Break-Resilient Codes for Forensic 3D Fingerprinting

Canran Wang[*], **Jin Sima**[†], and **Netanel Raviv**[*]
[*]Department of Computer Science and Engineering, Washington University in St. Louis,
[†]Department of Electrical and Computer Engineering, University of Illinois Urbana Champaign,
canran@wustl.edu, jsima@illinois.edu, netanel.raviv@wustl.edu

2024 IEEE International Symposium on Information Theory (ISIT) | 979-8-3503-8284-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISIT57864.2024.10619135

***Abstract*—3D printing brings about a revolution in consumption and distribution of goods, but poses a significant risk to public safety. Any individual with internet access and a commodity printer can now produce untraceable firearms, keys, and dangerous counterfeit products. To aid government authorities in combating these new security threats, objects are often tagged with identifying information. This information, also known as fingerprints, is written into the object using various bit embedding techniques, such as varying the width of the molten thermoplastic layers. Yet, due to the adversarial nature of the problem, it is important to devise *tamper-resilient* fingerprinting techniques, so that the fingerprint could be extracted even if the object was damaged. This paper focuses on a special type of adversarial tampering, where the adversary breaks the object to at most a certain number of parts. This gives rise to a new adversarial coding problem, which is formulated and investigated herein. We survey the existing technology, present an abstract problem definition, provide lower bounds for the required redundancy, and construct a code which attains it up to asymptotically small factors.**

I. Introduction

Three dimensional (3D) printing has become increasingly accessible in recent years. The prevalence of this technology, however, exposes the public to various security threats, such as production of untraceable firearms, reconstruction of keys from online images, counterfeiting medical equipment or vehicle parts, and more [1]. To aid law enforcement agencies in combating these threats, fingerprinting 3D printers with uniquely traceable data into the object (e.g., timestamp, geolocation, printer ID, etc.) is becoming increasingly important.

Various techniques for fingerprinting 3D printed objects have been proposed in the literature [2]–[11]. Any such bit embedding technique, however, is exposed to a wide range of adversarial tampering threats. Therefore, bit embedding techniques must be coupled with coding techniques which guarantee successful retrieval at the presence of adversarial tampering. Any such adversarial model must be quantified by a corresponding security parameter which restricts the adversary's capabilities, and affects the information rate.

While some adversarial threats to 3D printed information can be well modeled by known coding-theoretic notions such as erasures, substitutions, or deletions, the 3D printing medium introduces new adversarial noise models that have not been studied in the past. Specifically, this paper focuses on an adversarial model in which the information is broken apart maliciously, to at most a certain number of fragments.

We propose a mathematical framework which does not pertain to any particular bit embedding technology, and merely assume that information bits are embedded in some form within the object. The adversary is able to break the object apart in at most t arbitrary positions, where t is a tunable security parameter. The reasoning behind this adversarial model stems from the nature of 3D printing technologies. In the context of this technology, embedded bits are tied to physical elements inside the object, whose size is roughly in the order of magnitude of millimeters, and hence it is possible to break an arbitrarily small number of bits away from the object. Therefore, we do not pose any restriction on the number of bits in the resulting fragments. On the other hand, time constraints, access to instruments, or simply physical strength limitations keep the adversary from breaking the object to arbitrarily many fragments, and hence the parameter t is expected to be small with respect to the codeword length.

The goal of the encoder in our mathematical framework is to recover the original information bits from the (at most) $t+1$ resulting fragments. As commonly assumed in security problems, we assume the adversary is fully aware of the encoding method, and wishes to interfere with the decoding process as much as possible within its capability, which is quantified by the parameter t. Importantly, we do not require the fragments to be put back together, and aim for a complete reconstruction of the information bits based exclusively on the information content of the fragments.

Similar problems have been recently studied in the literature. Several works studied the so-called *sliced-channel* model, in which the information bits are sliced at several evenly-spaced locations, producing a set of substrings of equal size [12]–[14]. The *torn paper coding* problem has been studied by [15]–[17], where the information string is being cut by a probabilistic process, producing substrings of random lengths. More closely related, the adversarial counterpart of torn-paper coding was studied in [18] with a restriction that all fragment lengths are between some upper and lower bounds.

In this paper we extend the above line of works and provide nearly optimal code construction for any number of breaks t. That is, for a given parameter t, we wish to construct a t-break resilient code $\mathcal{C}$ of some length n and minimum redundancy, where redundancy is defined as $n - \log|\mathcal{C}|$. We begin by presenting a simple reduction to traditional (that is, substitution-correcting) codes, which implies that the minimal

redundancy of a t-break code is $\Omega(t \log n)$, and then provide a code construction with $O(t \log n \log \log n)$ redundancy.

A. Preliminaries and Problem Definition

Our setup includes an encoder which holds a binary string $\mathbf{x} \in \{0,1\}^k$ (an information word) for some integer k, that is to be encoded to a string $\mathbf{c} \in \{0,1\}^n$ (a codeword). After encoding $\mathbf{x}$, the codeword $\mathbf{c}$ is embedded in a 3D printed object in an arbitrary fashion. For a security parameters t, an adversary breaks these n bits at arbitrary t locations or less, resulting in a multiset of at most $t+1$ fragments.

