USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Secure Information Embedding in Forensic 3D Fingerprinting

Canran Wang, Jinwen Wang, Mi Zhou, Vinh Pham, Senyue Hao, Chao Zhou, Ning Zhang, and Netanel Raviv, *Washington University in St. Louis*

https://www.usenix.org/conference/usenixsecurity25/presentation/wang-canran

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

Secure Information Embedding in Forensic 3D Fingerprinting

Canran Wang*, Jinwen Wang*, Mi Zhou, Vinh Pham, Senyue Hao,
Chao Zhou, Ning Zhang, and Netanel Raviv
Washington University in St. Louis

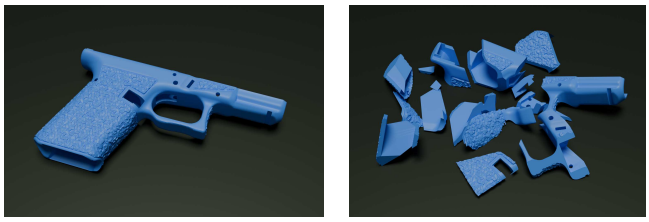
Abstract

Printer fingerprinting techniques have long played a critical role in forensic applications, including the tracking of counterfeiters and the safeguarding of confidential information. The rise of 3D printing technology introduces significant risks to public safety, enabling individuals with internet access and consumer-grade 3D printers to produce untraceable firearms, counterfeit products, and more. This growing threat calls for a better mechanism to track the production of 3D-printed parts.

Inspired by the success of fingerprinting on traditional 2D printers, we introduce SIDE (**S**ecure **I**nformation Embe**D**ding and **E**xtraction), a novel fingerprinting framework tailored for 3D printing. SIDE addresses the adversarial challenges of 3D print forensics by offering both secure information embedding and extraction. First, through novel coding-theoretic techniques, SIDE is both *break-resilient* and *loss-tolerant*, enabling fingerprint recovery even if the adversary breaks the print into fragments and conceals a portion of them. Second, SIDE further leverages Trusted Execution Environments (TEE) to secure the fingerprint embedding process.

1 Introduction

3D printing is revolutionizing the consumption and distribution of goods, but also introduces unprecedented security risks that are absent in traditional 2D printing. With internet access and commercial 3D printers, individuals can fabricate untraceable firearms (*ghost guns* [1], [2]) and other illicit items, with little to no technical expertise. For example, one such weapon was implicated in the recent killing of Brian Thompson, the CEO of UnitedHealthcare [3], [4] (Fig. 1). To assist authorities and law enforcement in addressing these threats, forensic techniques offer a promising path forward.



(a) (b)

Figure 1: (a) Renderings of a Glock 19 frame design. (b) Fingerprinting fails when the frame is broken, with portions missing.

Existing Fingerprinting Solutions: Fingerprinting is a widely employed forensic technique that embeds uniquely traceable data into printed documents, e.g. timestamps, geolocations, and printer IDs. These embedded fingerprints can subsequently be extracted to trace the perpetrator. In the realm of 2D printer fingerprinting, existing approaches generally fall into two broad categories: *active* and *passive* methods. Active methods involve deliberately placing invisible markers (e.g., a grid of dots [5] or traits from modulated laser intensity [6]). Passive methods, on the other hand, rely on inherent variations of individual printers, including imperfections or speed fluctuation patterns in specific printer components [7]. Extending these concepts to 3D printing, current fingerprinting efforts have explored a variety of techniques, e.g., embedding tags on the object surface by varying layer thickness [8] or printing speed [9], altering layer material [10], inserting cavities [11], embedding RFID tags [12] or QR codes [13], [14], and inserting acoustic barcodes using surface bumps [15]. In addition, some studies focus on intrinsic signatures unique to 3D printers, including characteristics of stepper motors [16] and heat systems [17]. Despite the comprehensiveness of vectors investigated in previous studies, there is little focus on the resiliency of the fingerprint against an active adversary, who may tamper with the embedding software or destroy the fingerprint by breaking the printed tools.

Resilient 3D Fingerprinting Solution under Adversaries: To protect the integrity of 3D printing in adversarial scenar-

*These authors contributed equally to this work.

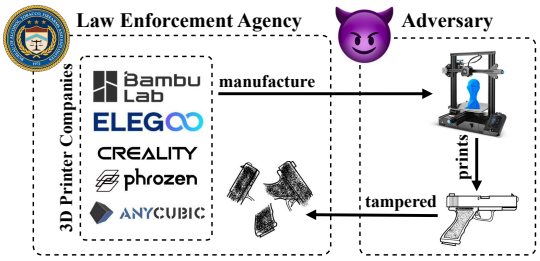


Figure 2: Law Enforcement Agencies Fail Fingerprinting

ios, we proposed SIDE, a framework for **S**ecure **I**nformation Embe**D**ding and **E**xtraction, designed to enhance the resiliency of the fingerprint. This necessitates cyber-physical protection of the fingerprint in both the physical printed part and the digital process that embeds the fingerprint. In this work, SIDE tackles two key technical challenges.

Preventing Information Corruption: An adversary who is fully aware of the embedding method can tamper with the embedded fingerprint by breaking the object and hiding some of the fragments, hindering the extraction of the embedded information. Existing fingerprinting solutions [18], which embed raw information into objects, cannot defend against such adaptive adversaries due to their lack of resilience to missing information. To address this challenge, we propose α-*break-resilient codes* (α-BRC), a family of coding-theoretic techniques designed for information extraction from 3D prints, with α serves as a security parameter. These codes improve robustness by allowing successful extraction of embedded bits under fragmentation and partial loss, i.e., even when the print is adversarially broken and some fragments are missing up to the specified security threshold.

Preventing Fingerprint Embedding Tempering: An adversary may also attempt to tamper with the processing that embeds the fingerprint into the physical part. For example, if the fingerprint is embedded by the slicing software, then an attacker can attempt to remove the fingerprint by tampering with G-code instructions. As a result, to maximize resiliency against digital attacks, SIDE inserts the fingerprint embedding subsystem into the process immediately before physical printing in the 3D printing pipeline. By leveraging a TEE, SIDE ensures the integrity of the embedding process that translates the digital fingerprint to its physical representation, even against a strong attacker who can compromise the OS kernel.

Prototype and Evaluations: We implemented and tested a prototype of SIDE, on a Creality Ender 3 3D FDM printer controlled by a Raspberry Pi 3B board, with OP-TEE V3.4 as secure world OS, and extend the results to an Elegoo Mars 4 SLA printer. To evaluate the efficiency and effectiveness SIDE, we measured both the runtime and memory overhead introduced from SIDE, and the fingerprint extraction success rate. The latter was measured under adversarial scenarios, with a combination of real-world and simulation-based experiments, with the help of a Leica S9D microscope and optical coherence tomography (OCT) devices. In summary, we make the following contributions.

- We propose a secure 3D fingerprinting mechanism named SIDE that improves resiliency to adversarial operations, including malicious manipulation on both the information embedding procedure and the 3D prints.
- We design and implement coding and decoding mechanisms with break-resiliency and loss-tolerance properties to defend against adversaries attempting to tamper with the fingerprinting information on 3D prints. To the best of our knowledge, practical codes with these specific properties have not been studied before. Additionally, we develop a trusted execution environment for the fingerprinting embedding process to protect its integrity against software attacks.
- We implemented a prototype of SIDE on the Creality Ender 3 3D printer controlled by a Raspberry Pi 3B board, and evaluated the efficiency and effectiveness via a combination of real-world and simulation-based experiments on 3D objects with varying shapes and sizes.

