Error-correcting Codes for Noisy Duplication Channels

Yuanyuan Tang and Farzad Farnoud (Hassanzadeh)
Electrical & Computer Engineering, University of Virginia, {yt5tz,farzad}@virginia.edu

Abstract

Because of its high data density and longevity, DNA is emerging as a promising candidate for satisfying increasing data storage needs. Compared to conventional storage media, however, data stored in DNA is subject to wider range of errors resulting from various processes involved in the data storage pipeline. In this paper, we consider correcting duplication errors for both exact and noisy tandem duplications of a given length k. Specifically, we design codes that can correct any number of exact duplication and one noisy duplication errors, where in the noisy duplication case the copy is at Hamming distance 1 from the original. Our constructions rely upon recovering the duplication root of the stored codeword. We characterize the ways in which duplication errors manifest in the root of affected sequences and design efficient codes for correcting these error patterns. We show that the proposed construction is asymptotically optimal.

Index Terms

DNA storage, exact tandem duplication, noisy tandem duplication, error-correcting codes

I. Introduction

The rapidly increasing amount of data and the need for long-term data storage have led to new challenges. In recent years, advances in DNA sequencing, synthesis, and editing technologies [15], [13] have made deoxyribonucleic acid (DNA) a promising alternative to conventional storage media. Compared to traditional media, DNA has several advantages, including high data density, longevity, and ease of copying information. For example, it may be possible to recover a DNA sequence after $10,000$ years and a single human cell contains an amount of DNA that can ideally hold 6.4 Gb of information [15]. However, DNA storage technologies also encounter many challenges. One obvious challenge is that a diverse set of errors are possible, including substitution, duplication, insertion, and deletion. This paper focuses on error-correcting codes for noisy duplication channels. In such case, in addition to exact duplication, noisy duplication, where an approximate copy is inserted into the sequence, may occur.

In duplication channels, (tandem) duplication errors generate copies of substrings of the sequence and insert each copy after the original substring [4]. This type of channel was first studied in the context of recovering from timing errors in communication systems that led to individual symbols being repeated [2]. The copying mechanism of DNA, however, allows multiple symbols being repeated, for example, via slipped-strand mispairings, where the slippage of the molecule copying DNA causes a substring to be repeated [4]. Properties of duplication in DNA have been studied from various vantage points, including the theory of formal languages and the entropy of DNA sequences (see, e.g., [8] and references therein). Codes for correcting duplication errors in the context of data storage in the DNA of living organisms, such as bacteria [10], were studied by [4], where optimal constructions for correcting exact duplications of constant length were presented. This and related problems were then further studied by a number of works including [5], [16], [6], [7], [1], [13]. Most related to this paper is [13], which studies error correction in duplication and substitution channels, when substitutions are independent from duplications and when they only occur in copies generated by duplications. The latter model, i.e., the *noisy duplication model*, which is motivated by the abundance of inexact copies in tandem repeat stretches in genomes [9], is the model studied in this work.

In the noisy duplication channel, two types of errors are possible: i) exact duplications, which insert an exact copy of a substring in tandem, such as $\mathsf{ACGTC} \to \mathsf{ACGT\underline{CGT}C}$; and ii) noisy duplications, which insert approximate copies, e.g., $\mathsf{ACGTC} \to \mathsf{ACGT\underline{CTT}C}$. In both cases, the length of the duplication refers to the length of the duplicated substring (3 in our preceding examples). In this paper, we limit our attention to exact and noisy tandem duplications of length k, referred to as k-TDs and k-NDs, respectively. Furthermore, we only consider noisy duplications where the copy and the original substring differ in one position. In other words, each noisy duplication can be viewed as an exact duplication followed by a substitution in the inserted copy.

We will design codes that correct (infinitely) many k-TD and a single k-ND errors, as a step towards codes that can correct t_1 k-TDs and t_2 k-NDs, for given t_1 and t_2. The proposed codes will rely on finding the duplication root of the stored codeword. The *duplication root* of a sequence $\boldsymbol{x}$ is the sequence obtained from $\boldsymbol{x}$ by removing all repeats of length k. While

This work was supported in part by NSF grants under grant nos. 1816409 and 1755773. This paper was presented in part at the 57th Annual Allerton Conference on Communication, Control, and Computing in 2019 [12]

k-TDs do not alter the duplication root, k-NDs do. Thus, we will first analyze the effect of noisy duplications on the root of the sequence. We show that the root may change in a variety of ways, leading to several error patterns. We then design efficient error-correcting codes that correct these errors via a number of transforms that simplify the different error patterns.

It was shown in [4] that the rate of the optimal code capable of correcting many k-TDs is

$$1 - \frac{(q-1)\log_q e}{q^{k+2}} + o(1), \tag{1}$$

as the length n of the code grows, where q is the size of the alphabet. The question then arises as to whether it is possible to correct an additional noisy duplication without a rate penalty. It is worth noting that the best known code for correcting an additional unrestricted substitution, i.e., a substitution that can occur anywhere rather than in a copy generated by duplication, has rate that is bounded from below by [13]

$$1 - \frac{2}{k}\log_q \frac{q}{q-1} + o(1). \tag{2}$$

which indicates a rate penalty. In contrast, we show that the proposed codes have the same asymptotic rate as (1), and are thus asymptotically optimal.

This paper is organized as follows. The notation and preliminaries are given in Section II. In Section III, we analyze the error patterns that manifest as the result of passing through the noisy duplication channel. Finally, the code construction and the corresponding code size are presented in Section IV. Note this

II. Notation and Preliminaries

Throughout the paper, Σ_q represents a finite alphabet of size q, assumed without loss of generality to be $\{0, 1, \ldots, q-1\}$. We use Σ_q^+ to denote the nonzero elements of Σ_q and Σ_q^* to denote all strings of finite length over Σ_q. In particular, Σ_q^* includes the empty string Λ. Furthermore, Σ_q^n represents the strings of length n over Σ_q. The set $\{1, \ldots, n\}$ is represented by $[n]$.

We use bold symbols, such as $\boldsymbol{x}$ and $\boldsymbol{y}_j$, to denote strings over Σ_q. The entries of strings are shown with normal symbols, e.g., $\boldsymbol{x} = x_1 x_2 \cdots x_n$ and $\boldsymbol{y}_j = y_{j1} y_{j2} \cdots y_{jm}$, where $x_i, y_{ji} \in \Sigma_q$. The indices of elements of words over Σ_q^* start from 1, unless otherwise stated. For two words $\boldsymbol{x}, \boldsymbol{y} \in \Sigma_q^*$, their concatenation is denoted as $\boldsymbol{xy}$, and $\boldsymbol{x}^m$ represents the concatenation of m copies of $\boldsymbol{x}$. Given a word $\boldsymbol{x} \in \Sigma_q^*$, the length of $\boldsymbol{x}$ is represented as $|\boldsymbol{x}|$. In addition, for a word $\boldsymbol{x} \in \Sigma_q^*$, the Hamming weight $\mathrm{wt}(\boldsymbol{x})$ denotes the number of non-zero symbols in $\boldsymbol{x}$. If a word $\boldsymbol{x} \in \Sigma_q^*$ can be expressed as $\boldsymbol{x} = \boldsymbol{uvw}$ with $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \Sigma_q^*$, then $\boldsymbol{v}$ is a substring of $\boldsymbol{x}$.

Given a word $\boldsymbol{x} \in \Sigma_q^*$, an (exact) *tandem duplication* of length k (k-TD) generates a copy of a substring $\boldsymbol{v}$ of $\boldsymbol{x}$ of length k and inserts the copy immediately after $\boldsymbol{v}$. More specifically, a k-TD can be expressed as [4]

$$T_{i,k}(\boldsymbol{x}) = \begin{cases} \boldsymbol{uvvw} & \text{if } \boldsymbol{x} = \boldsymbol{uvw}, |\boldsymbol{u}| = i, |\boldsymbol{v}| = k \\ \boldsymbol{x} & \text{if } |\boldsymbol{x}| < i + k \end{cases} \tag{3}$$

For example, given the alphabet $\Sigma_3 = \{0, 1, 2\}$ and $k = 3$, a k-TD may result in

$$\boldsymbol{x} = 1201210 \to \boldsymbol{x}' = T_{1,3}(\boldsymbol{x}) = 1201\underline{201}210, \tag{4}$$

where the underlined substring 201 is the copy. We refer to $\boldsymbol{x}'$ as a k-TD *descendant* of $\boldsymbol{x}$.

Given a word $\boldsymbol{x} \in \Sigma_q^n$ $n \geq k$, the *k-discrete-derivative* transform [4] is defined as $\phi(\boldsymbol{x}) = (\hat{\phi}(\boldsymbol{x}), \bar{\phi}(\boldsymbol{x}))$, where

$$\hat{\phi}(\boldsymbol{x}) = x_1 \cdots x_k, \bar{\phi}(\boldsymbol{x}) = x_{k+1} \cdots x_n - x_1 \cdots x_{n-k}. \tag{5}$$

where the subtraction is performed entry-wise modulo q. Continuing the example given in (4),

$$\begin{aligned} \boldsymbol{x} = 1201210 &\to \boldsymbol{x}' = 1201\underline{201}210, \\ \phi(\boldsymbol{x}) = 120, 0012 &\to \phi(\boldsymbol{x}') = 120, 0\underline{000}012. \end{aligned} \tag{6}$$

As seen in the example, after the k-TD in $\boldsymbol{x}$, $\bar{\phi}(\boldsymbol{x}')$ can be obtained by inserting 0^k into $\bar{\phi}(\boldsymbol{x})$, immediately after the i-th entry.