These fragments are given to the decoder in an unordered fashion, and the goal of the decoder is to reconstruct $\mathbf{x}$ exactly in all cases. The associated set of codewords in $\{0,1\}^n$ is called *a t-break code*, and is denoted by $\mathcal{C}$. The figure of merit of a given code is its *redundancy*, i.e., the quantity $n - \log|\mathcal{C}|$, where $|\cdot|$ denotes size. Although our context implies that t is a small number relative to n, we choose not refer to it as a constant in our asymptotic analysis in order to better understand the fine dependence of our scheme on it; in essence, our scheme applies to any n and any $t = o(\frac{n}{\log n \log\log n})$. Further, we also assume that the fragments are *oriented* , i.e., the decoder knows the correct orientation of fragments. We use standard notations for string manipulation, such as $\circ$ to denote concatenation, $|\mathbf{x}|$ to denote length, and for a string $\mathbf{x} = (x_1, \ldots, x_n)$ and $1 \le a < b \le n$ we let $\mathbf{x}[a:b] = (x_a, x_{a+1}, \ldots, x_b)$, and $\mathbf{x}[a:] = (x_a, \ldots, x_{|\mathbf{x}|})$, as well as $[n] \triangleq \{1, 2, \ldots, n\}$ for a positive integer n.

Finally, we make use of *mutually uncorrelated* (MU) codes, which is a set of codewords such that the prefix of one codeword does not coincide with the suffix of any (potentially identical) other. MU codes were firstly introduced in [19]–[21], extensively studied by [22]–[27], and have been applied to DNA-based data storage architectures [28]–[30].

For lack of space, proofs and additional details about 3D-printing technology are given in the full version [31] of this paper. Meanwhile, in a parallel work we have implemented a break-resilient coding framework, based on a bit-embedding method of varying layer widths, and successfully tested it in practice using commodity 3D printers [32].

II. Bounds

In this section we provide bounds on the minimum possible redundancy of a t-break code $\mathcal{C}$. We observe that certain constant-weight subcodes of $\mathcal{C}$ must have large Hamming distance, and a bound on the redundancy of $\mathcal{C}$ can be achieved by applying the well-known sphere-packing bound over them.

Theorem 1. *t-break code $\mathcal{C}$ satisfies $n - \log|\mathcal{C}| \ge \Omega(t \log \frac{n}{t})$.*

Note that Theorem 1 implies a minimum of $\Omega(t \log n)$ redundant bits whenever $t = O(n^{1-\epsilon})$ for any constant $\epsilon > 0$.

III. Code Construction

A. Overview

In our code construction, we employ the observation of [33], that *naturally occurring* patterns in a random string can be utilized to align a sequence of received bits against an original reference sequence. Our construction begins by selecting a uniformly random string $\mathbf{z}$. In $\mathbf{z}$, all occurrences of codewords of a mutually-uncorrelated code $\mathcal{C}_{\text{MU}}$ are identified, and each is called *a signature*. The property of MU codes assures that no two signatures overlap.

The pairwise ordering of the signatures are recorded, and a standard *systematic* Reed-Solomon encoding process is applied it to produce redundancy symbols, which are prepended to $\mathbf{z}$, in order to protect said ordering. A decoder that is given at most $t+1$ fragments of a codeword first identifies all discernible (i.e., not broken) occurrences of signatures and redundancy symbols. The correct pairwise ordering information is recovered from the identified signatures and redundancy symbols; it enables to correctly order all fragments which contain at least one signature. For the rest of fragments, additional signatures are defined in an recursive manner and similar mechanism is applied to recover their correct ordering. We now proceed to the details of this code construction.

B. Encoding

Let t be the security parameter, and $m > t$ be an integer; the code length, size, and redundancy will be functions of them, and will be discussed in the sequel. Let $\mathbf{z} \in \{0,1\}^m$ be a uniformly distributed random string, $c \ge 3$ be an integer, and $\mathcal{C}_{\text{MU}} = \{\mathbf{c}_1, \ldots, \mathbf{c}_{|\mathcal{C}_{\text{MU}}|}\}$ be a mutually uncorrelated code of length $c \log m$ and size $|\mathcal{C}_{\text{MU}}|$. As [30] provides an efficient binary MU code of any length n_{MU} and $\lceil \log(n_{\text{MU}}) \rceil + 4$ redundancy bits, we assume that $|\mathcal{C}_{\text{MU}}| \ge \frac{2^{c \log m}}{\beta c \log m}$, where $\beta = 32$.

Definition 1. *For a given string $\mathbf{z} \in \{0,1\}^m$, any occurrence of a codeword of $\mathcal{C}_{MU}$ in $\mathbf{z}$ is called a level-0 signature.*

To construct the code we reserve the $t+1$ lexicographically first codewords $\mathbf{m}_0, \mathbf{m}_1, \ldots, \mathbf{m}_t$ in $\mathcal{C}_{\text{MU}}$, and call them *markers*. Using the definitions of level-0 signatures and markers, we now turn to formulate the criteria by which we reject and resample the random string $\mathbf{z}$.

