a δ-expander. For any constant $\delta > 0$ it is possible to construct a δ-local expander with the property that $\mathsf{indeg}(G) \in \mathcal{O}(\log n)$ and $\mathsf{outdeg}(G) \in \mathcal{O}(\log n)$ i.e. no node is has more that $\mathcal{O}(\log n)$ incoming or outgoing edges [EGS75, ABP18].

Local expanders have several nice properties that have been recently exploited in the design and analysis of secure memory hard functions [ABH17, ABP17, BZ17, ABP18]. For example, these

graphs are maximally depth-robust [ABP18]. Even if we delete a *large* number of nodes $S \subseteq V$ the graph still contains a directed path of length $n - (1+\epsilon)|S|$ for some small constant ϵ dependent on δ — the constant ϵ can approach 0 as δ approaches 0 [ABP18]. More specifically, if we delete a large number of nodes $S \subseteq V$ at least $n - (1+\epsilon)|S|$ of the nodes have the property that they are α-good (with $\epsilon = \left(\frac{2-\alpha}{\alpha}\right)$) with respect to the deleted set S and *any* pair of α-good nodes u and v are connected by a directed path (provided that δ is sufficiently small) — a node v is α-good with respect to S if for any radius $r < v$ we have at most αr nodes in $S \cap [v-r+1, v]$ *and* for any radius $r \leq n - v + 1$ we have at most αr nodes in $S \cap [v, v+r-1]$.

Suppose that each node is colored red or green and that we are only allowed to query each node to obtain its color. If we let S denote the set of red nodes then we can develop an *efficient* randomized testing algorithm to check if a node v is α-good or not. The tester will make $\mathcal{O}(\mathsf{polylog}\, n)$ queries and, with high probability, will *accept* any node v that is α_1-good and will *reject* any node w that is *not* α_2-good for *any* constants $\alpha_2 > 4 \cdot \alpha_1$. Intuitively, for each $r \in \{2^1, 2^2, \ldots, 2^{\log n}\}$ the tester will sample $\mathcal{O}(\mathsf{polylog}\, n)$ nodes in the intervals $[v, r-1]$ and $[v-r+1, v]$ to make sure that the fraction of red nodes is at most $2\alpha_1$. If the tester determines that a node v is *at least* α_2-good for an appropriately small constant α_2 then there is a long directed path of length at least $n - (1+\epsilon)|S|$ containing all α_2-good nodes and in particular containing v. Furthermore, if $v < 3n/4$ then v has at least $n/4 - (1+\epsilon)|S|$ descendants in this path.

Collision Resistant Hash Functions Our constructions employ *collision resistant hash functions* as a building block. While most of the recent progress on memory hard functions in cryptography combines local expanders (depth-robust graphs) with random oracles (e.g., see [AS15, AT17, ABP18]), we stress that we do not need to work in the random oracle model[1]. Indeed, our constructions only assume the existence of collision resistant hash functions.

Weak CRLCCs We first explain our construction of Weak CRLCCs, which involves labeling a δ-local expander with k nodes. In particular, given an input word $x = (x_1, \ldots, x_k)$ (broken up into bit strings of length $\ell(\lambda)$) and a k node local expander graph G, the label of node v is computed as $\ell_{v,s} = H(s, x_v \circ \ell_{v_1,s} \circ \cdots \circ \ell_{v_d,s}) \in \{0,1\}^{\ell(\lambda)}$, where $\ell_{v_1,s}, \ldots, \ell_{v_d,s}$ are the labels of the parents $v_1, \ldots, v_d$ of node v, and $\circ$ denotes string concatenation. When $\ell(\lambda) \in \mathcal{O}(\mathsf{polylog}\, \lambda)$ we will select $\lambda \in \mathcal{O}(n)$ to ensure that $\ell(\lambda) \in \mathcal{O}(\mathsf{polylog}\, n)$. We use the notation Enc and Dec for the encoding and decoding of our construction, while we use ECC and ECCD to denote the efficient encoding and decoding algorithms for a good binary code with constant rate and relative distance (e.g., [Jus72, SS96]) which can decode efficiently from some constant fractions of errors.

We first apply ECC to $x_1, \ldots, x_k$ to obtain codewords $c_1, \ldots, c_k \in \{0,1\}^{\mathcal{O}(\ell(\lambda))}$ where $c_i = \mathsf{ECC}(x_i)$. Also, for $v \in [k]$ we let $c_{v+k} = \mathsf{ECC}(\ell_{v,s})$ which is the encoding of the label corresponding to the node v in G. The final output is $c = (c_1, \ldots, c_{2k-1}, c_{2k}, c_{2k+1}, \ldots, c_{3k})$ where $c_{2k+1} = \ldots = c_{3k} = c_{2k}$ consists of k copies of the last codeword c_{2k}. The final word is an n bit message with $n = \mathcal{O}(3k\ell(\lambda))$. By repeating this last codeword k times we ensure that it is *not possible* for the attacker to irreparably corrupt the final label $\ell_{k,s}$.

Given a (possibly corrupted) codeword c' produced by a PPT attacker $\mathcal{A}$ we let $x' = (x'_1, \ldots, x'_k)$ with $x'_i = \mathsf{ECCD}(c'_i)$ (possibly $\perp$) and we let $\ell'_{v,s} = \mathsf{ECCD}(c'_{v+k})$ for $v \in [k]$ and $\ell'_{k,s,j} = \mathsf{ECCD}(c'_{2k+j})$

[1] The random oracle model is a source of some controversy among cryptographers [KM07, Men12, Gol06, KM15] with some arguing that the framework can be used to develop cryptographic protocols that are *efficient and secure* [BR93], while others argue that the methodology is flawed e.g. [BL13, KM15].

for each $j \in [k]$. We say that a node v is *green* if it is locally consistent i.e.,

$$\ell'_{v,s} = H\left(s, x'_v \circ \ell'_{v_1,s} \circ \ldots \circ \ell'_{v_d,s}\right),$$

otherwise we say that the node is *red*. We first show that if a green node has the correct label $\ell'_{v,s} = \ell_{v,s}$ then it *must* be the case that $x'_v = x_v$ and $\ell'_{v_i,s} = \ell_{v_i,s}$ for each $i \leq d$ — otherwise $\mathcal{A}$ would have found a hash collision! If a graph contains too many red nodes then this is easily detectable by random sampling and our *weak* local decoder is allowed to output $\perp$ if it detects *any* red nodes. Our local decoder will first obtain the final label $\ell_{k,s}$ by random sampling some labels from $\ell'_{k,s,1}, \ldots, \ell'_{k,s,k}$ and checking to make sure each of these labels is equal to $\ell'_{k,s}$. If this check passes then with high probability we must have $\ell'_{k,s} = \ell_{k,s}$ since the attacker cannot corrupt too many of these labels. Second, our local decoder will test to make sure that the last node k is *at least* α_2-good. If this node is not α_2-good then we must have found a red node and our *weak* decoder may output $\perp$; otherwise the last node serves as an *anchor point*. In particular, since since label $\ell_{k,s} = \ell'_{k,s}$ in this case collision resistance now implies that for *any* α_2-good node v then we must have $x'_v = x_v$ and $\ell_{v,s} = \ell'_{v,s}$ since v *must* be connected to the node k by a green path.

Strong CRLCCs The reason that the previous construction fails to yield a strong CRLCCs is that it is possible for an attacker to *change* every node to a red-node by tampering with at most $\mathcal{O}\left(k \cdot \ell(\lambda)/\log k\right)$ bits. In particular, since the δ-local expander G has outdegree $\mathcal{O}\left(\log k\right)$ an attacker who tampers with just $\mathcal{O}\left(\ell(\lambda)\right)$ bits could tamper with one of the labels so that $x'_v \neq x_v$. Now for *every* $w \in [k]$ s.t. G contains the directed edge (v, w) the node w will be red and there are up to $\mathcal{O}\left(\log k\right) = \mathcal{O}\left(\log n\right)$ such nodes w.

We address this issue by first applying a degree reduction gadget to our family $\{G_t\}_{t=1}^{\infty}$ of δ-local expanders, where $t = \mathcal{O}\left(\frac{k}{\ell(\lambda)\cdot\log k}\right)$, to obtain a new family of DAGs as follows: First, we replace every node $u \in [t]$ from G_t with a chain $u_1, \ldots, u_m$ of $m = \mathcal{O}\left(\log t\right)$ nodes — we will refer to node $u \in [t]$ as the meta-node for this group. The result is a new graph with $k = m \cdot t$ nodes. Each of the nodes $u_1, \ldots, u_{m-1}$ will have constant indegree and constant outdegree. However, for $i < m$ we include the directed edges (u_i, u_{i+1}) and (u_i, u_m) — the node u_m will have $\mathsf{indeg}(u_m) \in \Theta(m)$. Furthermore, if we have an edge (u, v) in G_t then we will add an edge of the (u_m, v_j) for some $j < m$ — this will be done in such a way that maintains $\mathsf{indeg}(v_j) \leq 2$ for each $j < m$. Therefore, the node u_m will have $\mathsf{outdeg}(u_m) \in \Theta(\log t)$. We now note that if the label $\ell'_{u_m,s} = \ell_{u_m,s}$ and the node u_m is green then, by collision resistance, it must be the case that $\ell'_{u_i,s} = \ell_{u_i,s}$ and $x'_{u_i} = x_{u_i}$ for *every* u_i with $i \leq m$.

With this observation in mind we tweak our last construction by grouping the values $\ell_{v_i,s}$ and x_{v_i} into groups of size $m \in \mathcal{O}\left(\log t\right)$ before applying the error correcting code. In particular, given our input word x divided into $k = m \cdot t$ distinct $\ell(\lambda)$-bit strings x_{v_i} for $i \leq m$ and $v \leq t$, our final output will consist of $c = (c_1, \ldots, c_{3t})$ where $c_i = \mathsf{ECC}(x_{(i-1)m+1}, \ldots, x_{im})$ for $i \leq t$ and $c_{v+t} = \mathsf{ECC}\left(\ell_{v_1,s} \circ \ldots \circ \ell_{v_m,s}\right)$ and $c_{2t+1} = \ldots = c_{3t} = c_{2t}$ consists of t copies of c_{2t}. The final codeword c will have length $n \in \mathcal{O}\left(tm \cdot \ell(\lambda)\right) = \mathcal{O}\left(k \cdot \ell(\lambda)\right)$ bits. By grouping blocks together we get the property that an attacker that wants to alter an individual label $\ell_{v_i,s}$ *must* pay to flip *at least* $\Omega\left(\ell(\lambda) \cdot m\right)$ bits of c_{v+t}. Thus, the attacker is *significantly* restricted in the way he can tamper with the labels e.g., the attacker cannot tamper with one label in every group.