Copies generated by tandem duplications may not be always perfect. That is, the copy may not always be exact. Such a duplication is referred to as a *noisy duplication*. In this paper, we limit our attention to noisy duplications in which the copy is at Hamming distance 1 from the original. Continuing example (4), one symbol in the copy 201 may change,

$$\begin{aligned} \boldsymbol{x}' = 1201201210 &\to \boldsymbol{x}'' = 1201\underline{1}01210, \\ \phi(\boldsymbol{x}') = 120, 0000012 &\to \phi(\boldsymbol{x}'') = 120, 0\underline{2}00\underline{1}12. \end{aligned}$$

As seen in the example, a noisy duplication of length k (k-ND) can be regarded as an exact k-TD followed by a substitution. Given a word $\boldsymbol{x} \in \Sigma_q^*$, the tandem duplication results in $\boldsymbol{x}' = T_{i,k}(\boldsymbol{x})$ and the following substitution results in $\boldsymbol{x}'' = T_{i,k}(\boldsymbol{x}) + a\boldsymbol{e}_j$, where $(i+k+1) \leq j \leq (i+2k)$, $a \in \Sigma_q^+$, and $\boldsymbol{e}_j$ represents a unit vector with 1 in the j-th entry and 0 elsewhere. Note that the first k elements are not affected by exact or noisy duplications and $\hat{\phi}(\boldsymbol{x}) = \hat{\phi}(\boldsymbol{x}') = \hat{\phi}(\boldsymbol{x}'')$. Hence, we focus on changes in $\bar{\phi}(\cdot)$. The substitution changes at most two symbols of $\bar{\phi}(\boldsymbol{x}')$ and can be expressed as

$$\bar{\phi}(\boldsymbol{x}'') = \bar{\phi}(\boldsymbol{x}') + a\boldsymbol{\epsilon}_j, \tag{7}$$

where $\boldsymbol{\epsilon}_j = \boldsymbol{e}_{j-k} - \boldsymbol{e}_j$ if $(k+1) \leq j \leq (|\boldsymbol{x}'| - k)$ and $\boldsymbol{\epsilon}_j = \boldsymbol{e}_{j-k}$ if $(|\boldsymbol{x}'| - k + 1) \leq j \leq |\boldsymbol{x}'|$. We refer to $\boldsymbol{x}''$ as a *k-ND descendant of $\boldsymbol{x}$*.

Since noisy duplications may occur at any position, the word $\boldsymbol{x}$ can generate many descendants through noisy duplication errors. Let $D_k^{t(p)}(\boldsymbol{x})$ denote the *descendant cone* of $\boldsymbol{x}$ obtained after t duplications, p of which are noisy, where $t \geq p$. Furthermore, the descendant cone with many exact k-TDs and at most P noisy duplications, i.e., at most P substitution errors, can be expressed as

$$D_k^{*(\leq P)}(\boldsymbol{x}) = \bigcup_{p=0}^{p=P} \bigcup_{t=p}^{\infty} D_k^{t(p)}(\boldsymbol{x}). \tag{8}$$

In this paper, we limit our attention to $P = 1$.

We define a mapping operation $\mu : \Sigma_q^* \to \Sigma_q^*$ by removing all runs of 0^k in $\boldsymbol{z} \in \Sigma_q^*$. More specifically, consider a string $\boldsymbol{z}$ as

$$\boldsymbol{z} = 0^{m_0} w_1 0^{m_1} \cdots w_t 0^{m_{t+1}},$$

where $t = \mathrm{wt}(\boldsymbol{z})$, $w_1, \ldots, w_t \in \Sigma_q^+$, and $m_0, \ldots, m_{t+1}$ are non-negative integers. The mapping $\mu(\boldsymbol{z})$ is defined as

$$\mu(\boldsymbol{z}) = 0^{m_0 \bmod k} w_1 0^{m_1 \bmod k} \cdots w_t 0^{m_{t+1} \bmod k}.$$

Also, $\mathrm{RLL}(m)$ denotes the set of strings of length m containing no 0^k. In other words, $\mathrm{RLL}(m) = \{\boldsymbol{z} \in \Sigma_q^m | \mu(\boldsymbol{z}) = \boldsymbol{z}\}$.

According to [4], given a word $\boldsymbol{x} \in \Sigma_q^*$, after many (even infinite) k-TD errors, the string $(\hat{\phi}(\boldsymbol{x}), \mu(\bar{\phi}(\boldsymbol{x})))$ stays the same. To make use of this property, define the *duplication root* $\mathrm{drt}(\boldsymbol{x})$ as the string obtained from $\boldsymbol{x}$ after all copies of length k are removed. Note that we then have

$$\phi(\mathrm{drt}(\boldsymbol{x})) = (\hat{\phi}(\boldsymbol{x}), \mu(\bar{\phi}(\boldsymbol{x}))). \tag{9}$$

If $\mathrm{drt}(\boldsymbol{x}) = \boldsymbol{x}$, we call the word $\boldsymbol{x}$ irreducible. The set of all irreducible words of length n can be written as $\mathrm{Irr}(n) = \{\boldsymbol{x} \in \Sigma_q^n | \, \mathrm{drt}(\boldsymbol{x}) = \boldsymbol{x}\}$. In other words, an irreducible word $\boldsymbol{x} \in \Sigma_q^n$ satisfies $\bar{\phi}(\boldsymbol{x}) \in \mathrm{RLL}(n-k)$.

For a word $\boldsymbol{z} \in \Sigma_q^*$, we define its *indicator* $\Gamma(\boldsymbol{z}) : \Sigma_q^* \to \Sigma_2^*$ as $\Gamma(\boldsymbol{z}) = \Gamma_1(\boldsymbol{z}) \cdots \Gamma_{|\boldsymbol{z}|}(\boldsymbol{z})$, where

$$\Gamma_i(\boldsymbol{z}) = \begin{cases} 1, & \text{if } z_i \neq 0, \\ 0, & \text{otherwise.} \end{cases} \quad i = 1, \ldots, |\boldsymbol{z}|. \tag{10}$$

Based on (7), the substitution in a noisy duplication alters two symbols in $\bar{\phi}(\boldsymbol{x}')$ at distance k. For the purpose of error correction, it will be helpful to rearrange the symbols into k strings such that the two symbols affected by the substitution appear next to each other in one of the strings. More precisely, for $j \in [k]$, we define a *splitting* operation that extracts entries whose position is equal to j modulo k. That is, for $\boldsymbol{u} \in \Sigma_q^n$ and $j \in [k]$, define $\boldsymbol{u}_j = (\mu_{ji})_i = \mathrm{Sp}_k(\boldsymbol{u}, j)$ such that

$$\mu_{ji} = \mu_{j+(i-1)k}, \quad 1 \leq i \leq \left\lceil \frac{n-j}{k} \right\rceil + 1.$$

For $\boldsymbol{u} \in \Sigma_q^n$, we then define the *interleaving* operation $\mathrm{IL} : \Sigma_q^n \to \Sigma_q^n$ as the concatenation of $\mathrm{Sp}_k(\boldsymbol{u}, j), j \in [k]$,

$$\mathrm{IL}(\boldsymbol{u}) = \mathrm{Sp}_k(\boldsymbol{u}, 1) \cdots \mathrm{Sp}_k(\boldsymbol{u}, k).$$

Example 1. *Given an alphabet $\Sigma_3 = \{0, 1, 2\}$, $k = 3$, and $\boldsymbol{u}' = \bar{\phi}(\boldsymbol{x}') = 221200012$, after splitting $\boldsymbol{u}'$, we obtain*

$$\begin{aligned} \boldsymbol{u}'_1 &= \mathrm{Sp}_3(\boldsymbol{u}', 1) = 220, \\ \boldsymbol{u}'_2 &= \mathrm{Sp}_3(\boldsymbol{u}', 2) = 201, \\ \boldsymbol{u}'_3 &= \mathrm{Sp}_3(\boldsymbol{u}', 3) = 102, \\ \mathrm{IL}(\boldsymbol{u}') &= \boldsymbol{u}'_1 \boldsymbol{u}'_2 \boldsymbol{u}'_3 = 220201102. \end{aligned}$$

Based on (7)*, after one substitution error, we may obtain $\boldsymbol{u}'' = \bar{\phi}(\boldsymbol{x}'') = 221201\underline{0}11\underline{1}$. We then find*

$$\begin{aligned} \boldsymbol{u}''_1 &= \mathrm{Sp}_3(\boldsymbol{u}'', 2) = 201, \\ \boldsymbol{u}''_2 &= \mathrm{Sp}_3(\boldsymbol{u}'', 1) = 220, \end{aligned}$$

$$\begin{aligned}\boldsymbol{u}_3'' &= \mathrm{Sp}_3(\boldsymbol{u}'', 3) = 1\underline{11},\\ \mathrm{IL}(\boldsymbol{u}'') &= \boldsymbol{u}_1''\boldsymbol{u}_2''\boldsymbol{u}_3'' = 220201\underline{11}.\end{aligned}$$

We observe that the error is restricted to $\boldsymbol{u}_3''$ and that the two symbols changed by the substitution error are adjacent in $\mathrm{IL}(\boldsymbol{u}'')$, while they are not so in $\boldsymbol{u}''$.

Given a word $\boldsymbol{z} \in \Sigma_q^n$, we define the *cumulative-sum* operation $\mathrm{CS} : \Sigma_q^n \to \Sigma_q^n$, as $\boldsymbol{r} = \mathrm{CS}(\boldsymbol{z})$, where

$$r_i = \sum_{t=1}^{i} z_t \bmod q, \quad i = 1, \ldots, n. \tag{11}$$

We further define the *odd subsequence* $\mathrm{Od}(\boldsymbol{z})$ and the *even subsequence* $\mathrm{Ev}(\boldsymbol{z})$ of a word $\boldsymbol{z} \in \Sigma_q^*$ as two sequences containing symbols in the odd and even positions, respectively. More precisely, $\mathrm{Od}(\boldsymbol{z}) = \mathrm{Sp}_2(\boldsymbol{z}, 1)$ and $\mathrm{Ev}(\boldsymbol{z}) = \mathrm{Sp}_2(\boldsymbol{z}, 2)$.

Our results will rely on codes that can correct a single insertion or deletion. We thus recall the Varshamov-Tenengolts codes [11], [14], which are binary codes capable of correcting a single insertion or deletion (indel).