2 Background

3D Printing: Additive manufacturing, colloquially referred to as 3D printing, has emerged as a revolutionary technology with profound implications across various industries [19]. In contrast to the traditional *subtractive* manufacturing during which materials are consecutively removed from the workpiece, 3D printing refers to an *additive* process of creating a physical object and is typically done by laying down many thin layers of material in succession.

Numerous technologies have been developed for 3D printing. By and large, they differ by which material is in use and how layers are formed. A layer can either be formed by using a nozzle that deposits molten thermoplastics while shifting back and forth on a surface, e.g. Fused Deposition Modeling (FDM), by depositing a layer of liquid polymers and curing it by ultraviolet light exposure, e.g., Stereolithography (SLA), or by binding powdered material using high-energy laser beams, e.g., Selective Laser Sintering (SLS).

The additive manufacturing process begins by converting a given 3D model into discrete 2D diagrams using a *slicer* software. Each diagram represents a planar cross-section of the model along the printing direction at a certain height. Then, the slicer creates a series of machine commands to instruct the printer about how to produce the corresponding layers of the diagram sequence.

In this research, we focus on printers based on both FDM and SLA technologies for their prevalence in commodity 3D

printers. The commands for FDM printers, called *G-Codes*[1], include the nozzle movement along the x, y, and z axes, the extrusion of material, and the temperature of nozzle/bed. The commands for SLA printers may include the movement of print platform and the exposure time of each layer. Other aspects based on the printer's capabilities may also be specified in these commands.

A typical FDM printer involves four stepper motors, which are actuators that rotate in discrete angular steps of a constant degree. Three of the motors control the nozzle movement in the Cartesian space, and one is responsible for filament extrusion. However, a G-code command specifies only the expected action of the printer hardware in a relatively high level, while the low-level implementation is not addressed. For example, the command

```
G1 X98.302 Y96.831 E15.796 F400
```

merely instructs the printer to move its nozzle from its current position to location $(x,y) = (98.302, 96.831)$ (with its position relative to the z axis unchanged), and simultaneously extrude 15.796 millimeters of molten thermoplastic filament, at a feed rate (speed) of 400 mm/min. Completing this operation requires a series of *stepping events*, and each of them defines the exact timing and direction to trigger one of the four stepper motors for a single angular step.

The printer *firmware* bridges between the G-code and the printer's hardware, translating commands into precise actuator movements that drive the printing process. The translation process of a firmware is non-trivial, and has a significant impact on print quality. For instance, a sudden jerk of the printer nozzle may lead to uneven deposition of print material, compromising or even failing a print. In contrast, a nozzle movement with smooth velocity change is generally preferred.

Trusted Execution Environment: Trusted Execution Environments (TEEs) have emerged as a reliable solution for isolating sensitive and critical operations from untrusted software stacks in computing systems. Utilizing hardware and/or software isolation mechanisms, ARM TrustZone provides a secure execution space, known as *the secure world*, for trusted software stacks. User space applications within this environment are commonly referred to as *Trusted Applications* (TAs). The code and data of secure software in the secure world are protected from access or tampering by untrusted software stacks, which operate in the Rich Execution Environment (REE), also known as *the normal world*. Benefiting from diverse TEE solutions provided by CPU vendors for different architectures, such as ARM TrustZone, Intel Software Guard Extensions (SGX), and RISC-V Keystone, applications across various domains have been effectively protected. These domains include access control [20]–[22], cloud computing [23]–[28], and real-time systems [29]–[33].

[1]Note that there exist other languages for controlling 3D printing hardware, including variants of G-codes and propriety ones. Nevertheless, we collectively refer to those as G-codes for clarity.

3 Threat Model

We assume that the adaptive attacker is fully aware of SIDE's embedding and extraction schemes and attempts to bypass the fingerprinting mechanism through either the cyber vector or the physical vector. Physically, we assume the attacker can kinetically break apart the printed object to corrupt the embedded fingerprint information. Furthermore, the attacker is able to conceal some but not all of the fragments from the forensic investigation. This is since physically eliminating all traces of evidence often requires significantly greater domain expertise and specialized training. Moreover, we do not believe it is possible to forensically link a printed object to a specific printer if all identifiable fragments can be completely concealed, especially given that physical evidence often plays an important role in criminal trials. Digitally, we assume the attacker is capable of modifying the files on the file system, and leveraging exploitation tools to escalate privilege in the system, compromising the rich execution environment (e.g., normal world). However, the attacker cannot forge signatures for secure boot, and the secure software in TEE is free of vulnerability and can be trusted.

We further assume that attackers capable of building their own 3D printing devices, or capable of purchasing untraceable hardware, are out of scope. This is since these approaches often require significant expertise, incur substantial costs, and deteriorate the quality of the resulting print. While SIDE certainly has limitations in defending against resourceful attackers with strong expertise in additive manufacturing, it significantly raises the level of sophistication, prior knowledge, and expertise required from the adversary in order to remain undetected after committing the crime.

We further assume that hardware used by law enforcement during decoding, such as microscopes or computed tomography (CT) devices, is trusted and inaccessible to adversaries. Additionally, we assume that the adversaries will not intentionally damage printer components like sensors and actuators, as doing so would degrade print quality. The printer's processor is assumed to support a Trusted Execution Environment (TEE) and to be reliable. Lastly, side-channel and denial-of-service (DoS) attacks are considered out of scope.

4 Break-Resilient Codes

The secure information extraction feature of SIDE is attributed to the α-*break-resilient codes* (α-BRC) specifically developed for forensic fingerprinting purposes, in which α serves as a security parameter.

The α-BRC includes an encoder which take a binary string $\mathbf{w} \in \{0,1\}^k$, i.e., the fingerprint, as input, where

$$k = (l - \alpha) \cdot m - 1, \tag{1}$$

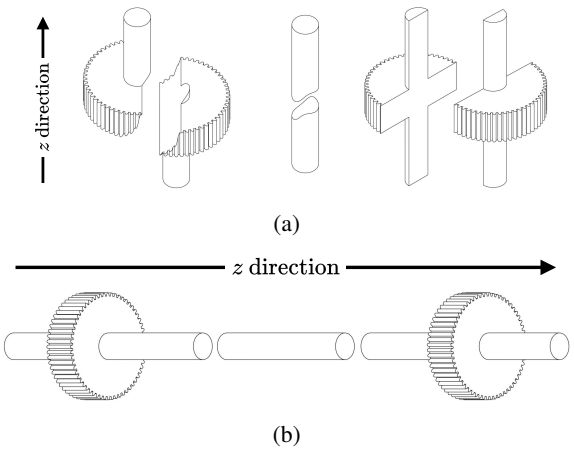


Figure 3: Fragments of a transmission shaft. Breaks in (a) cross multiple layers, and the assembly of fragments may be inferred from their overlapping bits. Breaks in (b) are perpendicular to the z direction, and the correct assembly (i.e., order) cannot be inferred from the fragments themselves.

with l and m being positive integers such that

$$l > \alpha + 1 \text{ and } m \geq \lceil 2 \log l \rceil + 2, \tag{2}$$

and outputs a codeword $\mathbf{c} \in \{0,1\}^n$. For positive integers s and t satisfying

$$4 \cdot t + 2s/(m + \lceil \log m \rceil + 4) \leq 4 \cdot \alpha, \tag{3}$$

the codeword $\mathbf{c}$ is both *t-break-resilient* and *s-loss-tolerant*. Specifically, this means that even if

1. (t-break-resiliency) $\mathbf{c}$ is broken into $t+1$ fragments, and
2. (s-loss-tolerance) some fragments are lost, totaling s bits,

then the α-BRC decoder can still recover $\mathbf{w}$ from the remaining (unordered) subset of fragments, thereby secure information extraction is guaranteed, resolving Challenge 2.

