Locally Decodable Codes with Randomized Encoding

Kuan Cheng [*] Xin Li [†] Yu Zheng [‡]

January 10, 2020

Abstract

We initiate a study of locally decodable codes with randomized encoding. Standard locally decodable codes are error correcting codes with a deterministic encoding function and a randomized decoding function, such that any desired message bit can be recovered with good probability by querying only a small number of positions in the corrupted codeword. This allows one to recover any message bit very efficiently in sub-linear or even logarithmic time. Besides this straightforward application, locally decodable codes have also found many other applications such as private information retrieval, secure multiparty computation, and average-case complexity.

However, despite extensive research, the tradeoff between the rate of the code and the number of queries is somewhat disappointing. For example, the best known constructions still need super-polynomially long codeword length even with a logarithmic number of queries, and need a polynomial number of queries to achieve a constant rate. In this paper, we show that by using a randomized encoding, in several models we can achieve significantly better rate-query tradeoff. In addition, our codes work for both the standard Hamming errors, and the more general and harder edit errors.

1 Introduction

Locally decodable codes (LDCs) are error correcting codes that allow one to decode any specific message symbol by querying only a few symbols from the received codeword, while still tolerating some (say a constant) fraction of errors. This enables the appealing feature of recovering any message symbol very efficiently without reading the entire codeword, and usually this can be done in sublinear time. This feature is especially useful when dealing with large data streams (e.g., databases), and when the recovering of the entire data stream is unnecessary. While locally decodable codes can be defined over an alphabet of any size at least two, in this paper we focus on the case of binary alphabet. Standard locally decodable codes have been studied extensively for Hamming errors, and have found many applications in theory and practice. These include for example private information retrieval, secure multiparty computation, average-case complexity and many more. We refer the reader to [Yek12] for an excellent survey on this topic.

However, in contrast to standard error correcting codes which can achieve excellent rate (the ratio between the length of the message and the length of the codeword), for locally decodable

[*]ckkcdh@hotmail.com. Department of Computer Science, University of Texas at Austin. Supported by a Simons Investigator Award (#409864, David Zuckerman) and NSF Award CCF-1617713.

[†]lixints@cs.jhu.edu. Department of Computer Science, Johns Hopkins University. Supported by NSF Award CCF-1617713 and NSF CAREER Award CCF-1845349.

[‡]yuzheng@cs.jhu.edu Department of Computer Science, Johns Hopkins University. Supported by NSF Award CCF-1617713 and NSF CAREER Award CCF-1845349.

codes the situation is somewhat disappointing. Specifically, if one considers the most natural and interesting case of tolerating a constant fraction of errors, then standard error correcting codes can simultaneously achieve a constant rate. For locally decodable codes, the rate also depends on the number of queries for decoding. Specifically, consider the standard setting where our goal is to recover any message bit with probability $2/3$. With r queries and for message length k, it is known that the codeword length needs to be at least $\Omega(k^{(r/r-1)})$ [KT00] and hence if r is a constant,[1] the codeword length needs to be at least polynomially long and thus achieving a constant rate is impossible. However, even for the case of $r = O(\log k)$ there is no explicit construction of locally decodable codes with constant rate, and in fact the best known construction in this case fails to achieve codeword length polynomially in k. Currently, the best known constructions need $r = k^{\Omega(1)}$ queries to achieve a constant rate [KSY11]. Below we briefly summarize the parameters of the best known constructions for different regimes of query complexity, while fixing the success probability of decoding any bit to be say $2/3$. For constant query $r = O(1)$, the best known construction is by using matching vector codes [DGY11], which gives for example a code that uses $r = 3 \cdot 2^{t-2}$ queries and can tolerate $\delta = O(1/r)$ fraction of errors. The codeword has length $\mathsf{expexp}_t((\log k)^{1/t}(\log\log k)^{1-1/t})$, which is slightly sub-exponential. In contrast, the lower bound only gives a polynomial codeword length. For query complexity $r = O(\log k)$, the best known construction is also due to matching vector codes, which gives codeword length $\mathsf{expexp}(\log\log^{3-o(1)} k)$. Note that this is super-polynomial, while the lower bound already becomes useless. For query complexity $r = \log^t k$ with constant $t > 1$, Reed-Muller codes give codeword length $k^{1+1/(t-1)+o(1)}$. Finally, for query complexity $r = k^{\varepsilon}$ for any constant $\varepsilon > 0$, multiplicity codes [KSY11] give codeword length $O(k)$ while tolerating some constant $\delta = \delta(\varepsilon)$ fraction of errors. Thus, there is a huge gap between the upper bounds and the lower bounds. This means that known constructions of locally decodable codes either need a lot of redundancy information, or need to use relatively large number of queries, which is undesirable. Closing this gap is an important open problem, which evidently appears to be quite hard.

Note that to ensure the property of local decoding, it is necessary to change the decoding from a deterministic algorithm into a randomized algorithm, since otherwise an adversary can just corrupt all the bits the decoding algorithm queries. Hence, the decoding becomes probabilistic and allows some small probability of failure. In this paper, with the goal of improving the rate of locally decodable codes in mind, we initialize the study of a relaxed version of locally decodable codes which is also equipped with randomized encoding. Now the probability of decoding failure is measured over the randomness of both the encoding and decoding.

There are several questions that we need to clarify in this new model. The first question is: *Does the decoding algorithm know and use the randomness of the encoding?* Although we cannot rule out the possibility of constructions where the decoding algorithm can succeed *without* knowing the randomness of the encoding, in this paper we only consider the most natural setting, where the decoding algorithm indeed knows and uses the randomness of encoding. After all, without the encoding's randomness the decoding algorithm does not even completely know the encoding function. The second question, as we are considering an adversarial situation, is: *Is the adversary allowed to know the randomness of the encoding?*

This turns out to be a very interesting question. Of course, an adversary who knows the randomness of encoding is much stronger, and thus we need a stronger construction of codes as well. In this paper, we provide a partial answer to this question, by showing that under some reasonable assumption of the code, a locally decodable code in our model where the adversary knows the randomness of encoding is equivalent (up to constant factors) to a standard locally

[1]For the case of $r = 2$ there is a stronger lower bound of $2^{\Omega(k)}$ [KdW04].

decodable codes. The assumption is that the code has the following homogenous property:

Property (*) : For any fixing of the encoding's randomness, any fixed error pattern, and any fixed target message bit, the success probability (over the decoding's randomness) of decoding this bit is the same for all possible messages.

We note that this property is satisfied by all known constructions of standard locally decodable codes. Part of the reason is that all known constructions are linear codes, and the decoding only uses non adaptive queries. We now have the following theorem.

Theorem 1. *Suppose there is an LDC with randomized encoding for Hamming errors, with message length k, codeword length n, that can tolerate δ fraction of errors and successfully decode any message bit with probability $1-\varepsilon$ using r queries. Then there also exists a (possibly non explicit) standard LDC, with message length $k/2$, codeword length n, that can tolerate δ fraction of errors and successfully decode any message bit with probability $1-2\varepsilon$ using r queries.*

Proof. Fix a specific random string used by the encoding, and a target message bit for decoding (say bit i). Since now the encoding becomes a deterministic function, and there are only finite number of possible error patterns, there is some error pattern that is the *worst* for decoding, i.e., the one that minimizes the success probability of decoding bit i. We say the fixed random string is *good* for bit i if this success probability is at least $1-2\varepsilon$. Note that the overall success probability of decoding bit i is at least $1-\varepsilon$, thus by a Markov argument the probability that a random string is good for bit i is at least $1/2$.

Now again, by an averaging argument, there exists a fixed string that is good for at least $1/2$ fraction of the message bits. Fix this string and the encoding now becomes deterministic. Without loss of generality assume that the string is good for the first half of the message bits. Now fix the rest of the message bits to any arbitrary string (e.g, all 0), we now have a standard LDC with message length $k/2$, codeword length n, that can tolerate δ fraction of errors and successfully decode any message bit with probability $1-2\varepsilon$ using r queries. $\square$

The above theorem shows that, in order to get significantly better rate-query tradeoff, we need to either forbid the adversary to know the randomness of encoding, or to construct codes that do not satisfy property (*) (e.g., using adaptive encoding or adaptive decoding). In this paper we study the first setting, which is naturally simpler, and we leave codes in the second setting as an interesting open problem. In general, there are two different models where the adversary is not allowed to know the randomness of encoding:

Shared randomness In this model, the encoder and the decoder share a private uniform random string. Thus, the adversary does not know the randomness used by the encoder; but he can add arbitrary errors to the codeword, including looking at the codeword first and then adaptively add errors.

Oblivious channel In this model, the encoder and the decoder do *not* share any randomness. However, the communication channel is oblivious, in the sense that the adversary can add any error pattern *non adaptively*, i.e., without looking at the codeword first.

Here we study both models. We now give our formal definitions of locally decodable codes with randomized encoding.

Definition 1 (LDC with a fixed Failure Probability)**.** *An $(n,k,\delta,q,\varepsilon)$ LDC with randomized encoding consists of a pair of randomized functions $\{\mathsf{Enc},\mathsf{Dec}\}$, such that:*

- $\mathsf{Enc}:\{0,1\}^k\to\{0,1\}^n$ *is the encoding function. For every message $x\in\{0,1\}^k$, $y=\mathsf{Enc}(x)\in\{0,1\}^n$ is the corresponding codeword.*

- $\mathsf{Dec} : [k] \times \{0,1\}^* \to \{0,1\}$ *is the decoding function. If the adversary adds at most δn errors to the codeword, then for every $i \in [k]$, every $y \in \{0,1\}^*$ which is a corrupted codeword,*

  $$\Pr[\mathsf{Dec}(i, y) = x_i] \geq 1 - \varepsilon,$$

  where the probability is taken over the randomness of both Enc *and* Dec*.*

- Dec *makes at most q queries to y.*

Remark. *Two remarks are in order. First, our definition is quite general in the sense that we don't restrict the type of errors. In particular, in this paper we study both Hamming errors (i.e., bit flips) and edit errors (i.e., insertions, deletions, and substitutions). Second, as mentioned before, the model can either assume shared private randomness or an oblivious adversarial channel.*

The above definition is for a fixed failure probability ε. However, sometimes it is desirable to achieve a smaller failure probability. For standard locally decodable codes, this can usually be done by repeating the decoding algorithm independently for several times, and then taking a majority vote. This decreases the failure probability at the price of increasing the query complexity. In contrast, in our new model, this approach is not always feasible. For example, in the extreme case one could have a situation where for some randomness used by the encoding, the decoding succeeds with probability 1; while for other randomness used by the encoding, the decoding succeeds with probability 0. In this case repeating the decoding algorithm won't change the failure probability. To rule out this situation, we also define a locally decodable codes with *flexible* failure probability.

