Locally Decodable/Correctable Codes for Insertions and Deletions

Alexander R. Block
Purdue University, West Lafayette, IN, USA
block9@purdue.edu

Jeremiah Blocki
Purdue University, West Lafayette, IN, USA
jblocki@purdue.edu

Elena Grigorescu
Purdue University, West Lafayette, IN, USA
elena-g@purdue.edu

Shubhang Kulkarni[1]
University of Illinois Urbana-Champaign, IL, USA
smkulka2@illinois.edu

Minshen Zhu
Purdue University, West Lafayette, IN, USA
zhu628@purdue.edu

Abstract

Recent efforts in coding theory have focused on building codes for insertions and deletions, called insdel codes, with optimal trade-offs between their redundancy and their error-correction capabilities, as well as *efficient* encoding and decoding algorithms.

In many applications, polynomial running time may still be prohibitively expensive, which has motivated the study of codes with *super-efficient* decoding algorithms. These have led to the well-studied notions of Locally Decodable Codes (LDCs) and Locally Correctable Codes (LCCs). Inspired by these notions, Ostrovsky and Paskin-Cherniavsky (Information Theoretic Security, 2015) generalized Hamming LDCs to insertions and deletions. To the best of our knowledge, these are the only known results that study the analogues of Hamming LDCs in channels performing insertions and deletions.

Here we continue the study of insdel codes that admit local algorithms. Specifically, we reprove the results of Ostrovsky and Paskin-Cherniavsky for insdel LDCs using a different set of techniques. We also observe that the techniques extend to constructions of LCCs. Specifically, we obtain insdel LDCs and LCCs from their Hamming LDCs and LCCs analogues, respectively. The rate and error-correction capability blow up only by a constant factor, while the query complexity blows up by a poly log factor in the block length.

Since insdel locally decodable/correctble codes are scarcely studied in the literature, we believe our results and techniques may lead to further research. In particular, we conjecture that constant-query insdel LDCs/LCCs do not exist.

2012 ACM Subject Classification Theory of computation → Error-correcting codes

Keywords and phrases Locally decodable/correctable codes, insert-delete channel

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2020.16

Funding *Alexander R. Block*: Supported by NSF CCF-1910659.
Jeremiah Blocki: Supported by NSF CCF-1910659, CNS-1755708, CNS-1704587 and CNS-1931443.
Elena Grigorescu: Supported by NSF CCF-1910659 and NSF CCF-1910411.
Minshen Zhu: Supported by NSF CCF-1910659.

[1] Work done while at Purdue University, USA.

1 Introduction

Building error-correcting codes that can recover from insertions and deletions (a.k.a. "insdel codes") has been a central theme in recent advances in coding theory [29, 24, 15, 18, 13, 12, 20, 19, 3, 14, 17, 16, 30, 11]. Insdel codes are generalizations of Hamming codes, in which the corruptions may be viewed as deleting symbols and then inserting other symbols at the deleted locations.

An insdel code is described by an encoding function $E : \Sigma^k \to \Sigma^n$, which encodes every message of length k into a codeword of block length n. The rate of the code is the ratio $\frac{k}{n}$. Classically, a decoding function $D : \Sigma^* \to \Sigma^k$ takes as input a string w obtained from some $E(m)$ after δn insertions and deletions and satisfies $D(w) = m$. A fundamental research direction is building codes with high communication rate $\frac{k}{n}$, that are robust against a large δ fraction of insertions and deletions, which also admit *efficient* encoding and decoding algorithms. It is only recently that efficient insdel codes with asymptotically good rate and error-correction parameters have been well-understood [17, 19, 16, 30, 11].

In modern applications, polynomial-time decoding may still be prohibitively expensive when working with large data, and instead *super-efficient* codes are even more desirable. Such codes admit very fast decoding algorithms that query only few locations into the received word to recover portions of the data. Ostrovsky and Paskin-Cherniavsky [33] defined the notion of Locally Decodable Insdel Codes,[2] inspired by the notion of Locally Decodable Codes (LDCs) for Hamming errors [22, 36]. A code defined by an encoding $E : \Sigma^k \to \Sigma^n$ is a q-query Locally Decodable Insdel Code (Insdel LDC) if there exists a randomized algorithm $\mathcal{D}$, such that: (1) for each $i \in [k]$ and message $m \in \Sigma^k$, $\mathcal{D}$ can probabilistically recover m_i, given query access to a word $w \in \Sigma^*$, which was obtained from $E(m)$ corrupted by δ fraction of insertions and deletions; and (2) $\mathcal{D}$ makes only q queries into w. The number of queries q is called the *locality* of the code.

The rate, error-correcting capability, and locality of the code are opposing design features, and optimizing all of them at the same time is impossible. For example, every 2-query LDCs for Hamming errors must have vanishing rate [23]. While progress in understanding these trade-offs for Hamming errors has spanned several decades [23, 38, 39, 6, 7, 26] (see surveys by Yekhanin [39] and by Kopparty and Saraf [27]), in contrast, the literature on the same trade-offs for the more general insdel codes is scarce. Namely, besides the results of [33], to the best of our knowledge, only Haeupler and Shahrasbi [19] consider the notion of locality in building synchronization strings, which are important components of optimal insdel codes.

The results of [33] provide a direct reduction from classical Hamming error LDCs to insdel LDCs, which preserves the rate of the code and error-correction capabilities up to constant factors, and whose locality grows only by a polylogarithmic factor in the block length.

In this paper we revisit the results of Ostrovsky and Paskin-Cherniavsky [33] and provide an alternate proof, using different combinatorial techniques. We also observe that these results extend to building Locally Correctable Insdel Codes (Insdel LCCs) from Locally Correctable Codes (LCCs) for Hamming errors. LCCs are a variant of LDCs, in which the decoder is tasked to locally correct every entry of the encoded message, namely $E(m)_i$, instead of the entries of the message itself. If the message m is part of the encoding $E(m)$, then an LCC is also an LDC. In particular, all linear LCCs (i.e, whose codewords form a vector space) are also LDCs.

[2] In [33], they are named *Locally Decodable Codes for Edit Distance*.

▶ **Theorem 1.** *If there exist q-query LDCs/LCCs with encoding $E : \Sigma^k \to \Sigma^n$, that can correct from δ-fraction of Hamming errors, then there exist binary $q \cdot \mathrm{polylog}(n)$-query Insdel LDCs/LCCs with codeword length $\Theta(n \log |\Sigma|)$, that can correct from $\Theta(\delta)$-fraction of insertions and deletions.*

We emphasize that the resulting LDC/LCC of Theorem 1 is a *binary* code, even if the input LDC/LCC is over some higher alphabet Σ.

Classical constructions of LDCs/LCCs for Hamming errors fall into three query-complexity regimes. In the constant-query regime, the best known results are based on matching-vectors codes, and give encoding that map k symbols into $\exp(\exp(\sqrt{\log k \log\log k}))$ symbols [38, 6, 7]. Since the best lower bounds are only quadratic [37], for all we know so far, it is possible that there exist constant-query complexity LDCs with polynomial block length. In the polylog k-query regime, Reed-Muller codes are examples of $\log^c k$-query LDCs/LCCs of block length $k^{1+\frac{1}{c-1}+o(1)}$ for some $c > 0$ (e.g., see [39]). Finally, there exist sub-polynomial (but super logarithmic)-query complexity LDCs/LCCs with constant rate [26]. These relatively recent developments improved upon the previous constant rate codes in the n^ϵ-query regime achieved by Reed-Muller codes, and later by more efficient constructions (e.g. [28]).

Given that our reduction achieves polylog n-query complexity blow-up, the results above in conjunction with Theorem 1 give us the following asymptotic results.