Definition 2. *A binary string $\mathbf{z} \in \{0,1\}^m$ is called* legit *if it satisfies the following properties.*

- *(I) Every interval of $(2\beta c \log^2 m + c \log m - 1)$ bits in $\mathbf{z}$ contains a level-0 signature.*
- *(II) Every two non-overlapping substrings of length $c \log m$ of $\mathbf{z}$ are distinct.*
- *(III) $\mathbf{z}$ does not contain any of the markers $\mathbf{m}_0, \ldots, \mathbf{m}_t$.*

Property (I) readily implies the following lemma, which is required in the sequel and is easy to prove.

Lemma 1. *Let $\mathbf{z} = \mathbf{z}_1 \circ \mathbf{s}_1 \circ \mathbf{z}_2 \circ \mathbf{s}_2 \circ \cdots \circ \mathbf{s}_{r-1} \circ \mathbf{z}_r$ be legit, where $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_r$ are (potentially empty) intervals of $\mathbf{z}$ separated by the level-0 signatures $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_{r-1}$ (each of length $c \log m$). Then, $|\mathbf{z}_j| < 2\beta c \log^2 m$ for every $j \in [r]$.*

A codeword $\mathbf{c}$ is constructed by feeding a legit string $\mathbf{z}$ to Algorithm 1 . First, the algorithm attaches $\mathbf{m}_0$ to the left hand side of $\mathbf{z}$ (line 1); the resulting string $\mathbf{y}$ is then called the *information region* of the codeword. Note that $\mathbf{m}_0$ becomes

Algorithm 1 Code Construction

Input: A legit binary string $\mathbf{z} \in \{0,1\}^m$. **Output:** A codeword $\mathbf{c} \in \{0,1\}^n$.

1: Let $\mathbf{u}_1, \ldots, \mathbf{u}_t$ be empty strings, and let $\mathbf{y} \leftarrow \mathbf{m}_0 \circ \mathbf{z}$.
2: Let r be the number of level-0 signatures in $\mathbf{y}$, and let $i_1, \ldots, i_r$ be their indices in ascending order.
3: Let SIGNATURES be a key-value store of size r such that $\texttt{SIGNATURES}[i_j] \leftarrow \mathbf{z}[i_j, i_j + c\log m - 1]$ for all $j \in [r]$.
4: Let $\mathbf{A} = [A_{a,b}] \in \mathbb{N}^{|\mathcal{C}_{\text{MU}}| \times |\mathcal{C}_{\text{MU}}|}$ be an all-0 matrix.
5: **for all** keys k in SIGNATURES in ascending order **do**
6: Let k_{next} be the smallest key in SIGNATURES larger than k, or $m+1$ if k is the largest.
7: Let $\mathbf{c}_a \triangleq \texttt{SIGNATURES}[k]$ and $\mathbf{c}_b \triangleq \texttt{SIGNATURES}[k_{\text{next}}]$, where $\mathbf{c}_a, \mathbf{c}_b \in \mathcal{C}_{\text{MU}}$, and let $A_{a,b} \leftarrow k_{\text{next}} - k$.
8: compA $\leftarrow$ COMPRESS-ADJACENCY-MATRIX$(\mathbf{A})$, $\mathbf{d}_1, \ldots, \mathbf{d}_{4t} \leftarrow$ ENCODE$((\texttt{compA}[1], \ldots, \texttt{compA}[|\mathcal{C}_{\text{MU}}|]), 4t)$.
9: **for all** $l \in [t]$ **do** $\mathbf{u}_l \leftarrow \mathbf{d}_{4l-3} \circ \mathbf{d}_{4l-2} \circ \mathbf{d}_{4l-1} \circ \mathbf{d}_{4l}$
10: Let NEWSIGNATURES, RESIDUALS be empty key-value stores.
11: **for** level $\leftarrow 1, \ldots, \log\log m + 6$ **do**
12: **for all** keys k in SIGNATURES in ascending order **do**
13: Let k_{next} be the smallest key in SIGNATURES larger than k, or $m+1$ if k is the largest.
14: **if** $k_{\text{next}} - k \geq 2c\log m$ **then**
15: $u \leftarrow (k + k_{\text{next}})/2$, $\texttt{NEWSIGNATURES}[u] \leftarrow \mathbf{y}[u, u + c\log m - 1]$
16: **else if** $k_{\text{next}} - k > c\log m$ **then** // $k_{\text{next}} - k \leq c\log m$ is impossible due the mutual correlation of $\mathcal{C}_{\text{MU}}$.
17: $v \leftarrow k + c\log m$, $\texttt{RESIDUALS}[v] \leftarrow$ PAD$(\mathbf{y}[v, k_{\text{next}} - 1])$
18: SIGNATURES $\leftarrow$ SIGNATURES $\cup$ NEWSIGNATURES, NEWSIGNATURES $\leftarrow$ EMPTY
19: $\mathbf{r}_1, \ldots, \mathbf{r}_{2t} \leftarrow$ ENCODE$((\texttt{SIGNATURES}[1], \texttt{SIGNATURES}[2], \ldots), 2t)$
20: **for all** $l \in [t]$ **do** $\mathbf{u}_l \leftarrow \mathbf{u}_l \circ \mathbf{r}_{2l-1} \circ \mathbf{r}_{2l}$
21: Let $k_1, k_2, \ldots$ be all keys in RESIDUALS in increasing order.
22: $\mathbf{t}_1 \ldots, \mathbf{t}_{3t} \leftarrow$ ENCODE$((\texttt{RESIDUALS}[k_1], \texttt{RESIDUALS}[k_2], \ldots), 3t)$
23: **for all** $l \in [t]$ **do** $\mathbf{u}_l \leftarrow \mathbf{u}_l \circ \mathbf{t}_{3l-2} \circ \mathbf{t}_{3l-1} \circ \mathbf{t}_{3l}$
24: $\mathbf{c} \leftarrow \mathbf{y}$
25: **for all** $l \in [t]$ **do** // Every $\mathbf{u}_l$ is segmented into binary strings of $c\log m/2$ bits, and $\mathbf{m}_l$ is inserted between every two.
26: $\mathbf{c} \leftarrow \mathbf{m}_l \circ \mathbf{u}_l[1 : c\log m/2] \circ \mathbf{m}_l \circ \mathbf{u}[c\log m/2 + 1 : c\log m] \circ \mathbf{m}_l \circ \cdots \circ \mathbf{c}$
27: **Output** $\mathbf{c}$