Definition 2 (LDC with Flexible Failure Probabilities)**.** *An (n, k, δ) LDC with randomized encoding and query complexity function $q : \mathbb{N} \times [0,1] \to \mathbb{N}$, consists of a pair of randomized algorithms $\{\mathsf{Enc}, \mathsf{Dec}\}$, such that:*

- $\mathsf{Enc} : \{0,1\}^k \to \{0,1\}^n$ *is the encoding function. For every message $x \in \{0,1\}^k$, $y = \mathsf{Enc}(x) \in \{0,1\}^n$ is the corresponding codeword.*

- $\mathsf{Dec} : [k] \times \{0,1\}^* \to \{0,1\}$ *is the decoding function. If the adversary adds at most δn errors to the codeword, then for every $i \in [k]$, every $y \in \{0,1\}^*$ which is a corrupted codeword, and every $\varepsilon \in [0,1]$,*

  $$\Pr[\mathsf{Dec}(i, y) = x_i] \geq 1 - \varepsilon,$$

  while Dec *makes at most $q = q(n, \varepsilon)$ queries to y. The probability is taken over the randomness of both* Enc *and* Dec*.*

Again, this definition can apply to both Hamming errors and edit errors, and both the model of shared randomness and the model of an oblivious adversarial channel.

To the best of our knowledge, locally decodable codes with randomized encoding have not been studied before in the literature. In this paper we provide several constructions with rate-query tradeoff better than standard locally decodable codes. We have the following theorems. The first one deals with a fixed decoding failure probability.

Theorem 2. *There exists a constant $0 < \delta < 1$ such that for every $k \in \mathbb{N}$ there is an efficient construction of $(n, k, \delta, q, \varepsilon)$ LDC with randomized encoding, where $n = O(k)$. For any $\varepsilon \in [0,1]$, the query complexity is as follows.*

- *for Hamming errors with shared randomness, $q = O(\log \frac{1}{\varepsilon})$;*

- *for Hamming errors with an oblivious channel,* $q = O(\log k \log \frac{1}{\varepsilon})$*;*
- *for Edit errors, both with shared randomness and with an oblivious channel,* $q = \mathsf{polylog} k \log \frac{1}{\varepsilon}$.

We can compare our theorem to standard locally decodable codes. Achieving $q = O(\log \frac{1}{\varepsilon})$ is equivalent to achieving constant query complexity for success probability $2/3$. For standard LDCs we know in this case it is impossible to achieve a constant rate, and the best known construction still has sub-exponential codeword length. Similarly, achieving $q = O(\log k \log \frac{1}{\varepsilon})$ is equivalent to achieving query complexity $O(\log k)$ for success probability $2/3$, and for standard LDCs the best known construction still has super-polynomial codeword length. In contrast, we can achieve a constant rate.

The next theorem deals with a flexible decoding failure probability. We note that one way to achieve this is to repeat the encoding several times independently, and send all the obtained codewords together. The decoding will then decode from each codeword and take a majority vote. However, this approach can decrease the rate of the code dramatically. For example, if one wishes to reduce the failure probability from a constant to $2^{-\Omega(k)}$, then one needs to repeat the encoding for $\Omega(k)$ times and the rate of the code decreases by a factor of $1/k$. In this work we use a different construction that can achieve a much better rate.

Theorem 3. *There exists a constant* $0 < \delta < 1$ *such that for every* $k \in \mathbb{N}$ *there is an efficient construction of* (n, k, δ) *LDC with randomized encoding and flexible failure probability, where* $n = O(k \log k)$*. The query complexity is as follows.*

- *for Hamming errors with shared randomness,* $q = O(\log k \log \frac{1}{\varepsilon})$*;*
- *for Hamming errors with an oblivious channel,* $q = O(\log^2 k \log \frac{1}{\varepsilon})$*;*
- *for Edit errors, both with shared randomness and with an oblivious channel,* $q = \mathsf{polylog} k \log \frac{1}{\varepsilon}$.

We can also compare this theorem to standard locally decodable codes. Again, achieving $q = O(\log k \log \frac{1}{\varepsilon})$ is equivalent to achieving query complexity $O(\log k)$ for success probability $2/3$, and for standard LDCs the best known construction still has super-polynomial codeword length. Similarly, achieving $q = O(\log^2 k \log \frac{1}{\varepsilon})$ is equivalent to achieving query complexity $O(\log^2 k)$ for success probability $2/3$, and for standard LDCs the best known construction using Reed-Muller codes needs codeword length $k^{2+o(1)}$. In contrast, we can achieve codeword length $O(k \log k)$.

1.1 Related works.

As mentioned earlier, standard locally decodable codes for Hamming errors have been studied extensively, with many beautiful constructions such as [Yek08], [Efr09], [DGY11], and we refer the reader to [Yek12] for an excellent survey.

For edit errors, the situation is quite different. Even for standard error correcting codes (not locally decodable), the progress for edit errors is much slower than that for Hamming errors. For example, the first asymptotically good code for edit errors is not known until the work of Schulman and Zuckerman in 1999 [SZ99]. In recent years, there has been a line of research [GW17], [GL16], [BG16], [Bel15], [HSV18], [CJLW18], [Hae19], [CJLW19] that makes constant progress and finally culminates in [CJLW18], [Hae19] which give codes that can correct t edit errors with redundancy (the extra information needed for the codeword) size $O(t \log^2(k/t))$, where k is the message length. This is within a $\log(k/t)$ factor to optimal. In [CJLW18], the authors also give a code that can correct t edit errors with redundancy $O(t \log k)$, which is optimal up to constants for $t \leq k^{1-\alpha}$, any constant $\alpha > 0$. In another line of work [HS17],

[HS18], [CHL$^+$19], near optimal codes for edit errors over a larger alphabet are constructed, using the constructions of *synchronization strings*.

In terms of locally decodable codes, no known constructions for Hamming errors can be used directly as a code against edit error. This is due to the shift caused by insertions and deletions, which causes the loss of information about the positions of the bits queried. The only known previous work of locally decodable codes for edit errors is the work by Ostrovsky and Paskin-Cherniavsky [OPC15]. There the authors provided a "compiler" to transform a standard locally decodable code for Hamming error to a locally decodable code for edit error. Their construction first takes a standard LDC as a black box, then concatenate it with an asymptoticaly good code for edit errors. They showed that the fraction of error tolerated and the rate of the resulted code both only decrease by a constant factor. Meanwhile, the query complexity increases by factor of $polylog(n)$, where n is the length of the codeword.

Standard error correcting codes with randomized encoding (but deterministic decoding) have been studied by Guruswami and Smith [GS16]. They constructed such codes for an oblivious adversarial channel that can add δ fraction of errors, where the rate of the codes approaches the Shannon capacity $1 - H(\delta)$. Their encoding and decoding run in polynomial time and the decoding error is exponentially small.

1.2 Technique Overview

1.2.1 Hamming Error

We start from constructions for Hamming errors and let the fraction of errors be some constant δ. To take advantage of a randomized encoding, our approach is to let the encoder perform a *random permutation* of the codeword. Assuming the adversary does not know the randomness of the encoding, this effectively reduces adversarial errors into random errors, in the following sense: if we look at any subset of coordinates of the codeword before the random permutation, then the expected fraction of errors in these coordinates after the random permutation is also δ. A stronger statement also follows from concentration bounds that with high probability this fraction is not far from δ, and in particular is also a constant. This immediately suggests the following encoding strategy: first partition the message into blocks, then encode each block with a standard asymptotically good code, and finally use a random permutation to permute all the bits in all resulted codewords. Now to decode any target bit, one just needs to query the bits in the corresponding codeword for the block that contains this bit. As the error fraction is only a constant, a concentration bound for *random permutations* shows that in order to achieve success probability $1 - \varepsilon$, one needs block length $O(\log(1/\varepsilon))$ and this is also the number of queries needed. To ensure that the adversary does not learn any information about the random permutation by looking at the codeword, we also use the shared randomness to add a mask to the codeword (i.e., we compute the XOR of the actual codeword with a random string). This gives our codes for a fixed failure probability, in the model of shared randomness.

To modify our construction to the model of an oblivious channel, note that here we don't need a random mask anymore, but the encoder has to tell the decoder the random permutation used. However the description of the random permutation itself can be quite long, and this defeats our purpose of local decoding. Instead, we use a *pseudorandom permutation*, namely an almost $\kappa = \Theta(\log \frac{1}{\varepsilon})$ wise independent permutation with error $\varepsilon/3$. Such a permutation can be generated by a short random seed with length $r = O(\kappa \log n + \log(1/\varepsilon))$, and thus is good enough for our application. The encoder will first run the encoding algorithm described previously to get a string y with length $n/2$, and then concatenate y with an encoded version $z \in \{0,1\}^{n/2}$ of the random seed. The decoder will first recover the seed and then perform local decoding as before. To ensure the seed itself can be recovered by using only local queries, we encode the seed by using a concatenation code where the outer code is a (n_1, k_1, d_1) Reed-Solomon code with alphabet size $\mathsf{poly}(n)$, and the inner code is an (n_2, k_2, d_2) asymptotically good code for $O(\log n)$ bits, where $n_1 n_2 = n/2, k_1 k_2 = r$, $d_1 = n_1 - k_1 + 1$, $n_2 = O(\log n)$. That is, each symbol of the outer code is encoded into another block of $O(\log n)$ bits. Now to recover the seed, the decoder first randomly chooses $8k_1$ blocks of the concatenated codes and decodes the symbols of the outer code. These decoded symbols will form a new (shorter) Reed-Solomon code, because they are the evaluations of a degree k_1 polynomial on $8k_1$ elements in $\mathbb{F}_2^{k_2}$. Furthermore, this code is still enough to recover the seed, because the seed is short and with high probability there are only a small constant fraction of errors in the decoded symbols. So the decoder can perform a decoding of the new Reed-Solomon code to recover the random seed with high probability. Note that the number of queries of this decoding is $8k_1 n_2 = O(r)$.

