Relaxed Locally Correctable Codes in Computationally Bounded Channels

Jeremiah Blocki, Venkata Gandikota, Elena Grigorescu, and Samson Zhou

***Abstract*— Error-correcting codes that admit *local* decoding and correcting algorithms have been the focus of much recent research due to their numerous applications. An important goal is to obtain the best possible tradeoffs between the number of symbols of the codeword that the local decoding algorithm must examine (the *locality*), 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, but these constructions require strong trusted setup assumptions. We study variants of locally decodable and locally correctable codes in computationally bounded, adversarial channels, in a setting with no trusted setup. The only assumption we require is the selection of the *public* parameters (seed) for a collision-resistant hash function. 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, which compare favorably with their classical analogs, crucially employ *collision-resistant hash functions* and *local expander graphs*, extending ideas from recent cryptographic constructions of memory-hard functions.**

***Index Terms*— Local codes, depth-robust graphs.**

Manuscript received July 24, 2019; revised December 31, 2020; accepted April 13, 2021. Date of publication April 28, 2021; date of current version June 16, 2021. The work of Jeremiah Blocki was supported in part by the National Science Foundation (NSF) under Grant CNS-1755708 and Grant CCF-1910659. The work of Elena Grigorescu was supported in part by the National Science Foundation (NSF) under Grant CCF-1649515, Grant CCF-1910659, and Grant CCF-1910411; and in part by the Purdue Research Foundation. The work of Samson Zhou was supported in part by the National Science Foundation (NSF) under Grant CCF-1649515 and in part by the Simons Investigator Award of David P. Woodruff. This article was presented in part at the Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP) and in part at the Proceedings of the 2019 IEEE International Symposium on Information Theory (ISIT). *(Corresponding author: Venkata Gandikota.)*

Jeremiah Blocki and Elena Grigorescu are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA (e-mail: jblocki@purdue.edu; elena-g@purdue.edu).

Venkata Gandikota was with the College of Information and Computer Sciences, University of Massachusetts, Amherst, MA 01003 USA. He is now with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13244 USA (e-mail: gandikota.venkata@gmail.com).

Samson Zhou was with Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA. He is now with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: samsonzhou@gmail.com).

Communicated by S. Lovett, Associate Editor for Complexity.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIT.2021.3076396.

Digital Object Identifier 10.1109/TIT.2021.3076396

I. 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 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) [41], [53]. 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., [6], [10], [12], [14], [18], [20], [42] and surveys [28], [54]). However, constructions of such codes suffer from apparently irreconcilable tension between locality and rate: existing codes with constant locality have slightly subexponential blocklength [24], [26], [55], and codes of linear blocklength have slightly subpolynomial query complexity [39]. For example, see surveys by Yekhanin [56] and by Kopparty and Saraf [40].

Ben-Sasson *et al.* [11] 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 relatively close to some codeword $c = \mathsf{Enc}(m) \in \Sigma^n$, the local decoder (1) outputs m_i if $w = c$; (2) otherwise, with probability $2/3$ outputs either m_i or $\perp$; and (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 $\mathsf{RLDC}s$ with constant query complexity and blocklength $n = k^{1+\epsilon}$.

Recently, Gur *et al.* [30] introduce the analogous notion of *relaxed locally correctable codes* ($\mathsf{RLCC}s$). 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 $\rho \cdot n$, for some $\rho > 0$. In fact, [30] 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 [11]. The reduction from [11], 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 [30] obtain significantly better parameters for $\mathsf{RLCC}s$ than for classical $\mathsf{LCC}s$; namely, they construct $\mathsf{RLCC}s$ with constant query complexity, polynomial block length, and constant error rate, and $\mathsf{RLCC}s$ with quasipolylogarithmic query complexity, linear blocklength (constant rate), with the caveat that the error rate is subconstant. These results immediately extend to $\mathsf{RLDC}s$, since their codes are *systematic*, meaning that the initial part of the encoding consists of the message itself.

In this work we study $\mathsf{RLDC}s$ and $\mathsf{RLCC}s$ in the more restricted, yet natural, *computationally bounded adversarial channel*, introduced by Lipton [44]. 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. Lipton argued that one can always reasonably assume that an adversarial channel is 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 [23], [31], [44], [47], [52] to show better error rate capabilities than in the Hamming model. See Section III for further details on related work.

Other work has focused on the construction of locally decodable codes when the sender and receiver have already exchanged cryptographic keys. Ostrovsky *et al.* [49] construct "private key" locally decodable codes with constant rate and a small (superconstant) locality against computationally bounded channels. Similarly, Hemenway and Ostrovky [32] and Hemenway *et al.* [33] construct public-key $\mathsf{LDC}s$ i.e., the encoding algorithm uses a secret key, but the decoder only needs to know the sender's public key. A crucial difference is that in our constructions of $\mathsf{RLDC}s$ and $\mathsf{RLCC}s$, the encoding algorithm is completely public and *does not* require the sender to use any secret key. Instead our constructions only rely on the existence of a collision-resistant hash function H, which will be used in both the encoding and decoding algorithms. In practice, one could instantiate H with SHA3 and no further setup assumptions are needed.

II. Our Contributions

We introduce the notion of *computationally relaxed locally correctable codes* (CRLCCs) which are analogous to $\mathsf{RLCC}s$ when the channel is computationally bounded. We construct CRLCCs with constant information and error rates as well as polylogarithmic locality, improving on [30] for bounded channels. Our codes are systematic, and therefore give *computationally relaxed locally decodable codes* (CRLDCs).

Since these codes interact with an adversarial channel, their strength is not only measured in their error correction and locality capabilities (as is the case for $\mathsf{RLCC}s$/$\mathsf{RLDC}s$ 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 Enc and Dec to denote encoding and decoding algorithms, respectively.

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 at position i of the closest codeword to w).

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

Definition 2.2: A *computational adversarial channel* $\mathcal{A}$ with error rate τ is an algorithm that interacts with a local code $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ 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. This public seed s is *fixed* once and for all.
(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 $w \in \Sigma^n$ and selects a challenge index $i \in [n]$; w is given to the receiver's Dec with query access along with the challenge index i.
(5) The receiver outputs $b \leftarrow \mathsf{Dec}^w(s, i)$.

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 $\mathsf{Dec}^w(s, i)$. We say that $\mathcal{A}(s)$ is γ-fooling Dec if $p_{\mathcal{A},s} > \gamma$. We say that $\mathcal{A}(s)$ is ρ-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 channel was γ-fooling Dec (resp. ρ-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 $(q,\tau,\rho(\cdot),\gamma(\cdot),\mu(\cdot))$-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 $\Pr[\mathsf{Fool}_{\mathcal{A},s}(\gamma(\lambda),\tau,\lambda)] \leq \mu(\lambda)$, 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 $\Pr[\mathsf{Limit}_{\mathcal{A},s}(\rho(\tau),\tau,\lambda)] \leq \mu(\lambda)$, where the randomness is taken over the selection of $s \leftarrow \mathsf{Gen}(1^\lambda)$ as well as $\mathcal{A}$'s random coins.

When $\mu(\lambda) = 0$, $\gamma(\lambda) = \frac{1}{3}$ is a constant 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 that satisfies conditions 1, 2 and 3 is a *Strong CRLCC* code.

Remark 2.4: We further explain the connections between our definition and the classical notion of RLDC*s*/RLCC*s*, and emphasize that the computational analogues are generalizations of the classical notions. We view the task of building a relaxed local decoder as a game in which the channel acts adversarially and tries to limit the ability of the decoder to correctly decode. We say that the channel "fools" the decoder if the decoder is unable to output the correct bit or $\perp$ except with some small probability γ. In our results we allow some negligible probability that the decoder is fooled for each i, while in the classical version the decoder is never fooled in outputing the correct answer with error at most $\gamma = 1/3$. Hence, this notion corresponds to the classical notion of success rate.

Also, note that just as in the classical notion, the set *Good* corresponds to the bits that should be decoded correctly w.h.p., and hence it relates to the notion of "relaxed decoding"; specifically, not all bits are decoded correctly w.h.p, but the set of those that are should be a constant fraction of all. In our results, we allow some negligible probability that the set Good is not large, and we refer to this case as in the channel "limits" the size of the correctly decoded bits.

Moreover, like in the case of the classical notion of RLCC, the notion of CRLCC requires decoding every index $i \in [n]$ w.h.p., and not just a random one. Also the construction should work for worst-case adversaries in a given class of adversaries. In the classical notion there is no limit to how much time or space the adversary can spend figuring out what entries of the received codeword to corrupt. In this paper we consider adversaries limited by polynomial time computations.

Again, as mentioned in the definitions above, by taking $\gamma = 1/3$, $\mu = 0$ and $\mathbb{A}$ to be the class of all possible adversaries, our definition exactly recovers the classical notion of RLCC*s*.

We construct Weak and Strong CRLCCs against probabilistic polynomial time (PPT) adversaries, under the assumption that *Collision-Resistant Hash Functions* (CRHF) exist. Briefly, a CRHF function is a pair (GenH,H) of algorithms, where GenH is a probabilistic polynomial time (PPT) algorithm that takes as input a security parameter λ and outputs a public seed $s \in \{0,1\}^*$; the algorithm $H : \{0,1\}^* \times \Sigma^* \to \Sigma^{\mathsf{L}(\lambda)}$, takes as input the seed s and a string $x \in \Sigma^*$ and outputs a hashed string $H(s,x)$ of length $\mathsf{L}(\lambda)$. We require that H is a deterministic algorithm running in polynomial time — the only randomness is the selection of the seed s by the generator GenH. The value $\mathsf{L}(\lambda)$ 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 a collision pair (x,x') is produced, i.e. $H(s,x) = H(s,x')$ and $x \neq x'$, is negligible in λ.

Theorem 2.5, our main result, states that it is possible to construct a constant information rate Strong CRLCC with polylog locality that withstands a constant error rate τ. Furthermore, the information rate $r(\tau)$ approaches 1 as τ approaches 0. By contrast, the classical RLCC*s* of [30] achieve constant information rate, but subconstant error rate and $(\log n)^{\mathcal{O}(\log\log n)}$ query complexity in the Hamming channel.

Theorem 2.5: Assuming the existence of a collision-resistant hash function (GenH,H) with length $\mathsf{L}(\lambda)$, there exist a constant $0 < \tau' < 1$ and negligible functions $\mu(\cdot),\gamma(\cdot)$ such that the following holds: for all $\tau \leq \tau'$, there are constants $0 < r(\tau),\rho(\tau) < 1$ and a $(\mathsf{L}(\lambda) \cdot \mathsf{polylog}\, n,\tau,\rho(\tau),\gamma(\cdot),\mu(\cdot))$-Strong CRLCC of blocklength n over the binary alphabet with rate $r(\tau)$.

Moreover, $\lim_{\tau\to 0} r(\tau) = \lim_{\tau\to 0}\rho(\tau) = 1$. In particular, if $\mathsf{L}(\lambda) = \mathsf{polylog}\,\lambda$ and $\lambda = \Theta(n)$, then the code is a $(\mathsf{polylog}\, n,\tau,\rho(\tau),\gamma(\cdot),\mu(\cdot))$-Strong CRLCC.

Our constructions are systematic and imply the existence of a Strong CRLDC with the same parameters.

Remark 2.6: We remark that any Weak CRLCC can be trivially converted into a Strong CRLCC e.g., by repeating the last bit of the codeword. Such trivial constructions either have low rate or are unable to recover a large fraction of original codeword bits. By sufficient replication, we can ensure that a majority of these repeated bits in the codeword are not corrupted and can be corrected with a simply majority vote. Though for such codes, the rate falls below $\frac{1}{3}$ if we need to recover more than $\frac{2}{3}$ of the original codeword symbols. By contrast, our Strong CRLCC has rate approaching 1. Furthermore, we have the property that *almost all bits* of the original codeword can be locally recovered. We also remark that our constructions are systematic and can be tweaked to yield the existence of a Strong CRLDC with the same parameters.

A. Technical Ingredients

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

1) 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 $|A| \leq |B|$ we say that the pair (A, B) contains a δ-expander if for all $X \subseteq A$ and $Y \subseteq B$ with $|X| > \delta|B|$ and $|Y| > \delta|B|$ there is an edge connecting X and Y *i.e.*, $(X \times Y) \cap E \neq \emptyset$. We say that a directed acyclic graph (DAG) $G = (V, E)$, with vertex set $V(G) = \{1, \ldots, n\}$, is a δ-local expander around a node v if for *any* radius $r > 0$ and *any* node $v \geq 2r$ the pair $A = \{v - 2r + 1, \ldots, v - r\}$ and $B = \{v - r + 1, \ldots, v\}$ contain a δ-expander *and* the pair $C = \{v, \ldots, v + r - 1\}$ and $D = \{v + r, \ldots, v + 2r - 1\}$ contain a δ-expander. When this property holds for *every* node $v \in V(G) = \{1, \ldots, n\}$ we simply say that the DAG G is a δ-local expander. For any constant $\delta > 0$ it is possible to (explicitly) 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 has more than $\mathcal{O}(\log n)$ incoming or outgoing edges [4], [27].

Local expanders have several nice properties that have been recently exploited in the design and analysis of secure data-independent memory hard functions (iMHFs) [2]–[4], [17]. For example, these graphs are maximally depth-robust [4]. 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 [4]. 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 $S \cap [v, v + r - 1]$. For more formal statements we point the reader to Section IV. In the context of memory hard functions each node in the graph corresponds to an intermediate data value generated during computation of the iMHFs and edges represent data-dependencies. It is known that an iMHF has high cumulative memory complexity [7] if and only if the underlying DAG G is depth-robust [1], [3].[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 $2\alpha_1$. Setting $\alpha_2 = \alpha$, if the tester determines that a node v is *at least* α_2-good for an appropriately small constant α_2 then we can be (almost) certain that the long green path which contains *all* $n - (1 + \epsilon)|S|$ of the α_2-good nodes also includes v. Furthermore, if $v < 3n/4$ then v has at least $n/4 - (1 + \epsilon)|S|$ descendants in this directed green path. Intuitively, in our construction any such node v must correspond to an *uncorrupted* portion of the codeword.

2) 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 [4], [7], [9]), we stress that we do not need to work in the random oracle model.[2] Indeed, our constructions only assume the existence of collision resistant hash functions along with a public seed s that is known to the encoder/decoder.

B. Technical Overview

Our construction of Strong CRLCC proceeds in two steps. First, we construct a $(\mathsf{polylog}\, n, \tau, \rho, \gamma, \mu(\cdot))$-Weak CRLCC (Theorem 5.9). Then we use a degree reduction technique to obtain a Strong CRLCC from it.

1) 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 $x = (x_1 \circ \ldots \circ x_k)$ (broken up into bit strings of length $\mathsf{L}(\lambda)$) and a k node local expander graph G, the label of node v is computed as $\ell_{v,s} = H(s, x_v \circ \ell_{v_1,s} \circ \ldots \circ \ell_{v_d,s}) \in \{0,1\}^{\mathsf{L}(\lambda)}$, 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 $\mathsf{L}(\lambda) \in \mathcal{O}(\mathsf{polylog}\, \lambda)$ we will select $\lambda \in \mathcal{O}(n)$ to ensure that $\mathsf{L}(\lambda) \in \mathcal{O}(\mathsf{polylog}\, n)$. We use the notation Enc and Dec for the encoding and decoding algorithms of our CRLCC construction, while we use EncJ and DecJ to denote the efficient encoding and decoding algorithms for a good binary code with constant rate and relative distance (e.g., [34], [51]) which can decode efficiently from some constant fractions of errors.