We now define a *meta-node* $u \in [t]$ containing $u_1, \ldots, u_m$, to be green if (1) node u_m is green, and (2) at least $2m/3$ of the nodes $u_1, \ldots, u_m$ are green. Even if we flip a (sufficiently small) constant fraction of the bits in c we can show that *most* of the meta-nodes must be green — in fact, most of

these meta-nodes must also be α_2-good. In particular, there are only two ways to turn a meta-node u red: (1) by paying to flip $\mathcal{O}(\ell(\lambda) \cdot m)$ bits in c_u or c_{u+t}, and (2) by corrupting at least $m/3$ other meta-nodes w such that the directed edge (w, u) is in G_t. Finally, we are able to argue that, by collision resistance, if a meta-node $u \in [t]$ is green and the meta-node u is correct (i.e., $x'_{u_j} = x_{u_j}$ and $\ell'_{u_j,s} = \ell_{u_j,s}$ for each $j \leq m$) and if we have the directed edge (w, u) in G_t then the label $\ell'_{w_m,s}$ must also be correct (thus, if the meta-node w is green then the meta-node must be correct). As long as the number of red nodes S is sufficiently small. It follows that *any* α_2-good meta-node $u < 3t/4$ is connected to at least $t/4 - (1+\epsilon)|S|$ green meta-nodes. Since the attacker does not have the budget to corrupt *all* $t/4 - (1+\epsilon)|S|$ of these meta-nodes it follows that the meta-node u *must* be correct. Once we determine that a meta-node u is correct ($x'_{u_j} = x_{u_j}$ and $\ell'_{u_j,s} = \ell_{u_j,s}$ for each $j \leq m$) we can easily recover $c_u = \mathsf{ECC}(x'_{u_1} \circ \ldots \circ x'_{u_m})$ and/or $c_{u+t} = \mathsf{ECC}(\ell'_{u_1,s} \circ \ldots \circ \ell'_{u_m,s})$ [2].

3 Related work

Classical LDCs/LCCs and relaxed variants in the Hamming channel The current best constructions for LDCs/LCCs can achieve any constant rate $R < 1$, any constant relative distance (i.e., minimum Hamming distance between codewords) $\delta < 1 - R$, and query complexity $\mathcal{O}\left(2^{\sqrt{\log n \log\log n}}\right)$ [KMRS17]. In the constant query complexity regime, for $q \geq 3$, the best codes achieve blocklength subexponential in the message length [Yek08, Efr12, DGY11]. For $q = 2$, Kerenidis and deWolf [KdW04] show an exponential in k lower bound for the blocklength of any LDC.

A notion similar to RLDCs, called Locally Decode/Reject code, was studied by Moshkovitz and Raz [MR10] in the context of building better PCPs. The notion concerns decoding batches of coordinates jointly, and it allows as output a list of possible messages. In the context of simplifying the proofs in [MR10], a related notion of decodable PCPs was studied in [DH09].

Non-local codes in computationally bounded channels In their initial works in the computationally bounded channel, Lipton [Lip94] and Ding, Gopalan and Lipton [DGL04] obtain error-correcting codes uniquely decodable (in the classical, global setting) with error rates beyond what is possible in the adversarial Hamming channel. Their model requires that the sender and receiver share a *secret* random string, unknown to the channel. This is a strong cryptographic assumption, as the model is unsuitable to common settings such as message broadcast. Micali *et al.* [MPSW05] address this drawback by proposing public-key error-correcting codes against computationally bounded channels. Their model is based on the observation that if one starts with a code that is *list-decodable*, encoding messages using a secret key and digitally signing them, one can prevent a PPT channel from producing valid signatures, while allowing the receiver to pick the unique message from the list with a valid signature. In follow-up work, Guruswami and Smith [GS16] removes the public-key setup, and obtains optimal rate error-correcting codes for channels that can be described by simple circuits. Their channels are either "oblivious", namely the error is independent of the actual message being sent, or they are describable by polynomial size circuits. Their results are based on the idea that the sender can choose a permutation and a "blinding

[2] If we want to recover c_{2t+j} we can simply draw $r \in \mathcal{O}(\mathsf{polylog}\, k)$ samples $s_1, \ldots, s_r$ from the set $c_{2t}, \ldots, c_{3t}$, decode to obtain $y_1, \ldots, y_r$ with $y_j = \mathsf{ECCD}(s_j)$, take a majority vote to obtain y_{maj} and output $\mathsf{ECC}(y_{maj})$.

factor" that are then embedded into the codeword together with the message. The channel cannot distinguish the hidden information since it operates with low complexity, while the receiver can.

LDC*s* in computationally bounded channels The notion of LDC*s* over computationally bounded channels was introduced in [OPS07], where the authors define private LDC*s*. In these constructions the sender and the receiver share a private key. They obtain codes of constant rate and $\mathcal{O}(\mathsf{polylog}\,\lambda)$ query complexity, where λ is the security parameter. Hemenway and Ostrovsky [HO08] build LDC*s* in the public-key model, and obtain codes of constant rate, and $\mathcal{O}(\lambda^2)$ locality. Hemenway *et al.* [HOSW11] improve on these results and obtain public-key LDC*s* with constant rate, constant error rate and $\mathcal{O}(\lambda)$ locality, which work under a weaker cryptographic assumption than that of [HO08].

4 Preliminaries

We use the notation $[n]$ to represent the set $\{1, 2, \ldots, n\}$. For any $x, y \in \Sigma^n$, let $\mathsf{dist}(x)$ denote the Hamming weight of x, i.e. the number of non-zero coordinates of x. Let $\mathsf{dist}(x, y) = \mathsf{dist}(x - y)$ denote the Hamming distance between the vectors x and y. For any vector $x \in \Sigma^n$, let $x[i]$ be the i^{th} coordinate of x. We also let $x \circ y$ denote the concatenation of x with y.

We denote a directed acyclic graph G with n vertices labelled in topological order by $G = ([n], E)$. A node $v \in V$ has indegree $\delta = \mathsf{indeg}(v)$ if there exist δ incoming edges $\delta = |(V \times \{v\}) \cap E|$. Thus, we say that graph G has indegree $\delta = \mathsf{indeg}(G)$ if the maximum indegree of any node of G is δ. A node with indegree 0 is called a source node and a node with no outgoing edges is called a sink. A node $v \in V$ has outdegree $\delta = \mathsf{outdeg}(v)$ if there exist δ outgoing edges $\delta = |(V \times \{v\}) \cap E|$. Thus, we say that graph G has outdegree $\delta = \mathsf{outdeg}(G)$ if the maximum outdegree of any node of G is δ. Finally, we say that graph G has degree $\delta = \mathsf{deg}(G)$ if the maximum degree of any node of G is δ, or equivalently $\max_{v \in V} \mathsf{outdeg}(v) + \mathsf{indeg}(v) = \delta$. We use $\mathsf{parents}_G(v) = \{u \in V : (u, v) \in E\}$ to denote the parents of a node $v \in V$. We will often consider graphs obtained from other graphs by removing subsets of nodes. Therefore if $S \subset V$, then we denote by $G - S$ the DAG obtained from G by removing nodes S and incident edges.

Let $\mathcal{C}$ be a (n, k) code that maps any k-length *message* over alphabet Σ to a unique n-length *codeword* over alphabet Σ. We say n is the *block length* of the code, and k/n is the *information rate*. Let $\mathsf{Enc} : \Sigma^k \to \Sigma^n$ denote the encoding map of $\mathcal{C}$. We define the *minimum distance* of $\mathcal{C}$ to be the quantity $\min_{c_1, c_2 \in \mathcal{C}} \mathsf{dist}(c_1, c_2)$ and the *relative minimum distance* of $\mathcal{C}$ to be the quantity $\min_{c_1, c_2 \in \mathcal{C}} \frac{\mathsf{dist}(c_1, c_2)}{n}$.

We shall use Enc and Dec to refer to the encoding and decoding algorithms of our construction, and ECC and ECCD to refer to the encoding and decoding algorithms for a binary code C_J with constant rate and relative distance. We will use Justesen codes [Jus72] in what follows.

Theorem 4.1 *[Jus72] For any $0 < R < 1$, there exist binary linear codes of rate R, that are efficiently decodable from $\Delta_J(R)$ fraction of errors, where $\Delta_J(R)$ is a function that only dependents on R.*

In particular, we use ECC and ECCD to denote the efficient encoding and decoding algorithms of a Justesen binary linear code of rate $R = \frac{1}{4}$, and we let $\Delta_J = \Delta_J(1/4)$ in the construction that follows.

Collision Resistant Hash Functions (CRHF)

Our code constructions involve the use of collision-resistant hash functions. We use the following definitions from Katz and Lindell [KL14].

Definition 4.2 *[KL14, Definition 4.11] A* hash function *with alphabet Σ and blocklength $\ell(.)$ is a pair $\Pi = (\mathsf{GenH}, H)$ of probabilistic polynomial time algorithms satisfying:*

- GenH *is a probabilistic algorithm which takes as input a security parameter 1^λ and outputs a public seed $s \in \{0,1\}^*$, where the security parameter 1^λ is implicit in the string s.*
- *H is a deterministic algorithm that takes as input a seed s and a string Σ^* and outputs a string $H(s,x) \in \Sigma^{\ell(\lambda)}$.*

A collision resistant hash function can be defined using the following experiment for a hash function $\Pi = (\mathsf{GenH}, H)$, an adversary $\mathcal{A}$, and a security parameter λ:

The collision-finding experiment $\mathsf{Hash} - \mathsf{coll}_{\mathcal{A},\Pi}(\lambda)$:

(1) $s \leftarrow \mathsf{GenH}(1^\lambda)$

(2) $(x, x') \leftarrow \mathcal{A}(s)$

(3) The output of the experiment is 1 if and only if $\mathcal{A}$ successfully finds a collision, i.e.

$$x \neq x' \text{ and } H(s,x) = H(s,x').$$

Then this experiment implicitly defines a collision resistant hash function.

Definition 4.3 *[KL14, Definition 4.12] A hash function $\Pi = (\mathsf{GenH}, H)$ is* collision resistant *if for all probabilistic polynomial time adversaries $\mathcal{A}$ there exists a negligible function negl such that*

$$\mathbf{Pr}\left[\mathsf{Hash} - \mathsf{coll}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

Background on δ-Local Expander Graphs

We now begin to describe the underlying DAGs in our code construction. We first define the class of graphs δ-expanders and δ-local expanders.

Definition 4.4 *Let $\delta > 0$. A bipartite graph $G = (U, V, E)$ is called a δ-expander if for all $M_1 \subseteq U$, $M_2 \subseteq V$ such that $|M_1| \geq \delta|U|$ and $|M_2| \geq \delta|V|$, there exists an edge $e \in M_1 \times M_2$.*

Definition 4.5 *Let $\delta > 0$ and $G = ([n], E)$ be a directed acyclic graph. G is called a δ-local expander if for every vertex $x \in [n]$ and every $r \leq \min\{x, n-x\}$, the bipartite graph induced by $U = [x, \ldots, x+r-1]$ and $V = [x+r, \ldots, x+2r-1]$ is a δ-expander.*

Theorem 4.6, due to Alwen *et al.* [ABP18], states that δ-local expanders exist with degree $\mathcal{O}(\log n)$.