We now turn to our constructions for flexible failure probability. Here we want to achieve failure probability from a constant to say 2^{-k}. For each fixed failure probability we can use the previous construction, and this means the block size changes from a constant to $O(k)$. Instead of going through all of these sizes, we can just choose $O(\log k)$ sizes and ensure that for any desired failure probability ε, there is a block size that is at most twice as large as the size we need. Specifically, the block sizes are $2^{2^i}, i = 1, 2, \ldots, \log k$. In this way, we have $O(\log k)$ different codewords, and we combine them together to get the final codeword. Now for any failure probability ε, the decoder can look for the corresponding codeword (i.e., the one where the block size is the smallest size larger than $1/\varepsilon$) and perform local decoding. However, we cannot simply concatenate these codewords together since otherwise δ fraction of errors can completely ruin some codeword. Instead, we put them up row by row into a matrix of $O(\log k)$ rows, and we encode each column of $O(\log k)$ bits with another asymptotically good binary code. Finally we concatenate all the resulted codewords together into our final codeword of length $O(k \log k)$. Note that now to recover a bit for some codeword, we need to query a whole block of $O(\log k)$ bits, thus the query complexity increases by a $\log k$ factor while the rate decreases by a $\log k$ factor.

1.2.2 Edit Error

Our constructions for edit errors follow the same general strategy, but we need several modifications to deal with the loss of index information caused by insertions and deletions. Our construction for edit errors can achieve the same rate as those for Hamming case, but the query complexity for both models increases by a factor of $\mathsf{polylog}k$. We now give more details. We start with the construction of an $(n = O(k), k, \delta = \Omega(1), q = \mathsf{polylog}k \log(1/\varepsilon), \varepsilon)$ LDC for any $\varepsilon \in (0, 1)$, in the model with shared randomness. As in the case of Hamming errors, the shared randomness is used in two places: a random permutation π and some random masks to hide information. The construction has two layers.

For the first layer, view the message $x \in \{0,1\}^k$ as a sequence over the alphabet $\{0,1\}^{\log k}$ and divide it into $k/(k_0 \log k)$ small blocks each containing k_0 symbols from $\{0,1\}^{\log k}$. Then, we encode each block with Enc_0, which is an asymptotically good (n_0, k_0, d_0) code for Hamming errors over the alphabet $\{0,1\}^{\log k}$. Concatenating these $k/(k_0 \log k)$ codewords gives us a string of length $N = \frac{n_0 k}{k_0 \log k}$ over the alphabet $\{0,1\}^{\log k}$. We then permute these N symbols using π to get $y' = B_1 \circ B_2 \circ \cdots \circ B_N$ with $B_i \in \{0,1\}^{\log k}$. Since n_0/k_0 is a constant, we have $N < k$ for large enough k.

We are now ready to do the second layer of encoding. In the following, for each $i \in [N]$, $b_i \in \{0,1\}^{\log k}$ is the binary representation of i and $r_i \in \{0,1\}^{\log k}$ is a random mask shared between the encoder and decoder. $\mathbb{C}_0 : \{0,1\}^{2\log k} \to \{0,1\}^{10 \log k}$ is an asymptotically good code for edit errors and the $\oplus$ notation means bit-wise XOR. For each $i \in [N]$, we compute

$B_i' = \mathbb{C}_0(b_i \circ (B_i \oplus r_i)) \in \{0,1\}^{10\log k}$. We output $y = B_1' \circ B_2' \circ \cdots \circ B_N' \in (\{0,1\}^{10\log k})^N$, which is of length $n = 10\frac{n_0}{k_0}k = O(k)$. Note that the use of the random masks r_i's is to hide the actual codeword, so that the adversary cannot learn any information about the permutation.

To decode a given message bit, we need to find the corresponding block. The natural idea to do this is by binary search. However, this may fail with high probability due to the constant fraction of edit errors. To solve this issue, we use the techniques developed by [OPC15], which use a similar second layer encoding and give a searching algorithm with the following property: Even with δ-fraction of edit errors, at least $1 - O(\delta)$ fraction of the blocks can be recovered correctly with probability at least $1 - \mathsf{neg}(k)$. The algorithm makes a total of $\mathsf{polylog}k$ queries to the codeword for each search.

We now describe the decoding. Assume the bit we want to decode lies in the i-th block of x. Let $C_i \in (\{0,1\}^{\log k})^{n_0}$ be the codeword we get from encoding the i-th block using Enc_0. With the information of π, we can find out n_0 indices i_1 to i_{n_0} such that C_i is equal to $B_{\pi^{-1}(i_1)} \circ B_{\pi^{-1}(i_2)} \circ \cdots \circ B_{\pi^{-1}(i_{n_0})}$. The decoding algorithm calls the searching algorithm from [OPC15] to find all blocks B_{i_1}' to $B_{i_{n_0}}'$ in the received codeword. We say a block is unrecoverable if the searching algorithm failed to find it correctly. By the same concentration bound used in the Hamming case and the result from [OPC15], the fraction of unrecoverable blocks is bounded by a small constant with high probability. Thus, we can decode C_i correctly with the desired success probability. In this process, each search takes $\mathsf{polylog}k$ queries and $n_0 = O(\log 1/\varepsilon)$ of searches are performed. The total number of queries made is thus $\mathsf{polylog}k \log 1/\varepsilon$.

For the model of an oblivious channel, again we use a pseudorandom permutation π that can be generated by $O(\log n \log 1/\varepsilon)$ random bits. We use the same binary code as we used in the Hamming case to encode it and then view it as a string over the alphabet $\{0,1\}^{\log k}$. It is then concatenated with the code described previously before the second layer of encoding. After that, the same second layer of encoding is applied. The random masks used in the previous construction are no longer needed since the adversary can not see the codeword.

The construction for a flexible failure probability is also similar to the Hamming case. We write the codes before the second layer of encoding as a matrix M. The only difference is that, each element in the matrix M is now a symbol in $\{0,1\}^{\log k}$. We then encode the j-th column with an error correcting code over the alphabet $\{0,1\}^{\log k}$ to get a codeword z_j, and concatenate them to get z. After that, we do the second layer of encoding on z.

2 Preliminaries

2.1 Edit distance

Edit distance, or Levenstein distance, is a popular metric for quantifying how similar two strings are. It is defined using three types of string operations: insertion, deletion, and substitution. An insertion operation inserts a character into the string at a certain position. A deletion operation deletes a particular character from the string. And a substitution operation substitutes a character with a different one.

Definition 3. *The edit distance between two strings $x, y \in \Sigma^*$, denoted by $d_E(x,y)$, is the minimal number of edit operations (insertion, deletion, and substitution) needed to transform one into another. Assume x and y have same length n, the normalized edit distance of x and y is $d_E(x,y)$ devided by n. We denote the normalized edit distance by $\Delta_E(x,y)$*

2.2 Pseudorandom permutation

In our construction against oblivious channel, one key idea is to integrate a short description of a pseudorandom permutation into the codeword. We will utilize the construction of pseudorandom permutations from [KNR09]. Here, we follow the definition from their work and introduce their results.

Definition 4 (Statistical Distance). *Let D_1, D_2 be distributions over a finite set Ω. The variation distance between D_1 and D_2 is*

$$\|D_1 - D_2\| = \frac{1}{2}\sum_{\omega\in\Omega} |D_1(\omega) - D_2(\omega)|$$

We say that D_1 and D_2 are δ-close if $\|D_1 - D_2\| \leq \delta$.

Definition 5 (k-wise δ-dependent permutation). *Let $\mathcal{F}$ be a family of permutations on n elements (allow repetition). Let $\delta > 0$, we say the family $\mathcal{F}$ is k-wise δ-dependent if for every k-tuple of distinct elements $\{x_1, x_2, \cdots, x_k\} \in [n]$, for $f \in \mathcal{F}$ chosen uniformly, the distribution $\{f(x_1), f(x_2), \cdots, f(x_k)\}$ is δ-close to uniform distribution*

In practice, we want to construction explicit families of permutations. Two related parameters are:

Definition 6 (Description length). *The description length of a permutation family $\mathcal{F}$ is the number of random bits, used by the algorithm for sampling permutations uniformly at random from $\mathcal{F}$.*

Definition 7 (Time complexity). *The time complexity of a permutation family $\mathcal{F}$ is the running time of the algorithm for evaluating permutation from $\mathcal{F}$*

It is known that we can construct families of k-wise almost independent permutations with short description length (optimal up to a constant factor).

Theorem 4 (Theorem 5.9 of [KNR09]). *Let P_n denote the set of all permutation over $\{0,1\}^n$. There exists a k-wise δ-dependent family of permutation $\mathcal{F} \subset P_n$. $\mathcal{F}$ has description length $O(kn + \log\frac{1}{\delta})$ and time complexity $\mathsf{poly}(n, k, \log\frac{1}{\delta})$.*

2.3 Concentration bound

In our proof, we use the following concentration bound from [CIL17]

Lemma 1 ([CIL17]). *Let $\pi : [n] \to [n]$ be a random permutation. For any set $S, W \subseteq [n]$, let $u = \frac{|W|}{n}|S|$. Then the following holds.*

- *for any constant $\delta \in (0,1)$,*

$$\Pr[|\pi(S) \cap W| \leq (1-\delta)\mu] \leq e^{-\delta^2\mu/2},$$

$$\Pr[|\pi(S) \cap W| \geq (1+\delta)\mu] \leq e^{-\delta^2\mu/3}.$$

- *for any $d \geq 6\mu$, $\Pr[|\pi(S) \cap W| \geq d] \leq 2^{-d}$.*

3 Random Encoding LDC for Hamming Errors

3.1 With a Certain Failure Probability

We first show the construction of a Random Encoding LDC with an arbitrarily fixed failure probability for the case of shared randomness.

