# Multiple Criss-Cross Insertion and Deletion Correcting Codes

Lorenz Welter, *Graduate Student Member, IEEE*, Rawad Bitar, *Member, IEEE*, Antonia Wachter-Zeh, *Senior Member, IEEE*, and Eitan Yaakobi, *Senior Member, IEEE*

***Abstract*— This paper investigates the problem of correcting multiple criss-cross insertions and deletions in arrays. More precisely, we study the unique recovery of $n \times n$ arrays affected by *t-criss-cross deletions* defined as any combination of $t_\mathrm{r}$ row and $t_\mathrm{c}$ column deletions such that $t_\mathrm{r} + t_\mathrm{c} = t$ for a given $t$. We show an equivalence between correcting $t$-criss-cross deletions and $t$-criss-cross insertions and show that a code correcting $t$-criss-cross insertions/deletions has redundancy at least $tn + t \log n - \log(t!)$. Then, we present an existential construction of a $t$-criss-cross insertion/deletion correcting code with redundancy bounded from above by $tn + \mathcal{O}(t^2 \log^2 n)$. The main ingredients of the presented code construction are systematic binary $t$-deletion correcting codes and Gabidulin codes. The first ingredient helps locating the indices of the inserted/deleted rows and columns, thus transforming the insertion/deletion-correction problem into a row/column erasure-correction problem which is then solved using the second ingredient.**

***Index Terms*— Insertion/deletion correcting codes, array codes, criss-cross deletion errors.**

## I. Introduction

CODES correcting insertions and deletions have recently witnessed an increased attention due to their application in DNA-based storage systems, file synchronization, and communication systems [2]–[8]. The problem of correcting insertions and deletions, referred to as *indel* errors, dates back to the 1960s. In [9], Levenshtein defined the notion of $t$-deletion-correcting codes and showed that a code can correct any combination of $t$ indels if and only if it can correct $t$ deletions. The main property of the codes that is usually optimized is the redundancy defined as $R \triangleq n - \log|\mathcal{C}|$ where $n$ is the length of the codewords in $\mathcal{C}$ and $|\mathcal{C}|$ is the cardinality of the code. Levenshtein bounded the redundancy of any binary $t$-indel-correcting code from below by $t \log n - \mathcal{O}(1)$. Moreover, he proved that the Varshamov-Tenengolts codes [10], originally designed to correct a single asymmetric error, can also correct a single indel and have redundancy roughly $\log(n+1)$ bits. Several recent works studied the problem of constructing binary $t$-indel-correcting codes, for $t > 1$, with redundancy approaching Levenshtein's bound [11]–[17]. Of particular importance to us is the work of Sima *et al.* [18] in which the authors present a binary systematic $t$-indel-correcting code with redundancy $4t\log(n) + o\log(n)$. This code can correct any combination of $t$ indel errors.

This paper considers the problem of coding for indels in the two-dimensional space. The motivation stems from the two-dimensional erasure and substitution correction problem where it has been shown that leveraging the structure of the array is more beneficial than applying one-dimensional error correcting codes on each dimension of the array. The deletion (and clearly also the indel) correction problem is however more involved due to the loss of synchronization in the locations of the inserted and deleted rows and columns. Along this line of thought, the trace-reconstruction problem, which is related to coding for deletions, is investigated for the two-dimensional space in [19]. Moreover, coding for deletions over the two-dimensional space is also considered in [20]–[22]. In [20], [21] codes that can correct bursts of deletions in the one-dimensional space are constructed. The main idea is to view the codeword as a binary array and use the structure of that array to detect and correct bursts of deletions that happen in the one-dimensional codeword. In [22], the authors consider the problem of database matching under column deletions.

Given a certain number of deletions $t$ and an array $\mathbf{X}$, we assume that the array can be affected by any combination of $t_\mathrm{r}$ row and $t_\mathrm{c}$ column deletions such that $t_\mathrm{r} + t_\mathrm{c} = t$. This type of deletions are referred to as *t-criss-cross deletions*. We define *t-criss-cross insertions* similarly. Our goal is to construct codes that can uniquely recover the array $\mathbf{X}$ from any $t$-criss-cross deletion or any $t$-criss-cross insertion and we refer to these codes as *t-criss-cross indel codes*. We borrow this terminology from previous works that studied the problem of correcting criss-cross erasures and substitution errors in the two-dimensional space, e.g., [23]–[30].

The first works to study the criss-cross deletion problem were [31]–[33]. In [31], [32], we investigated the problem of correcting exactly one row and one column inser-

Manuscript received June 8, 2021; revised November 12, 2021; accepted January 29, 2022. Date of publication February 16, 2022; date of current version May 20, 2022. This work was supported in part by the European Research Council (ERC) through the European Union's Horizon 2020 Research and Innovation Programme under Grant 801434 and in part by the Institute for Advanced Studies, Technical University of Munich, funded by the German Excellence Initiative and European Union Seventh Framework Programme under Grant 291763. An earlier version of this paper was presented in part at the 2021 IEEE International Symposium on Information Theory [1]. *(Corresponding author: Lorenz Welter.)*

Lorenz Welter, Rawad Bitar, and Antonia Wachter-Zeh are with the Institute for Communications Engineering, Technical University of Munich (TUM), 80333 Munich, Germany (e-mail: lorenz.welter@tum.de; rawad.bitar@tum.de; antonia.wachter-zeh@tum.de).

Eitan Yaakobi is with the Computer Science Department, Technion—Israel Institute of Technology, Haifa 3200003, Israel (e-mail: yaakobi@cs.technion.ac.il).

Communicated by L. Dolecek, Associate Editor for Coding Techniques.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIT.2022.3152398.

Digital Object Identifier 10.1109/TIT.2022.3152398

0018-9448 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

tion/deletion in arrays. We showed that the redundancy of codes designed for this special case is bounded from below by $2n + 2\log n - \mathcal{O}(1)$. We also presented an existential and an explicit construction with redundancy approximately $2\log n$ and $7\log n$ far from the lower bound, respectively. Furthermore, we showed that, for $t_\mathrm{r} = t_\mathrm{c}$, a code can correct any $t$-criss-cross deletion if and only if it can correct any $t$-criss-cross insertion. In [33], Hagiwara constructed codes correcting criss-cross deletions with at most $t_\mathrm{r}$ row deletions and at most $t_\mathrm{c}$ column deletions, for given values of $t_\mathrm{r}$ and $t_\mathrm{c}$. The constructed codes have redundancy in the order of $n(t_\mathrm{r}^2 + t_\mathrm{c}^2 + (t_\mathrm{r} + t_\mathrm{c})\log n)$. The construction splits the array into locators and information part. The locators are carefully structured arrays that can exactly recover the index of any deleted rows and columns in the array. Then, a tensor-product erasure-correcting code is used to recover the lost symbols in the information part.

Our contributions in this paper can be summarized as follows. We present an asymptotic upper bound (in the code length) on the cardinality of $t$-criss-cross indel codes. Our bound implies that the redundancy of any $t$-criss-cross indel code is bounded from below by approximately $tn+t\log n$. We extend the equivalence between correcting deletions and insertions to the general $t$-criss-cross deletion model considered in this paper. Then, we construct existential $t$-criss-cross indel codes based on locator arrays, binary systematic $t$-deletion correcting codes, and Gabidulin codes. We also show that this code can correct $t$-criss-cross insertions by providing an explicit decoder. The main improvements of our construction over the one in [33] is to use a collection of binary deletion-correcting codes to locate the indices of the deleted columns and a Gabidulin code to correct the erasures. This significantly reduces the redundancy of the code. However, small locator arrays are still needed to complement the deletion-correcting codes. Then, the deletion-correction problem is transformed into a row/column erasure-correction problem which can be solved by using Gabidulin codes that have optimal redundancy for row/column erasure-correction [24]. The redundancy of the presented construction is $tn+\mathcal{O}(t^2\log^2 n)$. For the considered problem setting, we substantially improve upon the current state-of-the-art construction of [33] that needs a redundancy of approximately $2n \cdot (t^2 + t\log n)$ in this setting.

## II. Definitions and Preliminaries

This section formally defines the codes and notations that are used throughout this paper. Let $\Sigma \triangleq \{0,1\}$ be the binary alphabet. We denote by $\Sigma^{n\times n}$ the set of all binary arrays of dimension $n \times n$. All logarithms are base 2 unless otherwise indicated.

For an integer $n \in \mathbb{N}$, the set $\{1,\ldots,n\}$ is denoted by $[n]$. For an array $\mathbf{X} \in \Sigma^{n\times n}$ and $i,j \in [n]$, we refer to the entry of $\mathbf{X}$ positioned at the $i$ row and the $j$ column by $X_{i,j}$. We denote the $i$ row and the $j$ column of $\mathbf{X}$ by $\mathbf{X}_{i,[n]}$ and $\mathbf{X}_{[n],j}$, respectively. Similarly, we denote by $\mathbf{X}_{[i_1:i_2],[j_1:j_2]}$ the subarray of $\mathbf{X}$ formed by rows $i_1$ to $i_2$ and their corresponding entries from columns $j_1$ to $j_2$. We denote by $\mathbf{X}^T$ the transpose of the array $\mathbf{X}$. Moreover, for two arrays $\mathbf{X} \in \Sigma^{n\times m_1}$ and $\mathbf{Y} \in \Sigma^{n\times m_2}$ we denote by $\mathbf{Z} = (\mathbf{X} \mid \mathbf{Y})$ the concatenation of these two arrays with $\mathbf{Z} \in \Sigma^{n\times(m_1+m_2)}$. For any binary array $\mathbf{X}$, we refer to the complement of $\mathbf{X}$, i.e., every bit in $\mathbf{X}$ is flipped, by $\overline{\mathbf{X}}$. In an array $\mathbf{X} \in \Sigma^{n\times m}$, a column-run of length $r$ is defined as a sequence of $r$ consecutive equal columns $\mathbf{X}_{[n],j} = \mathbf{X}_{[n],j+1} = \cdots = \mathbf{X}_{[n],j+r-1}$. Row-runs in an array $\mathbf{X}$ are defined similarly. Given a vector $\mathbf{x} \in \Sigma^n$ a run of length $r$ in $\mathbf{x}$ is defined as a sequence of $r$ consecutive equal bits $x_i = x_{i+1} = \cdots = x_{i+r-1}$.

For positive integers $t_\mathrm{r}, t_\mathrm{c}$ we define a $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross deletion in a binary array $\mathbf{X}$ to be the deletion of any $t_\mathrm{r}$ rows and $t_\mathrm{c}$ columns of $\mathbf{X}$. For a positive integer $t$, we define a *$t$-criss-cross deletion* in a binary array $\mathbf{X}$ to be the collection of all $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross deletions in $\mathbf{X}$ such that $t_\mathrm{r} + t_\mathrm{c} = t$. Further, $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross insertion and a *$t$-criss-cross insertion* are defined similarly. We denote by $\mathbb{D}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X})$ the set of all arrays that result from $\mathbf{X}$ after a $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross deletion (i.e., the two-dimensional deletion ball[1]). In a similar way we define the set $\mathbb{I}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X})$ for the insertion case. We refer to $\widetilde{\mathbf{X}}$ as the array resulting from a $t$-criss-cross deletion or insertion in $\mathbf{X}$, where the number and type of errors (deletions or insertions) that happened in $\mathbf{X}$ is clear from the context. A code $\mathcal{C} \subseteq \Sigma^{n\times n}$ that can correct any $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross deletion or any $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross insertion is called a *$(t_\mathrm{r}, t_\mathrm{c})$-criss-cross indel-correcting code*. A *$t$-criss-cross indel-correcting code* is defined similarly. We abbreviate those codes as $(t_\mathrm{r}, t_\mathrm{c})$-criss-cross indel code and *$t$-criss-cross indel code*, respectively. Throughout this paper we assume that $t$ is a constant with respect to $n$. We write $f(n) \approx g(n)$, $f(n) \lesssim g(n)$, and $f(n) \gtrsim g(n)$ if the equality or inequality holds for $n \to \infty$.

## III. Equivalence Between Insertion and Deletion Correction

In the following, we show the equivalence between $t$-criss-cross deletion-correcting codes and $t$-criss-cross insertion-correcting codes (Theorem 1). The proof of Theorem 1 follows by first showing that the equivalence holds for all $(t_\mathrm{r}, t_\mathrm{r}+c)$-criss-cross indel codes, where $c$ is a positive integer. Then, by symmetry the equivalence holds for $(t_\mathrm{c}+c, t_\mathrm{c})$-criss-cross indel codes which completes the proof.

*Theorem 1:* A code $\mathcal{C} \subset \Sigma^{n\times n}$ is a $t$-criss-cross deletion-correcting code if and only if $\mathcal{C}$ is a $t$-criss-cross insertion-correcting code.

We need the following results from [31] showing that any $(t_\mathrm{r}, t_\mathrm{r})$-criss-cross deletion-correcting code can also correct insertions and extending the properties of balls intersections from the one-dimensional space to the two-dimensional space for only one indel.

*Theorem 2 ([31]):* For all integers $t_\mathrm{r} \in [n-1]$, a code $\mathcal{C} \subset \Sigma^{n\times n}$ is a $(t_\mathrm{r}, t_\mathrm{r})$-criss-cross deletion-correcting code if and only if it is a $(t_\mathrm{r}, t_\mathrm{r})$-criss-cross insertion-correcting code.