Theorem 4.6 *[ABP18] For any $n > 0, \delta > 0$, there exists a δ-local expander $G = ([n], E)$ with indegree $\mathcal{O}(\log n)$ and outdegree $\mathcal{O}(\log n)$.*

The construction of Alwen *et al.* [ABP18] is *probabilistic*. In particular, they show that there is a *random distribution* over DAGs such that any sample from this distribution is a δ-local expander with high probability. The randomized construction of Alwen *et al.* [ABP18] closely follows an earlier construction of Erdos *et al.* [EGS75] [3].

We now list some properties of the δ-local expander graphs shown in [ABP18]. We will use these properties to construct the Dec algorithm for the CRLCC scheme.

Definition 4.7 (α-good node) *Let $G = ([n], E)$ be a DAG and let $S \subseteq [n]$ be a subset of vertices. Let $0 < c < 1$. We say $v \in [n] - S$ is α-good under S if for all $r > 0$:*

$$|[v - r + 1, v] \cap S| \leq \alpha r, \qquad |[v, v + r - 1] \cap S| \leq \alpha r.$$

Intuitively, we can view the subset S to be a set of "deleted" vertices. A vertex in the remaining graph is called α-good if at most α fraction of vertices in any of its neighborhood are contained in the deleted set. In our case, we will ultimately define S to be the nodes with the inconsistent labels.

The following result of [ABP18] shows that in any DAG G, even if we deleted a constant fraction of vertices, there still remain a constant fraction of vertices that are α-good.

Lemma 4.8 (Lemma 4.4 in [ABP18]) *Let $G = ([n], E)$ be a directed acyclic graph. For any $S \subset [n]$ and any $0 < \alpha < 1$, at least $n - |S| \left(\frac{2-\alpha}{\alpha}\right)$ nodes in $G - S$ are α-good under S.*

Alwen *et al.* [ABP18] also show that in a δ-local expander graph G any two α-good vertices in $G - S$ are connected.

Lemma 4.9 (Lemma 4.3 in [ABP18]) *Let $G = ([n], E)$ be a δ-local expander, $S \subseteq [n]$ and $0 \leq \alpha \leq 1$. If $\delta < \min((1 - \alpha)/2, 1/4)$, then any two α-good vertices $u, v \in G - S$ are connected in $G - S$.*

5 Construction of Weak-CRLCC

In this section we overview the construction of a constant rate weak-CRLCC scheme. In order to show the existence of a CRLCC scheme, we need to construct the PPT algorithms Gen, Enc and Dec. Our construction will use a CRHF $\Pi = (\mathsf{GenH}, H)$. In particular, $\mathsf{Gen}\left(1^\lambda\right)$ simply runs $\mathsf{GenH}\left(1^\lambda\right)$ to output the public seed s.

The Enc algorithm uses the CRHF to create labels for the vertices of a δ-local expander. We now define the recursive labeling process for the vertices of any given DAG G using H.

Definition 5.1 (Labeling) *Let $\Sigma = \{0,1\}^{\ell(\lambda)}$. Given a directed acyclic graph $G = ([k], E)$, a seed $s \leftarrow \mathsf{GenH}\left(1^\lambda\right)$ for a collision resistant hash function $H : \{0,1\}^* \to \{0,1\}^{\ell(\lambda)}$, and a message $x = (x_1, x_2, \cdots, x_k) \in \Sigma^k$ we define the labeling of graph G with x, $\mathsf{lab}_{G,s} : \Sigma^k \to \Sigma^k$ as $\mathsf{lab}_{G,s}(x) = (\ell_{1,s}, \ell_{2,s}, \cdots, \ell_{k,s})$, where*

$$\ell_{v,s} = \begin{cases} H(s, x_v), & \mathsf{indeg}(v) = 0 \\ H(s, x_v \circ \ell_{v_1,s} \circ \cdots \circ \ell_{v_d,s}), & 0 < \mathsf{indeg}(v) = d, \end{cases}$$

[3] While Alwen *et al.* [ABP18] only analyze the indegree of this construction it is trivial to see that the outdegree is also $\mathcal{O}(\log n)$ since the construction of Erdos *et al.* [EGS75] overlays multiple bipartite expander graphs on top of the nodes $V = [n]$. Each bipartite expander graph has $\mathsf{indeg}, \mathsf{outdeg} \in \mathcal{O}(1)$ and each node is associated with $\mathcal{O}(\log n)$ expanders.

where $v_1, \ldots, v_d$ are the parents of vertex v in G, according to some predetermined topological order. We omit the subscripts G, s when the dependency on the graph G and public hash seed H is clear from context.

5.1 Enc Algorithm

In this section we describe the Enc algorithm which takes as input a seed $s \leftarrow \mathsf{Gen}\left(1^\lambda\right)$ and a message $x \in \{0,1\}^k$, and returns a codeword $c \in \{0,1\}^n$. For a security parameter λ, let $H : \{0,1\}^* \to \{0,1\}^{\ell(\lambda)}$ be a CRHF (see Definition 4.3). We will assume that Enc has access to H. In this section we also let $\mathsf{ECC} : \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{4\cdot\ell(\lambda)}$ be a standard binary code with message length $\ell(\lambda)$ and block length $4 \cdot \ell(\lambda)$ (such as Justesen code [Jus72]) and let $\mathsf{ECCD} : \{0,1\}^{4\cdot\ell(\lambda)} \to \{0,1\}^{\ell(\lambda)}$ be the decoder that efficiently decodes from Δ_J fraction of errors.

Let $s \leftarrow \mathsf{Gen}(1^\lambda)$. Let $k' = k/\ell(\lambda)$ and for $\delta = 1/100$, let $G = ([k'], E)$ be a δ-local expander graph with indegree $\mathcal{O}(\log k')$.
$\mathsf{Enc}(s, x)$:
Input: $x = (x_1, \cdots, x_{k'}) \in \{0,1\}^k$ where each $x_i \in \{0,1\}^{\ell(\lambda)}$ and $k = k' \cdot \ell(\lambda)$.
Output: $c = (c_1, \cdots, c_{3k'-1}) \in \{0,1\}^n$ where each $c_i \in \{0,1\}^{4\cdot\ell(\lambda)}$ and $n = 4(3k'-1)\cdot\ell(\lambda)$

- Let $\mathsf{lab}_{G,s}(x) = \left(\ell_{1,s}, \cdots, \ell_{k',s}\right)$ be the labeling of the graph G with input x and seed s.
- The codeword c consists of the group-wise encoding of message x, followed by the encoded labeling of the graph using ECC, followed by $k'-1$ copies of encoded $\ell_{k'}$.

$$c = \left(\mathsf{ECC}(x_1), \cdots, \mathsf{ECC}(x_{k'}), \mathsf{ECC}(\ell_{1,s}), \cdots, \mathsf{ECC}(\ell_{k',s}), \underbrace{\mathsf{ECC}\left(\ell_{k',s}\right), \cdots, \mathsf{ECC}\left(\ell_{k',s}\right)}_{k'-1 \text{ times}}\right).$$

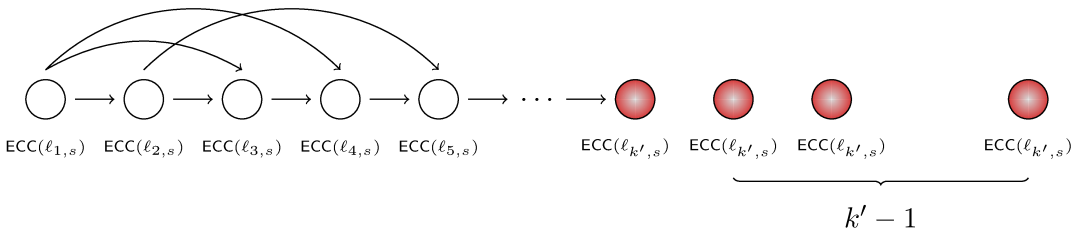

Fig. 1: An example of an encoding with an underlying graph with $k' = k/\ell(\lambda)$ nodes

The parameter $\delta = 1/100$ is arbitrary, and simply chosen to satisfy Lemma 4.9. Also, note that the codeword length is $n = 4 \cdot (3k' - 1) \cdot \ell(\lambda)$ over the binary alphabet $\{0,1\}$, where $\ell(\lambda)$ is the length of the output of the CRHF H and the original message had length $k = k' \cdot \ell(\lambda)$. Therefore, the rate $\frac{k}{n}$ of CRLCC scheme obtained from Enc is $\Theta(1)$.

5.2 Dec Algorithm

In this section, we detail a randomized algorithm $\mathsf{Dec} : \{0,1\}^n \times [n] \to \{\perp\} \cup \{0,1\}$ for $\mathsf{Enc} : \{0,1\}^k \to \{0,1\}^n$ described in Section 5.1. We assume Dec has access to the same public seed s

used by Enc as well as the δ-local expander used by Enc. We focus on the key ideas first and then provide the formal description of Dec. The notion of a green node is central to Dec.

Given a word (corrupted codeword) $w = (w_1, \ldots, w_{3k'-1}) \in (\{0,1\}^{4\cdot\ell(\lambda)})^{3k'-1}$ we define $x'_i = \mathsf{ECCD}(w_i) \in \{0,1\}^{\ell(\lambda)}$ for $i \leq k'$ and $\ell'_{v,s} = \mathsf{ECCD}(w_{k'+v})$ for $v \leq k'$ and $\ell'_{k',s,i} = \mathsf{ECCD}(w_{2k'+i})$ for $i < k'$.

Definition 5.2 (Green/red node) *We say that a node $v \in [k']$ with parents $v_1, \ldots, v_d$ is* green *if the label $\ell'_{v,s}$ is consistent with the hash of its parent labels i.e., $\ell'_{v,s} = H(s, x_v \circ \ell'_{v_1,s} \circ \ell'_{v_2,s} \circ \cdots \circ \ell'_{v_d,s})$. A node that is not green is a* red *node.*

We say that a node v is *correct* if $\ell'_{v,s} = \ell_{v,s}$ and $x'_v = x_v$. Lemma 5.3 highlights one of the key reasons why green nodes are significant in our construction.

Lemma 5.3 *Suppose that node v is* green *and* correct *(i.e., $\ell'_{v,s} = \ell_{v,s}$) and suppose that there is a directed path from node u to node v consisting* entirely *of green nodes. Then either the node u is also correct (i.e., $\ell'_{u,s} = \ell_{u,s}$) or the PPT adversary has produced a hash collision.*

Proof : If node v is green, then by definition $\ell'_{v,s} = H(s, x'_v \circ \ell'_{v_1,s} \circ \cdots \circ \ell'_{v_d,s})$ where $v_1, \ldots, v_d$ are the parents of node v. Moreover, if $\ell'_{v,s}$ is unaltered, then $\ell'_{v,s} = \ell_{v,s} = H(s, x_v \circ \ell_{v_1,s} \circ \ell_{v_2,s} \circ \cdots \circ \ell_{v_d,s})$. Thus,

$$H(s, x_v \circ \ell_{v_1,s} \circ \ell_{v_2,s} \circ \cdots \circ \ell_{v_d,s}) = H(s, x'_v \circ \ell'_{v_1,s} \circ \cdots \circ \ell'_{v_d,s}).$$

Either $x_v = x'_v$ and $\ell_{v_j,s} = \ell'_{v_j,s}$ for each $j \in [d]$ or we have found a hash collision. In particular, let v_j be the node that lies on the green path from u to v. Assuming we have no hash collision the node v_j is correct and green. We can extend the same argument inductively to argue that node u must be correct. □

Our local decoder will output $\perp$ if it ever detects any red nodes (Note that when the codeword is not corrupted we will have 0 red nodes).