▶ **Corollary 2.** *There exist* $\mathrm{polylog}(k)$*-query Insdel LDCs/LCCs encoding k symbols into $o(k^2)$ symbols, that can correct a constant fraction of insertions and deletions.*

▶ **Corollary 3.** *There exist $(\log k)^{O(\log\log k)}$-query Insdel LDCs/LCCs with constant rate, that can correct from a constant fraction of insertions and deletions.*

Our results, similarly to those in [33], do not have implications in the constant-query regime. We conjecture that there do not exist constant-query LDCs/LCCs, regardless of their rate. Since achieving locality against insertions and deletions appears to be a difficult task, and the area is in its infancy, we believe our results and techniques may motivate further research.

1.1 Overview of Techniques

Searching in a Nearly Sorted Array. To build intuition for our local decoding algorithm we consider the following simpler problem: We are given a nearly sorted array A of n distinct elements. By nearly sorted we mean that there is another sorted array A' such that $A'[i] = A[i]$ on all but n' indices. Given an input x we would like to quickly find x in the original array. In the worst case this would require time at least $\Omega(n')$ so we relax the requirement that we always find x to say that there are at most cn' items that we will fail to find for some constant $c > 0$.

To design our noisy binary search algorithm that meets the requirement we borrow a notion of local goodness used in the design and analysis of depth-robust graphs a combinatorial object that has found many applications in cryptography [8, 1, 2]. In particular, fixing A and A' (sorted) we say that an index j is corrupted if $A[j] \neq A'[j]$. We say that an index i is θ-locally good if for any $r \geq 0$ at most θ fraction of the indices $j \in [i, \ldots, i+r]$ are corrupted and at most θ fraction of the indices in $[i-r, i]$ are corrupted. If at most n' indices are corrupted then one can prove that at least $n - 2n'/\theta$ indices are θ-locally good [8].

As long as the constant θ is suitably small we can design an efficient randomize search procedure which (whp) will correctly locate x whenever $x = A[i]$ provided that the unknown index i is θ-locally good. Intuitively, suppose we have already narrowed down our search to the smaller range $I = [i_0, i_1]$. The rank of $x = A[i]$ in $A'[i_0], \ldots, A'[i_1]$ is exactly $i - i_0 + 1$ since

$A[i]$ is uncorrupted and the rank of x in $A[i_0], \ldots, A[i_1]$ can change by at most $\pm\theta(i - i_0 + 1)$ – at most $\theta(i_1 - i_0 + 1)$ indices $j' \in [i_0, i_1]$ can be corrupted since $i \in [i_0, i_1]$ is θ-locally good. Now suppose that we sample $t = \text{polylog}(n)$ indices $j_1, \ldots, j_t \in [i_0, i_1]$ and select the median y_{med} of $A[j_1], \ldots, A[j_t]$. With high probability the rank r of y_{med} in $A[j_1], \ldots, A[j_t]$ will be close to $(i_1 - i_0 + 1)/2$ i.e., $|r - (i_1 - i_0 + 1)/2| \leq \delta(i_1 - i_0 + 1)$ for some arbitrarily constant δ which may depend on the number of samples t. Thus, for suitable constants θ and δ whenever $x > y_{med}$ (resp. $x < y_{med}$) we can safely conclude that $i > i_0 + (i_1 - i_0 + 1)/8$ (resp. $i < i_1 - (i_1 - i_0 + 1)/8$) and search in the smaller interval $I' = [i_0 + (i_1 - i_0 + 1)/8, i_1]$ (resp. $I' = [i_0, i_1 - (i_1 - i_0 + 1)/8]$). In both cases the size of the search space is reduced by a constant multiplicative factor so the procedure will terminate after $O(\log n)$ rounds making $O(t \log n)$ queries. At its core our local decoding algorithm relies on a very similar idea.

Encoding. Our encoder builds off of the known techniques of concatenation codes. First, a message x is encoded via the outer code to obtain some (intermediate) encoding y. We then partition y into some number k blocks $y = y_1 \circ \cdots \circ y_k$ and append each block y_i with index i to obtain $y_i \circ i$. Each $y_i \circ i$ is then encoded with the inner encoder to obtain some d_i. Then each d_i is prepended and appended with a run of 0s (i.e., the buffers), to obtain c_i. The encoder then outputs $c = c_1 \circ \cdots \circ c_k$ as the final codeword. For our inner encoder, we in fact use the Schulman-Zuckerman (SZ) [34] edit distance code.

Decoding. Given oracle access to some corrupted codeword c', on input index i, the decoder simulates the outer decoder and must answer the outer decoder oracle queries. The decoder uses the inner decoder to answer these queries. However, there are two major challenges: (1) Unlike the Hamming-type errors, even only a few insertions and deletions makes it hard for the decoder to know where to probe; and (2) The boundaries between blocks can be ambiguous in the presence of insdel errors. We overcome these challenges via a variant of binary search, which we name NoisyBinarySearch, together with a buffer detection algorithm, and make use of a block decomposition to facilitate the analysis.

Analysis. The analyses of the binary search and the buffer detection algorithms are based on the notion of "good blocks" and "locally good blocks", which are natural extensions of the notion of θ-locally good discussed above. Recall that our encoder outputs a final codeword that is a concatenation of k smaller codeword "blocks"; namely $\mathsf{Enc}(x) = c_1 \circ \cdots \circ c_k$. Suppose c' is the corrupted codeword obtained by corrupting c with δ-fraction of insertion-deletion errors, and suppose we have a method of partitioning c' into k blocks $c'_1 \circ \cdots \circ c'_k$. Then we say that block c'_j is a γ-good block if it is within γ-fractional edit distance to the *uncorrupted* block c_j. Moreover, c'_j is (θ, γ)-locally good if at least $(1 - \theta)$ fraction of the blocks in every neighborhood around c'_j are γ-good and if the total number of corruptions in every neighborhood is bounded. Both notions of good and locally good blocks are necessary to the success of our binary search algorithm NoisyBinarySearch.

The goal of NoisyBinarySearch is to locate a block with a given index j, and the idea is to decode the corrupted codeword at random positions to get a list of decoded indices (recall that the index of each block is appended to it). Since a large fraction of blocks are γ-good blocks, the sampled indices induce a new search interval for the next iteration. In order to apply this argument recursively, we need that the error density of the search interval does not increase in each iteration. Locally good blocks provide precisely this property.

Comparison with the techniques of [33]. The Insdel LDC construction of [33] also uses Schulman-Zuckerman (SZ) [34] codes, except it opens them up and directly uses the inefficient greedy inner codes used for the final efficient SZ codes themselves. In our case, we observe that the efficiently decodable codes of [34] have the additional property described in Lemma 7, which states that small blocks have large weight. This observation implies a running time that is polynomial in the query complexity of the final codes, since it helps make the buffer-finding algorithms local. The analysis of [33] also uses a binary search component, but our analysis and their analysis differ significantly.

1.2 Related work

The study of codes for insertions and deletions was initiated by Levenstein [29] in the mid 60's. Since then there has been a large body of works concerning insdel codes, and we refer the reader to the excellent surveys of [35, 31, 32]. In particular, random codes with positive rate correcting from a large fraction of deletions were studied in [24, 15]. Efficiently encodable/decodable codes, with constant rate, and that can withstand a constant fraction of insertion and deletions were extensively studied in [34, 15, 18, 19, 19, 14, 3, 11]. A recent area of interest is building "list-decodable" insdel codes, that can withstand a larger fraction of insertions and deletions, while outputting a small list of potential codewords [20, 11, 30].