[1] Strictly speaking, the set $\mathbb{D}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X})$ must be called the two-dimensional deletion *sphere* of $\mathbf{X}$. However, we abuse terminology and refer to this set as the deletion ball to follow the nomenclature used by the literature on deletion-correcting codes. The same holds for the set $\mathbb{I}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X})$.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

*Lemma 3 ([31]):* For a positive integer $m$ and two arrays $\mathbf{X} \in \Sigma^{m\times m}$ and $\mathbf{Y} \in \Sigma^{m\times m}$,

$$\mathbb{D}_{1,0}(\mathbf{X}) \cap \mathbb{D}_{1,0}(\mathbf{Y})=\emptyset \Leftrightarrow \mathbb{I}_{1,0}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y})=\emptyset,$$
$$\mathbb{D}_{0,1}(\mathbf{X}) \cap \mathbb{D}_{0,1}(\mathbf{Y})=\emptyset \Leftrightarrow \mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{0,1}(\mathbf{Y})=\emptyset.$$

*Lemma 4 ([31]):* For a positive integer $m$ and two arrays $\mathbf{X} \in \Sigma^{(m+1)\times m}$ and $\mathbf{Y} \in \Sigma^{m\times (m+1)}$,

$$\mathbb{D}_{1,0}(\mathbf{X}) \cap \mathbb{D}_{0,1}(\mathbf{Y})=\emptyset \Leftrightarrow \mathbb{I}_{0,1}(\mathbf{X}) \cap \mathbb{I}_{1,0}(\mathbf{Y})=\emptyset.$$

We now use the previous results to prove Theorem 1.

*Proof:* [Proof of Theorem 1] The goal is to show that for any $\mathbf{X}_1, \mathbf{X}_2 \in \Sigma^{n\times n}$ and any two integers $t_\mathrm{r}$ and $t_\mathrm{c}$ such that $t_\mathrm{r} + t_\mathrm{c} = t$, the following holds.

$$\mathbb{D}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X}_1) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X}_2)=\emptyset \Leftrightarrow \mathbb{I}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X}_1) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{c}}(\mathbf{X}_2)=\emptyset.$$

We only show the equivalence for $(t_\mathrm{r}, t_\mathrm{r} + c)$-criss-cross indel codes, i.e., for any $\mathbf{X}_1, \mathbf{X}_2 \in \Sigma^{n\times n}$ and any integers $t_\mathrm{r}$ and $c$ such that $2t_\mathrm{r} + c = t$ we have

$$\mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_1) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_2) = \emptyset$$
$$\Leftrightarrow \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_1) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_2) = \emptyset.$$

Assuming that the equivalence holds for $(t_\mathrm{r}, t_\mathrm{r} + c)$-criss-cross indel codes, then the following implies that the equivalence holds for $(t_\mathrm{r} + c, t_\mathrm{r})$-criss-cross indel codes which completes the proof.

$$\mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_1) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_2) = \emptyset$$
$$\overset{(a)}{\Leftrightarrow} \mathbb{D}_{t_\mathrm{r}+c,t_\mathrm{r}}(\mathbf{X}_1^T) \cap \mathbb{D}_{t_\mathrm{r}+c,t_\mathrm{r}}(\mathbf{X}_2^T) = \emptyset$$
$$\overset{(b)}{\Leftrightarrow} \mathbb{I}_{t_\mathrm{r}+c,t_\mathrm{r}}(\mathbf{X}_1^T) \cap \mathbb{I}_{t_\mathrm{r}+c,t_\mathrm{r}}(\mathbf{X}_2^T) = \emptyset$$
$$\overset{(c)}{\Leftrightarrow} \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_1) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_2) = \emptyset.$$

The statements (a) and (c) follow trivially by examining the transpose of the arrays $\mathbf{X}_1$ and $\mathbf{X}_2$. The statement (b) follows from the assumed equivalence.

The proof of equivalence for $(t_\mathrm{r}, t_\mathrm{r} + c)$-criss-cross indel codes proceeds by contraposition, i.e., we show that $\mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_1) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_2) \neq \emptyset$ if and only if $\mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_1) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+c}(\mathbf{X}_2) \neq \emptyset$. In what follows, we only show the "only if" parts since the "if" parts follow similarly.

We prove the equivalence for $(t_\mathrm{r}, t_\mathrm{r} + c)$-criss-cross indel codes by induction over $c = t_\mathrm{c} - t_\mathrm{r}$.

### A. Base Case c = 1

For the reader's convenience, a flowchart of the proof is presented in Figure 1.

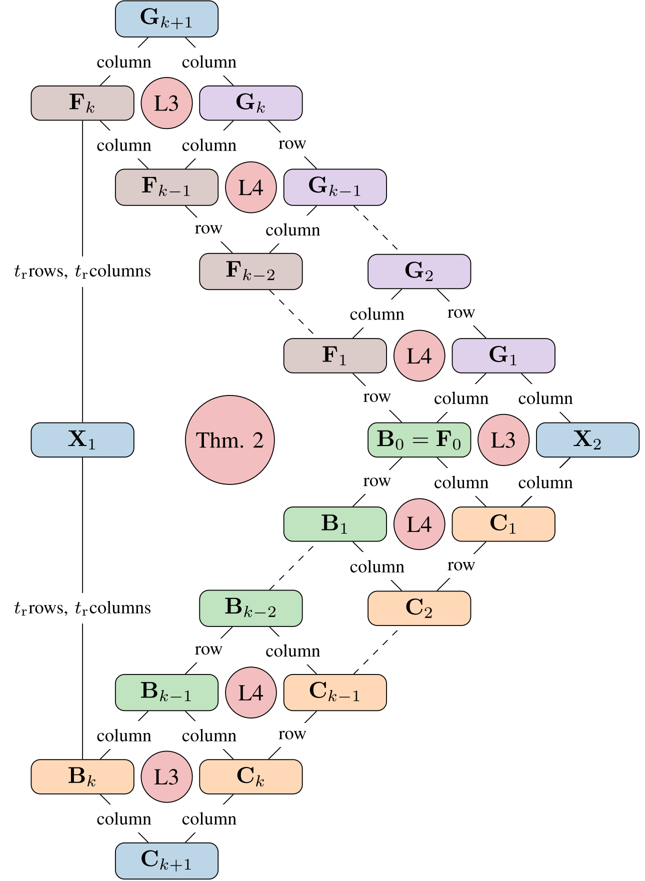


Fig. 1. A flowchart of the proof of Theorem 1. Given an array $\mathbf{C}_{k+1} \in \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{X}_1) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{X}_2)$, we show that there exists an array $\mathbf{G}_{k+1} \in \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{X}_1) \cap \mathbb{I}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{X}_2)$. The series of orange marked arrays are given by the first assumption. Out of these we can proof the existence of the series of green marked arrays by Lemma 3 and 4. The brown marked arrays are then given by applying Theorem 2. Lastly, the existence of the purple arrays can be shown again by Lemma 3 and 4.

Assume that there exists an array $\mathbf{E} \in \Sigma^{(n-t_\mathrm{r})\times(n-t_\mathrm{r}-1)}$ such that $\mathbf{E} \in \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{X}_1) \cap \mathbb{D}_{t_\mathrm{r},t_\mathrm{r}+1}(\mathbf{X}_2)$. Let $k = 2t_\mathrm{r}$, we define the series of arrays $\{\mathbf{C}_s\}_{s=1}^{k}$ to be the intermediate arrays obtained by deleting a row or a column from $\mathbf{X}_2$ to reach $\mathbf{E}$. For notational convenience we let $\mathbf{C}_0 \triangleq \mathbf{X}_2$ and $\mathbf{C}_{k+1} \triangleq \mathbf{E}$. We follow an alternating order of deletion between columns and rows, i.e.,

$$\mathbf{C}_s \in \begin{cases} \mathbb{D}_{0,1}(\mathbf{C}_{s-1}), & \text{if } s \text{ is odd,} \\ \mathbb{D}_{1,0}(\mathbf{C}_{s-1}), & \text{otherwise.} \end{cases}$$

Furthermore, we have $\mathbf{C}_{k+1} \in \mathbb{D}_{0,1}(\mathbf{C}_k)$. We denote by $\mathbf{B}_k \in \Sigma^{(n-t_\mathrm{r})\times(n-t_\mathrm{r})}$ the array resulting from deleting $t_\mathrm{r}$ columns and $t_\mathrm{r}$ rows from $\mathbf{X}_1$ such that $\mathbf{E} \in \mathbb{D}_{0,1}(\mathbf{B}_k)$.

We now want to show that there exists a series of arrays $\{\mathbf{B}_s\}_{s=0}^{k-1}$ such that

$$\mathbf{B}_s \in \begin{cases} \mathbb{I}_{0,1}(\mathbf{B}_{s+1}) \cap \mathbb{I}_{0,1}(\mathbf{C}_{s+1}), & \text{if } s \text{ is odd,} \\ \mathbb{I}_{1,0}(\mathbf{B}_{s+1}) \cap \mathbb{I}_{0,1}(\mathbf{C}_{s+1}), & \text{otherwise.} \end{cases}$$

By definition, $k$ is even and $\mathbf{C}_{k+1} \in \mathbb{D}_{0,1}(\mathbf{B}_k) \cap \mathbb{D}_{0,1}(\mathbf{C}_k)$. By Lemma 3 there exists an array $\mathbf{B}_{k-1} \in \Sigma^{(n-t_\mathrm{r})\times(n-t_\mathrm{r}+1)}$ such that $\mathbf{B}_{k-1} \in \mathbb{I}_{0,1}(\mathbf{B}_k) \cap \mathbb{I}_{0,1}(\mathbf{C}_k)$. Moreover, we know that there exists $\mathbf{C}_{k-1} \in \Sigma^{(n-t_\mathrm{r}+1)\times(n-t_\mathrm{r})}$ such that $\mathbf{C}_k \in \mathbb{D}_{1,0}(\mathbf{C}_{k-1})$. Consequently, we have that $\mathbf{C}_k \in \mathbb{D}_{0,1}(\mathbf{B}_{k-1}) \cap \mathbb{D}_{1,0}(\mathbf{C}_{k-1})$. By Lemma 4 there exists a $\mathbf{B}_{k-2} \in \Sigma^{(n-t_\mathrm{r}+1)\times(n-t_\mathrm{r}+1)}$ such that $\mathbf{B}_{k-2} \in \mathbb{I}_{1,0}(\mathbf{B}_{k-1}) \cap \mathbb{I}_{0,1}(\mathbf{C}_{k-1})$. Following the same arguments as above, one can show that for all even values of $s \in \{1, \ldots, k\}$, the arrays $\mathbf{C}_s$, $\mathbf{B}_s$ and $\mathbf{C}_{s+1}$ satisfy $\mathbf{C}_{s+1} \in \mathbb{D}_{0,1}(\mathbf{C}_s) \cap \mathbb{D}_{0,1}(\mathbf{B}_s)$. Thus, by Lemma 3 there exists an array $\mathbf{B}_{s-1} \in \mathbb{I}_{0,1}(\mathbf{C}_s) \cap \mathbb{I}_{0,1}(\mathbf{B}_s)$. Similarly, for all odd values of $s \in \{1, \ldots, k\}$, the arrays $\mathbf{C}_s$, $\mathbf{B}_s$ and $\mathbf{C}_{s+1}$ satisfy $\mathbf{C}_{s+1} \in \mathbb{D}_{1,0}(\mathbf{C}_s) \cap \mathbb{D}_{0,1}(\mathbf{B}_s)$. Thus, by Lemma 4 there exists an array $\mathbf{B}_{s-1} \in \mathbb{I}_{1,0}(\mathbf{C}_s) \cap \mathbb{I}_{0,1}(\mathbf{B}_s)$.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

Subsequently, we will end up with an array $\mathbf{B}_0 \in \Sigma^{n\times n}$ such that $\mathbf{B}_k \in \mathbb{D}_{t_r,t_r}(\mathbf{X}_1) \cap \mathbb{D}_{t_r,t_r}(\mathbf{B}_0)$. Therefore, by Theorem 2 there exists an array $\mathbf{F}_k \in \Sigma^{(n+t_r)\times(n+t_r)}$ such that $\mathbf{F}_k \in \mathbb{I}_{t_r,t_r}(\mathbf{X}_1) \cap \mathbb{I}_{t_r,t_r}(\mathbf{B}_0)$.