Construction 1. *Given integers $m \geq 1$ and $0 \leq \alpha \leq (m-1)$, the binary Varshamov-Tenengolts (VT) code [11] $C_{VT}(\alpha, m)$ is given as*

$$C_{VT}(\alpha, m) = \{\boldsymbol{z} \in \Sigma_2^* | \sum_{i=1}^{|\boldsymbol{z}|} i z_i = \alpha \bmod m\}. \tag{12}$$

Compared to the binary indel-correcting code, correcting indels in non-binary sequences is more challenging. We will use Tenengolts' q-ary single-indel-correcting code [14], which relies on the mapping $\zeta : \Sigma_q^* \to \Sigma_2^*$, where the i-th position of $\zeta(\boldsymbol{z})$ is

$$\zeta_i(\boldsymbol{z}) = \begin{cases} 1, & \text{if } z_i \geq z_{i-1}, \\ 0, & \text{if } z_i < z_{i-1}. \end{cases} \qquad i = 2, 3, \ldots, |\boldsymbol{z}|. \tag{13}$$

with $\zeta_1(\boldsymbol{z}) = 1$.

Construction 2. *Based on Tenengolts' q-ary code [14], given integers $m \geq 1$, $0 \leq \alpha \leq (q-1)$ and $0 \leq \beta \leq (m-1)$, we construct the code $C_{Tq}(\alpha, \beta, m)$ over Σ_q^* as*

$$\begin{aligned} C_{Tq}(\alpha,\beta, m) = \Bigg\{ \boldsymbol{z} \in \Sigma_q^* \Big| & \sum_{j=1}^{|\boldsymbol{z}|} z_j = \alpha \bmod q, \\ & \sum_{i=1}^{|\boldsymbol{z}|} (i-1)\zeta_i(\boldsymbol{z}) = \beta \bmod m \Bigg\}. \end{aligned} \tag{14}$$

III. Noisy duplication channels

To enable designing error-correcting codes, in this section, we study the relation between the input and output sequences in *noisy duplication channels*. As before, we consider channels with many (possibly infinite) exact duplications and at most one noisy duplication in which one of the copied symbols is altered.

If a code $C \in \Sigma_q^n$ corrects many k-TD and one k-ND errors, then for any two distinct codewords $\boldsymbol{c}_1, \boldsymbol{c}_2 \in C$, we have

$$D_k^{*(\leq 1)}(\boldsymbol{c}_1) \cap D_k^{*(\leq 1)}(\boldsymbol{c}_2) = \varnothing. \tag{15}$$

This can be shown to be equivalent to

$$\begin{aligned} \mathrm{drt}(\boldsymbol{c}_2) &\neq \mathrm{drt}(\boldsymbol{c}_1), \\ \mathrm{drt}(D_k^{*(\leq 1)}(\boldsymbol{c}_1)) \cap \mathrm{drt}(D_k^{*(\leq 1)}(\boldsymbol{c}_2)) &= \varnothing. \end{aligned} \tag{16}$$

Since k-TDs do not alter the root of the sequence, $\mathrm{drt}(\boldsymbol{c}_2) \neq \mathrm{drt}(\boldsymbol{c}_1)$ ensures that k-TD errors can be corrected. Noisy tandem duplications however alter the roots. In fact, they may produce sequences with roots whose lengths are different from the roots of the stored sequences. Since the codewords have distinct roots, it suffices to recover the root of the retrieved word to correct any errors. We will restrict our constructions to codes whose codewords are irreducible, and thus are their own roots. While this is not necessary, it will simplify the code construction, as we will show, and does not incur a large penalty in terms of the size of the code.

For noisy duplication channels, given a codeword $\boldsymbol{x} \in \Sigma_q^n$, the generation of descendants $\boldsymbol{x}'' \in D_k^{*(\leq 1)}(\boldsymbol{x})$ includes three different cases: only k-TDs; k-TDs followed by one k-ND; and k-TDs, followed by a k-ND, followed by more k-TDs. Since

Table I

THE CHANGES IN $\boldsymbol{\mu}_j$ AND $\boldsymbol{s}_j$, $j \in [k]$ AS A RESULT OF EXACT AND NOISY DUPLICATIONS, WHEN THE POSITION OF THE SUBSTITUTION IN $\boldsymbol{x}''$ SATISFIES $k < p \leq (|\boldsymbol{x}''| - k)$. HERE $a, b, c \in \Sigma_q$, $d \in \Sigma_2$, $\bar{a} = -a$, AND $a, b \neq 0$. FURTHERMORE, $\Lambda \to \boldsymbol{u}$ AND $\boldsymbol{u} \to \Lambda$ REPRESENT INSERTION AND DELETION OF THE STRING $\boldsymbol{u}$, RESPECTIVELY. ROWS MARKED BY $(*)$ INDICATE THAT THIS TYPE OF ERROR OCCURS FOR AT MOST ONE VALUE OF $j \in [k]$. ROWS MARKED BY $(\$)$, RELATED TO ERROR-CORRECTION CODE, ARE DISCUSSED IN THE NEXT SECTION

$\lvert\boldsymbol{\mu}''\rvert - \lvert\boldsymbol{\mu}\rvert$	$\boldsymbol{\mu} \to \boldsymbol{\mu}''$	$\boldsymbol{\mu}_j \to \boldsymbol{\mu}_j''$	$\boldsymbol{s}_j \to \boldsymbol{s}_j''$
$+2k$	insert $0^{j-1}a0^{k-j}$ and $0^{t-1}(0-a)0^{k-t}$	$\Lambda \to a\bar{a}$ $(*)$ $\Lambda \to 00$ $(\$)$ $c \to 0c0$ $(\$)$	$\Lambda \to 11$ $\Lambda \to 00$ $d \to 0d0$
$+k$	insert $0^{j-1}a0^{k-j}$ and substitute $b_i \to (b_i - a)$	$c \to a(c-a), c \neq a$ $(*)$ $a \to a0$ $(\Lambda \to 0)$ $(\$)$ $\Lambda \to 0$ $(\$)$	$0 \to 11, 1 \to 11$ $1 \to 10$ $(\Lambda \to 0)$ $\Lambda \to 0$
	substitute $0 \to a$ and insert $0^{t-1}(0-a)0^{k-t}$	$0 \to a\bar{a}$ $(*)$ $\Lambda \to 0$ $(\$)$	$0 \to 11$ $\Lambda \to 0$
0	insert $0^{j-1}a0^{k-j}$ and delete $0^{t-1}a0^{k-t}$ with a at the same position	$b0 \to 0b$ $(\$)$ stay same	$10 \to 01$ stay same
	substitute $0 \to a$ and $b_i \to (b_i - a)$ with distance k	$0c \to a(c-a)$ $(*, \$)$ stay same	$00 \to 11, 01 \to 11, 01 \to 10$ stay same
$-k$	substitute $0 \to a$ and delete $0^{t-1}a0^{k-t}$	$0 \to \Lambda$ $(\$)$	$0 \to \Lambda$

Table II

THE CHANGES IN $\boldsymbol{\mu}_j$ AND $\boldsymbol{s}_j$, $j \in [k]$ AS A RESULT OF EXACT AND NOISY DUPLICATION, WHEN THE POSITION OF THE SUBSTITUTION IN $\boldsymbol{x}''$ SATISFIES $(|\boldsymbol{x}''| - k) < p \leq |\boldsymbol{x}''|$. HERE THE NOTATION IS THE SAME AS THAT OF TABLE I

$\lvert\boldsymbol{\mu}''\rvert - \lvert\boldsymbol{\mu}\rvert$	$\boldsymbol{\mu} \to \boldsymbol{\mu}''$	$\boldsymbol{\mu}_j \to \boldsymbol{\mu}_j''$	$\boldsymbol{s}_j \to \boldsymbol{s}_j''$
$+k$	insert $0^{j-1}a0^{k-j}$	$\Lambda \to a$ $(*)$ $\Lambda \to 0$ $(\$)$	$\Lambda \to 1$ $\Lambda \to 0$
0	substitute $0 \to a$	$0 \to a$ $(*, \$)$ stay same	$0 \to 1$ stay same

the root is not affected by the k-TDs, to study $\mathrm{drt}(D_k^{*(\leq 1)}(\boldsymbol{x}))$, we only need to consider the second case, i.e., we focus on descendants $\boldsymbol{x}''$ immediately after the noisy duplication.

Given an irreducible string $\boldsymbol{x} \in \Sigma_q^n$ with $n > 2k$, our goal is to characterize $\mathrm{drt}(D_k^{*(\leq 1)}(\boldsymbol{x}))$. Based on (5), we have

$$\phi(\boldsymbol{x}) = (\hat{\phi}(\boldsymbol{x}), \bar{\phi}(\boldsymbol{x})) = (\boldsymbol{y}, \boldsymbol{z}), \tag{17}$$

where $\boldsymbol{y} = \hat{\phi}(\boldsymbol{x}) \in \Sigma_q^k$ and $\boldsymbol{z} = \bar{\phi}(\boldsymbol{x}) \in \Sigma_q^{n-k}$. Since $\boldsymbol{x}$ is an irreducible string, the string $\boldsymbol{z}$ contains no runs of 0^k, i.e. $\boldsymbol{z} = \mu(\boldsymbol{z})$.

After many k-TDs and one k-ND, we have a descendant $\boldsymbol{x}'' \in D_k^{*(\leq 1)}(\boldsymbol{x})$. Since the substitution only occurs in the copy, the first k symbols always stay the same. Thus $\boldsymbol{x}''$ satisfies

$$\phi(\boldsymbol{x}'') = (\hat{\phi}(\boldsymbol{x}''), \bar{\phi}(\boldsymbol{x}'')) = (\hat{\phi}(\boldsymbol{x}), \bar{\phi}(\boldsymbol{x}'')) = (\boldsymbol{y}, \boldsymbol{z}''). \tag{18}$$

Based on (9), it suffices to study the problem in the transform domain, i.e., we want to obtain all possible $(\boldsymbol{y}, \mu(\boldsymbol{z}''))$ derived from $(\boldsymbol{y}, \mu(\boldsymbol{z}))$. Our code constructions in the next section will also rely on certain sequences derived from $\mu(\boldsymbol{z})$. The next theorem characterizes how these sequences can be altered by k-TDs and one k-ND.