We first apply EncJ to $x_1, \ldots, x_k$ to obtain codewords $c_1, \ldots, c_k \in \{0,1\}^{\mathcal{O}(\mathsf{L}(\lambda))}$ where $c_i = \mathsf{EncJ}(x_i)$. Also, for $v \in [k]$ we let $c_{v+k} = \mathsf{EncJ}(\ell_{v,s})$ which is the encoding of the label corresponding to the node v in G. The final output is $c = (c_1 \circ \ldots \circ c_{2k-1} \circ c_{2k} \circ c_{2k+1} \circ \ldots \circ c_{3k})$ 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 $n = \mathcal{O}(k\mathsf{L}(\lambda))$. 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 $x' = (x'_1 \circ \ldots \circ x'_k)$ with $x'_i = \mathsf{DecJ}(c'_i)$ (possibly $\perp$) and we let $\ell'_{v,s} = \mathsf{DecJ}(c'_{v+k})$ for $v \in [k]$ and $\ell'_{k,s,j} = \mathsf{DecJ}(c'_{2k+j})$ 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

[1]Oversimplifying a bit, if an attacker attempts to reduce memory usage during computation then the attacker's running time will increase dramatically since, by depth-robustness, there will be a long chain of dependent data-values that that the attacker needs to recompute in the near future.

[2]The random oracle model is a source of some controversy among cryptographers [29], [37], [38], [45] with some arguing that the framework can be used to develop cryptographic protocols that are *efficient and secure* [15], while others argue that the methodology is flawed e.g. [13], [38].

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 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.

2) Strong CRLCCs : The reason that the previous construction fails to yield a high rate Strong CRLCCs is that it is possible for an attacker to *change* every node to a red-node by tampering with at most $\mathcal{O}(k \cdot \mathsf{L}(\lambda)/\log k)$ bits. In particular, since the δ-local expander G has outdegree $\mathcal{O}(\log k)$ an attacker who tampers with just $\mathcal{O}(\mathsf{L}(\lambda))$ 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 $\mathcal{O}(\log k) = \mathcal{O}(\log n)$ 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 on t nodes, where $t = \mathcal{O}\left(\frac{k}{\mathsf{L}(\lambda)\cdot\log k}\right)$, 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 $m = \mathcal{O}(\log t)$ nodes — we will refer to node $u \in [t]$ as the meta-node for this group. The result is a new graph G with $k = m \cdot t$ nodes. Each of the nodes $u_1, \ldots, u_{m-1}$ will have constant indegree and constant outdegree. 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 $m \in \mathcal{O}(\log t)$ before applying the error correcting code. In particular, given our input word x divided into $k = m \cdot t$ distinct $\mathsf{L}(\lambda)$-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 $c_i = \mathsf{EncJ}(x_{(i-1)m+1}, \ldots, x_{im})$ for $i \leq t$ and $c_{v+t} = \mathsf{EncJ}(\ell_{v_1,s} \circ \ldots \circ \ell_{v_m,s})$ and $c_{2t+1} = \ldots = c_{3t} = c_{2t}$ consists of t copies of c_{2t}. The final codeword c will have length $n \in \mathcal{O}(tm \cdot \mathsf{L}(\lambda)) = \mathcal{O}(k \cdot \mathsf{L}(\lambda))$ 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* $\Omega(\mathsf{L}(\lambda) \cdot m)$ 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 say that the *meta-node* $u \in [t] = V(G_t)$ containing nodes $u_1, \ldots, u_m \in V(G)$ is green if (1) node u_m is green, and (2) at least $2m/3$ of the nodes $u_1, \ldots, u_m$ are green. We say that an edge $(u, v) \in E(G_t)$ in the meta-graph is red if the corresponding edge $(u_m, v_j) \in E(G)$ is incident to a red node; otherwise we say that the edge (u, v) is green. We can then define the green meta-graph G_g by dropping all red edges from G_t. We remark that, 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-good with respect to the set R of red meta-nodes. In particular, there are only two ways to turn a meta-node u red: (1) by paying to flip $\mathcal{O}(\mathsf{L}(\lambda) \cdot m)$ bits in c_u or c_{u+t}, or (2) by corrupting at least $m/3$ other meta-nodes w such that the directed edge (w, u) is in G_t.

Furthermore, we can introduce the (unknown) set T of tampered meta-nodes where for $i \leq 3t$ we say that $\tilde{c}_i$ is tampered if $\mathsf{EncJ}(\mathsf{DecJ}(\tilde{c}_i)) \neq c_i$. In this case the hamming distance between c_i and $\tilde{c}_i$ must be at least $\mathcal{O}(\mathsf{L}(\lambda) \cdot m)$ so the number $|T|$ of tampered meta-nodes cannot be too large. While the set T is potentially unknown we can still show that *most* meta-nodes are α-good with respect to the set $R \cup T$. In particular, there must exist some meta-node $\frac{3t}{4} \leq v \leq t$ s.t. v is α-good with respect to the set $R \cup T$. By collision resistance, it follows that for any meta-node $u < v$ which is connected to v via a green path in G_g we have $u \notin T$ i.e., u is correct; otherwise we would have found a hash collision.

While G_t is a δ-local expander the DAG G_g may not be. However, we can show that, for suitable constants α and δ, if a node u is α-good with respect to the set $R \cup T$ then G_g has 2δ-local expansion around u. Furthermore, if $u < v$ has 4δ-local expansion in G_g then we can prove that G_g contains a path to the meta-node u. Our local decoder will test whether or not G_g has local expansion by strategically sampling edges to see if they are green or red. Our decoder will output $\perp$ with high probability if u does *not* have 4δ-local expansion in G_g which is the desired behavior since we are not sure if there is a green path connecting u to v in this case. Whenever u does have 2δ-local expansion in G_g the decoder will output $\perp$ with negligible probability which is again the desired behavior since there is a green path from u to v in this case (and, hence, $u \notin T$ has not been tampered).

Remark 2.7: Recall that in Theorem 2.5 we construct a $(\mathsf{polylog}\, n, \tau, \rho(\tau), \gamma(\lambda), \mu(\lambda))$-Strong CRLCC where $\mu(\lambda)$ represents the probability that the adversarial channel outputs a corrupted codeword $\tilde{c}$ and challenge index $i \in [k]$ that $\gamma(\lambda)$-fools the local decoder. Intuitively, we can think of $\mu(\lambda)$ as the probability that the corrupted word generated by the channel yields a hash collision and we can think of $\gamma(\lambda)$ as the probability that we mis-identify a node (or meta-node) as being α-good. If there are no hash collisions then the *only* way that the local decoding algorithm can be fooled is when it mis-identifies a node (or meta-node) as being α-good. We show that *both* $\mu(\cdot)$ and $\gamma(\cdot)$ are negligible functions.

III. Related Work

A. 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 $\mathcal{O}\left(2^{\sqrt{\log n \log\log n}}\right)$ [39]. In the constant query complexity regime, for $q \geq 3$, the best codes achieve blocklength subexponential in the message length, k [24], [26], [55]. For $q = 2$, Kerenidis and deWolf [35] show a lower bound exponential in k for the blocklength of any LDC. Subsequent works of [8], [19] improve the information rate and query complexity tradeoffs. In particular, [19] construct RLCCs with $O(q)$ query complexity and almost linear blocklength, $n = k^{1+\alpha}$, for an arbitrarily small constant $\alpha > 0$. This closes the gap to the lower bound of super-linear blocklength [43] for any constant query RLCC.

A notion similar to RLDCs, called Locally Decode/Reject code, was studied by Moshkovitz and Raz [48] in the context of building better PCPs. The codes allow decoding batches of coordinates jointly, and it allows as output a list of possible messages. In the effort of simplifying the proofs in [48], a related notion of decodable PCPs was studied in [25].

B. Non-Local Codes in Computationally Bounded Channels

In their initial works in the computationally bounded channel, Lipton [44] and Ding *et al.* [23] 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 strong requirement makes the model unsuitable in many realistic settings where the sender and receiver do not share a secret key e.g., message broadcast. Micali *et al.* [47] 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*, by 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 [31] remove the public-key setup, and obtain 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 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.

C. LDCs in Computationally Bounded Channels

The notion of LDCs over computationally bounded channels was introduced in [49], where the authors define private LDCs. 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 [32] build LDCs in the public-key model, and obtain codes of constant rate, and $\mathcal{O}(\lambda^2)$ locality. Hemenway *et al.* [33] improve on these results and obtain public-key LDCs with constant rate, constant error rate and $\mathcal{O}(\lambda)$ locality, which work under a weaker cryptographic assumption than that of [32].

D. Other Applications of Depth-Robust Graphs

Depth-robust

graphs have found many applications in cryptography. These applications include the construction of memory hard functions with *provably high* cumulative memory complexity [1]–[3], [7], [17], sustained space complexity [4] and bandwidth hardness [16], [50]. Depth robust graphs have also been used to construct *proofs of space* [22] and they were also used in the first construction of a *proof of sequential work* [46] although Cohen and Pietrzak [21] found an alternate construction of a protocol for proofs of sequential work that does not involve depth-robust graphs. Finally, depth-robust graphs have also been used to derive cumulative space in the black-white pebble game [4], [5] which is of interest in the study of proof complexity. To the best of our knowledge we are the first to apply depth-robust graphs (more specifically δ-local expanders) in the area of coding theory to construct (relaxed) locally correctable codes.

IV. Preliminaries: Justesen Codes, CRHFs and Local Expanders

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 $G = (V = [n], E)$. We will assume that $u < v$ for all directed edges $(u, v) \in E$ so that $1, \ldots, n$ is a valid topological ordering. 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 EncJ_R and DecJ_R to refer to the encoding and decoding algorithms for a binary code $C_{J,R}$ which will be used as subroutines in our $\mathsf{CRLCC/CRLDC}$ constructions. When the rate R is clear from context we will drop the subscript and simply write EncJ and DecJ. We instantiate EncJ_R and DecJ_R with Justesen codes [34] though we remark that any binary code with constant rate and relative distance would work.

Theorem 4.1: [34] 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 depends on R.

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 [36].

Definition 4.2: [36, Definition 4.11] A *hash function* with alphabet Σ and blocklength $\mathsf{L}(\cdot)$ 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 λ and outputs a public seed $s \in \{0,1\}^*$, where the security parameter 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 $H(s, x) \in \Sigma^{\mathsf{L}(\lambda)}$.

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: [36, 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-bipartite 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 $e \in M_1 \times M_2$.

The notion of a δ-bipartite expander we use is a bit different from the standard notions of vertex expansion e.g., we only require that $M_1 \subseteq U$ has a large neighborhood if $|M_1| \geq \delta|U|$ itself is sufficiently large. However, the notion of a δ-bipartite expander is commonly used to construct and analyze depth-robust graphs [2], [4], [27].

Definition 4.5: We say that a DAG $G = ([n], E)$ has δ-local expansion *around a node* $x \in [n]$ if for all $r \leq \min\{\frac{x}{2}, \frac{n-x+1}{2}\}$

(1) the bipartite graph induced by $U = [x, \ldots, x + r - 1]$ and $V = [x+r, \ldots, x+2r-1]$ is a δ-bipartite expander.
(2) the bipartite graph induced by $A = [x - r + 1, \ldots, x]$ and $B = [x-2r+1, \ldots, x-r]$ is a δ-bipartite expander.

If this property holds for *every* node $x \in [n]$ we simply say that G is a δ-local expander.

Theorem 4.6, due to Alwen *et al.* [4, Lemma 4], states that δ-local expanders exist with indegree $\mathcal{O}(\log n)$. The construction of Alwen *et al.* [4] 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.* [4] closely follows an earlier construction of Erdos *et al.* [27].

While Alwen *et al.* [4] only analyze the indegree of this construction it is easy to verify that the outdegree is also $\mathcal{O}(\log n)$ since the construction of Erdos *et al.* [27] 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 $\mathcal{O}(\log n)$ bipartite expanders.

Theorem 4.6 [4]: For any $n > 0, \delta > 0$, there exists a δ-local expander $G = ([n], E)$ with indegree $\mathcal{O}(\log n)$ and outdegree $\mathcal{O}(\log n)$.

We now list some properties of the δ-local expander graphs shown in [4]. 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 $0 < \alpha < 1$. We say $v \in [n] - S$ is α-good under S if for all $1 \leq u < v$ and $n \geq w > v$ we have:

$$|[u, v] \cap S| \leq \alpha(v - u + 1), \qquad |[v, w] \cap S| \leq \alpha(w - v - 1).$$

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 [4] 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.[3]

Lemma 4.8 (Lemma 6 in [4]): 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.

Claim 4.9 rephrases a claim from [4, Claim 2]. Technically, [4, Claim 2] assumes that the entire graph G is a δ-local expander, but the proof only uses local expansion around x.

Claim 4.9 (Claim 2 in [4]): Let $G = (V = [n], E)$ be a δ-local expander around x and suppose that $x \in [n]$ is an α-good node under $S \subseteq [n]$. Let $r > 0$ be given. If $2\delta < (1-\alpha)$ then all but $2\delta r$ of the nodes in $[x, x+r-1] \setminus S$ are reachable from x in $G - S$. Similarly, x can be reached from all but $2\delta r$ of the nodes in $[x-r+1, x] \setminus S$. In particular, if $\delta < 1/4$ then more than half of the nodes in $[x, x+r-1] \setminus S$ (resp. in $[x-r+1, x] \setminus S$) are reachable from x (resp. x is reachable from) in $G - S$.

Alwen *et al.* [4] also show that in a δ-local expander graph G any two α-good vertices in $G - S$ are connected. Technically, the statement of [4, Lemma 5] assumes that G is a δ-local expander, but their proof only requires that G is a δ-local expander around nodes u and v.

Lemma 4.10 (Lemma 5 in [4]): Let $G = ([n], E)$ be a δ-local expander around nodes u and v, $S \subseteq [n]$ and $0 \leq \alpha \leq 1$. If $\delta < \min((1-\alpha)/2, 1/4)$, and nodes u and v are both α-good with respect to S then u and v are connected in $G - S$.

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

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

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.

We remark that similar labeling rules have been used in the construction of memory hard functions and proofs of space e.g., see [7], [22]. We stress two key differences. First, in our setting there is a message x that must be integrated into the labeling function. Second, we only assume that H is collision-resistant while formal security proofs for memory hard functions and proofs of space utilize the random oracle model.

[3] In [4] a node $u \notin S$ is called γ-good if for all $u < v$ (resp. $w > v$) at least γ fraction of the nodes in $[u, v]$ (resp. $[v, w]$) are *not* deleted. In our notation such a node would be α-good with $\alpha = 1 - \gamma$.

A. 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 $H : \{0,1\}^* \to \{0,1\}^{\mathsf{L}(\lambda)}$ be a CRHF (see Definition 4.3). We will assume that Enc has access to H. In this section we also fix the rate $R = 1/4$ in Theorem 4.1 to obtain an efficient binary code $\mathsf{EncJ} : \{0,1\}^{\mathsf{L}(\lambda)} \to \{0,1\}^{4\cdot\mathsf{L}(\lambda)}$ with message length $\mathsf{L}(\lambda)$ and block length $4 \cdot \mathsf{L}(\lambda)$ along with the corresponding decoding algorithm $\mathsf{DecJ} : \{0,1\}^{4\cdot\mathsf{L}(\lambda)} \to \{0,1\}^{\mathsf{L}(\lambda)}$ that efficiently decodes from $\Delta_J = \Delta_J(0.25)$ fraction of errors.

