Deterministic Document Exchange Protocols and Almost Optimal Binary Codes for Edit Errors

KUAN CHENG, Center on Frontiers of Computing Studies, Peking University, China
ZHENGZHONG JIN and XIN LI, Department of Computer Science, Johns Hopkins University, USA
KE WU, Computer Science Department, Carnegie Mellon University, USA

We study two basic problems regarding edit errors, document exchange and error correcting codes. Here, two parties try to exchange two strings with length roughly n and edit distance at most k, or one party tries to send a string of length n to another party through a channel that can introduce at most k edit errors. The goal is to use the least amount of communication or redundancy possible. Both problems have been extensively studied for decades, and in this article, we focus on deterministic document exchange protocols and binary codes for insertions and deletions (insdel codes). It is known that for small k (e.g., $k \leq n/4$), in both problems the optimal communication or redundancy size is $\Theta(k \log \frac{n}{k})$. In particular, this implies the existence of binary codes that can correct ε fraction of edit errors with rate $1 - \Theta(\varepsilon \log(\frac{1}{\varepsilon}))$. However, known constructions are far from achieving these bounds.

In this article, we significantly improve previous results on both problems. For document exchange, we give an efficient deterministic protocol with communication complexity $O(k \log^2 \frac{n}{k})$. This significantly improves the previous best-known deterministic protocol, which has communication complexity $O(k^2 + k \log^2 n)$ [4]. For binary insdel codes, we obtain the following results:

(1) An explicit binary insdel code with redundancy $O(k \log^2 \frac{n}{k})$. In particular this implies an explicit family of binary insdel codes that can correct ε fraction of insertions and deletions with rate $1-O(\varepsilon \log^2(\frac{1}{\varepsilon})) = 1-\widetilde{O}(\varepsilon)$. This significantly improves the previous best-known result, which only achieves rate $1-\widetilde{O}(\sqrt{\varepsilon})$ [14], [15], and is optimal up to a $\log(\frac{1}{\varepsilon})$ factor.
(2) An explicit binary insdel code with redundancy $O(k \log n)$. This significantly improves the previous best-known result of Reference [6], which only works for constant k and has redundancy $O(k^2 \log k \log n)$; and that of Reference [4], which has redundancy $O(k^2 + k \log^2 n)$. Our code has optimal redundancy for $k \leq n^{1-\alpha}$, any constant $0 < \alpha < 1$. This is the first explicit construction of binary insdel codes that has optimal redundancy for a wide range of error parameters k.

In obtaining our results, we introduce several new techniques. Most notably, we introduce the notion of *ε-self-matching hash functions* and *ε-synchronization hash functions*. We believe our techniques can have further applications in the literature.

This work is supported by NSF grant CCF-1617713 and NSF CAREER award CCF-1845349. A preliminary version appeared in the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS'18).
Authors' addresses: K. Cheng, Center on Frontiers of Computing Studies, Courtyard No. 5, Jingyuan, Peking University. No. 5 Yiheyuan Road, Haidian District, Beijing, P.R. China 100871; email: ckkcdh@pku.edu.cn; Z. Jin and X. Li, Department of Computer Science, Johns Hopkins University, 3400 North Charles Street, Baltimore, MD, 21218, USA; emails: zjin12@jhu.edu, lixints@cs.jhu.edu; K. Wu, Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15213, USA; email: kew2@andrew.cmu.edu.

0004-5411/2022/11-ART44 $15.00
https://doi.org/10.1145/3561046

CCS Concepts: • **Mathematics of computing → Coding theory;**

Additional Key Words and Phrases: Document exchange, coding for insertions and deletions

ACM Reference format:
Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. 2022. Deterministic Document Exchange Protocols and Almost Optimal Binary Codes for Edit Errors. *J. ACM* 69, 6, Article 44 (November 2022), 39 pages.
https://doi.org/10.1145/3561046

1 INTRODUCTION

Given two strings x, y over some finite alphabet Σ, the edit distance between them $\mathsf{ED}(x, y)$ is defined as the minimum number of edit operations (insertions, deletions, and substitutions) to change x into y. Being one of the simplest metrics, edit distance has been extensively studied due to its wide applications in different areas. For example, in natural language processing, edit distance is used in automatic spelling correction to determine possible corrections of a misspelled word; and in bioinformatics it can be used to measure the similarity between DNA sequences. In this article, we study the general question of recovering from errors caused by edit operations. Note that without loss of generality, we can consider only insertions and deletions, since a substitution can be replace by a deletion followed by an insertion, and this at most doubles the number of operations. Thus, from now on, we will only be interested in insertions and deletions, and we define $\mathsf{ED}(x, y)$ to be the minimum number of such operations required to change x into y.

Insertion and deletion errors happen frequently in practice. For example, they occur in the process of reading magnetic and optical media, in genetic mutation where DNA sequences may change, and in internet protocols where some packets may get lost during routing. Another typical situation where these errors can occur is in distributed file systems, e.g., when a file is stored in different machines and being edited by different people working on the same project. These files then may have different versions that need to be synchronized to remove the edit errors. In this context, we study the following two basic problems regarding insertion and deletion errors:

- *Document exchange.* In this setting, two parties Alice and Bob each holds a string x and y, and we assume that their edit distance is bounded by some parameter k. The goal is to have Alice send a sketch to Bob based on her string x and the edit distance bound k, such that Bob can recover Alice's string x based on his string y and the sketch. Naturally, we would like to require both the message length and the computation time of Alice and Bob to be as small as possible.
- *Error correcting codes.* In this setting, two parties Alice and Bob are linked by a channel where the number of worst-case insertion and deletions is bounded by some parameter k. Given any message, the goal is to have Alice send an encoding of the message to Bob through the channel, so despite any possible insertion and deletion errors that may happen, Bob can recover the correct message after receiving the (possibly modified) codeword. Again, we would like to minimize both the codeword length (or equivalently, the number of redundant bits) and the encoding/decoding time. This is a generalization of the classical error correcting codes for Hamming errors.

It can be seen that these two problems are closely related. In particular, a solution to the document exchange problem can often be used to construct an error correcting code for insertion and deletion errors. In this article, we focus on the setting where the strings have a binary alphabet, arguably the most popular and important setting in computer science. In this case, assume that Alice's string (or the message she wants to send) has length n, then it is known that for small k (e.g., $k \leq n/4$) both the optimal sketch size in document exchange and the optimal number of redundant bits in an error correcting code is $\Theta(k\log(\frac{n}{k}))$, and this is true even for Hamming errors. In addition, both optima can be achieved using exponential time, with the first one using a greedy coloring algorithm and the second one using a greedy sphere packing algorithm (which is essentially what gives the Gilbert-Varshamov bound). See Appendix A for more details.

It turns out that in the case of Hamming errors, both optima (up to constants) can also be achieved efficiently in polynomial time. This is done by using sophisticated linear Algebraic Geometric codes [19]. As a special case, one can use Reed-Solomon codes to achieve $O(k\log n)$ in both problems. However, the situation becomes much harder once we switch to edit errors, and our understanding of these two basic problems lags far behind the case of Hamming errors. We now survey related previous work below.

Document exchange. Historically, Orlitsky [25] was the first one to study the document exchange problem. His work gave protocols for generally correlated strings x, y using a greedy graph coloring algorithm, and in particular he obtained a deterministic protocol with sketch size $O(k\log n)$ for edit errors. However, the running time of the protocol is exponential in k. The main question left there is whether one can design a document exchange protocol that is both communication-efficient and time-efficient.

There has been considerable progress afterwards [11, 21, 22]. Specifically, Irmak et al. [21] gave a randomized protocol that achieves sketch size $O(k\log(\frac{n}{k})\log n)$ and running time $\tilde{O}(n)$. Independently, Jowhari [22] also obtained a randomized protocol with sketch size $O(k\log^2 n\log^* n)$ and running time $\tilde{O}(n)$. A recent work by Chakraborty et al. [9] introduced a clever randomized embedding from the edit distance metric to the Hamming distance metric, and thus obtained a protocol with sketch size $O(k^2\log n)$ and running time $\tilde{O}(n)$. Using the embedding in Reference [9], Belazzougui and Zhang [5] gave an improved randomized protocol with sketch size $O(k(\log^2 k+\log n))$ and running time $\tilde{O}(n+\mathrm{poly}(k))$, where the sketch size is asymptotically optimal for $k = 2^{O(\sqrt{\log n})}$.

All of the above protocols, except the exponential time protocol of Orlitsky [25], are however, randomized. In practice, a deterministic protocol is certainly more useful than a randomized one. Thus, one natural and important question is whether one can construct a deterministic protocol for document exchange with small sketch size (e.g., polynomial in $k\log n$) and efficient computation. This question is also important for applications in error correcting codes, since a randomized document exchange protocol is not very useful in designing such codes. It turns out that this question is quite tricky, and no such deterministic protocols are known even for $k > 1$ until the work of Belazzougui [4] in 2015, where he gave a deterministic protocol with sketch size $O(k^2 + k\log^2 n)$ and running time $\tilde{O}(n)$.

Error correcting codes. Error correcting codes are fundamental objects in both theory and practice. Starting from the pioneering work of Shannon, Hamming, and many others, error correcting codes have been intensively studied in the literature. This is true for both standard Hamming errors such as symbol corruptions and erasures, and edit errors such as insertions and deletions. While the study of codes against standard Hamming errors has been a great success, leading to a near complete knowledge and a powerful toolbox of techniques together with explicit constructions that match various bounds, our understanding of codes for insertion and deletion errors (insdel codes for short) is still rather poor. Indeed, insertion and deletion errors are strictly more general than Hamming errors, and the study of codes against such errors has resisted progress for quite some time, as demonstrated by previous work that we discuss below.

Since insertion and deletion errors are strictly more general than Hamming errors, all the upper bounds on the rate of standard codes also apply to insdel codes. Moreover, by using a similar sphere packing argument, similar lower bounds on the rate (such as the Gilbert-Varshamov bound) can also be shown. In particular, one can show (e.g., Reference [23]) that for binary codes, to encode a message of length n against k insertion and deletion errors with $k \leq n/2$, the optimal number of redundant bits is $\Theta(k \log(\frac{n}{k}))$; and to protect against ε fraction of insertion and deletion errors, the optimal rate of the code is $1 - \Theta(\varepsilon \log(\frac{1}{\varepsilon}))$.However, if the alphabet of the code is large enough, then one can potentially recover from an error fraction approaching 1 or achieve the singleton bound: a rate $1 - \varepsilon$ code that can correct ε fraction of insertion and deletion errors.

However, achieving these goals have been quite challenging. In 1966, Levenshtein [23] first showed that the Varshamov-Tenengolts code [29] can correct one deletion with roughly $\log n$ redundant bits, which is optimal. Since then, many constructions of insdel codes have been given, but all constructions are far from achieving the optimal bounds. In fact, even correcting two deletions requires $\Omega(n)$ redundant bits, and even the first explicit asymptotically good insdel code (a code that has constant rate and can also correct a constant fraction of insertion and deletion errors) over a constant alphabet did not appear until the work of Schulman and Zuckerman in 1999 [26], who gave such a code over the binary alphabet. We refer the reader to the survey by Mercier et al. [24] for more details about the extensive research on this topic.

In the past few years, there has been a series of work trying to improve the situation for both the binary alphabet and larger alphabets. Specifically, for larger alphabets, a line of work by Guruswami et al. [7, 8, 14, 15] constructed explicit insdel codes that can correct $1 - \varepsilon$ fraction of errors with rate $\Omega(\varepsilon^5)$ and alphabet size $\mathsf{poly}(1/\varepsilon)$; and for a fixed alphabet size $t \geq 2$ explicit insdel codes that can correct $1 - \frac{2}{t+\sqrt{t}} - \varepsilon$ fraction of errors with rate $(\varepsilon/t)^{\mathsf{poly}(1/\varepsilon)}$. These works aim to tolerate an error fraction approaching 1 by using a sufficiently large alphabet size. Another line of work by Haeupler et al. [10, 17, 18] introduced and constructed a combinatorial object called *synchronization string*, which can be used to transform standard error correcting codes into insdel codes, at the price of increasing the alphabet size. Using explicit constructions of synchronization strings, Reference [17] achieved explicit insdel codes that can correct δ fraction of errors with rate $1 - \delta - \varepsilon$ (hence approaching the singleton bound), although the alphabet size is exponential in $\frac{1}{\varepsilon}$.

For the binary alphabet, which is the focus of this article, it is well known that no non-trivial code can tolerate an error fraction larger than 1/2 or achieve the singleton bound. Instead, the major goal here is to construct explicit insdel codes for some small fraction (ε) or some small number (k) of errors that can achieve the optimal rate of $1 - \Theta(\varepsilon \log(\frac{1}{\varepsilon}))$ or the optimal redundancy of $\Theta(k \log(\frac{n}{k}))$, which is analogous to achieving the Gilbert-Varshamov bound for standard error correcting codes. Slightly less ambitiously, one can ask to achieve redundancy $O(k \log n)$, which is optimal when $k \leq n^{1-\alpha}$ for any constant $\alpha > 0$ and easy to achieve in the case of Hamming errors by using Reed-Solomon codes. In this context, Guruswami et al. [14, 15] constructed explicit insdel codes that can correct ε fraction of errors with rate $1 - \tilde{O}(\sqrt{\varepsilon})$, which is the best possible by using code concatenation. For any fixed constant k, another work by Brakensiek et al. [6] constructed an explicit insdel code that can encode an n-bit message against k insertions/deletions with $O(k^2 \log k \log n)$ redundant bits, which is asymptotically optimal when k is a fixed constant. We remark that the construction in Reference [6] only works for constant k and does not give anything when k becomes larger (e.g., $k = \log n$). Finally, using his deterministic document exchange protocol, Belazzougui [4] constructed an explicit insdel code that can encode an n-bit message against k insertions/deletions with $O(k^2 + k \log^2 n)$ redundant bits. In summary, there remains a huge gap between the known constructions and the optimal bounds in the case of a binary alphabet. In particular, even achieving $O(k \log n)$ redundancy has been far out of reach.

1.1 Our Results

In this article, we significantly improve the situation for both document exchange and binary insdel codes. Our new constructions of explicit insdel codes are actually almost optimal for a wide range of error parameters k. First, we have the following theorem that gives an improved deterministic document exchange protocol:

Theorem 1. *There exists a single round deterministic protocol for document exchange with communication complexity (sketch length)* $O(k \log^2 \frac{n}{k})$, *time complexity* $\mathrm{poly}(n)$, *where* n *is the length of the string and* k *is the edit distance upper bound.*

Note that this theorem significantly improves the sketch size of the deterministic protocol in Reference [4], which is $O(k^2 + k \log^2 n)$. In particular, our protocol is interesting for k up to $\Omega(n)$ while the protocol in Reference [4] is interesting only for $k < \sqrt{n}$.

We can use this theorem to get improved binary insdel codes that can correct ε fraction of errors.

Theorem 2. *There exists a constant* $0 < \alpha < 1$ *such that for any* $0 < \varepsilon \leq \alpha$ *there exists an explicit family of binary error correcting codes with codeword length* n *and message length* m, *that can correct up to* $k = \varepsilon n$ *edit errors with rate* $m/n = 1 - O(\varepsilon \log^2 \frac{1}{\varepsilon})$.

Note that the rate of the code is $1 - O(\varepsilon \log^2 \frac{1}{\varepsilon}) = 1 - \widetilde{O}(\varepsilon)$, which is optimal up to an extra $\log(\frac{1}{\varepsilon})$ factor. This significantly improves the rate of $1 - \widetilde{O}(\sqrt{\varepsilon})$ in References [14, 15].

For the general case of k errors, the document exchange protocol also gives an insdel code.

Theorem 3. *For any* $n, k \in \mathbb{N}$ *with* $k \leq n/4$, *there exists an explicit binary error correcting code with message length* n, *codeword length* $n + O(k \log^2 \frac{n}{k})$ *that can correct up to* k *edit errors.*

When k is small, e.g., $k = n^{\alpha}$ for some constant $\alpha < 1$, the above theorem gives $O(k \log^2 n)$ redundant bits. Our next theorem shows that we can do better, and in fact, we can achieve redundancy $O(k \log n)$, which is asymptotically optimal for small k.

Theorem 4. *For any* $n, k \in \mathbb{N}$, *there exists an explicit binary error correcting code with message length* n, *codeword length* $n + O(k \log n)$ *that can correct up to* k *edit errors.*

Note that in this theorem, the number of redundant bits needed is $O(k \log n)$, which is asymptotically optimal for $k \leq n^{1-\alpha}$, any constant $0 < \alpha < 1$. This significantly improves the construction in Reference [6], which only works for constant k and has redundancy $O(k^2 \log k \log n)$, and the construction in Reference [4], which has redundancy $O(k^2 + k \log^2 n)$. In fact, this is the first explicit construction of binary insdel codes that have optimal redundancy for a wide range of error parameters k, and this brings our understanding of binary insdel codes much closer to that of standard binary error correcting codes. In particular, for the first time in history, we obtain an explicit insdel code comparable to what we can achieve for Hamming errors nearly 60 years ago using the popular Reed-Solomon code.

Remark 1. In all our insdel codes, both the encoding function and the decoding function run in time $\mathrm{poly}(n)$.

Remark 2. Our constructions can be easily extended to larger alphabets, so we just focus on the binary alphabet in this article.

Independent work. In a recent independent work [16], Haeupler also studied the document exchange problem and binary error correcting codes for edit errors. Specifically, he also obtained a deterministic document exchange protocol with sketch size $O(k \log^2 \frac{n}{k})$, which leads to an error correcting code with the same redundancy, thus matching our Theorems 1, 2, and 3. He further gave a randomized document exchange protocol that has optimal sketch size $O(k \log \frac{n}{k})$. However, for small k, our Theorem 4 gives a much better error correcting code. Our code is optimal for $k \leq n^{1-\alpha}$, any constant $0 < \alpha < 1$, and thus better than the code given in Reference [16].

Subsequent work. Roughly one year after our results became public and presented at FOCS'2018, a subsequent work [28] gave an alternative construction of error correcting codes for k deletion errors with redundancy $8k \log n + o(\log n)$, which has a better constant before $k \log n$ compared to our result. However, their construction has *exponential* encoding and decoding time of $O(n^{2k+1})$. Their techniques are based on a generalization of the Varshamov-Tenengolts code [29].

1.2 Overview of the Techniques

In this section, we provide a high-level overview of the ideas and techniques used in our constructions. We start with the document exchange protocol.

Document exchange. Our starting point is the randomized protocol by Irmak, Mihaylov, and Suel [21], which we refer to as the IMS protocol. The protocol is one round, but Alice's algorithm to generate the message proceeds in $L = O(\log(\frac{n}{k}))$ levels. In each level Alice computes some sketch about her string x, and her final message to Bob is the concatenation of the sketches. After receiving the message, Bob's algorithm also proceeds in L levels, where in each level he uses the corresponding sketch to recover part of x.

More specifically, recall that Alice generates the message to Bob in $L = O(\log(\frac{n}{k}))$ levels. To do this, in the first level Alice divides her string into $\Theta(k)$ blocks where each block has size $O(\frac{n}{k})$, and in each subsequent level every block is divided evenly into two blocks, until the final block size becomes $O(\log n)$. Therefore, this takes $L = O(\log(\frac{n}{k}))$ levels. Using shared randomness, for any $j \in [L]$, in level j Alice picks a set of random hash functions, one for each block (note that there are $\Theta(k2^{j-1})$ blocks in level j), which outputs $O(\log n)$ bits. Alice then computes the hash values of these functions on each block. In the first level, Alice's sketch is just the concatenation of the $O(k)$ hash values. In all levels j where $j > 1$, Alice treats the hash values (each with $O(\log n)$ bits) as a string over an alphabet of size poly(n) and obtains the sketch for this level by computing the redundant symbols of a systematic error correcting code (e.g., the Reed-Solomon code) that encodes the above string, where the code can correct $O(k)$ erasures and symbol corruptions. Note that this sketch has size $O(k \log n)$, and thus the total sketch or message size is $O(k \log n \log(\frac{n}{k}))$, because the final message from Alice to Bob is simply the concatenation of the sketches from all levels.