Theorem 1. *Let $\boldsymbol{x} \in \Sigma_q^n$ and let $\boldsymbol{x}'' \in D_k^{*(\leq 1)}(\boldsymbol{x})$ be a descendent of $\boldsymbol{x}$ (produced by passing through the noisy duplication channel). Furthermore, let*

$$\boldsymbol{z} = \bar{\phi}(\boldsymbol{x}), \qquad \boldsymbol{\mu} = \mu(\boldsymbol{z}),$$
$$\boldsymbol{\mu}_j = \mathrm{Sp}_k(\boldsymbol{\mu}, j), \qquad \boldsymbol{s}_j = \Gamma(\boldsymbol{\mu}_j).$$

We define $\boldsymbol{z}'', \boldsymbol{\mu}'', \boldsymbol{\mu}_j'', \boldsymbol{s}_j''$, similarly, based on $\boldsymbol{x}''$. The differences between sequences defined based on $\boldsymbol{x}$ and $\boldsymbol{x}''$ are given in Table I and Table II.

The theorem is proved in the appendix.

Note that the length of $\boldsymbol{\mu}$ can change by $-k$, 0, k, or $2k$. This means that the noisy duplication may manifest as deletions, insertions, or substitutions in $\boldsymbol{\mu}$. Furthermore, the complex error patterns in $\boldsymbol{\mu}$ are simplified when we consider $\boldsymbol{\mu}_j, j \in [k]$. The errors marked by $(*)$ occur for at most one value of j. These correspond to positions affected by the substitution. (Rows marked by $(\$)$ relate to our error-correction strategy and are discussed in the next section.)

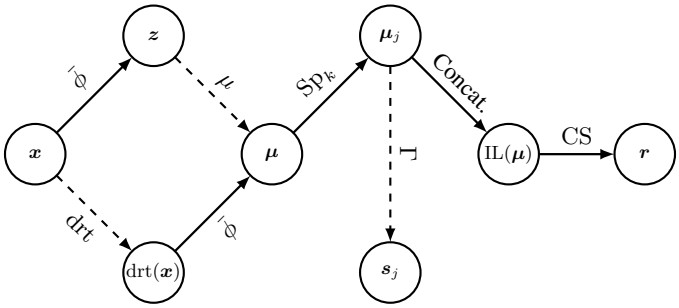

Figure 1. The various mapping used in the paper. "Concat." stands for concatenation. Solid edges indicate invertible mappings, where we have assumed $x_1 \cdots x_k$ is known, since these symbols are not affected by the channel. The mapping μ is generally non-invertible, but in our constructions, since we assume $\boldsymbol{x}$ is irreducible, if we recover $\boldsymbol{\mu} = \mu(\boldsymbol{x})$, we can recover $\boldsymbol{x}$.

Now that we have determined all changes from $(\boldsymbol{y}, \boldsymbol{\mu})$ to $(\boldsymbol{y}, \boldsymbol{\mu}'')$ resulting from passing through the noisy duplication channel, we consider the code design to correct many exact k-TDs and at most one noisy duplication in the next section.

IV. Error-correcting codes for noisy duplication channels

Recall from Section III that we are interested in constructing a code $C \subseteq \mathrm{Irr}(n) \cap \Sigma_q^n$ that can correct many exact k-TDs and at most one noisy duplication. Based on (16), for any code that corrects k-TDs, two distinct codewords must have distinct roots. Thus, for a stored codeword $\boldsymbol{x}$ and the retrieved word $\boldsymbol{x}''$, if we can recover the duplication root $\mathrm{drt}(\boldsymbol{x})$ of $\boldsymbol{x}$ from $\boldsymbol{x}''$, we can recover the codeword $\boldsymbol{x}$. But we have made a further simplifying assumption that $C \subseteq \mathrm{Irr}(n)$ and thus $\boldsymbol{x} = \mathrm{drt}(\boldsymbol{x})$.

As shown in Theorem 1, duplication errors manifest in various ways in $\mathrm{drt}(\boldsymbol{x}'')$ and its counterpart in the μ-transform domain $\mu(\bar{\phi}(\boldsymbol{x}''))$. Hence, for error correction, we utilize several sequences derived from $\boldsymbol{x}$, including $\boldsymbol{\mu}_j$ and $\boldsymbol{s}_j$, $j \in [k]$, as defined in Theorem 1. Furthermore, we define $\boldsymbol{r} = \mathrm{CS}(\mathrm{IL}(\boldsymbol{\mu}))$ and $\boldsymbol{r}'' = \mathrm{CS}(\mathrm{IL}(\boldsymbol{\mu}''))$. We note that $\boldsymbol{r}$ (similarly $\boldsymbol{r}''$) can be directly found by rearranging the elements $x_{k+1} \cdots x_n$.

The relationship between these mappings is illustrated in Figure 1. In the figure, solid edges represent invertible mappings. Since $\boldsymbol{x}$ is irreducible, the stored codeword can be recovered if any of $\boldsymbol{\mu}$, $(\boldsymbol{\mu}_j)_{j\in[k]}$, $\mathrm{IL}(\boldsymbol{\mu})$ or $\boldsymbol{r}$ are recovered (note that $x_1 \cdots x_k$ are not affected by errors). We use these mappings to simplify and correct different error patterns described by Theorem 1 in an efficient manner.

The motivation behind defining $\boldsymbol{\mu}_j$, $j \in [k]$, is to convert insertions and deletions of blocks of length k into simpler errors involving one or two symbols. Some of the errors, marked by $(\$)$ in Tables I and II, involve 0s, which appear in the same positions in $\boldsymbol{s}_j$ and $\boldsymbol{\mu}_j$. Correcting these errors in $\boldsymbol{s}_j$ is more efficient since it will rely on binary codes rather than q-ary codes. We will first correct these errors in $\boldsymbol{s}_j$ and then correct the corresponding $\boldsymbol{\mu}_j$. Finally, the cumulative-sum mapping CS turns errors marked by $(*)$, e.g., $\Lambda \to a\bar{a}$ into a single q-ary insertion or substitution. Importantly, in each case there is only one such error. So if other errors are corrected, we can concatenate $\boldsymbol{\mu}_j$, $j \in [k]$, and then correct the single occurrence of this error.

We will construct an error-correcting code that will allow us to recover $\boldsymbol{\mu}$ from $\boldsymbol{\mu}''$. As discussed, for certain errors occurring in $\boldsymbol{\mu}_j$, specifically those marked by $(\$)$ in Tables I and II, we may do so by correcting errors in $\boldsymbol{s}_j$, via Construction 3 below.

The indicator vectors $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k)$ are subject to several error patterns: insertion of 11; insertion of two 0s with distance at most 2; indel of 1 or 0; swaps of two adjacent elements; and substitution of one or two 0s with one or two 1s. The following code can correct a single occurrence of one of these errors, as shown in the next theorem. A slightly modified version of this code is used for the noisy duplication channel.

Construction 3. *Given integers $0 \le a \le 2(n+1)$, $0 \le b \le 4$, and $0 \le c \le 2n$, we construct the code $C_{(a,b,c)}$ as*

$$C_{(a,b,c)} = \{\boldsymbol{u} \in \Sigma_2^n | \boldsymbol{u} \in C_{VT}(a, 2n+3), \tag{19}$$

$$\sum_{i=1}^{n} u_i = b \bmod 5, \tag{20}$$

$$\sum_{i=1}^{n} i \left(\sum_{j=1}^{j=i} u_j \right) = c \bmod (2n+1)\}, \tag{21}$$

where $n = |\boldsymbol{u}|$.

Theorem 2. *The code $C_{(a,b,c)}$ can correct all error patterns shown in the $\boldsymbol{s}_j$ column of Tables I and II.*

The theorem is proved in the appendix.

Since $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k)$ are weight indicators of $(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k)$, the 0s in $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k)$ and $(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k)$ coincide. However, if a 1 is deleted from a run of 1s in $\boldsymbol{s}_j$, we will not be able to identify which symbol is deleted from $\boldsymbol{\mu}_j$. This means that after recovering $\boldsymbol{s}_j$ from $\boldsymbol{s}_j''$ we can recover $\boldsymbol{\mu}_j$ only in certain cases, specifically, those marked by ($\$$) in Table I and Table II. Interestingly, the errors not corrected by recovering $\boldsymbol{s}_j, j \in [k]$ are marked by ($*$), indicating that they occur only for a single value of j. Hence, to correct these errors, we apply the code constraints to the concatenation of $\boldsymbol{\mu}_j, j \in [k]$, rather than to each $\boldsymbol{\mu}_j$ separately.

Construction 4. *Define* $C_{nd} \subseteq \Sigma_q^n$ *as*

$$C_{nd} = \{\boldsymbol{x} \in \mathrm{Irr}(n) \cap \Sigma_q^n | \boldsymbol{\mu} = \mu(\bar{\phi}(\boldsymbol{x})), \tag{22}$$

$$\boldsymbol{\mu}_j = \mathrm{Sp}_k(\boldsymbol{\mu}, j), \boldsymbol{s}_j = \Gamma(\boldsymbol{\mu}_j), \tag{23}$$

$$\boldsymbol{s}_j \in C_{VT}(a_j, 2|\boldsymbol{s}_j| + 3), \tag{24}$$

$$\sum_{i=1}^{|\boldsymbol{s}_j|} i \left(\sum_{t=1}^{t=i} s_{jt} \right) = c_j \bmod (2|\boldsymbol{s}_j| + 1), \tag{25}$$

$$\sum_{j=1}^{k} \sum_{i=1}^{|\boldsymbol{s}_j|} s_{ji} = b \bmod 5, \tag{26}$$

$$\mathrm{Od}(\mathrm{IL}(\boldsymbol{\mu})) \in C_{Tq}(\bar{a}_1, \bar{b}_1, \lceil \frac{n-k}{2} \rceil), \tag{27}$$

$$\mathrm{Ev}(\mathrm{IL}(\boldsymbol{\mu})) \in C_{Tq}(\bar{a}_2, \bar{b}_2, \lceil \frac{n-k}{2} \rceil), \tag{28}$$

$$\mathrm{CS}(\mathrm{IL}(\boldsymbol{\mu})) \in C_{Tq}(\bar{a}_3, \bar{b}_3, n-k)\}, \tag{29}$$

$$\mathrm{IL}(\boldsymbol{\mu}) \in C_{Tq}(\bar{a}_4, \bar{b}_4, n-k), \tag{30}$$

where $j, a_j, c_j, b, \bar{a}_i, \bar{b}_i$ *are integers satisfying* $j \in [k]$, $0 \le a_j \le 2(|\boldsymbol{s}_j|+1)$, $0 \le c_j \le 2|\boldsymbol{s}_j|$, $0 \le b \le 4$, $0 \le \bar{a}_1, \bar{a}_2, \bar{a}_3, \bar{a}_4 < q$, $0 \le \bar{b}_1, \bar{b}_2 \le \lfloor \frac{n-k}{2} \rfloor$, *and* $0 \le \bar{b}_3, \bar{b}_4 < n-k$.