In [19], Haeupler and Shahrasbi construct explicit synchronization strings which can be locally decoded, in the sense that each index of the string can be computed using values located at only a small number of other indices. Synchronization strings are powerful combinatorial objects that can be used to index elements in constructions of insdel codes. These explicit and locally decodable synchronizations strings were then used to imply near linear time interactive coding scheme for insdel errors.

There are various other notions of "noisy search" that have been studied in the literature. Dhagat, Gacs, and Winkler [5] consider a noisy version of the game "Twenty Questions". In this problem, an algorithm searches an array for some element x, and a bounded number of incorrect answers can be given to the algorithm queries, and the goal is to minimize the number of queries made by an algorithm. Feige et al. [9] study the depth of *noisy decision trees*: decision trees where each node gives the incorrect answer with some constant probability, and moreover each node success or failure is independent. Karp and Kleinberg [21] study noisy binary search where direct comparison between elements is not possible; instead, each element has an associated biased coin. Given n coins with probabilities $p_1 \leq \ldots \leq p_n$, target value $\tau \in [0, 1]$, and error ϵ, the goal is to design an algorithm which, with high probability, finds index i such that the intervals $[p_i, p_{i+1}]$ and $[\tau - \epsilon, \tau + \epsilon]$ intersect. Braverman and Mossel [4], Klein et al. [25] and Geissmann et al. [10] study noisy sorting in the presence of recurrent random errors: when an element is first queried, it has some (independent) probability of returning the incorrect answer, and all subsequent queries to this element are fixed to this answer. We note that each of the above notions of "noisy search" are different from each other and, in particular, different from our noisy search.

1.3 Organization

We begin with some general preliminaries in Section 2. In Section 3 we present the formal encoder and decoder. In Section 4 we define block decomposition which play an important role in our analysis. In Section 5, Section 6, and Section 7 we prove correctness of our local decoding algorithm in a top-down fashion. All missing proofs are deferred to the full-version.

2 Preliminaries

We now define some notation used throughout this work. We use $[n]$ to represent the set $\{1, 2, \ldots, n\}$. More generally, for integers $a < b$, we let $[a, b]$ denote the set $\{a, a+1, \ldots, b\}$. All logarithms will be base 2 unless specified otherwise. We denote $x \circ y$ as the concatenation of string x with string y. For any $x \in \Sigma^n$, $x[i]$ denotes the i^{th} coordinate of x. A function $\mathsf{negl}(n)$ is said to be *negligible* in n if $\mathsf{negl}(n) = o\left(n^{-d}\right)$ for any $d \in \mathbb{N}$. For any $x, y \in \Sigma^n$, $\mathsf{HAM}(x, y) = |\{i : x[i] \neq y[i]\}|$ denotes the hamming distance between x and y. Furthermore, $\mathsf{ED}(x, y)$ denotes the edit distance between x and y i.e. the minimum number of symbol insertions and deletions to transform x into y. For any string $x \in \Sigma^*$ with finite length, we denote $|x|$ as the length of x. The fractional Hamming distance (resp., edit distance) is $\mathsf{HAM}(x, y)/|x|$ (resp., $\mathsf{ED}(x, y)/(2|x|)$).

▶ **Definition 4** (Locally Decodable Codes for Hamming and Insdel errors)**.** *A code with encoding function $E : \Sigma_M^k \to \Sigma_C^n$ is a (q, δ, ϵ)-*Locally Decodable Code *(LDC) if there exists a randomized decoder $\mathcal{D}$, such that for every message $m \in \Sigma_M^k$ and index $i \in [k]$, and for every $w \in \Sigma_C^*$ such that $\mathrm{dist}(w, E(m)) \leq \delta$ the decoder makes at most q queries to w and outputs m_i with probability $\frac{1}{2} + \epsilon$; when* dist *is the fractional Hamming distance then this is a Hamming LDC; when* dist *is the fractional edit distance then this is an Insdel LDC. We also say that the code is binary if $\Sigma_C = \{0, 1\}$.*

▶ **Definition 5** (Locally Correctable Codes for Hamming and Insdel errors)**.** *A code with encoding function $E : \Sigma_M^k \to \Sigma_C^n$ is a (q, δ, ϵ)-*Locally Correctable Code *(LCC) if there exists a randomized decoder $\mathcal{D}$, such that for every message $m \in \Sigma_M^k$ and index $i \in [n]$, and for every $w \in \Sigma_C^*$ such that $\mathrm{dist}(w, E(m)) \leq \delta$ the decoder makes at most q queries to w and outputs $E(m)_i$ with probability $\frac{1}{2} + \epsilon$; when* dist *is the fractional Hamming distance then this is a Hamming error LCC; when* dist *is the fractional edit distance then this is an Insdel LCC. We also say that the code is binary if $\Sigma_C = \{0, 1\}$.*

Our construction, like most insdel codes in the literature, is obtained via adaptations of the simple but powerful operation of code concatenation. If C_{out} is an "outer" code over alphabet Σ_{out} with encoding function $E_{out} : \Sigma_{out}^k \to \Sigma_{out}^n$, and C_{in} is an "inner" code over alphabet Σ_{in} with encoding function $E_{in} : \Sigma_{out} \to \Sigma_{in}^p$, then the concatenated code $C_{out} \bullet C_{in}$ is the code whose codewords lie in Σ_{in}^{np}, obtained by first applying E_{out} to the message, and then applying E_{in} to each symbol of the resulting outer codeword.

3 Insdel LDCs/LCCs from Hamming LDCs/LCCs

We give our main construction of Insdel LDCs/LCCs from Hamming LDCs/LCCs. Our construction can be viewed as a procedure which, given outer codes C_{out} and binary inner codes C_{in} satisfying certain properties, produces binary codes $C(C_{out}, C_{in})$. This is formulated in the following theorem, which implies Theorem 1.

▶ **Theorem 6.** *Let C_{out} and C_{in} be codes such that*

- *C_{out} defined by $\mathsf{Enc}_{out} : \Sigma^k \to \Sigma^m$ is an a $(\ell_{out}, \delta_{out}, \epsilon_{out})$-LDC/LCC (for Hamming errors).*
- *C_{in} is family of binary polynomial-time encodable/decodable codes with rate $1/\beta_{in}$ capable of correcting δ_{in} fraction of insdel errors. In addition, there are constants $\alpha_1, \alpha_2 \in (0, 1)$ such that for any codeword c of C_{in}, any substring of c with length $\geq \alpha_1|c|$ has fractional Hamming weight at least α_2.*

Then $C(C_{out}, C_{in})$ *is a binary* $\left(\ell_{out} \cdot O\left(\log^4 n'\right), \Omega(\delta_{out}\delta_{in}), \epsilon - \mathsf{negl}(n')\right)$*-Insdel LDC, or a binary* $\left(\ell_{out} \cdot O\left(\log^5 n'\right), \Omega(\delta_{out}\delta_{in}), \epsilon - \mathsf{negl}(n')\right)$*-Insdel LCC, respectively. Here the codewords of* C *have length* $n = \beta m$ *where* $\beta = O\left(\beta_{in} \log |\Sigma|\right)$*, and* n' *denotes the length of received word.*

For the inner code, we make use of the following efficient code constructed by Schulman-Zuckerman [34].

▶ **Lemma 7** (SZ-code [34]). *There exist constants* $\beta_{in} \geq 1$*,* $\delta_{in} > 0$*, such that for large enough values of* $t > 0$*, there exists a code* $SZ(t) = (\mathsf{Enc}, \mathsf{Dec})$ *where* $\mathsf{Enc} : \{0,1\}^t \to \{0,1\}^{\beta_{in} t}$ *and* $\mathsf{Dec} : \{0,1\}^{\beta_{in} t} \to \{0,1\}^t \cup \{\perp\}$ *capable of correcting* δ_{in} *fraction of insdel errors, having the following properties:*

1. Enc *and* Dec *run in time poly*(t)*;*
2. *For all* $x \in \{0,1\}^t$*, every interval of length* $2 \log t$ *of* $Enc(x)$ *has fractional Hamming weight* $\geq 2/5$*.*

We formally complete the proof of correctness of Theorem 6 in Section 5. We only prove the correctness of the LDC decoder since it is cleaner and captures the general strategy of the LCC decoder as well. We dedicate the remainder of this section to outlining the construction of the encoding and decoding algorithms.