the first level-0 signature in $\mathbf{y}$, i.e., $\mathbf{y}[1 : c\log m] = \mathbf{m}_0$ and marks the transition between the information region and the to-be-attached *redundancy region*.

The algorithm continues and stores all level-0 signatures (line 3) in the key-value (KV) store SIGNATURES. It then creates a $|\mathcal{C}_{\text{MU}}| \times |\mathcal{C}_{\text{MU}}|$ matrix $\mathbf{A}$ which records the correct pairwise ordering, and pairwise positional distance, of level-0 signatures. Specifically, every element $A_{a,b}$ counts the number of bits between a pair of adjacent signatures $\mathbf{s}_j = \mathbf{c}_a$ and $\mathbf{s}_{j+1} = \mathbf{c}_b$ in $\mathbf{z}$ for codewords $\mathbf{c}_a, \mathbf{c}_b \in \mathcal{C}_{\text{MU}}$ (line 5–7).

Thanks to Property (II) and Property (III) , every row of $\mathbf{A}$ is either the all-0 vector, or contains exactly one non-zero element in one of the last $|\mathcal{C}_{\text{MU}}| - t - 2$ positions. As a result, there exist less than $|\mathcal{C}_{\text{MU}}|$ possible positions for such a non-zero element in each row of $\mathbf{A}$. Meanwhile, thanks to Lemma 1, the number of different possible values for such a non-zero element (if exists) is less than $2\beta c\log^2 m$.

As such, there exist at most $|\mathcal{C}_{\text{MU}}| \cdot 2\beta c\log^2 m \leq m^c \cdot m^c = m^{2c}$ different possible values for one individual row of $\mathbf{A}$. This fact enables to compress $\mathbf{A}$ into compA, a vector in $\mathbb{F}_{2^{2c\log m}}^{|\mathcal{C}_{\text{MU}}|}$ (line 8). This is done in the function COMPRESS-ADJACENCY-MATRIX. The vector compA is later encoded using a systematic Reed-Solomon code to produce $4t$ parity symbols[1] (line 8). The algorithm then proceeds recursively.

During recursive step $i \geq 1$, the algorithm adds level-ℓ signatures to SIGNATURES, by locating the midpoint between every two existing signatures (line 14–15). If the interval between two consecutive signatures is too short to contain a signature (line 16), it is padded to a string of length $c\log m$, stored in a KV store RESIDUALS (line 17), and referred to as a *residual*. The padding is performed by attaching a 1 and sufficiently many 0's until the padded interval is $c\log m$ bits long ([31, line 2, Alg. 3]). For the sake of encoding and decoding, it is important to note that this padding operation is injective and reversible. The signatures in level-1 to level-ℓ, as stored in SIGNATURES, are encoded similar to the adjacency matrices to produce t redundancy symbols[2] (line 19).

The recursion proceeds until no new signatures can fit between two adjacent existing signatures. Due to Lemma 1, at most $2\beta\log m - 1$ non-overlapping signatures, each of

[1]The encoding is performed over the field of size m^{2c}, and requires at least $|\mathcal{C}_{\text{MU}}| + 6t$ distinct field elements; this is the case since $m^{2c} - |\mathcal{C}_{\text{MU}}| - 4t \geq m^{2c} - m^c - t \geq 0$.

[2]The encoding is performed over the field of size $2^{c\log m} = m^c$. Since there are at most $m/(c\log m)$ residuals/signatures in $\mathbf{z}$, the encoding requires $m/(c\log m) + 2t$ distinct field elements. The encoding is feasible due to the fact that $m^c - m/(c\log m) - 2t \geq m^{c-1} - 2t \geq m^{c-2} - t \geq 0$.

length $c \log m$, can fit in between any two level-0 signatures. Meanwhile, a total of $2^\ell - 1$ signatures reside in such an interval after ℓ recursive steps, since every level-ℓ signature is defined in the middle of intervals separated by signatures from level-0 to level-$(\ell-1)$. Hence, the recursion is guaranteed to terminate after $\log(2\beta \log m) = \log\log m + 6$ steps.