Let $\mathbf{F}_0 \triangleq \mathbf{B}_0$ and $\mathbf{F}_{-1} \triangleq \mathbf{C}_1$, we denote again the series of arrays $\{\mathbf{F}_s\}_{s=1}^{k}$ that are the intermediate arrays obtained by a row or a column insertion starting from $\mathbf{F}_0$ until reaching $\mathbf{F}_k$. We again consider alternating insertions of rows and columns, i.e.,

$$\mathbf{F}_s \in \begin{cases} \mathbb{I}_{1,0}(\mathbf{F}_{s-1}), & \text{if } s \text{ is odd,} \\ \mathbb{I}_{0,1}(\mathbf{F}_{s-1}), & \text{otherwise.} \end{cases}$$

Define the array $\mathbf{G}_0 \triangleq \mathbf{X}_2$, we will now show the existence of the series of arrays $\{\mathbf{G}_s\}_{s=1}^{k+1}$ such that

$$\mathbf{G}_s \in \begin{cases} \mathbb{I}_{0,1}(\mathbf{G}_{s-1}) \cap \mathbb{I}_{0,1}(\mathbf{F}_{s-1}), & \text{if } s \text{ is odd,} \\ \mathbb{I}_{1,0}(\mathbf{G}_{s-1}) \cap \mathbb{I}_{0,1}(\mathbf{F}_{s-1}), & \text{otherwise.} \end{cases}$$

Following the same arguments used to construct the series $\{\mathbf{B}_s\}_{s=0}^{k-1}$, one can show that for all even values of $s \in \{0,\ldots,k\}$, the arrays $\mathbf{F}_s$, $\mathbf{G}_s$ and $\mathbf{F}_{s-1}$ satisfy $\mathbf{F}_{s-1} \in \mathbb{D}_{0,1}(\mathbf{F}_s) \cap \mathbb{D}_{0,1}(\mathbf{G}_s)$. Thus, by Lemma 3 there exists an array $\mathbf{G}_{s+1} \in \mathbb{I}_{0,1}(\mathbf{F}_s) \cap \mathbb{I}_{0,1}(\mathbf{G}_s)$. Similarly, for all odd values of $s \in \{0,\ldots,k\}$, the arrays $\mathbf{F}_s$, $\mathbf{G}_s$ and $\mathbf{F}_{s-1}$ satisfy $\mathbf{F}_{s-1} \in \mathbb{D}_{1,0}(\mathbf{F}_s) \cap \mathbb{D}_{0,1}(\mathbf{G}_s)$. Thus, by Lemma 4 there exists an array $\mathbf{G}_{s+1} \in \mathbb{I}_{1,0}(\mathbf{G}_s) \cap \mathbb{I}_{0,1}(\mathbf{F}_s)$. As a consequence, we have shown that if there exists an array $\mathbf{C}_{k+1} \in \mathbb{D}_{t_r,t_r+1}(\mathbf{X}_1) \cap \mathbb{D}_{t_r,t_r+1}(\mathbf{X}_2)$, then there exists an array $\mathbf{G}_{k+1} \in \mathbb{I}_{t_r,t_r+1}(\mathbf{X}_1) \cap \mathbb{I}_{t_r,t_r+1}(\mathbf{X}_2)$.

### B. Induction Hypothesis

For any integer $c \geqslant 1$ it holds that for any $\mathbf{X}_1, \mathbf{X}_2 \in \Sigma^{n\times n}$, $\mathbb{D}_{t_r,t_r+c}(\mathbf{X}_1) \cap \mathbb{D}_{t_r,t_r+c}(\mathbf{X}_2) \neq \emptyset$ if and only if $\mathbb{I}_{t_r,t_r+1}(\mathbf{X}_1) \cap \mathbb{I}_{t_r,t_r+c}(\mathbf{X}_2) \neq \emptyset$.

### C. Induction Step

Assume that the induction hypothesis holds for all values $0 \leqslant t_c - t_r \leqslant c$. We prove that the hypothesis holds for $t_c - t_r = c+1$.

Assume that there exists an array $\mathbf{E} \in \Sigma^{(n-t_r)\times(n-t_r-c-1)}$ such that $\mathbf{E} \in \mathbb{D}_{t_r,t_r+c+1}(\mathbf{X}_1) \cap \mathbb{D}_{t_r,t_r+c+1}(\mathbf{X}_2)$. Let $k = 2t_r$, define $\mathbf{C}_0 \triangleq \mathbf{X}_2$ and $\mathbf{C}_{k+c+1} \triangleq \mathbf{E}$. We denote by $\{\mathbf{C}_s\}_{s=0}^{k+c+1}$ the series of arrays resulting from a deletion of a row or a column starting from $\mathbf{X}_2$ until obtaining $\mathbf{E}$ as follows

$$\mathbf{C}_s \in \begin{cases} \mathbb{D}_{0,1}(\mathbf{C}_{s-1}), & \text{if } s \text{ is odd or } s > k, \\ \mathbb{D}_{1,0}(\mathbf{C}_{s-1}), & \text{otherwise.} \end{cases}$$

We denote by $\mathbf{B}_{k+c} \in \Sigma^{(n-t_r)\times(n-t_r-c)}$ the array resulting from deleting $t_r$ columns and $t_r + c$ rows from $\mathbf{X}_1$ such that $\mathbf{E} \in \mathbb{D}_{0,1}(\mathbf{B}_{k+c})$. We now want to show that there exists a series of arrays $\{\mathbf{B}_s\}_{s=0}^{k+c-1}$ such that

$$\mathbf{B}_s \in \begin{cases} \mathbb{I}_{0,1}(\mathbf{B}_{s+1}) \cap \mathbb{I}_{0,1}(\mathbf{C}_{s+1}), & \text{if } s \text{ is odd or } s \geqslant k, \\ \mathbb{I}_{1,0}(\mathbf{B}_{s+1}) \cap \mathbb{I}_{0,1}(\mathbf{C}_{s+1}), & \text{otherwise.} \end{cases}$$

Following the same arguments as in the base case, one can show that for all even values of $s \in \{0,\ldots,k\}$ and for all values of $k \leqslant s \leqslant k+c$, the arrays $\mathbf{C}_s$, $\mathbf{B}_s$ and $\mathbf{C}_{s+1}$ satisfy $\mathbf{C}_{s+1} \in \mathbb{D}_{0,1}(\mathbf{C}_s) \cap \mathbb{D}_{0,1}(\mathbf{B}_s)$. Thus, by Lemma 3 there exists an array $\mathbf{B}_{s-1} \in \mathbb{I}_{0,1}(\mathbf{C}_s) \cap \mathbb{I}_{0,1}(\mathbf{B}_s)$. Similarly, for all odd values of $s \in \{0,\ldots,k\}$, the arrays $\mathbf{C}_s$, $\mathbf{B}_s$ and $\mathbf{C}_{s+1}$ satisfy $\mathbf{C}_{s+1} \in \mathbb{D}_{1,0}(\mathbf{C}_s) \cap \mathbb{D}_{0,1}(\mathbf{B}_s)$. Thus, by Lemma 4 there exists an array $\mathbf{B}_{s-1} \in \mathbb{I}_{1,0}(\mathbf{C}_s) \cap \mathbb{I}_{0,1}(\mathbf{B}_s)$. Subsequently, we will end up with an array $\mathbf{B}_0 \in \Sigma^{n\times n}$ such that $\mathbf{B}_{k+c} \in \mathbb{D}_{t_r,t_r+c}(\mathbf{X}_1) \cap \mathbb{D}_{t_r,t_r+c}(\mathbf{B}_0)$. Therefore, by using the induction hypothesis, we can show the existence of the array $\mathbf{F}_{k+c} \in \Sigma^{(n+t_r)\times(n+t_r+c)}$ such that $\mathbf{F}_{k+c} \in \mathbb{I}_{t_r,t_r+c}(\mathbf{X}_1) \cap \mathbb{I}_{t_r,t_r+c}(\mathbf{B}_0)$.

Let $\mathbf{F}_0 \triangleq \mathbf{B}_0$ and $\mathbf{F}_{-1} \triangleq \mathbf{C}_1$, we denote again the series of arrays $\{\mathbf{F}_s\}_{s=1}^{k+c}$ as the intermediate arrays obtained by a row or a column insertion starting from $\mathbf{F}_0$ until reaching $\mathbf{F}_k$ as follows

$$\mathbf{F}_s \in \begin{cases} \mathbb{I}_{0,1}(\mathbf{F}_{s-1}) & \text{if } s \text{ is even or } s > k, \\ \mathbb{I}_{1,0}(\mathbf{F}_{s-1}) & \text{otherwise.} \end{cases}$$

Define the array $\mathbf{G}_0 \triangleq \mathbf{X}_2$, we will now show the existence of the series of arrays $\{\mathbf{G}_s\}_{s=1}^{k+c+1}$ such that

$$\mathbf{G}_s \in \begin{cases} \mathbb{I}_{0,1}(\mathbf{G}_{s-1}) \cap \mathbb{I}_{0,1}(\mathbf{F}_{s-1}) & \text{if } s \text{ is odd or } s > k, \\ \mathbb{I}_{1,0}(\mathbf{G}_{s-1}) \cap \mathbb{I}_{0,1}(\mathbf{F}_{s-1}) & \text{otherwise.} \end{cases}$$

Following the same arguments as in the base case, one can show that for all even values of $s \in \{0,\ldots,k\}$ and $k \leqslant s \leqslant k+c$, the arrays $\mathbf{F}_s$, $\mathbf{G}_s$ and $\mathbf{F}_{s-1}$ satisfy $\mathbf{F}_{s-1} \in \mathbb{D}_{0,1}(\mathbf{F}_s) \cap \mathbb{D}_{0,1}(\mathbf{G}_s)$. Thus, by Lemma 3 there exists an array $\mathbf{G}_{s+1} \in \mathbb{I}_{0,1}(\mathbf{F}_s) \cap \mathbb{I}_{0,1}(\mathbf{G}_s)$. Similarly, for all odd values of $s \in \{0,\ldots,k\}$, the arrays $\mathbf{F}_s$, $\mathbf{G}_s$ and $\mathbf{F}_{s-1}$ satisfy $\mathbf{F}_{s-1} \in \mathbb{D}_{1,0}(\mathbf{F}_s) \cap \mathbb{D}_{0,1}(\mathbf{G}_s)$. Thus, by Lemma 4 there exists an array $\mathbf{G}_{s+1} \in \mathbb{I}_{1,0}(\mathbf{G}_s) \cap \mathbb{I}_{0,1}(\mathbf{F}_s)$. As a consequence, we have shown that if there exists an array $\mathbf{C}_{k+c+1} \in \mathbb{D}_{t_r,t_r+c+1}(\mathbf{X}_1) \cap \mathbb{D}_{t_r,t_r+c+1}(\mathbf{X}_2)$, then there exists an array $\mathbf{G}_{k+c+1} \in \mathbb{I}_{t_r,t_r+c+1}(\mathbf{X}_1) \cap \mathbb{I}_{t_r,t_r+c+1}(\mathbf{X}_2)$. This concludes the induction. ■

## IV. Upper Bound on the Cardinality

This section presents an asymptotic upper bound on the cardinality of any $t$-criss-cross indel code. It implies an asymptotic lower bound on the redundancy of any binary $t$-criss-cross indel code, denoted by $R_B(n,t)$.

*Lemma 5:* Any upper bound on the cardinality of a $q$-ary $t$-deletion-correcting code $\mathcal{C}_{q,n,t}$ with $q = 2^n$ is also an upper bound on the cardinality of a binary $t$-criss-cross indel code.

*Proof:* Note that a $2^n$-ary $t$-deletion-correcting code $\mathcal{C}_{2^n,n,t}$ can be seen also as a binary $t$ column deletion-correcting code by interpreting the symbols as binary columns. Since a $t$-criss-cross indel code $\mathcal{C}$ can correct any combination of $t_r$ row and $t_c$ column deletions such that $t_r + t_c = t$, in particular it can also correct any $t$ column deletions. Therefore, any upper bound on the size of $\mathcal{C}_{2^n,n,t}$ is also a valid upper bound on the size of $\mathcal{C}$. ■

*Corollary 6:* For any binary $t$-criss-cross indel code $\mathcal{C}$ it holds that

$$|\mathcal{C}| \lesssim \frac{t! 2^{n^2}}{(2^n - 1)^t n^t}.$$

Consequently, we have $R_B(n,t) \gtrsim tn + t\log(n) - \log(t!)$.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

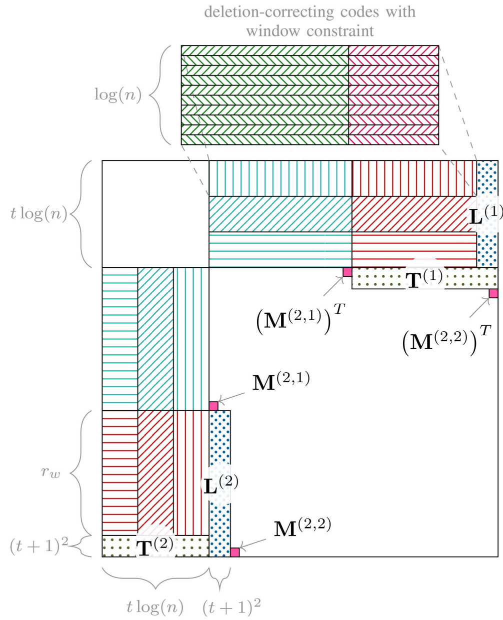


Fig. 2. Illustration of an array contained in the locator set $\mathcal{L}_t(n)$ for $t=3$. In the first $t\log(n)$ rows there are $t$ blocks each consisting of a systematic part (*cyan*) and a redundancy part (red). Each row is encoded using a systematic $t$-deletion-correcting code (zoomed in part). In addition, in the systematic part of each block a window constrained is imposed. Those blocks are used to locate column deletions. This structure is protected with the arrays $\mathbf{L}^{(1)}$ (*blue*) against row deletions and $\mathbf{T}^{(1)}$ (*brown*) against column deletions. Lastly, to locate the borders of $\mathbf{T}^{(1)}$ we use the marker arrays $\mathbf{M}^{(2,1)}$ and $\mathbf{M}^{(2,2)}$ (*pink*). A symmetric structure locates row deletions.

*Proof:* From [34], we have for a $q$-ary $t$-deletion correcting code that $|\mathcal{C}_{q,n,t}| \lesssim \frac{t!q^n}{(q-1)^t n^t}$ when $q$ is fixed. By following the same arguments as in the proof of [34, Theorem 4.3] we can show that this bound holds also true when $q=2^n$. Therefore, for any binary $t$-criss-cross indel code $\mathcal{C}$ it holds by Lemma 5 that

$$|\mathcal{C}| \leqslant |\mathcal{C}_{2^n,n,t}| \lesssim \frac{t!2^{n^2}}{(2^n-1)^t n^t}.$$

Therefore, we have

$$R_{\mathrm{B}}(n,t) \gtrsim n^2 - \log(|\mathcal{C}|) \approx tn + t\log(n) \ - \log(t!).$$

■

## V. Code Construction

In this section we present an existential construction of $t$-criss-cross indel codes. We start with an intuitive road map to our code construction and then formally define each ingredient.