On Bob's side, again his recovering algorithm proceeds in $L = O(\log(\frac{n}{k}))$ levels, where in each level $j \in [L]$ Bob uses the sketch from Alice in the corresponding level to recover two things: Alice's hash values in level j, and part of the string x. To do this, Bob always maintains a string $\tilde{x}$ that is the partially corrected version of x. Initially $\tilde{x}$ is set to be the empty string, and in each level $j \in [L]$ Bob tries to use his string y to fill $\tilde{x}$, as follows: First, Bob recovers all the hash values of Alice in level j (notice that the hash values of the first level are directly sent to Bob). Suppose Bob has successfully recovered all the hash values in level j (which is trivially true for $j = 1$), Bob then tries to match every block of Alice's string in level j with one substring of the same length in his string y, by finding such a substring with the same hash value (note that Alice and Bob both know the hash functions by using shared randomness). We say such a match is bad if the substring Bob finds is *not* the same as Alice's block (i.e., a hash collision). The key idea here is that if the hash functions output $O(\log n)$ bits, and they are chosen independently randomly, then with high probability a bad match only happens if the substring Bob finds contains at least one edit error. Moreover, Bob can find at least $l_j - k$ matches, where l_j is the number of blocks in level j, since there are at least this number of blocks without any error. Bob then uses the matched substrings in y to fill the corresponding level-j blocks in $\tilde{x}$ and leaves the unmatched blocks blank. From the above discussion, one can see that there are at most k unmatched blocks and at most k mismatched bad blocks. Therefore, in level $j + 1$ when both parties divide every block evenly into two blocks,

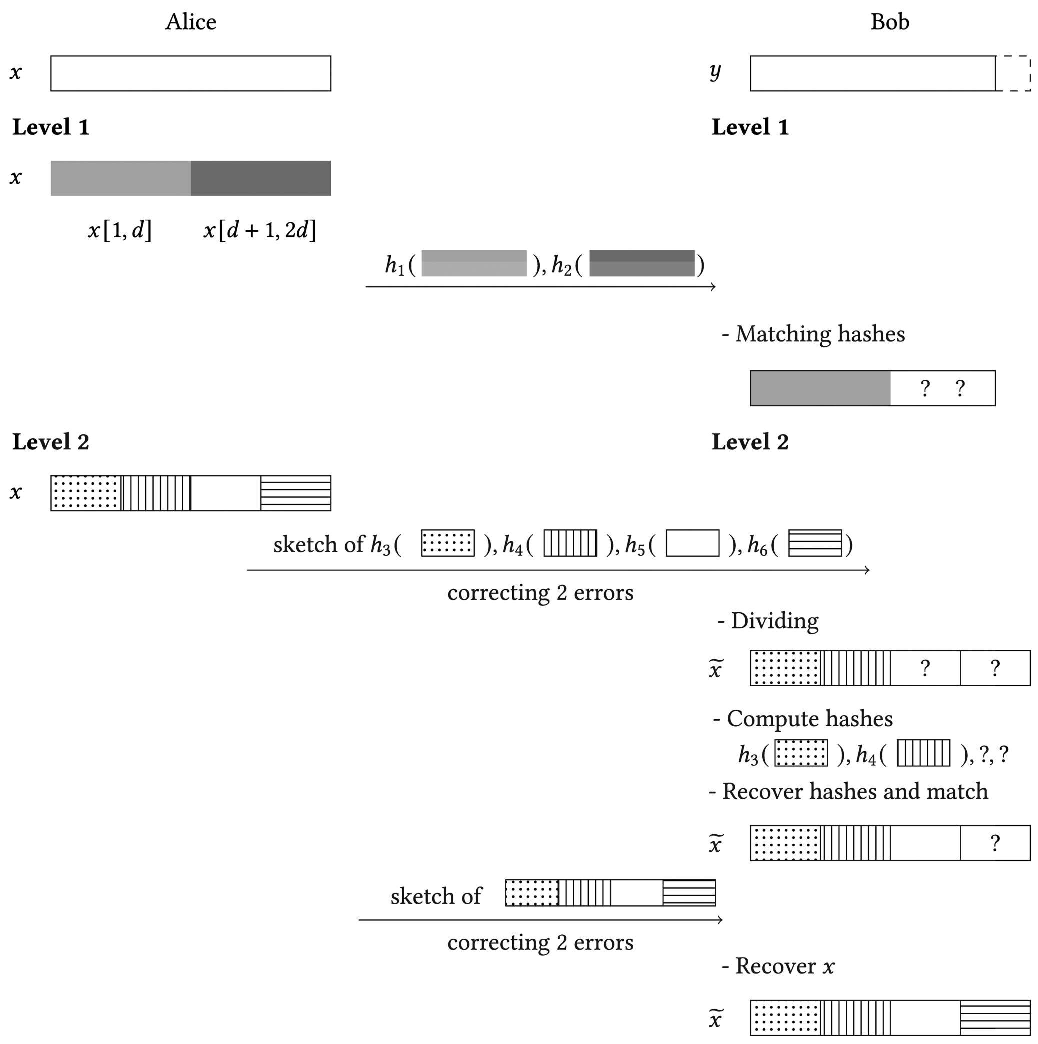

Fig. 1. An example of IMS protocol assuming no hash collisions and edit distance $k = 1$. Bob's string y is obtained from one deletion of the last bit of x. A block with question mark means that this block is unmatched.

x and $\tilde{x}$ have at most $4k$ different blocks. This implies that there are also at most $4k$ different hash values in level $j + 1$, and hence Bob can indeed correctly recover all the hash values of Alice in level $j+1$ using the redundancy of the error correcting code. See Figure 1 for an example for $k = 1$, assuming no hash collisions.

Our deterministic protocol for document exchange is a derandomized version of the IMS protocol, with several modifications. First, we observe that the IMS protocol as we presented above, can already be derandomzied. This is because that to ensure a bad match only happens if the substring Bob finds contains at least one edit error, we in fact just need to ensure that under any hash function, no *block* of x can have a collision with a *substring* of the same length in x itself. We emphasize one subtle point here: Alice's hash function is applied to a *block* of her string x, while when trying to fill $\tilde{x}$, Bob actually checks every *substring* of his string y, since edit errors can cause position shifts. Therefore, we need to consider hash collisions between blocks of x and substrings of x. If the hash functions are chosen independently randomly, then such a collision happens with probability $1/\mathsf{poly}(n)$, and thus by a union bound with high probability no collision happens. However, notice that if we write out the outputs of all hash functions on all inputs, then any collision is only concerned with two outputs that consist of $O(\log n)$ bits. Thus, it is enough to use $O(\log n)$-wise independence to generate these outputs. To further save the random bits used, we can instead use an almost κ-wise independent sample space with $\kappa = O(\log n)$ and error $\varepsilon = 1/\mathsf{poly}(n)$. Using for example the construction by Alon et al. [2], this results in a total of $O(\log n)$ random bits (the seed), and thus Alice can exhaustively search for a fixed set of hash functions in polynomial time. Now in each level j, Alice's sketch will also include the specific seed that is used to generate the hash functions, which has $O(\log n)$ bits, as a description of the hash functions used in level j. Note that this only adds $O(\log n \log \frac{n}{k})$ to the final sketch size. Bob's algorithm is essentially the same as before, except now in each level j he also needs to use the description of the hash functions to compute the hash functions.

The above construction gives a deterministic document exchange protocol with sketch size $O(k \log n \log \frac{n}{k})$, but our goal is to further improve this to $O(k \log^2 \frac{n}{k})$. We achieve this by reducing the output size of all hash functions from $O(\log n)$ bits to $O(\log(n/k))$ bits by using a relaxed version of hash functions with nice "self-matching" properties. To motivate our construction, first observe that in each level j, when Bob tries to match every block of Alice's string with one substring of the same length in his string y, it is not only true that Bob can find a matching of size at least $l_j - k$ (where l_j is the number of blocks in level j), but also true that Bob can find a *monotone* matching of at least this size. A monotone matching here means a matching that does not have edges crossing each other. In this monotone matching, there are at most k bad matches caused by edit errors, and thus there exists a *self-matching* between x and itself with size at least $l_j - 2k$. In the previous construction, we in fact ensure that all these $l_j - 2k$ matches are correct. To achieve better parameters, we instead relax this condition and only require that at most k of these self-matches are bad. Note that, if this is true, then the total number of different blocks between x and $\tilde{x}$ is still $O(k)$ and thus, we can again use an error correcting code against $O(k)$ errors to send the redundancy of hash values in level $j+1$.

This relaxation motivates us to introduce *ε-self-matching hash functions*, which is similar in spirit to ε-self-matching strings introduced in Reference [17]. Formally, we have the following definitions:

Definition 1 (Monotone Matching). For every $n, n', t, p, q \in \mathbb{N}, q \le p, n = n'p$, any hash functions $h_1, h_2, \ldots, h_{n'}$ where $\forall i \in [n'], h_i : \{0,1\}^p \to \{0,1\}^q$, given two strings $x, y \in \{0,1\}^n$, a monotone matching of size t between x, y under hash functions $h_1, \ldots, h_{n'}$ is a sequence of pairs of indices $w = ((i_1, j_1), (i_2, j_2), \ldots, (i_t, j_t)) \in ([n] \times [n])^t$ s.t. $1 \le i_1 < i_2 < \cdots < i_t \le n, \forall l \in [t], i_l \pmod p = 1, j_1 + p - 1 < j_2, \ldots, j_{t-1} + p - 1 < j_t, j_t + p - 1 \le n$ and $\forall l \in [t], h_{i_l}(x[i_l, i_l + p)) = h_{i_l}(y[j_l, j_l + p))$. When $h_1, \ldots, h_{n'}$ are clear from the context, we simply say that w is a monotone matching between x, y.

For $l \in [t]$, if $x[i_l, i_l + p) = y[j_l, j_l + p)$, we say (i_l, j_l) is a good match, otherwise, we say it is a bad match. We say w is a correct matching if all matches in w are good.

If x and y are the same in terms of their binary expression, then w is called a self-matching.

For simplicity, in the rest of the article, when we say a matching w, we always mean a monotone matching.

Definition 2 (ε-self-matching Hash Function). Let $p, q, n, n' \in \mathbb{N}$ be such that $n = n'p$. For any $0 < \varepsilon < 1$ and $x \in (\{0,1\}^p)^{n'}$, we say that a sequence of hash functions $h_1, h_2, \ldots, h_{n'}$ where

$\forall i \in [n'], h_i : \{0,1\}^p \to \{0,1\}^q$ is a sequence of ε-self-matching hash functions with respect to x, if any matching between x and $y \in \{0,1\}^n$ under $h_1, h_2, \ldots, h_{n'}$, where y is the binary expression of x, has at most εn bad matches.

The advantage of using ε-self-matching hash functions is that the output size of the hash functions can be reduced. Specifically, we show that a sequence of ε-self-matching hash functions exists with output size $O(\log(1/\varepsilon))$ bits, when the block (input) size is at least $c \log(1/\varepsilon)$ bits for some constant $c > 1$. Furthermore, we can generate such a sequence of ε-self-matching hash functions with high probability by again using an almost κ-wise independent sample space with $\kappa = O(kb_i)$, where b_i is the current block (input) size, and error $\varepsilon = 1/\mathsf{poly}(n)$. The idea is that in a monotone matching with εn bad matches, we can divide the matching gradually into small intervals such that at least one small interval will have the same fraction of bad matches. Thus, to ensure the ε-self-matching property, we just need to make sure every small interval does not have more than ε fraction of bad matches, and this is enough by using the almost κ-wise independent sample space.

As discussed above, we need to ensure that there are at most k bad matches in a self-matching, thus, we set $\varepsilon = \frac{k}{n}$. Consequently, now the output of the hash functions only has $O(\log(n/k))$ bits instead of $O(\log n)$ bits. Next, in each level j, to get optimal sketch size, instead of using the Reed-Solomon code, we will be using an Algebraic Geometric code [19] that has redundancy $O(k \log \frac{n}{k})$. The almost κ-wise independent sample space in this case again uses only $O(\log n)$ random bits, so in each level Alice can exhaustively search the correct hash functions in polynomial time and include the $O(\log n)$ bits of description in her sketch. This gives Alice's algorithm with total sketch size $O(k \log^2 \frac{n}{k})$. On Bob's side, we need another modification: In each level after Bob recovers all the hash values, instead of simply searching for a match for every block, Bob runs a dynamic programming to find the longest monotone matching between his string y and the sequence of hash values. He then fills the blocks of $\tilde{x}$ using the corresponding substrings of matched blocks.

Error correcting codes. Our deterministic document exchange protocol can be used to directly give an insdel code for k edit errors. The idea is that to encode an n-bit message x, we can first compute a sketch of x with size r and then encode the small sketch using an insdel code against $4k$ edit errors. Since the sketch size is larger than k, we can use an asymptotically good code such as the one by Schulman and Zuckerman [26], which results in an encoding size of $n_0 = O(r)$. The actual encoding of the message is then the original message concatenated with the encoding of the sketch.

To decode, we can first obtain the sketch by looking at the last $n_0 - k$ bits of the received string. The edit distance between these bits and the encoding of the sketch is at most $4k$, and thus, we can get the correct sketch from these bits. Now, we look at the bits of the received string from the beginning to index $n + k$. The edit distance between these bits and x is at most $3k$, thus if r is a sketch for $3k$ edit errors, then we will be able to recover x by using r. This gives our first insdel code with redundancy $O(k \log^2 \frac{n}{k})$.

We now describe our second insdel code, which has redundancy $O(k \log n)$ and uses many more interesting ideas. The basic idea here is again to compute a sketch of the message and encode the sketch, as we described above. However, we are not able to improve the sketch size of $O(k \log^2 \frac{n}{k})$ in general. Instead, our first observation here is that if the n-bit message is a *uniform random* string, then we can actually do better. Thus, we will first describe how to come up with a sketch of size $O(k \log n)$ for a uniform random string and then use this to get an encoding for any given n-bit string.

To warm up, we first explain a simple algorithm to compute a sketch of size $O(k \log^2 n)$ in this case. A uniform random string has many nice properties. In particular, for some $B = \Theta(\log n)$, one can show that with high probability, every length B substring in a uniform random string is *distinct*. If this property holds (which we refer to as the *B-distinct property*), then Alice can compute a sketch as follows: First create a vector V of length $2^B = \mathsf{poly}(n)$, where in each entry indexed by the string $s \in \{0,1\}^B$, Alice looks at the substring s in x and records the following in the entry indexed by s: the bit left to it and the bit right to it. We also need three special symbols, one to indicate the case of no left bit, one to indicate the case of no right bit, and one to indicate the case that there is no such string s in x. Thus, it is enough to use an alphabet of size 8 for each entry. Similarly, Bob creates a vector V' from his string y. Notice that the entries in V have no collisions, since we assume that Alice's string is B-distinct, while some entries in V' may have collisions due to edit errors, in which case Bob just treats it as there is no corresponding substring in y. One can then show that V and V' differ in at most $O(kB) = O(k \log n)$ entries. Now Alice can use the Reed-Solomon code to send a redundancy of size $O(k \log^2 n)$, and Bob can recover the correct vector V. Bob can then recover the string x by picking an entry in V and growing the string on both ends gradually until obtaining the full string x.

We now explain how to reduce the sketch size. In the above approach, V and V' can differ in $O(k \log n)$ entries, since Alice is looking at *every* substring of length B in x. To improve this, instead, we will have Alice first partition her string into several blocks and then just look at the substrings corresponding to each block. Ideally, we want to make sure that each block has length at least B so again all blocks are distinct. Alice then creates the vector V of length 2^B by using the length B-prefix of each block as the index in V, and for each entry in V Alice will record some information. Bob will do the same thing using his string y to create another vector V', and we will argue that V and V' do not differ much so Alice can send some redundancy information to Bob, and Bob can recover the correct V based on V' and the redundancy information.

However, the partitions need to be done carefully. For example, we cannot just partition both strings sequentially into blocks of size some $T \geq B$, since, if so, then a single insertion/deletion at the beginning of x could result in the case where all blocks of x and y are distinct. Instead, we will choose a specific string p with length s, for some parameter s that we will choose appropriately. We call this string p a *pattern*, and we will use this pattern to divide the blocks in x and y. More specifically, in our construction, we will simply choose $p = 1 \circ 0^{s-1}$, the string with a 1 followed by $s-1$ 0's. We use p to divide the string x as follows: Whenever p appears as a substring in x, we call the index corresponding to the first bit of p in x a *p-split point*. The set of p-split points then naturally gives a partition of the string x (and also y) into blocks. We note that the idea of using patterns and split points is also used in Reference [6]. However, there the construction uses $2k+1$ patterns and takes a majority vote, which is why the sketch has a k^2 factor. Here, instead, we use a single pattern, and thus, we can avoid the k^2 factor.

We now describe how to choose the pattern length s. For the blocks of x and y obtained by the split points, we certainly do not want the block size to be too large. This is because if there are large blocks, then an adversary can create errors in these blocks, and large blocks need longer sketches to recover from error. At the same time, we do not want the block size to be too small either. This is because if the block size is too small, then some of the blocks may actually be the same, while we would like to keep all blocks of x to be distinct. In particular, we would like to keep the size of every block in x to be at least B. If x is a uniformly random string, then the pattern p appears with probability 2^{-s} for any length s substring, and thus the expected distance between two consecutive appearances of p is 2^s. Moreover, one can show that with high probability any interval of length some $O(s2^s \log n)$ contains a p-split point. We will call this *property* 1. If this property holds, then we can argue that every block of x has length $O(s2^s \log n)$.

To ensure that each block is not too short, we simply look at a p-split point and the immediately next p-split point after it. If the distance between these two split points is less than $2^s/2$, then we just ignore the first p-split point. In other words, we change the process of dividing x into blocks so

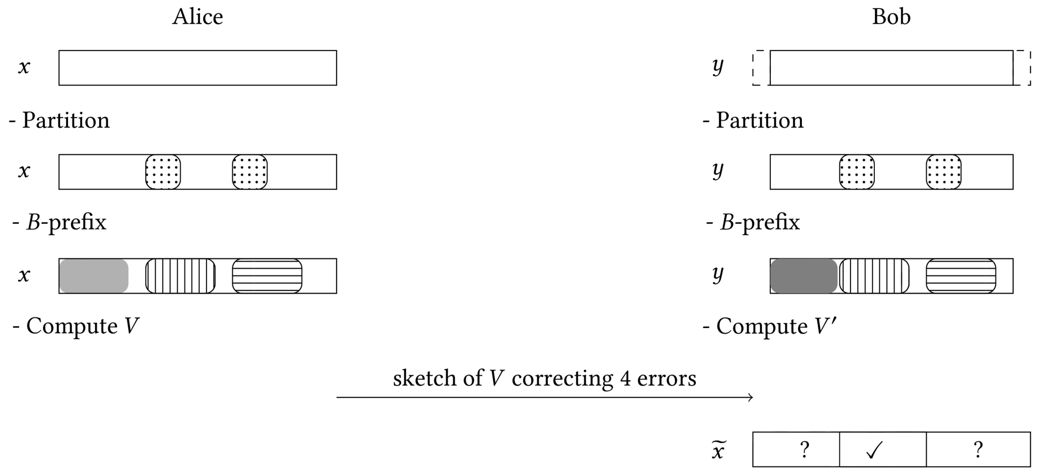

Fig. 2. An example of Stage 1 with edit distance $k = 2$. Bob's string y is gained from one deletion of the first bit and one deletion of the last bit from x. The dotted rounded corner block represents the pattern p, while other rounded corner blocks represent the corresponding B-prefixes. At the end of Stage 1, Bob recovers a string $\widetilde{x}$, with the second block correctly filled and other blocks unmatched.

we will only use a p-split point if the next p-split point is at least $2^s/2$ away from it, and we call such a p-split point a good p-split point. We again show that if x is uniformly random, then with high probability every block (recall that any block starts with a p-split point) of length some $O(2^s \log n)$ contains a good p-split point. We call this *property* 2. Combined with the previous paragraph, we can now argue that with high probability every interval of length some $O(s2^s \log n) + O(2^s \log n) = O(s2^s \log n)$ contains a p-split point that we will choose. By setting $s = \log\log n + O(1)$, we can ensure that every block of x has length at least $B = \Theta(\log n)$ and at most $O(s2^s \log n) = \text{poly}\log(n)$.