We now consider two cases. **Case 1:** The input index $i \geq 8k - 4\ell(\lambda)$. This corresponds to a bit query within the last k' blocks. From the construction of the code, we know that the last k' blocks of the encoding are the same, i.e. $c_{2k'+j} = c_{2k'}$ for all $1 \leq j \leq k'$. In this case, Dec^w simply queries $\mathcal{O}(\log k')$ blocks in $[2k', 3k'-1]$, finds the decodes them to the nearest codeword and returns the majority codeword. Since the adversary cannot corrupt many blocks beyond decoding radius, the majority will return the correct block with high probability. See Section 5.2.1 for the formal proof.

Case 2: If the input index $i < 8k - 4\ell(\lambda)$, Dec^w uses the properties of the δ-local expander graph to detect whether the corresponding label has been tampered or not. Let i' denote the index of the node in G associated with the queried bit i *i.e.*, $i' = \left\lceil \frac{i}{4\cdot\ell(\lambda)} \right\rceil \mod k'$. We check to make sure that (1) the node k' is *at least* α-good, and (2) the node i' is also α-good. If either check fails then we will output $\perp$. Otherwise, by Lemma 5.3 the node i' must be correct (with high probability) since (1) there is a green path connecting any two α-good nodes in a δ-local expander, and (2) the node k' must be correct (with high probability).

The core part of the Dec^w then is the probabilistic procedure to verify that a node v is α-good. First, it is clear that there is a deterministic procedure $\mathsf{IsGreen}^w(v)$ that checks whether a given node $v \in [k']$ is green or not in $\mathcal{O}(\log n)$ coordinate queries to w (see Lemma 5.5 for the formal statement). Now in order to verify if a node v is α-good, we need to estimate the fraction of green nodes in any neighborhood of it. This is achieved by designing a tester that *accepts* v if it is

$\alpha/4$-good with respect to the set S of red nodes and rejects with high probability if v is not α-good (formalized in Lemma 5.6).

The key observation behind Lemma 5.6 is that it suffices to check that $\left|[v, v+2^j-1] \cap S\right| \leq \alpha \cdot 2^j/4$ for each $j \leq \log k'$ and that $\left|[v-2^j+1, v] \cap S\right| \leq \alpha \cdot 2^j/4$ for each $j \leq \log k'$. For each j we can sample $r = \mathcal{O}(\mathsf{polylog}\, k')$ random nodes in each of the intervals $[v-2^j+1, v]$ and $[v, v+2^j-1]$ and use the subroutine $\mathsf{IsGreen}$ to count the number of red (resp. green) nodes in each interval. If for every $j \leq \log k'$ the number of red nodes is both intervals *at most* $\alpha \cdot 2^j/2$ we accept. Otherwise, we reject.

We now proceed to formalize the decoder Dec for the Weak-CRLCC described above.

5.2.1 Query Procedure for $i \geq 8k - 4\ell(\lambda)$

In this section, we describe the procedure for recovering a coordinate for input $i \geq 8k - 4\ell(\lambda)$. We show that regardless of the underlying graph, any adversary that can change at most $\frac{\Delta_J k}{4}$ coordinates of a codeword obtained from Enc, cannot prevent relaxed local correction for a query on the last k' blocks.

Consider the following algorithm $\mathsf{Dec}_1^w(s, i)$ for any $i \geq 8k - 4\ell(\lambda)$:

Dec_1^w: Input : Index $i \in [n]$ such that $i \geq 8k - 4\ell(\lambda)$.

(1) Sample $\Theta\left(\frac{\log^3 k}{\ell(\lambda)}\right)$ blocks $\{w_t\}$ of w for t between $2k'$ and $3k'$ uniformly at random.

(2) Decode each of the queried blocks w_t to the corrected codeword, $c_t := \mathsf{ECC}(\mathsf{ECCD}(w_t))$. (could possibly be a $\perp$ if ECCD fails to decode).

(3) Let $c_{maj} = \text{majority}\{c_t\}$ be the codeword of ECC which occurs majority of the times in Step (2) above.

(4) Output symbol $(i \mod 4\ell(\lambda))$ of c_{maj}.

We show in Lemma 5.4 that $\mathbf{Pr}\left[\mathsf{Dec}_1^w(s, i) = c[i]\right] \geq 1 - \mathsf{negl}(n)$ and uses at most $\mathcal{O}\left(\log^3 n\right)$ queries.

Lemma 5.4 *Let* Enc *be as described in Section 5.1. For any* $8k - 4\ell(\lambda) \leq i \leq n$, Dec_1^w *does the following:*

(1) For any $x \in \{0,1\}^k$ *and* $c = \mathsf{Enc}(s, x)$, *it holds that* $\mathsf{Dec}_1^c(s, i) = c[i]$.

(2) For any $x \in \{0,1\}^k$, $c = \mathsf{Enc}(s, x)$ *and* $w \in \{0,1\}^n$ *generated by any PPT adversary such that* $\mathsf{dist}(c, w) \leq \frac{\Delta_J k}{4}$, *it holds that*

$$\mathbf{Pr}\left[\mathsf{Dec}_1^w(s, i) = c[i]\right] \geq 1 - \frac{1}{n^{\log^2 n}}.$$

Moreover, Dec_1^w *makes at most* $\mathcal{O}\left(\log^3 n\right)$ *queries to* w.

Proof : First, note that for any codeword obtained from Enc, Dec_1 will always output the correct symbol.

Let w be the received word which is obtained by altering at most $\frac{\Delta_J k}{4}$ coordinates of some codeword c obtained from Enc, i.e., $0 < \mathsf{dist}(c, w) \leq \frac{\Delta_J k}{4}$. From Enc, we know that the last k' blocks of any codeword are exactly the same. Since $\mathsf{dist}(c, w) \leq \frac{\Delta_J k}{4}$, at most $\frac{1}{4}$ fraction of these blocks are modified in more than Δ_J fraction of their coordinates. Therefore, at least $\frac{3}{4}$ fraction of the last k' blocks can be corrected to a unique codeword closest to them. Thus, each query finds an correct codeword block with probability at least $\frac{3}{4}$. Then by standard Chernoff bounds, the probability that the majority of the $\Theta\left(\frac{\log^3 k}{\ell(\lambda)}\right)$ queries are correct is at least $1 - e^{-\Omega(s)}$, where $s = \Theta\left(\frac{\log^3 k}{\ell(\lambda)}\right)$ is the number of sampled blocks.

Since Dec_1 queries $\mathcal{O}\left(\frac{\log^3 k}{\ell(\lambda)}\right)$ blocks each of length $4\ell(\lambda)$ bits, the complexity of Dec_1 is at most $\mathcal{O}\left(\log^3 k\right) = \mathcal{O}\left(\log^3 n\right)$. □

5.2.2 Query Procedure for $i < 8k - 4\ell(\lambda)$

In this section, we describe the algorithm for recovering a coordinate for input $i < 8k - 4\ell(\lambda)$. Before we describe the algorithm, we need a few definitions and properties of the code we constructed in Section 5.1.

Recall that for a codeword obtained from Enc, the first k' blocks of length $4\ell(\lambda)$ each, corresponds to the encoding of the message symbols, and the next k' blocks correspond to the encoding of labels of the nodes of a fixed δ-local expander graph $G = ([k'], E)$ generated using the k' message blocks.

In the next lemma, we describe an algorithm which verifies whether a given node of G is green with respect to the labels obtained from the received vector $w \in \{0,1\}^n$. Recall from the definition of a green node that we need to query only the labels of the parents of a given vertex v to verify if it is green. Since the indegree of G is $\mathcal{O}(\log k')$, we can check if a particular node is green by making at most $\mathcal{O}(\log k')$ block queries to w. We now formalize the verification procedure.

Lemma 5.5 *Let $G = ([k'], E)$ be δ-local expander with indegree $\mathcal{O}(\log k')$ used by Enc and let $w = (w_1, \ldots, w_{3k'-1}) \in \{0,1\}^n$ be the corrupted word obtained from the PPT adversary. There exists an algorithm $\mathsf{IsGreen}$ that uses $\mathcal{O}(\ell(\lambda) \log(n/\ell(\lambda)))$ coordinate queries to w and outputs whether a given node $v \in [k']$ of G is green or not.*

Proof : We claim the following algorithm achieves the desired properties:

$\mathsf{IsGreen}$: Input: Node $v \in [k']$ of G with indegree $d = \mathcal{O}(\log k')$.

(1) Query and decode the blocks $w_{k'+v_j}$ for $v_j \in \mathsf{parents}_G(v)$. Let $\ell'_{k'+v_j} := \mathsf{ECCD}(w_{k'+v_j})$. Return 'Red' if ECCD fails on any input.

(2) Query and decode the block w_v. Let $x'_v := \mathsf{ECCD}(w_v)$. Return 'Red' if ECCD fails.

(3) If $\mathsf{ECCD}(w_{k'+v}) = H(s, x'_v \circ w_{k'+v_1} \circ w_{k'+v_2} \circ \cdots \circ w_{k'+v_d})$, where $v_j \in \mathsf{parents}_G(v)$ for $j \in [d]$, then output 'Green'

(4) Else output 'Red'.

$\mathsf{IsGreen}$ outputs whether or not the label of node v, *i.e.* $\mathsf{ECCD}(w_{k'+v})$, is green by querying at most $\mathcal{O}(\log k')$ blocks of w. Since each block is $4\ell(\lambda)$ bits long, and $k' = k/\ell(\lambda) = \mathcal{O}(n/\ell(\lambda))$, the number of bits queried is equivalently $\mathcal{O}(\ell(\lambda)\log(n/\ell(\lambda)))$. □

Let $w \in \{0,1\}^n$ be the received word obtained from the PPT adversary. Let $S \subseteq [k']$ be the set of red nodes of G with respect to its labeling obtained from w. Let $0 < \alpha < 1$. Recall from Definition 4.7 that we call a node $i \in [k']$ of G to be α-good under the set $S \subseteq [k']$ if for all $r > 0$, the number of nodes of S in any r-sized neighborhood of i is at most $\alpha \cdot r$. We now give an algorithm to locally verify if a given node of G is α-good with high probability.

Lemma 5.6 *For any $\alpha < \frac{3}{4}$, there exists a procedure that makes $\mathcal{O}\left(\ell(\lambda)\log^{3+\epsilon}\left(\frac{n}{\ell(\lambda)}\right)\right)$ coordinate queries to w and for any node $i \in [k']$ of G does the following:*

- *Accepts if i is $\frac{\alpha}{4}$-good under S with probability $1 - \mathsf{negl}(n)$.*
- *Rejects if i is not α-good under S with probability $1 - \mathsf{negl}(n)$.*

Proof : Consider the following algorithm IsGood, where S is defined to be the set of red nodes:

IsGood: Input: Node $i \in [k']$ of G, $\alpha < \frac{3}{4}$, space parameter $\epsilon > 0$.