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

- Let $\mathsf{lab}_{G,s}(x) = (\ell_{1,s}, \ldots, \ell_{k',s})$ be the labeling of the graph G with input x and seed s.
- The codeword c consists of the block encoding of message x, followed by the encoded labeling of the graph using EncJ, followed by k' copies of encoded $\ell_{k'}$. In particular, for each $1 \leq i \leq k'$ we have $c_i = \mathsf{EncJ}(x_i)$, $c_{i+k'} = \mathsf{EncJ}(\ell_{i,s})$ and $c_{i+2k'} = \mathsf{EncJ}(\ell_{k',s}) = c_{2k'}$.

The parameter $\delta = 1/100$ is arbitrary and is simply chosen to satisfy Lemma 4.10. Also, note that the codeword length is $n = 4 \cdot 3k' \cdot \mathsf{L}(\lambda) = 12k$ over the binary alphabet $\{0,1\}$, where $\mathsf{L}(\lambda)$ is the length of the output of the CRHF H and the original message had length $k = k' \cdot \mathsf{L}(\lambda)$. Therefore, the information rate $\frac{k}{n}$ of CRLCC scheme obtained from Enc is $\Theta(1)$.

B. 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 V-A. 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.

Definition 5.2 (Green/red node): Given a (possibly corrupted) codeword $w = (w_1 \circ \ldots \circ w_{3k'}) \in (\{0,1\}^{4\cdot\mathsf{L}(\lambda)})^{3k'}$ we define $x'_i = \mathsf{DecJ}(w_i) \in \{0,1\}^{\mathsf{L}(\lambda)}$ for $i \leq k'$ and

$\ell'_{v,s} = \mathsf{DecJ}(w_{k'+v})$ for $v \leq k'$ and $\ell'_{k',s,i} = \mathsf{DecJ}(w_{2k'+i})$ for $i \leq k'$. 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., $\ell'_{v,s} = H(s, x'_v \circ \ell'_{v_1,s} \circ \ell'_{v_2,s} \circ \ldots \circ \ell'_{v_d,s})$. A node that is not green is a *red* node.

We say that a node v is *correct* if $\ell'_{v,s} = \ell_{v,s}$.

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 $\ell'_{v,s} = H(s, x'_v \circ \ell'_{v_1,s} \circ \ldots \circ \ell'_{v_d,s})$ where $v_1, \ldots, v_d$ are the parents of node v. Moreover, if $\ell'_{v,s}$ is correct, then $\ell'_{v,s} = \ell_{v,s}$, *i.e.*,

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

So, either the adversary has successfully found a hash collision, or $x_v = x'_v$ and $\ell_{v_j,s} = \ell'_{v_j,s}$ for each $j \in [d]$. Let v_j be the parent node of v that lies on the green path from u to v. Assuming we have no hash collision, we can conclude that the node v_j is *both* correct and green. Extending the same argument iteratively along the green path from u to v_j, we get that either node u must be correct, or the adversary would have found a hash collision. □

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 depending on what index $i \in [n]$ of the received word w we are asked to decode. Recall that the codeword is $n = 12k$ bits long.

Case 1: The input index is $i > 8k - 4\mathsf{L}(\lambda)$. This corresponds to a bit query within the last $k' + 1$ blocks. From the construction of the code, we know that the last $k' + 1$ 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 some random blocks $w_{j_1}, w_{j_2}, \ldots$ with $j_1, j_2, \ldots \in [2k', 3k']$, and computes $y_{j_s} = \mathsf{DecJ}(w_{j_s})$ and $w''_{j_s} = \mathsf{EncJ}(y_{j_s})$. Since the adversary cannot corrupt too many blocks beyond the decoding radius we expect that most of the recovered blocks are correct i.e., $w''_{j_s} = c_{j_s} = c_{2k'}$. Thus, a simple majority vote will return the correct codeword $c_{2k'}$ with high probability and return the corresponding bit $i' = 1 + (i \mod 4\mathsf{L}(\lambda))$. See Section V-B.1 for the formal proof.

Case 2: If the input index is $i \leq 8k - 4\mathsf{L}(\lambda)$, we let i' denote the index of the node in G associated with the queried bit i *i.e.*, $i' = 1 + \left(\left\lfloor \frac{i}{4 \cdot \mathsf{L}(\lambda)} \right\rfloor \mod k'\right)$. We then check to make sure that (1) both nodes i' and k' are α-good w.r.t. the (unknown) set S of red nodes induced by the corrupted codeword c', and (2) node k' is correct. If either check fails then we will output $\perp$. Assuming that both nodes i' and k' are both α-good and k' is correct (and excluding negligible outcomes where a hash collision is produced) Lemma 5.3 implies that the node i' must be correct since there is a green path connecting any two α-good nodes in a δ-local expander.

Testing that node k' is correct follows case 1 i.e., sample + majority vote. The core part of the Dec^w 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 using $\mathcal{O}(\log n)$ 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 IsGood that *accepts* v (whp) if it is $\alpha/4$-good with respect to the set S of red nodes and rejects (whp) if v is not α-good. The procedure IsGood is 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 (with replacement) in each of the intervals $[v - 2^j + 1, v]$ and $[v, v + 2^j - 1]$ and use the subroutine 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 in both intervals is *at most* $\alpha \cdot 2^j/2$ we accept; otherwise, we reject. See Section V-B.2 for a formal proof.

1) Query Procedure for $i > 8k - 4\mathsf{L}(\lambda)$: In this section, we describe the procedure for recovering a coordinate for input $i > 8k - 4\mathsf{L}(\lambda)$. 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 outputting $\perp$ or the correct bit for a query on the last $k' + 1$ blocks.

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

Dec_1^w: Input: Index $i \in [n]$ such that $i > 8k - 4\mathsf{L}(\lambda)$.

(1) Sample $\Theta\left(\frac{\log^3 k}{\mathsf{L}(\lambda)}\right)$ blocks $\{w_t\}$ of w for $t \in [2k', 3k']$ uniformly at random.
(2) Decode each of the queried blocks w_t to the corrected codeword, $c_t := \mathsf{EncJ}(\mathsf{DecJ}(w_t))$. (could possibly be a $\perp$ if DecJ fails to decode).
(3) Let $c_{maj} = \text{majority}\{c_t\}$ be the codeword of EncJ which occurs majority of the times in Step (2) above.
(4) Output symbol $1 + (i \mod 4\mathsf{L}(\lambda))$ 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 V-A. For any $8k - 4\mathsf{L}(\lambda) < i \leq n$, 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.

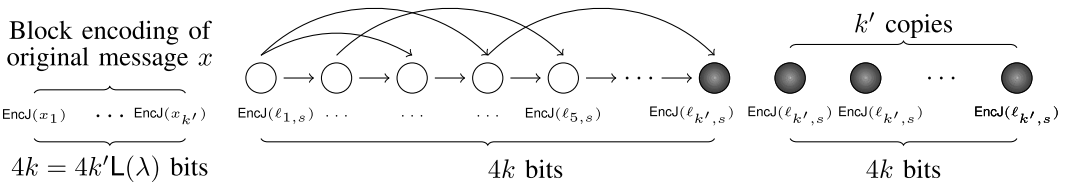

Fig. 1. An example of an encoding with an underlying graph with $k' = k/\mathsf{L}(\lambda)$ nodes.

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 $k' + 1$ 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 $k' + 1$ 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 $\Theta\left(\frac{\log^3 k}{\mathsf{L}(\lambda)}\right)$ queries are correct is at least $1 - e^{-\Omega(n_s)}$, where $n_s = \Theta\left(\frac{\log^3 k}{\mathsf{L}(\lambda)}\right)$ is the number of sampled blocks.

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

2) Query Procedure for $i \leq 8k - 4\mathsf{L}(\lambda)$: In this section, we describe the algorithm for recovering a coordinate for input $i \leq 8k - 4\mathsf{L}(\lambda)$. Before we describe the algorithm, we need a few definitions and properties of the code we constructed in Section V-A.

Recall that for a codeword obtained from Enc, the first k' blocks of length $4\mathsf{L}(\lambda)$ 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.

Recall that we say that a node v is *green* if it is locally consistent with the hash of its parents. 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 $w \in \{0,1\}^n$. From the definition of a green node it follows 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 $w = (w_1, \ldots, w_{3k'}) \in \{0,1\}^n$ be the corrupted codeword obtained from the PPT adversary. There exists an algorithm $\mathsf{IsGreen}$ that uses $\mathcal{O}(\mathsf{L}(\lambda)\log(n/\mathsf{L}(\lambda)))$ 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 all $v_j \in \mathsf{parents}_G(v)$. Let $\ell'_{k'+v_j} := \mathsf{DecJ}(w_{k'+v_j})$. Return 'Red' if DecJ fails on any input.
(2) Query and decode the block w_v. Let $x'_v := \mathsf{DecJ}(w_v)$. Return 'Red' if DecJ fails.
(3) If $\mathsf{DecJ}(w_{k'+v}) = H(s, x'_v \circ w_{k'+v_1} \circ w_{k'+v_2} \circ \cdots \circ w_{k'+v_d})$, 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.* $\mathsf{DecJ}(w_{k'+v})$, is green by querying at most $\mathcal{O}(\log k')$ blocks of w. Since each block is $4\mathsf{L}(\lambda)$ bits long, and $k' = k/\mathsf{L}(\lambda) = \mathcal{O}(n/\mathsf{L}(\lambda))$, the number of bits queried is equivalently $\mathcal{O}(\mathsf{L}(\lambda)\log(n/\mathsf{L}(\lambda)))$. □

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: Let $\epsilon > 0$. For any $\alpha < \frac{3}{4}$, there exists a procedure that makes $\mathcal{O}\left(\mathsf{L}(\lambda)\log^{3+\epsilon}\left(\frac{n}{\mathsf{L}(\lambda)}\right)\right)$ 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 $U_{2p} \subseteq [i, i + 2^p - 1]$ with replacement.
  - If the fraction of red nodes in U_{1p} or U_{2p} is larger than $\frac{3\alpha}{8}$, 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 $\mathcal{O}\left(\log k'\right)$ block queries to w. Therefore the number of coordinates of w queried by IsGood is at most $\mathcal{O}\left(\mathsf{L}(\lambda)\log^{3+\epsilon}\left(n/\mathsf{L}(\lambda)\right)\right)$.

To prove the correctness, first observe that if the node i is red, then $i \in S$ and therefore by definition it is not α-good for any α. IsGood always rejects such nodes.

Suppose node i is $\alpha/4$-good. Then 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 $\alpha/4$ fraction of red nodes. So, on sampling uniformly random $\log^{1+\epsilon} k'$ vertices from these intervals, the probability that we see more than $3\alpha/8$ fraction of red nodes is at most $e^{-\frac{\alpha}{12}\log^{1+\epsilon} k'}$ which follows from Chernoff bounds. Taking a union bound over all $\log k'$ intervals, IsGood accepts any $\alpha/4$-good node with probability at least $1 - e^{-\frac{\alpha}{24}\log^{1+\epsilon}(k')}$. Therefore IsGood accepts any $\alpha/4$-good node with probability at least $1 - \mathsf{negl}(k') = 1 - \mathsf{negl}(n)$ for $\mathsf{L}(\lambda) = \mathsf{polylog}\, n$.

In order to show that the algorithm rejects any node which is not α-good with high probability, we first show that any node which is not α-good under S is not $\alpha/2$-good for some interval of size 2^p, $p > 0$. The rejection probability then follows from Chernoff bounds. 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

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

where the inequality (*) results from i not being α-good. So, the probability that this interval goes undetected by IsGood is at most $e^{-\frac{\alpha}{32}\log^{1+\epsilon} k'}$. Therefore, IsGood rejects any node which is not α-good with probability at least $1 - \mathsf{negl}(k')$. □

Now, we describe the decoding procedure for any index $i \leq 8k - 4\mathsf{L}(\lambda)$ using the IsGood procedure. 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 from the encoding procedure in Section V-A, the first $8k - 4\mathsf{L}(\lambda)$ bits correspond to either the encoding of the message bits using EncJ (the first $4k$ bits) or the encoding of the labels of the nodes the δ-local expander (the next $4k$ bits). For any index query corresponding to the encoding of the label bits ($i > 4k$), we check if the label is tampered with or not. If it is not tampered, then the decoder returns the received bit, else it returns a $\perp$. For any $i \leq 4k$, the decoder first tests if its corresponding label is tampered. If it is not tampered, then the decoder returns the corresponding received bit, else it returns a $\perp$.

Recall that using Algorithm Dec_1 we can successfully correct any index $i > 8k - 4\mathsf{L}(\lambda)$. 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 4.10 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 correct. We now formalize this and describe Dec_2 in Lemma 5.7.

Lemma 5.7: Let Enc be as described in Section V-A. For any $1 \leq i \leq 8k - 4\mathsf{L}(\lambda)$, 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 $\mathcal{O}\left(\mathsf{L}(\lambda)\log^{3+\epsilon}\left(n/\mathsf{L}(\lambda)\right)\right)$ queries to input w for any constant $\epsilon > 0$.

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

Dec_2^w: Input: Index $i \leq 8k - 4\mathsf{L}(\lambda)$.
(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 $1 + \left(\lfloor \frac{i}{4 \cdot \mathsf{L}(\lambda)} \rfloor \mod k'\right)$ 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 $\mathcal{O}\left(\mathsf{L}(\lambda)\log^{3+\epsilon}\left(n/\mathsf{L}(\lambda)\right)\right)$. 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 $\mathcal{O}\left(\mathsf{L}(\lambda)\log^{3+\epsilon}\left(n/\mathsf{L}(\lambda)\right)\right)$ queries for any $\epsilon > 0$. Therefore the query complexity of Dec_2 is $\mathcal{O}\left(\mathsf{L}(\lambda)\log^{3+\epsilon}\left(n/\mathsf{L}(\lambda)\right)\right)$.

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 4.10, 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 $i > 8k - 4\mathsf{L}(\lambda)$, 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 V-A. 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 $\mathsf{dist}(c, w) \leq \frac{\Delta_{JK}}{4}$,

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

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

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

Theorem 5.9: Assuming the existence of a collision-resistant hash function (GenH, H) with length $\mathsf{L}(\lambda)$, there exist a constant $0 < \tau' < 1$ and negligible functions $\mu(\cdot), \gamma(\cdot)$ such that the following holds: for all $\tau \leq \tau'$, there exists a constant $0 < \rho(\tau) < 1$ and a $(\mathsf{L}(\lambda) \cdot \mathsf{polylog}\, n, \tau, \rho(\tau), \gamma(\cdot), \mu(\cdot))$-Weak CRLCC of blocklength n over the binary alphabet.

In particular, if $\mathsf{L}(\lambda) = \mathsf{polylog}\, \lambda$ and $\lambda = \Theta(n)$, then the code is a $(\mathsf{polylog}\, n, \tau, \rho(\tau), \gamma(\cdot), \mu(\cdot))$-Weak CRLCC.

Proof: 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 V-A and the decoder Dec described in Section V-B 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 $n = 3k' \cdot 4 \cdot \mathsf{L}(\lambda) = 12k$. 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 $\mathsf{dist}(\mathsf{Enc}(x), w) \leq \frac{\Delta_{JK}}{4}$, Dec^w queries at most $\mathcal{O}\left(\mathsf{L}(\lambda)\log^{3+\epsilon}(n/\mathsf{L}(\lambda))\right)$ coordinates of w and returns $b \in \{\mathsf{Enc}(x)[i], \perp\}$ 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.