Finally, the residuals are encoded similar to the adjacency matrices, producing $3t$ redundant symbols $\mathbf{t}_1, \ldots, \mathbf{t}_{3t}$ (line 22), and appended to the redundancy strings $\mathbf{u}_1, \ldots, \mathbf{u}_t$. The redundancy string $\mathbf{u}_l$ is of length $(6+2\log\log m)\cdot c \log m$, and undergo an *instrumentation* process. By instrumentation we mean inserting marker $\mathbf{m}_l$ in between every interval of $c \log m/2$ bits of $\mathbf{u}_l$ ([31, Fig. 3]). Each instrumented redundancy string is then $(6+2\log\log m)\cdot 3c \log m$ bits long; they are attached one by one to the left of the information region $\mathbf{y}$ to create the final codeword $\mathbf{c}$ (line 25–26) of length

$$n = |\mathbf{c}| = m + (6 + 2\log\log m) \cdot 3c \log m \cdot t + c \log m.$$

C. Decoding

A procedure for extracting the correct legit string $\mathbf{z}$ from at most $t+1$ fragments of the respective codeword $\mathbf{c}$ is given in Algorithm 2. Recall that a codeword $\mathbf{c}$ consists of an information region and a redundancy region; the former is $\mathbf{m}_0$ followed by a legit string $\mathbf{z}$, and the latter includes t instrumented redundancy string $\mathbf{u}_1, \ldots, \mathbf{u}_t$. Algorithm 2 begins with the attempt to distinguish the transition point, i.e. $\mathbf{m}_0$, from all fragments. If a fragment containing $\mathbf{m}_0$ is found, it is broken at $\mathbf{m}_0$ such that the two segments belong to different regions (line 2). Then, the decoding algorithm sorts the fragments to those that are from the redundancy region (line 3), and those that are from the information region (line 4).

A preliminary analysis of the $\mathbf{z}$-fragments, whose purpose is to extract the surviving level-0 signatures into an approximate adjacency matrix $\mathbf{A}'$, is given in lines 6-10. In this analysis, all codewords of $\mathcal{C}_{\text{MU}}$ present in the fragments are located and identified as level-0 signatures, and the collection of all level-0 signatures is coalesced into a pairwise ordering *approximate* adjacency matrix $\mathbf{A}'$.

The analysis of the redundancy fragments is conducted in lines 12-14, during which all markers $\mathbf{m}_l$, if not broken, are identified. Recall that since $\mathbf{u}_l$ is instrumented with $\mathbf{m}_l$, it can be identified by observing a series of $2\cdot(6+2\log\log m)$ markers $\mathbf{m}_l$ separated by $c \log m/2$ bits. All extracted redundancy strings are placed in a KV-store `R-STRINGS`.

The decoding algorithm proceeds to correct the constructed adjacency matrix $\mathbf{A}'$ to $\mathbf{A}$, i.e., the *correct* redundancy matrix generated in Algorithm 1 from $\mathbf{z}$, using the collected redundancy strings (line 16) and a standard Reed-Solomon decoder. The success of the decoding process is guaranteed as follows.

Theorem 2. *Line 16 outputs the correct adjacency matrix* $\mathbf{A}$.

Having obtained $\mathbf{A}$, the algorithm allocates string $\mathbf{y}' = \mathbf{m}_0 \circ *^m$ to represent the information region of $\mathbf{c}$ (line 18). The $*$'s represent the m unknown values of the original $\mathbf{z}$, which are preceded by $\mathbf{m}_0$. Using the correct pairwise ordering and pairwise positional distance of all level-0 signatures in $\mathbf{A}$, in lines 20–22, the algorithm traverses all adjacent pairs of level-0 signatures and positions all signatures appropriately in $\mathbf{y}'$. In line 25 which follows, the algorithm *affixes* the fragments in `Z-FRAGMENTS` to their correct position in $\mathbf{y}'$; by *correct* we mean that $\mathbf{y}'[i : i + |f| - 1] = \mathbf{f}$ if $\mathbf{y}[i : i + |f| - 1] = \mathbf{f}$, where $\mathbf{y} = \mathbf{m}_0 \circ \mathbf{z}$ (line 1, Alg. 1). Then, all remaining fragments in `Z-FRAGMENTS` contain no level-0 signatures.

In the **while** loop starting at line 26, the decoding algorithm proceeds with the extraction of higher level signatures. This is done by repeatedly traversing all signatures that have already been affixed to $\mathbf{z}$. In traversal ℓ, the algorithm locates the midpoint u between every two adjacent signatures in $\mathbf{y}'$, as long as the gap between them is large enough to fit at least one more signature. Then, the algorithm identifies and collects the $c \log m$ bits which begin at u as a level-ℓ signature. This will result in `UPDATED-SIGS`, a KV store of signatures from level-0 to level ℓ. Note that some entries in `UPDATED-SIGS` may be `empty` when the respective part of $\mathbf{y}'$ contains a $*$ (line 33, Alg. 2). The decoder is guaranteed to correctly affix all fragments in `Z-FRAGMENTS`, stated as follows.