(1) If i is not *green*, then Reject.

(2) For every $p \in \{1, \ldots, \log k'\}$ do:

- Sample $\log^{1+\epsilon} k'$ nodes $U_{1p} \subseteq [i - 2^p + 1, i]$ and $U_{2p} \subseteq [i, i + 2^p - 1]$.
- If the fraction of red nodes in U_{1p} or U_{2p} is larger than $\frac{4}{3}\alpha$, then Reject.

(3) Accept otherwise.

Note that IsGood samples at most $\mathcal{O}\left(\log^{2+\epsilon} k'\right)$ nodes of G and for each of those nodes, we check if it is green. Using Algorithm $\mathsf{IsGreen}$ described in Lemma 5.5, we can verify if a node is green using only $\mathcal{O}(\log k')$ block queries to w. Therefore the number of coordinates of w queried by IsGood is at most $\mathcal{O}\left(\ell(\lambda)\log^{3+\epsilon}(n/\ell(\lambda))\right)$.

If the node i is red, then $i \in S$ and therefore by definition it is not α-good for any α. On the other hand, if i is $\alpha/4$-good, then we know that for any r-sized neighborhoods of i, i.e $[i - r + 1, i]$ and $[i, i + r - 1]$ have at most $\alpha \cdot r/4$ red nodes. In particular, for all $p \in [\log k']$, the neighborhoods $U_{1p} = [i - 2^p + 1, i]$ and $U_{2p} = [i, i + 2^p - 1]$ contain at most $\alpha \cdot 2^{p-2}$ red nodes. So on sampling $\log^{1+\epsilon} k'$ nodes uniformly at random from these intervals of size 2^p, the probability that we sample at least $\frac{4}{3}\alpha \log^{1+\epsilon} k'$ red nodes is at most $e^{-13\alpha \log^{1+\epsilon}(k')/12}$. Taking a union bound over all $\log k'$ intervals, IsGood accepts any $\alpha/4$-good node with probability at least $1 - \mathsf{negl}(k') = 1 - \mathsf{negl}(n)$ for $\ell(\lambda) = \mathsf{polylog}\, n$.

We now show that any node which is not α-good under S is not $\alpha/4$-good for some interval of size 2^p, $p > 0$. Therefore, IsGood rejects any node which is not α-good with probability at least $1 - \mathsf{negl}(k')$. If node i is not α-good under S, then there exists a neighborhood I of i such that the number of red nodes in I is at least $\alpha|I|$. Let $R(X)$ denote the number of red nodes for some subset $X \subseteq V$. Let $U_1 \subseteq I \subseteq U_2$ where $|U_1| = 2^{p^*}$ and $|U_2| = 2^{p^*+1}$ for some integer $p^* > 0$. Then

$$R(U_2) \geq R(I) \overset{*}{\geq} \alpha|I| \geq \alpha|U_1| = \alpha \cdot 2^{p^*} \geq \frac{\alpha}{4} \cdot 2^{p^*+1}.$$

where the inequality (*) results from i not being α-good. $\square$

Let (w, i, x) be the challenge provided by any PPT adversary such that $\mathsf{dist}(c, w) \leq \tau$, where $c = \mathsf{Enc}(x)$. Recall that using Algorithm Dec_1 we can successfully correct any index $i \geq 8k - 4\ell(\lambda)$. Therefore, to design Dec_2, we assume the correctness of node k' using Dec_1. The procedure Dec_2 first checks whether nodes i and node k' are α-good for some $\alpha < 3/4$. If they are both α-good, then by Lemma 4.9 there is a path of green nodes connecting i and node k'. Node k' now serves as an anchor point by being both α-good and correct due to the repetition code, so then Lemma 5.3 implies node i is also unaltered. We now formalize this and describe Dec_2 in Lemma 5.7.

Lemma 5.7 *Let* Enc *be as described in Section 5.1. For any* $1 \leq i < 8k - 4\ell(\lambda)$*, there exists a procedure* Dec_2 *with the following properties:*

(1) For any $x \in \{0,1\}^k$ *and* $c = \mathsf{Enc}(s, x)$, $\mathsf{Dec}_2^c(s, i) = c[i]$.

(2) For any $x \in \{0,1\}^k$, $c = \mathsf{Enc}(s, x)$ *and* $w \in \{0,1\}^n$ *generated by any PPT adversary such that* $\mathsf{dist}(c, w) \leq \frac{\Delta_J k}{4}$,

$$\mathbf{Pr}\left[\mathsf{Dec}_2^w(s, i) \in \{c[i], \perp\}\right] \geq 1 - \mathsf{negl}(n).$$

Moreover, Dec_2 *makes at most* $\mathcal{O}\left(\ell(\lambda)\log^{3+\epsilon}\left(n/\ell(\lambda)\right)\right)$ *queries to input* w *for any constant* $\epsilon > 0$.

Proof : We claim the following procedure has the desired properties.

Dec_2^w: Input: Index $i < 8k - 4\ell(\lambda)$.

(1) Let S be the set of red nodes of G with respect to the labels obtained from w.

(2) Let $\alpha < \frac{3}{4}$.

(3) Reconstruct the label of k'-th node, $\ell_{k',s}$ using a call to Dec_1.

(4) If node k' of G with label $\ell_{k',s}$ and node $\left\lceil \frac{i}{4\cdot\ell(\lambda)} \right\rceil \mod k'$ are α-good under S, then return w_i,

(5) Else return $\perp$.

First we show that the query complexity of Dec_2 is at most $\mathcal{O}\left(\ell(\lambda)\log^{3+\epsilon}\left(n/\ell(\lambda)\right)\right)$. Observe that Dec_1 described in Lemma 5.4 uses majority decoding to reconstruct the entire block $w_{2k'+j}$, for any $0 \leq j \leq k' - 1$. Moreover, it uses at most $\mathcal{O}\left(\log^3 n\right)$ coordinate queries to w in order to reconstruct any of the last k' blocks with $1 - \mathsf{negl}(n)$. Therefore we can assume that after Step (3), the label $\ell_{k',s}$ of G is correct with very high probability. Also, using the procedure described in Lemma 5.6, we can check if both the nodes k' and i are α-good using at most $\mathcal{O}\left(\ell(\lambda)\log^{3+\epsilon}\left(n/\ell(\lambda)\right)\right)$ queries for any $\epsilon > 0$. Therefore the query complexity of Dec_2 is $\mathcal{O}\left(\ell(\lambda)\log^{3+\epsilon}\left(n/\ell(\lambda)\right)\right)$.

To show (1), observe that if w is a codeword produced by Enc, then the label $\ell_{k',s}$ is reconstructed correctly with probability 1 by Dec_1 in the first step. Also since $S = \emptyset$, all the nodes of G are α-good. Therefore, Dec_2 always returns the correct codeword symbol.

Now, if w is any string produced by a PPT adversary such that $0 < \mathsf{dist}(c, w) \leq \frac{\Delta_J k}{4}$, then from the analysis of Dec_1 in Lemma 5.4, we know that $\ell_{k',s}$ is reconstructed correctly with probability at least $1 - \mathsf{negl}(n)$. If both nodes i and k' of G are α-good under S, then by Lemma 4.9, we know

that there exists a path in G from node i to node k' in G which consists of only green nodes. So, node i is a green ancestor of node k'. The correctness of w_i then follows from Lemma 5.3. □

We now combine the two correctors described above to obtain our local corrector Dec as follows:

Dec^w: Input : Index i.

(1) If $i \geq 8k - 4\ell(\lambda)$, return $\mathsf{Dec}_1^w(s, i)$

(2) Else return $\mathsf{Dec}_2^w(s, i)$.

Lemma 5.8 *Let* Enc *be as described in Section 5.1. For any* $i \in [n]$, Dec *does the following:*

(1) For any $x \in \{0,1\}^k$ *and* $c = \mathsf{Enc}(s, x)$, $\mathsf{Dec}^c(s, i) = c[i]$.

(2) For any $x \in \{0,1\}^k$, $c = \mathsf{Enc}(s, x)$ *and* $w \in \{0,1\}^n$ *generated by any PPT adversary such that* $\mathsf{dist}(c, w) \leq \frac{\Delta_J k}{4}$,

$$\mathbf{Pr}\left[\mathsf{Dec}^w(s, i) \in \{c[i], \perp\}\right] \geq 1 - \mathsf{negl}(n).$$

Moreover, Dec *makes at most* $\mathcal{O}\left(\ell(\lambda) \log^{3+\epsilon}(n/\ell(\lambda))\right)$ *queries to input* w *for any constant* $\epsilon > 0$.

Proof : The proof follows from Lemma 5.7 and Lemma 5.4. Since $n = (3k' - 1) \cdot 4\ell(\lambda)$, any query on an index $i < 8k - 4\ell(\lambda)$ is handled by Lemma 5.7 while a query on an index $i \geq 8k - 4\ell(\lambda)$ is handled by Lemma 5.4. Since each scenario uses at most $\mathcal{O}\left(\ell(\lambda) \log^{3+\epsilon}(n/\ell(\lambda))\right)$ queries, the algorithm also uses $\mathcal{O}\left(\ell(\lambda) \log^{3+\epsilon}(n/\ell(\lambda))\right)$ queries. Note that for $\ell(\lambda) = \mathsf{polylog}\, n$, the query complexity of the decoder is $\mathsf{polylog}\, n$. □

We now show that our construction $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a CRLCC scheme.

Proof of Theorem 2.4: Gen on input a security parameter λ simulates the generator algorithm GenH of the CRHF to output a random seed s.

Consider $s \leftarrow \mathsf{Gen}(1^\lambda)$, Enc described in Section 5.1 and the decoder Dec described in Section 5.2 above. We claim that the triple $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a CRLCC scheme.

From the construction of Enc, we know that the block length of a codeword produced by Enc is $n = (3k' - 1) \cdot 4 \cdot \ell(\lambda) < 12k$. Therefore, the information rate of the CRLCC is $\mathcal{O}(1)$.

From Lemma 5.8 we know that on input (w, i, x) generated by any PPT adversary such that $\mathsf{dist}(\mathsf{Enc}(x), w) \leq \frac{\Delta_J k}{4}$, Dec^w queries at most $\mathcal{O}\left(\ell(\lambda) \log^{3+\epsilon}(n/\ell(\lambda))\right)$ coordinates of w and returns $d \in \{\mathsf{Enc}(x)[i], \perp\}$ with probability at least $1 - \mathsf{negl}(n)$. Also, Dec on input any valid encoding $(\mathsf{Enc}(x), i, x)$ returns $\mathsf{Enc}(x)[i]$ with probability 1. □

6 Strong-CRLCC

In this section, we give an improved construction that locally corrects a majority of input coordinates.

The Barrier: The challenge in correcting a constant fraction of the nodes in the previous construction is that an adversary that is permitted to change $\mathcal{O}\left(\frac{n}{\log n}\right)$ symbols can turn all nodes red in a graph that has outdegree $\mathcal{O}(\log n)$.