We refer the reader to Appendix VIII-A and Appendix IX-A for the list of parameters and the proof flowchart for the construction of Weak-CRLCC. □

VI. 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 just $\mathcal{O}\left(\frac{n}{\log n}\right)$ symbols can turn all nodes red in a graph that has outdegree $\mathcal{O}(\log n)$. In particular, for any node $v \in [k']$ an adversary can corrupt the corresponding block c_v with just $\mathcal{O}(\mathsf{L}(\lambda))$ corruptions. In addition to node v any node w with a directed edge $(v, w) \in E$ will also be red. For a weak-CRLCC it is not a problem if all nodes are red as the decoder is allowed to output $\perp$, but this is not allowed for a strong-CRLCC decoder.

Key Idea: We overcome the above mentioned barrier using three ideas. First, we run the original δ-local expander G_t with t nodes through a degreereduction gadget to obtain a new graph G with $k = t \cdot m$ nodes in which each node each node u in G_t (called the *meta-node*) corresponds to m nodes $u_1, \ldots, u_m$ in G. Furthermore, each of the nodes $u_1, \ldots, u_{m-1}$ will have at most 2 incoming edges and at most 2 outgoing edges. The final node u_m has $\mathsf{indeg}(u_m) \in \Theta(m)$, and $\mathsf{outdeg}(u_m) \in \Theta(\log t)$. Second, during the encoding process we group the values associated with each meta-node so that the attacker who corrupts any of these nodes must pay to corrupt the entire meta-node. Third, we define and analyze the green subgraph G_g of the meta-graph G_t which is obtained by discarding red edges. While the graph G_g is not given explicitly we can locally test whether or not specific nodes/edges are red. To analyze the green subgraph we call a meta-node u green if u_m is green and at least $2m/3$ of the corresponding nodes $u_1, \ldots, u_m$ are green. This ensures that most meta-nodes are green. We elaborate on these ideas below:

We group the labels $\ell_{u,s} = (\ell_{u_1,s}, \ldots, \ell_{u_m,s})$ and data values $x_u = (x_{u_1}, \ldots, x_{u_m})$ and include the encodings $c_u = \mathsf{EncJ}(x_u)$ and $c_{u+t} = \mathsf{EncJ}(\ell_{u,s})$ in our final codeword. Notice that if we receive a codeword $c' \neq c$ we can first preprocess each block c'_i to obtain $\tilde{c}_i = \mathsf{EncJ}(\mathsf{DecJ}(c'_i))$. Thus, if $\mathsf{EncJ}(\mathsf{DecJ}(c'_i)) = c_i$ we will say that the block i is correct (untampered) — even if $c'_i \neq c_i$. We say that a meta-node u is tampered if either of the blocks c_u or c_{u+t} were tampered and we use T to denote the set of tampered meta-nodes (Note: the set T is not given to the decoder, but is useful to define T for analysis). We can show that most meta-nodes are not contained in the set T since the attacker must alter a constant fraction of the bits in c_u or c_{u+t}. In particular, the attacker has to flip *at least* $\Omega(m \cdot \mathsf{L}(\lambda))$ bits to tamper *any* of the meta-nodes. In this way we can ensure that an attacker must tamper with *at least* $\Omega(t \cdot m \cdot \mathsf{L}(\lambda))$ bits to make $\Omega(t)$ meta-nodes red.

We introduce the notion of the *green* subgraph G_g of the meta-graph G_t (see Definition 6.6). In particular, G_g

is obtained from G_t by discarding all red-edges (an edge $(u,v) \in E(G_t)$ is red if the corresponding edge $(u_m, v_j) \in E(G)$ in G is incident to a red-node). As before a node v_i in G is green if its hash value is locally consistent e.g., $\ell'_{v_i,s} = H(s, x'_{v_i} \circ \ell'_{v_i^1} \circ \ldots \circ \ell'_{v_i^d})$ where $v_i^1, \ldots, v_i^d$ are the parents of node v_i in G and for each meta-node u in G_t we define $\ell'_u = \left(\ell'_{u_1,s}, \ldots, \ell'_{u_m,s}\right) = \mathsf{DecJ}(c'_{u+t})$ and $x'_u = \left(x'_{u_1}, \ldots, x'_{u_m}\right) = \mathsf{DecJ}(c'_u)$. We say that a meta-node u in G_t is green if at least $\frac{2}{3}$ of the corresponding nodes $\{u_1, \ldots, u_m\}$ in G are green *and* the last node u_m is also green (Definition 6.2). We show that the same key properties we used for the construction of Weak-CRLCC still hold with respect to the meta-nodes i.e., if there is a green path connecting meta-node u to meta-node v and the meta-node v is untampered (i.e., $c_v = \mathsf{EncJ}\left(\mathsf{DecJ}(c'_v)\right)$ and $c_{v+t} = \mathsf{EncJ}\left(\mathsf{DecJ}(c'_{v+t})\right)$) then the meta-node u is also untampered (i.e., $c_u = \mathsf{EncJ}\left(\mathsf{DecJ}(c'_u)\right)$ and $c_{u+t} = \mathsf{EncJ}\left(\mathsf{DecJ}(c'_{u+t})\right)$); otherwise we would have found a hash collision (Lemma 6.14). If we let R denote the subset of red meta-nodes and if we let T denote the subset of (unknown) tampered meta-nodes we have G_g contains the graph $G_t - (R \cup T)$. One can prove that most meta-nodes are α-good with respect to the set $R \cup T$ and that if a meta-node u is α-good with respect to the set $R \cup T$ then the green graph G_g also has 2δ-local expansion around the meta-node u (Lemma 6.10). Finally, if G_g has δ'-local expansion around both of the meta-nodes u and v with $\delta' \leq 4\delta$ then there is a green path connecting u and v (Lemma 6.12).

The remaining challenge is that the set T is unknown to the local decoder.[4] Thus, we cannot directly test if a node is α-good with respect to $R \cup T$. What we can do is develop a test that accepts all meta-nodes that are α-good with respect to $R \cup T$ with high probability and rejects all meta-nodes u that do not have 4δ-local expansion in G_g by sampling the number of red/green edges in between $A_i = [u, u+2^i-1]$ and $B_i = [u+2^i, u+2^{i+1}-1]$ for $i \geq 1$. In particular, we restrict our attention to edges in the δ-expander graph $H_{i,\delta}$ connecting A_i and B_i in G_t, where $H_{i,\delta}$ has maximum indegree d_δ for some constant d_δ which depends on δ. If G_g is not a 4δ-local expander around u then for some interval i a large number of red edges between A_i and B_i *must* have been removed — at least $c_2\delta 2^i$ red edges out of at most $2^i \times d_\delta$ edges in $H_{i,\delta}$. By contrast, if u is α-good with respect to $R \cup T$ then we can show that at most $c_1\alpha 2^i d_\delta$ edges are deleted. For a suitably small value of $\alpha < \frac{\delta}{d_\delta}$ we can ensure that $c_1\alpha 2^i d_\delta < 4\ c_2\delta 2^i$. Thus, our tester can simply sample random edges in $H_{i,\delta}$ and accept if and only the observed fraction $f_{r,i}$ of red edges is at most, say, $2\ c_1\alpha 2^i d_\delta$ (Lemma 6.9).

Graph Degree Reduction: We now describe our procedure ReduceDegree that takes as input a t node δ-local expander DAG G_0 with degree $m = O(\log t)$ and outputs a new graph G with mt nodes. G has the property that *most* nodes have indegree two.

[4]By contrast, it is easy to check if a node (resp. meta-node) is red/green by checking if the hash value(s) are locally consistent.

ReduceDegree : (Let G_0 be a δ-local expander.)
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.

We call G_0 the *meta-graph* of G and we refer to nodes in G_0 as *meta-nodes*. In particular, the meta-node $u \in V(G_0)$ corresponds to the *simple* nodes $u_1, \ldots, u_m \in V(G)$. Notice that for two meta-nodes $u < v \in V(G_0)$ there is an edge $(u,v) \in E(G_0)$ in the meta-graph if and only if G has an edge of the form (u_m, v_i) for some $i \leq m$. While our encoding algorithm will use the graph G to generate the labeling, it will be helpful to reason about meta-nodes in our analysis since the meta-graph G_0 is a δ-local expander.

Note: The degree reduction technique used in our work slightly deviates from that used in [4]. In particular, similar to the gadget in [4], we replace each meta node u with a path graph $u_1 \to u_2 \to \ldots \to u_m$. In our construction, we additionally connect all nodes u_i, $i < m$ in the path to the final node u_m.

A. 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 $H : \{0,1\}^* \to \{0,1\}^{\mathsf{L}(\lambda)}$ be a CRHF (see Definition 4.3). Recall that EncJ and DecJ denote the encoding and decoding algorithms of a good code (for instance the Justesen code) of rate $R = R(\tau)$ that can decode efficiently from some constant $\Delta_J = \Delta_J(\tau)$ fraction of errors (Note: as $\tau \to 0$ we have $\Delta_J(\tau) \to 0$ and $R(\tau) \to 1$).

Let $\beta = \beta(\tau) > 0$ be a parameter which can be tuned. For $t = \mathcal{O}\left(\frac{k}{\beta \mathsf{L}(\lambda) \cdot \log k}\right)$, let $G_0 = ([t], E)$ be a δ-local expander graph on t vertices and degree (indegree and outdegree) $m = \mathcal{O}(\log t)$ where $\delta = \delta(\tau) < \frac{1}{4}$ is a parameter than can be tuned as needed.[5] Let $G := \mathsf{ReduceDegree}(G_0)$ be the graph with $k' := \frac{k}{\beta \mathsf{L}(\lambda)}$ nodes output by the degree reducing procedure applied to G_0. From ReduceDegree it then follows that $k' = mt = \mathcal{O}(t \log t)$. Crucially, the graph has the property that for $\alpha = \alpha(\tau) < 1 - 2\delta(\tau)$ and any red/green coloring of the nodes of the graph, any pair of α-good meta-nodes with respect to the set T of tampered nodes are connected by a *green* path.

[5]As δ decreases the degree of G_0 increases. In particular, we have $\mathsf{indeg}(G_0) = d_\delta \log n$ where d_δ is some constant which depends on δ — as δ decreases d_δ increases. We will also have $m = d_\delta \log n$ nodes in each metanode of G so m increases as δ decreases. This will increase the *locality* of our decoding algorithm, but only by a constant factor.

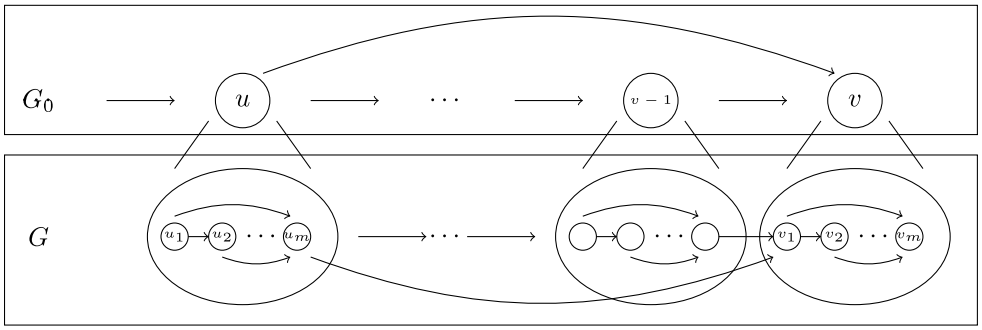

Fig. 2. An example of a degree reduction graph.

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

$\mathsf{Enc}\ (s, x)$:

Input: $x = (x_1 \circ \ldots \circ x_{k'}) \in \left(\{0,1\}^{\beta \mathsf{L}(\lambda)}\right)^{k'}$, where $k = \overline{k' \cdot \beta \mathsf{L}(\lambda)}$

Output: $c = (c_1 \circ \ldots \circ c_{3t}) \in \{0,1\}^n$, where $t = k'/m$ and $n = \frac{tm\mathsf{L}(\lambda)}{R} \cdot (\beta + 2)$.

- 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, H (see Definition 5.1).
- Let $U_1 := (\ell_{1_1,s} \circ \ell_{1_2,s} \circ \ldots \circ \ell_{1_m,s}), \ldots, U_t := (\ell_{t_1,s} \circ \ell_{t_2,s} \circ \ldots \circ \ell_{t_m,s})$.
- Let $T_1 := (x_1 \circ x_2 \circ \ldots \circ x_m), \ldots, T_t := (x_{(t-1)m+1} \circ x_{(t-1)m+2} \circ \ldots \circ x_{tm})$.
- The codeword c consists of the encoding of groups of message bits, followed by encoding using EncJ of groups of labels, followed by $t = \frac{k'}{m}$ copies of the last encoding.

$$c = (\mathsf{EncJ}(T_1) \circ \mathsf{EncJ}(T_2) \circ \ldots \circ \mathsf{EncJ}(T_t) \circ \mathsf{EncJ}(U_1) \circ \mathsf{EncJ}(U_2) \circ \ldots \circ \mathsf{EncJ}(U_t) \circ \underbrace{\mathsf{EncJ}(U_t) \circ \ldots \circ \mathsf{EncJ}(U_t)}_{t \text{ times}}),$$ where

$$\mathsf{EncJ}(U_i) = \mathsf{EncJ}(\ell_{i_1,s} \circ \ldots \circ \ell_{i_m,s})\ \forall i \in [t].$$

The codeword $c = \mathsf{Enc}(x)$ consists of 3 parts. The first k/R bits correspond to the message symbols x passed $m \cdot \beta\mathsf{L}(\lambda)$ bits at a time through EncJ of rate R. The next $k/(\beta R)$ bits correspond to t codewords $\mathsf{EncJ}(U_j)$, and finally the last $k/(\beta R)$ bits correspond to the repetitions of the final EncJ codeword, $\mathsf{EncJ}(U_t)$. Therefore the length of any codeword produced by Enc is $n = \frac{k}{R} \cdot \frac{\beta+2}{\beta}$. The information rate of the constructed code is therefore $R \cdot \left(\frac{\beta}{\beta+2}\right)$. Therefore appropriately choosing the parameters β and R, we get the rate of the CRLCC approaching 1.

B. Dec *Algorithm*

Let κ be an error parameter to be fixed later. We can show that for any $w \in \{0,1\}^n$ with $\mathsf{dist}(w, c) \leq \frac{\Delta_J \cdot Rn}{\kappa} = \frac{\Delta_J \cdot k}{\kappa}$ the number of tampered meta-nodes $T \subset V(G_0)$ is very small. Let $\mathsf{corrupted}(\tau)$ denote the maximum number of meta-nodes v that the attacker can either corrupt or ensure that v is not α-good with respect to T then it can be shown that $\mathsf{corrupted}(\tau)/t \to 0$ as $\tau \to 0$ i.e., almost all of the meta-nodes u are α-good with respect to the set T of tampered meta-nodes and that any such node u has 2δ-local expansion in the green subgraph G_g. We also show that *any* meta-node $v < t - \mathsf{corrupted}(\tau) - 1$ that has δ'-local expansion with $\delta' \leq 4\delta$ must also be correct because at least one of the last $\mathsf{corrupted}(\tau)+1$ nodes must be α-good and correct and v will be connected to this node via an all green path since we have 4δ-local expansion around both nodes in the green subgraph G_g. While the set T is potentially unknown it is possible to design a test which (with high probability) accepts all α-good meta-nodes and rejects (with high probability) any meta-node u with the property that the green subgraph G_g does not have 4δ-local expansion around u. We can distinguish between the two cases by strategically sampling edges in each of the intervals $A_i = [u, u + 2^i - 1]$ and $B_i = [u + 2^i, u + 2^{i+1} - 1]$. The original meta-graph G_0 contains a δ-bipartite expander H_i connecting A_i and B_i. If u does not have 4δ-local expansion in G_g then for some i a large fraction of the edges in H_i must be red. By contrast, if u is α-good with respect to T then we can show that the number of red edges in H_i will be much smaller. Thus, we can ensure that we can decode at least $\rho(\tau) \geq (1 - 2 \times \mathsf{corrupted}(\tau)/t)$ fraction of the bits in the original codeword so that $\rho(\tau) \to 1$ as $\tau \to 0$. 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 now formalize this notion.