Theorem 5. *There is an efficient construction of $(n, k = \Omega(n), \delta = \Theta(1), q = O(\log n \log \frac{1}{\varepsilon}), \varepsilon)$ random LDC for the case of shared randomness, with arbitrarily $\varepsilon \in [0, 1]$.*

Construction 1. *We construct an $(n, k = \Omega(n), \delta = \Theta(1), q = O(\log n \log \frac{1}{\varepsilon}), \varepsilon)$-locally decodable binary code for arbitrary $\varepsilon \in [0, 1]$, for the case of shared randomness.*

Let δ_0, γ_0 be some proper constants in $(0, 1)$.

Let $(\mathsf{Enc}_0, \mathsf{Dec}_0)$ be an asymptotically good binary (n_0, k_0, d_0) error correcting code with $n_0 = O(\log \frac{1}{\varepsilon})$, $k_0 = \gamma_0 n_0$, $d_0 = 2\delta_0 n_0 + 1$.

Encoding function $\mathsf{Enc} : \{0, 1\}^{k=\Omega(n)} \to \{0, 1\}^n$ is a random function as follows.

1. *On input $x \in \{0, 1\}^k$, cut x into blocks evenly of length k_0 s.t. $x = (x_1, \ldots, x_{k/k_0}) \in (\{0, 1\}^{m_0})^{k/k_0}$; (If there are less than k bits in the last block then pad 0's)*
2. *Compute $y' = \mathsf{Enc}_0(x_1) \circ \ldots \circ \mathsf{Enc}_0(x_{k/k_0})$;*
3. *Generate a random permutation $\pi : [n] \to [n]$, where $n = \kappa k/k_0$;*
4. *Let $y = (y'_{\pi^{-1}(1)}, \ldots, y'_{\pi^{-1}(n)})$, i.e. permute y' using π;*
5. *Output $z = y \oplus w$ where w is a uniform random string.*

Decoding function $\mathsf{Dec} : [k] \times \{0, 1\}^n \to \{0, 1\}^k$ is as follows.

1. *On the input i and z, XOR it with w to get y;*
2. *Use r to reconstruct π;*
3. *Find i' s.t. the i-th bit of the message is in the i'-th block of m_0 bits;*
4. *Query the bits $Q = \{\, y_{\pi(j)} \mid j \in (\ (i' - 1)m_0,\ i'm_0\ ]\, \}$ to get $y'_{i'}$;*
5. *$x_{i'} = \mathsf{Dec}_0(y'_{i'})$;*
6. *Output $x[i]$.*

Proof of Theorem 5. Consider Construction 1.

We claim that the random permutation π and the z are independent. Actually for arbitrary permutation σ and arbitrary n-bit string a,

$$\Pr[\pi = \sigma, z = a] = \frac{1}{n!} \times \frac{1}{2^n}.$$

Because conditioned on any fixed permutation, to let $z = a$, there is a unique choice of w.

On the other hand, $\Pr[\pi = \sigma] = \frac{1}{n!}$ and $\Pr[z = a] = 2^{-n}$. This shows the independence of π and z.

So even if the adversary operates based on knowing z, the set of positions of the tampered bits are independent of π.

Since the number of errors is at most δn, the expected number of corrupted bits in $y'_{i'}$ is at most δn_0. By Lemma 1, the probability that the number of corrupted bits in $y'_{i'}$ is at most $1.1\delta n_0$, is at least $1 - 2^{-0.01 n_0/3} \geq 1 - \varepsilon/2$. By choosing n_0 to be large enough. Let $delta_0 = 1.1\delta$. Then Dec_0 can recover the desired bit from $y'_{i'}$ which has at most $1.1\delta n_0$ errors.

Note that the construction runs in polynomial time because generating random permutation, permuting bits, $(\mathsf{Enc}_0, \mathsf{Dec}_0)$ are all in polynomial time. The number of queries is $n_0 = O(\log \frac{1}{\varepsilon})$, since we only query bits in $y'_{i'}$. $\square$

Next we give the construction for oblivious channels.

We use the almost t-wise independent random permutations instead of random permutations for this case. A random function $\pi : [n] \to [n]$ is an ε almost t-wise independent random permutation if for every t elements $i_1, \ldots, i_t \in [n]$, $(\pi(i_1), \ldots, \pi(i_t))$ has statistical distance at most ε from $(\pi'(i_1), \ldots, \pi'(i_t))$ where π' is a random permutation over $[n]$. Kaplan, Naor and Reingold [KNR09] gives an explicit construction.

Next we show an efficient constructable $(n, k, \delta, q, \varepsilon)$-locally decodable code, which is asymptotically good, with $q = O(\log(1/\varepsilon))$ and ε can be arbitrary in $[0, 1]$.

Before the main construction, we first show a binary "locally decodable code" which can recover the whole message locally when the message length is sufficiently small.

Lemma 2. *For every $k \leq cn$ with sufficiently small constant $c = c(\delta) < 1$, every sufficiently small constant δ, there is an explicit $(n, k, \delta n)$ binary ECC, which has a randomized decoding algorithm s.t. it can compute the message with success probability $1 - 2^{-\Theta(k/\log n)}$, querying at most $q = O(k)$ bits.*

Proof. Consider the concatenation of an (n_1, k_1, d_1) Reed-Solomon code with alphabet $\{0,1\}^{n_2 = O(\log n_1)}$, $n_1 = n/n_2$, $k_1 = k/k_2$, and an explicit $(n_2, k_2 = n_2 - 2d_2(\log \frac{n_2}{d_2}), d_2 = \Theta(n))$ binary ECC.

The concatenated code is an $(n_1 n_2, k_1 k_2, d_1 d_2)$ code. Note that $n_1 n_2 = n$, $k_1 k_2 = k$. Also $d_1 = n_1 - k_1 + 1 = \Theta(n_1)$, $d_2 = O(n_2)$ so $d_1 d_2 = O(n)$.

We only need to show the decoding algorithm.

Given a codeword, we call the encoded symbols (encoded by the second code) of the first code as blocks. The algorithm randomly picks $8k_1$ blocks and query them. For each block, it calls the decoding function of the second code. After this we get $8k_1$ symbols of the first code. Then we use the decoding algorithm of a $(8k_1, k_1, 7k_1 + 1)$ Reed Solomon code to get the message.

Next we argue that this algorithm successes with high probability. Assume there are $\delta n = d_1 d_2/8$ errors, then there are at most $d_1/4$ codewords of the second code which are corrupted for at least $d_2/2$ bits. Since the distance of the second code is d_2, there are $n_1 - d_1/4$ blocks can be decoded to their correct messages.

Hence the expectation of the number of correctly recovered symbols of the first code is $\frac{n_1 - d_1/4}{n_1} 8k_1 \geq 6k_1$. Thus by Chernoff Bound, with probability $1 - 2^{-\Theta(k_1)}$, there are at least $5k_1$ symbols are correctly recovered. Note that if we look at the $8k_1$ queried blocks they also form a $(8k_1, k_1, 7k_1 + 1)$ Reed-Solomon code since it is the evaluation of the degree $k_1 - 1$ polynomial on $8k_1$ distinct values in the field $\mathbb{F}_2^{n_2}$. Thus, our recovered symbols form a string which has only distance $3k_1$ from a codeword of the code. So by using the decoding algorithm of the code we can get the correct message.

$\square$

Construction 2. *We construct an $(n, k = \Omega(n), \delta = \Theta(1), q = O(\log n \log \frac{1}{\varepsilon}), \varepsilon)$-locally decodable binary code for arbitrary $\varepsilon \in [0, 1]$.*

Let δ_0, γ_0 be some proper constants in $(0, 1)$.

Let $(\mathsf{Enc}_0, \mathsf{Dec}_0)$ be an asymptotically good binary (n_0, k_0, d_0) error correcting code with $n_0 = O(\log \frac{1}{\varepsilon})$, $k_0 = \gamma_0 n_0$, $d_0 = 2\delta_0 n_0 + 1$.

Encoding function $\mathsf{Enc} : \{0,1\}^{k=\Omega(n)} \to \{0,1\}^n$ *is a random function as follows.*

1. *On input* $x \in \{0,1\}^k$, *cut* x *into blocks evenly of length* k_0 *s.t.* $x = (x_1, \ldots, x_{k/k_0}) \in (\{0,1\}^{m_0})^{k/k_0}$*; (If there are less than* k *bits in the last block then pad* 0*'s)*
2. *Compute* $y' = \mathsf{Enc}_0(x_1) \circ \ldots \circ \mathsf{Enc}_0(x_{k/k_0})$*;*
3. *Generate a* $\varepsilon_\pi = \varepsilon/10$*-almost* $\kappa = O(\log \frac{1}{\varepsilon})$*-wise independent random permutation* $\pi : [n/2] \to [n/2]$ *using Theorem 4, where* $n/2 = \kappa k/k_0$*, the randomness used is* $r \in \{0,1\}^{d_\pi = O(\kappa \log n + \log(1/\varepsilon_\pi))}$*;*
4. *Let* $y = (y'_{\pi^{-1}(1)}, \ldots, y'_{\pi^{-1}(n)})$*, i.e. permute* y' *using* π*;*
5. *Use an* $(n/2, |r|, \delta n)$ *ECC from Lemma 2 to encode* r*, getting* $z \in \{0,1\}^{n/2}$*;*
6. *Output* $y \circ z$*.*

Decoding function $\mathsf{Dec} : [k] \times \{0,1\}^n \to \{0,1\}^k$ *is as follows.*

1. *On the input* i *and* $y \circ z$*, call the decoding algorithm from Lemma 2 on* z *to get* r*;*
2. *Use* r *to reconstruct* π*;*
3. *Find* i' *s.t.* $x[i]$ *is in* $x_{i'}$*;*
4. *Query the bits* $Q = \{\, y_{\pi(j)} \mid j \in (\ (i'-1)m_0,\ i'm_0\ ]\}$ *to get* $y'_{i'}$*;*
5. $x_{i'} = \mathsf{Dec}_0(y'_{i'})$*;*
6. *Output* $x[i]$*.*

Lemma 3. *The encoding and decoding are both efficient.*

Proof. Every step in the construction can be realized in polynomial time. The called functions in the construction are all from efficient constructions. □

Lemma 4. *The number of queries for the decoding algorithm is* $q = O(\log n \log \frac{1}{\varepsilon})$*.*

Proof. In step 1 of the decoding, calling the decoding algorithm from Lemma 2 takes $O(d_\pi)$ number of queries.