Key Idea: We overcome the above mentioned barrier using three ideas. First, we run the original δ-local expander G_t through an indegree reduction gadget to obtain a new graph G with $k' = t \cdot m$ nodes in which *almost every* node has at most 2 incoming/outgoing edges and each node u in G_t (called the *meta-node*) corresponds to m nodes $u_1, \ldots, u_m$ in G.

Second, we introduce the notion of *green meta-node* u and show that the same key properties we used for the construction of Weak-CRLCC still hold with respect to the meta-nodes. A meta-node $u = \{u_1, \ldots, u_m\}$ is defined to be correct if all of its corresponding nodes $u_1, \ldots, u_m$ are correct (i.e., $\ell'_{u_j,s} = \ell_{u_j,s}$ and $x'_{u_j} = x_{u_j}$ for each $j \leq m$). However, we stress that the definition of a green meta-node does not require that *all* u_j are green. Instead we require that u_m is green and *most* of the nodes $u_1, \ldots, u_{m-1}$ are green. Crucially, this definition transfers the depth robustness properties of the δ-local expander G_t to G. In particular, if a meta-node v is green and correct and there is a directed path from u to v in G_t containing *only green* meta-nodes, then u must also be correct (or we will find a hash collision).

Third, we group the labels $\ell_{u_1,s}, \ldots, \ell_{u_m,s}$ into blocks before encoding them using a good error correcting code with constant rate and constant distance. This forces the attacker to flip *at least* $\Omega(m \cdot \ell(\lambda))$ bits to tamper *any* of the meta-nodes. In this way we can ensure that an attacker must tamper with *at least* $\Omega(t \cdot m \cdot \ell(\lambda))$ bits to make $\Omega(t)$ meta-nodes red. Therefore, by choosing appropriate parameters t and m, we get the Strong-CRLCC.

Metagraph

We will define the notion of a metagraph used in our approach to construct Strong-CRLCCs. For any graph $G = (V, E)$, let $M_1, M_2, \ldots M_m$ be a partition of the vertex set V. We call the graph $G_m = (V_m, E_m)$ with the following properties to be the *metagraph* of G.

Nodes: For every partition M_i of V, there is a unique node $v_i \in V_m$. We call v_i the *simple node* and M_i its *meta-node*.

Edges: There exists an edge between v_i and v_j in G_m if and only if there exists an edge between u and v in G for some $u \in M_i$ and $v \in M_j$.

Graph Degree Reduction

We now describe our procedure ReduceDegree that reduces the degree of many nodes in a graph, while simultaneously yielding a desirable meta-graph.

ReduceDegree: (Let G_0 be a δ-local expander with $\delta = 1/100$.)
Input: Graph G_0 with t vertices and $m = \max\{\mathsf{indeg}(G_0), \mathsf{outdeg}(G_0)\} + 1$
Output: Graph G with $m \cdot t$ vertices.

- For each node u in G_0 we add m nodes $u_1, \ldots, u_m$ to G. Along with each of the directed edges (u_i, u_{i+1}) and (u_i, u_m) for $i < m$.
- For each directed edge (u, v) in G_0 we connect the final node u_m to the first node v_j that currently has indegree *at most* 1.

Again, the choice of $\delta = 1/100$ is only to satisfy Lemma 4.9.

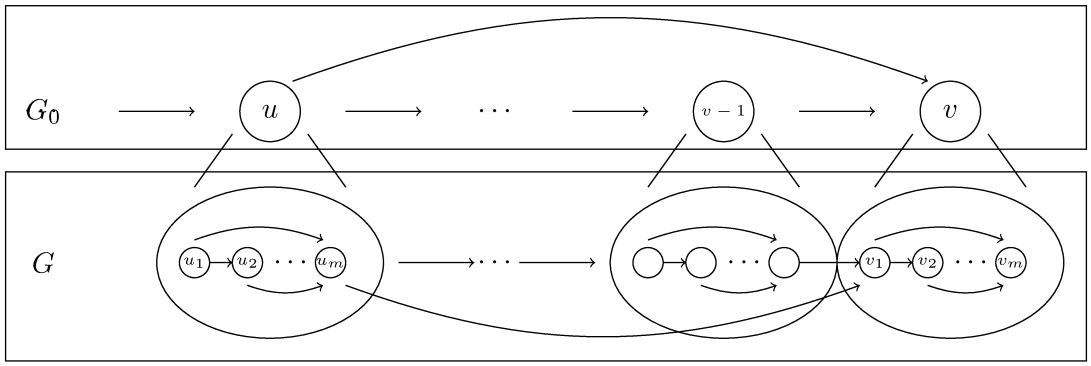

Fig. 2: An example of a degree reduction graph

6.1 Enc Algorithm

In this section we describe the Enc algorithm which takes as input a seed $s \leftarrow \mathsf{Gen}\left(1^{\lambda}\right)$ and a message $x \in \{0,1\}^k$ and returns a codeword $c \in \{0,1\}^n$. For a security parameter λ, let $H : \{0,1\}^* \to \{0,1\}^{\ell(\lambda)}$ be a CRHF (see Definition 4.3). Recall that, ECC and ECCD denote the encoding and decoding algorithms of a good code of rate 1/4 that can decode efficiently from Δ_J fraction of errors.

For $t = \mathcal{O}\left(\frac{k}{\ell(\lambda)\cdot\log k}\right)$, let $G_0 = ([t], E)$ be a δ-local expander graph on t vertices and degree (indegree and outdegree) $m = \mathcal{O}(\log t)$. Let $G := \mathsf{ReduceDegree}(G_0)$ be the graph with $k' := \frac{k}{\ell(\lambda)}$ nodes output by the degree reducing procedure applied to G_0.

Let $s \leftarrow \mathsf{Gen}(1^{\lambda})$
Enc(s, x):
Input: $x = (x_1, \cdots, x_{k'}) \in \left(\{0,1\}^{\ell(\lambda)}\right)^{k'}$, where $k = k' \cdot \ell(\lambda)$
Output: $c = (c_1, \cdots, c_{3t-1}) \in \left(\{0,1\}^{4m\cdot\ell(\lambda)}\right)^{3t-1}$, where $t = k'/m$ and $n = \mathcal{O}(k)$.

- Let $(\ell_{1_1,s}, \ldots, \ell_{1_m,s}, \ldots, \ell_{t_1,s} \ldots, \ell_{t_m,s}) = \mathsf{lab}_{G,s}(x)$ be the labeling of the graph G with input x using the CRHF (see Definition 5.1).
- Let $U_1 := (\ell_{1_1,s}, \ell_{1_2,s}, \cdots, \ell_{1_m,s}), \ldots, U_t := (\ell_{t_1,s}, \ell_{t_2,s}, \cdots, \ell_{t_m,s})$.
- Let $T_1 := (x_1, x_2, \ldots, x_m), \ldots, T_t := (x_{(t-1)m+1}, x_{(t-1)m+2}, \cdots, x_{tm})$.
- The codeword c consists of the encoding of groups of message bits, followed by encoding using ECC of groups of labels, followed by $t = \frac{k'}{m}$ copies of the last encoding.

$$c = (\mathsf{ECC}(T_1), \mathsf{ECC}(T_2), \cdots, \mathsf{ECC}(T_t), \mathsf{ECC}(U_1), \mathsf{ECC}(U_2), \cdots, \mathsf{ECC}(U_t), \underbrace{\mathsf{ECC}(U_t), \cdots, \mathsf{ECC}(U_t)}_{t \text{ times}}),$$

where $\mathsf{ECC}(U_i) = \mathsf{ECC}(\ell_{i_1,s}, \ldots, \ell_{i_m,s})$ for all $i \in [t]$.

Lemma 6.3 is the key lemma that did not hold for our previous construction. It states that if for any $w \in \{0,1\}^n$, $\mathsf{dist}(w, c) \leq \frac{\Delta_J \cdot k}{4}$ then the number of red meta-nodes is very small. Lemma 6.6

is analogous to Lemma 5.3 and asserts that if there is a green path from meta-node u to meta-node v and v is correct then meta-node u must also be correct. It follows that *any* meta-node $v < \frac{t}{4}$ that is α-good must also be correct because v is connected to all other α-good meta nodes, and there are too many α-good meta nodes for the attacker to corrupt them all. With this observation it is relatively straightforward to extend the ideas from the previous section to build a strong Dec with locality $\mathcal{O}(\mathsf{polylog}\, n)$.

We also remark that it is possible to allow the rate of our code to approach 1. For example, we can split the initial input x into chunks of size $x_i \in \{0,1\}^{100\lambda}$ so that the size of each label $\ell_{i,s}$ is *significantly* smaller than the size of the corresponding chunk (We would also need to modify the underlying ECC so the rate of this code is also close to 1).

6.2 Dec Algorithm

Observe that at the end of the construction, each meta-node contains exactly one node (*i*.e. the final node) with indegree more than 2 (it has indegree m) and outdegree more than 2 (it has outdegree $\mathcal{O}(m)$).

The codeword $c = \mathsf{Enc}(x)$ consists of 3 parts. The first $4k$ bits correspond to the message symbols x passed $m \cdot \ell(\lambda)$ bits at a time through ECC. The next $4k$ bits correspond to t codewords $\mathsf{ECC}(U_j)$, and finally the last $4k - 4m \cdot \ell(\lambda)$ bits correspond to the repetitions of the final ECC codeword, $\mathsf{ECC}(U_t)$. Therefore the length of any codeword produced by Enc is $\mathcal{O}(n)$ since $m = \mathcal{O}(\log k)$.

Claim 6.1 $\mathsf{outdeg}(G_0) = \Theta(\log k)$.

Definition 6.2 *We call a meta-node M_i* green *if both:*

- *At least $1 - \Delta_J$ fraction of the underlying nodes in M_i are green.*
- *The final node in M_i is green.*

We call a meta-node red *if the meta-node is not green. Similarly, we call a meta-node M_i* correct *if we are able to correctly decode M_i. Observe that M_i can contain tampered nodes but still be correct.*

Observe that encoding the labels of a meta node using a good code, we ensure that a meta-node can only be corrupted by changing $4m \cdot \ell(\lambda) \cdot \Delta_J$ bits of the meta-node or finding a hash collision:

Lemma 6.3 *If $\mathsf{dist}(w, c) \leq \frac{\Delta_J k}{4}$, then at most $\frac{t}{4}$ meta-nodes in G are red.*

Proof : Observe that a meta-node can be made red by changing at least $4m \cdot \ell(\lambda) \cdot \Delta_J$ bits or by finding a hash collision. Therefore, by changing at most $\frac{\Delta_J k}{4}$ bits of a codeword, an adversary can make at most $\frac{k}{4m\ell(\lambda)} = \frac{k'}{4m} = \frac{t}{4}$ meta-nodes in G red. □

Similar to the decoder Dec of Section 5.2, which splits the decoding procedure for the first $8k - 4\ell(\lambda)$ indices and a separate decoding procedure for the remaining indices, the current decoding procedure has separate subroutine to handle indices corresponding to the first $2t - 1$ blocks and a separate decoding procedure for the remaining indices. In the following, we define $m(i)$ to be the meta-node corresponding to index i. That is,

$$m(i) = \begin{cases} \left\lceil \frac{i}{4m\ell(\lambda)} \right\rceil, & i \leq 4k \\ \left\lceil \frac{i-4k}{4m\ell(\lambda)} \right\rceil, & 4k < i \leq 8k, \\ t, & i > 8k \end{cases}$$

6.2.1 Query Procedure for $m(i) = t$

In the case that $m(i) = t$, we run a procedure $\mathsf{Dec}^w_{\geq t}(s, i)$ that takes advantage of the fact that any adversary that can change at most $\frac{\Delta_J k}{4}$ coordinates of any codeword obtained from Enc. Thus, the adversary cannot prevent local correction for a query on the last t groups corresponding to the repetition of ECC codeword. Similar to Dec^w_1 in Section 5.2, the algorithm $\mathsf{Dec}^w_{\geq t}(s, i)$ randomly samples a number of points, takes the majority of the samples, and performs decoding using ECCD.