Recall that the codeword $c = \mathsf{Enc}(x)$ consists of 3 parts of t blocks each. The first k/R bits (first t blocks) correspond to the message symbols x passed $m \cdot \beta\mathsf{L}(\lambda)$ bits at a time through EncJ of rate R. The next $k/(\beta R)$ bits (second set of t blocks) correspond to t codewords $\mathsf{EncJ}(U_j)$, $j \in [t]$, and finally the last $k/(\beta R)$ bits (third set of t blocks) correspond to the repetitions of the final block, $\mathsf{EncJ}(U_t)$.

Note: For simplicity of presentation, we let $\beta = 1$ and $R = 1/4$ for the discussion of the decoder in this section. The proof extends similarly for other values of the parameters β and R.

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

Similar to the decoder Dec of Section V-B, 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 $\mathsf{map}(i)$ to be the meta-node corresponding to index i. That is,

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

Before we describe the decoder for the Strong CRLCC, we need a few definitions and supporting lemmas.

The following claim follows from the construction of the metagraph G_0 which was shown in [4]. While [4] was only concerned with bounding the indegree a symmetric argument shows that the outdegree of G_0 is also $\Theta(\log k)$. In particular, [4] constructs the graph G_0 by overlaying $\Theta(\log k)$ constant degree graphs. Thus, the indegree and outdegree of every node is $\Theta(\log k)$. The bounded outdegree allows us to test meta-nodes with a bounded (in this case logarithmic) number of queries.

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

Let $c = (c_1 \circ \ldots \circ c_{3t}) = \mathsf{Enc}(s, x)$. Recall that for a meta-node $u \in [t] = V(G_0)$ we have $c_u = \mathsf{EncJ}(x_{(u-1)m+1} \circ \ldots \circ x_{um})$ and $c_{u+t} = \mathsf{EncJ}(\ell_{u_1,s} \circ \ldots \circ \ell_{u_m,s})$. Given a (possibly) corrupted c' codeword we let $\tilde{c}_i = \mathsf{EncJ}(\mathsf{DecJ}(c'_i))$. We note that if c'_i and c_i are sufficiently close then we will always have $\tilde{c}_i = c_i$. If $\tilde{c}_i \neq c_i$ we say that the block i was tampered and if $\tilde{c}_i = c_i$ we say that the block is *correct*. We let $T \subseteq [t-1]$ denote the set of all nodes $u < t$ (excluding the last node t) with the property that either blocks u or $u+t$ were *tampered*. We also let $\left(\ell'_{u_1,s} \circ \ldots \circ \ell'_{u_m,s}\right) = \mathsf{DecJ}(c_{u+t})$ and $\left(x_{(u-1)m+1} \circ \ldots \circ x_{um}\right) = \mathsf{DecJ}(c_u)$. We say that a node $u_i \in V(G)$ with $\mathsf{parents}(u_i) = u_i^1, \ldots u_i^d \in V(G)$ is green if $\ell'_{u_i,s} = H\left(s, x_{(u-1)m+i} \circ \ell'_{u_i^1,s} \circ \ldots \circ \ell'_{u_i^d,s}\right)$ i.e., the given label of node u_i is consistent with the labeling rule given the data value $x_{(u-1)m+i}$ and the labels of u_i's parents. We say that an edge $(u_m, v_i) \in E(G)$ is green if both u_m and v_i are green.

Definition 6.2: We call a meta-node $u \in V(G_0)$ *green* if *both*:

- At least $\frac{2}{3}$ fraction of the corresponding nodes $u_1, \ldots, u_m \in V(G)$ are green.
- The final node u_m in the meta-node is also green.

We call a meta-node *red* if the meta-node is not green. Similarly, we call a meta-node u *correct* if we are able to correctly decode the blocks c_u and c_{u+t}.

We stress that a *red* meta-node is not *necessarily* tampered nor is a *green* meta-node *necessarily* correct.

1) Query Procedure for $\mathsf{map}(i) = t$*:* Let $\kappa > 1$ be an error parameter. In the case that $\mathsf{map}(i) = t$, we run a procedure $\mathsf{Dec}^w_{=t}(s, i)$ that takes advantage of the fact that any adversary that can change at most $\frac{\Delta_J k}{\kappa}$ coordinates of any codeword obtained from Enc, cannot prevent local correction for a query on the last t blocks corresponding to the repetition of EncJ codeword. Similar to Dec^w_1 in Section V-B, the algorithm $\mathsf{Dec}^w_{=t}(s, i)$ randomly samples a number of blocks, performs decoding using DecJ and takes the majority of the samples.

Lemma 6.3: Let Enc be as described in Section VI-A, and $\kappa > 1$. For any $i \in [n]$ such that $\mathsf{map}(i) = t$, there exists an algorithm $\mathsf{Dec}^w_{=t}$ that 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 $\mathsf{dist}(c, w) \leq \frac{\Delta_J k}{\kappa}$,

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

Moreover, $\mathsf{Dec}^w_{=t}$ makes at most $\mathcal{O}\left(\mathsf{L}(\lambda) \cdot \log^{2+\epsilon} n\right)$ coordinate queries to input w.

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

$\mathsf{Dec}^w_{=t}(s, i)$: Input : Index $i \in [n]$ with $\mathsf{map}(i) = t$.

(1) Sample (with replacement) $r := \mathcal{O}\left(\log^{1+\epsilon} n\right)$ blocks $\{\tilde{y_1}, \ldots, \tilde{y_r}\}$ each of length $4m\mathsf{L}(\lambda)$ from the last t blocks.
(2) Let $y_j = \mathsf{EncJ}(\mathsf{DecJ}(\tilde{y_j}))$ for every $j \in [r]$ to correct each of the sampled blocks.
(3) Output majority of $\{y_j[\hat{i}] \mid j \in [r]\}$, where $\hat{i} = i \pmod{4m\mathsf{L}(\lambda)}$ and $y_j[\hat{i}]$ is the $\hat{i}$-th coordinate in y_j.

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

Let w be received word which is obtained by altering at most $\frac{\Delta_J k}{\kappa}$ coordinates of some codeword i.e, $0 < \mathsf{dist}(c, w) \leq \frac{\Delta_J k}{\kappa}$ for some codeword c obtained from Enc.

We know that the last t blocks of any codeword are exactly the same, *i.e.*, $c_{2t} = c_{2t+1} = \cdots = c_{3t}$.

Since $\mathsf{dist}(c, w) \leq \frac{\Delta_J k}{\kappa}$, at most $\frac{t}{4\kappa}$ blocks are corrupted in more than Δ_J fraction of coordinates. One can therefore decode all blocks among the last t blocks correctly except for those $\frac{t}{4\kappa}$ blocks with large corruption.

Since we query blocks uniformly at random, each query finds a block with large corruption with probability at most $\frac{1}{4\kappa}$. Therefore, the probability that majority of the queried blocks are corrupted beyond repair is at most $\left(\frac{1}{4\kappa}\right)^{r/2} = \mathsf{negl}(n)$, for $r = \mathcal{O}\left(\log^{1+\epsilon} n\right)$.

Since $\mathsf{Dec}_{=t}$ queries $\mathcal{O}\left(\log^{1+\epsilon} n\right)$ blocks each of length $4m \cdot \mathsf{L}(\lambda)$ and $m = \mathcal{O}(\log n)$, then at most $\mathcal{O}\left(\mathsf{L}(\lambda) \cdot \log^{2+\epsilon} n\right)$ queries are made in total. □

2) Query Procedure for $\mathsf{map}(i) < t$*:* Recall from the construction that G_0 is the metagraph of G. We define the green subgraph of G_0 to be the subgraph consisting of exactly the edges whose corresponding endpoints in G are both green nodes. We show there exists an efficient procedure to check whether an edge is in the green subgraph, which we shall use to determine if nodes in G have δ-local expansion in the green subgraph. If there is local expansion around nodes u and v in the green subgraph, then without loss of generality, there exists a path from u to v that only consists of green

nodes by Lemma 6.12. Hence if v is correct, then we show in Lemma 6.14 that either u is also correct, or a hash collision has been found. Nevertheless, this procedure is vacuous if there is a low number of nodes u, v with local expansion in the green subgraph. To address this concern, we show in Lemma 6.10 that for a specific range of α, if a node u is α-good with respect to a tampered set, then there must also be 2δ-local expansion around u in the green subgraph.

In this section, we design the corrector for indices $i \in [n]$ such that $\mathsf{map}(i) < t$.

Recall that we call a meta-node green if most of its nodes including the final node are green. First, we claim the existence of a procedure to test whether a meta-node is green.

Lemma 6.4: Let G_0 be the meta-graph with t vertices used by Enc and let $w = (w_1, \cdots, w_{3t}) \in \{0,1\}^n$ be the corrupted word obtained from the PPT adversary. There exists a procedure $\mathsf{IsGreenMeta}$ that uses $\mathcal{O}\left(\mathsf{L}(\lambda) \cdot \log^2 n\right)$ coordinate queries to w and checks whether a meta-node u of G_0 with labels corresponding to w is green.

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

$\mathsf{IsGreenMeta}(u, w)$: Input : Meta-node index $u \in [t]$, corrupted codeword $w \in \overline{\{0,1\}}^n$

(1) Check if the final node in meta-node u is green:
- Query coordinates of w_{t+u} and retrieve $(\ell'_{u_1}, \ldots, \ell'_{u_m}) := \mathsf{DecJ}(w_{t+u})$.
- Query the coordinates of w_u and retrieve $(x'_{u_1}, \ldots, x'_{u_m}) := \mathsf{DecJ}(w_u)$.
- Check whether $\ell'_{u_m} = H(s, x'_{u_m} \circ \ell'_{u_1} \circ \cdots \circ \ell'_{u_{m-1}})$ for the final node u_m in meta-node u, where $u_1, \ldots, u_{m-1}$ are the parents of u_m, which are all in meta-node u.
- If ℓ'_{u_m} is not consistent, then output 'Red'.

(2) For each node u_j in meta-node u, check whether u_j is green:
- Let u_{j-1} (in meta-node u) and p_r (r-th node in some meta-node p) be the parents of u_j.
- Query w_{t+p} and retrieve ℓ'_{p_r} using $\mathsf{DecJ}(w_{t+p})$.
- Check whether $\ell'_{u_j} = H(s, x'_{u_j} \circ \ell'_{u_{j-1}} \circ \ell'_{p_r})$.

(3) If at least $\frac{2}{3}$ fraction of the nodes in meta-node u are green, then output 'Green'.

(4) Else, output 'Red'.

Recall from Definition 6.2 that a green meta-node first requires that at least $\frac{2}{3}$ of the underlying nodes in meta-node u are green. Since a node is green if its label is consistent with the labels of its parents, then the procedure $\mathsf{IsGreenMeta}$ properly recognizes whether at least $\frac{2}{3}$ of the underlying nodes in u are green after decoding using DecJ. Secondly, a green meta-node requires that the final node in u is green. In this case, the final node u_m has $m-1$ parents, and again the procedure recognizes whether $\ell'_{u_m} = H(s, x'_{u_m} \circ \ell'_{u_1} \circ \cdots \circ \ell'_{u_{m-1}})$ after obtaining the labels using DecJ.

Observe that in Step (1), we query two blocks of w each of length $4m\mathsf{L}(\lambda)$ bits. Also, in Step (2), for each of the $m-1$ nodes u_j in the meta-node u we query one additional block of w corresponding to the parent meta-node of u_j. Since each block is of length $4m\mathsf{L}(\lambda)$ bits long, the total query complexity is $\mathcal{O}\left(m^2\mathsf{L}(\lambda)\right) = \mathcal{O}\left(\mathsf{L}(\lambda) \cdot \log^2 n\right)$ for $m = \mathcal{O}(\log n)$. □

We now define the notion of a green edge in the meta-graph G_0.

Let u and v be meta-nodes which are connected in G_0. From the construction of the degree reduced graph G, we know that there exists an edge from the m-th node of the meta-node u to some node v_u of the meta-node v. We say that an edge from meta-node u to meta-node v is green if the corresponding end points in G are green, *i.e.*, the node u_m of meta-node u and the node v_u in meta-node v are green. We say an edge is *red* if it is not green. Note that it is possible for the end node v_u to be red even if the meta-node v is green.

Given any $w \in \{0,1\}^n$, we now describe a procedure $\mathsf{IsGreenEdge}(u, v, w)$ which locally verifies whether a given edge (u, v) of G_0 is green or not.

Lemma 6.5: Let G_0 be the meta-graph with t vertices used by Enc and let $w = (w_1, \cdots, w_{3t}) \in \{0,1\}^n$ be the corrupted word obtained from the PPT adversary. There exists a procedure $\mathsf{IsGreenEdge}$ that makes $\mathcal{O}(\mathsf{L}(\lambda)\log(n))$ coordinate queries to w and checks whether an edge (u, v) of G_0 with labels corresponding to w is green.

Proof:

$\mathsf{IsGreenEdge}(u, v, w)$: Input : Meta graph edge (u, v), corrupted codeword $w \in \overline{\{0,1\}}^n$

(1) Check if the final node in meta-node u is green:
- Query coordinates of w_{t+u} and retrieve $(\ell'_{u_1}, \ldots, \ell'_{u_m}) := \mathsf{DecJ}(w_{t+u})$.
- Query the coordinates of w_u and retrieve $(x'_{u_1}, \ldots, x'_{u_m}) := \mathsf{DecJ}(w_u)$.
- Check whether $\ell'_{u_m} = H(s, x'_{u_m} \circ \ell'_{u_1} \circ \cdots \circ \ell'_{u_{m-1}})$ for the final node u_m in meta-node u, where $u_1, \ldots, u_{m-1}$ are the parents of u_m, which are all in meta-node u.
- If ℓ'_{u_m} is not consistent, then output 'Red' edge.

(2) Check if the node v_u in meta-node v is green:
- Query coordinates of w_{t+v} and retrieve $(\ell'_{v_1}, \ldots, \ell'_{v_m}) := \mathsf{DecJ}(w_{t+v})$.
- Query the coordinates of w_v and retrieve $(x'_{v_1}, \ldots, x'_{v_m}) := \mathsf{DecJ}(w_v)$.
- Check whether $\ell'_{v_u} = H(s, x'_{v_u} \circ \ell'_{v_{u-1}} \circ \ell'_{u_m})$
- If ℓ'_{v_u} is not consistent, then output 'Red' edge.

(3) Output 'Green' edge.

The procedure $\mathsf{IsGreenEdge}$ first checks if the node u_m of meta-node u is green, and then checks if the connecting node v_u of meta-node v is green. The correctness of $\mathsf{IsGreenEdge}$ then follows from the definition. Note that checks (1) and (2) both query 2 blocks of w, each of length $4\ m\mathsf{L}(\lambda)$. Hence the query complexity of the $\mathsf{IsGreenEdge}$ is $\mathcal{O}(m\mathsf{L}(\lambda)) = \mathcal{O}(\mathsf{L}(\lambda)\log(n))$, for $m = \mathcal{O}(\log(n))$. □

Using the definition of a green edge, we now define a green subgraph of G_0.

Definition 6.6: The green subgraph of G_0, denoted by G_g, is the subgraph of G_0 that contains all green edges of G_0.

A key step in designing the decoder is to verify whether G_g has local expansion property around some given meta-node. Before we describe the verification procedure and why it is necessary, we introduce notations and list some important properties of the meta-graph G_0 and its green subgraph G_g.