Theorem 3. *The* ***while*** *loop starting at line 26 will eventually terminate, and by then every fragment in* `Z-FRAGMENTS` *is correctly affixed to* $\mathbf{y}'$.

Theorem 3 implies that all $*$'s in $\mathbf{y}'$ now reside in residuals. The next theorem states the correct recovery of residuals.

Theorem 4. *Line 44 outputs a KV store such that* `REPAIRED-RESIDUALS`$[i] = \text{PAD}(\mathbf{r})$ *for every residual* $\mathbf{r} = \mathbf{y}[i : i + |\mathbf{r}|]$.

Finally, the decoder affixes all residuals in `REPAIRED-RESIDUALS` to $\mathbf{z}$, and after which $\mathbf{y}' = \mathbf{y}$. Recall that the marker $\mathbf{m}_0$ is attached to the left of $\mathbf{z}$ by the decoder, and needs to be removed to obtain $\mathbf{y}$. This concludes the proof of correctness of our construction as follows.

Theorem 5. *Let* $\mathbf{z} \in \{0,1\}^m$ *be a legit string, let* $\mathbf{c}$ *be the output of Algorithm 1 with input* $\mathbf{z}$*, and let* $\mathbf{f}_1, \ldots, \mathbf{f}_\ell$ *be fragments of* $\mathbf{c}$ *for some* $\ell \leq t+1$*. Then, Algorithm 2 with input* $\mathbf{f}_1, \ldots, \mathbf{f}_\ell$ *outputs* $\mathbf{z}$.

IV. Redundancy Analysis

We now present an analysis of the redundancy in the encoding process (Section III-B). Inspired by ideas from [33, Theorem 4.4], the next theorem bounds the success probability of choosing a legit binary string $\mathbf{z}$ (Definition 2).

Theorem 6. *A uniform random string* $\mathbf{z} \in \{0,1\}^m$ *is legit (Definition 2) with probability* $1 - 1/\operatorname{poly}(m)$.

With Theorem 6, we can formally provide the redundancy of our scheme in the following corollary.

Corollary 1. *The code has redundancy of* $O(t \log n \log\log n)$.

V. Acknowledgment

This research was supported by the National Science Foundation under Grant CNS-2223032.

Algorithm 2 Decode

Input: A multiset FRAGMENTS of at most $t+1$ (unordered) fragments of a codeword $\mathbf{c} \in \mathcal{C}$.
Output: The binary string $\mathbf{z}$ such that Algorithm 1 with input $\mathbf{z}$ yields $\mathbf{c}$.