In Construction 4, the constraints (24), (25), and (26) play the same role as the code in Construction 3, and the constraints (27), (28), and (29) can correct the error patterns of $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ not marked by ($\$$) in Table I and Table II. The constraint (24) corrects one insertion/deletion or two insertions of 0s or 1s in adjacent positions over Σ_2. The constraint (25) corrects one transposition of $\{0, 1\}$ in two adjacent positions. The constraint (26) is a weight-indicating equation for $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$. The constraints (27), (28), (30), and (29) can correct one insertion/deletion in $\mathrm{Od}(\mathrm{IL}(\boldsymbol{\mu}))$, $\mathrm{Ev}(\mathrm{IL}(\boldsymbol{\mu}))$, $\mathrm{IL}(\boldsymbol{\mu})$, and $\boldsymbol{r} = \mathrm{CS}(\mathrm{IL}(\boldsymbol{\mu}))$ over Σ_q, respectively.

Theorem 3. *The error-correcting code* C_{nd} *proposed in Construction 4 can correct infinitely many exact* k*-TD and up to one* k*-ND errors. There exists one such code with size*

$$\frac{|\mathrm{Irr}(n)|}{5q^4 \lceil \frac{n-k}{2} \rceil^2 (4 \lceil \frac{n}{k} \rceil^2 - 1)^k (n-k)^2} \le |C_{nd}| \le |\mathrm{Irr}(n)|. \tag{31}$$

For a code $C \subseteq \Sigma_q^n$, define its rate $R_n(C)$ as $\frac{1}{n} \log_q |C|$. From (31),

$$\frac{1}{n} \log_q |\mathrm{Irr}(n)| - \frac{(2k+4)}{n} \log_q n - \frac{2k}{n} \log_q 2 - \frac{4}{n} - \frac{1}{n} \log_q 5 \le R_n(C_{nd}) \le \frac{1}{n} \log_q |\mathrm{Irr}(n)|.$$

It can then be shown that if $q + k \ge 4$, as $n \to \infty$,

$$\begin{aligned} R_n(C_{nd}) &= \frac{1}{n} \log_q |\mathrm{Irr}(n)| + o(1) \\ &= 1 - \frac{(q-1) \log_q e}{q^{k+2}} + o(1). \end{aligned} \tag{32}$$

Since this is asymptotically the same as the rate of the code correcting only k-TDs [4], the code proposed here is asymptotically optimal. Furthermore, it outperforms the code proposed in [13] for correcting a single unrestricted substitution in addition to correcting many k-TDs.

REFERENCES

[1] Y. M. Chee, J. Chrisnata, H. M. Kiah, and T. T. Nguyen, "Deciding the Confusability of Words under Tandem Repeats," *arXiv:1707.03956 [math]*, Jul. 2017.

[2] L. Dolecek and V. Anantharam, "Repetition error correcting sets: Explicit constructions and prefixing methods," *SIAM Journal on Discrete Mathematics*, vol. 23, no. 4, pp. 2120–2146, 2010.

[3] R. Gabrys, E. Yaakobi, and O. Milenkovic, "Codes in the Damerau distance for deletion and adjacent transposition correction," *IEEE Transactions on Information Theory*, vol. 64, no. 4, pp. 2550–2570, 2017.

[4] S. Jain, F. Farnoud, M. Schwartz, and J. Bruck, "Duplication-correcting codes for data storage in the DNA of living organisms," *IEEE Transactions on Information Theory*, vol. 63, no. 8, pp. 4996–5010, 2017.

[5] ——, "Noise and uncertainty in string-duplication systems," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 3120–3124.

[6] M. Kovacevic and V. Y. Tan, "Asymptotically optimal codes correcting fixed-length duplication errors in DNA storage systems," *IEEE Communications Letters*, vol. 22, no. 11, pp. 2194–2197, 2018.

[7] A. Lenz, A. Wachter-Zeh, and E. Yaakobi, "Duplication-Correcting Codes," *arXiv:1712.09345 [cs, math]*, Dec. 2017.

[8] H. Lou, M. Schwartz, and F. Farnoud, "Evolution of N-gram Frequencies under Duplication and Substitution Mutations," in *IEEE Int. Symp. Information Theory (ISIT)*, Jun. 2018.

[9] D. Pumpernik, B. Oblak, and B. Borštnik, "Replication slippage versus point mutation rates in short tandem repeats of the human genome," *Molecular Genetics and Genomics*, vol. 279, no. 1, pp. 53–61, 2008.

[10] S. L. Shipman, J. Nivala, J. D. Macklis, and G. M. Church, "CRISPR–Cas encoding of a digital movie into the genomes of a population of living bacteria," *Nature*, vol. 547, no. 7663, pp. 345–349, Jul. 2017.

[11] N. J. Sloane, "On single-deletion-correcting codes," *Codes and designs*, vol. 10, pp. 273–291, 2000.

[12] Y. Tang and F. Farzad (Hassanzadeh), "Error-correcting codes for noisy duplication channels," in *57th Annual Allerton Conference on Communication, Control, and Computing*. IEEE, 2019, pp. 1–7.

[13] Y. Tang, Y. Yehezkeally, M. Schwartz, and F. Farnoud, "Single-error detection and correction for duplication and substitution channels," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019.

[14] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion," *IEEE Transactions on Information Theory*, vol. 30, no. 5, pp. 766–769, 1984.

[15] S. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic, "DNA-based storage: Trends and methods," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 1, no. 3, pp. 230–248, 2015.

[16] Y. Yehezkeally and M. Schwartz, "Reconstruction codes for DNA sequences with uniform tandem-duplication errors," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 2535–2539.

APPENDIX A
PROOF OF THEOREM 1

Proof. In a noisy duplication channel with many exact k-TDs and at most one k-ND, given a string $\boldsymbol{x} \in \Sigma_q^*$, let $\phi(\boldsymbol{x}) = (\boldsymbol{y}, \boldsymbol{z})$ with $\boldsymbol{y} = \hat{\phi}(\boldsymbol{x}) \in \Sigma_q^k$ and $\boldsymbol{z} = \bar{\phi}(\boldsymbol{x}) \in \Sigma_q^*$. Since the k-TDs do not change the duplication root $\mathrm{drt}(\boldsymbol{x})$, we focus our attention to the substitution that will change the duplication root. After many exact k-TDs, we obtain $\boldsymbol{x}' \in D_k^{\geq 1(0)}(\boldsymbol{x})$, a descendant of $\boldsymbol{x}$. After the substitution error, we have $\boldsymbol{x}'' \in D_k^{\geq 1(1)}(\boldsymbol{x})$. Since the following k-TD errors do not change the duplication root $\mathrm{drt}(\boldsymbol{x}'')$, we focus on the descendants $\boldsymbol{x}'$ and $\boldsymbol{x}''$.

Let $\phi(\boldsymbol{x}') = (\boldsymbol{y}, \boldsymbol{z}')$ and $\phi(\boldsymbol{x}'') = (\boldsymbol{y}, \boldsymbol{z}'')$. In the transform domain, the string $\boldsymbol{z}'$ can be expressed as

$$\boldsymbol{z}' = \boldsymbol{u} a_1 a_2 \cdots a_i \cdots a_k b_1 b_2 \cdots b_i \cdots b_k \boldsymbol{v}.$$

where $\boldsymbol{u}, \boldsymbol{v} \in \Sigma_q^*$ and $a_i, b_i \in \Sigma_q, i \in [k]$. Let the length of the run of 0s on the left side of a_i be m_1 and on the right side of a_i be m_2 (ending at b_i and excluding a_i, b_i), i.e., the substring $c0^{m_1}a_i0^{m_2}d$ with $a, b \in \Sigma_q^+$. Similarly, we define m_3 and m_4 as the length of the run of 0s on the left side and right side of b_i, starting from a_i and excluding a_i, b_i. Based on (7), if the substitution position p satisfies $k < p \leq (|\boldsymbol{x}'| - k)$, the substitution changes two symbols; if $(|\boldsymbol{x}'| - k) < p \leq |\boldsymbol{x}'|$, the substitution changes one symbol.

First, we consider the substitution position satisfying $k < p \leq (|\boldsymbol{x}'| - k)$ such that two symbols of $\boldsymbol{z}'$ changes. The 2 symbols in $\boldsymbol{z}'$ have a distance of k. After the substitution, we have

$$\boldsymbol{z}'' = \boldsymbol{u} a_1 a_2 \cdots (a_i + a) \cdots a_k b_1 b_2 \cdots (b_i - a) \cdots b_k \boldsymbol{v},$$

where $a \in \Sigma_q^+$. Based on (6), since the substitution only occurs in the copy of a k-TD, we have $a_i = 0$ and $m_1 + m_2 + 1 \geq k$.

Since the length between a_i and b_i is k, we have two cases for m_2 and m_3:

- If $m_2 + m_3 < k$, then $m_2 < (k-1)$ and $m_3 < (k-1)$, which means that the substring between a_i and b_i must contain at least one non-zero symbol.
- If $m_2 + m_3 \geq k$, then $m_2 = m_3 = (k-1)$, which means that the substring between a_i and b_i is 0^{k-1}.

A) Descendants with $m_2 + m_3 < k$: Since the substring between a_i and b_i must contain at least one non-zero symbol, the changes in $\mu(\boldsymbol{z}')$, as well as $\mu(\boldsymbol{z})$, caused by a_i and b_i, can be analyzed independently. If the non-zero symbol is $d \in \Sigma_q^+$, with a_i and b_i on the left and right side respectively, the changes in $\mu(\boldsymbol{z}')$ can be separately studied on the two sides of d. In the following, we use $0^{j-1}a0^{k-j}$ or $0^{t-1}a0^{k-t}$ to denote a substring of length k with $\mathrm{wt}(0^{j-1}a0^{k-j}) = \mathrm{wt}(0^{t-1}a0^{k-t}) = 1$, where $j, t \in [k]$ and $a \in \Sigma_q^+$.