### A. Road Map

Our construction uses structured arrays so that the indices of the inserted/deleted rows and columns can be exactly recovered. Then, the set of structured arrays is intersected with arrays of a Gabidulin code (that can correct row/column erasures) to recover the arrays of the code. The structure is depicted in Figure 2.

We structure the $n\times n$ codewords $\mathbf{C}$ as follows. We protect the columns with indices between $t\log n+1$ and $n-(t+1)^2$ using $t\log n$ codes where each one is a binary systematic $t$-indel-correcting code. We divide those codes into $t$ blocks each of size $\log n$. We impose what we call a *window constraint* on the columns of the systematic part of every block. This constraint ensures that every $t+1$ consecutive columns are different. Therefore, the indices of the deleted columns within the systematic part can be located by using all $\log n$ indel-correcting codes of any block (Claim 9). In case of insertions, the indices of the inserted columns within the systematic part can be located up to an interval of length at most $2t$ containing at most $t$ columns of the original array. This ambiguity arises when the inserted rows/columns are equal to collections of rows/columns of the original array within an interval. We call this phenomenon *block confusions* (cf. Section V-B and Claim 10).

In the redundancy part, runs may exist. Thus, the recovery of the index of the deleted columns is only guaranteed within the corresponding run. To recover the exact location of the deleted columns here, we protect the redundancy part of the codes by appending (from below) what we call a *locator array* that can detect the exact positions of column deletions within this part (Claim 7). We call this array $\mathbf{T}^{(1)}$. In case of insertions, the locator array is used to recover the index of the inserted columns up to block confusions of length at most $2t$ (Claim 8).

Note that for the window constraint to work, we need to have all $\log n$ indel-correcting codes of the considered block. Therefore, we use the subarray $\mathbf{C}_{[1:t\log n],[n-(t+1)^2:n]}$ as a locator array $\mathbf{L}^{(1)}$ that can detect the exact position of a deleted row within the first $t\log n$ rows (Claim 7). As a result, if all $t$ deletions are row deletions within the first $t\log n$ rows, then the locator array is enough to recover all the indices (Lemma 13). Otherwise, we have at least one block of the $t$ blocks that is not affected by a row deletion. This block is used to recover the deleted columns with indices in the range $t\log n+1$ to $n-(t+1)^2$ (Lemma 14). The same arguments hold for the insertion case (Claim 8, Lemma 16 and Lemma 17). The only difference is the possibility of the inserted row/column causing ambiguities for the exact indices of the insertions.

One more step is needed. We must be able to locate the position of the locator arrays within the resulting $(n-t_{\mathrm{r}})\times(n-t_{\mathrm{c}})$ array $\widetilde{\mathbf{C}}$. Therefore, we put four *marker arrays* after the locators that are detectable even after $t$ insertions or deletions. We call those arrays $\mathbf{M}^{(1,1)}$ and $\mathbf{M}^{(1,2)}$.

The same structure (transposed) is used to index the rows. In addition, the columns with indices between 1 and $t\log n$ are protected by the locator array used for protecting the indel-correcting codes indexing the rows. Note the claims and lemmas mentioned before also include the statements to recover the row indices.

The whole code is intersected with a Gabidulin code [35] that can correct row/column erasures. Once the positions of the deleted rows and columns are known to the decoder, those positions are marked as erasures and corrected using the Gabidulin code. In case of insertions, all inserted rows and columns with exactly recovered indices are removed. In the case where inserted rows/columns create block confusions we can simply delete all involved rows/columns and insert

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

erasures. The window constraint guarantees that we do not delete more than $t$ rows/columns of the original array by this procedure, hence assuring to not exceed the erasure correction capability of the Gabidulin code.

In the next subsections, we formally define the five main ingredients of our code: *(i)* the locator arrays; *(ii)* the binary systematic $t$-deletion-correcting codes with window constraints; *(iii)* the marker arrays; *(iv)* the *locator set* which is the combination of all the previously mentioned parts; and *(v)* a Gabidulin code [35] that is used to correct row/column erasures.

### B. Locator Arrays

For a positive integer $a$, we denote by $\mathbf{I}_a$ the identity array of dimension $a \times a$ and by $\mathbf{1}_a$ and $\mathbf{0}_a$ the all-one row vector and all-zero row vector of length $a$, respectively. We use $\otimes$ to indicate the Kronecker product. We thus have the following definition from [33].

*Definition 1 (Locator arrays):* We set $\mathbf{L}' \in \Sigma^{(t+1)\times(t+1)^2}$ as $\mathbf{L}' \triangleq \mathbf{I}_{t+1} \otimes \mathbf{1}_{t+1}$. More precisely, $\mathbf{L}'$ has the following structure

$$\mathbf{L}' = \begin{pmatrix} \mathbf{1}_{t+1} & \mathbf{0}_{t+1} & \dots & \mathbf{0}_{t+1} \\ \mathbf{0}_{t+1} & \mathbf{1}_{t+1} & \dots & \mathbf{0}_{t+1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{t+1} & \mathbf{0}_{t+1} & \dots & \mathbf{1}_{t+1} \end{pmatrix}.$$

Let $s$ be a multiple of $(t+1)$ such that $s \geqslant \lceil \frac{t}{2} \rceil (t+1)$. We define the locator array $\mathbf{L}_s \in \Sigma^{s\times(t+1)^2}$ as

$$\mathbf{L}_s \triangleq \mathbf{1}^T_{\frac{s}{t+1}} \otimes \mathbf{L}'.$$

Moreover, we define the locator array $\mathbf{T}_s \in \Sigma^{(t+1)^2\times s}$ to be the transpose of $\mathbf{L}_s$, i.e.,

$$\mathbf{T}_s \triangleq \mathbf{L}_s^T = \mathbf{1}_{\frac{s}{t+1}} \otimes \mathbf{L}'^T.$$

Throughout the paper we drop $s$ in the notation $\mathbf{L}_s$ and $\mathbf{T}_s$ when the value of $s$ is clear from the context.

*Claim 7 (Deletion Detection in $\mathbf{L}_s$ and $\mathbf{T}_s$):* Let $\mathbf{L}_s$ be an array affected by $t_\mathrm{r}$ row and $t_\mathrm{c}$ column deletions such that $t_\mathrm{r} + t_\mathrm{c} = t$. Divide $\mathbf{L}_s$ into $(t+1)$ subarrays each consisting of $(t+1)$ consecutive columns of $\mathbf{L}_s$. By examining $\widetilde{\mathbf{L}}_s$, we can locate the exact positions of the deleted rows. We can also determine the number of column deletions that happened in each subarray of $\mathbf{L}_s$.

Let $\mathbf{T}_s$ be an array affected by $t_\mathrm{r}$ row and $t_\mathrm{c}$ column deletions such that $t_\mathrm{r} + t_\mathrm{c} = t$. The same statement above for $\mathbf{L}_s$ holds for $\mathbf{T}_s$ by switching rows for columns.

*Proof:* We prove the first part of the claim, while the second part follows similarly since $\mathbf{T}_s = \mathbf{L}_s^T$ and row deletions in $\mathbf{T}_s$ can be seen as column deletions in $\mathbf{L}_s$ and vice versa.

By construction of $\mathbf{L}_s$, for any $i \in [s-t]$ and $j \in [t]$ it holds that $\mathbf{L}_{i,[(t+1)^2]} \neq \mathbf{L}_{i+j,[(t+1)^2]}$. This property holds true even in the presence of at most $t$ column deletions in $\mathbf{L}_s$. Thus, due to the fixed structure of $\mathbf{L}_s$ one can uniquely determine the exact indices of the deleted rows.

Moreover, we divide $\mathbf{L}_s$ in subarrays consisting of $(t+1)$ columns. For any $a \in [t+1]$ and $b \in [t+1]$, we have $\mathbf{L}_{[s],(a-1)(t+1)+1} = \mathbf{L}_{[s],(a-1)(t+1)+b}$. In words, we have $(t+1)$ identical columns in a subarray. This property holds true even if there were at most $t$ row deletions in $\mathbf{L}_s$. Furthermore, for $s \geqslant \lceil \frac{t}{2} \rceil (t+1)$ and $a, b, c \in [t+1]$, it always holds that $\mathbf{L}_{[s],(a-1)(t+1)+1} \neq \mathbf{L}_{[s],(c-1)(t+1)+b}$, unless $a = c$. Therefore, we can determine the deleted columns within any subarray by counting the number of missing columns. ■

We briefly elaborate on the dimension constraint of $\mathbf{L}_s$, i.e., $s \geqslant \lceil \frac{t}{2} \rceil (t+1)$. If $\mathbf{L}_s$ consists of less than $\lceil \frac{t}{2} \rceil$ copies of $\mathbf{L}'$, then a deletion of two consecutive rows in each block will lead to an impossibility in locating a column deletion within two adjacent subarrays. Recall that $\mathbf{1}_a$ is the all-one vector of length $a$ and let $t = 3$ and $s = 4$, we assume the following deletion pattern:

$$\underbrace{\begin{pmatrix} \mathbf{1}_4 & \mathbf{0}_4 & \mathbf{0}_4 & \mathbf{0}_4 \\ \mathbf{0}_4 & \mathbf{1}_4 & \mathbf{0}_4 & \mathbf{0}_4 \\ \mathbf{0}_4 & \mathbf{0}_4 & \mathbf{1}_4 & \mathbf{0}_4 \\ \mathbf{0}_4 & \mathbf{0}_4 & \mathbf{0}_4 & \mathbf{1}_4 \end{pmatrix}}_{\mathbf{L}'} \xrightarrow[\text{2nd column del.}]{\text{1st \& 2nd row del.}} \underbrace{\begin{pmatrix} \mathbf{0}_7 & \mathbf{1}_4 & \mathbf{0}_4 \\ \mathbf{0}_7 & \mathbf{0}_4 & \mathbf{1}_4 \end{pmatrix}}_{\tilde{\mathbf{L}}'}$$

Even though we can locate the row deletions, we cannot locate the column deletion within a subarray of $(t+1)$ columns. Thus, it is necessary to have another copy of $\mathbf{L}'$ within $\mathbf{L}_s$ to guarantee a successful detection of column deletions. In the general case, this extends to needing at least $\lceil \frac{t}{2} \rceil$ copies of $\mathbf{L}'$ in $\mathbf{L}_s$. Note that this example can be directly applied for $\mathbf{T}_s$ by switching in the argument rows and columns.

For the following statements and proofs in case of insertions we have to define the terminology of block confusions.

*Block confusions:* Consider an array $\mathbf{Z} \in \Sigma^{n\times m}$. Let $\widetilde{\mathbf{Z}}$ be the resulting array after $t$-criss-cross insertions in $\mathbf{Z}$. Given a subset $\mathcal{B} \subseteq [n]$, an integer $a \in [n]$, and a non-negative integer $c \leqslant n$, we define a row block confusion in $\widetilde{\mathbf{Z}}$ if $\widetilde{\mathbf{Z}}_{a+c',[m]} = \widetilde{\mathbf{Z}}_{a+b+c',[m]}$ for all $b \in \mathcal{B}$ and all non-negative $c'$ such that $c' \leqslant c$. In other words, we declare a block confusion after insertions, when a collection of $c'$ consecutive rows in $\widetilde{\mathbf{Z}}$ are the same as other collections of consecutive rows within an interval determined by $a$ and $\mathcal{B}$. This actually leads to the fact that an insertion locating algorithm cannot determine the exact location (up to the block confusion) of the inserted rows even when knowing $\mathbf{Z}$. For convenience, we provide the following illustration of a row block confusion.

$$\underbrace{\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}}_{\mathbf{Z}} \xrightarrow[\text{1st \& 2nd row}]{\text{insertion in}} \underbrace{\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}}_{\tilde{\mathbf{Z}}}$$

Given both $\mathbf{Z}$ and $\widetilde{\mathbf{Z}}$, an insertion locating algorithm cannot distinguish whether the insertion happened in the first and second row or third and fourth row. We call this a row block confusion with parameters $a = 1$, $\mathcal{B} = \{2\}$, and $c = 1$.

In the case that $\mathbf{Z}$ satisfies predetermined properties we can narrow down the parameter range of the possible block confusions. Let $\mathbf{Z} \in \Sigma^{n\times m}$ be an array satisfying $\mathbf{Z}_{i,[m]} \neq \mathbf{Z}_{i+j,[m]}$ for all $i \in [n-t]$ and $j \in [t]$. Then, it follows that $\mathcal{B} \subseteq [t]$, $a \in [n]$, and $0 \leqslant c \leqslant t$. Therefore, the

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

window of a row block confusion, defined as $[a, \max(\mathcal{B})+c] \triangleq \{a, \ldots, \max(\mathcal{B})+c\}$, consists of at most $2t$ rows. In addition, within this window there exists at least one and at most $t$ rows of the original array $\mathbf{Z}$. Moreover, the bounds on the maximum number of original columns in the confusion and the *length* of the confusion, defined as the total number of rows in the block confusion, scale proportionally with the number of insertions responsible for the confusion. Note that the aforementioned property on the array $\mathbf{Z}$ is satisfied by $\mathbf{L}_s$.