Remark 1. SIDE encodes information into objects using physical elements such as variations in layer thickness. With this setting, every break falls into one of two categories: those that cross multiple layers (Fig.3a) and those that do not (Fig.3b). Breaks in the former category result in overlapping bits (i.e., bits which are shared between two or more fragments), potentially providing information that enables fragment assembly (see Section 4.3). Furthermore, if one fragment which contains shared bits is concealed, the shared bits remain accessible through other overlapping fragments. Breaks in the former category, however, do not create shared bits, and if a fragment in Figure 3b is concealed, all of its embedded bits are omitted from law enforcement.

α-BRC are thus designed to handle the worst-case scenario, where t represents the number of breaks that either produce no overlapping bits or cannot be resolved using overlapping bits, and s denotes the number of codeword bits entirely absent from all fragments confiscated by law enforcement.

4.1 Preliminaries

The following notions from coding theory are employed as basic building blocks for BRC.

Systematic Reed-Solomon (RS) Codes: A special type of Reed-Solomon codes, which have been widely employed in communication systems and data storage applications; for an introduction to the topic see [18, Ch. 5].

For integers k and n such that $n > k > 0$, a systematic $[n,k]$ RS code is a set vectors of length n called codewords, each entry of which is taken from $\mathbb{F}_q$, a finite field with q elements. The first k entries of each codeword in a systematic RS code carry information in raw form, and the remaining $n-k$ contain redundant field elements that are used for error correction. Reed-Solomon codes are *maximum distance separable* (MDS), a property which allows the recovery of a codeword after being corrupted by x errors (incorrect symbols with unknown locations) and y erasures (incorrect symbols with known locations), as long as $n-k \geq 2x+y$. Further, in this paper we focus on the binary field and its extensions, where $q = 2^z$ for some integer $z \geq 1$, and any element in $\mathbb{F}_q$ can be represented by a binary string of length z.

Run-length Limited Codes: A run-length limited (RLL) code has codewords in which the length of runs of repeated bits is bounded. We employ the RLL code from [34, Algorithm 1] in this paper for its simplicity of implementation.

Mutually Uncorrelated Codes: A mutually uncorrelated (MU) code has the property that for every two (possibly identical) codewords, the prefix of one is not identical to the suffix of another. As such, the codewords of a MU code do not overlap with each other when appearing as substrings of a binary string. MU codes have been extensively investigated in the past [34]–[43]. In this paper, we adopt a classic construction of MU code, in which each codeword starts with $\lceil \log k \rceil$ zeros followed by a one, where k is the length of the (binary) information word. The last bit is fixed to one, and the remaining bits are free from zero runs of length $\lceil \log k \rceil + 1$.

Distinct Block Codes. Inspired by ideas from [34, Algorithm 1] and [44, Algorithm 1], we provide an encoding process that maps an input word to an array of distinct binary strings, and offer the inverse operation. These procedures serve as an important component in BRC. Both algorithms, as well as proofs of their correctness, are given in Appendix A.

4.2 Encoding

The encoding procedure of α-BRC takes a binary string $\mathbf{w}$ as input and outputs a codeword, and it is provided in Algorithm 1. At high level, $\mathbf{w}$ will be converted to a sequence

Algorithm 1 ENCODE (α-BRC Encoding)

Input: An information word $\mathbf{w} \in \{0,1\}^k$, where $k = (l-\alpha)\cdot m - 1$, and l, m are positive integers s. t. $m \geq \lceil 2\log l \rceil + 2$.
Output: A codeword $\mathbf{c} \in \{0,1\}^n$, where $n = l\cdot(m + \lceil \log m \rceil + 4) + \alpha\cdot(4m+11)$.

1: Let $\mathbf{u} \leftarrow \mathbf{p}^{(0)} \circ \ldots \circ \mathbf{p}^{(\alpha-1)} \circ \mathbf{w}$, where $\mathbf{p}^{(i)}$ is the binary representation of $i \in [0, \alpha-1]$ using m bits.
2: Let $\texttt{dStrings} \leftarrow (\mathbf{u}_1, \ldots, \mathbf{u}_l) = \text{D-ENCODE}(\mathbf{u})$.
3: Let $\texttt{next}$ be a key-value store with keys and values being elements in $\{0,1\}^m$.
4: **for all** keys $\mathbf{s}$ in $\texttt{next}$ in ascending order **do**
5: **if** there exist $i \in [0, l-2]$ such that $\mathbf{s} = \mathbf{u}_i$ **then** $\texttt{next}[\mathbf{s}] \leftarrow \mathbf{u}_{i+1}$ **else** $\texttt{next}[\mathbf{s}] \leftarrow \mathbf{s}$
6: $\mathbf{r}_1, \ldots, \mathbf{r}_{4\alpha} \leftarrow \text{RS-ENCODE}((\texttt{next}[\mathbf{p}^{(0)}] \circ 0, \texttt{next}[\mathbf{p}^{(1)}] \circ 0 \ldots, \texttt{next}[\mathbf{p}^{(2^{m-1})}] \circ 0), 4\alpha)$,
7: **for** $i \in [0, \alpha-1]$ **do** $\mathbf{d}_i \leftarrow \mathbf{r}_{4i} \circ \mathbf{r}_{4i+1} \circ \mathbf{r}_{4i+2} \circ \mathbf{r}_{4i+3}$
8: $\mathbf{c} = \text{MU}(\mathbf{u}_0) \circ \text{RLL}(\mathbf{d}_0) \circ \ldots \circ \text{MU}(\mathbf{u}_{a-1}) \circ \text{RLL}(\mathbf{d}_{\alpha-1}) \circ \text{MU}(\mathbf{u}_a) \circ \ldots \circ \text{MU}(\mathbf{u}_{l-1})$
9: **return** $\mathbf{c}$

of *distinct* binary strings with the method introduced in Appendix A, and their order will be recorded and then protected using a systematic Reed-Solomon code. Synchronization issues among the symbols of the RS code will be resolved using the RLL and MU techniques mentioned earlier.

Let the information word be $\mathbf{w} \in \{0,1\}^k$, and let $\mathbf{p}^{(i)} \in \{0,1\}^m$ be the binary representation of integer i. The encoding of $\mathbf{w}$ begins by prepending $\mathbf{w}$ with $\mathbf{p}^{(0)}, \mathbf{p}^{(1)}, \ldots, \mathbf{p}^{(\alpha-1)}$, and as shown in line 1, resulting in

$$\mathbf{u} = \mathbf{p}^{(0)} \circ \ldots \circ \mathbf{p}^{(\alpha-1)} \circ \mathbf{w} \in \{0,1\}^{\alpha m + k} \overset{(1)}{=} \{0,1\}^{l\cdot m - 1}. \quad (4)$$

Then, as shown in line 2, the resulting string $\mathbf{u}$ is fed into the function D-ENCODE (Alg. 3, Appendix A) and mapped to an array of l *pairwise-distinct* binary strings of length m, i.e.,

$$\texttt{dStrings} = (\mathbf{u}_0, \ldots, \mathbf{u}_{l-1}), \quad (5)$$

where $\mathbf{u}_i \neq \mathbf{u}_j$ for all distinct i and j in $\{0, 1, \ldots, l-1\}$.