1) The changes on the left side of d is caused by changing a_i. Since $a_i = 0$, then $a = a_i + a \neq 0$.

Table III
THE CHANGES IN $\mu(\boldsymbol{z})$ WITH $m_2+m_3<k$.

a_i and b_i	$\lvert\boldsymbol{\mu}''\rvert-\lvert\boldsymbol{\mu}\rvert$	$\boldsymbol{\mu}\to\boldsymbol{\mu}''$
1a and 2(a)iA	0	insert $0^{j-1}a0^{k-j}$ and delete $0^{t-1}a0^{k-t}$
1a and 2(a)iB	$+k$	insert $0^{j-1}a0^{k-j}$ and $a\to 0$
1a and 2(a)ii	$+k$	insert $0^{j-1}a0^{k-j}$ and $b_i\to(b_i-a)$
1a and 2(b)i	$+2k$	insert $0^{j-1}a0^{k-j}$ and $0^{t-1}(q-a)0^{k-t}$
1a and 2(b)ii	$+k$	insert $0^{j-1}a0^{k-j}$ and $0\to(0-a)$
1b and 2(a)iA	$-k$	$0\to a$ and delete $0^{t-1}a0^{k-t}$
1b and 2(a)iB	0	two substitutions ($0\to a$ and $a\to 0$)
1b and 2(a)ii	0	two substitutions($0\to a$ and $b_i\to(b_i-a)$)
1b and 2(b)i	$+k$	$0\to a$ and insert $0^{t-1}(0-a)0^{k-t}$
1b and 2(b)ii	0	two substitutions($0\to a$ and $0\to(0-a)$)

a) If $\left\lfloor\frac{m_1+m_2+1}{k}\right\rfloor>\left\lfloor\frac{m_1}{k}\right\rfloor$, the length before d increases by k and the substring $0^{j-1}a0^{k-j}$ is inserted in $\mu(\boldsymbol{z}')$, before the symbol d.
b) If $\left\lfloor\frac{m_1+m_2+1}{k}\right\rfloor=\left\lfloor\frac{m_1}{k}\right\rfloor$, the length before d stays the same and 0 is substituted by a at a_i.

2) The changes on the right side of d is caused by changing b_i.
a) If $b_i\neq 0$,
i) if $b_i-a=0$,
A) if $\left\lfloor\frac{m_3+m_4+1}{k}\right\rfloor>\left\lfloor\frac{m_4}{k}\right\rfloor$, the length of $\mu(\boldsymbol{z}')$ after d decreases by k and a substring $0^{t-1}a0^{k-t}$ is deleted from $\mu(\boldsymbol{z}')$.
B) if $\left\lfloor\frac{m_3+m_4+1}{k}\right\rfloor=\left\lfloor\frac{m_4}{k}\right\rfloor$, the length after d stays the same and a is substituted by 0 at b_i.
ii) if $b_i-a\neq 0$, the length after d stays the same and b_i is substituted by (b_i-a).
b) If $b_i=0$, then $b_i-a\neq 0$.
i) if $\left\lfloor\frac{m_3+m_4+1}{k}\right\rfloor>\left\lfloor\frac{m_4}{k}\right\rfloor$, the length of $\mu(\boldsymbol{z}')$ after d increases by k and the substring $0^{t-1}(0-a)0^{k-t}$ is inserted in $\mu(\boldsymbol{z}')$.
ii) if $\left\lfloor\frac{m_3+m_4+1}{k}\right\rfloor=\left\lfloor\frac{m_4}{k}\right\rfloor$, the length after d stays the same and 0 is substituted by $(0-a)$ at b_i.

Since $\boldsymbol{\mu}=\mu(\boldsymbol{z})$ and $\mu(\boldsymbol{z})=\mu(\boldsymbol{z}')$, the changes from $\boldsymbol{\mu}=\mu(\boldsymbol{z}')$ to $\boldsymbol{\mu}''=\mu(\boldsymbol{z}'')$ are shown in Table III classified based on a_i and b_i.

B) Descendants with $m_2+m_3>k$: Based on the analysis above, when $m_2+m_3>k$, the substring between a_i and b_i is 0^{k-1}. Hence $\boldsymbol{z}'$ can be rewritten as

$$\boldsymbol{z}'=\boldsymbol{u}0^{m_1}a_i0^{k-1}b_i0^{m_4}\boldsymbol{v},$$

where $\boldsymbol{u},\boldsymbol{v}\in\Sigma_q^*$. After one substitution, $\boldsymbol{z}''$ can be expressed as

$$\boldsymbol{z}''=\boldsymbol{u}\underline{0^{m_1}(a_i+a)0^{k-1}(b_i-a)0^{m_4}}\boldsymbol{v},$$

where $a_i=0$ and $a\in\Sigma_q^+$. Since the length of $\mu(\boldsymbol{z}')$ is influenced by the *underlined substring* above, we focus on the changes of this segment.

The length of the underlined substring satisfies

$$\left\lfloor\frac{m_1+m_4+k+1}{k}\right\rfloor=\left\lfloor\frac{m_4}{k}\right\rfloor+\left\lfloor\frac{m_1}{k}\right\rfloor+1,$$

or

$$\left\lfloor\frac{m_1+m_4+k+1}{k}\right\rfloor=\left\lfloor\frac{m_4}{k}\right\rfloor+\left\lfloor\frac{m_1}{k}\right\rfloor+2.$$

The two cases are discussed below in detail.

If the length of the underlined substring satisfies $\left\lfloor\frac{m_1+m_4+k+1}{k}\right\rfloor=\left\lfloor\frac{m_4}{k}\right\rfloor+\left\lfloor\frac{m_1}{k}\right\rfloor+1$, then the changes in $\mu(\boldsymbol{z}')$ consist of two cases (based on the change from (a_i,b_i) to (a_i+a,b_i-a)):

1) if $(a_i,b_i)=(0,q_i)$ with $q_i\neq 0$, then we again have two cases:
a) if a_i+a,b_i-a are non-zero, the length of $\mu(\boldsymbol{z}')$ increases by k, and the substring $0^{j-1}a0^{k-j}$ is inserted in $\mu(\boldsymbol{z}')$ and b_i is substituted by b_i-a.
b) if $(a_i+a,b_i-a)=(q_i,0)$, we have $\mu(\boldsymbol{z}'')=\mu(\boldsymbol{z}')$.

2) if $(a_i, b_i) = (0, 0)$, then $a_i + a, b_i - a$ are non-zero, the length of $\mu(\boldsymbol{z}')$ increases by k, and the substring $0^{j-1}a0^{k-j}$ is inserted in $\mu(\boldsymbol{z}')$ and 0 is substituted by $(0 - a)$ at b_i.

Similarly, if the length of the underlined substring satisfies $\left\lfloor \frac{m_1 + m_4 + k + 1}{k} \right\rfloor = \left\lfloor \frac{m_4}{k} \right\rfloor + \left\lfloor \frac{m_1}{k} \right\rfloor + 2$, the changes in $\mu(\boldsymbol{z}')$ also contain two cases:

1) if $(a_i, b_i) = (0, q_i)$, then there are two different cases:
   a) if $a_i + a, b_i - a$ are non-zero, the length of $\mu(\boldsymbol{z}')$ increases by k, and the substring $0^{j-1}a0^{k-j}$ is inserted in $\mu(\boldsymbol{z}')$ and b_i is substituted by $b_i - a$.
   b) if $(a_i + a, b_i - a) = (q_i, 0)$, we have $\mu(\boldsymbol{z}'') = \mu(\boldsymbol{z}')$.
2) if $(a_i, b_i) = (0, 0)$, then $a_i + a, b_i - a$ are non-zero, the length of $\mu(\boldsymbol{z}')$ increases by $2k$, and the string $0^{j-1}a0^{k-j}$ and $0^{t-1}(0 - a)0^{k-t}$ are inserted in $\mu(\boldsymbol{z}')$

Since the k-TDs do not change the duplication root, we have $\mathrm{drt}(\boldsymbol{x}) = \mathrm{drt}(\boldsymbol{x}')$ and $\mu(\boldsymbol{z}) = \mu(\boldsymbol{z}')$. Based on the analysis above, the changes in $\mu(\boldsymbol{z})$ caused by one substitution can be divided into four different cases:

- if $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})| + 2k$, then $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by inserting one $0^{j-1}a0^{k-j}$ and one $0^{t-1}(0 - a)0^{k-t}$. Furthermore, a and $(0 - a)$ have distance k.
- if $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})| + k$, then $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by either inserting $0^{j-1}a0^{k-j}$ and substituting $b_i \to (b_i - a)$ or inserting $0^{t-1}(0 - a)0^{k-t}$ and substituting $0 \to a$. In both cases, a and $(b_i - a)$ have a distance of k.
- if $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})|$, three different cases occur. First, $\mu(\boldsymbol{z}'') = \mu(\boldsymbol{z})$, there are no changes. Second, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by two substitutions ($0 \to a$ and $b_i \to (b_i - a)$ with distance k). Third, the string $0^{j-1}a0^{k-j}$ is inserted and $0^{t-1}a0^{k-t}$ is deleted, where a stays in the same position. In the third case, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by swapping 0^e with a substring (the form of d or $c\Sigma_q^{e-2}d$ with $e \neq 0$ and $c, d \in \Sigma_q^+$) between $a_i = 0$ and $b_i = a$, where the distance of the begining of the two substrings is k. Furthermore, the integer e satisfies $1 \leq e \leq (k - 1)$.
- if $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})| - k$, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by deleting $0^{t-1}a0^{k-t}$ and substituting $0 \to a$.