3.1 Encoding and Decoding Algorithms

In our construction of $C(C_{out}, C_{in})$, we denote the specific code of Lemma 7 as our inner code $C_{in} = (\mathsf{Enc}_{in}, \mathsf{Dec}_{in})$. For our purpose, it is convenient to view the message as a pair in $[m] \times \Sigma^{\log m}$. The encoding function $\mathsf{Enc}_{in} \colon [m] \times \Sigma^{\log m} \to \{0,1\}^{\beta_{in}\left(1+\log|\Sigma|\right)\log m}$ maps a $\log m$-long string over alphabet Σ appended with an index from set $[m]$ – i.e. a (padded) message of bit-length $\left(1 + \log |\Sigma|\right) \log m$ – to a binary string of length $\beta_{in}\left(1 + \log |\Sigma|\right) \log m$. The inner decoder Dec_{in} on input y' returns x if $\mathsf{ED}\left(y', y\right) \leq \delta_{in} \cdot 2|y|$ where $y = \mathsf{Enc}_{in}(x)$. The information rate of this code is $R_{in} = 1/\beta_{in}$.

We describe our final encoding and decoding algorithms next.

The Encoder (Enc). Given an input string $x \in \Sigma^k$ and outer code $C_{out} = (\mathsf{Enc}_{out}, \mathsf{Dec}_{out})$, our final encoder Enc does the following: (1) Computes the outer encoding of x as $s = \mathsf{Enc}_{out}(x)$; (2) For each $i \in [m/\log m]$, groups $\log m$ symbols $s[(i-1)\log m] \cdots s[i \log m - 1]$ into a single block $b_i \in \Sigma^{\log m}$; (3) For each $i \in [m/\log m]$, computes the i^{th} block of the inner encoding as $Y^{(i)} = \mathsf{Enc}_{in}(i \circ b_i)$ – i.e. computes inner encoding of the i^{th} block concatenated with the (padded) index i; (4) For some constant $\alpha \in (0,1)$ (to be decided), appends a $\alpha \log m$-long buffer of zeros before and after each block; and (5) Outputs the concatenation of the buffered blocks (in indexed order) as the final codeword $c = \mathsf{Enc}(x) \in \{0,1\}^n$, where

$$c = \left(0^{\alpha \log m} \circ Y^{(1)} \circ 0^{\alpha \log m}\right) \circ \cdots \circ \left(0^{\alpha \log m} \circ Y^{(m/\log m)} \circ 0^{\alpha \log m}\right). \tag{1}$$

Denoting $\beta = 2\alpha + \beta_{in}\left(1 + \log |\Sigma|\right)$, the length of c is

$$n = \left(2\alpha \log m + \beta_{in}\left(1 + \log |\Sigma|\right) \log m\right) \cdot \frac{m}{\log m} = \beta m.$$

The LDC Decoder (Dec). We start off by describing the high-level overview of our decoder Dec and discuss the challenges and solutions behind its design. As defined in Equation (1), our encoder Enc, on input $x \in \Sigma^k$, outputs a codeword $c = c_1 \circ \cdots \circ c_d \in \{0,1\}^n$ where $d = m/\log m$. The decoder setting is as follows: on input $i \in [k]$ and query access to the corrupted codeword $c' \in \{0,1\}^{n'}$ such that $\mathsf{ED}(c, c') \leq 2n\delta$, our final decoder Dec needs to output the message symbol $x[i]$ with high probability. Notice that if Dec had access to the original codeword $s = \mathsf{Enc}_{out}(x)$, then Dec could simply run $\mathsf{Dec}_{out}(i)$ while supplying it with oracle access to this codeword s. This naturally motivates the following decoding strategy – simulate Dec_{out}'s oracle access to the codeword s via answering Dec_{out}'s queries by decoding the appropriate bits using Dec_{in}. We give a detailed description of this strategy next.

Let $Q_i = \{q_1, \ldots, q_{\ell_{out}}\} \subset [m]$ be a set of indices which $\mathsf{Dec}_{out}(i)$ makes queries to. [3] We observe that if our decoder had oracle access to the uncorrupted codeword c, then answering these queries would be simple:

1. For each $q \in Q_i$, let $b_j = s[(j-1)\log m] \cdots s[j \log m - 1]$ be the block which contains $s[q]$. In particular, $q = (j-1)\log m + r_j$ for some $r_j \in [0, \log m - 1]$,
2. Obtain block c_j by querying oracle c and obtain $Y^{(j)}$ by removing the buffers from c_j,
3. Obtain $j \circ b_j$ by running $\mathsf{Dec}_{in}(Y^{(j)})$, then return $s[q] = b_j[r_j]$ to Dec_{out}.

In fact, it suffices for Dec_{out}'s queries to be answered with symbols consistent with any string s' such that $\mathsf{HAM}(s, s') \leq \delta_{out} m$ – for then, the correctness of the output would follow from the correctness of Dec_{out}. We are still going to carry out the strategy mentioned above, except that now we are given a corrupted codeword c'.

For the purposes of analysis, we first define the notion of a *block decomposition* of the corrupted codeword c'. Informally, a block decomposition is simply a partitioning of c' into contiguous blocks. Our first requirement for successful decomposition is that there must exist a block decomposition $c' = c'_1 \circ \cdots \circ c'_d$ that is "not too different" from the original decomposition $c = c_1 \circ \cdots \circ c_d$.[4] In particular, we require that $\sum_j \mathsf{ED}(c'_j, c_j) \leq 2n\delta$. Proposition 9 guarantees this property. Next, we define the notion of γ-*good* (see Definition 10). The idea here is that if a block c'_j is γ-good (for appropriate γ), then we can run Dec_{in} on c'_j and obtain $j \circ b_j$. As the total number of errors is bounded, it is easy to see that all but a small fraction of blocks are γ-good (Lemma 14). At this point, we are essentially done if we can decode c'_j for any given γ-good block j.

An immediate challenge we are facing is that of *locating* a specific γ-good block c'_j, while maintaining overall locality. The presence of insertions and deletions may result in uneven block lengths, making the task of locating a specific block non-trivial. However, since the γ-good blocks, which make up majority of the blocks, must be in the correct relative order, it is conceivable to perform a *binary search* type algorithm over the blocks of c' to find block c'_j. The idea is to maintain a search interval and iteratively reduce its size. In each iteration, the algorithm samples a small number of blocks and obtains their (appended) indices. As the vast majority of blocks are γ-good, these indices will guide the binary search algorithm in narrowing down the search interval. There is one problem with this argument. The density of γ-good blocks may go down as the search interval gets smaller. In fact, it is impossible to *locally* locate a block c'_j surrounded by many bad blocks even if it is γ-good. This is where the notion of (θ, γ)-locally good (see Definition 12) helps us: if a block c'_j is (θ, γ)-locally

[3] This is for ease of presentation. Our construction also supports adaptive queries.

[4] We note that we do not need to know this decomposition explicitly, and that its existence is sufficient for our analysis.

good, then $(1-\theta)$-fraction of blocks in every neighborhood around c'_j are γ-good, and every neighborhood around c'_j has a bounded number of errors. Therefore, as long as the search interval contains a locally good block, we can lower bound the density of γ-good blocks.