Let u be any meta-node of G_0. Let $A^{u,r} := [u, u+r-1]$ and $B^{u,r} := [u+r, u+2r-1]$ and $r < \frac{1}{2}(t-u)$, then from the definition of a δ-local expander, we know that G_0 contains δ-expander between $A^{u,r}$ and $B^{u,r}$. Let us denote this subgraph by $H^{u,r}$, i.e., $H^{u,r} = (A^{u,r} \cup B^{u,r}, E^{u,r})$ with $E^{u,r} \subseteq (A^{u,r} \times B^{u,r}) \cap E$. We assume the following property of G_0: each node $v \in A^{u,r} \cup B^{u,r}$ is indicent to at most d_δ edges in $H^{u,r}$ for some constant $d_\delta \in \mathcal{O}(1)$.

Note: Prior constructions of δ-local expanders such as [4] can be used to satisfy this requirement though [4] does not explicity make use of this particular property. In particular, our δ-local expander G_0 can be constructed using a family δ'-bipartite graphs with constant indegree [4], [27]. The subgraph $H^{u,r}$ can be obtained by taking the edges from $O(1)$ of these δ'-bipartite graphs — we may need to use more than one if the nodes in $H^{u,r}$ are not perfectly aligned with the δ'-bipartite expanders used to construct G_0.

Define $H_g^{u,r}$ to be the green subgraph of $H^{u,r}$, *i.e.*, the subgraph of $H^{u,r}$ restricted to G_g.

Lemma 6.7: Let n_r be the number of red edges in $H^{u,r}$. If $H_g^{u,r}$ is not a 4δ-expander, then $n_r \geq 3\delta r$.

Proof: If $H_g^{u,r}$ is not a 4δ-expander, then there exist subsets $X \subseteq A^{u,r}$, $Y \subseteq B^{u,r}$ with $|X|, |Y| \geq 4\delta r$ such that there are no edges between X and Y in $H_g^{u,r}$. Let Y' denote the nodes that are not connected to X in $H^{u,r}$. Since $H^{u,r}$ is a δ-expander we have $|Y'| \leq \delta r$. , at most δr nodes $Y' \subset Y$ are not connected to any node in X.Thus, $H^{u,r}$ contains at least $|Y - Y'| \geq 3\delta r$ red edges connecting nodes in $Y - Y'$ to X. So the number of edges in $H^{u,r}$ are at least $3\delta r$ all of which are red. □

Remark 6.8: Though the observation states the property for descendants of a meta-node u, i.e., $A^{u,r} = [u, u+r-1]$ and $B^{u,r} = [u+r, u+2r-1]$, we note that it also holds for ancestors as well, *i.e.*, $A^{u,r} = [u-r+1, u]$ and $B^{u,r} = [u-2r-1, u-r]$.

Using the key observation described above, we now give an efficient procedure to verify if the green meta-graph G_g has 4δ-local expansion around a given meta-node u. The idea is to estimate the number of red edges in increasing intervals around the given meta-node and reject if this estimate is large for any interval.

Lemma 6.9: Let G_0 be the meta-graph with t vertices used by Enc and let $w = (w_1, \cdots, w_{3t}) \in \{0,1\}^n$ be the corrupted word obtained from the PPT adversary. There exists a procedure IsLocalExpander that makes $\mathcal{O}\left(m\mathsf{L}(\lambda)\log^{2+\epsilon} t\right)$ coordinate queries to w and rejects any meta-node u around which G_g does not have 4δ-local expansion with probability $1 - \mathsf{negl}(n)$.

Proof: Consider the following algorithm IsLocalExpander:

IsLocalExpander(u, w, δ): Input : meta node u, corrupted codeword $w \in \{0,1\}^n$

(1) Use IsGreenMeta to verify if meta-node u is Green. Reject if it is Red.
(2) For every $p \in \{1, \ldots, \log t\}$:
- Consider a bipartite expander $H^{u,2^p}$, with node sets $A_p = [u, u+2^p-1]$ and $B_p = [u+2^p, u+2^{p+1}-1]$.
- Randomly sample $s := \log^{1+\epsilon} t$ edges from the subgraph with replacement.
- Count the number of red edges in the sample.
- If the number of red edges in the sample is larger than $\frac{5}{2}\delta s$, then 'Reject'.
- Consider a bipartite expander $H'^{u,2^p}$ with node sets $A'_p = [u-2^p+1, u]$ and $B'_p = [u-2^{p+1}+1, u-2^p]$.
- Randomly sample $s := \log^{1+\epsilon} t$ edges from the subgraph with replacement.
- Count the number of red edges in the sample.
- If the number of red edges in the sample is larger than $\frac{5}{2}\delta s$, then 'Reject'.

(3) Accept otherwise.

First, the procedure IsLocalExpander checks if the input node is green using IsGreenMeta procedure, that takes $\mathcal{O}\left(\mathsf{L}(\lambda)\log^2 n\right)$ queries to the input work w (Lemma 6.4).

Note that IsLocalExpander calls the procedure IsGreenEdge for every sampled edge to check whether it is red or green. From Lemma 6.5, we know that each call to IsGreenEdge requires $\mathcal{O}(m\mathsf{L}(\lambda))$ coordinate queries to w. The total number of calls to IsGreenEdge is at most $2\log^{2+\epsilon} t$ since we sample at most $\log^{1+\epsilon} t$ distinct edges from each of the $\log t$ subgraph $H^{u,2^p}$. Therefore, the total query complexity of IsLocalExpander is upper bounded by $\mathcal{O}\left(m\mathsf{L}(\lambda)\log^{2+\epsilon} t\right)$.

Suppose u is a meta-node around which G_g does not have 4δ-local expansion. Then there exists a fixed node set $[u, u+2^p-1]$ and $[u+2^p, u+2^{p+1}-1]$ whose corresponding subgraph has at least $3\delta 2^p$ red edges, by Lemma 6.7. Therefore on sampling uniformly at random with replacement, we expect to see at least $3\delta s$ red edges. So the probability that we see at most $\frac{5}{2}\delta s$ red edges is at most $\exp(-\mathcal{O}\left(\log^{1+\epsilon} t\right))$, from standard Chernoff bounds.

Note that we accept u if we see at most $\frac{5}{2}\delta s$ red edges in all the $2\log t$ intervals. By taking union bound the probability that we accept a meta-node around which G_g does not have 4δ-expansion is at most $\frac{2\log t}{\exp(-\mathcal{O}\left(\log^{1+\epsilon} t\right))}$. □

Now, we list some key properties of the meta-nodes around which G_g has local expansion. These properties are important to understand why we needed the local expansion verification procedure. Essentially we show that if any meta-node is α-good under the set of tampered meta-nodes, then G_g has 2δ-local expansion about it (for appropriately chosen value of

α). Then we show that any two meta-nodes around which G_g has local expansion are connected by a path of green nodes in G. Now similar to the weak CRLCC decoder, we can argue about the correctness of any ancestor of a correct node in a green path (unless the adversary finds a hash collision).

Recall that we call a meta-node u to be α-good under T if every interval for every interval $|[u, u+r-1] \cap T| \leq \alpha r$. We now show that if a meta-node u is α-good under T, then G_g has 2δ-local expansion around u.

Lemma 6.10: Let u be a meta-node that is α-good with respect to T. Then there is 2δ-local expansion around u in G_g for any $\alpha < \frac{\delta}{2}$.

Proof: Given any $r > 0$, consider the intervals $A = [u, u+r]$ and $B = [u+r+1, u+2r]$. Recall that G_0 is a δ-local expander. Let $X \subseteq A$ and $Y \subseteq B$ be given sets with size $2\delta r$ each. Since u is α-good with respect to T, then $|T \cap X| \leq 2\alpha r$ and $|T \cap Y| \leq 2\alpha r$. Thus, $|X - T| \geq (2\delta - 2\alpha)r$ and similarly, $|Y - T| \geq (2\delta - 2\alpha)r$. Setting $\alpha < \frac{\delta}{2}$ shows that both $X - T$ and $Y - T$ contains at least δr nodes, and so there exists an edge between $X - T$ and $Y - T$. Therefore, there is local expansion around u. □

Moreover, we see that the number of red edges in the 2δ-expander is at most $3\alpha r d_\delta$.

Lemma 6.11: If u is a meta-node that is α-good under T, then for any $r > 0$ the number of red edges in between the intervals $[u, u+r-1]$ and $[u+r, u+2r-1]$ is at most $3\alpha r d_\delta$.

Proof: Since u is α-good under T, the intervals $A := [u, u+r-1]$ and $B := [u+r, u+2r-1]$ contain at most αr and $2\alpha r$ tampered nodes respectively. Since we assumed that the δ-expander contained in A and B has constant degree d_δ, the number of red edges is upper bounded by $3\alpha r d_\delta$. □

So in order to ensure that this number is not too large (for IsLocalExpander to succeed), we choose α such that $3\alpha r d_\delta \ll 3\delta r$. This guarantees that the procedure IsLocalExpander definitely accepts nodes that are α-good under T.

We now show that any two meta-nodes with local expansion property are connected by a path of green edges in G_g. This also ensures that the corresponding nodes are connected by a path of green nodes in the underlying graph G.

Lemma 6.12: If G_g has 4δ-local expansion around the meta nodes u and v, then there exists a path from u to v in G_g.

Proof: Let r be such that $v = u + 2r - 1$. Consider the 4δ expander between $A := [u, u+r-1]$ and $B := [u+r, u+2r-1]$. We show using Claim 6.13 that u is connected to at least $4\delta r$ meta-nodes in A, and similarly v is reachable from at least $4\delta r$ meta-nodes in B. Now since G_g has 4δ expansion around u, there exists an edge between any two large enough subset of nodes of A and B. Hence, the set of nodes reachable to u in A are connected to the set of nodes which are reachable from v in B.

It now remains to show that u and v are connected to at least $4\delta r$ meta-nodes in A and B respectively. Let i be such that $2^i \leq r < 2^{i+1}$. From Claim 6.13, we get that u is connected to at least $\frac{3}{4} \cdot 2^i = \frac{3}{8} \cdot 2^{i+1} \geq 4\delta r$ (for $\delta < \frac{1}{16}$) meta-nodes in the interval $[u, u+r-1]$. Using a similar argument we conclude that v is also connected to at least $4\delta r$ nodes in $[v-r+1, v] = [u+r, u+2r-1]$.

We now prove Claim 6.13.

Claim 6.13: For any $i > 0$, the number of meta-nodes in the interval $[u, u+2^i-1]$ reachable from u is at least $\frac{3}{4} \cdot 2^i$. Similarly, the number of meta-nodes in the interval $[u-2^i+1, u]$ reachable from u is at least $\frac{3}{4} \cdot 2^i$

Proof: We prove this claim by induction on i. Let $R_i(u)$ denote the set of meta-nodes in $[u, u+2^i-1]$ reachable from u. We have to show that $|R_i(u)| \geq \frac{3}{4} \cdot 2^i$.

In the base case, for $i = 0$, we know from 4δ-expansion of $[u, u+1]$ around u that there is at least one edge between u and $u+1$ in G_g. Hence, $|R_0(u)| \geq 1$.

Let us assume the induction hypothesis for all $i \leq i_0$. To prove the induction step for $i = i_0+1$, consider the intervals $A := [u, u+2^{i_0}-1]$ and $B := [u+2^{i_0}, u+2^{i_0+1}-1]$. Let NR denote the set of non-reachable meta-nodes in B not reachable from A. We know that $|NR| < 4\delta \cdot 2^{i_0}$. This follows from the fact that G_g has 4δ-local expansion around u, and hence if $|NR| \geq 4\delta \cdot 2^{i_0}$, then there would be a an edge between $R_{i_0}(u)$ and NR which would contradict the definition of NR. Therefore,

$$\begin{aligned} R_{i_0+1}(u) &\geq R_{i_0}(u) + 2^{i_0} - |NR| \\ &\geq \frac{3}{4} \cdot 2^{i_0} + 2^{i_0} - 4\delta \cdot 2^{i_0} \quad \text{(from IH)} \\ &\geq \frac{3}{4} \cdot 2^{i_0+1} \quad (\text{for } \delta < \tfrac{1}{16}). \end{aligned}$$

□

□

Lemma 6.14: Suppose there exists a path from meta-node u to v in the green subgraph G_g. If v is untampered (correct), then either u is also untampered or the adversary has found a hash collision.

Proof: The proof of this lemma follows from the proof of Lemma 5.3 once we show that there exists a path in G with all green nodes from any node in the meta-node u to some node in the meta-node v.

Consider the path from meta-nodes u to v in G_g. Any two adjacent edges (p, q) and (q, r) in (u, v) path in G_g corresponds to the edge from the last node p_m of p to some node q_i $(i < m)$ of meta-node q, and an edge from the last node q_m of q to some node r_j $(j < m)$ of meta-node r. Since these edges are green, we know that the nodes p_i, p_m and q_j are green. Now, from the construction of the graph G, we conclude that there is an edge from p_i to p_m. Therefore, there is a path in G from any node in the meta-node u to some node in the meta-node v with all green nodes. □

Equipped with all the necessary procedures, we now present the decoder for the Strong CRLCC for any coordinate query $i \in [n]$ such that $\mathsf{map}(i) < t$.

$\mathsf{Dec}^{w}_{<t}$: Input : Index $i \in [n]$ with $u := \mathsf{map}(i) < t$,

$\frac{\delta}{20d_\delta} \leq \alpha \leq \frac{\delta}{10d_\delta}$, $\delta < \frac{1}{16}$ space parameter $\epsilon > 0$.

(1) Use $\mathsf{IsLocalExpander}$ to verify if G_g has 4δ-local expansion around u.

(2) If $\mathsf{IsLocalExpander}$ accepts u, then
- If $u < \frac{3t}{4}$ then return w_i.
- Else if $\frac{3t}{4} \leq u < t$:
  - Use $\mathsf{Dec}^w_{=t}(w, j)$ for all $8tm\mathsf{L}(\lambda) - 4m\mathsf{L}(\lambda) \leq j < 8tm\mathsf{L}(\lambda)$ to reconstruct the last block w_{2t}.
  - If G_g has 4δ-local expansion around meta-node t using the constructed labeling, then return w_i, else return $\perp$.

(3) If $\mathsf{IsLocalExpander}$ rejects u, then return $\perp$.

Lemma 6.15: Let Enc be as described in Section VI-A, and $\kappa > 1600\ d_\delta$. For any $i \in [n]$ such that $\mathsf{map}(i) < t$, $\mathsf{Dec}_{<t}$ 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 $\mathsf{dist}(c, w) \leq \frac{\Delta_J k}{\kappa}$,

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

Moreover, $\mathsf{Dec}^w_{<t}$ makes at most $\mathcal{O}\left(\mathsf{L}(\lambda)\log^{3+\epsilon} n\right)$ queries to input w.

Before we prove the correctness of Lemma 6.15, we show that any PPT adversary that generates a corrupted codeword w such that $\mathsf{dist}(w, c) \leq \frac{\Delta_J k}{\kappa}$, cannot manage to corrupt many meta-nodes. Let T denote the set of tampered meta-nodes of G_0.

Lemma 6.16: Let $w \in \{0,1\}^n$ be a corrupted codeword generated by any PPT adversary such that $\mathsf{dist}(w, C) \leq \frac{\Delta_J k}{\kappa}$, then at most $\frac{t}{4\kappa}$ meta-nodes are tampered.