In conclusion, the changes from $\boldsymbol{\mu} = \mu(\boldsymbol{z})$ to $\boldsymbol{\mu}'' = \mu(\boldsymbol{z}'')$ caused by one substitution are described in the first and second columns of Table I. We now discuss the changes in $\boldsymbol{\mu}_j$, i.e., the difference between $\boldsymbol{\mu}_j$ and $\boldsymbol{\mu}_j''$ for $j \in [k]$. This is done by considering four cases:

- If $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})| + 2k$, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by inserting a $0^{j-1}a0^{k-j}$ and a $0^{t-1}(0 - a)0^{k-t}$. For $j \in [k]$, the length of each $\boldsymbol{\mu}_j$ increases by 2. For one value of j, $a(0 - a)$ is inserted in $\boldsymbol{\mu}_j$ and two 0s are inserted in the other $(k - 1)$ strings with a distance at most 2.
- If $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})| + k$, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by inserting $0^{j-1}a0^{k-j}$ or $0^{t-1}(0 - a)0^{k-t}$ and substituting $(b_i \to (b_i - a))$ or $(0 \to a)$. For $j \in [k]$, the length of $\boldsymbol{\mu}_j$ increases by 1. For one value of j, the insertion and substitution $b_i \to a(b_i - a)$ occur in $\boldsymbol{\mu}_j$ and 0 is inserted into each of the other $(k - 1)$ strings.
- If $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})|$, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ in three different cases. First, $\mu(\boldsymbol{z}'') = \mu(\boldsymbol{z})$, there are no changes. Second, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by substituting two symbols $(0 \to a, b_i \to (b_i - a))$ with distance k. For one value of j, the substitutions $(0b_i \to a(b_i - a))$ occur in $\boldsymbol{\mu}_j$ and the other $(k - 1)$ strings stay the same. Third, $\mu(\boldsymbol{z}'')$ is obtained from $\mu(\boldsymbol{z})$ by inserting $0^{j-1}a0^{k-j}$ and deleting $0^{t-1}(0 - a)0^{k-t}$. For $j \in [k]$, at least one $\boldsymbol{\mu}_j$ swaps $(b0) \to (0b)$ with $b \in \Sigma_q^+$ and the other strings stay the same.
- If $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z}'')| - k$, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by deleting $0^{t-1}a0^{k-t}$ and substituting $0 \to a$. For $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$, one 0 is deleted from each of the k strings.

The changes of $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ can be summarized in the third column of Table I. The forth column is obtained by noting that $\boldsymbol{s}_j = \Gamma(\boldsymbol{\mu}_j), j \in [k]$. This completes the proof of Table I.

Second, we consider the case in which the substitution position p satisfies $(|\boldsymbol{x}'| - k) < p \leq |\boldsymbol{x}'|$, which means that one symbol in $\boldsymbol{z}$ changes. Since one substitution only changes one symbol in $\boldsymbol{z}'$, we have

$$\boldsymbol{z}'' = \boldsymbol{u}a_1a_2 \cdots (a_i + a) \cdots a_k.$$

where $a \in \Sigma_q^+$. Since the substitution only occurs in a tandem duplication copy, we have $a_i = 0$ and $m_1 + m_2 + 1 \geq k$. Note that $a = a_i + a \neq 0$. There are two cases to consider:

1) If $\left\lfloor \frac{m_1 + m_2 + 1}{k} \right\rfloor > \left\lfloor \frac{m_1}{k} \right\rfloor$, then the length of $\mu(\boldsymbol{z}')$ increases by k and the substring $0^{j-1}a0^{k-j}$ is inserted into $\mu(\boldsymbol{z}')$.
2) If $\left\lfloor \frac{m_1 + m_2 + 1}{k} \right\rfloor = \left\lfloor \frac{m_1}{k} \right\rfloor$, then the length of $\mu(\boldsymbol{z}')$ stays the same and 0 is substituted by a at a_i.

We can then find the difference between $\boldsymbol{\mu}_j$ and $\boldsymbol{\mu}_j''$, and $\boldsymbol{s}_j$ and $\boldsymbol{s}_j''$, $j \in [k]$, which are listed in Table II. This completes the proof of Theorem 1.

□

APPENDIX B
THE PROOF OF THEOREM 2

Proof. Given a codeword $\boldsymbol{s} \in C_{(a,b,c)}$, after many exact k-TDs and at most one substitution error, we obtain a descendant $\boldsymbol{s}'' \in \Sigma_2^*$. Since the error-correcting code $C_{(a,b,c)}$ wants to recover $\boldsymbol{s}$ by $\boldsymbol{s}''$, we consider four cases based on the differences between $\boldsymbol{s}$ and $\boldsymbol{s}''$.

1) If $|\boldsymbol{s}''| = |\boldsymbol{s}| - 1$, based on Theorem 1, one 0 is deleted from $\boldsymbol{s}$. Since the VT code [11] can correct one binary insertion or deletion, $\boldsymbol{s}$ can be recovered by inserting one 0 based on (19).
2) If $|\boldsymbol{s}''| = |\boldsymbol{s}|$, $\boldsymbol{s}$ contains three types of changes: one substitution $00 \to 11$, one substitution $0 \to 1$, or one adjacent transposition between 0 and 1. Based on (20), we have $\sum_{i=1}^{n} s_i'' = (b + b'') \bmod 5$, where b'' is helpful to distinguish the three cases. If $b'' = 2$, one substitution $00 \to 11$ between $\boldsymbol{s}$ and $\boldsymbol{s}''$ has occurred. We have $\sum_i i s_i'' = a + 2p + 1 \bmod (2n+3)$, where p is the position of the substitution. Hence, we can recover $\boldsymbol{s}$ by one substitution $11 \to 00$ at the position p of $\boldsymbol{s}''$. If $b'' = 1$, once substitution $0 \to 1$ has occurred. We have $\sum_i i s_i'' = a + p \bmod (2n+3)$. Hence, we can recover $\boldsymbol{s}$ by one substitution $1 \to 0$ at position p of $\boldsymbol{s}''$. If $b'' = 0$, an adjacent transposition has occurred in $\boldsymbol{s}$. If the transposition occurs at p, for the constructed string $\{\boldsymbol{s}^{cs} | s_i^{cs} = \sum_{j=1}^{i} s_j, i \in [|\boldsymbol{s}|]\}$, the string $\boldsymbol{s}^{cs}$ and $\boldsymbol{s}^{cs''}$ only differ at position p with $|s_p^{cs} - s_p^{cs''}| = 1$ [3]. Then we have $\sum_i i \left(\sum_{j=1}^{i} s_j''\right) = c \pm p \bmod (2n+1)$. Thus, we can recover $\boldsymbol{s}$ by swapping the two symbols at positions p and $(p+1)$ of $\boldsymbol{s}''$.
3) If $|\boldsymbol{s}''| = |\boldsymbol{s}| + 1$, based on Theorem 1, $\boldsymbol{s}''$ is derived from $\boldsymbol{s}$ in three ways: inserting a 0, inserting a 1, or inserting a 1 and substituting $0 \to 1$. Based on (19), we have $\sum_i i s_i'' = (a + a'') \bmod (2n+3)$. If $a'' \le \mathrm{wt}(\boldsymbol{s}'')$, one 0 is inserted in $\boldsymbol{s}$, we can recover $\boldsymbol{s}$ by deleting one 0 [11]. If $a'' > \mathrm{wt}(\boldsymbol{s})$, one of the other two cases has occurred in $\boldsymbol{s}$. Based on $\sum_i s_i'' = (b + b'') \bmod 5$, we can distinguish the two cases. If $a'' > \mathrm{wt}(\boldsymbol{s}'')$ and $b'' = 1$, one 1 is inserted in $\boldsymbol{s}$. Then we can recover $\boldsymbol{s}$ by deleting a 1 from $\boldsymbol{s}''$ [11]. If $a'' > \mathrm{wt}(\boldsymbol{s}'')$ and $b'' = 2$, $\boldsymbol{s}''$ is derived from $\boldsymbol{s}$ by inserting one 1 and substituting $0 \to 1$. We have $a'' = 2p + 1 + r_1$, where p denotes the position of the insertion and r_1 represents the number of 1s on the right side of the substituted symbol. During the recovery process, we assume the predicted position and the number of 1s on the right side of the predicted position as p' and r_1', respectively. If $r_1' < r_1$, then $2p' + 1 + r_1' > 2p + 1 + r_1$. If $r_1' > r_1$, then $2p' + 1 + r_1' < 2p + 1 + r_1$. If $r_1' = r_1$, then $2p' + 1 + r_1' = 2p + 1 + r_1$. After obtaining $p = p'$, we can recover $\boldsymbol{s}$ by substituting $11 \to 0$ at position p of $\boldsymbol{s}''$.
4) If $|\boldsymbol{s}''| = |\boldsymbol{s}| + 2$, based on Theorem 1, $\boldsymbol{s}''$ is derived from $\boldsymbol{s}$ in three ways: inserting 11, inserting 00, or inserting two 0s separated by 1. Based on (19), we have $\sum_i i s_i'' = (a + a'') \bmod (2n+3)$. If $a'' > 2\,\mathrm{wt}(\boldsymbol{s}'')$, 11 is inserted in $\boldsymbol{s}$ and $a'' = 2\,\mathrm{wt}(\boldsymbol{s}'') + 2l_0 + 1$, where l_0 denotes the number of 0s at the left side of the inserting position. Then we can recover $\boldsymbol{s}$ by deleting one 11 from $\boldsymbol{s}''$ after l_0 0s from the beginning. If $a'' \le 2\,\mathrm{wt}(\boldsymbol{s}'')$, two 0s are inserted in $\boldsymbol{s}$. If $a'' = 0 \bmod 2$, 00 is inserted in $\boldsymbol{s}$ and $a'' = 2r_1$, where r_1 denotes the number of 1s on the right side of the insertion position. Then we can recover $\boldsymbol{s}$ by deleting 00 from $\boldsymbol{s}''$ before r_1 1s from the end of $\boldsymbol{s}''$. If $a'' = 1 \bmod 2$, two 0s are inserted in $\boldsymbol{s}$ separated by 1 and $a'' = 2r_1 + 1$. Similarly, we can recover $\boldsymbol{s}$ by deleting two 0s before r_1 and $r_1 + 1$ 1s from the end of $\boldsymbol{s}''$.