In step 4 of the decoding, the number of queries is n_0.

So the total number of queries is $n_0 + O(d_\pi) = O(\log n \log \frac{1}{\varepsilon})$. □

Lemma 5. *Assume there are* δn *errors, then with probability* $1 - \varepsilon$*, the decoding can get* x *correctly.*

Proof. For z, the number of errors is at most δn. Thus by Lemma 2, the decoding can get r correctly with probability $1 - \varepsilon/3$ by letting the constant factor in $|r| = d_\pi$ to be large enough.

For y, the number of errors is at most δn. If π is a uniform random permutation then the expected number of corrupted bits in $y'_{i'}$ is $2\delta n_0$.

By Lemma 1, the probability that the number of corrupted bits in $y'_{i'}$ is at most $2.2\delta n_0$ is at least $1 - 2^{-0.01\delta n_0/3} \geq 1 - \varepsilon/3$ if letting the constant factor in n_0 to be large enough.

When π is an ε_π-almost n_0-wise independent permutation, the probability that the number of corrupted bits in $y'_{i'}$ is at most $2.2\delta n_0$ is at least $1 - 2^{-0.01\delta n_0/3} + \varepsilon_\pi \geq 1 - \varepsilon/2$.

Let $\delta = \delta_0/3$. Then the decoding Dec_0 can recover the correct message from $2.2\delta n_0$ errors.

Hence by a union bound, with probability at least $1 - \varepsilon/3 - \varepsilon/2 \geq 1 - \varepsilon$, the decoding can recover $x_{i'}$ correctly.

Note that once $x_{i'}$ is recovered, $x[i]$ is in it and thus is correctly recovered. □

Theorem 6. *There is an efficient construction of* $(n, k = \Omega(n), \delta = \Theta(1), q = O(\log n \log \frac{1}{\varepsilon}), \varepsilon)$ *random LDC for oblivious channels,* $\varepsilon \in [0, 1]$.

Proof. We use Construction 2. The theorem directly follows from Lemma 3, 4, 5.

□

3.2 With Flexible Failure Probabilities

Next we consider a stronger definition which can handle all decoding failure probabilities.

Intuitively we want to use the previous construction for different *eps*, and then combine them. It turns out this indeed can work but with some extra techniques while the information rate is $\Theta(1/\log n)$.

Construction 3. *We construct a* $(n, k = \Theta(n/\log n), \delta = \Theta(1))$*-locally decodable binary code.*

Let $(\mathsf{Enc}_0, \mathsf{Dec}_0)$ *be a binary* (n_0, k_0, d_0) *error correcting code with* $n_0 = \gamma_0^{-1} \log n$, $k_0 = \log n$, $d_0 = 2\delta_0 n_0 + 1$, *for constant* $\gamma_0, \delta_0 \in [0, 1]$.

Let $(\mathsf{Enc}_i, \mathsf{Dec}_i)$ *be the* $(n_i = n/n_0, k, \delta_i = \Theta(1), q_i, \varepsilon = 2^{-2^i})$ *LDC from Theorem 6,* $i \in [\log n]$.

Encoding function $\mathsf{Enc} : \{0,1\}^{k=\Omega(n)} \to \{0,1\}^n$ *is as follows.*

1. *On input* $x \in \{0,1\}^k$, *compute* $y_i = \mathsf{Enc}_i(x)$ *for every* $i \in [\log n]$.
2. *Let* M *be a* $\log n \times n/\log n$ *matrix s.t.* $M[i][j]$ *is the* j*-th bit of* y_i.
3. *Output* $z = (\mathsf{Enc}_0(M_1), \ldots, \mathsf{Enc}_0(M_{n_1})$ *where* $M_j, j \in [n_1]$ *is the* j*-th column of* M.

Decoding function $\mathsf{Dec} : [k] \times \{0,1\}^* \times [0,1] \to \{0,1\}$ *is as follows.*

1. *On input* u, z, ε, *find the smallest* i *s.t.* $2^{-2^i} \leq \varepsilon$; *If it cannot be found, then query the whole* z;
2. *Compute* $w = \mathsf{Dec}_i(u, y_i)$ *but whenever* Dec_i *wants to query an* j*-th bit of* y_i, *we query* z_j *and get* $y_i[j]$ *from* $\mathsf{Dec}_0(z_j)$;
3. *Output* w.

Lemma 6. *The encoding and decoding are both efficient.*

Proof. Every step in the construction can be realized in polynomial time. The called functions in the construction are all from efficient constructions.

□

Lemma 7. *The number of queries for the decoding algorithm is* $q = O(\log^2 n \log \frac{1}{\varepsilon})$.

Proof. In step 2 of the decoding, calling the decoding algorithm from Theorem 6 takes $O(\log n \log \frac{1}{\varepsilon})$ number of queries. For each such query, the algorithm actually queries n_0 bits. So the total number of queries is as stated.

□

Lemma 8. *Assume there are $\delta = \delta_0 \min_{j\in[n_1]}\{\delta_j\}$ fraction of errors, then with probability $1-\varepsilon$, the decoding can get x correctly.*

Proof. As $\delta = \delta_0 \min_j\{\delta_j\}$, there are at most $\delta_j n_1$ number of z_js which each has more than δ_0 fraction of errors.

As a result, for y_j, there are at most $\delta_j n_1$ number of bits which cannot be computed correctly during decoding step 2. So Dec_j can compute $x[i]$ correctly by Theorem 6.

□

Theorem 7. *There is an efficient construction of $(n, k = \Omega(n/\log n), \delta = \Theta(1))$ random LDC which can recover any one message bit with probability $1-\varepsilon$, for any $\varepsilon \in [0,1]$, by doing $q = O(\log^2 n \log\frac{1}{\varepsilon})$ queries.*

Proof. It directly follows from Construction 3, Lemma 6, 7, 8.

□

Similarly we also have the following result for the case of shared randomness

Theorem 8. *There is an efficient construction of $(n, k = \Omega(n/\log n), \delta = \Theta(1))$ random LDC which can recover any one message bit with probability $1-\varepsilon$, for any $\varepsilon \in [0,1]$, by doing $q = O(\log^2 n \log\frac{1}{\varepsilon})$ queries.*

Proof Sketch. We modify Construction 3 by letting $(\mathsf{Enc}_i, \mathsf{Dec}_i), i \in [\log n]$ be from Theorem 5.

Note that the outputted codeword is independent of the permutations used in $(\mathsf{Enc}_i, \mathsf{Dec}_i), i \in [\log n]$ due to the using of masks in the construction of Theorem 5. This guarantees that a similar analysis can still work for this case.

□

4 Constructions against Edit Error

The constructions against edit error requires an additional layer of encoding. To make our proof easier to describe, we will utilize the following greedily constructed code from [SZ99].

Lemma 9 (Implicit from [SZ99])**.** *For small enough δ, there exists code $\mathbb{C} : \{0,1\}^n \to \{0,1\}^{4n}$ such that the edit distance between any two codewords is at least δ and for any codeword, every interval has at least half of 1's.*

The proof uses the same argument from [SZ99]. The only difference is that their proof only considers insertion and deletion. Our slightly modified proof includes substitution as a type of error.

Proof. For a codeword y, the number of words in $\{0,1\}^n$ that is within edit distance $2d$ is at most $\binom{n}{2d}^2 2^{2d}$. That is because each substitution can be replaced by a deletion and an insertion. For any two codewords x and y such that $d_E(x,y) \le d$, x can be transformed into y with at most $2d$ insertions and $2d$ deletions. There are $\binom{n}{2d}$ ways to delete $2d$ characters from y, $\binom{n}{n-2d}$ ways to put the remaining characters in proper position of x, and 2^{2d} ways to choose to the inserted characters.

Substitute d with δn for some small constant δ, since $\binom{n}{k} \le (\frac{ne}{k})^k$, we have

$$\binom{n}{2\delta n}^2 2^{2\delta n} \le (\frac{ne}{2\delta n})^{4\delta n} 2^{2\delta n} \le (\frac{e^4}{4\delta^4})^{\delta n} = (2^{\log\frac{e^4}{4\delta^4}})^{\delta n}$$

Pick $\delta \in (0, 0.018)$, we have $\binom{n}{2\delta n}^2 2^{2\delta n} \leq 2^{1/2n}$. This gives a greedy construction of edit error code with rate $\frac{1}{2}$ and tolerates a δ fraction of edit error for constant $\delta \leq 0.018$. We then ensure every interval has at least half 1's by inserting 1's into every other position. Note that this operation will not decrease the edit distance. It gives us a code with rate 1/4. □

In the discussion below, we will use the following error correcting code for edit distance:

$$\mathbb{C}_0 : \{0,1\}^N \to \{0,1\}^{5N}$$

such that for each message $S \in \{0,1\}^N$, $\mathbb{C}_0(S)$ is composed of two parts. The first part is a codeword $\mathbb{C}_0'(S)$ where $\mathbb{C}_0' : \{0,1\}^N \to \{0,1\}^{4N}$ is a greedily constructed code that can tolerate a constant fraction of edit errors. The second part is a buffer of 0's of length N. That is, $\mathbb{C}_0(S) = \mathbb{C}_0'(S) \circ 0^N$. We assume the normalized edit distance between any two codewords of $\mathbb{C}_0(S)$ is at least δ'.

Another key component in our construction against edit error is a searching algorithm developed by [OPC15].

Their work provides an algorithm for searching from a weighted sorted list L with a constant fraction of errors. An element (i, a_i) in the sorted list is composed of two parts, an index i and content in that element, a_i. All elements in the list are sorted by their index, i.e. the j-th element in L is (j, a_j). Beyond that, each element is equipped with a non-negative weight. When sampling from the list, each element is sampled with probability proportional to its weight.

In [OPC15], the authors proved the following result.