Proof: We first observe that a meta-node can only be altered beyond repair by changing at least $4m \cdot \mathsf{L}(\lambda) \cdot \Delta_J$ bits. Therefore, by changing at most $\frac{\Delta_J k}{\kappa}$ bits of the codeword we have $|T| \leq \frac{k}{4\kappa m\mathsf{L}(\lambda)} = \frac{k'}{4\kappa m} = \frac{t}{4\kappa}$ meta-nodes in G are tampered. □

Using these key lemmas and properties, we now prove the correctness of the decoder.

Proof of Lemma 6.15:

Query Complexity: The query complexity of the decoder is dominated by the query complexity of $\mathsf{IsLocalExpander}$. From Lemma 6.9, it then follows that the query complexity of $\mathsf{Dec}^w_{<t}$ is at most $\mathcal{O}\left(m\mathsf{L}(\lambda)\log^{2+\epsilon} t\right) = \mathcal{O}\left(m\mathsf{L}(\lambda)\log^{2+\epsilon} n\right)$ for $m = \mathcal{O}(\log n)$.

Correctness (1): If $w \in C$, then the green graph $G_g = G_0$ and hence, G_g has δ expansion around all meta-nodes u. Also, since $T = \emptyset$, all meta-nodes are α-good in G_0 with respect to T. Therefore, $\mathsf{Dec}_{<t}$ accepts a codeword with probability 1.

Correctness (2): Now, let w be a corrupted codeword such that $\mathsf{dist}(d, c) \leq \frac{\Delta_J k}{\kappa}$. Let $u = \mathsf{map}(i)$ be the meta-node corresponding to the queried index $i \in [n]$.

Case (i): Let $u < \frac{3t}{4}$, and let G_g have 4δ-local expansion around u. We show that there exists a descendent v of u in G_g such that (1) G_g is 4δ-local expander around v and, (2) v is untampered (correct).

Recall that T denotes the set of tampered meta-nodes. From Lemma 6.16, we know that at most $\frac{t}{4\kappa}$ meta-nodes can be tampered by the adversary. Since G_0 is a δ-local expander, from Lemma 4.8 we know that the number of α-good meta-nodes in G_0 with respect to the set of tampered meta-nodes T is at least $t - |T|\left(\frac{2-\alpha}{\alpha}\right)$. Therefore, for any $\alpha > \frac{1}{200\ d_\delta}$ and $\kappa \geq 1600\ d_\delta$, there are at least $\frac{15t}{16}$ α-good meta-nodes in G_0 w.r.t T. Lemma 6.10 then implies that G_g has 4δ-local expansion around all these α good meta-nodes. So at most $\frac{t}{16} + \frac{t}{16} = \frac{t}{8}$ meta-nodes which do not satisfy conditions (1) or (2). Therefore, there exists at least one among the last $\frac{t}{4}$meta-nodes which satisfy both the conditions.

Case (ii): Let $u \geq \frac{3t}{4}$, and let t be the corrected meta-node using $\mathsf{Dec}^w_{=t}$ (see Lemma 6.3). If G_g has 4δ-local expansion around u and t, then assuming that the adversary has not found a hash collision, we can conclude from Lemma 6.14, that u is untampered. So $\mathsf{Dec}^w_{<t}$ returns the correct value for all such coordinate queries.

The probability that the decoder returns a wrong value is upper bounded by the probability that the procedure $\mathsf{IsLocalExpander}$ wrongly accepts a meta-node about which G_g does not have 4δ local expansion or if the adversary successfully finds a hash collision. This happens with $\mathsf{negl}(n)$ probability as shown in Lemma 6.9. □

Equipped with the decoder for the Strong-CRLCC constructed in Section VI-A, we now prove Theorem 2.5.

Proof of Theorem 2.5: 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 VI-A and the decoder Dec described in Section VI-B above. We claim that the triple $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a Strong-CRLCC.

From the construction of Enc, we know that the block length of a codeword produced by Enc is $n = \frac{k}{R}\left(\frac{\beta+2}{\beta}\right)$. Therefore, for appropriate choice of $\beta = \beta(\tau)$ such that $1/\beta \to 0$ as $\tau \to 0$, the information rate of the CRLCC approaches 1 since $R = R(\tau) \to 1$ as $\tau \to 0$.

From Lemma 6.3 we know that on input (w, i, x) generated by any PPT adversary such that $\mathsf{dist}(\mathsf{Enc}(x), w) \leq \frac{\Delta_J k}{\kappa}$ and $\mathsf{map}(i) = t$, $\mathsf{Dec}^w_{=t}$ queries at most $\mathcal{O}\left(\mathsf{L}(\lambda)\log^{2+\epsilon} n\right)$ coordinates of w and returns $b = \mathsf{Enc}(x)[i]$ with probability at least $1 - \mathsf{negl}(n)$. If $\mathsf{map}(i) < t$, then from Lemma 6.15, we know that $\mathsf{Dec}^w_{<t}$ queries $\mathcal{O}\left(\mathsf{L}(\lambda)\log^{3+\epsilon} n\right)$ coordinates of w and returns $b \in \{\mathsf{Enc}(x)[i], \perp\}$ with probability at least $1 - \mathsf{negl}(n)$. Also, both the decoders, $\mathsf{Dec}_{=t}$, and $\mathsf{Dec}_{<t}$ on input any valid encoding $(\mathsf{Enc}(x), i, x)$ return $\mathsf{Enc}(x)[i]$ with probability 1.

To conclude, we need to show that for any received word generated by a PPT adversary most meta-nodes are corrected by the decoder (i.e $\mathsf{Dec}^w(s, i) \neq \perp$). In particular, we show that most meta-nodes of G_0 are untampered, and G_g had 4δ local expansion around each of these untampered meta-nodes.

This follows from the fact that there are at most $\frac{t}{16}$ meta-nodes are not α-good under the set of tampered nodes. Note that for any query to a meta-node $u < \frac{3t}{4}$ the decoder $\mathsf{Dec}^w(s,i)$ returns the correct codeword symbol, *i.e.*, $c[i]$ unless u itself is not α-good under the set of tampered nodes. So, for at least $\frac{3t}{4} - \frac{t}{16} = \frac{11t}{16}$ meta-node queries, $\mathsf{Dec}^w(s,i)$ does not return a $\perp$.

We refer the reader to Appendix VIII-B and Appendix IX-B for the list of parameters and the proof flowchart for the construction of Strong-CRLCC. □

VII. Computationally Relaxed Locally Decodable Codes (CRLDC)

In this section we present the formal definition of Computationally Relaxed Locally Decodable Codes (CRLDC) — defined informally in the main body of the paper. We first tweak the definition of a *local code* so that local decoding algorithm takes as input an index from the range $i \in [k]$ instead of $i \in [n]$ i.e., the goal is to decode a bit of the original message as opposed to a bit of the original codeword.

Definition 7.1: 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 $i \in [k]$, 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 at position i of the original message x i.e., the string x s.t. $\mathsf{Enc}(s,x)$ is closest codeword to w).

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.

Similarly, the notional of a computational adversarial channel is almost identical except that the channel challenges the Dec with an index $i \in [k]$ (as opposed to $i \in [n]$) and the decoder is supposed to output the ith bit of the original message. Apart from this change Definition 7.2 and Definition 2.2 are identical.

Definition 7.2: A *computational adversarial channel* $\mathcal{A}$ with error rate τ is an algorithm that interacts with a local code $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ 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 $w \in \Sigma^n$ and selects a challenge index $i \in [k]$; w is given to the receiver's Dec with query access along with the challenge index i.
(5) The receiver outputs $b \leftarrow \mathsf{Dec}^w(s,i)$.

We define $\mathcal{A}(s)$'s *probability of fooling* Dec on this round to be $p_{\mathcal{A},s} = \Pr[b \notin \{\perp, x_i\}]$, 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 $|\mathsf{Good}_{\mathcal{A},s}| < \rho \cdot k$, where $\mathsf{Good}_{\mathcal{A},s} \subseteq [k]$ is the set of indices j such that $\Pr[\mathsf{Dec}^w(s,j) = x_j] > \frac{2}{3}$. 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.

Definition 7.3 ((Computational) Relaxed Locally Decodable Codes (CRLDC)): A local code $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a $(q,\tau,\rho,\gamma(\cdot),\mu(\cdot))$-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 x_i.
(2) For all $\mathcal{A} \in \mathbb{A}$ we have $\Pr[\mathsf{Fool}_{\mathcal{A},s}(\gamma(\lambda),\tau,\lambda)] \leq \mu(\lambda)$, 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 $\mu(\lambda) = 0$, $\gamma(\lambda) = \frac{1}{3}$ is a constant and $\mathbb{A}$ is the set of all (computationally unbounded) channels we say that the code is a (q,τ,ρ,γ)-RLDC. 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)-CRLDC (computational relaxed locally correctable code).

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

As we remarked in the main body of the paper our construction of a Strong CRLCC is also a strong CRLDC. In particular, Theorem 7.4 is identical to Theorem 2.5 except that we replaced the word CRLCC with CRLDC.

Theorem 7.4: Assuming the existence of a collision-resistant hash function (GenH, H) with length $\mathsf{L}(\lambda)$, there exist a constant $0 < \tau' < 1$ and negligible functions $\mu(\cdot), \gamma(\cdot)$ such that the following holds: for all $\tau \leq \tau'$, there are constants $0 < r(\tau), \rho(\tau) < 1$ and a $(\mathsf{L}(\lambda) \cdot \mathsf{polylog}\, n, \tau, \rho(\tau), \gamma(\cdot), \mu(\cdot))$-Strong CRLDC of blocklength n over the binary alphabet with rate $r(\tau)$.

Moreover, $\lim_{\tau \to 0} r(\tau) = \lim_{\tau \to 0} \rho(\tau) = 1$. In particular, if $\mathsf{L}(\lambda) = \mathsf{polylog}\, \lambda$ and $\lambda = \Theta(n)$, then the code is a $(\mathsf{polylog}\, n, \tau, \rho(\tau), \gamma(\cdot), \mu(\cdot))$-Strong CRLDC.

Proof: (sketch) The encoding algorithm in our strong CRLDC and strong CRLCC constructions are identical. The only change that we need to make is to *tweak* the local decoding algorithm to output bits of the original message instead of bits of the codeword. This task turns out to be trivial. In particular, the first part of the codeword in our construction is formed by partitioning the original message x into mt blocks $x = x_1 \circ \ldots \circ x_{tm}$, partitioning these blocks into t groups $T_1 = (x_1 \circ \ldots x_m), \ldots, T_t = (x_{(t-1)m+1} \circ \ldots x_{tm})$ and outputting $c_j = \mathsf{EncJ}(T_j)$ for each $j \leq t$. Because our rate $r(\tau)$ approaches 1 these bits account for *almost all* of the codeword.

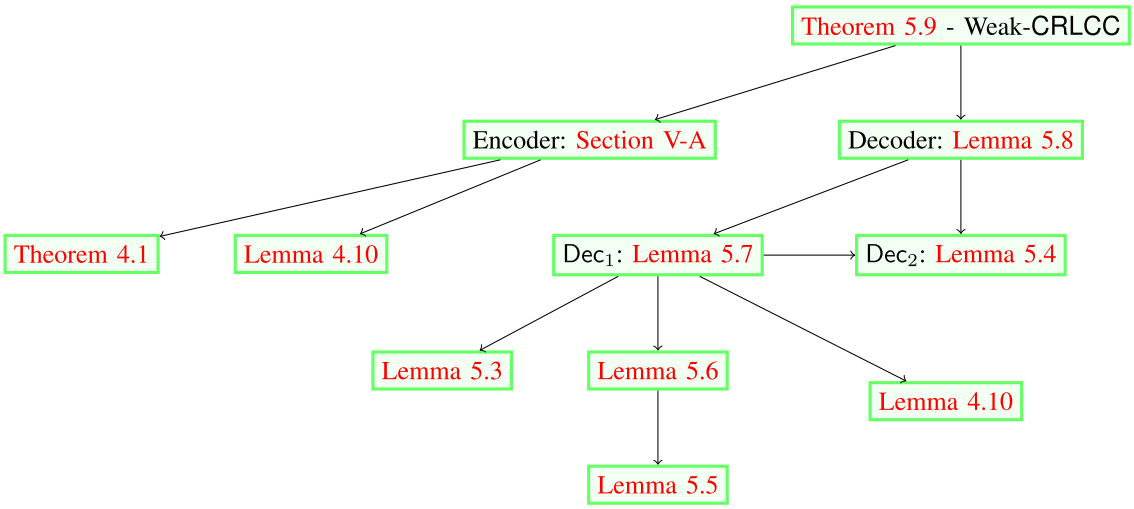

Fig. 3. The following figure depicts the dependency graph for the proof of Theorem 5.9.

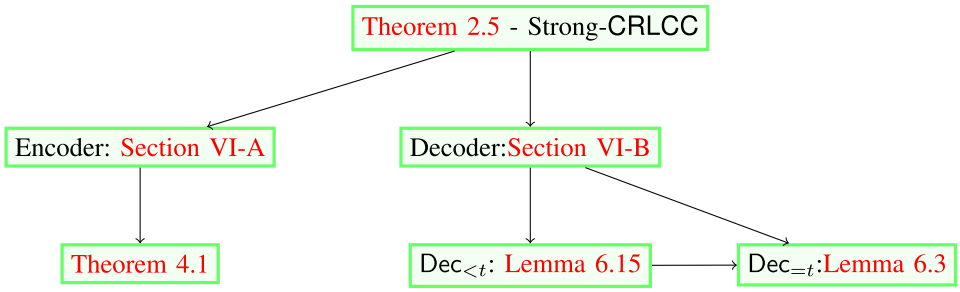

Fig. 4. Dependency graph for the proof of Theorem 2.5.

The CRLCC decoding algorithm is given (possibly tampered) codeword $\tilde{c} = \tilde{c}_1 \circ \tilde{c}_2 \ldots$. When the decoding algorithm is challenged for one of the original bits of c_j it works as follows: (1) use our randomized algorithm to verify that the corresponding metanode v_j in our graph is α-good (if not we output $\perp$), (2) if v_j is α-good then run $\mathsf{DecJ}(\tilde{c}_j)$ to recover the *original* $T_j = (x_{(j-1)m+1}, \ldots, x_{jm})$, (3) run $\mathsf{EncJ}(T_j)$ to recover the original c_j and then find the appropriate bit of the codeword to output. The CRLDC decoding algorithm can simply omit step (3). Once we have recovered the original T_j we can find the appropriate bit of the original message to output. □

APPENDIX

VIII. PARAMETERS

A. Parameters for Weak-CRLCC

- λ: Security parameter. We set $\lambda = \mathcal{O}(n)$.
- $\mathsf{L}(\lambda)$: Length of the hash function. We consider $\mathsf{L}(\lambda) = \mathsf{polylog}(\lambda)$.
- Σ: Code alphabet. We consider $\Sigma = \{0,1\}$ for our constructions.
- k: length of message.
- n: length of codeword. We construct codes with $n = 12k$.
- $k' = k/\mathsf{L}(\lambda)$: number of nodes in the δ- local expander graph G.
- $\mathsf{degree}(G) = \mathcal{O}(\log k')$.
- $\delta < \min((1-\alpha)/2, 1/4)$.
- $\alpha \in (0, 3/4)$.
- $\epsilon > 0$, $\epsilon = \Theta(1)$.