We define a column block confusion similarly. The array $\mathbf{T}_s$ satisfies the desired properties on the columns to limit the parameter range of column block confusions. In the sequel, we will drop the row and column term when referring to block confusions when it is clear from the context.

*Claim 8 (Insertion Detection in $\mathbf{L}_s$ and $\mathbf{T}_s$):* Let $\mathbf{L}_s$ be affected by $t_{\mathrm{r}}$ row and $t_{\mathrm{c}}$ column insertions such that $t_{\mathrm{r}} + t_{\mathrm{c}} = t$. Divide $\mathbf{L}_s$ into $(t+1)$ subarrays each consisting of $(t+1)$ consecutive columns of $\mathbf{L}_s$. By examining $\widetilde{\mathbf{L}}_s$, we can locate the positions of the inserted rows up to a row block confusion of length at most $2t$ containing at most $t$ columns of the original array $\mathbf{L}_s$. We can also determine the number of column insertions (and possibly their positions up to a column block confusion within the subarray or within the adjacent subarray) that happened in each subarray of $\mathbf{L}_s$.

Let $\mathbf{T}_s$ be affected by $t_{\mathrm{r}}$ row and $t_{\mathrm{c}}$ column insertions such that $t_{\mathrm{r}} + t_{\mathrm{c}} = t$. The same statement above for $\mathbf{L}_s$ holds for $\mathbf{T}_s$ by switching rows for columns.

*Proof:* The proof follows the same steps as in Claim 7. In terms of row insertions, the difference is that if the inserted rows in $\mathbf{L}_s$ create a block confusion, then we cannot distinguish between the original rows and the inserted rows in a window of length at most $t+1$ rows. This follows from the construction of $\mathbf{L}_s$. In terms of column insertions, if the inserted column is different from the column run in which it is inserted and from both adjacent column runs, then it can be exactly located. Otherwise, we can only count the number of insertions that happened in each block up to a confusion with an adjacent block. This is due to possible column block confusions at the column runs located in adjacent subarrays. ■

### C. Indel-Correcting Codes With Window Constraints

*Indel-correcting codes:* We use the construction of [18] for our binary systematic $t$-indel-correcting code. We briefly recall the results of [18]. Given a sequence $\boldsymbol{k} \in \Sigma^{\kappa}$, one can compute a redundancy vector $\boldsymbol{r_k} \in \Sigma^{r_k}$ with $r_k \leqslant 4t\log(\kappa) + o(\log(\kappa))$. The resulting sequence $(\boldsymbol{k}|\boldsymbol{r_k})$ can be uniquely recovered after $t$ indel errors. Note that $\boldsymbol{r_k}$ is a function of the information $\boldsymbol{k}$ and $\rho_\kappa$ is a function of the information length $\kappa$ and the number of indel errors $t$.

*Window constraint:* We define the window constraint as the set $\mathcal{W}_t(\ell, w) \subseteq \Sigma^{\ell\times w}$, where for any $\mathbf{W} \in \mathcal{W}_t(\ell, w)$, $i \in [w-t]$ and $j \in [t]$, it holds that $\mathbf{W}_{[\ell],i} \neq \mathbf{W}_{[\ell],i+j}$.

For an array $\mathbf{W} \in \mathcal{W}_t(\ell, w)$, let $\mathbf{R_W} \in \Sigma^{\ell\times r_w}$ be the array formed such that for any $i \in [\ell]$ the $i$ row of $\mathbf{R_W}$ is the redundancy vector corresponding to the $i$ row of $\mathbf{W}$; computed using the systematic construction in [18]. We refer to the array $\mathbf{R_W} \in \Sigma^{\ell\times r_w}$ as the redundancy array. Let $m \triangleq w + r_w$, we define $\mathcal{D}_t^{(1)}(\ell, m)$ as the set of all arrays resulting from the concatenation of $\mathbf{W}$ and $\mathbf{R_W}$, i.e.,

$$\mathcal{D}_t^{(1)}(\ell, m) \triangleq \left\{ \mathbf{D} \in \Sigma^{\ell\times m} : \begin{array}{c} \mathbf{D} = (\mathbf{W} \mid \mathbf{R_W}), \\ \text{s.t.} \mathbf{W} \in \mathcal{W}_t(\ell, w) \end{array} \right\}.$$

In words, $\mathcal{D}_t^{(1)}(\ell, m)$ is the set of binary systematic $t$-indel-correcting codes in which the systematic part satisfies the imposed window constraint. This set will be used to index the columns of our arrays in the constructed code. We define $\mathcal{D}_t^{(2)}(\ell, m) \triangleq \left\{ \mathbf{D}^T : \mathbf{D} \in \mathcal{D}_t^{(1)}(\ell, m) \right\}$. This set is going to be used for indexing the rows.

*Claim 9:* Let $\mathbf{D} = (\mathbf{W} \mid \mathbf{R_W}) \in \mathcal{D}_t^{(1)}(\ell, m)$ be an array affected by $t$ column deletions and no row deletions, we can locate the exact positions of the deleted columns in the subarray $\mathbf{W}$.

The same holds for any array in $\mathcal{D}_t^{(2)}(\ell, m)$ by switching in the argument rows and columns.

*Proof:* Assume $\widetilde{\mathbf{D}} = (\widetilde{\mathbf{W}} \mid \widetilde{\mathbf{R}}_{\mathbf{W}})$ is the array obtained after the deletions. For each row in $\widetilde{\mathbf{W}}$ we can use the corresponding redundancy in $\widetilde{\mathbf{R}}_{\mathbf{W}}$ to correct the deletions that happened in this row [18]. We start by looking at the position of the first recovered bit in each row. In each row, this position may be unique or may be in an interval of possible positions (run). The exact location of the column is then determined by the unique position in which all runs (of all rows) intersect. The intersection is guaranteed to be unique by the imposed window constraint; since for any $i \in [w-t]$, and $j \in [t]$, it holds that $\mathbf{W}_{[\ell],i} \neq \mathbf{W}_{[\ell],i+j}$. This process is repeated for all recovered bits until all $t$ positions are determined.

A similar argument follows for the second statement of the claim. ■

*Claim 10:* Given an array $\mathbf{D} = (\mathbf{W} \mid \mathbf{R_W}) \in \mathcal{D}_t^{(1)}(\ell, m)$ affected by $t$ column insertions and no row insertions, we can locate the positions of the inserted columns in the subarray $\mathbf{W}$ up to column block confusion of length at most $2t+1$ containing at most $t$ columns of the original array $\mathbf{D}$..

The same holds for any array in $\mathcal{D}_t^{(2)}(\ell, m)$ by switching in the argument rows and columns.

*Proof:* The proof follows the same technique of the proof of Claim 9. The only exception arises if the inserted columns create a column block confusion. However, the window constraint guarantees that in the worst case the block confusion occurs in a window of length at most $2t$. Moreover, within this block confusion there exists at most $t$ columns of the original array. ■

### D. Marker Arrays

We define the following arrays of dimension $(t+1) \times (t+1)$ which will operate as *markers* to locate the position of the locator arrays in the resulting $\widetilde{\mathbf{C}}$. Recall that we use four locator arrays in our construction, namely $\mathbf{L}^{(1)}$, $\mathbf{L}^{(2)}$, $\mathbf{T}^{(1)}$, and $\mathbf{T}^{(2)}$, cf. Figure 2. We only need marker arrays for $\mathbf{T}^{(1)}$ and $\mathbf{L}^{(2)}$. The position of $\mathbf{L}^{(1)}$ and $\mathbf{T}^{(2)}$ can be then determined. The first marker array $\mathbf{M}^{(2,1)}$, put on top of $\mathbf{L}^{(2)}$, consists of the first $t+1$ columns of $\mathbf{L}'$. The second marker array $\mathbf{M}^{(2,2)}$,

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

put on the right of $\mathbf{L}^{(2)}$, consists of the complement of the last $t+1$ columns of $\mathbf{L}'$. The marker arrays $\mathbf{M}^{(1,1)}$ and $\mathbf{M}^{(1,2)}$ are the transpose of $\mathbf{M}^{(2,1)}$ and $\mathbf{M}^{(2,2)}$, respectively.

### *E. Locator Set*

We formally define the sets of arrays in $\Sigma^{n\times n}$ that form our code. Let $\mathbf{X} \in \Sigma^{n\times n}$, we start with the set of arrays that are used to index the columns. This set is denoted by $\mathcal{H}_t(\ell,n)$. The arrays in this set have the first $t\ell$ columns divided into $t$ blocks. The columns whose indices are between $t\ell+1$ and $n-(t+1)^2$ of each row consist of a systematic $t$-deletion-correcting code in which the systematic part satisfies the window constraint. We can write

$$\mathcal{H}_t(\ell,n) \triangleq \left\{\mathbf{X}\colon \begin{array}{l}\mathbf{X}_{[(a-1)\ell+1:a\ell],[t\ell+1:n-(t+1)^2]}\\ \quad\in \mathcal{D}_t^{(1)}(\ell,n-t\ell-(t+1)^2)\ \forall a\in[t]\end{array}\right\}.$$

The set of arrays $\mathcal{V}_t(\ell,n)$ that are used to index the rows is defined similarly to $\mathcal{H}_t(\ell,n)$ by replacing columns with rows

$$\mathcal{V}_t(\ell,n) \triangleq \left\{\mathbf{X}\colon \begin{array}{l}\mathbf{X}_{[t\ell+1:n-(t+1)^2],[(b-1)\ell+1:b\ell]}\\ \quad\in \mathcal{D}_t^{(2)}(\ell,n-t\ell-(t+1)^2)\ \forall b\in[t]\end{array}\right\}.$$

For a value of $r_w$ that divides[2] $t+1$, the set of arrays $\mathcal{E}_t(\ell,n)$ that contain the locator arrays in the positions shown in Figure 2 is defined as follows.

$$\mathcal{E}_t(\ell,n) \triangleq \left\{\mathbf{X}\colon \begin{array}{l}\mathbf{X}_{[1:t\ell],[n-(t+1)^2+1:n]}=\mathbf{L}_{t\ell},\\ \mathbf{X}_{[t\ell+1:t\ell+(t+1)^2],[n-r_w-(t+1)^2+1:n]}=\mathbf{T}_{r_w+(t+1)^2},\\ \mathbf{X}_{[n-(t+1)^2+1:n],[1:t\ell]}=\mathbf{T}_{t\ell},\\ \mathbf{X}_{[n-r_w-(t+1)^2+1:n],[t\ell+1:t\ell+(t+1)^2]}=\mathbf{L}_{r_w+(t+1)^2},\end{array}\right\}.$$

The set of arrays that contain the marker arrays in the positions shown in Figure 2, is defined as follows.

$$\mathcal{M}_t(\ell,n) \triangleq \left\{\mathbf{X}\colon \begin{array}{l}\mathbf{X}_{[t\ell+1:t\ell+(t+1)],[n-r_w-(t+1)^2-(t+1)+1:n-r_w-(t+1)^2]}\\ \quad=\mathbf{M}^{(1,1)},\\ \mathbf{X}_{[t\ell+(t+1)^2+1:t\ell+(t+1)^2+(t+1)],[n-(t+1)+1:n]}\\ \quad=\mathbf{M}^{(1,2)},\\ \mathbf{X}_{[n-r_w-(t+1)^2-(t+1)+1:n-r_w-(t+1)^2],[t\ell+1:t\ell+(t+1)]}\\ \quad=\mathbf{M}^{(2,1)},\\ \mathbf{X}_{[n-(t+1)+1:n],[t\ell+(t+1)^2+1:t\ell+(t+1)^+(t+1)]}\\ \quad=\mathbf{M}^{(2,2)}\end{array}\right\}.$$

We can conclude this subsection by defining the *locator set* that is the set of all arrays that have the structure required by our code to recover the indices of the inserted or deleted columns and rows. The locator set is the intersection of all the previously defined sets.

*Definition 2 (Locator Set):* We define the following set:

$$\mathcal{L}_t(n) \triangleq \mathcal{H}_t(\ell,n)\cap\mathcal{V}_t(\ell,n)\cap\mathcal{E}_t(\ell,n)\cap\mathcal{M}_t(\ell,n).$$

[2]If the value of $r_w$ does not divide $t+1$, then one can simply expand the dimension of the locator arrays in $\mathcal{E}_t(\ell,n)$ to the next multiple of $t+1$ that is greater than $r_w+(t+1)^2$.

The defining parameters of $\mathcal{L}_t(n)$ are only $t$ and $n$. By fixing those, all other parameters can be obtained from the imposed constraints. Most noteworthy parameters are $w$ and $r_w$, which are functions of $n$ and $t$. Moreover, we point out that a good choice for the parameter $\ell$ is $\log n$, which is mainly motivated by the redundancy optimization of the construction, thoroughly discussed in Section VII. Additionally, due to the aforementioned constraints on the locator arrays, $t\ell$ and $r_w+(t+1)^2$ need to be multiples of $(t+1)$ and $t\ell \geqslant \lceil \frac{t}{2}\rceil (t+1)$. For an illustration of such arrays we refer to Figure 2. Our construction works only when $\log n$ is a multiple of $t+1$ since we choose $\ell=\log n$.