Lemma 6.4 *Let* Enc *be as described in Section 6.1. For any* $i \in [n]$ *such that* $m(i) = t$*, there exists an algorithm* $\mathsf{Dec}^w_{\geq t}$ *that does the following:*

(1) For any $x \in \{0,1\}^k$ *and* $c = \mathsf{Enc}(s, x)$, $\mathsf{Dec}^c_{\geq t}(s, i) = c[i]$.

(2) For any $x \in \{0,1\}^k$, $c = \mathsf{Enc}(s, x)$ *and* $w \in \{0,1\}^n$ *generated by any PPT adversary such that* $\mathsf{dist}(c, w) \leq \frac{\Delta_J k}{4}$,

$$\mathbf{Pr}\left[\mathsf{Dec}^w_{\geq t}(s, i) = c[i]\right] \geq 1 - \frac{n}{n^{\log n}}.$$

Moreover, $\mathsf{Dec}^w_{\geq t}$ *makes at most* $\mathcal{O}\left(\ell(\lambda) \cdot \log^2 n\right)$ *queries to input* w.

Proof : Consider the following algorithm $\mathsf{Dec}^w_{\geq t}(i)$ for any $i \in [n]$ with $m(i) \geq t$:

$\mathsf{Dec}^w_{\geq t}$: Input : Index $i \in [n]$ with $m(i) \geq t$.

(1) For each $0 \leq j < 4m\ell(\lambda)$, sample $\mathcal{O}(\log n)$ indices of w greater than $8k - 4m\ell(\lambda)$ and congruent to $j \pmod{4m\ell(\lambda)}$ uniformly at random.

(2) For each $0 \leq j < 4m\ell(\lambda)$, choose the symbol $\tilde{w}_j$ that corresponds to the majority of the samples from W_j.

(3) Correct the word $(\tilde{w}_1, \cdots, \tilde{w}_{4m\ell(\lambda)})$ using ECCD and output index $i \pmod{4m\ell(\lambda)}$.

First, note that for any codeword obtained from Enc, $\mathsf{Dec}_{\geq t}$ will always output the correct symbol.

Let w be received word which is obtained by altering at most $\frac{\Delta_J k}{4}$ coordinates of some codeword i.e, $0 < \mathsf{dist}(c, w) \leq \frac{\Delta_J k}{4}$ for some codeword c obtained from Enc. We know that the last t coordinates $\pmod{4m\ell(\lambda)}$ of any codeword are exactly the same, $c[j] = c[j + 4m\ell(\lambda)]$ for all $j \geq 8k - 4m\ell(\lambda)$. Since $\mathsf{dist}(c, w) \leq \frac{\Delta_J k}{4}$, at most $\frac{\Delta_J}{8}$ of such coordinates $j \pmod{4m\ell(\lambda)}$ are corrupted in more than $1/4$ fraction of the blocks. For the remaining j, each query finds an unmodified symbol with probability at least $1 - \frac{\Delta_J}{4}$. Then by standard Chernoff bounds over the $\mathcal{O}(\log n)$ random queries for each $j \pmod{4m}$, the probability that the majority of the $\mathcal{O}\left(\log^2 n\right)$ queries are unmodified is at least $1 - \frac{1}{n^{\log n}}$. Then by a simple union bound, the probability that the majority of the symbols in meta-node M_t are corrected is at least $1 - \frac{n}{n^{\log n}}$.

Since $\mathsf{Dec}_{\geq t}$ makes $\mathcal{O}(\log n)$ queries for each of the $m \cdot \ell(\lambda)$ congruence classes and $m = \mathcal{O}(\log n)$, then $\mathcal{O}\left(\ell(\lambda) \cdot \log^2 n\right)$ queries are made in total. □

6.2.2 Query Procedure for $m(i) < t$

Observe from the construction that G_0 is the metagraph of G Since G_0 is a δ-local expander, then for any $0 < \alpha < 1$ and subset S of vertices, the metagraph $G_0 - S$ contains at least $t - |S|\left(\frac{2}{\alpha} - 1\right)$ α-good nodes by Lemma 4.8. Also, from Lemma 4.9, we know that any two α-good nodes are connected by a path in $G_0 - S$. Thus, by letting S be the set of all red meta-nodes, any two α-good meta-nodes in G_0 are connected by a path of green meta-nodes. We use these observations to design the corrector for $m(i) < t$.

First, we claim the existence of a procedure to test whether a meta-node is green.

Lemma 6.5 *Let $w = (w_1, \cdots w_{3t-1}) \in \{0,1\}^n$. There exists a procedure* IsGreenMeta *that uses coordinate $\mathcal{O}\left(\ell(\lambda) \cdot \log^2 n\right)$ queries to w and checks whether a meta-node i of G_0 with labels corresponding to w is green.*

Proof : We claim the following procedure satisfies the desired properties:

IsGreenMeta(i): Input : Meta-node index $i \in [t]$.

(1) Check if the final node in meta-node i is green:

- Query coordinates of w_{t+i} and retrieve $(\ell'_{i_1}, \ldots, \ell'_{i_m}) := \mathsf{ECCD}(w_{t+i})$.
- Query the coordinates of w_i and retrieve $(x'_{i_1}, \ldots, x'_{i_m}) := \mathsf{ECCD}(w_i)$.
- Check whether $\ell'_{i_m} = H(s, x'_{i_m} \circ \ell'_{i_1} \circ \cdots \circ \ell'_{i_{m-1}})$ for the final node i_m in meta-node i, where $i_1, \ldots, i_{m-1}$ are the parents of i_m, which are all in meta-node i.
- If ℓ'_{i_m} is not consistent, then output 'Red'.

(2) For each node i_j in meta-node i, check whether i_j is green:

- Let i_{j-1} (in meta-node i) and $p(j)_r$ (in meta-node $p(j)$) be the parents of i_j.
- Query w_{t+r} and retrieve $\ell'_{p(j)_r}$ using $\mathsf{ECCD}(w_{t+p(j)})$.
- Check whether $\ell'_{i_j} = H(s, x'_{i_j} \circ \ell'_{i_{j-1}} \circ \ell'_{p(j)_r})$.

(3) If at least $1 - \Delta_J$ of the nodes in meta-node i are green, then output 'Green'.

(4) Else, output 'Red'.

Recall from Definition 6.2 that a green meta-node first requires that at least $1-\Delta_J$ of the underlying nodes in meta-node i are green. Since a node is green if its label is consistent with the labels of its parents, then the procedure IsGreenMeta properly recognizes whether at least $1 - \Delta_J$ of the underlying nodes in i are green after decoding using ECCD. Secondly, a green meta-node requires that the final node in i is green. In this case, the final node i_m has $m - 1$ parents, and again the procedure recognizes whether $\ell'_{i_m} = H(s, x'_{i_m} \circ \ell'_{i_1} \circ \cdots \circ \ell'_{i_{m-1}})$ after obtaining the labels using ECCD.

To see that the IsGreenMeta does at most $\mathcal{O}\left(\ell(\lambda) \cdot \log^2 n\right)$ coordinate queries, we observe that in Step (1), we query two blocks of w each of length $4m\ell(\lambda)$ bits. Also, in Step (2), for each of the $m-1$ nodes in the meta-node i we query one additional block of w to check if it is green. Therefore the total query complexity is $\mathcal{O}\left(m^2\ell(\lambda)\right) = \mathcal{O}\left(\ell(\lambda) \cdot \log^2 n\right)$. □

We now introduce the algorithm for recovering a coordinate for input i whose meta-node $m(i)$ is less than t.

$\mathsf{Dec}^{w}_{<t}$: Input : Index $i \in [n]$ with $m(i) < t$ of G, $\frac{1}{2} < \alpha < \frac{3}{4}$, space parameter $\epsilon > 0$.

(1) If meta-node $m(i)$ is not *green*, then return $\perp$.

(2) For every $p \in \{1, \ldots, \log t\}$ do:

- Sample $\log^{1+\epsilon} k$ meta-nodes $U_{1p} \subseteq [i - 2^p + 1, i]$ and $U_{2p} \subseteq [i, i + 2^p - 1]$.
- Use the procedure $\mathsf{IsGreenMeta}$ to check the fraction of red meta-nodes in U_{1p} or U_{2p}.
- If the fraction of red meta-nodes in U_{1p} or U_{2p} is larger than $\frac{4}{3}\alpha$, then return $\perp$.

(3) Otherwise, compute $\mathsf{ECCD}(w_{m(i)})$ if $i \leq 4k$ or, $\mathsf{ECCD}(w_{t+m(i)})$ if $i > 4k$ and return the $i \pmod{4m\ell(\lambda)}$-th coordinate.

We crucially use the following lemma to prove the correctness of $\mathsf{Dec}_{<t}$ procedure given in Lemma 6.7.

Lemma 6.6 *The meta-node parents of a meta-node v of G_0 that is green and correct must also be correct unless a hash collision was found.*

Proof : Let $v \in [t]$ be a meta-node that is green and correct and let u be a green meta-node parent of v in the metagraph G_0. Let the nodes in v be $v_1, \ldots, v_m$ and the nodes in u be $u_1, \ldots, u_m$. Let $(x'_{v_1}, \ldots, x'_{v_m}) := \mathsf{ECCD}(w_v)$ and $\ell' = (\ell'_{v_1}, \ldots, \ell'_{v_m}) := \mathsf{ECCD}(w_{t+v})$. If v is green and correct, then $\ell'_{v_t,s} = \ell_{v_t,s}$ and $x'_{v_t} = x_{v_t}$ for each $1 \leq t \leq m-1$. Let v_u be the node of v connected to u so that $\ell'_{v_u,s} = H(s, x'_{v_u} \circ \ell'_{v_{u-1},s} \circ \cdots \ell'_{v_{um},s})$. Similarly, $\ell_{v_u,s} = H(s, x_{v_u} \circ \ell_{v_{u-1},s} \circ \cdots \ell_{v_{um},s})$, so the label for u_m must be correct.