Note that due to the implementation of D-ENCODE, the first α elements of $\texttt{dStrings}$ remain intact, i.e., $\mathbf{u}_i = \mathbf{p}^{(0)}$ for all $i < a$, and they are referred as *markers*. In the next step, a key-value store $\texttt{next}$ is defined to represent the *ordering* of elements in $\texttt{dStrings}$ as follows (line 4–5). For every key $\mathbf{s} \in \{0,1\}^m$, the value $\texttt{next}[\mathbf{s}]$ is defined as

$$\texttt{next}[\mathbf{s}] = \begin{cases} \mathbf{u}_{i+1} & \text{if } \mathbf{s} = \mathbf{u}_i \text{ for } i \in [0, l-2], \\ \mathbf{s} & \text{otherwise.} \end{cases}$$

Note that the value $\texttt{next}[\mathbf{s}]$ is well defined, since $\mathbf{u}_i \neq \mathbf{u}_j$ for every $i \neq j$ by the pairwise-distinct property of $\texttt{dStrings}$. It is also worth noting that the mapping from $\texttt{dStrings}$ to $\texttt{next}$ is injective; one may recover $\texttt{dStrings}$ from $\texttt{next}$ by observing every key $\mathbf{r}$ such that $\texttt{next}(\mathbf{r}) \neq \mathbf{r}$, and connecting every two $\mathbf{r}_a, \mathbf{r}_b$ of them if $\texttt{next}(\mathbf{r}_a) = \mathbf{r}_b$.

We proceed to the treatment of $\texttt{next}$. Since the values in $\texttt{next}$ are binary strings of length m, we append each of them with a 0, and hence they can be regarded as symbols in the finite field $\mathbb{F}_{2^{m+1}}$. They are sorted by their corresponding keys and fed into a systematic Reed-Solomon encoder, which then generates 4α redundancy strings $\mathbf{r}_1, \ldots, \mathbf{r}_{4\alpha} \in \{0,1\}^{m+1}$ (line 6). Note that such encoding is feasible since the codeword length $2^m + 4\alpha$ is smaller than the number of elements in $\mathbb{F}_{2^{m+1}}$[2].

The codeword $\mathbf{c}$ consists of two parts. The first region is called the *information region*, as it is generated from $\mathbf{u}_\alpha, \ldots, \mathbf{u}_{l-1}$, which directly originate from the information word $\mathbf{w}$. The second region is called the *redundancy region*. As the name suggests, it is made from the redundant bits generated from $\texttt{next}$.

Define an encoding function MU which maps $\mathbf{u}_i \in \{0,1\}^m$ to a codeword $\text{MU}(\mathbf{u}_i) \in \{0,1\}^{m+\lceil \log m \rceil + 4}$ of a mutually-uncorrelated code $\mathcal{C}_{\text{MU}}$. The information region is hereby defined as

$$\text{MU}(\mathbf{u}_\alpha) \circ \ldots \circ \text{MU}(\mathbf{u}_{l-1}) \in \{0,1\}^{(l-\alpha)\ldots(m+\lceil \log m \rceil + 4)}.$$

In addition, for $i \in [0, \alpha-1]$, define

$$\mathbf{d}_i = \mathbf{r}_{4i} \circ \mathbf{r}_{4\alpha+1} \circ \mathbf{r}_{4i+2} \circ \mathbf{r}_{4i+3} \in \{0,1\}^{4m+4}$$

as the concatenation of four redundancy strings.

Then, let RLL be an encoding function that maps $\mathbf{d}_i$ to a binary sequence $\text{RLL}(\mathbf{d}_i) \in \{0,1\}^{4m+11}$, called *redundancy packet*, which is free of zero runs longer than $\lceil \log m \rceil + 1$. The redundancy region is then defined as

$$\text{MU}(\mathbf{u}_0) \circ \text{RLL}(\mathbf{d}_0) \circ \text{MU}(\mathbf{u}_{\alpha-1}) \circ \text{RLL}(\mathbf{d}_{\alpha-1}) \in \{0,1\}^{\alpha\ldots(5m+\lceil \log m \rceil + 15)}.$$

Finally, the codeword $\mathbf{c}$ is the two regions combined (line 8):

$$\mathbf{c} = \text{MU}(\mathbf{u}_0) \circ \text{RLL}(\mathbf{d}_0) \circ \ldots \circ \text{MU}(\mathbf{u}_{\alpha-1}) \circ \text{RLL}(\mathbf{d}_{\alpha-1}) \circ \text{MU}(\mathbf{u}_\alpha) \circ \ldots \circ \text{MU}(\mathbf{u}_{l-1}) \in \{0,1\}^{l(m+\lceil \log m \rceil + 4) + \alpha(4m+11)}.$$

[2] RS codes requires the finite field size to be greater or equal to the length of the codeword, i.e., $2^{m+1} \geq 2^m + 4\alpha$; this is the case due to (2) and the fact that $\alpha \geq 1$.

Algorithm 2 DECODE (α-BRC Decoding)

Input: A multiset FRAGMENTS of unordered and partially missing fragments of a codeword $\mathbf{c} \in \mathcal{C}$.
Output: The information word $\mathbf{w}$ such that Algorithm 1 with input $\mathbf{w}$ yields $\mathbf{c}$.

1: Let $\mathbf{r}_i \leftarrow$ erasure for all $i \in [0, 4\alpha - 1]$.
2: **for all** codeword $\mathrm{MU}(\mathbf{u}_i) \in \mathcal{C}_{\mathrm{MU}}$ in the fragments in FRAGMENTS and $\mathbf{u}_i = \mathbf{p}^{(i)}$ for some integer $i < \alpha$ **do**
3: **if** the number of bits after $\mathbf{m}_i$ is less than $4m + 11$ **then** continue to next MU codeword.
4: Let $\mathbf{m}_i$ be the $4m + 11$ bits after $\mathrm{MU}(\mathbf{u}_i)$, and $\mathbf{d}_i \leftarrow \text{DE-RLL}(\mathbf{m}_i)$.
5: $\mathbf{r}_{4i}, \mathbf{r}_{4i+1}, \mathbf{r}_{4i+2}, \mathbf{r}_{4i+3} \leftarrow \mathbf{d}_i[0, m], \mathbf{d}_i[m+1, 2m+1], \mathbf{d}_i[2m+2, 3m+2], \mathbf{d}_i[3m+3, 4m+3]$
6: Let approxNext be a key-value store such that $\texttt{approxNext}[\mathbf{s}] = \mathbf{s}$ for all $\mathbf{s} \in \{0,1\}^m$.
7: **for all** fragment $\mathbf{f} = \mathbf{f}_{\text{start}} \circ \mathrm{MU}(\mathbf{u}_u) \circ \ldots \circ \mathrm{MU}(\mathbf{u}_{u+v}) \circ \mathbf{f}_{\text{end}} \in$ FRAGMENTS where $u \geq \alpha$ **do**
8: **for all** $c \in [0, v-1]$ **do** $\texttt{approxNext}[\mathbf{u}_{u+c}] = \mathbf{u}_{u+c+1}$.
9: next $\leftarrow \text{RS-DECODE}(\texttt{approxNext}, \mathbf{r}_1, \ldots, \mathbf{r}_{4\alpha})$
10: Let $\texttt{dStrings} = (\mathbf{u}_1, \ldots, \mathbf{u}_l)$ be an array of such that $\texttt{next}(\mathbf{u}_i) = \mathbf{u}_{i+1}$.
11: $\mathbf{u} \leftarrow \text{D-DECODE}(\texttt{dStrings})$
12: **return** $\mathbf{u}[\alpha \cdot m :]$

4.3 Preprocessing

Recall that breaks may cross multiple layers, resulting in overlaps between bit strings extracted from confiscated fragments. If a MU codeword is found in two bit strings, the two strings can be merged into one due to the uniqueness of MU codewords (i.e., a MU codeword appears at most once in the BRC codeword). This uniqueness arises from the pairwise distinct property of the elements in dStrings, as defined in (5).

Hence, prior to BRC decoding, the bit strings from confiscated fragments undergo a *preprocessing* stage in which strings that share a MU codeword are merged.