### *F. Construction*

Let $\mathbb{F}_q$ denote the finite field of size $q$, $\mathbb{F}_q^n$ the vector space of length $n$ over $\mathbb{F}_q$, and the $\mathbb{F}_q^{n\times n}$ the matrix space over $\mathbb{F}_q$ of dimension $n\times n$. We write $\mathcal{C}_{\mathrm{Gab}}(n,t)\subseteq \mathbb{F}_2^{n\times n}$ to refer to a linear[3] Gabidulin code which is able to correct any pattern of $t_\mathrm{r}$ row and $t_\mathrm{c}$ column erasures in an $n\times n$ array as long as $t_\mathrm{r}+t_\mathrm{c}=t$ [24]. This is equivalent to stating that its minimum rank distance[4] is at least $t+1$. Now we are able to present our existential construction..

*Construction 1:* The code $\mathcal{C}_{t,n}\subseteq\Sigma^{n\times n}$ is the set of arrays that belong to

$$\mathcal{L}_t(n)\cap\mathcal{C}_{\mathrm{Gab}}(n,t).$$

*Theorem 11:* The code $\mathcal{C}_{t,n}$ described in Construction 1 is a $t$-criss-cross indel code.

A rough concept of our construction is as follows. We assume that the decoder knows whether a $t$-criss-cross deletion or insertion has happened from the dimension of the received array. In our codewords, we first introduce the structure $\mathcal{L}_t(n)$ to locate the indices of the inserted or deleted columns and rows. With this knowledge we can introduce erasures into the missing rows and columns and convert the deletion problem into an erasure problem which can be solved by the Gabidulin code $\mathcal{C}_{\mathrm{Gab}}(n,t)$ [24]. We call this type of decoding the *locate-decode strategy*. Theorem 11 will be proven by providing a generic decoding strategy in the next section.

## VI. DECODER

Assume a codeword $\mathbf{C}\in\mathcal{C}_{t,n}$ is transmitted and let $t_\mathrm{c}$ and $t_\mathrm{r}$ be such that $t_\mathrm{c}+t_\mathrm{r}=t$. The decoder receives an array $\widetilde{\mathbf{C}}\in\Sigma^{(n-t_\mathrm{r})\times(n-t_\mathrm{c})}$ obtained from $\mathbf{C}$ by $t_\mathrm{r}$ row and $t_\mathrm{c}$ column deletions or an array $\widetilde{\mathbf{C}}\in\Sigma^{(n+t_\mathrm{r})\times(n+t_\mathrm{c})}$ obtained from $\mathbf{C}$ by $t_\mathrm{r}$ row and $t_\mathrm{c}$ column insertions. The dimension of the received array is assumed to be known to the decoder. As mentioned before we first focus on locating the indices of inserted/deleted rows and columns.

[3]Note that such a Gabidulin code can be represented as a set of vectors in $\mathbb{F}_{2^n}^n$ as well and is linear in $\mathbb{F}_{2^n}$. For our application, it is sufficient that such a Gabidulin code is also $\mathbb{F}_2$-linear and we will always represent the codewords as binary $n\times n$ matrices.

[4]for the definition of the rank distance, cf. [35].

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

### A. Locating the Indices

Let us denote the set of indices of the rows and columns that got inserted or deleted by $\mathcal{I}^{(t_\mathrm{r})} \subset [n+t_\mathrm{r}]$ and $\mathcal{I}^{(t_\mathrm{c})} \subset [n+t_\mathrm{c}]$, respectively, with $|\mathcal{I}^{(t_\mathrm{r})}| + |\mathcal{I}^{(t_\mathrm{c})}| = t_\mathrm{r} + t_\mathrm{c} = t$. For clarity of presentation, we first present the decoding strategy for locating deletions. Subsequently, we present the decoding strategy for locating insertions.

*Claim 12:* Given the array $\widetilde{\mathbf{C}}$ affected by $t_\mathrm{r}$ row and $t_\mathrm{c}$ column deletions, the marker arrays $\widetilde{\mathbf{M}}^{(1,1)}$, $\widetilde{\mathbf{M}}^{(1,2)}$, $\widetilde{\mathbf{M}}^{(2,1)}$, and $\widetilde{\mathbf{M}}^{(2,2)}$ can be located.

*Proof:* We will focus on locating the arrays $\widetilde{\mathbf{M}}^{(2,1)}$ and $\widetilde{\mathbf{M}}^{(2,2)}$. A similar proof can be given to find the arrays $\widetilde{\mathbf{M}}^{(1,1)}$ and $\widetilde{\mathbf{M}}^{(1,2)}$ by exploiting the symmetric properties of $\mathbf{T}^{(1)}$ and $\mathbf{L}^{(1)}$.

Using Claim 7 we can locate the leftmost column of $\widetilde{\mathbf{L}}^{(2)}$. Moreover, the bottom row of $\widetilde{\mathbf{L}}^{(2)}$ is given directly by the position of $\mathbf{L}^{(2)}$ in the codeword itself.

Note that we only have to detect row or column deletions rather than exactly locating them. Our goal is to locate the rightmost column of $\widetilde{\mathbf{L}}^{(2)}$. Recall that in the last $t+1$ rows of $\widetilde{\mathbf{L}}^{(2)}$ there is for sure an $\widetilde{\mathbf{L}}'$ which originates from $\mathbf{L}'$ affected by possible row and column deletions. By a similar argument as in Claim 7, one can detect the number of column deletions that happened in each subarray of $\mathbf{L}'$ of $t+1$ consecutive columns, starting from the left. This stems from the fact that number of column runs in $\widetilde{\mathbf{L}}'$ is $t+1$ minus the number of row deletions in $\mathbf{L}'$. Recall that from Claim 7 we know the number of row deletions in $\mathbf{L}'$. The remaining ingredient is to know when $\widetilde{\mathbf{L}}'$ ends. This is guaranteed since the columns of the marker $\mathbf{M}^{(2,2)}$ are the complement of the last $t+1$ columns in $\mathbf{L}'$. Therefore, we are guaranteed to have at least one column of $\widetilde{\mathbf{M}}^{(2,2)}$ marking the end of $\widetilde{\mathbf{L}}'$ even in the presence of row or column deletions.

Given the rightmost column of $\widetilde{\mathbf{L}}^{(2)}$ we focus on locating the marker $\widetilde{\mathbf{M}}^{(2,1)}$ and therefore the topmost row of $\widetilde{\mathbf{L}}^{(2)}$. Recall that $\mathbf{L}^{(2)}$ consists of $\frac{s}{t+1}$ arrays $\mathbf{L}'$ stacked on top of each other. Using the same argument as in Claim 7, we can locate every row deletion in $\mathbf{L}^{(2)}$ until the topmost subarray $\mathbf{L}'$. This is true even in the presence of column deletions, since column deletions do not change the fact that every $t+1$ consecutive rows in $\mathbf{L}^{(2)}$ are different. By the choice of the marker $\mathbf{M}^{(2,1)}$, we ensure that the number of row deletions in the topmost $\mathbf{L}'$ can be detected. This is true because the marker has the same structure of the first $t+1$ columns of $\mathbf{L}'$. ∎

*Lemma 13:* Given the array $\widetilde{\mathbf{C}}$ affected by $t_\mathrm{r}$ row and $t_\mathrm{c}$ column deletions, any row with index $i \in \mathcal{I}^{(t_\mathrm{r})}$ such that $1 \leqslant i \leqslant t\ell$ or $n - r_w - (t+1)^2 < i \leqslant n$ can be exactly recovered. Similarly, any column index $j \in \mathcal{I}^{(t_\mathrm{c})}$ such that $1 \leqslant j \leqslant t\ell$ or $n - r_w - (t+1)^2 < j \leqslant n$ can be exactly recovered.

*Proof:* We focus on recovering the indices of the deleted rows such that $i \in \mathcal{I}^{(t_\mathrm{r})}$ and the indices of the deleted columns such that $j \in \mathcal{I}^{(t_\mathrm{c})}$ that satisfy $1 \leqslant i \leqslant t\ell$ and $n - r_w - (t+1)^2 < j \leqslant n$. Recovering the remaining indices of the statement follows by the symmetry of the construction.

From Claim 12, the location of $\widetilde{\mathbf{L}}^{(1)}$ and $\widetilde{\mathbf{T}}^{(1)}$ in $\widetilde{\mathbf{C}}$ can be exactly recovered. Therefore, by Claim 7 we can locate any column deletions with indices $n - r_w - (t+1)^2 < j \leqslant n$ by decoding $\widetilde{\mathbf{T}}^{(1)}$. Consequently, having the location of $\widetilde{\mathbf{L}}^{(1)}$ and using Claim 7, we can recover the indices of the deleted rows that satisfy $1 \leqslant i \leqslant t\ell$. Similarly, we can obtain the indices with $n - r_w - (t+1)^2 < i \leqslant n$ and $1 \leqslant j \leqslant t\ell$. ∎

*Lemma 14:* Given the array $\widetilde{\mathbf{C}}$ affected by $t_\mathrm{r}$ row and $t_\mathrm{c}$ column deletions, any row index $i \in \mathcal{I}^{(t_\mathrm{r})}$ such that $t\ell < i \leqslant n - r_w - (t+1)^2$ can be exactly recovered. Similarly, any column index $j \in \mathcal{I}^{(t_\mathrm{c})}$ such that $t\ell < j \leqslant n - r_w - (t+1)^2$ can be exactly recovered.

*Proof:* We start by proving that the column indices can be recovered. We want to leverage the structure imposed by the set $\mathcal{H}_t(\ell, n)$. For an array $\mathbf{C} \in \mathcal{H}_t(\ell, n)$, each row of the subarray $\mathbf{C}_{[1:t\ell],[t\ell+1:n-(t+1)^2]}$ is encoded using a binary systematic $t$-indel-correcting code. In addition, the columns $\mathbf{C}_{[1:t\ell],j}$ such that $t\ell < j \leqslant n - r_w - (t+1)^2$ are the systematic part of this code. Recall that the rows are divided into $t$ blocks, each of size $\ell$, where in each block the columns $t < j \leqslant n - r_w - (t+1)^2$ satisfy the window constraint. We assume that at least one column in this interval is deleted. Therefore, at most $(t-1)$ rows can be deleted in $\mathbf{C}$. This means that there exists at least one block of $\ell$ rows that is not affected by any row deletion. For the deletion case, by Lemma 13 we can locate this block. By Claim 9 we can recover the indices of the columns deleted within the range $t < j \leqslant n - r_w - (t+1)^2$. Similarly, we can obtain the indices with $t\ell < i \leqslant n - r_w - (t+1)^2$ by leveraging the structure imposed by $\mathcal{V}_t(\ell, n)$ using Claim 9. ∎

We now present the equivalent results for the insertion case. In the following statements we only highlight the differences from the deletion case. The main difference is that we need to tackle possible insertion patterns which can create block confusions in our received array $\widetilde{\mathbf{C}}$. In general, the strategy is as follows. Since all rows and columns of the codeword $\mathbf{C}$ satisfy the window constraint, even after $t$-criss-cross insertions, we can exploit the fact that insertions can only create a confusion of length at most $2t$ with at most $t$ rows/columns of the original array. Therefore, we can delete the block confusions and turn the insertion locating problem into a deletion locating problem, which we have shown earlier how to solve.

*Claim 15:* Given the array $\widetilde{\mathbf{C}}$ affected by $t_\mathrm{r}$ row and $t_\mathrm{c}$ column insertions, the marker array $\widetilde{\mathbf{M}}^{(1,1)}$ can be located up to row and column block confusion of length at most $2t$, and $\widetilde{\mathbf{M}}^{(1,2)}$ can be located up to column block confusion of length at most $2t$ and row block confusion of length at most $2t+1$. The marker array $\widetilde{\mathbf{M}}^{(2,1)}$ can be located up to row and column block confusion of length at most $2t$, and $\widetilde{\mathbf{M}}^{(1,2)}$ can be located up to column block confusion of length at most $2t$ and row block confusion of length at most $2t+1$.

*Proof:* We have the same preliminary information as in Claim 8 and 12. However, if the inserted rows/columns create block confusions exactly at the border of the arrays $\mathbf{L}^{(1)}$ and $\mathbf{T}^{(1)}$, or $\mathbf{L}^{(2)}$ and $\mathbf{T}^{(2)}$ we can locate the desired arrays only up to the length of the block confusion. By construction, the length of the block confusion is limited to the parameter range given in the statement of the claim. Moreover, we use the following strategy for locating the border between $\mathbf{L}^{(1)}$ and $\mathbf{T}^{(1)}$ in presence of block confusions, where the same can

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

be applied to the border of $\mathbf{L}^{(2)}$ and $\mathbf{T}^{(2)}$ by symmetry of the construction. The idea is to use a block of indel codes to resolve the row block confusion. Note that there must be at least one indel code block which is not affected by insertions due to the existence of $t$ indel code subarrays. Moreover, we can determine the affected arrays by Claim 8. We declare the border of the indel code to be the first expected row index of $\mathbf{T}^{(1)}$. Note that by declaring a wrong border, we add at most $t$ insertions to the indel code. By Claim 10 we can decode the indel code. The correct border can then be determined by the first index which is not marked as an insertion in the indel code subarray. In case no insertions are declared at the beginning of the subarray, then the chosen border is correct.

Moreover, in case $\widetilde{\mathbf{M}}^{(1,2)}$ and $\widetilde{\mathbf{M}}^{(2,2)}$ are located up to a column block confusion and a row block confusion, respectively, one can simply ignore the confusion to locate $\widetilde{\mathbf{M}}^{(1,1)}$ and $\widetilde{\mathbf{M}}^{(2,1)}$ since the inserted columns must follow the structure of $\mathbf{T}^{(1)}$ and $\mathbf{L}^{(2)}$ to create a confusion. ∎

Observe that the marker arrays $\widetilde{\mathbf{M}}^{(1,2)}$ and $\widetilde{\mathbf{M}}^{(2,2)}$ can be located up to a row, or respectively a column block, confusion of length at most $2t+1$. The confusion may contain more than $t$ original rows/columns of the original array. However, since the indices of the block confusions are within the index range of the indel codes, we can use those as stated in Lemma 17 to tackle this problem.