Since u is green, then $\ell'_{u_m,s} = H(s, x'_{u_m} \circ \ell'_{u_1,s} \circ \cdots \circ \ell'_{u_{m-1},s})$. Similarly, $\ell'_{u_m,s} = H(s, x_{u_m} \circ \ell_{u_1,s} \circ \cdots \circ \ell_{u_{m-1},s})$. Assuming no hash collisions are found, then $x_{u_m} = x'_{u_m}$ and $\ell'_{u_t,s} = \ell_{u_t,s}$ for each $1 \leq t \leq m-1$, so u is both green and correct (after decoding using ECCD). □

We now justify the correctness of the $\mathsf{Dec}_{<t}$ procedure.

Lemma 6.7 *Let Enc be as described in Section 6.1. For any $i \in [n]$ such that $m(i) < t$, $\mathsf{Dec}_{<t}$ does the following:*

(1) For any $x \in \{0,1\}^k$ and $c = \mathsf{Enc}(s, x)$, $\mathsf{Dec}^{c}_{<t}(s, i) = c[i]$.

(2) For any $x \in \{0,1\}^k$, $c = \mathsf{Enc}(s, x)$ and $w \in \{0,1\}^n$ generated by any PPT adversary such that $\mathsf{dist}(c, w) \leq \frac{\Delta_J}{4}$,

$$\mathbf{Pr}\left[\mathsf{Dec}^{w}_{<t}(s, i) \in \{c[i], \perp\}\right] \geq 1 - \mathsf{negl}(n).$$

Moreover, $\mathsf{Dec}^{w}_{<t}$ makes at most $\mathcal{O}\left(\ell(\lambda) \cdot \log^{4+\epsilon} n\right)$ queries to input w.

Proof : First we show that the query complexity of $\mathsf{Dec}_{<t}$ is at most $\mathcal{O}\left(\log^{4+\epsilon} n\right)$. Observe that we query $\log t$ indices p. For each p, we sample $\log^{1+\epsilon} k$ meta-nodes to test whether the meta-nodes are green. The $\mathsf{IsGreenMeta}$ procedure checks if a meta-node is green using $\mathcal{O}\left(\ell(\lambda) \cdot \log^2 n\right)$ coordinate queries to w. Therefore, the total number of queries is $\mathcal{O}\left(\ell(\lambda) \cdot \log^{4+\epsilon} n\right)$.

Note that if w is a codeword produced by Enc, then the meta-node $m(i)$ containing index i is reconstructed correctly by $\mathsf{Dec}_{<t}$ by $\mathsf{ECCD}(w_{m(i)})$ or $\mathsf{ECCD}(w_{m(i)+t})$ accordingly. Therefore, $\mathsf{Dec}_{<t}$ always returns the correct codeword symbol.

On the other hand, if w is any string produced by a PPT adversary such that $0 < \mathsf{dist}(c, w) \leq \frac{\Delta_J k}{4}$, then $\mathsf{Dec}_{<t}$ first checks whether the meta-node $m(i)$ containing i is α-good. Recall that G_0 is a δ-local expander by construction. Thus if meta-node $m(i)$ is α-good under the set of red meta-nodes S, then by Lemma 4.9, that there exists a path in G_0 from meta-node $m(i)$ to a later meta-node that is α-good that consists of only green nodes. Hence, node $m(i)$ is a green ancestor of a green and correct meta-node. The correctness of $w[i]$ then follows by applying a simple induction on Lemma 6.6. We can without loss of generality assume that the last meta-node *i.*e. t is correct and green using the $\mathsf{Dec}_{\geq t}$ described in Section 6.2.1. Therefore, we know the existence of such a later node which is green and correct. □

Combining the two decoders in Section 6.2.1 and Section 6.2.2, we get a decoder for CRLCC which uses at most $\mathcal{O}\left(\ell(\lambda) \cdot \log^2 n\right) = \mathsf{polylog}\, n$ for $\ell(\lambda) = \mathsf{polylog}\, n$. Now we show that we indeed get a Strong-CRLCC and the decoder does not return a $\perp$ for majority of the coordinate queries.

Lemma 6.8 *Let* Dec *be defined by running* $\mathsf{Dec}^w_{<t}$ *on inputs* $i \in [n]$ *where* $m(i) < t$ *and* $\mathsf{Dec}^w_{\geq t}$ *on inputs* i *such that* $m(i) = t$*. Then*

$$\mathbf{Pr}_i\left[\mathsf{Dec}^w(s, i) \neq c[i]\right] < \frac{1}{4}.$$

Proof : Recall from Lemma 6.3 that the number of red meta-nodes is at most $\frac{k'}{4m}$. By Lemma 4.8 the number of α-good nodes in G_0 is at least $\frac{k'}{m} - \frac{k'}{4m}\left(\frac{2}{\alpha} - 1\right)$. Thus, there exists $\alpha < 1$ so that the number of α-good nodes in G_0 is greater than $\frac{7k'}{12m}$. Since the adversary can only corrupt a small constant fraction of coordinates of the codeword, then by a simple Pigeonhole argument there is an α-good meta-node j that is also correct, for $j > \frac{t}{4}$. Therefore, each index i such that $m(i) \leq \frac{t}{4}$ that is in an α-good meta node has a descendant that is both α-good and correct. Hence, applying an induction on Lemma 6.6 shows that i is correctable. □

References

[ABH17] Joël Alwen, Jeremiah Blocki, and Ben Harsha. Practical graphs for optimal side-channel resistant memory-hard functions. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 1001–1017. ACM Press, October / November 2017. 2.1

[ABP17] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 3–32. Springer, Heidelberg, May 2017. 2.1

[ABP18] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In *Advances in Cryptology - EUROCRYPT - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, 2018. (to appear). 2.1, 2.1, 4, 4.6, 4, 4, 4.8, 4, 4.9, 3

[ALRW17] Alexandr Andoni, Thijs Laarhoven, Ilya P. Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 47–66, 2017. 1

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 595–603. ACM Press, June 2015. 2.1

[AT17] Joël Alwen and Björn Tackmann. Moderately hard functions: Definition, instantiations, and applications. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 493–526. Springer, Heidelberg, November 2017. 2.1

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 21–31, 1991. 1

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006. A preliminary version appeared in the Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC). 1

[BK95] Manuel Blum and Sampath Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, 1995. 1

[BL13] Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2013. 1

[BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993. 1

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. 1

[BZ17] Jeremiah Blocki and Samson Zhou. On the depth-robustness and cumulative pebbling cost of Argon2i. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 445–465. Springer, Heidelberg, November 2017. 2.1

[CGdW13] Victor Chen, Elena Grigorescu, and Ronald de Wolf. Error-correcting data structures. *SIAM J. Comput.*, 42(1):84–111, 2013. 1

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998. 1

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic*

Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding, pages 402–414, 1999. 1

[DGL04] Yan Ding, Parikshit Gopalan, and Richard Lipton. Error correction against computationally bounded adversaries. Manuscript, 2004. 1, 3

[DGY11] Zeev Dvir, Parikshit Gopalan, and Sergey Yekhanin. Matching vector codes. *SIAM J. Comput.*, 40(4):1154–1178, 2011. 1, 3

[DH09] Irit Dinur and Prahladh Harsha. Composition of low-error 2-query pcps using decodable pcps. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 472–481, 2009. 3

[Efr12] Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012. 1, 3

[EGS75] Paul Erdös, Ronald L. Graham, and Endre Szemeredi. On sparse graphs with dense long paths. Technical report, Stanford University, Stanford, CA, USA, 1975. 2.1, 4, 3

[Gas04] William I. Gasarch. A survey on private information retrieval (column: Computational complexity). *Bulletin of the EATCS*, 82:72–107, 2004. 1

[Gol06] Oded Goldreich. On post-modern cryptography. Cryptology ePrint Archive, Report 2006/461, 2006. http://eprint.iacr.org/2006/461. 1

[GRR18] Tom Gur, Govind Ramnarayan, and Ron D. Rothblum. Relaxed locally correctable codes. In *9th Innovations in Theoretical Computer Science Conference, ITCS*, pages 27:1–27:11, 2018. 1, 2

[GS16] Venkatesan Guruswami and Adam D. Smith. Optimal rate code constructions for computationally simple channels. *J. ACM*, 63(4):35:1–35:37, 2016. 1, 3

[HO08] Brett Hemenway and Rafail Ostrovsky. Public-key locally-decodable codes. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Proceedings*, pages 126–143, 2008. 1, 3

[HOSW11] Brett Hemenway, Rafail Ostrovsky, Martin J. Strauss, and Mary Wootters. Public key locally decodable codes with short keys. In *14th International Workshop, APPROX, and 15th International Workshop, RANDOM, Proceedings*, pages 605–615, 2011. 1, 3

[Jus72] Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Trans. Information Theory*, 18(5):652–656, 1972. 2.1, 4, 4.1, 5.1

[KdW04] Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *J. Comput. Syst. Sci.*, 69(3):395–420, 2004. 3

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. 4, 4.2, 4.3

[KM07] Neal Koblitz and Alfred J. Menezes. Another look at "provable security". *Journal of Cryptology*, 20(1):3–37, January 2007. 1

[KM15] Neal Koblitz and Alfred Menezes. The random oracle model: A twenty-year retrospective. Cryptology ePrint Archive, Report 2015/140, 2015. http://eprint.iacr.org/2015/140. 1

[KMRS17] Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. High-rate locally correctable and locally testable codes with sub-polynomial query complexity. *J. ACM*, 64(2):11:1–11:42, 2017. 1, 3

[KS16] Swastik Kopparty and Shubhangi Saraf. Guest column: Local testing and decoding of high-rate error-correcting codes. *SIGACT News*, 47(3):46–66, 2016. 1

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 80–86, 2000. 1

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. 1

[Lip94] Richard J. Lipton. A new approach to information theory. In *STACS, 11th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, pages 699–708, 1994. 1, 3

[Men12] Alfred Menezes. Another look at provable security (invited talk). In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, page 8. Springer, Heidelberg, April 2012. 1

[MPSW05] Silvio Micali, Chris Peikert, Madhu Sudan, and David A. Wilson. Optimal error correction against computationally bounded noise. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 1–16, 2005. 1, 3

[MR10] Dana Moshkovitz and Ran Raz. Two-query PCP with subconstant error. *J. ACM*, 57(5):29:1–29:29, 2010. A preliminary version appeared in the Proceedings of 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008). 3

[OPS07] Rafail Ostrovsky, Omkant Pandey, and Amit Sahai. Private locally decodable codes. In *Automata, Languages and Programming, 34th International Colloquium, ICALP, Proceedings*, pages 387–398, 2007. 1, 3

[SS96] Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Trans. Information Theory*, 42(6):1710–1722, 1996. 2.1

[STV99] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the XOR lemma (abstract). In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity, Atlanta, Georgia, USA, May 4-6, 1999*, page 4, 1999. 1

[Tre04] Luca Trevisan. Some applications of coding theory in computational complexity. *CoRR*, cs.CC/0409044, 2004. 1

[Yek08] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1):1:1–1:16, 2008. 1, 3

[Yek12] Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3):139–255, 2012. 1