4.4 Decoding

Algorithm 2 provides a procedure for extracting the information word $\mathbf{w}$ from the *unordered* and *partially missing* fragments of the respective codeword $\mathbf{c}$. The crux of this procedure is to reconstruct the key-value store next defined previously, and recover the information word $\mathbf{w}$ from it.

Specifically, the decoding algorithm creates a key-value store approxNext, which is slightly different from next, using the information which appears in the confiscated fragments. Alongside the correctly identified redundancy strings, approxNext goes through a Reed-Solomon decoding process and is corrected to next. Having the correct next in hand, the correct dStrings (5) can be found since the mapping from the latter to the former is injective. Then, dStrings is fed into D-DECODE (Alg. 4, Appendix A), which is the inverse process of D-ENCODE (Alg. 3) to produce $\mathbf{u}$ (4), whose suffix is the information word $\mathbf{w}$.

In more detail, the decoding starts by distinguishing and decoding the discernible codewords of $\mathcal{C}_{\mathrm{MU}}$ from the fragments. Let $\mathrm{MU}(\mathbf{u}_i)$ be a discernible codeword in $\mathcal{C}_{\mathrm{MU}}$ which fully resides within one fragment, where $\mathbf{u}_i$ is its respective decoding. If $\mathbf{u}_i = \mathbf{p}^{(i)}$ for some integer $i < a$, it means that $\mathbf{u}_i$ is a marker, and hence the $(4m + 11)$ bits after it consist of a redundancy packet (line 2). This redundancy packet, if residing in the fragment, is passed to an RLL-decoder which yields four redundancy strings $\mathbf{r}_{4i}, \mathbf{r}_{4i+1}, \mathbf{r}_{4i+2}, \mathbf{r}_{4i+3}$ (line 4–5).

The other discernible codewords of $\mathcal{C}_{\mathrm{MU}}$, i.e., those encoded from non-markers ($\mathbf{u}_i$'s for $i \geq a$), are used to construct a key-value store approxNext. Initially, $\texttt{approxNext}(\mathbf{s}) = \mathbf{s}$ for every key $\mathbf{s} \in \{0,1\}^m$ (line 6). For each fragment $\mathbf{f}$, let

$$\mathbf{f} = \mathbf{f}_{\text{start}} \circ \mathrm{MU}(\mathbf{u}_u) \circ \ldots \circ \mathrm{MU}(\mathbf{u}_{u+v}) \circ \mathbf{f}_{\text{end}},$$

where $\mathbf{f}_{\text{start}}$ and $\mathbf{f}_{\text{end}}$ are the (possibly empty) prefix and suffix of $\mathbf{f}$ with no discernible codeword from $\mathcal{C}_{\mathrm{MU}}$ that is encoded from a non-marker. In line 8, the decoder updates

$$\texttt{approxNext}[\mathbf{u}_{u+c}] = \mathbf{u}_{u+c+1},$$

for every $c \in [0, v-1]$. The above process stops once no more codewords in $\mathcal{C}_{\mathrm{MU}}$ can be found.

The decoding algorithm proceeds to correct the constructed key-value store approxNext to next, i.e., the key-value store generated in Algorithm 1 from $\mathbf{w}$, using the collected redundancy strings and a standard Reed-Solomon decoder (line 9).

Next, the array dStrings is obtained from next (line 10), and the function D-DECODE (Alg. 4, Appendix A) is employed to recover $\mathbf{u}$. Recall that $\mathbf{u} = \mathbf{p}^{(0)} \circ \ldots \circ \mathbf{p}^{(a-1)} \circ \mathbf{w}$ in (4), and hence the decoding procedure concludes by returning the k rightmost bits of $\mathbf{u}$ (line 12).

Together, correct decoding is guaranteed by the following theorem, whose proof is provided in Appendix B.

Theorem 1. Line 12 of Algorithm 2 returns correct information word $\mathbf{w}$ if $4 \cdot t + 2s/(m + \lceil \log m \rceil + 4) \leq 4 \cdot \alpha$.

4.5 Trusted Information Embedding

Figure 6 shows the system design of trusted fingerprint embedding procedures. To prevent attackers from tampering the fingerprint embedding procedures, SIDE executes the 3D

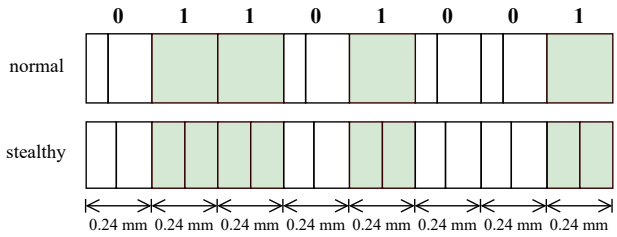


Figure 4: Demonstration of embedding 01101001 with parameters $x = 0.08$ and $(y, \varepsilon) = (0.12, 0.02)$. In the normal settings, each 0 is represented by two layers of 0.08 and 0.16 millimeters, and each 1 is represented by one layers of 0.24 millimeters. In the stealthy settings, each 0 is represented by two layers of 0.10 and 0.14 millimeters, and each 1 is represented by two layers of 0.12 millimeters. In either case, the length required for embedding one bit is 0.24 mm.

fingerprint embedding procedure and dependencies in TEE, including fingerprint information encoding (codec), object model slicing (layer-gen), toolpath generation (toolpath-gen), and 3D printer driver (firmware). TEEs are constrained by hardware resource limitations, notably their limited memory for the secure domain. Consequently, attempting to execute the entire fingerprint embedding procedure naïvely within the TEE risks print failures due to insufficient secure heap space, which is inadequate to handle the size of 3D model files and the substantial intermediate data generated during the slicing process. To mitigate this limitation, we propose a *progressive slicing* strategy, which reduces peak memory usage for procedures with the highest heap memory demand. This approach leverages the inherent layer-by-layer nature of the 3D printing process. Instead of pre-slicing the entire 3D model and generating a G-code file prior to printing, slicing is performed dynamically in an on-demand manner during the printing process.

5 SIDE Implementation

This section details the implementation of SIDE, including bit embedding, extraction, and TEE integration.

5.1 Bit Embedding Method

We address bit embedding at two levels: normal and stealthy. The normal approach focuses solely on the readability of the embedded bits, while the stealthy approach imposes an additional requirement of indiscernibility.

Normal Embedding. In the proposed normal method, there are three layer thicknesses: x, $2x$, and $3x$, where x is a base thickness. A 0 bit is represented by two consecutive layers of thickness x and $2x$, respectively, and a 1 bit is represented by a single layer of thickness $3x$. This method has three key advantages. First, it improves the readability as the substantial difference between the layers minimizes confusion during bit reading. Second, it provides consistent embedding density, defined as the number of bits embedded per unit distance. Since both 0 and 1 are represented using the same total thickness $3x$, the embedding density remains consistent for both 0 and 1. This ensures that the required object height depends solely on the length of the codeword, rather than its content. Third, it includes directional information, as the arrangement of layers for 0's implies the direction of the codeword.

Stealthy Embedding. The stealthy embedding method prioritizes indiscernibility by minimizing differences in layer thickness between layers. In this method, every bit is represented by two layers. Specifically, a 1 is represented by two layers of y millimeters, and a 0 is represented by two layers of $y-\varepsilon$ and $y+\varepsilon$ millimeters, respectively. The ε shall be imperceptible to naked eyes but still discernible to bit extraction equipments. This approach offers the added benefit of stealthiness, making it harder for the adversary to distinguish the embedded bits. Yet, it demands higher resolution for accurate bit extraction.

An illustration of both embedding methods is given in Figure 4. In reality, the x, y and ε are determined to accommodate the resolutions of the printer and the bit extraction equipment.