Lemma 10. *[Theorem 16 of [OPC15]] Assume L' is a corrupted version of a weighted sorted list L with k elements, such that the total weight fraction of corrupted elements is some constant δ of the total weight of L. And the weights have the property that all sequences for $r \geq 3$ elements in the list have total weight in the range $[r/2, 2r]$. Then, there is an algorithm for searching L. For at least a $1 - O(\delta)$ fraction of the original list's elements, it recovers them with probability at least $1 - \mathsf{neg}(k)$. It makes a total of $O(\log^3 k)$ queries.*

This lemma enables us to search from a weighted sorted list (with corruption) with few ($\mathsf{polylog}k$) queries. To make our proof self-contained, we will describe how to turn the encoded message in our construction into a weighted sorted list in the proof.

We are now ready to describe our construction.

4.1 Construction with Fixed Failure Probability

4.1.1 Shared Randomness

We first give the construction of an $(n, k = \Omega(n), \delta = O(1), q = \mathsf{polylog}k \log \frac{1}{\varepsilon}, \varepsilon)$ randomized LDC. As before, We start with the construction assuming shared randomness.

Construction 4. *We construct an $(n, k = \Omega(n), \delta = O(1), q = \mathsf{polylog}k \log \frac{1}{\varepsilon}, \varepsilon)$-LDC with randomized encoding.*

Let δ_0, γ_0 be some proper constants in $(0, 1)$.

Let $(\mathsf{Enc}_0, \mathsf{Dec}_0)$ be an asymptotically good (n_0, k_0, d_0) error correcting code for Hamming distance over alphabet $\{0,1\}^{10 \log k}$. Here we pick $n_0 = O(\log \frac{1}{\varepsilon})$, $k_0 = \gamma_0 n_0$, and $d = 2\delta_0 n_0 + 1$.

Let π be a random permutation. And $r_i \in \{0,1\}^{\log k}$ for $i \in [\frac{n_0 k}{k_0 \log k}]$ be $\frac{n_0 k}{k_0 \log k}$ random masks. Both π and r_i's are shared between the encoder and decoder.

Let $\mathbb{C}_0 : \{0,1\}^{2 \log k} \to \{0,1\}^{10 \log k}$ be the asymptotically good code for edit error described previously that can tolerate a δ' fraction of edit error.

The encoding function $\mathsf{Enc} : \{0,1\}^k \to \{0,1\}^n$ is a random function as follows

1. *On input $x \in \{0,1\}^k$, view x as a string over alphabet$\{0,1\}^{\log k}$of length $k/\log k$. We write $x = x_1 x_2 \cdots x_{k/\log k} \in (\{0,1\}^{\log k})^{k/\log k}$;*

2. *Divide x into small blocks of length k_0, s.t. $x = B_1^{(0)} \circ B_2^{(0)} \circ \cdots \circ B_{k/(k_0 \log k)}^{(0)})$. Here, $B_i^{(0)} \in (\{0,1\}^{\log k})^{k_0}$ for $i \in [k/(k_0 \log k)]$ is a concatenation of k_0 symbols in x;*

3. *Encode each block with Enc_0. Concatenate them to get $y^{(1)} = \mathsf{Enc}_0(B_1^{(0)}) \circ \mathsf{Enc}_0(B_2^{(0)}) \circ \cdots \circ \mathsf{Enc}_0(B_{k/(k_0 \log k)}^{(0)})$. Notice that each $\mathsf{Enc}_0(B_1^{(0)})$ is composed of n_0 symbols in $\{0,1\}^{\log k}$. Write $y^{(1)}$ as a string over alphabet $\{0,1\}^{\log k}$, we have $y^{(1)} = B_1^{(1)} \circ B_2^{(1)} \circ \cdots \circ B_{\frac{n_0 k}{k_0 \log k}}^{(1)}$ such that $B_i^{(1)} \in \{0,1\}^{\log k}$;*

4. *Let $N = \frac{n_0 k}{k_0 \log k}$. Permute these N symbols of $y^{(1)}$ with permutation π to get $y^{(2)} = B_1^{(2)} \circ B_2^{(2)} \circ \cdots \circ B_N^{(2)}$ such that $B_{\pi(i)}^{(2)} = B_i^{(1)}$;*

5. *Let $b_i \in \{0,1\}^{\log k}$ be the binary representation of $i \in [N]$. This is fine since $N < k$ when k is larger enough. We compute $B_i^{(3)} = \mathbb{C}_0(b_i \circ (B_i^{(2)} \oplus r_i)) \in \{0,1\}^{10 \log k}$ for each $i \in [T]$. We get $y = B_1^{(3)} \circ B_2^{(3)} \circ \cdots \circ B_N^{(3)} \in (\{0,1\}^{10 \log k})^N$;*

6. *Output y.*

The decoding function $\mathsf{Dec} : [k] \times \{0,1\}^n \to \{0,1\}$ takes two inputs, an index $i_0 \in [k]$ of the message bit the decoder wants to know and $\omega \in \{0,1\}^n$, the received (possibly corrupted) codeword. It proceeds as follows

1. *On input index i_0 and the received codeword ω. We assume the i_0-th bit lies in $B_i^{(0)}$, i.e. the i-th block of x;*

2. *Notice that $\mathsf{Enc}_0(B_i^{(0)}) = B_{(i-1)n_0+1}^{(1)} \circ B_{(i-1)n_0+2}^{(1)} \circ \cdots \circ B_{in_0}^{(1)}$. For each $j \in \{(i-1)n_0 + 1, (i-1)n_0 + 2, \ldots, in_0\}$, search from y to find the block $B_{\pi(j)}^{(3)}$ using the algorithm from [OPC15]. Then we can get a possibly corrupted version of $\mathsf{Enc}_0(B_i^{(0)})$*

3. *Run the decoding algorithm Dec_0 to find out $B_i^{(0)}$. This gives us the i_0-th bit of x.*

Lemma 11. *The above construction 4 gives an efficient $(n, k = \Omega(n), \delta = O(1), q = \mathsf{polylog} k \log \frac{1}{\varepsilon}, \varepsilon)$ randomized LDC against edit error.*

Proof. We first show both the encoding and decoding can be done in polynomial time. Although for the ease of description, we picked the greedily constructed code $\mathbb{C}_0$, which can be inefficient. The codeword size of $\mathbb{C}_0$ is $O(\log k)$. Decoding one block can be finished in time polynomial in k. And for our purpose, we need to encode $O(k/\log k)$ blocks and decode $\mathsf{polylog} n \log \frac{1}{\varepsilon}$ blocks. Thus, the additional time caused by this layer of encoding is polynomial in n. For the rest part, the analysis is similar to that of the Hamming case. Thus, our code is polynomial time. We note that $\mathbb{C}_0$ can be replaced by an efficient code for edit error.

We use the same notation as in our construction. Let $y = B_1^{(3)} \circ B_2^{(3)} \circ \cdots \circ B_N^{(3)}$ be the correct codeword. We denote the length of each block $B_j^{(3)}$ by $b = 10 \log k$ and if we view y as a binary string, the length of y is $N'(= Nb)$, which is $O(k)$. We call the received (possibly corrupted) codeword ω. Since we can always truncate (or pad with 0) to make the length of ω be N', which will only increase the edit distance by a factor no more than 2. Without loss of generality, we assume the length of ω is also N'.

The decoding function Dec has two inputs: an index $i_0 \in [k]$ of the message bit the decoder wants to see and the received codeword ω. The decoder also has access to the shared randomness, which has two parts: a permutation π and N random masks, each of length $\log k$. The first step is to figure out the indices of the blocks it wants to query by using the shared permutation π. Then, we can query these blocks one by one. We assume the i_0-th bit lies in $B_i^{(0)}$. Then, notice that $\mathsf{Enc}_0(B_i^{(0)}) = B_{(i-1)n_0+1}^{(1)} \circ B_{(i-1)n_0+2}^{(1)} \circ \cdots \circ B_{in_0}^{(1)}$. Our goal is to find the block $B_{\pi(j)}^{(3)}$ for each $j \in \{(i-1)n_0 + 1, (i-1)n_0 + 2, \ldots, in_0\}$.

One thing we need to clarify is how to query a block since we do not know the starting point of each block in the corrupted codeword. This is where we use the techniques developed by [OPC15]. In the following, assume we want to find the block $B_i^{(3)}$.

We can view y as a weighted sorted list L of length N such that the i-th element in L is simply $B_i^{(3)}$ with weight b. We show that ω can be viewed as a corrupted list L'. There is a match from y to ω which can be described by a function $f : [N] \to [N] \cup \{\perp\}$. If the i-th bit is preserved after the edit error, then $f(i) = j$ where j is the position of that particular bit in ω. If the i-th bit is deleted, then $f(i) = \perp$.

We say the i-th bit in y is preserved after the corruption if $f(i) \neq \perp$. For each block $B_i^{(3)}$ that is not completely deleted, let $v_i = f(u_i)$, such that u_i is the index of the first preserved bit in the block $B_i^{(3)}$. We say the block $B_i^{(3)}$ is *recoverable* if $\Delta_E(B_i^{(3)}, \omega_{[v_i', v_i'+b-1]}) \le \delta'$ for some index v_i' that is at most $\delta' b$ away from v_i.

Claim 1. *If $\Delta_E(\omega, y) \le \delta$, each block is recoverable with probability at least $1 - \delta/\delta'$.*

Proof. The code $\mathbb{C}_0$ is greedily constructed and resilient to δ' fraction of error. Here, δ is picked smaller than δ'. To make an block corrupted, the adversary needs to produce at least a δ' fraction of edit error in that block. The adversary channel can corrupt at most $\frac{\delta}{\delta'}$ fraction of all blocks. □

The searching algorithm requires sampling some elements from the sorted list L' corresponding to the corrupted codeword ω. We now explain how to do the sampling. We start by first randomly sample a position r and read a substring of $2r + 1$ bits $\omega_{[r-b,r+b]}$ from ω. Then, we try each substring in $\omega_{[r-b,r+b]}$ of length b from left to right until we find first substring that is δ_0-close to some codeword of $\mathbb{C}$ under the normalized edit distance. If we did find such a substring, we decode it and get $b_j \circ B_j^{(2)} \oplus r_j$. Otherwise, output a special symbol $\perp$.