Now for each block of x, Alice creates in the vector V an entry indexed by the length B-prefix of this block. The entry contains the length of this block and the length B-prefix of the next block, which has a total of $O(\log n)$ bits. Similarly, Bob also creates a vector V'. We show that our approach of choosing p-split points can ensure that V and V' differ in at most $O(k)$ entries, thus Alice can send a sketch with $O(k \log n)$ bits to Bob (using the Reed-Solomon code) and Bob can recover the correct V. The structure of V guarantees that Bob can learn the correct order of the length B-prefixes of all Alice's blocks and the lengths of all Alice's blocks. At this point Bob will again try to fill a string $\tilde{x}$, where for each of Alice's blocks Bob searches for a substring with the same length B-prefix and the same length. We show that after this step, at most $O(k)$ blocks in $\tilde{x}$ are incorrectly filled or missing. This completes stage 1 of the sketch. See Figure 2.

We now move to stage 2, where Bob correctly recovers the $O(k)$ blocks in $\tilde{x}$ that are incorrectly filled or missing. One way to do this is by noticing that every block has size at most $\text{poly}\log(n)$, thus, we can use the derandomized IMS protocol we described before, which will last $O(\log\log n)$ levels and thus have sketch size $O(k \log \frac{n}{k} \log\log n)$ (since we start with block size $\text{poly}\log(n)$). However, our goal is to do better and achieve sketch size $O(k \log n)$.

To achieve this, we modify the IMS protocol so in stage 2, in each level, we do not just divide every block into 2 smaller blocks. Instead, we divide every block evenly into $\log^{0.4} n$ smaller blocks. We continue this process until the final block size is $O(\log n)$, and thus this only takes $O(1)$ levels. In each level, we do something similar to our deterministic document exchange protocol: Alice sends a description of a sequence of hash functions to Bob, together with some redundancy of the hash values of her blocks. Bob recovers the correct hash values and uses a dynamic programming to find the longest monotone matching between substrings of y and the blocks of x, using the recovered hash values of blocks of x and applying the hashing functions on substrings of y. Bob then tries to fill the blocks of $\tilde{x}$ by using the matched substrings of y.

For the above approach to work, we need to ensure three things: First, Alice's description of the hash functions (sent to Bob) should be short, ideally only $O(k \log n)$ bits. Recall that the description of the hash functions is a short string (e.g., the seed of a pseudorandom generator in our derandomized IMS protocol) for Bob to recover and thus compute the hash functions. This is not to be confused with the outputs of the hash functions. Second, as in the derandomized IMS protocol, Alice cannot simply send all hash outputs, since this will be too long. Instead she can only send some redundancy of the hash values (in the derandomized IMS protocol, we use the redundancy of a systematic algebraic geometry code), and we need to make sure this redundancy only has $O(k \log n)$ bits. Finally, as in the derandomized IMS protocol, in each level of stage 2, Bob needs to first recover the correct hash values of Alice's blocks in this level and then compute a longest monotone matching between these blocks and substrings of y, under the recovered hash values and hash functions. Here, as before, we need to make sure that the matching Bob finds contains at most $O(k)$ unmatched blocks and mismatched blocks. This ensures that we can prove the correctness of the protocol by induction, and that in the last level, there are only $O(k)$ blocks in $\tilde{x}$ that are different from x. A sketch correcting $O(k)$ blocks of Hamming errors is sufficient to finish the protocol.

We ensure the three properties as follows: For the second property, we design the hash functions so the outputs only have $O(\log^{0.5} n)$ bits. One issue here is that if in some level j there are $O(k)$ unmatched blocks and mismatched blocks, then in level $j + 1$, after dividing, there may be $O(k \log^{0.4} n)$ unmatched blocks and mismatched blocks. Naively using an error correcting code for this number of errors will result in a redundancy of size $\Omega(k \log^{0.4} n \log(n/k))$. However, we observe that these blocks are actually concentrated (i.e., they are smaller blocks in a larger block of the previous level). Thus, we can pack all the hash values of the $\log^{0.4} n$ smaller blocks together into a package, and the total size of the hash values is $\log^{0.4} n \cdot O(\log^{0.5} n) = O(\log n)$. We then show that again there are at most $O(k)$ different packages between Alice's version and Bob's version, thus it is enough to send $O(k \log n)$ bits of redundancy. See Figure 3 for a pictorial representation.

The third property and the first property are actually related and require new ideas. Specifically, we cannot use the ε-self-matching hash functions as we discussed earlier, since that would require the outputs of the hash functions to have $O(\log(1/\varepsilon)) = O(\log(n/k))$ bits, while we can only afford $O(\log^{0.5} n)$ bits. To solve this problem, we strengthen the notion of ε-self-matching hash functions and introduce *ε-synchronization hash functions*, similar in spirit to the notion of ε-synchronization strings introduced in Reference [17]. Specifically, we have the following definition:

Definition 3. Let $T, n', B, R \in \mathbb{N}$ be such that $T \geq B$, and $0 < \varepsilon < 1$. Let x be a string of length $n = n'T$, and $x_T = (x[1, T], x[T + 1, 2T], \ldots, x[(n' - 1)T + 1, n])$ be a partition of x into blocks of size T. Let $\Phi = (\Phi[1], \Phi[2], \ldots, \Phi[n'])$ be a sequence of functions, where for any $k \in [n']$, $\Phi[k]$ is a function from $\{0, 1\}^B$ to $\{0, 1\}^R$.

For a string y of length m, and some indices $0 \leq l_1 < r_1 \leq n'$, $0 \leq l_2 < r_2 \leq m$, let $\mathsf{MATCH}_\Phi(x_T(l_1, r_1], y(l_2, r_2])$ denote the size of the maximum monotone matching between $x_T(l_1, r_1]$ and $y(l_2, \min(r_2 + T, m + 1))$ under $\Phi(l_1, r_1]$. We say Φ is a sequence of ε-synchronization hash functions with respect to x, if it satisfies the following properties:

- For any three integers i, j, k where $i < Tj$ and $j < k$, denote $l_1 = k - j$ and $l_2 = Tj - i$.

$$\mathsf{MATCH}_\Phi(x_T(j, k], x(i, Tj]) < \varepsilon \left(l_1 + \frac{l_2}{T} \right) \tag{1}$$

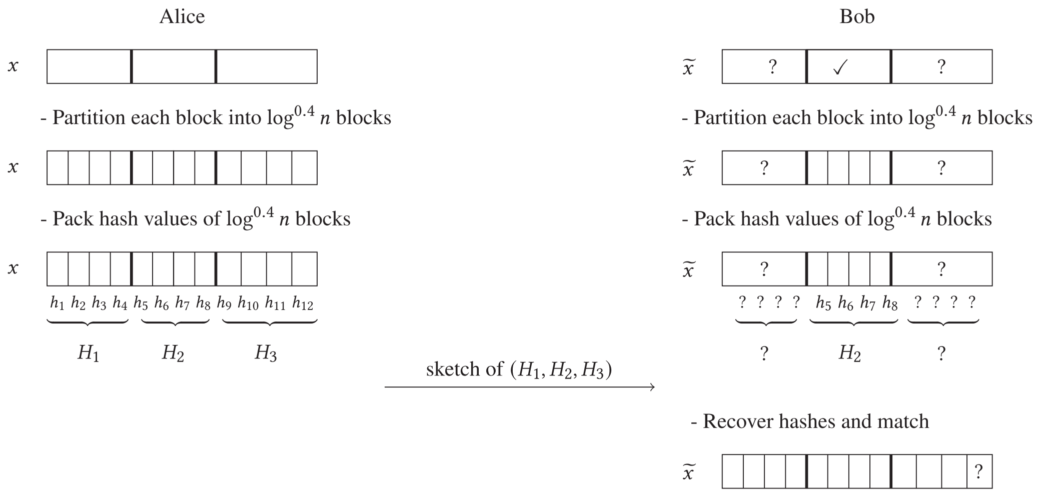
Alice
Bob
x
$\tilde{x}$
?
✓
- Partition each block into $\log^{0.4} n$ blocks
- Pack hash values of $\log^{0.4} n$ blocks
$h_1\ h_2\ h_3\ h_4\ h_5\ h_6\ h_7\ h_8\ h_9\ h_{10}\ h_{11}\ h_{12}$
H_1
H_2
H_3
sketch of (H_1, H_2, H_3)
- Recover hashes and match

Fig. 3. Pictorial representation of one level in Stage 2, where Alice divides each block (divided by thick line) into $\log^{0.4} n$ small blocks, and packs the hash values of every $\log^{0.4} n$ small blocks into one package. For example, in the plot the hash values h_1, h_2, h_3, h_4 are packed into one package H_1. At the end of this level, only the last small block is unmatched.

- For any three integers i, j, k where $i < j$ and $k > T(j-1)+1$, denote $l_1 = j - i$ and $l_2 = k - T(j-1) - 1$.

$$\mathsf{MATCH}_\Phi(x_T(i,j], x(T(j-1)+1, k]) < \varepsilon\left(l_1 + \frac{l_2}{T}\right) \tag{2}$$

It can be seen that ε-synchronization hash functions are indeed stronger than ε-self-matching hash functions, and we show that even a sequence of ε-synchronization hash functions with $\varepsilon = \Omega(1)$ is enough to guarantee that there are at most $O(k)$ unmatched blocks and mismatched blocks (in fact, the number of unmatched blocks and mismatched blocks is bounded by $\frac{1+2\varepsilon}{1-2\varepsilon}k$). A similar property for ε-synchronization strings is also shown in Reference [17].

Our construction of the ε-synchronization hash functions actually consists of two parts. In the first part, Alice generates a sequence of hash functions that ensures the ε-synchronization property for relatively large intervals (i.e., when $l_1 + \frac{l_2}{T}$ is large). We show that this sequence of hash functions can again be generated by using an almost κ-wise independent sample space that uses $O(\log n)$ random bits (the argument is similar to that of ε-self-matching hash functions). Thus, Alice can exhaustively search for a fixed set of hash functions in polynomial time, and send Bob the description using $O(\log n)$ bits. The output of these hash functions has $O(\log^{0.5} n)$ bits. To ensure the ε-synchronization property for small intervals, Alice computes a hash function for each block, such that when the inputs are restricted to length B substrings (note that the inputs to the hash functions we define are always length B strings) in a small interval, this function is *injective*. Since all substrings of length B are distinct in x, this ensures that the outputs of the same hash function within a small interval are also distinct, and thus the ε-synchronization property also holds for small intervals. The output of these hash functions has $O(\log\log n)$ bits. We show that these functions can be constructed deterministically using an almost κ-wise independent sample space that uses $O(\log\log n)$ random bits, and hence also has description size $O(\log\log n)$. Bob will do the same thing based on his "partially corrected version" $\tilde{x}$. We show that if in level j there are $O(k)$ mismatched and unmatched blocks between x and $\tilde{x}$, then in level $j+1$

the set of functions computed by Alice and the set of functions computed by Bob have at most $O(k \log^{0.4} n)$ different elements, but the different elements are again concentrated. Now, we use the same trick as before and pack each successive $\log^{0.6} n$ such functions together, which only needs $O(\log^{0.6} n \log\log n) = O(\log n)$ bits to describe. After the packing it is again true that the set of functions computed by Alice and the set of functions computed by Bob have at most $O(k)$ different packages, thus Alice can send a sketch of $O(k \log n)$ bits for Bob to recover the correct set of hash functions. The final sequence of ε-synchronization hash functions is then the combination of these two sets of functions, where the output has $O(\log^{0.5} n) + O(\log\log n) = O(\log^{0.5} n)$ bits, and the description that Alice sends has $O(\log n) + O(k \log n) = O(k \log n)$ bits. This concludes our algorithm to generate a sketch of size $O(k \log n)$ for a uniform random string.

We now turn to remove the assumption of a uniform random string x. Put simply, our idea is that given any arbitrary string x, we first compute the XOR of x and a *pseudorandom* string z (a mask), to turn x into a pseudorandom string as well. We will use $O(\log n)$ random bits to generate z and show that with high probability over the random bits used, the XOR of x and z satisfies the B-distinct property, as well as property 1 and property 2. For this purpose, we construct three **pseudorandom generators (PRGs)**, one for each property. The B-distinct property can be ensured by again using an almost κ-wise independent sample space that uses $O(\log n)$ random bits. For property 1, we divide the string of length n into blocks of length $T = O(s2^s \log n)$ and generate the same mask for every block. Within each block, we can view it as a sequence of $t = O(\log n)$ sub-blocks each of length $s2^s$. For each sub-block, the test of whether it contains the pattern p can be implemented as a DNF with size poly $\log n$, so we can use a PRG for DNF to fool this test with constant error, and this uses poly $\log\log(n)$ random bits. We then use a random walk on a constant-degree expander graph of length t to generate the full mask, which guarantees that the pattern occurs with probability $1 - 1/\mathrm{poly}(n)$. A union bound now shows that property 1 holds with high probability, and the PRG has seed length $O(\log n)$. For property 2, we can essentially do the same thing, i.e., divide the string of length n into blocks of length $T = O(2^s \log n)$ and generate the same mask for every block. For each block, again we use a PRG for DNF together with a random walk on a constant-degree expander graph to get a PRG with seed length $O(\log n)$. Finally, we take the XOR of the outputs of the three PRGs, and we show that with high probability all three properties hold for $x \oplus z$.

Since the PRG has seed length $O(\log n)$, again we can exhaustively search for a fixed z that works for the string x, and z can be described by $O(\log n)$ bits. The encoding of x is then $x \oplus z$, together with an encoding of the concatenation of the description of z and the sketch of $x \oplus z$. To decode, one can first recover $x \oplus z$ and then recover x by using the description of z and the PRG.

1.3 More on ε-synchronization Hash Functions

Our definition and construction of ε-synchronization hash functions turn out to have several tricky issues. First, there may be other possible definitions of such hash functions, but for our application the current definition is the most suitable one. Second, in our construction of the ε-synchronization hash functions, we crucially use the fact that the string x has the B-distinct property. This is because if x does not have this property, then when we try to find a maximum monotone matching, it may be the case that every pair in the matching is illy matched (e.g., the strings corresponding to the pairs are in different positions but the strings themselves are the same) and this will cause a problem in our analysis. This problem may be solved by modifying the definition of ε-synchronization hash functions, but it will potentially result in a larger output size of the hash functions (e.g., $\Omega(\log n)$ bits), which we cannot afford.

Finally, our construction of the ε-synchronization hash functions consists of two separate sets of hash functions, one for large intervals and one for small intervals. In the construction of hash functions for small intervals, we again crucially use the fact that the string x has the B-distinct property, so for each block both parties can compute a hash function and Alice can just send a short sketch for Bob to recover all the hash fnctions. We note that here one cannot simply apply the (deterministic) Lovász Local Lemma as in References [10, 17, 18], since this will result in a very long description of the hash functions, and Alice cannot just send it to Bob.

Organization of this article. In Section 2, we introduce some notation and basic techniques used in this article. In Section 3, we give the deterministic protocol for document exchange. In Section 4, we give a protocol for document exchange of a uniform random string. In Section 5, we construct error correcting codes for edit errors using results from previous sections. Finally, we conclude with some open problems in Section 6. Appendix A provides details on the asymptotic behavior of the optimal redundancy of document exchange and insdel codes.

2 PRELIMINARIES

2.1 Notation

Let Σ be an alphabet (which can also be a set of strings). For a string $x \in \Sigma^*$,

(1) $|x|$ denotes the length of the string.
(2) $x[i,j]$ denotes the substring of x from position i to position j (both ends included).
(3) $x[i]$ denotes the ith symbol of x.
(4) $x \circ x'$ denotes the concatenation of x and some other string $x' \in \Sigma^*$.
(5) B-prefix denotes the first B symbols of x.
(6) x^N denotes the concatenation of N copies of string x.

We use U_n to denote the uniform distribution on $\{0,1\}^n$.

2.2 Edit Distance and Longest Common Subsequence

Definition 4 (Edit Distance). For any two strings $x \in \Sigma^n, x' \in \Sigma^{n'}$, the edit distance $\mathsf{ED}(x,x')$ is the minimum number of edit operations (insertions and deletions) required to transform x into x'.

Definition 5 (Longest Common Subsequence). For any strings x, x' over Σ, the longest common subsequence of x and x' is the longest pair of subsequences of x and x' that are equal as strings. $\mathsf{LCS}(x,x')$ denotes the length of the longest common subsequence between x and x'.

Note that $\mathsf{ED}(x,x') = |x| + |x'| - 2\mathsf{LCS}(x,x')$.

2.3 Pseudorandom Generator

Definition 6 (Pseudorandom Generator). A function $g : \{0,1\}^r \to \{0,1\}^n$ is a **pseudorandom generator (PRG)** against a function $f : \{0,1\}^n \to \{0,1\}$ with error ε if

$$|\Pr[f(U_n) = 1] - \Pr[f(g(U_r)) = 1]| \le \varepsilon,$$

where r is called the seed length of g.

We also say g ε-fools function f. Similarly, g ε-fools a class of function $\mathcal{F}$ if g fools all functions in $\mathcal{F}$.

Definition 7. A CNF formula ϕ is of the form $\phi = \bigwedge_{i=1}^m C_i$, where each term C_i is an OR of literals (variables or negations).

A DNF formula ϕ is of the form $\phi = \bigvee_{i=1}^m C_i$, where each term C_i is an AND of literals.

A formula ϕ is said to be of width w if every term C_i involves at most w distinct variables.

Theorem 5 (PRG for CNF/DNFs [12]). *There exists an explicit PRG $g : \{0,1\}^r \to \{0,1\}^n$ s.t. for every $n, m \in \mathbb{N}, \varepsilon > 0$, every CNF/DNF f with n variables, m terms,*

$$|\Pr[f(U_n) = 1] - \Pr[f(g(U_r)) = 1]| \le \varepsilon,$$

where $|g(n, m, \varepsilon, U_r)| = n$, $r = O(\log n + \log^2(m/\varepsilon) \cdot \log\log(m/\varepsilon))$.

2.4 Almost K-wise Independence

Definition 8 (ε-almost κ-wise Independence in The Max Norm [2]). Random variables $X_1, X_2, \dots, X_n$ over $\{0,1\}$ are ε-almost κ-wise independent in the max norm if for all distinct $i_1, i_2, \dots, i_\kappa \in [n]$, $\forall x \in \{0,1\}^\kappa$, $|\Pr[X_{i_1} \circ X_{i_2} \circ \cdots \circ X_{i_\kappa} = x] - 2^{-\kappa}| \le \varepsilon$.

A function $g : \{0,1\}^d \to \{0,1\}^n$ is an ε-almost κ-wise independence generator in max norm if $g(U) = X = X_1 \circ \cdots X_n$ are ε-almost κ-wise independent in max norm, where U is the uniform distribution over $\{0,1\}^d$.

In the following, unless otherwise specified, when we say ε-almost κ-wise independence, we mean that it is in the max norm.

Theorem 6 (ε-almost κ-wise Independence Generator [2]). *There exists an explicit construction s.t. for every $n, \kappa \in \mathbb{N}$, $\varepsilon > 0$, it computes an ε-almost κ-wise independence generator $g : \{0,1\}^d \to \{0,1\}^n$, where $d = O(\log \frac{\kappa \log n}{\varepsilon})$.*

The construction is highly explicit in the sense that, $\forall i \in [n]$, the ith output bit can be computed in time $\mathrm{poly}(\kappa, \log n, \frac{1}{\varepsilon})$ given the seed and i.

2.5 Random Walk on Expander Graphs

Definition 9 ((n, d, λ)-expander Graphs). Let G be a d-regular graph on n vertices, and let $\lambda(G)$ be the second largest eigenvalue of the normalized adjacency matrix A_G of G. If G is an n-vertex d-regular graph and $\lambda(G) \le \lambda$ for some number $\lambda < 1$, then we say that G is an (n, d, λ)-expander graph.

A family of graphs $\{G_n\}_{n\in\mathbb{N}}$ is a (d, λ)-expander graph family if there are some constants $d \in \mathbb{N}$ and $\lambda < 1$ such that for every n, G_n is an (n, d, λ)-expander graph.