Our *noisy binary search algorithm* essentially implements this idea. The algorithm on input block index j, attempts to find block j. If block j is (θ,γ)-locally good, then we can guarantee that our noisy binary search algorithm will find j except with negligible probability (see Theorem 18). Thus it is desirable that the number of (θ,γ)-locally good blocks is large; if this is so, the noisy binary search is effectively providing oracle access to a string s' which is close to s in Hamming distance, and thus the outer decoder is able to decode $x[i]$ with high probability. Lemma 15 exactly guarantees this property.

The discussion above, however, requires knowing the boundaries of each block c'_j. As the decoder is oblivious to the block decomposition, it is going to work with *approximate boundaries* which can be found locally by a *buffer search algorithm* described as follows. Recall that by construction c_j consists of $Y^{(j)}$ surrounded by buffers of $(\alpha \log m)$-length 0-runs. So to find $Y^{(j)}$, it suffices to find the buffers surrounding $Y^{(j)}$. Our buffer search algorithm can be viewed as a "local variant" of the buffer search algorithm of Schulman and Zuckerman [34]. It is designed to find approximate buffers surrounding a block c'_j if it is γ-good. Then the string in between two buffers is identified as a corrupted codeword and is decoded to $j \circ b_j$. The success of the algorithm depends on γ-goodness of the block being searched and requires that any substring of a codeword from C_{in} has "large enough" Hamming weight. In fact, our inner code given by Lemma 7 gives us this exact guarantee. All together, this enables the noisy binary search algorithm to use the buffer finding algorithm to search for a block c'_j.

We formalize the decoder outlined above. On input $i \in [k]$, Dec simulates $\mathsf{Dec}_{out}(i)$ and answers its queries. Whenever $\mathsf{Dec}_{out}(i)$ queries an index $j \in [m]$, Dec expresses $j = (p-1)\log m + r_j$ for $p \in [m/\log m]$ and $r_j \in [0, \log m - 1]$, and runs $\mathsf{NoisyBinarySearch}(c', p)$ (which calls the BUFF-FIND algorithm) to obtain a string $b' \in \Sigma^{\log m}$ (or $\perp$). Then it feeds the r_j-th symbol of b' (or $\perp$) to $\mathsf{Dec}_{out}(i)$. Finally, Dec returns the output of $\mathsf{Dec}_{out}(i)$.

The LCC Decoder (Dec). Similar to the LDC decoder, our LCC decoder Dec does the following: We denote $B = 2\alpha \log m + \beta_{in}\left(1 + \log|\Sigma|\right)\log m$. On input $j \in [n]$, Dec first expresses $j = (p-1)B + r_j$ for some $p \in [m/\log m]$ and $0 \le r_j < B$, and checks whether j is inside a buffer. Specifically, if $r_j \in [0, \log m) \cup [B - \log m, B)$, it outputs 0. Otherwise, it simulates $\mathsf{Dec}_{out}((p-1)\log m + r)$ for each $0 \le r < \log m$, and answers their queries. Whenever Dec_{out} queries $i \in [m]$, Dec expresses $i = (b-1)\log m + r_i$ for some $b \in [m/\log m]$ and $0 \le r_i < \log m$, and runs $\mathsf{NoisyBinarySearch}(c', b)$ to obtain a string $S \in \Sigma^{\log m}$ (or $\perp$), and answers the query with S_{r_i} (or $\perp$). Finally, denoting by s_r the output of $\mathsf{Dec}_{out}((p-1)\log m + r)$, Dec returns the $(r_j - \log m + 1)$-th bit of $\mathsf{Enc}_{in}\left(p \circ s_0 s_1 \dots s_{\log m - 1}\right)$.

Efficiency. We note that the efficiency of our compiler depends on the constituent inner and outer codes. Let $T(\mathsf{Enc}_{in}, l)$, $T(\mathsf{Enc}_{out}, l)$, $T(\mathsf{Enc}, l)$ denote the run-times of the inner, outer and final encoders respectively on inputs of length l. Similarly, let $T(\mathsf{Dec}_{in}, l)$, $T(\mathsf{Dec}_{out}, l)$, $T(\mathsf{Dec}, l)$ denote the run-times of the inner, outer and final decoders respectively having query access to corrupted codewords of length l. Then we have following run-time relations

$$T(\mathsf{Enc}, k) = T(\mathsf{Enc}_{out}, k) + O(m/\log m) \cdot T(\mathsf{Enc}_{in}, \log|\Sigma| \cdot \log m + \log m),$$
$$T(\mathsf{Dec}, n') = T(\mathsf{Dec}_{out}, m) + \ell_{out} \cdot O\left(\log^3 n'\right) \cdot T(\mathsf{Dec}_{in}, \beta \log m).$$

Here, n' is the length of the corrupted codeword.

4 Block Decomposition of Corrupted Codewords

The analysis of our decoding procedure relies on a so-called buffer finding algorithm and a noisy binary search algorithm. To analyze these algorithms, we introduce the notion of a *block decomposition* for (corrupted) codewords, as well as what it means for a block to be *(locally) good*.

For convenience, we now fix some notations for the rest of the paper. We fix an arbitrary message $x \in \Sigma^k$. We use $s = \mathsf{Enc}_{out}(x) \in \Sigma^m$ for the encoding of x by the outer encoder. Let $\tau = \log m$ be the length of each block and $d = m/\log m$ be the number of blocks. For $i \in [d]$, we let $b_i \in \Sigma^\tau$ denote the i-th block $s[(i-1)\tau]s[(i-1)\tau+1]\dots s[i\tau-1]$, and let $Y^{(i)}$ denote the encoding $\mathsf{Enc}_{in}(i \circ b_i)$. Recall that $\alpha\tau$ is the length of the appended buffers for some $\alpha \in (0,1)$, and the parameter $\beta = 2\alpha + \beta_{in}(1+\log|\Sigma|)$. Thus $|Y^{(i)}| = (\beta - 2\alpha)\tau$. The final encoding is given by

$$c = \tilde{Y}^{(1)} \circ \tilde{Y}^{(2)} \circ \cdots \circ \tilde{Y}^{(d)},$$

where $\tilde{Y}^{(j)} = 0^{\alpha\tau} \circ Y^{(j)} \circ 0^{\alpha\tau}$ and $|\tilde{Y}^{(j)}| = \beta\tau$. The length of c is $n = d\beta\tau = \beta m$. We let $c' \in \{0,1\}^{n'}$ denote a corrupted codeword satisfying $\mathsf{ED}\left(c, c'\right) \le \delta \cdot 2n$.

A *block decomposition* of a (corrupted) codeword c' is a non-decreasing mapping $\phi\colon [n'] \to [d]$ for $n', d \in \mathbb{Z}^+$. We say a set $I \subseteq [n']$ is an interval if $I = \emptyset$ (i.e., an empty interval) or $I = \{l, l+1, \dots, r-1\}$ for some $1 \le l < r \le n'$, in which case we write $I = [l, r)$. For an interval $I = [l, r)$, we write $c'[I]$ for the substring $c'[l]c'[l+1]\dots c'[r-1]$. Finally, $c[\emptyset]$ stands for the empty string.

We remark that since ϕ is non-decreasing, for every $j \in [d]$ the pre-image $\phi^{-1}(j)$ is an interval. Since ϕ is a total function, it induces a partition of $[n']$ into d intervals $\left\{\phi^{-1}(j)\colon j \in [d]\right\}$. The following definition plays an important role in the analysis.