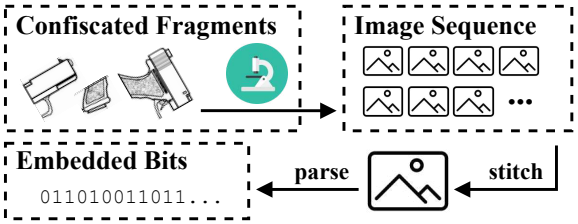


Figure 5: Procedure of bit extraction.

5.2 Bit Extraction Method

The extraction of bits is performed by inspecting the thicknesses of the layers in the fragments using specialized equipment and determining the corresponding bits. In this section, we describe an optical extraction method that employs a microscope to inspect the ridges on the surface of the print[3], with a graphical illustration given in Figure 5.

Microscopes typically have a limited field of view, making it impossible to inspect an entire fragment at once. To overcome this limitation, we mount the fragment on a motorized rail slider. During the extraction process, the fragment slides over the microscope's field of view, while the microscope takes a series of pictures. The pictures are taken so that every two consecutive pictures overlap, which allows us to fuse them together and obtain a picture of the entire fragment.

[3]If surface examination is not feasible (e.g., due to post-processing done on the surface), a computed-tomography based method can be utilized. However, it is beyond the scope of this paper

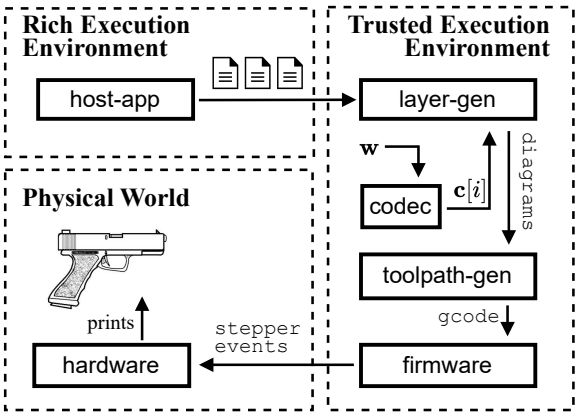


Figure 6: TEE Protected Embedding Procedure

To automatically read bits from the stitched images, we developed `bit-parser`, a program capable of parsing bits in both normal and stealthy embedding settings. In the case of normal embedding, `bit-parser` begins by identifying the layers representing 1's; they are characterized by a single, thickest layer of $3x$ millimeters, making them easily distinguishable from others. The program then counts the layers between these thick layers, with each pair of consecutive layers corresponding to a 0.

For stealthy embedding, `bit-parser` examines every two consecutive layers. If the first layer is thinner than the second by a specific threshold, the pair corresponds to a 0. The program then counts the layers between the layer pairs representing 1's, with each pair of consecutive layers corresponding to a 1. This concludes the process of bit extraction.

5.3 TEE Protected Embedding

Following the blueprint in Figure 6, SIDE involves a host-app that runs in the normal world to serve as the frontend interface for printer users, and its backend, which consists of four functional modules (codec, layer-gen, toolpath-gen, and firmware), that fully reside in the TEE. The host bridges the normal world with the trusted world using C Foreign Function Interface (CFFI). Upon receiving a 3D model from the user, it first cuts it into consecutive segments in the z direction, and passes them to SIDE backend one after the other. The thickness of segments equals to the distance required to represent a bit determined by the bit embedding method.

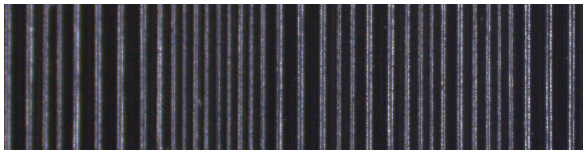
The actual slicing, as well as the bit embedding, is executed in the backend. The codec module is designed to perform encoding, mapping an information word $\mathbf{w}$ to a break-resilient codeword $\mathbf{c}$; the details of the encoding process were given in Section 4 and implemented in Algorithm 1. Recall that every bit $\mathbf{c}[i]$ instructs the slicing of the corresponding segment. If $\mathbf{c}[i] = 1$, then the segment is sliced to a layer of $3x$ mm with normal embedding, or $2y$ mm with stealthy embedding. Otherwise, it is sliced to two layers of x mm and $2x$ mm with normal embedding, respectively, or $y-\varepsilon$ mm and $y+\varepsilon$ mm with stealthy embedding (see Section 5.3 for details). The slicing is performed by the layer-gen module. For each layer, it generates the cross-sectional `diagram`, and feed them to toolpath-gen along with their corresponding heights, i.e., their distances to the printer bed. With these inputs from layer-gen, the toolpath-gen generates nozzle toolpath (represented by G-code) used to manufacture these layers. Both the layer-gen module and the toolpath-gen module are developed on top of t43 [45].

Finally, the firmware performs the parsing of G-code generated from toolpath-gen. It is a collection of core functionalities provided by Klipper, including the computation of precise nozzle movement and the generation of stepper events. The stepper events are then converted to signals passed to the printer hardware. This concludes the handling of $\mathbf{c}[i]$. Upon finishing the slicing of a segment, the host-app module is triggered to feed in the next layer segment, and the printing process is concluded after handling all segments. Since the entire printing process is hidden in the trusted world and no intermediate data (e.g., a G-code file) is exposed to the user, the adversary is unable to strip off the embedded bits.

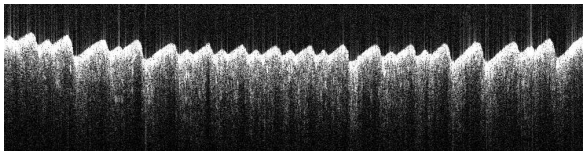
In prototyping our design, we employ a Creality Ender 3 3D printer, and a Raspberry Pi 3B board (with OP-TEE V3.4 support enabled in its ARMv8-A architecture, and Raspbian Linux 4.14.98-v7 installed for the normal world) to serve as the control board. Our development is heavily based on the Klipper open-source project [46] and t43 [45]. The former is a 3D printer firmware known for offering high precision stepper movements offering support to printers with multiple micro-controllers, and is suitable for running on low-cost devices such as Raspberry Pi. The latter is an open-source slicer program with basic functionalities and is purely written in C, making it suitable for trusted environments with limited language support.

6 Evaluation

This section presents a comprehensive evaluation of SIDE, focusing on fingerprint recovery, its impact on the printing process, and print quality. Specifically, we provide: (1) Experiments and simulations assessing fingerprint recovery from broken prints. (2) Analysis of the BRC code rate and the minimum object dimensions required for successful fingerprinting. (3) Analysis the practicality of stealthy embedding. (4) Print quality comparisons across normal embedding, stealthy embedding, and no embedding. (5) Assessment of printer imperfections through analysis and experiments. (6) Evaluation of TEE integration overhead and its effects on the printing process and print quality.



(a)



(b)

Figure 7: The readings of (a) an FDM fragment using the Leica S9D microscope, in which every bright line is the center of a layer, and (b) an SLA fragment using OCT equipment, which shows the cross-section of fragment surface, in which a ridge represents a layer. Both fragments are printed with the normal embedding method.

(a) $\beta = 0$

(b) $\beta = 10$.

(c) $\beta = 50$.

(d) $\beta = 100$.

Figure 8: Results of fragmentation under different β values.

6.1 Fingerprint Recovery

We conducted experiments and simulations to validate fingerprint recoverability. The experiments replicate the forensic information flow, including BRC encoding, bit embedding, fragmentation, bit extraction, and BRC decoding, demonstrating the core functionality of SIDE. In simulations, we assess fingerprint recovery success under varying conditions, such as model size and shape, extent of fragmentation, and fragment loss. The results confirm the robustness and practicality of SIDE for forensic fingerprinting applications.