A random walk on a graph G proceeds by first randomly choosing a vertex of G and then, for each step, randomly selects a neighbor of the current vertex and moves to that neighbor.

Theorem 7 (Random Walk on Expander Graphs, [1, 20] Theorem 3.11). *Let $A_0, \dots, A_t$ be sets of vertices with densities $\alpha_0, \dots, \alpha_t$, respectively, in an (n, d, λ)-expander graph G. Let $X_0, \dots, X_t$ be a random walk on G. Then*

$$\Pr[\forall i, X_i \in A_i] \le \prod_{i=0}^{t-1} (\sqrt{\alpha_i \alpha_{i+1}} + \lambda).$$

It is well known that some expander families have strongly explicit constructions.

Theorem 8 (Explicit Expander Family [3]). *For every constant $\lambda \in (0,1)$ and some constant $d \in \mathbb{N}$ depending on λ, there exists a strongly explicit (d, λ)-expander family.*

Strongly explicit means that finding the neighbor of a given vertex can be done in time polylogarithmic of the number of vertices.

2.6 Error Correcting Codes (ECC) and Document Exchange Protocols

An (n, m, d)-code C is an ECC (for Hamming errors) with codeword length n, message length m. The Hamming distance between every pair of codewords in C is at least d.

Next, we recall the definition of ECC for edit errors.

Definition 10. An ECC $C \subseteq \{0,1\}^n$ for edit errors with message length m and codeword length n consists of an encoding mapping $Enc : \{0,1\}^m \to \{0,1\}^n$ and a decoding mapping $Dec : \{0,1\}^* \to \{0,1\}^m \cup \{Fail\}$. The code can correct k edit errors if for every y, s. t. $\mathsf{ED}(y, Enc(x)) \le k$, we have $Dec(y) = x$. The rate of the code is defined as $\frac{m}{n}$.

A code is systematic if for every codeword, the corresponding message is a prefix of the codeword.

An ECC family is explicit (or has an explicit construction) if both encoding and decoding can be done in polynomial time.

We will utilize linear algebraic geometry codes to compute the redundancy of the Hamming error case.

Theorem 9 ([13, 19, 27] Systematic Algebraic Geometry Code). *There exists an explicit construction of algebraic geometry linear $(n,m,d)_q$-code with $d + m \ge n - \frac{n}{\sqrt{q}-1.1}$, $q = \lceil \frac{n}{d} \rceil^2$ a prime power, polynomial-time decoding when the number of errors is less than half of the distance. Here, n, q should be at least some fixed constants.*

Moreover for every message $x \in \mathbb{F}_q^m$, the codeword is $x \circ z$ for some redundancy $z \in \mathbb{F}_q^{n-m}$. In other words, the code is systematic.

To construct ECC from document exchange protocol, we need to use a previous result about asymptotically good binary ECC for edit errors given by Schulman and Zuckerman [26].

Theorem 10 ([26]). *There exists an explicit binary ECC family in which a code with codeword length n, message length $m = \Omega(n)$, can correct up to $k = \Omega(n)$ edit errors.*

Definition 11. A document exchange protocol against k edit errors is a two-party (Alice and Bob) communication protocol defined as follows:

Alice has a string $x \in \Sigma^n$ and Bob has a string $y \in \Sigma^{n'}$, where $\mathsf{ED}(x,y) \le k$. The algorithm of Alice and the algorithm of Bob conduct an interactive computation and Bob finally outputs x.

The protocol is one-way if the communication is one-way from Alice to Bob. If the protocol is one-way, then the communication transcript is also called the sketch of x. The time complexity for the protocol is the time complexity of algorithms of both Alice and Bob.

3 DETERMINISTIC DOCUMENT EXCHANGE PROTOCOL

We derandomize the IMS protocol given by Irmak et al. [21] by first constructing ε-self-matching hash functions and then use them to give a deterministic protocol.

3.1 Monotone Matching under Hash Functions

Consider the following matching property between strings under some given hash functions:

Let x, y be strings of length n. Cut x into $\bar{n}$ blocks of length $p \le n$. Without loss of generality, assume $n = p\bar{n}$. Let $H(p, q, \bar{n})$ be a sequence of hash functions $h_1, \dots, h_{\bar{n}}$ s.t. $\forall i \in [\bar{n}], h_i : \{0,1\}^p \to \{0,1\}^q$. When $p, q, \bar{n}$ is clear from contexts, we simply use H.

Definition 12. A (monotone) matching of size t between x, y under hash functions $h_1, \dots, h_{\bar{n}}$ is a sequence w of pairs of indices, i.e., $w = ((i_1, j_1), (i_2, j_2), \dots, (i_t, j_t)) \in ([n] \times [n])^t$ s.t.

- $1 \le i_1 < i_2 < \cdots < i_t \le n, \forall l \in [t], i_l \pmod{p} = 1$;
- $1 \le j_1 < j_2 < \cdots < j_t \le n, \forall l \in [t], j_l + p \le j_{l+1}$;
- $\forall l \in [t], h\left(x\left[i_l, i_l + p\right)\right) = h\left(y\left[j_l, j_l + p\right)\right), h = h_{\lceil i_l/p \rceil}$.

For $l \in [t]$, if $x[i_l, i_l + p) = y[j_l, j_l + p)$, we say (i_l, j_l) is a good match, otherwise, we say it is a bad pair. We say w is a correct matching if all pairs in w are good.

If x and y are the same string, then w is called a self-matching of x.

For the matching w between x, y under hash functions $h_1, \dots, h_{\bar{n}}$, we simply say it as a matching w, if x, y and $h_1, \dots, h_{\bar{n}}$ are clear from the context.

The next lemma shows that the maximum matching between two strings under some hash functions can be computed in polynomial time, using dynamic programming.

Lemma 1. *There is an algorithm s.t. for every $H(p, q, \bar{n})$, every $x, y \in \{0,1\}^{n=p\bar{n}}$, given hash values $h_1(x[1, 1+p)), \dots, h_{\bar{n}}(x(n-p, n])$ and y, it can compute the maximum matching between x, y under $H(p, q, \bar{n})$ in time $O(n^2(t_H + \log n))$, where t_H is the time complexity of computing an arbitrary function in H.*

(Note that x is not necessary to be part of the input).

Proof. We present a dynamic programming to compute the maximum matching.

For every $j' \in [\bar{n}], j \in [n]$, let $f(j', j)$ be the size of the maximum matching between $x[1, j'p]$ and $y[1, j]$ under H. We compute f as follows. The base case is

$$f[j', j] = 0, j' = 0, j \in [n]; \qquad f[j', j] = 0, j' \in [\bar{n}], j \in [p].$$

The transition function is

$$f(j', j) = \begin{cases} \max\left(f(j'-1, j-p) + 1, f(j'-1, j), f(j', j-1)\right), & \text{if } h_{j'}\left(x(j'p-p, j'p]\right) = h_{j'}\left(y(j-p, j]\right); \\ \max\left(f(j'-1, j), f(j', j-1)\right), & \text{if } h_{j'}\left(x(j'p-p, j'p]\right) \neq h_{j'}\left(y(j-p, j]\right). \end{cases}$$

Although f only computes the size of the maximum matching, we can record the corresponding matching, i.e., the sequence of pairs of indices, every time when we compute $f(j', j), j' \in [\bar{n}], j \in [n]$. So, finally, we can get the maximum matching after computing $f(\bar{n}, n)$ correctly.

Now, we show correctness. For j', j, by definition of f and the position of the last match in the maximum matching, there are the following three situations: First, the last match is matching the last block of $x[1, j'p]$ to the suffix $y(j-p, j]$ if they can be matched. For this case $f(j', j) = f(j'-1, j-p) + 1$. Second, the last match is in between the first $j'-1$ blocks of x and $y[1, j]$. Then $f(j', j) = f(j', j-1)$. Third, the last match is in between the first j' blocks of x and $y[1, j-p]$. Then $f(j', j) = f(j'-1, j)$. Also for basic cases, when $j' = 0$ or $j \le p$, f should be 0. As a result, the algorithm is correct.

Next, we consider time complexity. We need to compute $f(j', j), j' \in [\bar{n}], j \in [n]$ one-by-one and $n\bar{n} = O(n^2)$. Every time we compute an $f(j', j)$, we need to compute a constant number of evaluations of the hash functions and append a pair of indices to some previous records to create the current record of the maximum matching. That takes time $O(\log n + t_H)$. So, the overall time complexity is $O(n^2(\log n + t_H))$. □

Next, we show that a matching between two strings with edit distance k induces a self-matching in one of the strings, where the number of bad pairs decreases by at most k.

Lemma 2. *Given two sequences $x \in \{0,1\}^{n=p\bar{n}}, y \in \{0,1\}^{O(n)}$ s.t. $\mathsf{ED}(x, y) \le k$, if there exists a matching w between x and y under $H(p, q, \bar{n})$ having at least m bad pairs, then there is a self-matching w' of x with size $|w'| \ge |w| - k$, having at least $m - k$ bad pairs.*

Proof. Assume without loss of generality the m bad pairs in the matching are $(j'_1, j_1), \dots, (j'_m, j_m)$.

As the number of edit errors is upper bounded by k, to edit y back to x, we only need to do insertions and deletions in at most k of the substrings $y[j_1, j_1 + p), y[j_2, j_2 + p), \dots, y[j_m, j_m + p)$. Other substrings of y that are in the matching, are unmodified. They induce a self-matching w' of

x, which is a subsequence of w, matching unmodified blocks to themselves. Note that only at most k entries of w are excluded. So, the size of w' is at least $|w| - k$. Also, the number of bad pairs in the matching is at least $m - k$. □

The following property shows that a matching between two long intervals induces a matching between two shorter intervals s.t. the ratio between the matching size and the total interval length is barely changed:

Lemma 3. *Given two sequences $x \in \{0,1\}^{n=p\bar{n}}, y \in \{0,1\}^{n'=O(n)}$, if there is a matching w between x and y under $H(p,q,\bar{n})$ having size t, then for every $t^* \leq t$, there is a matching w^*, subsequence of w, between x_0 interval of x and y_0 interval of y, s.t. $|w^*| = t^*$ and $|x_0| + |y_0| \leq \frac{2t^*}{t}(n+n')$.*

Proof. We consider the following recursive procedure.

At the beginning (the first round), let $w_1 = w$. We know that $n + n' \leq \frac{t}{t}(n+n')$.

For the ith round, assume we have a matching w_i between $x[j_1, j_2]$, $y[j_1', j_2']$ with $|w_i| = t_i$ and $|x[j_1,j_2] + y[j_1',j_2']| \leq \frac{t_i}{t}(n+n')$. Let $w_i = ((\rho_1,\rho_1'),\ldots,(\rho_{t_i},\rho_{t_i}'))$.

We pick $l = \lfloor t_i/2 \rfloor$. Cut w_i into two parts, the first l pairs and the remaining ones. The first part is a matching between $x_1 = x[j_1, \rho_l + p)$ and $y_1 = y[j_1, \rho_l' + p)$. The second part is a matching between $x_2 = x[\rho_l + p, j_2]$ and $y_2 = y[\rho_l' + p, j_2]$.

Note that either $|x_1| + |y_1| \leq \frac{l}{t}(n+n')$ or $|x_2| + |y_2| \leq \frac{t_i - l}{t}(n+n')$. Because if not, then

$$|x[j_1,j_2]| + |y[j_1',j_2']| = |x_1| + |x_2| + |y_1| + |y_2| > \frac{t_i}{t}(n+n'),$$

which contradicts the assumption that $|x[j_1,j_2] + y[j_1',j_2']| \leq \frac{t_i}{t}(n+n')$. As a result, we can pick one of the two sequences as w_{i+1}.

We can go on doing this until the i^*th round in which there is a matching w_{i^*} of size $t_{i^*} \in [t^*, 2t^*)$, whose corresponding pair of substrings are x_0 and y_0. We know that $|x_0| + |y_0| \leq \frac{t_{i^*}}{t}(n+n')$. We pick t^* pairs in w_{i^*} and this gives a matching between x_0 and y_0 s.t. $|x_0| + |y_0| \leq \frac{2t^*}{t}(n+n')$. □

3.2 ε-self-matching Hash Functions

We now describe an explicit construction for a family of sequences of ε-self-matching hash functions. Recall the definition (ε-self-matching hash function.

Definition 13 (ε-self-matching Hash Function). Let $p, q, n, n' \in \mathbb{N}$ be such that $n = n'p$. For any $0 < \varepsilon < 1$ and $x \in (\{0,1\}^p)^{n'}$, we say that a sequence of hash functions $h_1, h_2, \ldots, h_{n'}$ where $\forall i \in [n'], h_i : \{0,1\}^p \to \{0,1\}^q$ is a sequence of ε-self-matching hash functions with respect to x, if any matching between x and $y \in \{0,1\}^n$ under $h_1, h_2, \ldots, h_n$, where y is the binary expression of x, has at most εn bad matches.

Theorem 11. *There exists an algorithm that, for every $\bar{n}, p \in \mathbb{N}$, $\varepsilon \in (0,1)$, on input $x \in \{0,1\}^{n=\bar{n}p}$, outputs a description of ε-self-matching functions $H(\bar{n}, p, q = \Theta(\log \frac{1}{\varepsilon}))$, in time* poly$(n)$*, where H can be described using $O(\log n)$ bits.*

Furthermore, each hash function can be computed in time poly(n) *given the description.*

To prove this theorem, we consider the following construction:

3.2.1 Explicit Construction. Let c_1, c_2, c_3, c_4, c_5 be large enough constants. Let $n, p, q \in \mathbb{N}$, $\varepsilon \in (0,1)$, $n \geq p \geq q$, $q = c_1 \log \frac{1}{\varepsilon}$.

On input $x \in \{0,1\}^{n=\bar{n}p}$:

(1) Let $g : \{0,1\}^d \to \{0,1\}^{q\bar{n}2^p}$ be the ε_g-almost κ-wise independence generator from Theorem 6, where $\kappa = c_2 \log n$, $\varepsilon_g = 1/n^{c_3}$, $d = c_4 \log n$;

(2) View the output of g as in a two dimension array $(\{0,1\}^q)^{[\bar{n}]\times\{0,1\}^p}$;
(3) Let $m = c_5\frac{\log n}{q}$ and then exhaustively search $u \in \{0,1\}^d$ s.t.
($\star$) For every consecutive $t_1 \le \frac{2m}{p\varepsilon}$-blocks $x_0 \in \{0,1\}^{pt_1}$ of x and every length $t_2 \le \frac{2m}{\varepsilon}$ substring x_0' of x, x_0 and x_0' have no matching of m bad matches under functions

$$h_i(\cdot) = g(u)[i][\cdot], i \in [\bar{n}],$$

where $g(u)[i][\cdot]$ is the corresponding entry in the two dimension array $g(u)$. (The evaluation of a hash function is s.t. given u, $i \in [\bar{n}]$, input $v \in \{0,1\}^p$, then $h_i(v) = g(u)[i][v]$.)
(4) Return u, which is the description of the ε-self-matching hash functions.

3.2.2 *Analysis.*

Lemma 4. *There exists an u such that the assertion ($\star$) holds.*

Proof. We consider a uniform random string u and show that with positive probability u satisfies assertion ($\star$).

Fix a pair of substrings $x[j_1, j_2]$, $x[j_1', j_2']$, and a sequence of pairs $w^* = ((\rho_1, \rho_1'), \ldots, (\rho_m, \rho_m'))$ between them of size $m = c_5\frac{\log n}{q}$ s.t. $\forall l, x[\rho_l, \rho_l + p) \ne x[\rho_l', \rho_l' + p)$. The probability

$$\begin{aligned}
&\Pr_u[w^* \text{ is a matching.}]\\
&= \Pr_u\left[\forall l \in [m], h_{i_l}(x[\rho_l, \rho_l + p)) = h_{i_l}\left(x[\rho_l', \rho_l' + p)\right)\right]\\
&= \sum_{a \in (\{0,1\}^q)^m} \Pr_u\left[\forall l \in [m], h_{i_l}(x[\rho_l + p)) = h_{i_l}\left(x[\rho_l', \rho_l' + p)\right) = a_l\right]\\
&\le 2^{qm}\left(\left(\frac{1}{2^q}\right)^{2m} + \varepsilon_g\right)\\
&\le \frac{1}{2^{qm}} + 2^{qm}\varepsilon_g\\
&= \frac{1}{n^{c_5}} + \frac{1}{n^{c_3-c_5}}.
\end{aligned}$$

Here, the first inequality is by the definition of matching. The second equality is dividing the events into non-intersecting events according to hash values. The first inequality is due to that we can take $\kappa \ge 2qm$. The second inequality is due to that we take c_3, c_5 to be large enough constants. The total number of pairs $x[j_1, j_2]$, $x[j_1', j_2']$ is at most n^4. For each fixed pair there are at most

$$\binom{t_1}{m}\binom{t_2}{m} \le \left(\frac{2e}{p\varepsilon}\right)^{c_5\left(\frac{\log n}{q}\right)}\left(\frac{2e}{\varepsilon}\right)^{c_5\left(\frac{\log n}{q}\right)} = \left(\frac{4e^2}{p\varepsilon^2}\right)^{c_5\left(\frac{\log n}{\log q}\right)}$$

number of different matchings of size m. If c_1 is large enough, then $q = c_1 \log\frac{1}{\varepsilon}$ is large enough. Thus, by the union bound, the probability that the assertion ($\star$) holds is at least $1 - (\frac{1}{n^{c_5}} + \frac{1}{n^{c_3-c_5}})(\frac{4e^2}{p\varepsilon^2})^{c_5\left(\frac{\log n}{\log q}\right)} > 0$. □

Lemma 5. *Construction 3.2.1 gives a sequence of ε-self-matching hash functions.*

Proof. For sake of contradiction, let w be a self-matching of x, under the hash functions given by u, having at least $\varepsilon n+1$ bad pairs. We pick all the bad pairs in this matching to get a self-matching $\tilde{w}$.

By Lemma 3, with $t = \varepsilon n + 1, t^* = m, n = n' = |x|$, there are two substrings, $x[j_1, j_2]$ and $x[j_1', j_2']$, $1 \le j_1 \le j_2 \le [n], j_1 \pmod p = 1, 1 \le j_1' \le j_2' \le [n], |x[j_1, j_2]| + \left|x[j_1', j_2']\right| = \frac{t^*}{t} \cdot 2n \le \frac{2m}{\varepsilon}$, having a matching of bad pairs of size at least m, which is a subsequence of $\tilde{w}$. Note that $|x[j_1, j_2]|, |x[j_1, j_2]| \leq \frac{2m}{\varepsilon}$. This contradicts $(\star)$. □

Lemma 6. *The computation of any hash function in the family takes polynomial time.*

Proof. For evaluation, by Theorem 6, given a seed and a position index, the corresponding bit in g's output can be computed in time $\mathrm{poly}(\kappa, \log(q\bar{n}2^p), \frac{1}{\varepsilon_g}) = \mathrm{poly}(n)$. The output of a function has q bits. One can compute the q bits one-by-one and the total running time is still $\mathrm{poly}(n)$. □

Lemma 7. *The construction is in polynomial time.*

Proof. Checking the assertion $(\star)$ takes time $\mathrm{poly}(n)$. To see this, first note that there are $\mathrm{poly}(n)$ pairs of x_0' and x_0. Also for each pair, the total number of matchings of size $m = \Theta(\log n/q)$ is at most

$$\binom{t_1}{m}\binom{t_2}{m} \leq \binom{\frac{4m}{p\varepsilon}}{m}\binom{\frac{4m}{\varepsilon}}{m} \leq (\frac{4e}{\varepsilon})^{2m} = (\frac{4e}{\varepsilon})^{\Theta(\frac{\log n}{q})} = (\frac{4e}{\varepsilon})^{\Theta(\frac{\log n}{\log \frac{1}{\varepsilon}})} = \mathrm{poly}(n).$$

For a specific matching, one can first compute the hash values in time $\mathrm{poly}(n)$ by Lemma 6. Then check whether it is a matching of bad pairs and compute the size of the matching in time $\mathrm{poly}(n)$.