B. Parameters for Strong-CRLCC

- t: number of meta-nodes in δ-local expander G_0
- $m = \max\{\mathsf{indeg}(G_0), \mathsf{outdeg}(G_0)\} + 1$, $m = \mathcal{O}(\log t)$.
- k: length of message.
- $k' = mt = k/\beta\mathsf{L}(\lambda) = \mathcal{O}(t \log t)$: number of nodes in the degree-reduced graph
- $R(\tau)$, $\Delta_J(\tau)$: Information rate and relative distance of DecJ. As $\tau \to 0$, $R(\tau) \to 1$, $\Delta_J(\tau) \to 0$. We set $R = 1/4$ in the decoder for the ease of presentation.
- $\beta = \beta(\tau) > 0$. We require $1/\beta(\tau) \to 0$, as $\tau \to 0$. We set $\beta = 1$ in the decoder for the ease of presentation.
- n: length of codeword. We construct codes with $n = \frac{k}{R} \cdot \frac{\beta+2}{\beta}$.
- $\delta = \delta(\tau) < 1/16$. and, $\delta < \min((1-\alpha)/2, 1/4)$.
- $\alpha = \alpha(\tau) < 1 - 2\delta(\tau)$. We choose $\frac{\delta}{20\ d_\delta} \leq \alpha \leq \frac{\delta}{10\ d_\delta}$. Therefore, for $\delta < 1/16$, we have $\alpha > 1/200\ d_\delta$.
- $d_\delta = m/\log n = \Theta(1)$ that depends on δ. As δ decreases, d_δ increases.
- $\kappa > 1600\ d_\delta$: error parameter.
- $\epsilon > 0$, $\epsilon = \Theta(1)$.

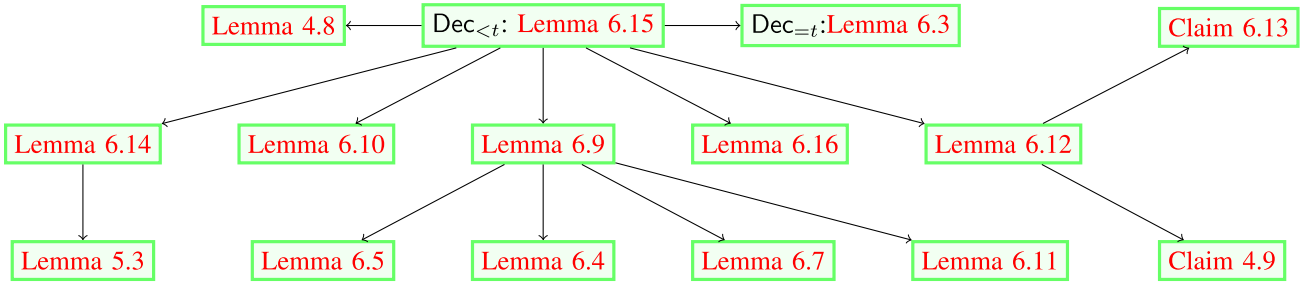

Fig. 5. Dependency graph for Lemma 6.15.

IX. Proof Flowcharts

A. Proof Flow for Weak-CRLCC

See Fig. 3.

B. Proof Flow for Strong-CRLCC

See Figs. 4 and 5.

Acknowledgment

The authors are indebted to the anonymous reviewers, whose suggestions and comments greatly helped improve the quality of the presentation.

References

[1] J. Alwen and J. Blocki, "Efficiently computing data-independent memory-hard functions," in *Advances in Cryptology—CRYPTO 2016* (Lecture Notes in Computer Science), vol. 9815, M. Robshaw and J. Katz, Eds. Heidelberg, Germany: Springer, Aug. 2016, pp. 241–271.

[2] J. Alwen, J. Blocki, and B. Harsha, "Practical graphs for optimal side-channel resistant memory-hard functions," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. New York, NY, USA: ACM, 2017, pp. 1001–1017.

[3] J. Alwen, J. Blocki, and K. Pietrzak, "Depth-robust graphs and their cumulative memory complexity," in *Advances in Cryptology—EUROCRYPT 2017* (Lecture Notes in Computer Science), vol. 10212, J.-S. Coron and J. B. Nielsen, Eds. Heidelberg, Germany: Springer, 2017, pp. 3–32.

[4] J. Alwen, J. Blocki, and K. Pietrzak, "Sustained space complexity," in *Advances in Cryptology—EUROCRYPT 2018* (Lecture Notes in Computer Science), vol. 10821, J. B. Nielsen and V. Rijmen, Eds. Heidelberg, Germany: Springer, 2018, pp. 99–130.

[5] J. Alwen, S. F. de Rezende, J. Nordström, and M. Vinyals, "Cumulative space in black-white pebbling and resolution," in *Proc. 8th Innov. Theor. Comput. Sci. Conf. (ITCS)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[6] A. Andoni, T. Laarhoven, I. Razenshteyn, and E. Waingarten, "Optimal hashing-based time-space trade-offs for approximate near neighbors," in *Proc. 28th Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2017, pp. 47–66.

[7] J. Alwen and V. Serbinenko, "High parallel complexity graphs and memory-hard functions," in *Proc. 47th ACM STOC*, R. A. Servedio and R. Rubinfeld, Eds. New York, NY, USA: ACM, 2015, pp. 595–603.

[8] V. R. Asadi and I. Shinkar, "Relaxed locally correctable codes with improved parameters," 2020, *arXiv:2009.07311*. [Online]. Available: http://arxiv.org/abs/2009.07311

[9] J. Alwen and B. Tackmann, "Moderately hard functions: Definition, instantiations, and applications," in *Theory of Cryptography* (Lecture Notes in Computer Science), vol. 10677, Y. Kalai and L. Reyzin, Eds. Heidelberg, Germany: Springer, 2017, pp. 493–526.

[10] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy, "Checking computations in polylogarithmic time," in *Proc. 23rd Annu. ACM Symp. Theory Comput. (STOC)*, 1991, pp. 21–31.

[11] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, "Robust PCPs of proximity, shorter PCPs, and applications to coding," *SIAM J. Comput.*, vol. 36, no. 4, pp. 889–974, Jan. 2006.

[12] M. Blum and S. Kannan, "Designing programs that check their work," *J. ACM*, vol. 42, no. 1, pp. 269–291, Jan. 1995.

[13] D. J. Bernstein and T. Lange, "Non-uniform cracks in the concrete: The power of free precomputation," in *Advances in Cryptology—ASIACRYPT 2013* (Lecture Notes in Computer Science), vol. 8270, K. Sako and P. Sarkar, Eds. Heidelberg, Germany: Springer, 2013, pp. 321–340.

[14] M. Blum, M. Luby, and R. Rubinfeld, "Self-testing/correcting with applications to numerical problems," *J. Comput. Syst. Sci.*, vol. 47, no. 3, pp. 549–595, Dec. 1993.

[15] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. ACM CCS*, V. Ashby, Ed. New York, NY, USA: ACM, 1993, pp. 62–73.

[16] J. Blocki, L. Ren, and S. Zhou, "Bandwidth-hard functions: Reductions and lower bounds," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 1820–1836.

[17] J. Blocki and S. Zhou, "On the depth-robustness and cumulative pebbling cost of Argon2i," in *Theory of Cryptography* (Lecture Notes in Computer Science), vol. 10677, Y. Kalai and L. Reyzin, Eds. Heidelberg, Germany: Springer, 2017, pp. 445–465.

[18] V. Chen, E. Grigorescu, and R. D. Wolf, "Error-correcting data structures," *SIAM J. Comput.*, vol. 42, no. 1, pp. 84–111, Jan. 2013.

[19] A. Chiesa, T. Gur, and I. Shinkar, "Relaxed locally correctable codes with nearly-linear block length and constant query complexity," in *Proc. 14th Annu. ACM-SIAM Symp. Discrete Algorithms*. Philadelphia, PA, USA: SIAM, 2020, pp. 1395–1411.

[20] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998.

[21] B. Cohen and K. Pietrzak, "Simple proofs of sequential work," in *Advances in Cryptology—EUROCRYPT 2018* (Lecture Notes in Computer Science), vol. 10821, J. B. Nielsen and V. Rijmen, Eds. Heidelberg, Germany: Springer, 2018, pp. 451–467.

[22] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," in *Advances in Cryptology—CRYPTO 2015* (Lecture Notes in Computer Science), vol. 9216, R. Gennaro and M. J. B. Robshaw, Eds. Heidelberg, Germany: Springer, 2015, pp. 585–605.

[23] Y. Ding, P. Gopalan, and R. Lipton, "Error correction against computationally bounded adversaries," Tech. Rep., 2004.

[24] Z. Dvir, P. Gopalan, and S. Yekhanin, "Matching vector codes," *SIAM J. Comput.*, vol. 40, no. 4, pp. 1154–1178, Jan. 2011.

[25] I. Dinur and P. Harsha, "Composition of low-error 2-query PCPs using decodable PCPs," in *Proc. 50th Annu. IEEE Symp. Found. Comput. Sci.*, Oct. 2009, pp. 472–481.

[26] K. Efremenko, "3-query locally decodable codes of subexponential length," *SIAM J. Comput.*, vol. 41, no. 6, pp. 1694–1703, Jan. 2012.

[27] P. Erdös, R. L. Graham, and E. Szemerédi, "On sparse graphs with dense long paths," *Comput. Math. Appl.*, vol. 1, nos. 3–4, pp. 365–369, 1975.

[28] I. W. Gasarch, "A survey on private information retrieval (column: Computational complexity)," *Bull. EATCS*, vol. 82, nos. 72–107, p. 113, 2004.

[29] O. Goldreich, "On post-modern cryptography," Cryptol. ePrint Arch., Tech. Rep. 2006/461, 2006. [Online]. Available: https://eprint.iacr.org/

[30] T. Gur, G. Ramnarayan, and D. R. Rothblum, "Relaxed locally correctable codes," in *Proc. ITCS*, 2018, pp. 27:1–27:11.

[31] V. Guruswami and A. Smith, "Optimal rate code constructions for computationally simple channels," *J. ACM*, vol. 63, no. 4, pp. 1–37, Nov. 2016.

[32] B. Hemenway and R. Ostrovsky, "Public-key locally-decodable codes," in *Proc. Annu. Int. Cryptol. Conf.* Berlin, Germany: Springer, 2008, pp. 126–143.

[33] B. Hemenway, R. Ostrovsky, M. J. Strauss, and M. Wootters, "Public key locally decodable codes with short keys," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Berlin, Germany: Springer, 2011, pp. 605–615.
[34] J. Justesen, "Class of constructive asymptotically good algebraic codes," *IEEE Trans. Inf. Theory*, vol. IT-18, no. 5, pp. 652–656, Sep. 1972.
[35] I. Kerenidis and R. D. Wolf, "Exponential lower bound for 2-query locally decodable codes via a quantum argument," *J. Comput. Syst. Sci.*, vol. 69, no. 3, pp. 395–420, Nov. 2004.
[36] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2014.
[37] N. Koblitz and A. J. Menezes, "Another look at 'provable security,'" *J. Cryptol.*, vol. 20, no. 1, pp. 3–37, Jan. 2007.
[38] N. Koblitz and A. Menezes, "The random oracle model: A twenty-year retrospective," Cryptol. ePrint Arch., Tech. Rep. 2015/140, 2015. [Online]. Available: https://eprint.iacr.org/
[39] S. Kopparty, O. Meir, N. Ron-Zewi, and S. Saraf, "High-rate locally correctable and locally testable codes with sub-polynomial query complexity," *J. ACM*, vol. 64, no. 2, pp. 1–42, Jun. 2017.
[40] S. Kopparty and S. Saraf, "Guest column: Local testing and decoding of high-rate error-correcting codes," *ACM SIGACT News*, vol. 47, no. 3, pp. 46–66, Aug. 2016.
[41] J. Katz and L. Trevisan, "On the efficiency of local decoding procedures for error-correcting codes," in *Proc. 32nd Annu. ACM Symp. Theory Comput. (STOC)*, 2000, pp. 80–86.
[42] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," *J. ACM*, vol. 39, no. 4, pp. 859–868, Oct. 1992.
[43] T. Gur and O. Lachish, "On the power of relaxed local decoding algorithms," *SIAM J. Comput.*, vol. 50, no. 2, pp. 788–813, 2021.
[44] R. J. Lipton, "A new approach to information theory," in *Proc. STACS*, 1994, pp. 699–708.
[45] A. Menezes, "Another look at provable security (invited talk)," in *Advances in Cryptology—EUROCRYPT 2012* (Lecture Notes in Computer Science), vol. 7237, D. Pointcheval and T. Johansson, Eds. Heidelberg, Germany: Springer, 2012, p. 8.
[46] M. Mahmoody, T. Moran, and S. P. Vadhan, "Publicly verifiable proofs of sequential work," in *Proc. ITCS*, R. D. Kleinberg, Ed. New York, NY, USA: ACM, 2013, pp. 373–388.
[47] S. Micali, C. Peikert, M. Sudan, and D. A. Wilson, "Optimal error correction against computationally bounded noise," in *Proc. Theory Cryptogr. Conf.* Berlin, Germany: Springer, 2005, pp. 1–16.
[48] D. Moshkovitz and R. Raz, "Two-query PCP with subconstant error," *J. ACM*, vol. 57, no. 5, pp. 1–29, Jun. 2010.
[49] R. Ostrovsky, O. Pandey, and A. Sahai, "Private locally decodable codes," in *Proc. Int. Colloq. Automata, Lang., Program. (ICALP)*, 2007, pp. 387–398.
[50] L. Ren and S. Devadas, "Bandwidth hard functions for ASIC resistance," in *Theory of Cryptography* (Lecture Notes in Computer Science), vol. 10677, Y. Kalai and L. Reyzin, Eds. Heidelberg, Germany: Springer, 2017, pp. 466–492.
[51] M. Sipser and D. A. Spielman, "Expander codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1710–1722, Nov. 1996.
[52] R. Shaltiel and J. Silbak, "Explicit list-decodable codes with optimal rate for computationally bounded channels," in *Proc. Approximation, Randomization, Combinat. Optim. Algorithms Techn. (APPROX/RANDOM)*, 2016, pp. 45:1–45:38.
[53] M. Sudan, L. Trevisan, and S. Vadhan, "Pseudorandom generators without the XOR lemma (extended abstract)," in *Proc. 31st Annu. ACM Symp. Theory Comput. (STOC)*, 1999, p. 4.
[54] L. Trevisan, "Some applications of coding theory in computational complexity," *CoRR*, vol. cs.CC/0409044, 2004. [Online]. Available: http://arxiv.org/abs/cs.CC/0409044
[55] S. Yekhanin, "Towards 3-query locally decodable codes of subexponential length," *J. ACM*, vol. 55, no. 1, pp. 1–16, Feb. 2008.
[56] S. Yekhanin, "Locally decodable codes," *Found. Trends Theor. Comput. Sci.*, vol. 6, no. 3, pp. 139–255, 2011.

Jeremiah Blocki is currently an Assistant Professor with the Department of Computer Science, Purdue University. His primary research areas include cryptography, data-privacy, and security with a special focus on password authentication and memory hard functions. He was a recipient of the NSF CAREER Award in 2021.

Venkata Gandikota is currently an Assistant Professor with the Department of Electrical Engineering and Computer Science, Syracuse University. Previously, he was a Post-Doctoral Researcher with the University of Massachusetts at Amherst and Johns Hopkins University. His research focuses on coding theory and foundational problems in machine learning.

Elena Grigorescu is currently an Associate Professor with the Department of Computer Science, Purdue University. She is also interested in sublinear models of computations, recovering data affected by noise, and limits of computation in a variety of models.

Samson Zhou received the B.S. and M.Eng. degrees from MIT in 2010 and 2011, respectively, and the Ph.D. degree from Purdue University. He spent one year as a Post-Doctoral Researcher with Indiana University. He is currently a Post-Doctoral Researcher with Carnegie Mellon University. His research focuses on the theoretical foundations of data science, including sublinear algorithms with an emphasis on streaming algorithms, machine learning, and numerical linear algebra.