if there exists $\mathbf{f} \in$ FRAGMENTS and index i such that $\mathbf{m}_0 = \mathbf{f}[i : i + c\log m - 1]$ **then**
    Remove $\mathbf{f}$, then add $\mathbf{f}[0 : i-1]$ and $\mathbf{f}[i :]$ to FRAGMENTS.
Let R−FRAGMENTS be the set of fragments that contains $\mathbf{m}_l$ for some $l \in [0, t]$ (fragments in the redundancy region).
Let Z−FRAGMENTS be the remaining fragments that are either at least $3c\log m$ bits, or contain at least one discernible level-0 signature, i.e., codeword of $\mathcal{C}_{\text{MU}}$ (fragments in the $\mathbf{z}$ region).
Let $\mathbf{A}' = [A'_{a,b}] \in \mathbb{N}^{|\mathcal{C}_{\text{MU}}| \times |\mathcal{C}_{\text{MU}}|}$ be an all-0 matrix.
for all $\mathbf{f}$ in Z−FRAGMENTS **do**
    for all level-0 signatures with index $i \in [|\mathbf{f}|]$ in $\mathbf{f}$ in ascending order **do**
        if i_{next} exists, defined as the smallest index of a level-0 signature that is greater than i **then**
            Let $\mathbf{c}_a \triangleq \mathbf{f}[i, i + c\log m - 1]$ and $\mathbf{c}_b \triangleq \mathbf{f}[i_{\text{next}}, i_{\text{next}} + c\log m - 1]$, where and $\mathbf{c}_a, \mathbf{c}_b \in \mathcal{C}_{\text{MU}}$.
            $A'_{a,b} \leftarrow i_{\text{next}} - i$.
Let R−STRINGS be an empty KV store.
for all $\mathbf{f}$ in R−FRAGMENTS **do**
    for all $\mathbf{u}'_l$, defined as consecutive length-$c\log m/2$ intervals separated by $2 \cdot (6 + 2\log\log m)$ occurrence of $\mathbf{m}_l$s **do**
        R−STRINGS$[l] \leftarrow \mathbf{u}'_l$.REMOVE$(\mathbf{m}_l)$
approx−compA $\leftarrow$ COMPRESS-ADJACENCY-MATRIX$(\mathbf{A}')$
compA $\leftarrow$ REPAIR-ADJ-MATRIX(approx−compA, R−STRINGS).
$\mathbf{A} \leftarrow$ DECOMPRESS-ADJACENCY-MATRIX(compA)
Let $\mathbf{y}' \leftarrow \mathbf{m}_0 \circ *^m$, cursor $\leftarrow 1$
Let $a \in [|\mathcal{C}_{\text{MU}}|]$ be the index of $\mathbf{m}_0$ in $\mathcal{C}_{\text{MU}}$, and let $b \in [|\mathcal{C}_{\text{MU}}|]$ be the index of the unique nonzero element in row a of $\mathbf{A}$.
while $b \neq$ empty **do**
    cursor $\leftarrow$ cursor $+ A_{a,b}$, $\mathbf{y}'$[cursor : cursor $+ c\log m - 1] \leftarrow \mathbf{c}_b$
    $a \leftarrow b$, and let $b \in [|\mathcal{C}_{\text{MU}}|]$ be index the unique nonzero element in row a of $\mathbf{A}$ if exists, and empty otherwise.
Let SIGNATURES be a KV store such that for all level-0 signatures $\mathbf{s}$ with index i in $\mathbf{y}'$, SIGNATURES$[i] = \mathbf{s}$
Let UPDATED−SIGS be an empty KV store, and let level $\leftarrow 0$
Z−FRAGMENTS, $\mathbf{y}' \leftarrow$ AFFIX-FRAGMENTS(SIGNATURES, Z−FRAGMENTS, $\mathbf{z}$)
while Z−FRAGMENTS is not empty **do**
    level $\leftarrow$ level $+ 1$
    for all keys i in SIGNATURES in ascending order **do**
        Let i_{next} be the smallest key greater than i, or $m+1$ if i is the greatest.
        UPDATED−SIGS$[i] \leftarrow$ SIGNATURES$[i]$
        if $i_{\text{next}} - i \geq 2c\log m$ **then**
            $u \leftarrow (i + i_{\text{next}})/2$
            if $\mathbf{y}'[u, u + c\log m - 1]$ contains $*$ **then** UPDATED−SIGS$[u] \leftarrow$ empty
            else UPDATED−SIGS$[u] \leftarrow (\mathbf{y}'[u, u + c\log m - 1])$
    SIGNATURES $\leftarrow$ REPAIR-SIGNATURES(UPDATED−SIGS, R−STRINGS, level)
    for all keys i in SIGNATURES **do** $\mathbf{y}'[i : i + c\log m - 1] \leftarrow$ SIGNATURES$[i]$
    UPDATED−SIGS $\leftarrow$ empty KV store.
    Z−FRAGMENTS, $\mathbf{y}' \leftarrow$ AFFIX-FRAGMENTS(SIGNATURES, Z−FRAGMENTS, $\mathbf{y}'$)
Let RESIDUALS be an empty KV store.
for all keys i in SIGNATURES in ascending order **do**
    Let i_{next} be the smallest key greater than i, or $m+1$ if i is the greatest.
    if $\mathbf{y}'[i + c\log m : i_{\text{next}} - 1]$ contains $*$ **then** RESIDUALS$[i] \leftarrow$ empty
    else RESIDUALS$[i] \leftarrow$ PAD$(\mathbf{y}'[i + c\log m : i_{\text{next}} - 1])$
REPAIRED−RESIDUALS $\leftarrow$ REPAIR-RESIDUALS(RESIDUALS, R−STRINGS)
for all keys i in REPAIRED−RESIDUALS **do**
    $\mathbf{r} \leftarrow$ DE-PAD(REPAIRED−RESIDUALS$[i]$)
    $\mathbf{y}'[i : i + |\mathbf{r}| - 1] \leftarrow \mathbf{r}$
return $\mathbf{y}'[c\log m + 1 :]$

References

[1] M. Zhou, *3d-printed gun controversy: Everything you need to know*, accessed Sep. 15st, 2023, https://www.cnet.com/news/politics/the-3d-printed-gun-controversy-everything-you-need-to-know/, 2018.

[2] A. Delmotte, K. Tanaka, H. Kubo, T. Funatomi, and Y. Mukaigawa, "Blind watermarking for 3-d printed objects by locally modifying layer thickness," *IEEE Trans. Multimedia*, vol. 22, no. 11, pp. 2780–2791, 2019.

[3] M. Suzuki, P. Dechrueng, S. Techavichian, P. Silapasuphakornwong, H. Torii, and K. Uehira, "Embedding information into objects fabricated with 3-d printers by forming fine cavities inside them," *Electronic Imaging*, vol. 2017, no. 7, pp. 6–9, 2017.

[4] K. A. ElSayed, A. Dachowicz, and J. H. Panchal, "Information embedding in additive manufacturing through printing speed control," in *Proc. Workshop on Additive Manufacturing (3D Printing) Security*, 2021, pp. 31–37.

[5] Z. Li, A. S. Rathore, C. Song, S. Wei, Y. Wang, and W. Xu, "Printracker: Fingerprinting 3d printers using commodity scanners," in *Proc. ACM Conf. Comput. Commun. Security*, 2018, pp. 1306–1323.

[6] J. Voris, B. F. Christen, J. Alted, and D. W. Crawford, *Three dimensional printed objects with embedded identification elements*, US Patent 9,656,428, May 2017.