Note that there are $\mathrm{poly}(n)$ number of different seeds. So, generating the description takes polynomial time. □

Proof of Theorem 11. We use Construction 3.2.1, the theorem directly follows from Lemmas 4, 5, 7. □

3.3 Document Exchange Protocol

Our deterministic protocol for document exchange is as follows:

3.3.1 The Protocol. Let $n \in \mathbb{N}$. Let k be the number of edit errors. We focus on $k = \alpha n$, where $\alpha < 1$ is a small constant, since otherwise, we can simply let Alice send her input string.

Both Alice's and Bob's algorithms have $L = O(\log \frac{n}{k})$ levels.

We assume that n is dividable by $2^i k$ for all $i \in [L]$. We will handle the other case at the end of this section.

Ingredients.

- For every $i \in [L]$, let C_i be a systematic ECC from Theorem 9 with message length l_i, correcting $6k$ errors, over alphabet set $\{0,1\}^{b^* = O(\log \frac{n}{k})}$.
- Let C_{final} be a systematic ECC from Theorem 9 with message length l_L, correcting $3k$ errors, over alphabet set b_L.

Construction. Alice: On input $x \in \{0,1\}^n$;

(1) We set up the following parameters:
  (a) For every $i \in [L]$, in the ith level,
    (i) The block size is $b_i = \frac{n}{2^i k}$. We choose L properly s.t. $b_L = O(\log \frac{n}{k})$.
    (ii) The number of blocks $l_i = n/b_i$;
(2) For the ith level:
(2.1) Construct a sequence H_i of $\varepsilon = \frac{k}{n}$-self-matching hash functions $H_i = (h_1, \ldots, h_{l_i})$, for x by Theorem 11, with $b^* = O(\log \frac{n}{k})$, where each $h_j : \{0,1\}^{b_i} \to \{0,1\}^{b^*}, j \in [l_i]$. Let the description of the hash functions be $u[i] \in \{0,1\}^{O(\log n)}$ by Theorem 11;
(2.2) Compute

$$v[i] = (h_1(x[1, b_i]), h_2(x[b_i + 1, 2b_i]), \ldots, h_{l_i}(x(l_i b_i - b_i, l_i b_i]));$$

(2.3) Compute the code redundancy part $z[i]$ of $C_i\ (v[i])$;
(3) Compute the redundancy $z_{\text{final}} \in (\{0,1\}^{b_L})^{\Theta(k)}$ for the C_{final} encoding of the Lth level blocks;
(4) Send $u = (u[1], u[2], \ldots, u[L])$, $z = (z[1], z[2], \ldots, z[L])$, $v[1]$, z_{final}.

Bob: On input $y \in \{0,1\}^{n'=O(n)}$ and received $u, z, v[1], z_{\text{final}}$;

(1) Create $\tilde{x} \in \{0,1,*\}^n$ (i.e., his current version of Alice's x), initiating it to be $*^n$;
(2) For the ith level,
(2.1) Compute $H_i = (h_1, \ldots, h_{l_i})$ using u;
(2.2) Apply the decoding of C_i on

$$h_1\left(\tilde{x}[1,b_i]\right), h_2\left(\tilde{x}[b_i+1, 2b_i]\right), \ldots, h_{l_i}\left(\tilde{x}(l_ib_i - b_i, l_ib_i]\right) \text{ together with redundancy } z_i$$

to get the sequence of correct hash values $v[i]$. Notice that $v[1]$ is received directly, thus Bob does not need to compute it;
(2.3) Compute $w = \left((\rho_1, \rho'_1), \ldots, (\rho_{|w|}, \rho'_{|w|})\right) \in ([l_i] \times [n'])^{|w|}$, which is the maximum matching between x and y under H_i, using $v[i]$, by Lemma 1;
(2.4) Modify $\tilde{x}$ according to the matching, i.e., let $\tilde{x}\left[\rho_j, \rho_j + b_i\right) = y\left[\rho'_j, \rho'_j + b_i\right)$ for every $j \in [|w|]$;
(3) In the Lth level, apply the decoding of C_{final} on the Lth level blocks of $\tilde{x}$ and z_{final} to get x;
(4) Return x.

3.3.2 *Analysis.*

Lemma 8. *Both Alice's and Bob's algorithms are in polynomial time.*

Proof. We first consider Alice's algorithm. For the ith level, $i \in [L]$, dividing x into blocks takes time $O(n)$. Computing the description and doing evaluation of ε-self-matching hash functions takes time poly(n) by Theorem 11. The redundancy $z[i], i \in [L]$ and z_{final} can be computed in polynomial time by Theorem 9. Thus, the time complexity for Alice is poly(n).

Next, we consider Bob's algorithm. At each level i, decoding of $h_1(\tilde{x}[1]) \circ h_2(\tilde{x}[2]) \circ \ldots \circ h_{l_i}(\tilde{x}[l_i]) \circ z[i]$ takes poly(n) time by Theorem 9. By Lemma 1, computing the maximum matching between x and y under H_i takes time $O(n^2(t_H + \log n))$ where t_H is the time complexity of evaluating any one of the hash functions. By Theorem 6, t_H is poly(n), so computing the maximum matching takes poly(n). Decoding on the last level of $\tilde{x}$ blocks and z_{final} also takes polynomial time by Theorem 9. So, the overall running time for Bob is also poly(n). □

Lemma 9. *The communication complexity is* $O(k \log^2 \frac{n}{k})$.

Proof. For every $i \in [L]$, $|u[i]| = O(\log n)$ by Theorem 11. Thus, the total number of bits of u is $L \cdot O(\log n) = O(\log \frac{n}{k} \log n)$.

Also for every $i \in [L]$, by Theorem 9 the number of bits in z_i is $O(k \log \frac{n}{k})$. So, the total number of bits of z is $O(k \log^2 \frac{n}{k})$.

Again by Theorem 9, z_{final} has $O(k \log \frac{n}{k})$ bits.

Note that $v[1]$ has $O(k \log \frac{n}{k})$ bits, since there are $l_1 = O(k)$ blocks in the first level and the length of each hash value is $O(\log \frac{n}{k})$.

Thus, the total communication complexity is $O(\log \frac{n}{k} \log n + k \log^2 \frac{n}{k}) = O(k \log^2 \frac{n}{k})$, since $k \log \frac{n}{k} \geq \log n$ when $k \leq \alpha n$. □

Next, we show the correctness of the construction. We first establish a series of lemmas.

Lemma 10. *For any $i \leq L-1$, if $v[i]$ is correctly recovered, then in the ith level the number of bad pairs in w is at most $2k$.*

Proof. We prove by contradiction.

Suppose there are more than $2k$ bad pairs in w. As $\mathsf{ED}(x,y) \leq k$, by Lemma 2, there is a self-matching having more than k bad pairs. By picking all the wrong pairs in this matching, we get a self-matching $\tilde{w}$ having size more than $k = \varepsilon n$ such that all its matches are wrong. This is a contradiction to the fact that $h_1, \ldots, h_{l_i}$ is a sequence of ε-self-matching hash functions. □

Next, we show that w is large enough so in each level Bob can recover many blocks of x correctly.

Lemma 11. *For any $i \leq L-1$, in the ith level, we have $|w| \geq l_i - k$.*

Proof. The k edits that the adversary deploys on x can change at most k blocks. The remaining unchanged blocks induce a matching between x and y, having size at least $l_i - k$. Since w is the maximum matching between x and y, $|w| \geq l_i - k$. □

Lemma 12. *Bob computes x correctly.*

Proof. We first show that for every level i, Bob can recover $v[i]$ correctly, by induction.

For the first level, this is true, because $v[1]$ is sent directly to him from Alice.

For level $i = 2, \ldots, L-1$, assume Bob gets $v[i-1]$ correctly. By Lemma 10, the matching w has at most $2k$ bad pairs. By Lemma 11, $|w| \geq l_i - k$. Thus, $|w|$ gives at least $l_i - k - 2k = l_i - 3k$ good matches. So, according to w, Bob can recover at least $l_i - 3k$ blocks of x correctly. Thus, in the ith level, there are at most $3k \times 2 = 6k$ bad blocks in $\tilde{x}$. So, $h_1(\tilde{x}_1) \circ \ldots \circ h_{l_i}(\tilde{x}_{l_i}) \circ z[i]$ can be viewed as a corrupted codeword with at most $6k$ errors. As C_i can correct $6k$ errors, by Theorem 9 Bob can compute $v[i]$ correctly.

Note that for the last level, there are at least $n - 3k$ correctly matched pairs of blocks. So, there are at most $3k$ bad blocks in $\tilde{x}$. As C_{final} can correct $3k$ errors, by Theorem 9, Bob can use z_{final} and $\tilde{x}$ to compute the message x correctly. □

Theorem 12. *There exists a deterministic protocol for document exchange, having communication complexity (redundancy) $O(k \log^2 \frac{n}{k})$, time complexity $\mathsf{poly}(n)$, where n is the input size and k is the edit distance upper bound.*

Proof. Construction 3.3.1 gives the deterministic protocol for document exchange. The communication complexity is $O(k \log^2 \frac{n}{k})$ by Lemma 9. The time complexity is $\mathsf{poly}(n)$ by Lemma 8. The correctness is proved by Lemma 12. □

Remark 3. Recall that in Construction 3.3.1, we assume n is dividable by $2^i k$ for all $i \in [L]$. If this is not the case, then we can just pad 0s to the end of x to meet this property. Assume that the length of the string after padding 0s is n''. Since n'' is at most twice of the original length n, the communication complexity is $O(k \log^2 \frac{n''}{k}) = O(k \log^2 \frac{n}{k})$.

4 DOCUMENT EXCHANGE FOR A UNIFORM RANDOM STRING

In this section, we consider document exchange protocols for a random string, i.e., we consider that Alice's message is a uniform random string x, and Bob's string x' is adversarially perturbed and has distance $\leq k$ from x. A construction is given for this case with high success probability, where the probability is over the random choice of x. Formally, we prove the following theorem:

Theorem 13. *There exists a deterministic document exchange protocol for a uniformly random string with success probability $1 - 1/\mathsf{poly}(n)$ and communication complexity (redundancy) $O(k \log n)$.*

Next, we show our protocol and prove the theorem. We will utilize the protocol to obtain an error correcting code in Section 5.

4.1 Properties of Uniform Random Strings

Definition 14 (p-split Point [6]). For a string $p \in \{0,1\}^s$ and $x \in \{0,1\}^n$, a p-split point of x is an index $1 \le i \le n - s + 1$ such that $x[i, i+s) = p$.

Definition 15 (Next p-split Point). Let p and x be two strings and i be a p-split point of x. Define the *next p-split point* of i to be the smallest j such that j is a p-split point and $j > i$. If such j does not exist, then we define the *next p-split point* of i to be $(n+1)$.

We will use the following properties of a uniform random string:

Theorem 14. *For a uniform random string $x \in \{0,1\}^n$, let $p = 1 \circ 0^{s-1}$ be a string of length s. There exist three integers $B_1 = O(s2^s \log n), B_2 = O(2^s \log n)$ and $B = O(\log n)$ such that the following properties hold with probability* $1 - 1/\text{poly}(n)$*:*

Property 1. *Any interval of x with length B_1 contains a p-split point.*
Property 2. *Any interval of x with length B_2 starting at a p-spit point contains a p-split point i such that its next p-split point j satisfies $j - i > 2^s/2$.*
B-**distinct.** *Every two substrings of length B at different positions of x are distinct.*

Proof. For Property 1, let $B_1 = 2s2^s\lceil\log n\rceil$. For any fixed interval I of length B_1, we divide I into small intervals of length s. Then, we have at least $2 \cdot 2^s\lceil\log n\rceil$ small intervals. As the strings in each small interval are independent, the probability that p does not appear in I is at most

$$\left(1 - \frac{1}{2^s}\right)^{2\cdot 2^s\lceil\log n\rceil} < \left(\frac{1}{e}\right)^{2\log n} < \frac{1}{n^2}. \tag{3}$$

Applying the union bound, the probability that such an interval I exists is at most $n/n^2 = 1/n$.

For Property 2, let $B_2 = 2^s\lceil\log n\rceil$. If there is a fixed interval $I = [i, i+B_2)$ such that i is a p-split point, and I violates the second property, i.e., the p-split points $i = i_1 < i_2 < i_3 \ldots$ in I satisfy $i_{t+1} - i_t \le 2^s/2$ for $t = 1, 2, \ldots$. Denote the random variables $x_t = i_{t+1} - i_t, t = 1, 2, \ldots$. Applying the union bound, $\Pr[x_t \le 2^s/2] \le (2^s/2)/2^s = 1/2$. As x_t's are independent, and we have at least $B_2/(2^s/2) = 2\lceil\log n\rceil$ such random variables x_t, we have

$$\Pr[x_t \le 2^s/2, \forall t \le 2\lceil\log n\rceil] \le (\Pr[x_t \le 2^s/2])^{2\lceil\log n\rceil} \le (1/2)^{2\log n} < \frac{1}{n^2}.$$

Applying the union bound, the probability that such an interval I exists is at most $n/n^2 = 1/n$.

For B-distinct, let $B = 3\lceil\log n\rceil$. Consider any two distinct fixed indices $1 \le i \le n - B + 1$ and $1 \le j \le n - B + 1$. Without loss of generality, we can assume $i < j$. Since for any $0 \le k < B$, $x[i+k]$ is independent of $x[j+k]$, we have $\Pr[x[i, i+B) = x[j, j+B)] = (1/2)^B$.

By the union bound, the probability that the B-distinct property does not hold is at most $n^2/2^B = n^2/n^3 = 1/n$.

By the union bound, the probability that all properties are satisfied is at least $1 - 3/n$. □

In the rest of this section, we prove the following theorem:

Theorem 15. *There exists a deterministic document exchange protocol for any string with Property 1, Property 2, and the B-distinct property, where the communication complexity (redundancy) is $O(k \log n)$.*

Note that Theorem 13 is a direct corollary of Theorem 15.

4.2 ε-synchronization Hash Functions

In this subsection, we define and construct ε-synchronization hash functions.

Definition 16. For every constant $0 < \varepsilon < 1$, $c_1, c_2 \in \mathbb{N}$, and every $T, n', B, R \in \mathbb{N}$ with $n = n'T$, let x be a B-distinct string of length n, and $x_T = (x[1,T], x[T+1,2T], \dots, x[(n'-1)T+1,n])$ be the partition of x into blocks of size T. Let $\Phi = (\Phi[1], \Phi[2], \dots, \Phi[n'])$ be a sequence of functions, where $\Phi[t] : \{0,1\}^B \to \{0,1\}^R$ is function that maps B bits to R bits.

If y is a string of length m, $0 \le l_1 < r_1 \le n'$ and $0 \le l_2 < r_2 \le m$ be some indices, let $\mathsf{MATCH}_\Phi(x_T(l_1,r_1], y(l_2,r_2])$ denote the size of the maximum matching between $x_T(l_1,r_1]$ and $y(l_2, \min(r_2+T, m+1))$ under $\Phi(l_1,r_1]$ (see Definition 12).

Φ is a sequence of ε-synchronization hash functions with respect to x if it satisfies the following properties:

- For any three integers i,j,t where $i < Tj$ and $j < t$, denote $l_1 = t - j$ and $l_2 = Tj - i$.
$$\mathsf{MATCH}_\Phi(x_T(j,t], x(i,Tj]) < \varepsilon\left(l_1 + \frac{l_2}{T}\right). \tag{4}$$
- For any three integers i,j,t where $i < j$ and $t > T(j-1)+1$, denote $l_1 = j - i$ and $l_2 = t - T(j-1) - 1$.
$$\mathsf{MATCH}_\Phi(x_T(i,j], x(T(j-1)+1,t]) < \varepsilon\left(l_1 + \frac{l_2}{T}\right). \tag{5}$$

Construction. Our construction of ϵ-synchronization hash functions consists of two parts: a hash function for intervals that are far from each other and a hash function for intervals that are close to each other. Then, we concatenate these two hash functions to obtain the ϵ-synchronization hash function.

4.2.1 Hash Functions for Far Intervals.

Lemma 13. *For every constants $0 < \varepsilon < 1$, $c_1, c_2 \in \mathbb{N}$ with $c_1 < c_2$, and every $\log^{c_1} n \le B \le T \le \log^{c_2} n$ where T is the block size as in Definition 16, there exists a sequence of functions $\Phi = (\phi[1], \dots, \phi[m])$ where $m = n/T$, and a large enough constant c_3, satisfying the following properties:*

- *For every $t \in [m]$, $\phi[t]$ is a function from $\{0,1\}^B$ to $\{0,1\}^R$, where $R = c_3 \log^{0.5} n + \log T$;*
- *For any $0 \le i_1 < i_2 \le m$, $0 \le j_1 < j_2 \le n$, $l_1 = i_2 - i_1$, $l_2 = j_2 - j_1$, $l_1 + l_2/T \ge \log^{0.6} n$, when the overlap of $x(i_1T, i_2T]$ and $x(j_1, j_2+T)$ has less than T bits, we have*
$$\mathsf{MATCH}_\Phi\left(x_T(i_1,i_2], x(j_1,j_2]\right) < \varepsilon\left(l_1 + \frac{l_2}{T}\right).$$

There is an algorithm such that given $n \in \mathbb{N}$ and x, it can compute a description of Φ in polynomial time, where the length of the description is $O(\log n)$. Furthermore, given the description of ϕ_i, ϕ_i can be computed in polynomial time.

Proof. Let $g : \{0,1\}^d \to \{0,1\}^{Rm2^B}$ be an η-almost κ-wise independence generator from Theorem 6, where $\eta = 1/n^{c_4}$, $\kappa = c_5 \frac{\log n}{\eta} R$, $d = c_6 \log n$. Fix $\ell = c_7 \frac{\log n}{R} + \log^{0.5} n$. Here, c_4, c_5, c_6, c_7 are large enough constants.

View the output of g as in $(\{0,1\}^R)^{[m]\times\{0,1\}^B}$. Then exhaustively search a seed u for g s.t. for any length l_1^* interval $x_T(i_1^*, i_2^*]$ and any length l_2^* interval $x(j_1^*, j_2^*]$ where $l_1^* + l_2^*/T \le \frac{3\ell}{\varepsilon}$ and the overlap between the two intervals has less than T bits, there does not exist a matching of size ℓ between the two intervals under hash functions $\Phi(i_1^*, i_2^*]$. Finally, let $\phi[t](\cdot) = g(u)[t][\cdot]$ for every $t \in [m]$.

Now, we show the correctness of the construction by contradiction. Suppose there are $x_T(i_1, i_2], x(j_1, j_2]$ such that $\mathsf{MATCH}_\Phi(x_T(i_1, i_2], x(j_1, j_2]) \ge \varepsilon(l_1 + \frac{l_2}{T}) \ge \varepsilon \log^{0.6} n \ge \ell$ and the overlap between $x(i_1T, i_2T]$ and $x(j_1, j_2]$ has length less than T. Then the matching is actually between $x_T(i_1, i_2]$ and $x(j_1, \min(j_2+T-1, m)]$. By Lemma 3, there exist intervals $x_T(i_1^*, i_2^*]$ with length l_1^* and $x(j_1^*, j_2^*]$ with length l_2^* s.t. $l_1^* + l_2^*/T \le \frac{2\ell}{\varepsilon(l_1+\frac{l_2}{T})}(l_1 + \frac{l_2+T-1}{T}) \le \frac{3\ell}{\varepsilon}$ and there is a matching of size ℓ between the two intervals. Also note that by Lemma 3, $x_T(i_1^*, i_2^*]$ is a subinterval of $x_T(i_1, i_2]$ and $x(j_1^*, j_2^*]$ is a subinterval of $x(j_1, \min(j_2 + T - 1, m)]$. So, the overlap between $x_T(i_1^*, i_2^*]$ and $x(j_1^*, j_2^*]$ has less than T bits. This is a contradiction to our choice of u.