*Lemma 16:* Given the array $\widetilde{\mathbf{C}}$ affected by $t_\mathrm{r}$ row and $t_\mathrm{c}$ column insertions, any row with index $i \in \mathcal{I}^{(t_\mathrm{r})}$ such that $1 \leqslant i \leqslant t\ell$ or $n - r_w - (t+1)^2 < i \leqslant n$ can be recovered up to a row block confusion of length at most $2t$ consisting of at at most $t$ rows of the original array $\mathbf{C}$. Similarly, any column index $j \in \mathcal{I}^{(t_\mathrm{c})}$ such that $1 \leqslant j \leqslant t\ell$ or $n - r_w - (t+1)^2 < j \leqslant n$ can be recovered up to a column block confusion of length at most $2t$ consisting of at at most $t$ columns of the original array $\mathbf{C}$.

*Proof:* We focus on recovering the indices of the inserted rows such that $i \in \mathcal{I}^{(t_\mathrm{r})}$ and the indices of the inserted columns such that $j \in \mathcal{I}^{(t_\mathrm{c})}$ that satisfy $1 \leqslant i \leqslant t\ell$ and $n - r_w - (t+1)^2 < j \leqslant n$. Recovering the remaining indices of the statement follows by symmetry of the construction. In general, we can use the same strategy as presented for the deletion case using Claim 8 and 15. In case the marker arrays are located up to a confusion, one can ignore the rows/columns which created a confusion, i.e., we only consider one collection of the rows/columns of the confusion, since the inserted rows/columns must satisfy the fixed structure of the locator arrays $\mathbf{T}^{(1)}$, $\mathbf{L}^{(1)}$, $\mathbf{T}^{(2)}$, and $\mathbf{L}^{(2)}$. ∎

*Lemma 17:* Given the array $\widetilde{\mathbf{C}}$ affected by $t_\mathrm{r}$ row and $t_\mathrm{c}$ column insertions, any row index $i \in \mathcal{I}^{(t_\mathrm{r})}$ such that $t\ell < i \leqslant n - r_w - (t+1)^2$ can be recovered up to a row block confusion of length at most $2t$ consisting of at at most $t$ rows of the original array $\mathbf{C}$. Similarly, any column index $j \in \mathcal{I}^{(t_\mathrm{c})}$ such that $t\ell < j \leqslant n - r_w - (t+1)^2$ can be recovered up to a column block confusion of length at most $2t$ consisting of at most $t$ columns of the original array $\mathbf{C}$.

*Proof:* We start by proving that the column indices can be recovered. We want to leverage the structure imposed by the set $\mathcal{H}_t(\ell, n)$. We assume that at least one column in this interval is inserted. This means that there exists at least one block of $\ell$ rows that is not affected by any row insertion. By Lemma 16 we can locate this block by remarking that at least one insertion is needed to create a block confusion and therefore affect the block. By Claim 10 we can recover the indices of the columns inserted within the range $t < j \leqslant n - r_w - (t+1)^2$ up to a block confusion of length at most $2t$ containing at most $t$ columns of the original array. Similarly, we can obtain the indices with $t\ell < i \leqslant n - r_w - (t+1)^2$ by leveraging the structure imposed by $\mathcal{V}_t(\ell, n)$, cf., Lemma 16, and Claim 10. ∎

### *B. Recovering the Transmitted Array*

Now we can present the full proof of our code construction.

*Proof:* [Proof of Theorem 11] If a $t$-criss-cross deletion happened, we can apply Lemma 13 and 14 to determine the sets of indices $\mathcal{I}^{(t_\mathrm{r})}$ and $\mathcal{I}^{(t_\mathrm{c})}$. Then, for all $i \in \mathcal{I}^{(t_\mathrm{r})}$ and $j \in \mathcal{I}^{(t_\mathrm{c})}$ the decoder inserts row or column erasures in $\widetilde{\mathbf{C}}$ starting from the smallest index. Now the decoder applies a Gabidulin criss-cross erasure decoder to determine the values of the erased symbols [24].

In case of a $t$-criss-cross insertion we apply Lemma 16 and 17. The decoder deletes the inserted rows/columns which their positions are exactly recovered. For each block confusion, the decoder deletes the whole block confusion. This deletion strategy deletes at most $t$ rows/columns of the original array, since all rows/columns follow the window constraint. Thus, the decoder inserts row/column erasures in $\widetilde{\mathbf{C}}$ starting from the smallest index and applies a Gabidulin criss-cross erasure decoder to determine the values of the erased symbols [24]. ∎

## VII. REDUNDANCY

In this section we perform an analysis of the redundancy of our code denoted by $R(n,t)$. We will refer to the redundancy of each individual set $\mathcal{C}_{\mathrm{Gab}}(t,n)$, $\mathcal{L}_t(n)$, $\mathcal{H}_t(\ell,n)$, $\mathcal{V}_t(\ell,n)$, $\mathcal{E}_t(\ell,n)$, $\mathcal{W}_t(\ell,w)$ and $\mathcal{M}_t(\ell,n)$ by $R_*(n,t)$, where $*$ is replaced with the corresponding set letter. In the following, we give an intuition behind the computations of the redundancy.

Since $\mathcal{C}_{t,n} = \mathcal{L}_t(n) \cap \mathcal{C}_{\mathrm{Gab}}(t,n)$ and due to the fact that the Gabidulin code is a linear code, we can compute the code redundancy as follows.

$$R(n,t) \leqslant R_{\mathcal{L}}(n,t) + R_{\mathrm{G}}(n,t).$$

Moreover, since the intersected sets in the locator set $\mathcal{L}_t(n)$ impose constraints on disjoint positions in the $n \times n$ arrays, we can further split the redundancy as follows

$$R_{\mathcal{L}}(n,t) = R_{\mathcal{H}}(n,t) + R_{\mathcal{V}}(n,t) + R_{\mathcal{E}}(n,t) + R_{\mathcal{M}}(n,t).$$

The sets $\mathcal{H}_t(\ell,n)$ and $\mathcal{V}_t(\ell,n)$ impose similar constraints: $t$ disjoint subarrays constrained with the window constraint where each row is protected by a systematic $t$-deletion correcting code from [18].

*Claim 18:* The redundancy resulting from the constraints imposed by the two sets $\mathcal{H}_t(\ell,n)$ and $\mathcal{V}_t(\ell,n)$ is bounded as

$$R_{\mathcal{H}}(n,t) + R_{\mathcal{V}}(n,t) \leqslant 2t(R_{\mathcal{W}}(\ell,w) + \log(n) \cdot r_w),$$

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

where $w = n - t\log(n) - r_w - (t+1)^2$ and $r_w \leqslant 4t\log(n) + o(\log n)$.

*Proof:* The sets $\mathcal{H}_t(\ell, n)$ and $\mathcal{V}_t(\ell, n)$ impose the same constraints, i.e., each array belonging to any of these sets has $t$ subarrays protected by deletion-correcting codes with window constraints. Therefore, we have

$$R_{\mathcal{H}}(n,t) + R_{\mathcal{V}}(n,t) = 2t(R_{\mathcal{W}}(\ell, w) + \log(n) \cdot r_w),$$

where $r_w$ is the length of the redundancy vector used to protect a vector of length

$$w = n - t\log(n) - r_w - (t+1)^2 \qquad (1)$$

against $t$ deletions, and $\log(n)$ is the number of protected vectors in each subarray. Recall that for any integer $\kappa$, the redundancy for protecting a vector of length $\kappa$ is bounded by $r_\kappa \leqslant 4t\log(\kappa) + o(\log(\kappa))$ [18].

$$r_w \leqslant (4t+1)\log(n). \qquad (2)$$

We now focus on computing $R_{\mathcal{W}}(\ell, w)$. To compute an upper bound on the redundancy imposed by the window constraint $\mathcal{W}_t(\ell, w)$ we require a lower bound on $w$. Note that using a lower bound on $w$ only increases the redundancy imposed by $\mathcal{W}_t(\ell, w)$. This will be clear from the following calculations. From (1) and (2) we obtain

$$w \geqslant n - (5t+1)\log(n) - (t+1)^2.$$

We calculate a lower bound on $|\mathcal{W}_t(\ell, w)|$. On a high level, our calculations are interpreted as going through each column of an array in $\mathcal{W}_t(\ell, w)$ and counting the number of choices for this specific column. The first column is arbitrary, thus has $2^\ell$ choices. The second column is not allowed to be the same as the one before, thus it has $(2^\ell - 1)$ choices. The third column has $(2^\ell - 2)$ choices, since it cannot be the same as the two preceding columns. This process continues until we reach the $(t+2)^{\text{nd}}$ column. The number of choices for this vector is $(2^\ell - (t+1))$. Since the restriction is imposed on an interval of $(t+1)$ vectors, each remaining columns has $(2^\ell - (t+1))$ choices. Thus, for the window constraint the following holds.

$$\begin{aligned}|\mathcal{W}_t(\ell, w)| &\geqslant 2^\ell \cdot (2^\ell - 1) \cdot \ldots \cdot (2^\ell - (t+1)) \\ &\quad \cdot (2^\ell - (t+1))^{w-t-2} \\ &\geqslant (2^\ell - (t+1))^w \\ &= 2^{\ell w}\left(1 - \frac{t+1}{2^\ell}\right)^w.\end{aligned}$$

We denote the redundancy resulting from the constraints imposed by the window constraint as $R_{\mathcal{W}}(\ell, n - t\ell - r_w)$. We continue the calculations recalling that $\ell = \log(n)$.

$$\begin{aligned}&R_{\mathcal{W}}(\ell, n - t\ell - r_w) \\ &\leqslant \ell(n - t\ell - r_w) - \log\left(|\mathcal{W}_t(\ell, n - t\ell - r_w)|\right) \\ &\leqslant \log\left(\left(1 - \frac{t+1}{2^\ell}\right)^{n - t\ell - r_w}\right) \\ &\leqslant \log\left(\left(1 - \frac{t+1}{n}\right)^n\right) - \log\left(\left(1 - \frac{t+1}{n}\right)^{(5t+1)\log(n)}\right) \\ &\overset{(a)}{\leqslant} \log(e^{(t+1)}) - (5t+1)\log(n) \cdot \log\left(\left(1 - \frac{t+1}{n}\right)\right) \\ &\overset{(b)}{\leqslant} (t+1)\log(e) + (5t+1)\log(n) \\ &\leqslant (5t+1)\log(n) + 2(t+1).\end{aligned}$$

We used in $(a)$ the inequality $\left(1 - \frac{x}{n}\right)^n \leqslant e^x$ and exploited in $(b)$ the fact that $\frac{1}{2} \leqslant (1 - \frac{t+1}{n}) \leqslant 1$ for our choice of parameters and for sufficiently large $n$.

Recall that any array in $\mathcal{D}_t^{(1)}(\ell, n - t\ell - (t+1)^2)$ consists of $\log(n)$ binary $t$-deletion correcting codes. Therefore, we have

$$\begin{aligned}&R_{\mathcal{H}}(n,t) \\ &\leqslant t \cdot \Big(\underbrace{(5t+1)\log(n) + 2(t+1)}_{\text{window constraint}} \\ &\qquad + \underbrace{\log(n) \cdot (4t+1)\log(n)}_{\text{binary deletion correcting codes}}\Big) \\ &= (4t^2 + t)\log^2(n) + (5t^2 + t)\log(n) + 2t(t+1)^2.\end{aligned}$$

Since the arrays in $\mathcal{V}_t(\ell, n)$ have a similar structure imposed (only transposed) and the regions of the imposed constraints are disjoint, one can conclude that

$$\begin{aligned}&R_{\mathcal{H}}(n,t) + R_{\mathcal{V}}(n,t) \\ &\leqslant 2(4t^2 + t)\log^2(n) + 2(5t^2 + t)\log(n) + 4t(t+1)^2.\end{aligned}$$

■

Observe that the constraints for the remaining sets fix values for certain subarray boundaries. Therefore, the following can be obtained.

*Claim 19:* The redundancy $R_{\mathcal{L}}(n,t)$ resulting from the constraints imposed by the set $\mathcal{L}_t(n)$ is bounded as

$$R_{\mathcal{L}}(n,t) \leqslant (8t^2 + 2t)\log^2(n) + o(\log^2(n)).$$

*Proof:* We argued that since the different constraints are imposed on disjoint subarrays in $\mathcal{L}_t(n)$, then the redundancy $R_{\mathcal{L}}(n,t)$ can be written as

$$R_{\mathcal{L}}(n,t) = R_{\mathcal{H}}(n,t) + R_{\mathcal{V}}(n,t) + R_{\mathcal{E}}(n,t) + R_{\mathcal{M}}(n,t)$$

The redundancy imposed by the locator arrays and marker arrays is equal to the dimension of the subarrays with fixed entries. We can then write

$$\begin{aligned}R_{\mathcal{E}}(n,t) &\leqslant (6t^3 + 13t^2 + 8t + 1)\log(n), \\ R_{\mathcal{M}}(n,t) &= 4(t+1)^2.\end{aligned}$$

The other terms of the redundancy in $R_{\mathcal{L}}(n,t)$ are computed in Claim 18. ■

We can conclude this section with the statement on the redundancy $R(n,t)$ of the code $\mathcal{C}_{t,n}$ presented in Construction 1. Note that the redundancy added by the Gabidulin code is $tn$.