▶ **Definition 8** (closure intervals). *The* closure of an interval $I = [l, r) \subseteq [n']$ *is defined as* $\cup_{i=l}^{r-1}\phi^{-1}(\phi(i))$. *An interval I is a* closure interval *if the closure of I is itself. Equivalently, every closure interval has the form $\mathcal{I}[a, b] := \bigcup_{j=a}^{b}\phi^{-1}(j)$ for some $a, b \in [d]$.*

▶ **Proposition 9.** *There exists a block decomposition $\phi\colon [n'] \to [d]$ such that*

$$\sum_{j\in[d]} \mathsf{ED}\left(c'[\phi^{-1}(j)],\ \tilde{Y}^{(j)}\right) \le \delta \cdot 2n.$$

We now introduce the notion of *good blocks*. In the following definitions, we also fix an arbitrary block decomposition ϕ of c' enjoying the property guaranteed by Proposition 9.

▶ **Definition 10** (γ-good block). *For $\gamma \in (0,1)$ and $j \in [d]$ we say that block j is* γ-good *if* $\mathsf{ED}(c'[\phi^{-1}(j)], \tilde{Y}^{(j)}) \le \gamma\alpha\tau$. *Otherwise we say that block j is* γ-bad.

▶ **Definition 11** ((θ,γ)-good interval). *We say a closure interval $\mathcal{I}[a, b]$ is (θ, γ)-good if the following hold:*

1. $\sum_{j=a}^{b}\mathsf{ED}\left(c'[\phi^{-1}(j)], \tilde{Y}^{(j)}\right) \le \gamma \cdot (b-a+1)\alpha\tau$.
2. *There are at least $(1-\theta)$-fraction of γ-good blocks among those indexed by $\{a, a+1, \cdots, b\}$.*

▶ **Definition 12** ((θ,γ)-local good block). *For $\theta, \gamma \in (0,1)$ we say that block j is* (θ,γ)-local good *if for every $a, b \in [d]$ such that $a \le j \le b$ the interval $\mathcal{I}[a, b]$ is (θ, γ)-good. Otherwise, block j is (θ, γ)-locally bad.*

Note that in Definition 12, if j is (θ,γ)-locally good, then j is also γ-good by taking $a = b = j$.

▶ **Proposition 13.** *The following bounds hold:*

1. *For any γ-good block j, $(\beta-\alpha\gamma)\tau \le |\phi^{-1}(j)| \le (\beta+\alpha\gamma)\tau$.*
2. *For any (θ,γ)-good interval $\mathcal{I}[a,b]$, $(b-a+1)(\beta-\alpha\gamma)\tau \le \left|\mathcal{I}[a,b]\right| \le (b-a+1)(\beta+\alpha\gamma)\tau$.*

The following lemmas give upper bounds on the number of γ-bad and (θ,γ)-locally bad blocks.

▶ **Lemma 14.** *The total fraction of γ-bad blocks is at most $2\beta\delta/(\gamma\alpha)$.*

▶ **Lemma 15.** *The total fraction of (θ,γ)-local bad blocks is at most $(4/\gamma\alpha)(1+1/\theta)\delta\beta$.*

5 Outer Decoder

At a high level, the our decoding algorithm Dec the outer decoder Dec_{out} and must answer all oracle queries of Dec_{out} by simulating oracle access to some corrupted string s'. Recall that C_{out} with encoding function $\mathsf{Enc}_{out}\colon \Sigma^k \to \Sigma^m$ is a $(\ell_{out},\delta_{out},\epsilon_{out})$-LDC (for Hamming errors). There is a probabilistic decoder Dec_{out} such that for any $i \in [k]$ and string $s' \in \left(\Sigma \cup \{\perp\}\right)^m$ such that $\mathsf{HAM}\left(s',s\right) \le \delta_{out}\cdot m$ for some codeword $s = \mathsf{Enc}_{out}(x)$, we have

$$\Pr\left[\mathsf{Dec}_{out}^{s'}(i) = x[i]\right] \ge \frac{1}{2} + \epsilon_{out}.$$

Additionally, Dec_{out} makes at most ℓ_{out} queries to s'.

In order to run Dec_{out}, we need to simulate oracle access to such a string s'. To do so, we present our noisy binary search algorithm Algorithm 1 in Section 6. For now, we assume Algorithm 1 has the property stated in the following theorem.

▶ **Theorem 16.** *For $j \in [d]$, let $\mathbf{b}_j \in \Sigma^\tau \cup \{\perp\}$ be the random variable denoting the output of Algorithm 1 on input $(c',1,n'+1,j)$. We have*

$$\Pr\left[\Pr_{j\in[d]}\left[\mathbf{b}_j \ne b_j\right] \ge \delta_{out}\right] \le \mathsf{negl}(n'),$$

where the probability is taken over the joint distribution of $\left\{\mathbf{b}_j\colon j\in[d]\right\}$.

We note that $\mathbf{b}_j$'s do not need to be independent, i.e. two runs of Algorithm 1 can be correlated. For example, we can fix the random coin tosses of Algorithm 1 before the first run and reuse them in each call.

6 Noisy Binary Search

We present Algorithm 1 below. As mentioned in Section 5, the binary search algorithm discussed in this section can be viewed as providing the outer decoder with oracle access to some string $s' \in \left(\Sigma\cup\{\perp\}\right)^m$. Namely whenever the outer decoder queries an index $j\in[m]$ which lies in block p, we run Noisy-Binary-Search on $(c',1,n'+1,p)$ and obtain a string $b'_p \in \Sigma^{\log m}$ which contains the desired symbol $s'[j]$. For now, we analyze the query complexity of Algorithm 1.

▶ **Proposition 17.** *Algorithm 1 has query complexity $O\left(\log^4 n'\right)$.*

The following theorem shows that the set of indices which can be correctly returned by Algorithm 1 is captured by the locally good property.

Algorithm 1 Noisy binary search.

Input: An index $j \in [d]$, and oracle access to a codeword $c' \in \{0,1\}^{n'}$.
Output: A string $b \in \Sigma^\tau$ or $\perp$.

$N \leftarrow \Theta(\log^2 n')$
$\rho \leftarrow \min\left\{\frac{1}{4}\cdot\frac{\beta-\gamma}{\beta+\gamma}, 1-\frac{3}{4}\cdot\frac{\beta+\gamma}{\beta-\gamma}\right\}$
$C \leftarrow 36(\beta+\gamma)\tau$
function Noisy-Binary-Search(c', l, r, j)
  if $r-l \le C$ **then**
    $s \leftarrow$ Interval-Decode(l, r, j)
    return s
  end if
  $m_1 \leftarrow (1-\rho)l + \rho r$, $m_2 \leftarrow \rho l + (1-\rho)r$
  for $t \leftarrow 1$ to N **do**
    Randomly sample i from $\{m_1, m_1+1, \ldots, m_2-1\}$
    $j_t \leftarrow$ Block-Decode(i)
  end for
  $\tilde{j} \leftarrow$ median of $j_1, \ldots, j_N$ (ignore j_t if $j_t = \perp$)
  if $j \le \tilde{j}$ **then**
    return Noisy-Binary-Search(c', l, m_2, j)
  else
    return Noisy-Binary-Search(c', m_1, r, j)
  end if
end function

▶ **Theorem 18.** *If $j \in [d]$ is a (θ,γ)-locally-good block, running Algorithm 1 on input $(c', 1, n'+1, j)$ outputs b_j with probability at least $1 - \mathsf{negl}(n')$.*

As the only time Algorithm 1 interacts with c' is when it queries Block-Decode and Interval-Decode, the properties of these two algorithms are going to be essential to our proof. We briefly describe these two subroutines now.