To finish the proof, we need to show that there exists such a u. We show that a uniform random choice of u satisfies the requirement with high probability. Fix $i_1^*, i_2^*, j_1^*, j_2^*$, where $i_2^* - i_1^* = l_1^*$ and $j_2^* - j_1^* = l_2^*$. Fix a sequence of pairs $w = ((\rho_1, \rho_1'), \dots, (\rho_\ell, \rho_\ell'))$ between $x_T(i_1^*, i_2^*], x(j_1^*, j_2^*]$, the probability that w is a matching under Φ is

$$\begin{aligned} &\Pr\left[\forall t^* \in [\ell], \phi[t^*]\left(x_T[\rho_{t^*}]\right) = \phi[t^*]\left(x[\rho'_{t^*}, \rho'_{t^*} + T)\right)\right] \\ \le &\sum_{a_1, a_2, \dots, a_\ell \in \{0,1\}^R} \Pr\left[\forall t^* \in [\ell], \phi[t^*]\left(x_T[\rho_{t^*}]\right) = \phi[t^*]\left(x[\rho'_{t^*}, \rho'_{t^*} + T)\right) = a_{t^*}\right] \\ \le &\ 2^{R\ell}(2^{-2R\ell} + \eta) \\ = &\ 2^{-R\ell} + \eta 2^{R\ell}. \end{aligned} \tag{6}$$

Note that the overlap between $x_T(i_1^*, i_2^*], x(j_1^*, j_2^*]$ has less than T bits. So, for every $t^* \in [n]$, the B-prefix of $x_T[\rho_{t^*}]$ and $x[\rho'_{t^*}, \rho'_{t^*} + T)$ are not equal.

There are at most $\binom{l_1^*}{\ell}\binom{l_2^*}{\ell} \le (2e/\varepsilon)^\ell (2eT/\varepsilon)^\ell = (4e^2T/\varepsilon^2)^\ell$ different matchings between $x_T(i_1^*, i_2^*]$ and $x(j_1^*, j_2^*]$. As long as c_3, c_4 are large enough, by a union bound, a uniform random choice of u satisfies the requirement with probability at least $1 - 1/\mathsf{poly}(n)$.

The time complexity of this procedure is $\mathsf{poly}(n)$. This is because in the exhaustive search, we try $O(n^2 l_1^* l_2^*) = \mathsf{poly}(n)$ pairs of intervals, and for each pair, we try $\binom{l_1^*}{\ell}\binom{l_2^*}{\ell} \le (2e/\varepsilon)^\ell (2eT/\varepsilon)^\ell = \mathsf{poly}(n)$ different matchings. Moreover, we try at most $2^d = n^{c_6}$ number of different seeds to find u. □

4.2.2 *Hash Functions for Near Intervals.*

Lemma 14. *Let B, c be integers, and S be a subset of $\{0,1\}^B$ of size m. There exists a map $\theta : \{0,1\}^B \to \{0,1\}^{R_\theta}$ such that $R_\theta = c \log m$, θ restricted on S is injective and θ can be represented in $d = c_1(\log B + \log m)$ bits for some constant c_1. In addition, there is an algorithm such that given B, m, S, c, it can find such a θ whose binary representation has the smallest lexicographical order in time* $\mathsf{poly}(B, m)$*; furthermore, given the description of θ, θ can be computed in time* $\mathsf{poly}(B, m)$.

Proof. Let $g : \{0,1\}^d \to \{0,1\}^{R_\theta 2^B}$ be an η-almost $2R_\theta$-wise independence generator from Theorem 6, where $\eta = 1/m^{c_2}$ for a large enough constant c_2. Regard the output of g as a sequence in $(\{0,1\}^{R_\theta})^{\{0,1\}^B}$. We exhaustively find $u \in \{0,1\}^d$ s.t. θ is injective restricting on S, where $\theta(\cdot) = g(u)[\cdot]$. It remains to show that there exists such a u. Assume u is drawn from U_d. For each fixed pair of distinct elements $a, b \in S$,

$$\Pr[\theta(a) = \theta(b)] = \sum_{v \in \{0,1\}^{R_\theta}} \Pr[\theta(a) = \theta(b) = v] \le 2^{R_\theta}(2^{-2R_\theta} + \eta) \le \frac{1}{m^3},$$

as long as c, c_2 are large enough. Note that there are at most m^2 such pairs. By a union bound, with probability $1 - 1/m$, θ is injective restricted on S.

By Theorem 6, the evaluation of θ can be computed in time $\mathsf{poly}(B, m)$.

Note that the time complexity for finding θ is $\mathrm{poly}(B, m)$ because the exhaustive search tries 2^d different seeds. Checking whether θ is injective on S takes time $O(m^2)\mathrm{poly}(B, m) = \mathrm{poly}(B, m)$. □

ϵ-synchronization Hash Functions. We construct the ϵ-synchronization hash functions by combining the hash functions for far intervals and the hash functions for near intervals, as follows:

- For any $0 < \varepsilon < 1$ and $T, B \in \mathbb{N}, T \geq B$, let x be a B-distinct string of length $n = Tn'$, and $x_T = (x[1, T], x[T + 1, 2T], \dots, x[(n' - 1)T + 1, n])$ be the natural partition of x.
- For any $t \in [n']$, let S_t be the set containing all the substrings $x[i, i + B)$ satisfying $|T(t - 1) + 1 - i| < T \cdot \log^{0.6} n$ and $1 \leq i \leq n - B + 1$.
- From Lemma 14, we choose the $\theta[t]$ whose binary representation has the smallest lexicographical order that is injective on S_t.
- Let ϕ be a series of functions generated in Lemma 13. For any $t \in [n']$, let $\Phi[t] = (\phi[t], \theta[t])$, and define $\Phi[t](s) = (\phi[t](s), \theta[t](s))$ for all $s \in \{0, 1\}^B$.

Theorem 16. *The sequence Φ in the above construction is a sequence of ε-synchronization hash functions with respect to x.*

Proof. For three integers i, j, k where $i < Tj$ and $j < k$, denote $l_1 = k - j$ and $l_2 = Tj - i$.

If $l_1 + l_2/T \geq \log^{0.6} n$, then by Lemma 13, Equation (4) holds. As every matching under Φ must be a matching under ϕ, we have

$$\mathsf{MATCH}_\Phi(x_T(j, k], x(i, Tj]) \leq \mathsf{MATCH}_\phi(x_T(j, k], x(i, Tj]) \leq \varepsilon\left(l_1 + \frac{l_2}{T}\right). \tag{7}$$

If $l_1 + l_2/T < \log^{0.6} n$, then the total length of the interval $(i, Tk]$ is $Tk - i = Tl_1 + l_2 < T \log^{0.6} n$. From the construction of $(\theta[t])_{t \in [n']}$, $\theta[t], (j < t \leq k)$ maps the substrings of length B in $(i, Tk]$ into different values. Hence, we have $\mathsf{MATCH}_\Phi(x_T(j, k], x(i, Tj]) = 0$.

Thus, Equation (4) holds for all possible l_1 and l_2. By symmetry, Equation (5) also holds. □

4.2.3 Properties.

Lemma 15. *For any $0 < \varepsilon < 1$ and $T \in \mathbb{N}, T \geq B$, let x be a B-distinct string of length $n = Tn'$, and $x_T = (x[1, T], x[T + 1, 2T], \dots, x[(n' - 1)T + 1, n])$ be a partition of x. Let $\Phi = (\Phi[1], \Phi[2], \dots \Phi[n'])$ be a sequence of ε-synchronization hash functions with respect to x, and Π be a matching between x_T and x under Φ. We say a pair $(i, j) \in \Pi$ is a good pair, if $x_T[i] = x[j, j + T)$. Otherwise, we say it is a bad pair. Let g be the number of good pairs in Π, and b be the number of bad pairs in Π. Then*

$$b < 2\varepsilon(n' - g).$$

Proof. As the string x is B-distinct, a pair (i, j) is good if and only if $j = T(i - 1) + 1$. For any good pair (i, j), we remove the block $x_T[i]$ in x_T and $x[j, j + T)$ in x. Then, we list all bad pairs in Π $(i_1, j_1), (i_2, j_2), \dots, (i_b, j_b)$ such that $i_1 < i_2 < \cdots < i_b$. We use the bad pairs to divide the strings into disjoint intervals such that each bad pair is contained in an interval as in Definition 16, as follows: For each $\tau \in [b]$, there are two cases: $T(i_\tau - 1) + 1 > j_\tau$ and $T(i_\tau - 1) + 1 < j_\tau$.

If $T(i_\tau - 1) + 1 > j_\tau$, then we find the largest j'_τ such that $j'_\tau < T(i_\tau - 1) + 1$ and there exists an i'_τ such that $(i'_\tau, j'_\tau) \in \Pi$. Let b_τ be the number of bad pairs (i, j) in Π satisfying $i_\tau \leq i \leq i'_\tau$, denote $l_1^{(\tau)} = i'_\tau - i_\tau + 1$ and $l_2^{(\tau)} = T(i_\tau - 1) - j_\tau + 1$ (see Figure 4). Applying Equation (4), we obtain

$$b_\tau \leq \mathsf{MATCH}_\Phi(x_T[i_\tau, i'_\tau], x[j_\tau, T(i_\tau - 1)]) < \varepsilon(l_1^{(\tau)} + l_2^{(\tau)}/T).$$

If $T(i_\tau - 1) + 1 < j_\tau$, then we find the largest i'_τ satisfying $T(i'_\tau - 1) + 1 < j_\tau$. Then find the largest i''_τ such that $i''_\tau \leq i'_\tau$ and there exists j'_τ satisfying $(i''_\tau, j'_\tau) \in \Pi$. Recall that b_τ is the number of the

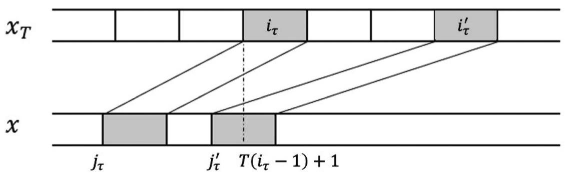

Fig. 4. Pictorial representation for the case $T(i_\tau - 1) + 1 > j_\tau$.

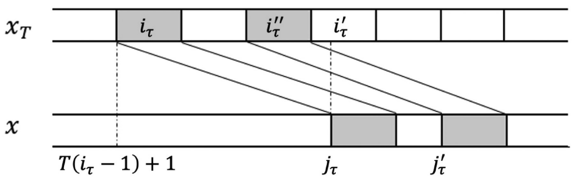

Fig. 5. Pictorial representation for the case $T(i_\tau - 1) + 1 < j_\tau$.

bad pairs (i, j) in Π satisfying $i_\tau \leq i \leq i'_\tau$. Let us denote $l_1^{(\tau)} = i'_\tau - i_\tau + 1$ and $l_2^{(\tau)} = j'_\tau - T(i'_\tau - 1) - 1$ (see Figure 5). Applying Equation (5), we obtain

$$b_\tau \leq \mathsf{MATCH}_\Phi(x_T[i_\tau, i'_\tau], x(T(i'_\tau - 1) + 1, j'_\tau]) < \varepsilon(l_1^{(\tau)} + l_2^{(\tau)}/T).$$

Hence, in both cases, we have $b_\tau < \varepsilon(l_1^{(\tau)} + l_2^{(\tau)}/T)$. Adding them up for all $\tau \in [b]$, we have

$$b = \sum_\tau b_\tau < \varepsilon\left(\sum_\tau l_1^{(\tau)} + \sum_\tau \frac{l_2^{(\tau)}}{T}\right). \tag{8}$$

On the right-hand side, as the intervals corresponding to $l_1^{(\tau)}, l_2^{(\tau)}$ are disjoint and there are g good pairs, we have $\sum_\tau l_1^{(\tau)} \leq n' - g$ and $\sum_\tau l_2^{(\tau)}/T \leq (n - Tg)/T = n' - g$. Hence, we obtain $b < 2\varepsilon(n' - g)$. □

Theorem 17. *For any $0 < \varepsilon < 1$ and $T \in \mathbb{N}, T, B \geq B$, let x be a B-distinct string of length $n = Tn'$, and $x_T = (x[1, T], x[T + 1, 2T], \ldots, x[(n' - 1)T + 1, n])$ be the natural partition of x. Let $\Phi = (\Phi[1], \Phi[2], \ldots, \Phi[n'])$ be a sequence of ε-synchronization hash functions with respect to x, and y is a string satisfying $\mathsf{ED}(x, y) \leq k$. If $\Pi = \mathsf{MATCH}_\Phi(x_T, y)$, and the number of bad pairs in Π is b, then*

$$b < \frac{1 + 2\varepsilon}{1 - 2\varepsilon}k. \tag{9}$$

Proof. As $\mathsf{ED}(x, y) \leq k$, there is a matching of size $n' - k$ between x_T and y under Φ. Hence, $|\Pi| \geq n' - k$. From Theorem 2, there exists a self-matching Π' between x_T and x under Φ. Let b' be the number of bad pairs in Π' and g' be the number of good pairs in Π'. Then the theorem guarantees $|\Pi'| \geq |\Pi| - k$ and $b' \geq b - k$. From Lemma 15, $b' < 2\varepsilon(n' - g')$. However, we have

$$b' + g' = |\Pi'| \geq |\Pi| - k \geq n' - 2k. \tag{10}$$

So, we obtain $b' < 2\varepsilon(n' - g') \leq 2\varepsilon(b' + 2k)$. Manipulating it, we derive an uppers bound for b'.

$$b' < \frac{4\varepsilon}{1 - 2\varepsilon}k. \tag{11}$$

Hence, we have

$$b \leq b' + k < \frac{1 + 2\varepsilon}{1 - 2\varepsilon}k. \tag{12}$$

□

4.3 The Document Exchange Protocol

Our construction proceeds in two stages. In the first stage, the two parties use a fixed small string p and find all p-split points in their strings. As the string x is uniform random, with high probability, the distance between any two adjacent p-split points is $O(s2^s \log n)$. But some p-split points may be too close to each other. So, we only choose the p-split points i such that the next p-split point of i is at least $2^s/2$ away and use these chosen p-split points to partition the string into blocks.

Ingredients. Our construction uses the following ingredients:

- ϵ-synchronization hash functions in Section 4.2.
- Algebraic geometry error correcting codes in Theorem 9.

Construction. Let n, m denote the input length. We set the parameters $s = \lceil \log\log n \rceil + 3, B = 3\lceil \log n \rceil, T_0 = s \cdot 2^s \cdot \lceil \log n \rceil$, and $p = 1 \circ 0^{s-1}$ is a fixed string of length s. For ease of representation, we assume n is a multiple of T_0.[1]

Remark 4. In our construction below, some number may not be exactly divisible by another. For better presentation, we will ignore this issue for now and discuss how to fix it with a simple padding trick at end of this section.

Stage I:

Alice: On input a uniform random string $x \in \{0,1\}^n$.

(1) Choose all p-split points i of x such that its next p-split point j satisfies $j - i > 2^s/2$. Denote the chosen p-split points as $i_1, i_2, \ldots, i_{n'}$. Partition the string x into blocks $x[1, i_2)$, $x[i_2, i_3)$, $x[i_3, i_4) \ldots, x[i_{n'}, n]$, and index these blocks as $1, 2, 3, \ldots, n'$.
(2) Create a set $V = \{(\mathsf{len}_b, \text{B-prefix}_b, \text{B-prefix}_{b+1}) \mid 1 \le b \le n' - 1\}$, where len_b is the length of the bth block, and B-prefix_b and B-prefix_{b+1} are the B-prefix of the bth block and the $(b+1)$-th block, respectively.
(3) Represent the set V as its indicator vector, which has size $\mathsf{poly}(n)$, and send the redundancy z_V being able to correct $4k$ Hamming errors, using Theorem 9 (or simply using a Reed-Solomon code).
(4) Partition the string x evenly into n/T_0 blocks, each of size T_0.

Bob: On input string $y \in \{0,1\}^m$ satisfying $\mathsf{ED}(x, y) \le k$, and the redundancy z_V sent by Alice.

(1) Choose all p-split points i of y such that its next p-split point j satisfies $j - i > 2^s/2$. Denote the chosen p-split points as $i'_1, i'_2, \ldots, i'_{m'}$. Partition the string y into blocks $y[1, i'_2)$, $y[i'_2, i'_3)$, $y[i'_3, i'_4) \ldots, y[i'_{m'}, m]$, and index these blocks as $1, 2, 3, \ldots, m'$.
(2) Create a set $V' = \{(\mathsf{len}_b, \text{B-prefix}_b, \text{B-prefix}_{b+1}) \mid 1 \le b \le m' - 1\}$ using the partition of y.
(3) Use the indicator vector of V' and the redundancy z_V to recover Alice's set V.
(4) Create an empty string $\tilde{x}$ of length n, and fill $\tilde{x}$ according to the set V in the following way: first find the element $(\mathsf{len}^{(1)}, \text{B-prefix}^{(1)}, \text{B-prefix}'^{(1)})$ in V such that for all b, $\text{B-prefix}^{(1)} \neq \text{B-prefix}_{b+1}$. Then partition $\tilde{x}[1, \mathsf{len}^{(1)}]$ as the first block, and fill $\tilde{x}[1, B]$ with $\text{B-prefix}^{(1)}$. Then find the element $(\mathsf{len}^{(2)}, \text{B-prefix}^{(2)}, \text{B-prefix}'^{(2)})$ such that $\text{B-prefix}^{(2)} = \text{B-prefix}'^{(1)}$, and partition $\tilde{x}[\mathsf{len}^{(1)} + 1, \mathsf{len}^{(1)} + \mathsf{len}^{(2)}]$ as the second block, and fill $\tilde{x}[\mathsf{len}^{(1)} + 1, \mathsf{len}^{(1)} + B]$ with $\text{B-prefix}^{(2)}$. Continue doing this until all elements in V are used to recover the partition of x.
(5) For each block b in $\tilde{x}$, if Bob finds a unique block b' in y such that the B-prefix of b' matches the B-prefix of b and the lengths of b and b' are equal, then Bob fills the block b using b'. If

[1]The construction can be easily extended to general case.

such b' does not exist or Bob has multiple choices of b', then Bob just leaves the block b as blank.

(6) Partition the string $\tilde{x}$ evenly into n/T_0 blocks, each of size T_0.

Stage II : Recall that, $T_0 = s2^s\lceil \log n\rceil = \text{poly}(\log n)$. Let $T' = \lceil \log^{0.6} n\rceil, T'' = \lceil \log^{0.4} n\rceil$. The second stage consists of $L = \left\lceil \frac{\log T_0}{\log T''}\right\rceil = \left\lceil \frac{\log(O(s2^s \log n))}{\log(\log^{0.4} n)}\right\rceil = O(1)$ levels. For $1 \le l \le L-1$, we set $T_l = T_{l-1}/T''$. In the last level, we set $T_L = B$, then $T_L \ge T_{L-1}/T''$. For ease of representation, in this stage, we assume n is a multiple of T_l, for all $l \in [L]$.[2]

Alice:

(1) For $l = 1, 2, \ldots L$, in lth level:

(1.1) Partition the string x evenly into $n'_l = n/T_l$ blocks, each of size T_l.

(1.2) Applying the construction in Section 4.2.2 with block size $T = T_l$, Alice gets a sequence of ε-synchronization hash functions $\Phi = (\Phi[1], \Phi[2], \ldots, \Phi[n'_l])$ with respect to x, where each $\Phi[t], t \in [n'_l]$ consists of a pair of functions $(\phi[t], \theta[t])$.

(1.3) Alice sends the description of $(\phi[t])_{t\in[n'_l]}$ to Bob. By Lemma 13, the description uses $O(\log n)$ bits.

(1.4) Alice packs every successive T' elements of $(\theta[t])_{t\in[n'_l]}$ into a vector V_θ, i.e., $V_\theta = (\theta[1, T'], \theta[T'+1, 2T'], \ldots)$. Recall that in the construction of the ε-synchronization hash function Φ in Section 4.2.2, we use S_t to denote the "near intervals" of tth block. Here, the size of S_t is at most $\text{poly}(\log n)$. Hence, by Lemma 14, each $\theta[t]$ has a description of size $O(\log\log n)$. Alice sends the redundancy z_θ being able to correct some $O(k)$ Hamming errors of V_θ, using Theorem 9.