Real World Scenarios: We conduct experiments to verify the recoverability of fingerprints, providing a proof of concept for SIDE. The experiments mimic the information flow in a forensic scenario, involving BRC encoding, bit embedding, fragmentation, bit extraction, and BRC decoding. Specifically, we prepared a fingerprint of 120 bits, and encoded it into 1-, 2-, and 3-BRC codewords of 281, 353, and 425 bits, respectively.

The experiments were carried out using both FDM and SLA printers, employing the bit embedding method introduced in Section 5.1. For the Creality Ender 3 FDM printer, we set $x = 0.08$, while for the Elegoo Mars 4 SLA resin printer, $x = 0.04$. These values were selected to balance information density (i.e., the number of bits embedded per unit length) with the resolution capabilities of the respective printers.

For each printer, we printed a cuboid of width 6mm and length 20mm, while the height is determined by the embedding method and codeword length. The printed cuboids were then manually broken apart to the maximum allowance of fragmentation by the embedded BRC codeword.

The fragments from FDM printer are examined using the methods described in Section 5.2 with the help of a Leica S9D microscope. The fragments from SLA printer, however, are examined with an Optical Coherence Tomography (OCT) device, since their layer thickness are beyond the resolution of optical microscopes. OCT is based on low-coherence interferometry to capture depth-resolved images with micro-level resolution in a non-invasive manner. It provides fast 3D imaging and quantitative, layer-by-layer analysis. Although widely used in biomedical and clinical diagnostics, OCT has also been adopted in non-biomedical fields such as industrial inspection [47], art conservation [48], and geology [49]. To extract the embedding information from 3D prints, we built our customized *spectral domain* OCT (SD-OCT) system using visible light with an axial resolution of 1.9 μm.

Finally, the extracted bits were fed to the BRC decoder (Alg. 2). In all cases, we achieved a 100% success rate, i.e., the information word perfectly matched the output of the decoding algorithm.

Simulation Environment: This section presents a simulation-based study of fingerprint recovery, which extends the experiments described in the previous section with three significant enhancements. First, it removes the constraints on the number and orientation of breaks.[4] Second, it introduces greater diversity in model shape and security parameter. Finally, it evaluates the decoding success rate under the condition of fragment loss. These enhancements provide more realistic simulations that closely mirror real-world forensic scenarios.

Each simulation is defined by three parameters, α, β, and ρ. With a fingerprint information of 128 bits (the value is chosen based on the length of serial numbers of printers used in experiments), an α-BRC codeword is generated. The parameter β determines the granularity of fragmentation. Specifically, a 3D Voronoi diagram from β randomly chosen points within the model mesh is generated, and the model is fractured using the planes separating the Voronoi cells. This procedure ensures at least β fragments since there are β Voronoi cells,

[4] In experiments, breaks are restricted to being orthogonal to the printing direction, and their number is limited by the maximum allowable fragmentation of embedded BRC codeword.

	4-BRC, 0.342 mm/bit				8-BRC, 0.215 mm/bit			
β \ ρ	0%	25%	50%	75%	0%	25%	50%	75%
20	100%	99.90%	97.05%	48.36%	100%	100%	100%	95.95%
40	100%	99.98%	99.00%	57.52%	100%	100%	100%	99.80%
60	100%	99.98%	98.80%	53.00%	100%	100%	100%	99.95%
80	100%	99.98%	98.17%	46.80%	100%	100%	100%	99.98%
100	100%	99.93%	96.75%	33.59%	100%	100%	100%	100%

Table 1: A portion of simulation results on the FMDA Glock frame using 4-BRC and 8-BRC, which demonstrates an exceptionally high probabilities of fingerprint recovery even in the extreme cases.

each corresponds to at least one fragment (see fragmentation results in Figure 8). Finally, the parameter ρ determines the ratio of fragments hidden from law enforcement.

Simulations are conducted on three models: the FMDA Glock frame (Fig. 1a), an AR-15 lower receiver, and the benign 3DBenchy. For each $\beta \in \{10, 20, \ldots, 100\}$, the previously described random fragmentation process is applied to each model, generating 128 fragmentation instances. For each of these 128 instances, we conceal a uniformly random ρ percent of the resulting fragments for each $\rho \in \{0\%, 25\%, 50\%, 75\%\}$, and repeat over 32 simulations. The decoder is then tested to determine whether it can recover the fingerprint from the remaining fragments.

We record the success rate of fingerprint recovery for every possible configuration (α, β, ρ) and every model. All simulation results are provided in Appendix C, while Table 1 offers a subset of them. With $\alpha = 8$, we observe an exceptionally high probability of fingerprint recovery even in the scenario when the printed Glock frame is broken into 100 fragments with 75% missing from the law enforcements.

6.2 Code Rate

The code rate r of BRC is defined as the ratio between the codeword length n and the information length k, i.e.,

$$r = \frac{k}{n} = \frac{l \cdot m - 1}{l(m + \lceil \log m \rceil + 4) + a(4m + 11)},$$

and plays a critical role in forensic applications. Given an object to print and a security parameter α, a higher code rate allows embedding more information bits, supporting more advanced forensic functionalities. These bits may include user IDs, geoposition data, and even the hash of model file. For example, embedding a unique user ID into a printed object can help trace adversaries using Manufacturing-as-a-Service (MaaS) to make the criminal tool. Geoposition data at the time of printing can aid law enforcement in tracking adversaries, while hash values act as watermarks for robust IP protection.

Conversely, given a fixed model and specific fingerprinting information, a higher code rate permits using a larger security parameter α, increasing resilience to breaks and fragment losses. Alternatively, when embedding fingerprints of a specific length with a given security parameter, a higher code rate

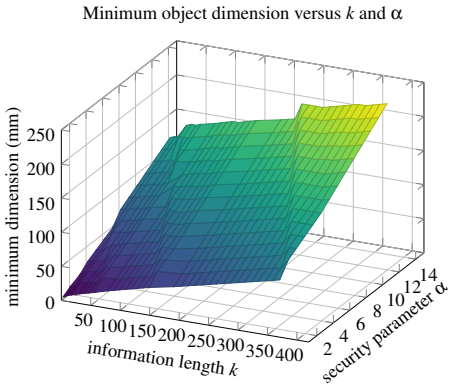
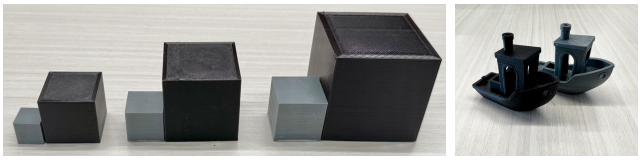


Figure 9: An illustration of minimum object dimension that allows for BRC codeword embedding, in which a bit is represented by 0.12 mm in object height, with respect to information length k and security parameter α.



(a) (b)

Figure 10: (a) From a single information word of 39 bits, we generate 1-BRC, 2-BRC, and 3-BRC codewords with lengths of 133, 191, and 249 bits, respectively. For each codeword, we calculate the minimum required dimensions based on the embedding parameters described in Section 5.1. Using these dimensions, we print information-bearing cubes with both the Ender 3 and Mars 4 printers. The resulting cubes have side lengths of 15.96 mm, 31.92 mm, 22.92 mm, 45.84 mm, 29.88 mm, and 59.76 mm. (b) 3DBenchy models printed for surface roughness evaluation.

reduces the required object dimension along the printing direction, and hence broadens the applicability of SIDE. To this end, we visualized the minimum object dimensions required to embed BRC codewords in Figure 9, and printed 6 cubes with BRC codeword embedded to validate the feasibility of embedding; each is of the minimum dimension corresponding to the information length and security parameter (Figure 10a).