- Block-Decode On input index $i \in [n']$, Block-Decode tries to find the block j that contains i, and attempts to decode the block to $j \circ b_j$. It returns the index j if the decoding was successful, and $\perp$ otherwise.
- Interval-Decode On input $l, r \in [n']$ and $j \in [d]$, Interval-Decode (roughly) runs the buffer search algorithm of Schulman and Zuckerman [34] over the substring $c'[l,r]$ to obtain a set of approximate buffers, and attempts to decode all strings separated by the approximate buffers. It returns b if any string is decoded to $j \circ b$, and $\perp$ otherwise.

For convenience, we will model Block-Decode as a function $\varphi\colon [n'] \to [d] \cup \{\perp\}$, and model Interval-Decode as a function $\psi\colon [n'] \to \Sigma^\tau \cup \{\perp\}$. The following properties of φ and ψ are what allow the proof to go through.

▶ **Theorem 19.** *The functions φ and ψ satisfy the following properties:*

1. *For any γ-good block j we have*
$$\Pr_{i \in \phi^{-1}(j)}\left[\varphi(i) \ne j\right] \le \gamma.$$
2. *Let $[l,r)$ be an interval with closure $\mathcal{I}[L, R-1]$ such that every block $j \in \{L, \ldots, R-1\}$ is γ-good. Then for every block j such that $\phi^{-1}(j) \subseteq [l,r)$, we have $\psi(j,l,r) = b_j$.*

7 Block Decode Algorithm

A key component of the Noisy Binary Search algorithm is the ability to decode γ-good blocks in the corrupted codeword c'. In order to do so, our algorithm will take explicit advantage of the γ-good properties of a block. We present our block decoding algorithm, named Block-Decode, in Algorithm 2.

Algorithm 2 Block-Decode.

Input: An index $i \in [n']$ and oracle access to (corrupted) codeword $c' \in \{0,1\}^{n'}$.
Output: Some string $\mathsf{Dec}(s)$ for a substring s of c', or $\perp$.

function Block-Decode$^{c'}(i)$
  $\mathsf{buff} \leftarrow$ Buff-Find$^{c'}_{\eta}(i)$
  if $\mathsf{buff} == \perp$ **then**
    return $\perp$
  else Parse buff as $(a,b),(a',b')$
    if $b < i < a'$ **then**
      return $\mathsf{Dec}_{in}(c'[b+1, a'-1])$
    end if
  end if
  return $\perp$
end function

Algorithm 3 Buff-Find$_\eta$.

Input: An index $i \in [n']$ and oracle access to (corrupted) codeword $c' \in \{0,1\}^{n'}$.
Output: Two consecutive δ_{b}-approximate buffers $(a,b),(a',b')$, or $\perp$.

function Buff-Find$^{c'}(i)$
  $j_s \leftarrow \max\{1, i-\eta\tau\}$, $j_e \leftarrow \min\{n'-\tau+1, i+\eta\tau\}$
  $\mathsf{buffs} \leftarrow [\,]$
  while $j_s \le j_e$ **do**
    if $\mathsf{ED}(0^\tau, c'[j_s, j_s+\tau-1]) \le \delta_{\mathsf{b}}\alpha\tau$ **then**
      $\mathsf{buffs.append}((j_s, j_s+\tau-1))$
    end if
    $j_s \leftarrow j_s + 1$
  end while
  for all $k \in \{0,1,\ldots,|\mathsf{buffs}|-2\}$ **do**
    $(a,b) \leftarrow \mathsf{buffs}[k]$, $(a',b') \leftarrow \mathsf{buffs}[k+1]$
    if $b < i < a'$ **then**
      return $(a,b),(a',b')$
    end if
  end for
  return $\perp$
end function

7.1 Buff-Find

The algorithm Block-Decode makes use of the sub-routine Buff-Find, presented in Algorithm 3. At a high-level, the algorithm Buff-Find on input i and given oracle access

to (corrupted) codeword c' searches the ball $c'[i-\eta\tau, i+\eta\tau]$ for all δ_{b}-*approximate buffers* in the interval, where $\eta \geq 1$ is a constant such that if $i \in \phi^{-1}(j)$ for any good block j then $c'[\phi^{-1}(j)] \subseteq c'[i-\eta\tau, i+\eta\tau]$. Briefly, for any $k \in \mathbb{N}$ and $\delta_{\mathsf{b}} \in (0,1/2)$ a string $w \in \{0,1\}^k$ is a δ_{b}-*approximate buffer* if $\mathsf{ED}(w, 0^k) \leq \delta_{\mathsf{b}} \cdot k$. For brevity we refer to approximate buffers simply as buffers. Once all buffers are found, the algorithm attempts to find a pair of consecutive buffers such that the index i is between these two buffers. If two such buffers are found, then the algorithm returns these two consecutive buffers. For notational convenience, for integers $a < b$ we let the tuple (a, b) denote a (approximate) buffer.

▶ **Lemma 20.** *Let $i \in [n']$ and $j \in [d]$. There exist constants $\gamma < \delta_{\mathsf{b}} \in (0,1/2)$ such that if $i \in \phi^{-1}(j)$ then* Buff-Find *finds buffers (a_1, b_1) and (a_2, b_2) such that $\mathsf{Dec}_{in}(c'[b_1+1, a_2-1]) = j \circ b_j$. Further, if $b_1 < i < a_2$ then* Block-Decode *outputs $j \circ b_j$.*

8 Parameter Setting and Proof of Theorem 6

In this section we list a set of constraints which our setting of parameters must satisfy, and then complete the proof of Theorem 6. These constraints are required by different parts of the analysis. Recall that $\delta_{out}, \delta_{in} \in (0,1)$ and $\beta_{in} \geq 1$ are given as parameters of the outer code and the inner code, and that $\beta = 2\alpha + \beta_{in}\left(1 + \log|\Sigma|\right)$. We have that $\beta \geq 2$ for any non-negative α.

▶ **Proposition 21.** *There exists constants $\gamma, \theta \in (0,1)$ and $\alpha = \Omega(\delta_{in})$ such that the following constraints hold:*

1. $\gamma \leq 1/12$ *and* $\theta < 1/50$*;*
2. $(\beta+\gamma)/(\beta-\gamma) < 4/3$*;*
3. $\alpha \leq 2\gamma/(\gamma+6)$*;*
4. $\alpha(1+3\gamma)/(\beta-2\alpha) < \delta_{in}$*.*

Proof. For convenience of the reader and simplicity of the presentation we work with explicit values and verify that they satisfy the constraints in Proposition 21. Let $\gamma = 1/12$ and $\theta = 1/51$, which satisfies constraint (1). Note that $\gamma < 2/7 \leq \beta/7$, hence

$$\frac{\beta+\gamma}{\beta-\gamma} < \frac{4}{3}$$

and constraint (2) is satisfied. We take $\alpha = 2\gamma\delta_{in}/(\gamma+6)$ so that $\alpha = \Omega(\delta_{in})$ and constraint (3) is satisfied. Note also that $\beta - 2\alpha = \beta_{in}(1+\log|\Sigma|) \geq 2$ which implies

$$\frac{\alpha(1+3\gamma)}{\beta-2\alpha} \leq \frac{\alpha(1+3\gamma)}{2} = \frac{\alpha(\gamma+3\gamma^2)}{2\gamma} < \frac{\alpha(\gamma+6)}{2\gamma} = \delta_{in}.$$

Therefore, constraint (4) is also satisfied. ◀

We let

$$\delta = \frac{\delta_{out}\alpha\gamma}{2\beta(1+1/\theta)} = \Omega\left(\delta_{in}\delta_{out}\right).$$

We now prove Theorem 6, which shows Theorem 1.