(1.5) For any $t \in [n'_l]$, Alice evaluates $\Phi[t]$ on the tth block of x, and obtains the hash values $I[t] = \Phi[t](x[T_l(t-1)+1, T_l(t-1)+B])$, and stores $(I[t])_{t\in[n'_l]}$ into a vector I. Then she packs every successive T'' elements of I into a vector V_I, i.e., $V_I = (I[1, T''], I[T''+1, 2T''], \ldots)$, and sends the redundancy z_I being able to correct some $O(k)$ Hamming errors of V_I, using Theorem 9.

(2) Alice evenly partitions her string x into n/T_L small blocks, each of size T_L.

(3) Then send a redundancy z_x being able to correct some $O(k)$ wrong blocks or unmatched blocks, using Theorem 9.

Bob:

(1) For $l = 1, 2, \ldots, L$, in the lth level, Bob receives the description of the functions $(\phi[t])_{t\in[n'_l]}$, and the redundancies z_θ, z_I. Finally he receives z_x.

(1.1) Partition the string $\tilde{x}$ evenly into $n'_l = n/T_l$ blocks, each of size T_l.

(1.2) For any $t \in [n'_l]$, denote $S_t = \{\tilde{x}[u, u+B) \mid |T(t-1)+1-u| < T_l \log^{0.6} n, 1 \le u \le n-B+1\}$. Bob applies Lemma 14 using S_t for any $t \in [n'_l]$, and obtains $(\theta'[t])_{t\in[n'_l]}$.

(1.3) Bob packs every successive T' elements of $(\theta'[t])_{t\in[n'_l]}$ into a vector $V'_\theta = (\theta'[1, T'], \theta'[T'+1, 2T'], \ldots)$. Then he uses the redundancy z_θ and V'_θ to recover V_θ. Bob unpacks $(\theta[t])_{t\in[n'_l]}$ from V_θ to obtain Φ.

(1.4) For any $t \in [n'_l]$, Bob evaluates the hash function $\Phi[t]$ on the tth block of $\tilde{x}$, so he obtains the hash values $I'[t] = \Phi[t](\tilde{x}[T_l(t-1)+1, T_l(t-1)+B])$. Bob packs every successive T'' elements of I' into the a vector V'_I, i.e., $V'_I = (I'[1, T''], I'[T''+1, 2T''], \ldots)$.

(1.5) Bob uses the redundancy z_I and V'_I to recover V_I, then obtains I from V_I by unpacking.

(1.6) Bob finds the maximum matching Π between x and y under Φ using I. Note that to find such a matching, Bob only needs to know the hash values of $\Phi[t]$ on the tth block of x,

[2]The construction can be extended to general case; see Remarks 4 and 5.

which can be obtained from the vector I. For each pair (a, b) in Π, Bob fills the ath block $\tilde{x}[T_l(a-1)+1, T_l a]$ with $y[b, b+T_l)$.

(2) Bob partitions $\tilde{x}$ evenly into blocks of length T_L, then uses the redundancy z_x to recover x.

4.4 Analysis

Lemma 16. *For Stage I, in Alice's partition of x and Bob's partition of y, the size of every block is greater than $2^s/2$. If all properties in Theorem 14 hold, then in Alice's partition of x, the size of every block is at most $B_1 + B_2 = O(s2^s \log n)$.*

Proof. We first prove that the size of every Alice's block is greater than $2^s/2$. The sizes of Bob's blocks follow in the same way. For any $t = 1, 2, \ldots n' - 1$, let j be the next p-split point of i_t, then $j - i_t > 2^s/2$. As i_{t+1} is also a p-split point, from the Definition 15, we have $i_{t+1} \geq j$. Hence, $i_{t+1} - i_t \geq j - i_t > 2^s/2$. For the first block $x[1, i_2]$, its length is i_2, which is greater than $i_2 - i_1 > 2^s/2$. For the last block $x[i_{n'}, n]$, let j be the next p-split point of $i_{n'}$, then $j \leq n+1$ and $j - i_{n'} > 2^s/2$. Hence, the length of this block is $n - i_{n'} + 1$, which is greater than $2^s/2$.

For sake of contradiction, assume there is a block $x[l, r)$ whose length is greater than $B_1 + B_2$, then by Property 1, there is a p-split point j in the range of $[l+1, l+B_1]$. Applying Property 2, there is a p-split point chosen by Alice in the range of $[j, j+B_2)$. This violates the assumption that $x[l, r)$ is a block. □

Lemma 17. *If $2^s/2 > B$, then the symmetric difference of V and V' has size at most $4k$.*

Proof. Without loss of generality, by symmetry, we only need to prove that the size of $V \setminus V'$ is at most $2k$.

As $\mathsf{ED}(x, y) \leq k$, there exists a series of k edit operations transforming x into y. For any element such that $(\mathsf{len}_b, \text{B-prefix}_b, \text{B-prefix}_{b+1}) \in V$, where b is a block index in the partition of x, if the bth block and the $(b+1)$-th block of x are not involved in the edit operations, then the bth block of x is still a block of some index b' in Bob's string y. By Lemma 16, the sizes of the bth block and the $(b+1)$-th block of x are at least $2^s/2 = 4\log n > B$, and so for the index b' in Bob's string y, we have $\mathsf{len}_{b'} = \mathsf{len}_b$, $\text{B-prefix}_{b'} = \text{B-prefix}_b$, and $\text{B-prefix}_{b'+1} = \text{B-prefix}_{b+1}$. Thus, $(\mathsf{len}_b, \text{B-prefix}_b, \text{B-prefix}_{b+1}) \in V'$. Hence, an element $(\mathsf{len}_b, \text{B-prefix}_b, \text{B-prefix}_{b+1}) \in V \setminus V'$ implies that the bth block or the $(b+1)$-th block of x is involved in the edit operations. So, the size of $V \setminus V'$ is upper bounded by $2k$. □

Theorem 18. *If $2^s/2 > B$, and all properties in Theorem 14 hold, then after Stage I, at most $O(k)$ blocks of $\tilde{x}$ contain unfilled bits or incorrectly filled bits.*

Proof. By Lemma 17, Bob recovers V correctly using the redundancy z_V.

As $\mathsf{ED}(x, y) \leq k$, there exists a series of k edit operations transforming x into y. In Bob's step (5), for every block b, if Bob does not find a block b' in y that matches the B-prefix and the length of b, then the block b must be involved in an edit operation. Hence, the number of such blocks is at most k.

For the case where Bob finds at least two blocks $b_1'^{(b)}$ and $b_2'^{(b)}$ in y such that the B-prefix and the length of $b_1'^{(b)}, b_2'^{(b)}$ both match that of b, one of $b_1'^{(b)}$ and $b_2'^{(b)}$ must be involved in an edit operation, otherwise they are both substrings of x and thus violate the B-distinct property. By the B-distinct property of the string x, $\{b_1'^{(b)}, b_2'^{(b)}\}_b$ are disjoint sets for different block b. As the total number of edit operations is at most k, the number of the blocks b that Bob finds multiple b' is at most k.

For the blocks Bob fills in, at most k of them are involved in edit operations, these blocks may be incorrectly filled. But for the remaining blocks, they must be correctly filled due to the B-distinct property. In total, the number of unfilled or incorrectly filled blocks is at most $3k$.

In Bob's step (6), Bob divides $\tilde{x}$ evenly into n/T_0 blocks, each of length T_0. By Lemma 16, the length of each block in step (5) is $O(s2^s \log n)$. As we set $T_0 = s2^s \lceil \log n \rceil$, each unfilled or incorrectly filled block in step (5) can only affect $O(s2^s \log n)/T_0 = O(1)$ blocks in step (6). Hence, the total number of blocks in step (6) containing unfilled or incorrectly filled bits is at most $O(k)$. □

Theorem 19. *If $2^s/2 > B$, $\varepsilon \le 1/3$ and all properties in Theorem 14 hold, then at the end of each level in Stage II, the total number of unmatched blocks and incorrectly matched blocks between x and $\tilde{x}$ is $O(k)$, where the constant hidden in big O notation is independent of the levels.*

Proof. We will prove by induction. We denote the the partition of $\tilde{x}$ in 0th level to be the partition in the last step of Stage I. For the lth level, $l \ge 1$, we define the *bad block* to be the block of $\tilde{x}$ containing unfilled or incorrectly filled bits in the $(l-1)$-th level. For the 0th level, by Theorem 18, the number of the bad blocks is at most $O(k)$, and thus the theorem holds for 0th level. Now, we assume the theorem holds for the $(l-1)$-th level, then the number of the bad blocks in the $(l-1)$-th level is bounded by $O(k)$. As we set $T_l = T_{l-1}/T''$, the length of each bad block is at most $O(T_l \cdot T'')$. Each of the bad blocks is divided evenly into $O(T'')$ smaller successive blocks in step 1 of the lth level.

For each $(\theta[t])_{t \in [n'_l]}$, $\theta[t]$ is computed deterministically by S_t in Construction 4.2.2, where S_t contains all substrings of length B of x in the range of $[T_l(t-1)+1-T_l \cdot \log^{0.6} n, T_l(t-1)+1+T_l \cdot \log^{0.6} n] \cap [1,n]$. As we pack every $T' = \lceil \log^{0.6} n \rceil$ successive $\theta[t]$ into one element in V_θ, and one block in x contains T_l bits, one bad block can only cause $O(T_l T'' + T_l \log^{0.6} n)/T_l T' = O(1)$ Hamming errors between V_θ and V'_θ. Hence, there are at most $O(k)$ Hamming errors between V_θ and V'_θ. Bob can thus recover Alice's V_θ correctly using the redundancy z_θ.

After Bob correctly recovers Φ, he evaluates the ε-synchronization hash function on each block. One *bad block* can only cause $O(T'')$ successive Hamming errors between I and I'. So, after packing every T'' successive hash value, the number of the Hamming errors between the vector V_I and $V_{I'}$ is upper bounded by $O(k)$. Hence, the redundancy z_I allows Bob to recover V_I.

Now Bob obtains the ε-synchronization hash functions Φ and its hash values, he computes $\Pi = \mathsf{MATCH}_\Phi(x_T, y)$ using the hash values in I. Since $\varepsilon \le 1/3$, by Theorem 17, the number of the bad pairs is upper bounded by $O(k)$. □

Proof of Theorem 13. We choose $s = \lceil \log \log n \rceil + 3$, so $2^s/2 \ge 4 \log n > B$. With probability $1 - 1/\mathrm{poly}(n)$, all properties in Theorem 14 hold. In each level in Stage II, we use a sequence of ε-synchronization hash functions with $\varepsilon = 1/3$, by the construction in Section 4.2.2.

By Theorem 19, after Stage II ends, there are $O(k)$ blocks of length B that are incorrectly matched or unmatched between x and $\tilde{x}$. Hence, the redundancy z_x allows Bob to recover x correctly.

In Stage I, Alice sends the redundancy z_V to Bob. The dimension of the indicator vector of the set V is $2^{2B}\mathrm{poly}\log n = \mathrm{poly}(n)$. If we use Reed-Solomon code or Theorem 9 to generate z_V, then the size of the redundancy z_V is $O(k \log n)$.

Stage II has a constant number of levels. In each level, Alice sends the description of $(\phi[t])_{t \in [n'_l]}, z_\theta, z_I$ to Bob. Finally Alice sends z_x to Bob. By Lemma 13, the description of $(\phi[t])_{t \in [n'_l]}$ has size $O(\log n)$. For the redundancy z_θ, the size of each element in V_θ is $O(\log^{0.6} n) \cdot O(\log \log n)$ bits, which is smaller than $O(\log n)$ bits. Hence, the size of z_θ is $O(k \log n)$. For z_I, the hash value of $\phi[t], (t \in [n'_l])$ can be stored in R bits, and the hash value of $\theta[t], (t \in [n'_l])$ can be stored in $O(\log \log n)$ bits, so the size of each element in V_I is $O(\log^{0.4} n) \cdot (R + O(\log \log n)) < O(\log n)$ bits. Hence, the size of z_I is $O(k \log n)$ bits. For z_x, its size is $O(kB) = O(k \log n)$ bits. Thus, in total the size of the redundancy is $O(k \log n)$. □

Remark 5. Again, the above proofs work when n is a multiple of T_0 and all subsequent T_{l-1}'s are multiples of T''. If this is not the case, then as stated in Remark 4, we can always pad a string of uniform random bits to both strings at the beginning to change n into n'', so n'' is of the form $BT_0(T'')^c$ for some integer c, while B, T_0 and T'' remain the same functions of n. Note that n'' is at most n^2 and so $\log n'' \leq 2 \log n$. The number of rounds in our protocol stays the same, while in each round the communication complexity increases by a factor of at most 2. Thus, the communication complexity of the whole protocol also increases by a factor of at most 2.

5 EXPLICIT BINARY ECC FOR EDIT ERRORS

In this section, we will show how to use the document exchange protocol for uniform random strings in Section 4 to construct ECC for edit errors.

Our general strategy is as follows: For any given message x, we use a PRG to generate a mask string s.t. the xor y of the mask and the message has the three properties: Property 1, Property 2, B-distinct. We ensure that the seed length of the PRG is small enough s.t. we can exhaustively search the seed s.t. y has the three properties. Then, we apply the method in Section 4 to create a sketch z for y. After that, we use an asymptotically good binary ECC for edit errors to encode the redundancy and the seed. Concatenating this with y gives the final codeword.

5.1 The Generator for the Mask

We show that there exists an explicit generator of seed length $O(\log n)$ s.t. given any message $x \in \{0,1\}^n$, w.h.p. the xor of x and the output of the generator has Property 1, Property 2, and B-distinct, where the randomness is over the uniform random seed of the generator. Formally, we have the following theorem:

Theorem 20. *There exists an explicit PRG $g : \{0,1\}^r \to \{0,1\}^n$ s.t. for every $x \in \{0,1\}^n$, with probability $1-1/\text{poly}(n)$, $g(U_r)+x$ satisfies Property 1, Property 2, and B-distinct, where $r = O(\log n)$. (Let s in Property 1, Property 2 be $\log\log n + O(1)$.)*

The generator g is the xor of three generators, each of which generates a string satisfying one of the three properties. We utilize random walks on expander graphs and PRG for AC^0 circuits to reduce the seed length of generators for Property 1 and Property 2. And, we use almost κ-wise independence generator for B-distinct.

Lemma 18. *There exists an explicit PRG $g_1 : \{0,1\}^{r_1} \to \{0,1\}^n$ s.t. for every $x \in \{0,1\}^n$, with probability $1 - 1/\text{poly}(n)$, $g_1(U_{r_1}) + x$ satisfies Property 1, where $r_1 = O(\log n)$.*

Proof. We cut x to be a sequence of blocks $x[1], \dots, x[t]$, each of length $B_1' = \lceil B_1/2 \rceil = O(s2^s \log n)$. W.l.o.g., we assume every n is dividable by B_1'. Since if not, we can simply ignore the last block whose length is less than B_1'. So, $t = n/B_1'$.

We show how to generate a block $v \in \{0,1\}^{B_1'}$ s.t. $x[1]+v$ contains a p-split point with probability $1 - 1/\text{poly}(n)$.

We further divide $x[1]$ into a sequence of $t_0 = B_1'/b_0 = O(\log n)$ smaller blocks, each of length $b_0 = \Theta(s2^s)$. Correspondingly, we view v as a sequence of t_0 blocks, each of length b_0.

For every $i \in [t_0]$, if $v[i]$ is uniform, then the probability that there is a p-split point in $x[1][i] + v[i]$ is at least $2/3$. This is because for every consecutive s bits, the probability that it is p is $\frac{1}{2^s}$. Since there are b_0/s number of distinct s-bits substrings in $x[1][i] + v[i]$, the probability that none of them is p is $(1 - \frac{1}{2^s})^{b_0/s}$. So, the probability that at least one of them is p is $1 - (1 - \frac{1}{2^s})^{b_0/s} = 1 - (1 - \frac{1}{2^s})^{O(2^s)} \geq 2/3$ if we let the constant in b_0 to be large enough. Note that checking whether $x[1][i] + v[i]$ has a p-split point can be done by a CNF/DNF f of size $m = O(b_0 s)$, since we can set up a test for every consecutive s-bits substring and each test checks whether the corresponding s-bits substring is p. So, if we instead use the PRG from Theorem 5, with error $\tilde{\varepsilon} = 1/6$, to generate $v[i]$, then

$$\Pr\left[f\left(x[1][i] + v[i]\right) = 1\right] \geq \Pr\left[f\left(x[1][i] + U_{b_0}\right) = 1\right] - \tilde{\varepsilon} \geq 2/3 - 1/6 = 1/2,$$

where the seed length is $\tilde{r} = O\left(\log^2 m \log\log m\right) = \mathrm{poly}\left(\log\log n\right)$.

We generate v by doing a random walk of length $t_0 = O(\log n)$, on an $(2^{\tilde{r}}, \Theta(1), \lambda)$-expander graph with constant $\lambda \in (0, 1)$. The expander we use is from the strongly explicit expander family of Theorem 8, so the random walk can be done in polynomial time.

Denote the vertices reached sequentially in the random walk as $w_1, w_2, \ldots, w_{t_0} \in \{0, 1\}^{\tilde{r}}$. The total number of random bits used here is $r_1 = \tilde{r} + O(t_0) = O(\log n)$. Note that for every i, the probability that $x[1][i] + v[i]$ contains a p-split point is $\Pr[f(x[1][i] + v[i]) = 1] \geq 1/2$. By Theorem 7, the probability that $x[1] + v$ does not contain a p-split point is

$$\Pr[\forall i, f(x[1][i] + v[i]) = 0] \leq \left(\frac{1}{2} + \lambda\right)^{O(t_0)} = 1/\mathrm{poly}(n),$$

if we pick the constant in t_0 to be large enough and λ to be some small constant, e.g., 1/3.

Let $g_1(U_{r_1}) = v^t$. By a union bound, with probability $1 - t/\mathrm{poly}(n) = 1 - 1/\mathrm{poly}(n)$, every block of $x + g_1(U_{r_1})$ has a p-split point. Note that $r_1 = \tilde{r} + O(\log n) = O(\log n)$. Since $B_1' = \lceil B_1/2 \rceil$, every interval of length B_1 must cover a whole block. Hence, there must be a p-split point in the interval. □

Next, we give a generator for Property 2.

Lemma 19. *There exists an explicit PRG $g_2 : \{0, 1\}^{r_2} \to \{0, 1\}^n$ s.t. for every $\forall x \in \{0, 1\}^n$, with probability $1 - 1/\mathrm{poly}(n)$, $g_1(U_{r_2}) + x$ satisfies Property 2, where $r_2 = O(\log n)$.*

Proof. Again let us view x as a sequence of blocks, each of length $B_2' = \lceil (B_2 - s)/2 \rceil$. W.l.o.g., we assume every n is dividable by B_2', since if not, we can simply ignore the last block, which is not a whole block. Denote $t = n/B_2'$.

We show how to generate a block $v \in \{0, 1\}^{B_2'}$ s.t. with probability $1 - 1/\mathrm{poly}(n)$, $x[1] + v$ contains an interval of length at least $2^s/2 + s$, which does not contain a substring p.

We further divide $x[1]$ into a sequence of $t_0 = B_2'/b_0 = O(\log n)$ smaller blocks, each of length $b_0 = 2^s/2 + s$. Correspondingly, we view v as a sequence of t_0 blocks, each of length b_0.

For every $i \in [t_0]$, if $v[i]$ is uniform, then we claim that the probability that $x[1][i] + v[i]$ contains a substring p is at most $\frac{1}{2^s} \cdot (\frac{2^s}{2}) = \frac{1}{2}$. This is because for a fixed consecutive s bits the probability, that it is p, is $\frac{1}{2^s}$. There are at most $\frac{2^s}{2}$ different intervals of length s. The claim holds by a union bound.

We generate v by doing a random walk, of length $t_0 = O(\log n)$, on an $(2^{b_0}, \Theta(1), \lambda)$-expander graph with constant $\lambda \in (0, 1)$. The expander we use is from the strongly explicit expander family of Theorem 8, so the random walk can be done in polynomial time.

Denote the vertices reached sequentially in the random walk as $w_1, w_2, \ldots, w_{t_0} \in \{0, 1\}^{b_0}$. The total length of random bits used here is $r_2 = b_0 + O(t_0) = O(\log n)$. Note that for every i, the probability that $x[1][i] + v[i]$ contains p is at most 1/2. By Theorem 7, the probability that every block of $x[1] + v$ contains p is

$$\Pr[\forall i, x[1][i] + v[i] \text{ contains a substring } p] \leq \left(\frac{1}{2} + \lambda\right)^{O(t_0)} = 1/\mathrm{poly}(n),$$

if we pick the constant in t_0 to be large enough and λ to be some small constant, e.g., 1/3.