We will regard consecutive intervals in ω as elements in the weighted sorted list where weight of the element is simply the length of the corresponding interval. The correspondence can be described as following.

For a recoverable block $B_i^{(3)}$, again, we let $v_i = f(u_i)$, such that u_i is the index of the first preserved character in block $B_i^{(3)}$. We want to find an interval I_i in ω to represent $B_i^{(3)}$ in the list L'. Since $B_i^{(3)}$ is a codeword of $\mathbb{C}_0$ of length b. Every interval in its first $4/5$ part has at least half of 1's and the last $1/5$ part are all 0's. For any $v \in [v_i - b + 2\delta' b, v_i - 2\delta' b]$, we know $\omega_{[v,v+b-1]}$ can not be δ' close to any codeword in $\mathbb{C}_0$. It is because the last $1/5$ part of $\omega_{[v,v+b-1]}$ contains at least $\delta' b$ 1's. Thus, in the sampling procedure, if $r \in [v_i + 2\delta' b, v_i + b]$, it will return $B_i^{(3)}$. The length of I_i is at least $1 - 2\delta' b$. Let I_i be the maximal inteval containing $[v_i + 2\delta' b, v_i + b]$, such that, if sampling r in I_i, it will output same codeword in $\mathbb{C}$. Also, we argue the length of the inteval I_i is no larger than $1 + 4\delta'$. Since if $r \ge v_i + (1 + 2\delta')b$ or $r \le v_i - 2\delta' b$, any substring of length b in $[r - b, r + b]$ is at least be δ' far from $B_i^{(3)}$ and thus output a different codeword. We note due to the existence of adversary insertion, it is possible to get $B_i^{(3)}$ outside of I_i. But this does not affect our analysis below.

The above method for sampling a block from ω gives us a natural way to interpret ω as a corrupted weighted list L'. The length of ω is equal to the sum of weights of all its elements. For each recoverable block, the interval I_i as described above is an element with weight equal to its length. We note that all I_i's are disjoint. We consider the remaining characters in ω as corrupted elements in the list L. For those elements, the sampling algorithm will output wrong result or $\perp$. Since the interval of the remaining elements can be large, we divide such interval into small intervals each has length no larger than b and consider each small intervals as an element.

Next, we argue the weight fraction of corrupted elements is small. Due to claim 1, there are at least a $1-\delta/\delta'$ fraction of blocks recoverable after the corruption. For each recoverable block $B_i^{(3)}$, we can find a interval I_i in ω with length at least $(1-2\delta_0)b$. Thus, the total weight of uncorrupted elements from the original list L is at least $(1-\delta/\delta')(1-2\delta')N = (1-2\delta-2\delta_0-\delta/\delta_0)N$. Let $\delta_1 = 2\delta+2\delta'+\delta/\delta' = O(\delta)$, we know the total weight of corrupted elements is at most δ_1 fraction of the total weight of list L'. We assume δ_1 is small by properly picking δ and δ'.

By lemma 10, $1-\delta_2$ fraction of elements can be decoded correctly with high probability $(1-\mathsf{neg}(n))$ for some constant $\delta_2 = O(\delta)$. Since the content of each block is protected by random masks. The adversary can learn nothing about the random permutation π used for encoding. Each block is recoverable with same probability. We want to search from y to find all blocks $B_{\pi(j)}^{(3)}$ for each $j \in \{(i-1)n_0+1, (i-1)n_0+2, \dots, in_0\}$ to get a possibly corrupted version of $\mathsf{Enc}_0(B_i^{(0)})$. The proof of success probability follows the same concentration bound as in the Hamming case. Here, the random masks serve the same purpose as in the Hamming case. The content of each block (not the index) is not known to the adversary. Thus, the adversary can learn nothing about the permutation or the message. By picking properly the code $(\mathsf{Enc}_0, \mathsf{Dec}_0)$, using the concentration bound, the fraction of unrecoverable blocks among these n_0 blocks is larger than $1-1.1\delta_2$ with probability at least $1-\varepsilon$.

Finally, we count the number of queries made. Searching one block queries $\mathsf{polylog} k$ symbols from the corrupted codeword ω. We need to search $n_0 = O(\log \frac{1}{\varepsilon})$ blocks. The total number of queries made is $\mathsf{polylog} k \log \frac{1}{\varepsilon}$. □

4.1.2 Oblivious Channel

Now, we consider the oblivious channel. In this model, we assume the adversary can not read the codeword. Same as the Hamming case, we want to send a description of a permutation to the decoder. The decoder can then use description to recover the permutation π which is used to encode the message. We can use the same binary error correcting code from Lemma 2 to encode the description.

We now give the construction.

Construction 5. *We construct an $(n, k = \Omega(n), \delta = O(1), q = \mathsf{polylog} k \log \frac{1}{\varepsilon}, \varepsilon)$-LDC with randomized encoding against the oblivious channel model.*

Let δ_0, γ_0 be some proper constants in $(0,1)$.

Let $(\mathsf{Enc}_0, \mathsf{Dec}_0)$ be an asymptotically good (n_0, k_0, d_0) error correcting code for Hamming distance on alphabet set $\Sigma = \{0,1\}^{10\log k}$. Here we pick $n_0 = O(\log \frac{1}{\varepsilon})$, $k_0 = \gamma_0 n_0$, and $d = 2\delta_0 n_0 + 1$.

Let $\mathbb{C}_0 : \{0,1\}^{2\log k} \to \{0,1\}^{10\log k}$ be the asymptotically good code for edit error as described above that can tolerate a δ' fraction of edit error.

The encoding function $\mathsf{Enc} : \{0,1\}^k \to \{0,1\}^n$ is a random function as follows:

1. *Encode the message x with Construction 4 without doing steps 4,5, and 6. This does not require knowing permutation π. View the sequence we get as a binary string. View the output as a binary string y with length $n/10$;*

2. *Let $N_1 = \frac{n}{20\log k}$. Generate a random seed $r \in \{0,1\}^d$ of lengh d. Here, $d = O(\kappa \log N_1 + \log(1/\varepsilon_\pi)) = O(\log n \log \frac{1}{\varepsilon})$. Use r to sample a $\varepsilon_\pi = \varepsilon/10$-almost $\kappa = O(\log \frac{1}{\varepsilon})$-wise independent random permutation $\pi : [N_1] \to [N_1]$ using Theorem 4;*

3. *Let δ_1 be a properly chosen constant larger than δ. Encoding r with an $(n/20, d, \delta_1 n)$ error correcting code from Lemma 2, we get $z \in \{0,1\}^{n/10}$;*

4. *View $y \circ z$ as a sequence in $\{0,1\}^{n/10}$. Divide $y \circ z \in \{0,1\}^{n/10}$ into small blocks of size $\log k$. Write $y \circ z$ as $y \circ z = B_1 \circ B_2 \circ \cdots \circ B_{n/(10\log k)}$;*

5. *Permute first half of blocks with random permutation π to get $u' = B_1^{(1)} \circ B_2^{(1)} \circ \cdots \circ B^{(1)}_{n/(10\log k)}$ such that $B^{(1)}_{\pi(i)} = B_i$ for $i \le n/(20\log k)$ and $B_i^{(1)} = B_i$ for i larger than $n/(20\log k)$;*

6. *Let b_i be the binary representation of i, for each $i \in n/(10\log k)$, encode $b_i \circ B_i^{(1)}$ with code $\mathbb{C}_0$ to get $B_i^{(2)}$;*

7. *Output $u = B_1^{(2)} \circ B_2^{(2)} \circ \cdots \circ B^{(2)}_{n/(10\log k)}$.*

The decoding function $\mathsf{Dec} : [k] \times \{0,1\}^n \to \{0,1\}$ *takes two inputs, an index $i_0 \in [k]$ of the message bit the decoder wants to know and $\omega \in \{0,1\}^n$, the received (possibly corrupted) codeword. It proceeds as follows:*

1. *Search for at most $O(\log n \log \frac{1}{\varepsilon})$ blocks to decode the random seed r;*

2. *Use r to generate the random permutation π;*

3. *Run the same decoding algorithm as in the Construction 4 on the first half of ω to decode x_{i_0}.*

Lemma 12. *The above construction 4 gives an efficient $(n, k = \Omega(n), \delta = O(1), q = \mathsf{polylog}(n) \log \frac{1}{\varepsilon}, \varepsilon)$ randomized LDC against an oblivious channel with edit error.*

Proof. The proof of efficiency of this construction follows directly from the proof of Lemma 11.

We denote the uncorrupted codeword by u and the received codeword by ω. We assume $\Delta_E(u, \omega) \le \delta$. We first divide it into two parts ω_1 and ω_2 each of equal length such that $\omega = \omega_1 \circ \omega_2$. Similarly, we have $u = u_1 \circ u_2$ with $|u_1| = |u_2| = n/2$. We have $\Delta_E(u_1, \omega_1) \le \delta$ and $\Delta_E(u_2, \omega_2) \le \delta$.

The first step is to decode r. By Lemma 2, we need to query $O(d)$ bits of z to decode r with high probability $1 - \mathsf{neg}(k)$. Thus, we need to query $O(\log k \log \frac{1}{\varepsilon})$ blocks in the second half of ω. We can then use the same searching algorithm described in the proof of Lemma 11 to query each block. Since we randomly choose blocks to query, we can recover r with probability $1 - \mathsf{neg}(k)$ by making $\mathsf{polylog} k \log \frac{1}{\varepsilon}$ queries to ω_2.

Once we know r, we can use it to generate the random permutation π. We run the decoding algorithm from Construction 4 with input i_0 and ω_1. The remaining part is the same as the proof of Lemma 11. Notice that recovering r takes $\mathsf{polylog} k \log \frac{1}{\varepsilon}$ queries. The query complexity for our code is still $\mathsf{polylog} k \log \frac{1}{\varepsilon}$. □

4.2 Construction with Flexible Failure Probability

4.2.1 Shared Randomness

We now give the construction for flexible failure peobability against edit error.

Construction 6. *We construct an $(n, k = \Theta(n/\log n), \delta = O(1))$ randomized LDC with query complexity function $q = \mathsf{polylog} k \log \frac{1}{\varepsilon}$ for any given failure probability ε.*

Let π be a random permutation. And $r_i \in \{0,1\}^{\log k}$ for each $i \in n/(10 \log k)$ be random masks. Both π and r_i's are shared between the encoder and decoder.