[7] J. Wee, C. I. Byatte, A. Rhoades, and D. McNeight, *Product authentication*, U.S. Patent App. 14/250,533, 2015.

[8] J. Wee, C. I. Byatte, A. Rhoades, and D. McNeight, *Objets de vertu*, U.S. Patent App. 14/485,880, 2015.

[9] C. Wei, Z. Sun, Y. Huang, and L. Li, "Embedding anti-counterfeiting features in metallic components via multiple material additive manufacturing," *Additive Manufacturing*, vol. 24, pp. 1–12, 2018.

[10] F. Chen, Y. Luo, N. G. Tsoutsos, M. Maniatakos, K. Shahin, and N. Gupta, "Embedding tracking codes in additive manufactured parts for product authentication," *Advanced Engineering Materials*, vol. 21, no. 4, p. 1 800 495, 2019.

[11] C. Harrison, R. Xiao, and S. Hudson, "Acoustic barcodes: Passive, durable and inexpensive notched identification tags," in *Proc. 25th Annual ACM Symp. User interface software and technology*, 2012, pp. 563–568.

[12] J. Sima, N. Raviv, and J. Bruck, "On coding over sliced information," *IEEE Trans. Inf. Theory*, vol. 67, no. 5, pp. 2793–2807, 2021.

[13] J. Sima, N. Raviv, and J. Bruck, "Robust indexing-optimal codes for dna storage," in *Proc. IEEE Int. Symp. Inf. Theory*, IEEE, 2020, pp. 717–722.

[14] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi, "Coding over sets for dna storage," *IEEE Trans. Inf. Theory*, vol. 66, no. 4, pp. 2331–2351, 2019.

[15] I. Shomorony and A. Vahid, "Torn-paper coding," *IEEE Trans. Inf. Theory*, vol. 67, no. 12, pp. 7904–7913, 2021.

[16] I. Shomorony and A. Vahid, "Communicating over the torn-paper channel," in *IEEE Globecom*, 2020, pp. 1–6.

[17] A. N. Ravi, A. Vahid, and I. Shomorony, "Capacity of the torn paper channel with lost pieces," in *Proc. IEEE Int. Symp. Inf. Theory*, IEEE, 2021, pp. 1937–1942.

[18] D. Bar-Lev, S. M. E. Yaakobi, and Y. Yehezkeally, "Adversarial torn-paper codes," *IEEE Trans. Inf. Theory*, 2023.

[19] V. Levenshtein, "Decoding automata invariant with respect to the initial state," *Problems of Cybernetics*, vol. 12, pp. 125–136, 1964.

[20] V. Levenshtein, "Maximum number of words in codes without overlaps," *Problemy Peredachi Informatsii*, vol. 6, no. 4, pp. 355–357, 1970.

[21] E. Gilbert, "Synchronization of binary messages," *IRE Trans. Inf. Theory*, vol. 6, no. 4, pp. 470–477, 1960.

[22] D. Bajic and J. Stojanovic, "Distributed sequences and search process," in *2004 IEEE International Conference on Communications*, vol. 1, 2004, 514–518 Vol.1.

[23] D. Bajic and T. Loncar-Turukalo, "A simple suboptimal construction of cross-bifix-free codes," *Cryptography and Communications*, vol. 6, no. 1, pp. 27–36, 2013.

[24] Y. M. Chee, H. M. Kiah, P. Purkayastha, and C. Wang, "Cross-bifix-free codes within a constant factor of optimality," *IEEE Trans. Inf. Theory*, vol. 59, no. 7, pp. 4668–4674, 2013.

[25] S. Bilotta, E. Pergola, and R. Pinzani, "A new approach to cross-bifix-free sets," *IEEE Trans. Inf. Theory*, vol. 58, no. 6, pp. 4058–4063, 2012.

[26] S. R. Blackburn, "Non-overlapping codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 9, pp. 4890–4894, 2015.

[27] G. Wang and Q. Wang, "Q-ary non-overlapping codes: A generating function approach," *IEEE Trans. Inf. Theory*, vol. 68, no. 8, pp. 5154–5164, 2022.

[28] S. M. H. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, "A rewritable, random-access dna-based storage system," *Nature Scientific Reports*, vol. 5, no. 14138, 2015.

[29] S. M. H. T. Yazdi, R. Gabrys, and O. Milenkovic, "Portable and error-free dna-based data storage," *Nature Scientific Reports*, vol. 7, no. 5011, 2017.

[30] M. Levy and E. Yaakobi, "Mutually uncorrelated codes for dna storage," *IEEE Trans. Inf. Theory*, vol. 65, no. 6, pp. 3671–3691, 2018.

[31] C. Wang, J. Sima, and N. Raviv, "Break-resilient codes for forensic 3d fingerprinting," 2023. arXiv: 2310.03897.

[32] C. Wang, J. Wang, M. Zhou, *et al.*, *Secure information embedding and extraction in forensic 3d fingerprinting*, 2024. arXiv: 2403.04918.

[33] K. Cheng, Z. Jin, X. Li, and K. Wu, "Deterministic document exchange protocols, and almost optimal binary codes for edit errors," in *IEEE 59th Annual Symp. Foundations of Computer Science (FOCS)*, 2018, pp. 200–211.