Let $g_2(U_{r_2}) = v^t$. By a union bound, the probability that every block of $x + g_2(U_{r_2})$ has at least 1 sub-block, which does not contain p as a substring, is at least $1 - t/\mathrm{poly}(n) = 1 - 1/\mathrm{poly}(n)$.

Note that for every interval of length $2B_2' + s \leq B_2$ starting at a p-split point i, the last $2B_2'$ bits must contain a block of length b_0, which does not contain substring p. Assume it is $x[j, j']$. We can find the left closest p-split point to the left end of $x[j, j']$. Assume its index is j_1. Note that this means $i \leq j_1 \leq j - 1$. Assume the right closest p-split point to $x[j' - s + 1]$ is j_2. We know $j_2 > j' - s + 1$. So, $j_2 - j_1 > j' - s + 1 - j + 1 = b_0 + 1 - s > 2^s/2$.

Thus, Property 2 is satisfied. □

Lemma 20. *There exists an explicit PRG $g_3 : \{0,1\}^{r_3} \to \{0,1\}^n$ s.t. for every $x \in \{0,1\}^n$, with probability* $1 - 1/\text{poly}(n)$, $g_3(U_r) + x$ *is B-distinct, where* $r_3 = O(\log n)$.

Proof. Let g_3 be the ε-almost κ-wise independence generator from Theorem 6, where $\varepsilon = 1/\text{poly}(n)$, $\kappa = 2B$, seed length $r_3 = O(\log \frac{\kappa \log n}{\varepsilon}) = O(\log n)$.

Given a fixed x, $g_3(U_{r_3}) + x$ is ε-almost κ-wise independent. So, for every pair of two intervals $u, v \in \{0,1\}^B$ of it,

$$\Pr[u = v] = \sum_{a \in \{0,1\}^B} \Pr[u = a, v = a] \leq \sum_{a \in \{0,1\}^B} \left(\frac{1}{2^{2B}} + \varepsilon\right) \leq \frac{1}{2^B} + 1/\text{poly}(n) \leq \frac{1}{2^{B-1}},$$

where the first inequality is due to the definition of ε-almost κ-wise independence. The second inequality holds, since $\varepsilon = 1/\text{poly}(n)$ is small enough. The third inequality is because ε is small enough and we can take the constant in $B = O(\log n)$ to be large enough. By a union bound over all $O(n^2)$ pair of u, v, it concludes that $g_3(U_{r_3}) + x \in \{0,1\}^n$ is B-distinct with probability $1 - 1/\text{poly}(n)$, since B is large enough. □

Now, we can prove Theorem 20.

Proof of Theorem 20. Let $g(U_r) = g_1(U_{r_1}) + g_2(U_{r_2}) + g_3(U_{r_3})$, where $U_r = (U_{r_1}, U_{r_2}, U_{r_3})$ is uniform random, $r = r_1 + r_2 + r_3 = O(\log n)$.

Fix $x \in \{0,1\}^n$. Note that $U_{r_1}, U_{r_2}, U_{r_3}$ are independent. So, we have the following:

By Lemma 18, with probability $1 - 1/\text{poly}(n)$, $g(U_r) + x = g_1(U_{r_1}) + (x + g_2(U_{r_2}) + g_3(U_{r_3}))$ has Property 1.

By Lemma 19, with probability $1 - 1/\text{poly}(n)$, $g(U_r) + x = g_2(U_{r_2}) + (x + g_1(U_{r_1}) + g_3(U_{r_3}))$ has Property 2.

By Lemma 20, with probability $1 - 1/\text{poly}(n)$, $g(U_r) + x = g_3(U_{r_3}) + (x + g_1(U_{r_1}) + g_2(U_{r_2}))$ has B-distinct.

So, by the union bound, with probability $1 - 1/\text{poly}(n)$, $g(U_r) + x \in \{0,1\}^n$ has Property 1, Property 2 and B-distinct. □

5.2 Constructing Binary ECC Using Redundancies

Based on our results of document exchange protocols, we can construct binary codes that can correct up to k edit errors. The idea is that the sketch sent by Alice in the document exchange protocol can be used to reconstruct the original message.

Lemma 21. *For every $n, r, k \in \mathbb{N}$, every $x \in \{0,1\}^n$, $z \in \{0,1\}^r$, if C is a binary code with message length r, codeword length n_C, that can correct up to $4k$ edit errors, then for every $y \in \{0,1\}^*$ s.t.* $\mathsf{ED}(y, x \circ C(z)) \leq k$, *one can get z from y using the decoding algorithm of C.*

Proof. We can run the decoding of C on $y[n + 1 - k, |y|]$, where $|y| \leq n + n_C + k$. Since there are at most k edit operations, $\mathsf{LCS}(y[n + 1 - k, |y|], C(z)) \geq n_C - k$. So,

$$\begin{aligned} \mathsf{ED}(y[n+1-k, |y|], C(z)) &= |y[n+1-k, |y|]| + n_C - 2\mathsf{LCS}(y[n+1-k, |y|], C(z)) \\ &\leq |y[n+1-k, |y|]| + n_C - 2(n_C - k) \\ &= |y| - (n + 1 - k) + 1 + n_C - 2(n_C - k) \\ &\leq (n + n_C + k) - (n + 1 - k) + 1 + n_C - 2(n_C - k) \\ &\leq 4k. \end{aligned} \tag{13}$$

Thus, the decoding can output the correct z. □

THEOREM 21. *If there exists an explicit document exchange protocol with communication complexity $r(n,k)$, where $n \in \mathbb{N}$ is the input size and $k \in \mathbb{N}$ is the upper bound on the edit distance, then there exists an explicit family of binary ECCs with codeword length $n_C = n + O(r(n,k))$, message length n, that can correct up to k edit errors.*

PROOF. The encoding algorithm is as follows: For message $x \in \{0,1\}^n$, we first compute the redundancy z for x and $3k$ edit errors using the document exchange protocol. Then, we encode the redundancy z to be $\tilde{c}$ using the binary ECC from Theorem 10, which can correct α fraction of errors with constant rate. Here, α is a constant such that $\alpha|\tilde{c}| \geq 4k$. Assume the length of the code is $n_0 = O(r)$. The final codeword c is the concatenation of the original message x and the encoded redundancy $\tilde{c}$. That is, $c = x \circ \tilde{c}$ and $|c| = n + n_0$.

By Lemma 21, we can get the redundancy z from the corrupted codeword c'. Note that there are at most k edit errors, $\mathsf{ED}(c'[1,n+k],x) = n+k+n-2\mathsf{LCS}(c'[1,n+k],x) \leq n+k+n-2(n-k) \leq 3k$. Here, $\mathsf{LCS}(c'[1,n+k],x) \geq n-k$ is because $c'[1,n+k]$ contains a subsequence of x, which are the symbols that are not deleted. There are at most k deletions so the subsequence has length at least $n-k$.

Finally, we run the document exchange protocol to compute the original message x, where Bob's string is $c'[1,n+k]$ and the redundancy from Alice is z. □

5.3 Binary ECC for Edit Errors with Almost Optimal Parameters

A direct corollary of Theorem 21 is the following binary ECC:

THEOREM 22. *For any $n,k \in \mathbb{N}$ with $k \leq n/4$, there exists an explicit binary error correcting code with message length n, codeword length $n + O(k \log^2 \frac{n}{k})$ that can correct up to k edit errors.*

PROOF. It follows directly from Theorems 21 and 12. □

COROLLARY 23. *There exists a constant $0 < \alpha < 1$ such that for any $0 < \varepsilon \leq \alpha$ there exists an explicit family of binary error correcting codes with codeword length n and message length m, which can correct up to $k = \varepsilon n$ edit errors with rate $m/n = 1 - O(\varepsilon \log^2 \frac{1}{\varepsilon})$.*

By utilizing the document exchange protocol in Section 4, the generator from Theorem 20, and the concatenation technique of Lemma 21, we can get a binary ECC for edit errors with even better parameters (in fact, optimal redundancy) for any $k = O(n^a)$, where a can be any constant less than 1.

THEOREM 24. *For any $n,k \in \mathbb{N}$, there exists an explicit binary error correcting code with message length n, codeword length $n + O(k \log n)$ that can correct up to k edit errors.*

PROOF. Without loss of generality, we can assume that $k \leq n/4$, since for larger k, we can just use the asymptotically good code from Theorem 10, where the codeword length is $O(k)$.

Fix a message $x \in \{0,1\}^n$. Let g be the generator from Theorem 20. We know that with probability $1 - 1/\mathsf{poly}(n)$, $x + g(U_r)$ has Property 1, Property 2, and B-distinct. Using an exhaustive search, we can find a seed $u \in \{0,1\}^r$ s.t. $y = x + g(u)$ has the three properties. This takes polynomial time, since $r = O(\log n)$.

Then, we apply the method in Section 4 to create a redundancy z for y, which can correct $3k$ errors, where $|z| = O(k \log n)$. We further add u to z to form a new sketch $v = z \circ u$ that has size $O(k \log n)$. By the same technique used in the proof of Theorem 21, we can construct an error correcting code for y with codeword length $n + O(k \log n)$, where the sketch is $v = z \circ u$. For the sake of completeness, we present the full construction below.

We first encode the redundancy v to be $\tilde{c}$ using the binary ECC from Theorem 10 that can correct $4k$ errors. The length of the codeword $\tilde{c}$ is $n_0 = O(k \log n)$. The final codeword c is the concatenation $c = y \circ \tilde{c}$ and $|c| = n + O(k \log n)$.

To decode, by Lemma 21, we can recover the redundancy $v = z \circ u$ from the corrupted codeword c'. We then run the document exchange protocol to compute the processed message y, where Bob's string is $c'[1, n+k]$ and the redundancy from Alice is z. Finally, we recover x from $x = y + g(u)$. □

6 DISCUSSIONS AND OPEN PROBLEMS

In this article, we constructed deterministic document exchange protocols and binary error correcting codes for edit errors. Our results significantly improve previous results, and in particular, we have obtained the first explicit constructions of binary insdel codes that are optimal or almost optimal for a wide range of error parameters k. We introduced several new techniques, most notably ε-self-matching hash functions and ε-synchronization hash functions. We note that while similar in spirit to ε-self-matching strings and ε-synchronization strings introduced in Reference [17], there are many important differences between these objects. For example, the objects we introduced are hash functions, while the objects in Reference [17] are fixed strings. In particular, although we can also show that random functions are ε-self-matching hash functions and ε-synchronization hash functions, the deterministic constructions of ε-self-matching hash functions and ε-synchronization hash functions depend on the string x. In contrast, ε-self-matching strings and ε-synchronization strings do not depend on any input string.

We also stress that, in general, larger alphabets make these constructions easier, since there are more structures in codes over larger alphabets. Furthermore, codes for Hamming errors (which we use a lot in our constructions) over larger alphabets have better parameters than those over the binary alphabet.

We believe our techniques can be useful in other applications, and one natural open problem is to get optimal deterministic document exchange protocols and binary insdel codes for all error parameters.

APPENDIX

A ASYMPTOTIC BEHAVIOR OF THE REDUNDANCY OF DOCUMENT EXCHANGE AND INSDEL CODES

It appears to be a folklore that for $k \le n/4$, the optimal redundancy of document exchange and insdel codes for k errors is $\Theta(k \log \frac{n}{k})$. For completeness, we provide a justification in this section.

Note that by our transformation from a document exchange protocol to an insdel code in Theorem 21, a document exchange protocol with redundancy r implies an insdel code with redundancy $O(r)$. Thus, an upper bound on the redundancy of document exchange implies roughly the same upper bound on the redundancy of insdel codes. Conversely, a lower bound on the redundancy of insdel codes implies roughly the same lower bound on the redundancy of document exchange.

We now prove the following two theorems:

Theorem 25 (Upper Bound). *For any $k \le n/4$ there exist document exchange protocols and error correcting codes against k insdel errors with redundancy $O(k \log \frac{n}{k})$.*

Proof. We use a simple graph coloring argument to design the protocol. Note that Alice's string has length n and Bob's string has length at most $n + k \le 5n/4$. We construct a bipartite graph with Alice's 2^n possible strings being the vertices on the left, and Bob's at most $n2^{5n/4} = 2^{O(n)}$ strings being the vertices on the right, such that there is an edge between two vertices iff the edit distance of these two strings is at most k.

We now claim that the right degree D_R of this graph is bounded by $(\frac{n}{k})^{O(k)}$. To see this, suppose a right vertex y is connected to a left vertex x. Then x and y has edit distance at most k, and every other left vertex x' connected to y also has edit distance at most k to y. Thus, by the triangle inequality every other left vertex x' connected to y has edit distance at most $2k$ to x. This implies that $\mathsf{LCS}(x, x') \geq n - k$. Therefore, the number of such left vertices is bounded by $\left(\binom{n}{n-k}\right)^2 \leq (\frac{en}{k})^2 = (\frac{n}{k})^{O(k)}$. Similarly, the left degree D_L is also bounded by $(\frac{n}{k})^{O(k)}$, since for any left vertex, all its neighbors have edit distance at most k to it.

We can now color the left vertices using at most $D_R D_L$ colors, such that for every right vertex, all its left neighbors have different colors, using a greedy approach as follows: We start by picking one left vertex and color it arbitrarily. Then, while there are left vertices not colored, we arbitrarily pick one such left vertex and color it. Note that this vertex can share a right neighbor with at most $D_L(D_R - 1)$ other left vertices. Hence, if we use $D_R D_L$ colors, then we can always pick a color for this vertex, which is different from all the other left vertices with which this vertex shares a right neighbor.

Now for the document exchange protocol, Alice can just send the description of the color corresponding to her string x. Bob can recover x by using the coloring of the graph and his own string y, since all of y's left neighbors have different colors. The communication complexity or the redundancy is at most $\log(D_R D_L) = O(k \log \frac{n}{k})$. □

Theorem 26 (Lower Bound). *For any $k \leq n/4$, any document exchange protocol or error correcting code against k insdel errors must have redundancy $\Omega(k \log \frac{n}{k})$.*

Proof. Note that a code against k insdel errors can also correct $k/2$ Hamming errors. Let the codeword length be $n' > n$. By the standard Hamming bound, we have

$$2^n \leq \frac{2^{n'}}{\sum_{i=0}^{k/2} \binom{n'}{i}} \leq \frac{2^{n'}}{\sum_{i=0}^{k/2} \binom{n}{i}} \leq \frac{2^{n'}}{(2n/k)^{k/2}}.$$

Thus, $n \leq n' - \frac{k}{2} \log \frac{2n}{k}$ and $n' - n \geq \frac{k}{2} \log \frac{2n}{k}$. □

ACKNOWLEDGMENTS

Most of the work was done while the Kuan Cheng and the Ke Wu were graduate students at Johns Hopkins University, USA.

REFERENCES

[1] Noga Alon, Uriel Feige, Avi Wigderson, and David Zuckerman. 1995. Derandomized graph products. *Computat. Complex.* 5, 1 (1995), 60–75.
[2] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. 1992. Simple constructions of almost k-wise independent random variables. *Rand. Struct. Algor.* 3, 3 (1992), 289–304.
[3] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach.* Cambridge University Press.
[4] Djamal Belazzougui. 2015. Efficient deterministic single round document exchange for edit distance. *arXiv preprint arXiv:1511.09229* (2015).
[5] Djamal Belazzougui and Qin Zhang. 2016. Edit distance: Sketching, streaming, and document exchange. In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science.* IEEE, 51–60.
[6] Joshua Brakensiek, Venkatesan Guruswami, and Samuel Zbarsky. 2017. Efficient low-redundancy codes for correcting multiple deletions. *IEEE Trans. Inf. Theor.* 64, 5 (2017), 3403–3410.
[7] Boris Bukh and Venkatesan Guruswami. 2016. An improved bound on the fraction of correctable deletions. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms.* ACM, 1893–1901.
[8] Boris Bukh, Venkatesan Guruswami, and Johan Håstad. 2017. An improved bound on the fraction of correctable deletions. *IEEE Trans. Inf. Theor.* 63, 1 (2017), 93–103. DOI: https://doi.org/10.1109/TIT.2016.2621044

[9] Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. 2016. Low distortion embedding from edit to Hamming distance using coupling. In *Proceedings of the 48th IEEE Annual Annual ACM SIGACT Symposium on Theory of Computing*. ACM.

[10] Kuan Cheng, Bernhard Haeupler, Xin Li, Amirbehshad Shahrasbi, and Ke Wu. 2019. Synchronization strings: Highly efficient deterministic constructions over small alphabets. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2185–2204.

[11] Graham Cormode, Mike Paterson, Suleyman Cenk Sahinalp, and Uzi Vishkin. 2000. Communication complexity of document exchange. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM, 197–206.

[12] Anindya De, Omid Etesami, Luca Trevisan, and Madhur Tulsiani. 2010. Improved pseudorandom generators for depth 2 circuits. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer Berlin, 504–517.

[13] Arnaldo Garcia and Henning Stichtenoth. 1996. On the asymptotic behaviour of some towers of function fields over finite fields. *J. Numb. Theor.* 61, 2 (1996), 248–273.

[14] Venkatesan Guruswami and Ray Li. 2016. Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes. In *Proceedings of the IEEE International Symposium on Information Theory*. IEEE, 620–624.

[15] Venkatesan Guruswami and Carol Wang. 2017. Deletion codes in the high-noise and high-rate regimes. *IEEE Trans. Inf. Theor.* 63, 4 (2017), 1961–1970.

[16] Bernhard Haeupler. 2019. Optimal document exchange and new codes for insertions and deletions. In *Proceedings of the IEEE 60th Annual Symposium on Foundations of Computer Science*. IEEE, 334–347.

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

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

[19] Tom Høholdt, Jacobus H. Van Lint, and Ruud Pellikaan. 1998. Algebraic geometry codes. *Handb. Coding Theor.* 1, Part 1 (1998), 871–961.

[20] Shlomo Hoory, Nathan Linial, and Avi Wigderson. 2006. Expander graphs and their applications. *Bull. Amer. Math. Soc.* 43, 4 (2006), 439–561.

[21] Utku Irmak, Svilen Mihaylov, and Torsten Suel. 2005. Improved single-round protocols for remote file synchronization. In *Proceedings of the IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*. IEEE, 1665–1676.

[22] Hossein Jowhari. 2012. Efficient communication protocols for deciding edit distance. In *Proceedings of the European Symposium on Algorithms*. Springer, 648–658.

[23] Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.* 10, 8 (1966), 707–710.

[24] Hugues Mercier, Vijay K. Bhargava, and Vahid Tarokh. 2010. A survey of error-correcting codes for channels with symbol synchronization errors. *IEEE Commun. Surv. Tutor.* 12, 1 (2010), 87–96.

[25] Alon Orlitsky. 1991. Interactive communication: Balanced distributions, correlated files, and average-case complexity. In *Proceedings of the 32nd Annual Symposium of Foundations of Computer Science*. IEEE Computer Society, 228–238.

[26] Leonard J. Schulman and David Zuckerman. 1999. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE Trans. Inf. Theor.* 45, 7 (1999), 2552–2557.

[27] Kenneth W. Shum, Ilia Aleshnikov, P. Vijay Kumar, Henning Stichtenoth, and Vinay Deolalikar. 2001. A low-complexity algorithm for the construction of algebraic-geometric codes better than the Gilbert-Varshamov bound. *IEEE Trans. Inf. Theor.* 47, 6 (2001), 2225–2241.

[28] Jin Sima and Jehoshua Bruck. 2021. On optimal k-deletion correcting codes. *IEEE Trans. Inf. Theor.* 67, 6 (2021), 3360–3375.

[29] G. M. Tenengolts and R. R. Varshamov. 1965. Code correcting single asymmetric errors(binary code correcting single asymmetric errors). *Avtomatika I Telemekhanika* 26 (1965), 288–292.

Received 3 August 2021; revised 29 April 2022; accepted 5 August 2022