Based on the analysis, we have proved Theorem 2 that the error-correcting code $C_{(a,b,c)}$ can correct all the error patterns in $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$ caused by many exact k-TDs and at most one substitution error in the noisy duplication channel. □

APPENDIX C
THE PROOF OF THEOREM 3

Proof. To prove Theorem 3, we have to show that the error-correcting code C_{nd} in Construction 4 can correct all error patterns in $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$. Based on Theorem 2, the code $C_{(a,b,c)}$ over Σ_2 can correct all error patterns shown in the $\boldsymbol{\mu}_j$ column of Tables I and II in rows marked by (\$). The constraints (27), (28), and (29) can correct the other error patterns.

Given a codeword $\boldsymbol{x} \in C_{nd} \subseteq \mathrm{Irr}(n) \cap \Sigma_q^n$, we have $\phi(\mathrm{drt}(\boldsymbol{x})) = (\boldsymbol{y}, \boldsymbol{\mu})$ with $\boldsymbol{y} = \hat{\phi}(\boldsymbol{x}) \in \Sigma_q^k$ and $\boldsymbol{\mu} = \mu(\boldsymbol{z}) = \boldsymbol{z} = \bar{\phi}(\boldsymbol{x}) \in \Sigma_q^{n-k}$. After many exact k-TDs and at most one substitution, we obtain a descendant $\boldsymbol{x}'' \in D_k^{*(\le 1)}(\boldsymbol{x})$ with $\phi(\boldsymbol{x}'') = (\boldsymbol{y}, \boldsymbol{z}'')$ and $\boldsymbol{z}'' = \bar{\phi}(\boldsymbol{x}'')$. In the following, we can recover the codeword $(\boldsymbol{y}, \boldsymbol{\mu})$ by correcting four types of error patterns in $(\boldsymbol{y}, \boldsymbol{\mu}'')$, where $\boldsymbol{\mu}'' = \mu(\boldsymbol{z}'')$. Based on the recovered $(\boldsymbol{y}, \boldsymbol{\mu})$, we can obtain the duplication root $\mathrm{drt}(\boldsymbol{x})$ and thus the codeword $\boldsymbol{x}$. The four cases are below:

- If $|\boldsymbol{\mu}''| = |\boldsymbol{\mu}| - k$, then a 0 is deleted from both $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ and $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$. By (24), we recover $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$ by inserting a 0 in each of them. Based on (10), the positions of 0s between $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ and $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$ coincide. We can recover $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ by inserting 0s at the same positions in $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$.
- If $|\boldsymbol{\mu}''| = |\boldsymbol{\mu}|$, $\{\boldsymbol{\mu}_j, j \in [k]\}$ contain two types of errors: transpositions of 0 and b in more than one $\boldsymbol{\mu}_j$, or the substitution either $0c \to a(c-a)$ or $0 \to a$ in one $\boldsymbol{\mu}_j$. By (24), we have

$$\sum_{i=1}^{|\boldsymbol{s}_j''|} i s_{ji}'' = (a_j + a_j'') \bmod (2|\boldsymbol{s}_j| + 3), \quad j \in [k].$$

If $\{a''_j, j \in [k]\}$ contain more than one non-zero integer, both $\{\boldsymbol{\mu}_j, j \in [k]\}$ and $\{\boldsymbol{s}_j, j \in [k]\}$ with non-zero $\{a''_j, j \in [k]\}$ contain one adjacent transposition of $(0, b)$ and $(0, 1)$, respectively. By (21), the transposition positions $\{p_j, j \in [k]\}$ can be obtained. Since both $\{\boldsymbol{\mu}_j, j \in [k]\}$ and $\{\boldsymbol{s}_j, j \in [k]\}$ contain adjacent transpositions at the same positions, we can recover $\{\boldsymbol{\mu}_j, j \in [k]\}$ by swapping two symbols starting at $\{p_j, j \in [k]\}$. If $\{a''_j, j \in [k]\}$ only contain one non-zero integer, say a''_1, three types of errors may occur based on the weight change of $\{\boldsymbol{s}_j, j \in [k]\}$ by (26). Based on the proof of Theorem 2, we can obtain the change position p_1 in $\boldsymbol{\mu}_1$ and $\boldsymbol{s}_1$. If $p_1 < |\boldsymbol{\mu}_1|$, according to Table I, $\boldsymbol{\mu}_1$ contains one substitution $0c \to a(c-a)$, we can recover $\boldsymbol{\mu}_1$ by the substitution $\mu'_{1p_1}\mu'_{1(p_1+1)} \to 0(\mu'_{1p_1} + \mu'_{1(p_1+1)})$. If $p_1 = |\boldsymbol{\mu}_1|$, according to Table II, $\boldsymbol{\mu}_1$ contains one substitution $0 \to a$, we can recover $\boldsymbol{\mu}_1$ by the substitution $\mu'_{1p_1} \to 0$.

- If $|\boldsymbol{\mu}''| = |\boldsymbol{\mu}| + k$, then $(k-1)$ of $\{\boldsymbol{\mu}_1, j \in [k]\}$ contain one insertion $\Lambda \to 0$, and one string, say $\boldsymbol{\mu}_k$, contains either one insertion $\Lambda \to a$ in Table II or one insertion and one substitution $c \to a(c-a)$ in Table I. By (19), the $(k-1)$ strings $\{\boldsymbol{\mu}_1, j \in [k-1]\}$ can be recovered. After that, we generate $\mathrm{IL}'(\boldsymbol{\mu}) = \boldsymbol{\mu}_1 \cdots \boldsymbol{\mu}_{(k-1)}\boldsymbol{\mu}'_k$ by concatenating the k strings. Compared to $\mathrm{IL}(\boldsymbol{\mu})$, $\mathrm{IL}'(\boldsymbol{\mu})$ contains either one insertion $\Lambda \to a$ or one insertion and one substitution $c \to a(c-a)$. Based on (27), (28) and Construction (2), we obtain the change $(\Delta\bar{a}_1, \Delta\bar{a}_2)$. If $\bar{\Delta}\bar{a}_1 + \Delta\bar{a}_2 \neq 0 \bmod q$, then $\mathrm{IL}'(\boldsymbol{\mu})$ contains one insertion $\Lambda \to a$. Then we can recover the insertion $\Lambda \to a$ by (30). If $\Delta\bar{a}_1 + \Delta\bar{a}_2 = 0 \bmod q$, $\mathrm{IL}'(\boldsymbol{\mu})$ contains one insertion and one substitution $c \to a(c-a)$. By (11) and the fact that $a+(c-a) = c$, we construct $r' = \mathrm{CS}(\mathrm{IL}'(\boldsymbol{\mu}))$ with one insertion. Since (29) can correct one insertion in $\mathrm{CS}(\mathrm{IL}'(\boldsymbol{\mu}))$, we can recover $\mathrm{CS}(\mathrm{IL}(\boldsymbol{\mu}))$, $\mathrm{IL}(\boldsymbol{\mu})$, and $\{\boldsymbol{\mu}_j, j \in [k]\}$.
- If $|\boldsymbol{\mu}''| = |\boldsymbol{\mu}| + 2k$, then $(k-1)$ strings of $\{\boldsymbol{\mu}_j, j \in [k]\}$ insert two 0s with distances at most 2, and one string such as $\boldsymbol{\mu}_1$ contains one insertion $a(0-a)$. Similar to the proof of Theorem 2, based on (24), we can recover $\{\boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k\}$ by deleting two 0s. After that, we generate the string $\mathrm{IL}'(\boldsymbol{\mu}) = \boldsymbol{\mu}'_1\boldsymbol{\mu}_2 \cdots \boldsymbol{\mu}_k$. Obviously, the string $\mathrm{IL}'(\boldsymbol{\mu})$ contains one insertion $a(0-a)$. When $\mathrm{IL}(\mu(z))$ is divided into two strings $\mathrm{Od}(\mathrm{IL}(\mu(z)))$ and $\mathrm{Ev}(\mathrm{IL}(\mu(z)))$, one symbol is inserted into each of $\mathrm{Od}(\mathrm{IL}(\mu(z)))$ and $\mathrm{Ev}(\mathrm{IL}(\mu(z)))$ to generate $\mathrm{Od}(\mathrm{IL}'(\mu(z)))$ and $\mathrm{Ev}(\mathrm{IL}'(\mu(z)))$. Since both (27) and (28) can correct an insertion of one symbol in $\mathrm{Od}(\mathrm{IL}(\boldsymbol{\mu}))$ and $\mathrm{Ev}(\mathrm{IL}(\boldsymbol{\mu}))$, respectively, we can recover $\mathrm{IL}(\boldsymbol{\mu})$ and $\{\boldsymbol{\mu}_j, j \in [k]\}$.

Having recovered $\{\boldsymbol{\mu}_j, j \in [k]\}$, we can reconstruct $\boldsymbol{\mu}$, the duplication root $\mathrm{drt}(\boldsymbol{x})$, and the codeword $\boldsymbol{x} \in C_{nd}$. Thus, the error-correcting code C_{nd} can correct all the error patterns caused by many exact k-TD and at most one substitution.

Because the integers $j, a_j, c_j, b, \bar{a}_1, \bar{a}_2, \bar{a}_3, \bar{a}_4, \bar{b}_1, \bar{b}_2, \bar{b}_3, \bar{b}_4$ can be any value in their corresponding ranges, the number of possible codes is $5q^3\lceil\frac{n-k}{2}\rceil^2(2\lceil\frac{n-k}{k}\rceil+3)^k(2\lceil\frac{n-k}{k}\rceil+1)^k(n-k)$. These codes partition the set $\mathrm{Irr}(n)$, so there is at least one code with size

$$|C_{nd}| \geq \frac{|\mathrm{Irr}(n)|}{5q^4\lceil\frac{n-k}{2}\rceil^2(2\lceil\frac{n-k}{k}\rceil+3)^k(2\lceil\frac{n-k}{k}\rceil+1)^k(n-k)^2}.$$

Since $\lceil\frac{n-k}{k}\rceil = \lceil\frac{n}{k}\rceil - 1$, the code size of C_{nd} can be rewritten as

$$|\mathrm{Irr}(n)| \geq |C_{nd}| \geq \frac{|\mathrm{Irr}(n)|}{5q^4\lceil\frac{n-k}{2}\rceil^2(4\lceil\frac{n}{k}\rceil^2-1)^k(n-k)^2}.$$

□