*Lemma 20:* The redundancy of the code $\mathcal{C}_{t,n}$ is bounded as

$$R(n,t) \leqslant tn + (8t^2 + 2t)\log^2(n) + o(\log^2(n)).$$

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

*Proof:* By construction we have that $\mathcal{C}_{t,n} = \mathcal{L}_t(n) \cap \mathcal{C}_{\mathrm{Gab}}(n,t)$. By Claim 19 we have that

$$|\mathcal{L}_t(n)| \geqslant \frac{2^{n^2}}{n^{((8t^2+2t)+o(1))\log(n)}}.$$

From [24] we have that $|\mathcal{C}_{\mathrm{Gab}}(n,t)| = \frac{2^{n^2}}{2^{tn}}$. Further, due to the fact that $\mathcal{C}_{\mathrm{Gab}}(n,t)$ is a linear code, there exists a coset such that the following is satisfied by means of the pigeon hole principle.

$$|\mathcal{C}_{t,n}| \geqslant 2^{n^2} \cdot \underbrace{\frac{1}{2^{tn}}}_{\text{Gabidulin Code}} \cdot \underbrace{\frac{1}{n^{((8t^2+2t)+o(1))\log(n)}}}_{\text{Locator Set}}$$

Hence, we can conclude that the total redundancy of the $\mathcal{C}_{t,n}$ satisfies

$$\begin{aligned} R(n,t) &= n^2 - \log(|\mathcal{C}_{t,n}|) \\ &\leqslant tn + (8t^2+2t)\log^2(n) + o(\log^2(n)). \end{aligned}$$

■

## VIII. Conclusion

In this work we have considered the $t$-criss-cross insertion/deletion problem in binary arrays. First, we have shown that the one-dimensional insertion-deletion equivalence also holds in the two-dimensional array setting. Moreover, we have shown that the asymptotic lower bound on the redundancy for any $t$-criss-cross correcting code is $R_{\mathrm{B}}(n,t) \geqslant tn + t\log(n) - \mathcal{O}(1)$. We have presented our $t$-criss-cross indel code construction which is based on the strategy of transforming the insertion/deletion problem to an erasure problem. The redundancy of the constructed $t$-criss-cross indel code is $\mathcal{O}(t^2\log^2(n))$ far from the derived lower bound. We note that given an order optimal systematic construction of $n$-ary $t$-indel-correcting codes, we could improve our construction such that the redundancy is only $\mathcal{O}(t^3\log(n))$ far from the derived lower bound. This results from replacing the $2t\log n$ binary $t$-indel-correcting codes indexing the columns/rows by $2t$ $n$-ary $t$-indel-correcting codes.

On a final note, improvements on the problem of coding for indel errors in arrays remain possible. It would be interesting to generalize the problem to study possible combinations of simultaneous insertions and deletions in arrays. In this case, generalizing the one-dimensional equivalence between insertions and deletions amounts to generalizing the equivalence between criss-cross insertion correcting codes and criss-cross deletion correcting codes (Theorem 1) to proving that a code able to correct $t$ criss-cross deletions can correct any number of $t_{\mathrm{r}}$ row insertions (or deletions) and $t_{\mathrm{c}}$ column deletions (or insertions) such that $t = t_{\mathrm{r}} + t_{\mathrm{c}}$. The next step would then be finding a code construction, with redundancy close to the lower bound derived in this paper, that can correct a mixtures of indel column and row errors. Further research topics in this direction include studying the characteristics of the indel spheres of an array $\mathbf{X}$.

## Acknowledgment

The authors would like to thank the associate editor and the anonymous reviewers for their valuable comments that contributed to the improvement of the quality of this work. Further, they want to thank Evagoras Stylianou for his observations concerning the decoder for detecting insertions, leading to a better explanation of the decoder.

## References

[1] L. Welter, R. Bitar, A. Wachter-Zeh, and E. Yaakobi, "Multiple criss-cross deletion-correcting codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2021, pp. 2798–2803.
[2] R. Heckel, G. Mikutis, and R. N. Grass, "A characterization of the DNA data storage channel," *Sci. Rep.*, vol. 9, no. 1, pp. 1–12, Dec. 2019.
[3] F. Sala, C. Schoeny, N. Bitouzé, and L. Dolecek, "Synchronizing files from a large number of insertions and deletions," *IEEE Trans. Commun.*, vol. 64, no. 6, pp. 2258–2273, Jun. 2016.
[4] R. Venkataramanan, H. Zhang, and K. Ramchandran, "Interactive low-complexity codes for synchronization from deletions and insertions," in *Proc. 48th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2010, pp. 1412–1419.
[5] R. Venkataramanan, V. N. Swamy, and K. Ramchandran, "Low-complexity interactive algorithms for synchronization from deletions, insertions, and substitutions," *IEEE Trans. Inf. Theory*, vol. 61, no. 10, pp. 5670–5689, Oct. 2015.
[6] S. M. S. T. Yazdi and L. Dolecek, "A deterministic polynomial-time protocol for synchronizing from deletions," *IEEE Trans. Inf. Theory*, vol. 60, no. 1, pp. 397–409, Jan. 2013.
[7] N. Ma, K. Ramchandran, and D. Tse, "Efficient file synchronization: A distributed source coding approach," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2011, pp. 583–587.
[8] L. Dolecek and V. Anantharam, "Using Reed–Müller RM(1, $m$) codes over channels with synchronization and substitution errors," *IEEE Trans. Inf. Theory*, vol. 53, no. 4, pp. 1430–1443, Apr. 2007.
[9] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Doklady Akademii Nauk SSR*, vol. 163, no. 4, pp. 845–848, 1965.
[10] R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors (in Russian)," *Automatika Telemkhanika*, vol. 161, no. 3, pp. 288–292, 1965.
[11] V. Guruswami and C. Wang, "Deletion codes in the high-noise and high-rate regimes," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 1961–1970, Apr. 2017.
[12] J. Brakensiek, V. Guruswami, and A. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," *IEEE Trans. Inf. Theory*, vol. 64, no. 5, pp. 3403–3410, May 2018.
[13] S. K. Hanna and S. E. Rouayheb, "Guess & check codes for deletions, insertions, and synchronization," *IEEE Trans. Inf. Theory*, vol. 65, no. 1, pp. 3–15, Jan. 2019.
[14] R. Gabrys and F. Sala, "Codes correcting two deletions," *IEEE Trans. Inf. Theory*, vol. 65, no. 2, pp. 965–974, Feb. 2019.
[15] J. Sima, N. Raviv, and J. Bruck, "Two deletion correcting codes from indicator vectors," *IEEE Trans. Inf. Theory*, vol. 66, no. 4, pp. 2375–2391, Apr. 2020.
[16] J. Sima and J. Bruck, "On optimal $k$-deletion correcting codes," *IEEE Trans. Inf. Theory*, vol. 67, no. 6, pp. 3360–3375, Jun. 2021.
[17] V. Guruswami and J. Hastad, "Explicit two-deletion codes with redundancy matching the existential bound," *IEEE Trans. Inf. Theory*, vol. 67, no. 10, pp. 6384–6394, Oct. 2021.
[18] J. Sima, R. Gabrys, and J. Bruck, "Optimal systematic $t$-deletion correcting codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2020, pp. 769–774.
[19] A. Krishnamurthy, A. Mazumdar, A. McGregor, and S. Pal, "Trace reconstruction: Generalized and parameterized," 2019, *arXiv:1904.09618*.
[20] C. Schoeny, A. Wachter-Zeh, R. Gabrys, and E. Yaakobi, "Codes correcting a burst of deletions or insertions," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 1971–1985, Apr. 2017.
[21] D. Smith, T. G. Swart, K. A. S. Abdel-Ghaffar, H. C. Ferreira, and L. Cheng, "Interleaved constrained codes with markers correcting bursts of insertions or deletions," *IEEE Commun. Lett.*, vol. 21, no. 4, pp. 702–705, Apr. 2017.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.

[22] S. Bakirtas and E. Erkip, "Database matching under column deletions," 2021, *arXiv:2105.09616*.

[23] R. M. Roth, "Maximum-rank array codes and their application to crisscross error correction," *IEEE Trans. Inf. Theory*, vol. 37, no. 2, pp. 328–336, Mar. 1991.

[24] E. M. Gabidulin and N. I. Pilipchuk, "Error and erasure correcting algorithms for rank codes," *Des., Codes Cryptogr.*, vol. 49, no. 1, pp. 105–122, Dec. 2008.

[25] D. Lund, E. M. Gabidulin, and B. Honary, "A new family of optimal codes correcting term rank errors," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2000, p. 115.

[26] V. R. Sidorenko, "Class of correcting codes for errors with a lattice configuration," *Problemy Reredachi Informatsii*, vol. 12, no. 3, pp. 165–171, Mar. 1976.

[27] M. Blaum and J. Bruck, "MDS array codes for correcting a single crisscross error," *IEEE Trans. Inf. Theory*, vol. 46, no. 3, pp. 1068–1077, May 2000.

[28] È. M. Gabidulin, "Optimum codes correcting lattice errors," *Problems Inf. Transmiss.*, vol. 21, no. 2, pp. 103–108, 1985.

[29] R. M. Roth, "Probabilistic crisscross error correction," *IEEE Trans. Inf. Theory*, vol. 43, no. 5, pp. 1425–1438, Sep. 1997.

[30] A. Wachter-Zeh, "List decoding of crisscross errors," *IEEE Trans. Inf. Theory*, vol. 63, no. 1, pp. 142–149, Jan. 2016.

[31] R. Bitar, L. Welter, I. Smagloy, A. Wachter-Zeh, and E. Yaakobi, "Crisscross insertion and deletion correcting codes," *IEEE Trans. Inf. Theory*, vol. 67, no. 12, pp. 7999–8015, Dec. 2021.

[32] R. Bitar, I. Smagloy, L. Welter, A. Wachter-Zeh, and E. Yaakobi, "Crisscross deletion correcting codes," in *Proc. Int. Symp. Inf. Theory Appl. (ISITA)*, Oct. 2020, pp. 304–308.

[33] M. Hagiwara, "Conversion method from erasure codes to multi-deletion error-correcting codes for information in array design," in *Proc. Int. Symp. Inf. Theory Appl. (ISITA)*, 2020, pp. 274–278.

[34] A. A. Kulkarni and N. Kiyavash, "Nonasymptotic upper bounds for deletion correcting codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 8, pp. 5115–5130, Aug. 2013.

[35] È. M. Gabidulin, "Theory of codes with maximum rank distance," *Problemy Peredachi Informatsii*, vol. 21, no. 1, pp. 3–16, 1985.



**Lorenz Welter** (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering and information technology from the Karlsruhe Institute of Technology (KIT), Germany, in 2015 and 2018, respectively. He is currently pursuing the Ph.D. degree with the Coding and Cryptography Group, Institute of Communications Engineering, Technical University of Munich (TUM), under the supervision of Prof. Wachter-Zeh. His research interests include coding theory and its applications with focus on insertion and deletion errors.

**Rawad Bitar** (Member, IEEE) received the Diploma degree in computer and communication engineering from the Faculty of Engineering, Lebanese University, Roumieh, Lebanon, in 2013, the M.S. degree from the Doctoral School, Lebanese University, Tripoli, Lebanon, in 2014, and the Ph.D. degree in electrical engineering from Rutgers, The State University of New Jersey, The State University of New Jersey, NJ, USA, in 2020. He is currently a Post-Doctoral Researcher at the Technical University of Munich. His research interests include information theory and coding theory with a focus on coding for insertions and deletions and coding for information theoretically secure distributed systems with application to machine learning.

**Antonia Wachter-Zeh** (Senior Member, IEEE) received the M.Sc. degree in communications technology from Ulm University, Germany, in 2009, and the Ph.D. degree from Ulm University and the Universite de Rennes 1, Rennes, France, in 2013. From 2013 to 2016, she was a Post-Doctoral Researcher at the Technion—Israel Institute of Technology, Haifa, Israel. From 2016 to 2020, she was a Tenure Track Assistant Professor at TUM. She is currently an Associate Professor at the Department of Electrical and Computer Engineering, Technical University of Munich (TUM), Munich, Germany. Her research interests include coding theory, cryptography, and information theory, and their application to storage, communications, privacy, and security. She was a recipient of the DFG Heinz Maier-Leibnitz-Preis and an ERC Starting Grant.

**Eitan Yaakobi** (Senior Member, IEEE) received the B.A. degree in computer science and mathematics and the M.Sc. degree in computer science from the Technion—Israel Institute of Technology, Haifa, Israel, in 2005 and 2007, respectively, and the Ph.D. degree in electrical engineering from the University of California, San Diego, in 2011.

From 2011 to 2013, he was a Post-Doctoral Researcher with the Department of Electrical Engineering, California Institute of Technology, and the Center for Memory and Recording Research, University of California, San Diego. He is currently an Associate Professor at the Computer Science Department, Technion—Israel Institute of Technology. His research interests include information and coding theory with applications to non-volatile memories, associative memories, DNA storage, data storage and retrieval, and private information retrieval. He received the Marconi Society Young Scholar in 2009 and the Intel Ph.D. Fellowship in 2010–2011.


Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on August 23,2022 at 16:17:06 UTC from IEEE Xplore. Restrictions apply.