Proof of Theorem 6. Recall that the decoder Dec works as follows. Given input index $i \in [k]$ and oracle access to $c' \in \{0,1\}^{n'}$, $\mathsf{Dec}^{c'}(i)$ simulates $\mathsf{Dec}^{s'}_{out}(i)$. Whenever $\mathsf{Dec}^{s'}_{out}(i)$ queries an index $j \in [m]$, the decoder expresses $j = (p-1)\tau + r_j$ for $p \in [d]$ and $0 \leq r_j < \tau$,

and runs Algorithm 1 on input $(c', 1, n'+1, p)$ to obtain a τ-long string b'_p. Then it feeds the (r_j+1)-th symbol of b'_p to $\mathsf{Dec}^{s'}_{out}(i)$. At the end of the simulation, $\mathsf{Dec}^{c'}(i)$ returns the output of $\mathsf{Dec}^{s'}_{out}(i)$.

For $p \in [d]$, let $b'_p \in \Sigma^\tau \cup \{\perp\}$ be a random variable that has the same distribution as the output of Algorithm 1 on input $(c', 1, n'+1, p)$. Define a random string $s' \in \left(\Sigma \cup \{\perp\}\right)^m$ as follows. For every $i \in [m]$ such that $i = (p-1)\tau + r$ for $p \in [d]$ and $0 \le r < \tau$,

$$s'[i] = \begin{cases} b'_p[r] & \text{if } b'_p \neq \perp, \\ \perp & \text{if } b'_p = \perp. \end{cases}$$

Since $b'_p = b_p$ implies $s'[(p-1)\tau + r] = s[(p-1)\tau + r]$ for all $0 \le r < \tau$, the event $E_s := \left\{\Pr_{j\in[m]}\left[s'[j] \neq s[j]\right] \le \delta_{out}\right\}$ is implied by the event $E_b := \{\Pr_{j\in[d]}\left[b'_j \neq b_j\right] \le \delta_{out}\}$. Theorem 16 implies that $\Pr[E_s] \ge \Pr[E_b] \ge 1 - \mathsf{negl}(n')$. According to the construction of Dec, from the perspective of the outer decoder, the string s' is precisely the string it is interacting with. Hence by properties of Dec_{out} we have that

$$\forall i \in [k], \quad \Pr\left[\mathsf{Dec}^{s'}_{out}(i) = x[i] \;\middle|\; E_s\right] \ge \frac{1}{2} + \epsilon_{out}.$$

Therefore by construction of Dec we have

$$\begin{aligned} \forall i \in [k], \quad \Pr\left[\mathsf{Dec}^{c'}(i) = x[i]\right] &\ge \Pr\left[E_s\right] \cdot \Pr\left[\mathsf{Dec}^{s'}_{out}(i) = x[i] \;\middle|\; E_s\right] \\ &\ge \left(1 - \mathsf{negl}(n')\right) \cdot \left(\frac{1}{2} + \epsilon_{out}\right) \ge \frac{1}{2} + \epsilon_{out} - \mathsf{negl}(n'). \end{aligned}$$

The query complexity of Dec is $\ell_{out} \cdot O\left(\log^4 n'\right)$ since it makes ℓ_{out} calls to Algorithm 1, which by Proposition 17 has query complexity $O\left(\log^4 n'\right)$. ◀

References

1. Joël Alwen, Jeremiah Blocki, and Ben Harsha. Practical graphs for optimal side-channel resistant memory-hard functions. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1001–1017, Dallas, TX, USA, October 31 – November 2 2017. ACM Press. doi:10.1145/3133956.3134031.
2. Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 99–130, Tel Aviv, Israel, April 29 – May 3 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-78375-8_4.
3. Joshua Brakensiek, Venkatesan Guruswami, and Samuel Zbarsky. Efficient low-redundancy codes for correcting multiple deletions. *IEEE Trans. Inf. Theory*, 64(5):3403–3410, 2018.
4. Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In Shang-Teng Huang, editor, *19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 268–276, San Francisco, CA, USA, January 20–22 2008. ACM-SIAM.
5. Aditi Dhagat, Péter Gács, and Peter Winkler. On playing "twenty questions" with a liar. In Greg N. Frederickson, editor, *3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 16–22, Orlando, Florida, USA, January 27–29 1992. ACM-SIAM.
6. Zeev Dvir, Parikshit Gopalan, and Sergey Yekhanin. Matching vector codes. *SIAM J. Comput.*, 40(4):1154–1178, 2011.

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

8 Paul Erdös, Ronald L. Graham, and Endre Szemerédi. On sparse graphs with dense long paths, 1975.

9 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, October 1994. doi:10.1137/S0097539791195877.

10 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Sorting with recurrent comparison errors, September 2017.

11 Venkatesan Guruswami, Bernhard Haeupler, and Amirbehshad Shahrasbi. Optimally resilient codes for list-decoding from insertions and deletions. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 524–537. ACM, 2020.

12 Venkatesan Guruswami and Ray Li. Coding against deletions in oblivious and online models. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 625–643. SIAM, 2018.

13 Venkatesan Guruswami and Ray Li. Polynomial time decodable codes for the binary deletion channel. *IEEE Transactions on Information Theory*, 65(4):2171–2178, 2018.

14 Venkatesan Guruswami and Ray Li. Polynomial time decodable codes for the binary deletion channel. *IEEE Trans. Inf. Theory*, 65(4):2171–2178, 2019.

15 Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, 2017.

16 Bernhard Haeupler. Optimal document exchange and new codes for insertions and deletions. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 334–347. IEEE Computer Society, 2019.

17 Bernhard Haeupler, Aviad Rubinstein, and Amirbehshad Shahrasbi. Near-linear time insertion-deletion codes and $(1+\epsilon)$-approximating edit distance via indexing. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 697–708. ACM, 2019.

18 Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: codes for insertions and deletions approaching the singleton bound. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 33–46. ACM, 2017.

19 Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: explicit constructions, local decoding, and applications. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 841–854. ACM, 2018.

20 Bernhard Haeupler, Amirbehshad Shahrasbi, and Madhu Sudan. Synchronization strings: List decoding for insertions and deletions. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

21 Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 881–890, New Orleans, LA, USA, January 7–9 2007. ACM-SIAM.

22 Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC*, pages 80–86, 2000.

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

24 Marcos Kiwi, Martin Loebl, and Jiri Matousek. Expected length of the longest common subsequence for large alphabets.

25 Rolf Klein, Rainer Penninger, Christian Sohler, and David P. Woodruff. Tolerant algorithms. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms – ESA 2011*, pages 736–747, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

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

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

28 Swastik Kopparty, Shubhangi Saraf, and Sergey Yekhanin. High-rate codes with sublinear-time decoding. *J. ACM*, 61(5):28:1–28:20, 2014.

29 Vladimir Iosifovich Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

30 Shu Liu, Ivan Tjuawinata, and Chaoping Xing. On list decoding of insertion and deletion errors. *CoRR*, abs/1906.09705, 2019. URL: `http://arxiv.org/abs/1906.09705`.

31 Hugues Mercier, Vijay K. Bhargava, and Vahid Tarokh. A survey of error-correcting codes for channels with symbol synchronization errors. *IEEE Communications Surveys and Tutorials*, 12, 2010.

32 Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels, July 2008.

33 Rafail Ostrovsky and Anat Paskin-Cherniavsky. Locally decodable codes for edit distance. In Anja Lehmann and Stefan Wolf, editors, *Information Theoretic Security*, pages 236–249, Cham, 2015. Springer International Publishing.

34 L. J. Schulman and D. Zuckerman. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE Transactions on Information Theory*, 45(7):2552–2557, 1999.

35 N.J.A. Sloane. On single-deletion-correcting codes. *arXiv: Combinatorics*, 2002.

36 Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the XOR lemma (abstract). In *CCC*, page 4, 1999.

37 David P. Woodruff. A quadratic lower bound for three-query linear locally decodable codes over any field. *J. Comput. Sci. Technol.*, 27(4):678–686, 2012.

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

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