With properly picked constant γ_0 and δ_0. Let $(\mathsf{Enc}_0, \mathsf{Dec}_0)$ be a (n_0, k_0, d_0) error correcting code on alphabet set $\{0,1\}^{\log k}$. Here, $n_0 = \gamma_0^{-1} \log n$, $k_0 = \log n$, $d_0 = 2\delta_0 n_0 + 1$, for constant $\gamma_0, \delta_0 \in [0,1]$.

For each $i \in [\log n]$, let Enc_i be the encoding algorithm of an $(n_i, k, \delta_i = O(1), q_i, \varepsilon = 2^{-2^i})$ randomized LDC from Construction 4 without steps 4,5, and 6. Let Dec_i be the correponding decoding algotithm.

Let $\mathbb{C}_1 : \{0,1\}^{3 \log k} \to \{0,1\}^{15 \log k}$ be the asymptotically good code for edit error as described above that can tolerate a δ' fraction of edit error.

Encoding function $\mathsf{Enc} : \{0,1\}^{k = \Omega(n)} \to \{0,1\}^n$ is as follows.

1. *On input $x \in \{0,1\}^k$, compute $y_i = \mathsf{Enc}_i(x)$ for every $i \in [\log n]$. As before, we view $y_i = B_1^i \circ B_2^i \circ \cdots \circ B_N^i$ as the concatenation of N symbols over alphabet $\{0,1\}^{\log k}$;*

2. *Let M be a $\log n \times N$ matrix such that $M[i][j] \in \{0,1\}^{\log k}$ is the j-th symbol of y_i;*

3. *Let M_j be the j-th column of M for each $j \in [N]$. We compute $z_j = \mathsf{Enc}_0(M_j)$. Notice that $z_j = \mathsf{Enc}_0(M_j) \in (\{0,1\}^{\log k})^{n_0}$ is a string of length n_0 over alphabet $\{0,1\}^{\log k}$;*

4. *Let $y^{(0)}$ be the concatenation of z_j's for $j \in [N]$. Then $y^{(0)} = z_1 \circ z_2 \cdots \circ z_N$ is a string of length $n_0 N$ over alphabet $\{0,1\}^{\log k}$. Let $n = 15 \log(k) n_0 N = O(k \log k)$;*

5. *Permute the symbols of $y^{(0)}$ with permutation π to get $y^{(1)} = B_1^{(1)} \circ B_2^{(1)} \circ \cdots \circ B_{n_0 N}^{(1)}$ such that $B_{\pi(i)}^{(1)} = B_i^{(0)}$;*

6. *Since $n_0 N = O(k)$, we assume $n_0 N = ck$ for some constant c. When k is large enough, we assume $ck < k^2$. Let $b_i \in \{0,1\}^{2 \log k}$ be the binary representation of i for each $i \in [n_0 N]$. We compute $B_i^{(2)} = \mathbb{C}_1(b_i \circ (B_i^{(1)} \oplus r_i)) \in \{0,1\}^{10 \log k}$ for each $i \in [n_0 N]$. We get $y = B_1^{(2)} \circ B_2^{(2)} \circ \cdots \circ B_{N'}^{(2)}) \in (\{0,1\}^{15 \log k})^{n_0 N}$;*

7. *Output y as a binary string of length n.*

Decoding function Dec is a randomized algorithm takes three inputs: an index of the bit the decoder wants to see,the received codeword ω, and the desired failure probability ε. It can be described as follows.

1. *On input i_0, ω, ε, find the smallest i such that $2^{-2^i} \leq \varepsilon$. If it cannot be found, then query the whole ω;*

2. *Compute $w = \mathsf{Dec}_i(i_0, y_i)$ but whenever Dec_i wants to query an j-th symbol of y_i, we decode z_j. Notice that $z_j = \mathsf{Enc}_0(M_j)$ contains n_0 symbols over alphabet $\{0,1\}^{\log k}$. Each symbol is encoded in a block in y. The indices of these blocks can be recovered by π. We search each of these n_0 blocks in ω. We can then get the j-th block of y_i from $\mathsf{Dec}_0(z_j)$;*

3. Output w.

Lemma 13. *The above construcion 6 gives an efficient $(n, k = \Theta(n/\log n), \delta = O(1))$ randomized LDC that can recover any bit with probability at least $1 - \varepsilon$ for any constant $\varepsilon \in (0, 1)$, with query complexity $q = \mathsf{polylog} k \log \frac{1}{\varepsilon}$.*

Proof. In the encoding, we need to encode $n_0 N = O(n/\log n)$ blocks with edit error code $\mathbb{C}_0$. And in the decoding, we need decode at most $\mathsf{polylog} n \log \frac{1}{\varepsilon}$ blocks. The time caused by the second layer is polynomial in n. Thus, our construction is efficient.

After a δ fraction of edit error. There is some constant $\delta_1 = O(\delta)$ such tath for at least $1 - \delta_1$ fraction of blocks, we can use the searching algorithm to find and decode them correctly with probability $1 - \mathsf{neg}(n)$. Thus, as is described in the construction, when ever we want to query the j-th symbol of y_i, we fist decode z_j. The symbols of z_j is encoded in n_0 blocks in ω. With π, we know the indices of these n_0 blocks. We can than perform the search as described in the proof of Lemma 11. By picking proper n_0, the decoding of z_j fails with a small probability. The rest of analysis follows directly from the proof of Lemma 11.

For the query complexity, we need to query $O(n_0 \log \frac{1}{\varepsilon})$ blocks. Since $n_0 = O(\log n) = O(\log k)$, the total number of queries made is still $\mathsf{polylog} k \log \frac{1}{\varepsilon}$. □

4.2.2 Oblivious Channel

Our construction for flexible failure probability against oblivious channel combines Construction 5 and Construction 6. More specifically, following the Construction 6, we replace the encoding functions Enc_i for each $i \in [\log n]$ from Construction 4 with the encoding functions from Constructon 5. The analysis follows directly. We omit the details.

5 Discussions and open problems

In this work we initiated a study of locally decodable codes with randomized encoding, and we give constructions for both Hamming errors and edit errors, that significantly improve the rate-query tradeoff of known constructions. Our work leaves many interesting open problems. We list some below.

Question 1: Are there constructions of locally decodable codes with randomized encoding, such that the decoding can succeed without known the randomness of encoding? If so how do such codes perform?

Question 2: If the adversary is allowed to know the randomness used by the encoding, is it still possible to achieve much better rate-query tradeoff by using locally decodable codes with randomized encoding?

Question 3: Is it possible to construct locally decodable codes with a constant rate, while achieving flexible failure probability, by using randomized encoding?

Question 4: Can we further improve the number of queries in our constructions for an oblivious channel?

Finally, even standard locally decodable codes for edit errors are still are poorly understood. For example, for a constant fraction of error and constant alphabet size, the best approach is still due to [OPC15] which needs at least $\mathsf{polylog}(k)$ queries. Can we prove a lower bound for query complexity in this case, or is there an LDC construction with constant number of queries?

References

[Bel15] Djamal Belazzougui. Efficient deterministic single round document exchange for edit distance. *CoRR*, abs/1511.09229, 2015.

[BG16] Boris Bukh and Venkatesan Guruswami. An improved bound on the fraction of correctable deletions. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1893–1901. ACM, 2016.

[CHL$^+$19] K. Cheng, B. Haeupler, X. Li, A. Shahrasbi, and K. Wu. Synchronization strings: Efficient and fast deterministic constructions over small alphabets. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019.

[CIL17] Kuan Cheng, Yuval Ishai, and Xin Li. Near-optimal secret sharing and error correcting codes in AC^0. In *Theory of Cryptography Conference*, pages 424–458. Springer, 2017.

[CJLW18] Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. Deterministic document exchange protocols, and almost optimal binary codes for edit errors. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2018.

[CJLW19] Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. Block edit errors with transpositions: Deterministic document exchange protocols and almost optimal binary codes. In *Proceedings of the 46th International Colloquium on Automata, Languages and Programming (ICALP)*, 2019.

[DGY11] Zeev Dvir, Parikshit Gopalan, and Sergey Yekhanin. Matching vector codes. *SIAM Journal on Computing*, 40(4), 2011.

[Efr09] Klim Efremenko. 3-query locally decodable codes of subexponential length. In *Proceedings of the 41st ACM Symposium on Theory of Computing (STOC)*. ACM, 2009.

[GL16] Venkatesan Guruswami and Ray Li. Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 620–624. IEEE, 2016.

[GS16] Venkatesan Guruswami and Adam Smith. Optimal rate code constructions for computationally simple channels. *Journal of the ACM (JACM)*, 63(4):35, 2016.

[GW17] V. Guruswami and C. Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, April 2017.

[Hae19] Bernhard Haeupler. Optimal document exchange and new codes for small number of insertions and deletions. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2019.

[HS17] Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: codes for insertions and deletions approaching the singleton bound. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 33–46. ACM, 2017.

[HS18] Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: Explicit constructions, local decoding, and applications. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing*, 2018.

[HSV18] Bernhard Haeupler, Amirbehshad Shahrasbi, and Ellen Vitercik. Synchronization strings: Channel simulations and interactive coding for insertions and deletions. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming*, 2018.

[KdW04] Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *Journal of Computer and System Sciences*, 69:395–420, 2004.

[KNR09] Eyal Kaplan, Moni Naor, and Omer Reingold. Derandomized constructions of k-wise (almost) independent permutations. *Algorithmica*, 55(1):113–133, 2009.

[KSY11] Swastik Kopparty, Shubhangi Saraf, and Sergey Yekhanin. High-rate codes with sublinear-time decoding. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*. ACM, 2011.

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the 32ndACM Symposium on Theory of Computing (STOC)*. ACM, 2000.

[OPC15] Rafail Ostrovsky and Anat Paskin-Cherniavsky. Locally decodable codes for edit distance. In *International Conference on Information Theoretic Security*, pages 236–249. Springer, 2015.

[SZ99] Leonard J Schulman and David Zuckerman. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE transactions on information theory*, 45(7):2552–2557, 1999.

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

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