In Figure 11, we demonstrate how the code rate r is affected by the information length $k = m \cdot l - 1$ and the security threshold $t \in [10]$.[5] Meanwhile, as we will introduce in Section 8.2, the method of using cyclically permutable codes (CPC) for the t-break-resilient property requires repeating the CPC codeword $t+1$ times, and hence the code rate is at most $1/(t+1)$. We also mark this value in the figure for every t in the same color. In the majority of cases, BRC outperforms CPC-based

[5] Since we fix $m = 12$, the maximum l we can have is $2^{(12-2)/2} - t = 32 - t$. Therefore, the set of values of k is different with different t.

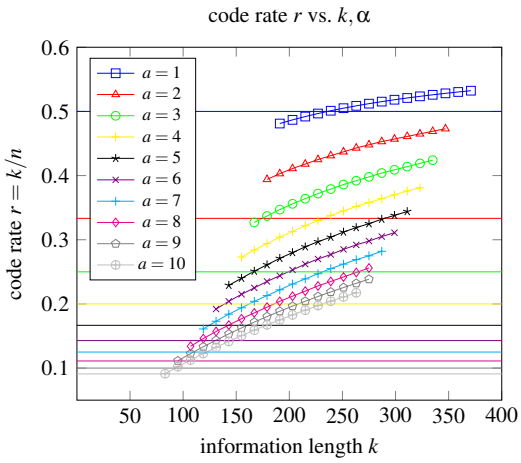


Figure 11: An illustration of the code rate r verses the information length k and security parameter α. The horizontal lines serve as upper bounds on the code rates of a traditional CPC-based scheme (Section 8.2). The curved lines represent the code rate of our scheme (Section 4), color coded by α.

scheme in terms of the code rate.

6.3 Stealthy Embedding

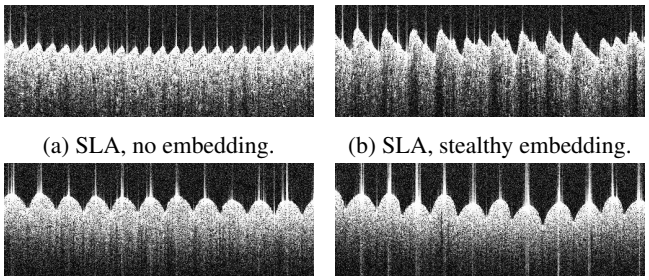
The primary goal of stealthy embedding is to minimize detectability, ensuring that the fingerprint remains hidden from adversaries while preserving its readability by forensic tools. Hence, the feasibility of a stealthy embedding method is determined by its *stealthiness* and *readability*.

To evaluate stealthiness, we conducted experiments using both FDM and SLA printers. Specifically, we use the stealthy embedding method introduced in Section 5.1 with parameter $(y, \varepsilon) = (0.12, 0.04)$ for the FDM printer and $(y, \varepsilon) = (0.6, 0.02)$ for the SLA printer. Results indicate that the differences in layer thicknesses were invisible to naked eyes under normal lighting conditions, unless the object is observed in certain angles. We further measured surface roughness with our SD-OCT system. The results, shown in Table 2, reveal a higher RMS value in prints with no embedded bits, but lower than prints using normal bit embedding.

Readability refers to the accuracy of bit extraction. With the extraction method introduced in Section 5.2 and the SLA printer with parameters $(y, \varepsilon) = (0.08, 0.04)$, we successfully extracted embedded bits using SD-OCT system. Yet, we observed a trade-off between stealthiness and readability: while reducing ε enhances stealthiness, it increases the demand for both high-resolution printer and extraction tools.

6.4 Impacts on Print Quality

Due to the inherently discrete, layer-by-layer nature of the additive manufacturing process, surface roughness is a common characteristic of 3D-printed objects, often necessitating post-processing techniques such as sanding or filing. Yet, SIDE's bit embedding requires to vary layer thickness, which can potentially increase surface roughness and expose the presence of the embedded bits to the adversary.



(a) SLA, no embedding. (b) SLA, stealthy embedding.

(c) FDM, no embedding. (d) FDM, stealthy embedding.

Figure 12: OCT scans on different materials and embeddings.

	No Embedding	Stealthy	Normal
Ender 3 (FDM)	10.741 μm	13.267 μm	14.719 μm
Mars 4 (SLA)	7.260 μm	11.275 μm	12.918 μm

Table 2: The RMS values with different embedding methods and materials.

To quantify this impact, we printed 3DBenchy with normal embedding, stealthy embedding, and no embedding (uniform 0.12 and 0.06 mm layer thicknesses for FDM and SLA, respectively) with both FDM and SLA printers. For each printed model, we used OCT to capture surface height deviations at five randomly chosen points of the flat part of its surface. With the collected data, we computed and averaged the root mean square (RMS) roughness value, which is a widely recognized metric for surface roughness that quantifies the root mean square of surface height deviations from the mean surface height. The results are provided in Table 2, which quantify the impacts on print quality.

6.5 Impact of Printer Imperfection

In this section we briefly discuss the effect that printer imperfections might have on the ability to embed information. Recall that SIDE embeds bits by varying layer thickness, and as a result, imperfections in 3D printers—particularly inaccuracies in the z-axis movement—can have negative effects on the embedding process. The z-axis movements are typically controlled by a stepper motor, which converts its rotational motion (i.e., discrete steps) into linear motion along the printing direction; the ratio between them is reflected in *z-step* value (e.g., 0.04 mm/step). Calibration of a printer involves matching this parameter in the printer firmware with the actual *z-step* value.

A perfectly manufactured printer would have a uniform z-step, i.e., the nozzle displacement along the z-axis, which is

	Peak Heap Usage		Print Time		Firmware Execution Time	
Object	w/ TEE	w/o TEE	w/ TEE (s)	w/o TEE (s)	w/ TEE	w/o TEE
Glock	1.475 MB	9.841 MB	53344.201	53344.103	163.85 s	163.85 s
AR-15	1.305 MB	8.438 MB	76831.497	76831.397	209.69 s	209.69 s
3DBenchy	0.965 MB	9.979 MB	11828.314	11828.241	34.90 s	34.90 s

Table 3: System Performance of SIDE.

triggered by one microstep, is uniform across the entire range. Yet, due to manufacturing errors, individual steps may lead to different nozzle displacements during printing, and the actual layer thickness may not align with the desired value.

SIDE's embedding method is designed to tolerate these imperfections. For normal embedding, layers with designed thicknesses of x, $2x$, and $3x$ may be in the range of $[(1-\delta)\cdot x,(1+\delta)\cdot x]$, $[(1-\delta)\cdot 2x,(1+\delta)\cdot 2x]$, and $[(1-\delta)\cdot 3x,(1+\delta)\cdot 3x]$, where δ upper bounds the magnitude of errors in each step. To avoid confusion in the reading of bits from layer thickness, these ranges must not overlap, requiring $\delta < 0.2$; this is an extremely low standards, considering that the common stepper motors used in 3D printers (e.g., NEMA 17) usually have $\delta = 0.05$.

To evaluate SIDE's robustness under these imperfections, a cuboid was printed using an FDM printer with noise added to the z-step value in the firmware; this simulates the behaviors of a poorly manufactured printer. Despite these imperfections, bits were successfully extracted, demonstrating the method's reliability even in the extreme real-world conditions.

6.6 Impact on Printing from TEE Protection

SIDE prevents attackers from exploiting vulnerabilities in untrusted software by leveraging TEE to isolate SIDE from untrusted components. To evaluate the impact of TEE protection on 3D printing, we measure (1) the execution delay of SIDE's software components, (2) the execution delay and memory overhead of the whole 3D printing procedures, and (3) the impact on the quality of printed objects with and without SIDE on Glock frame, AR-15 lower receiver, and 3DBenchy.

Software Components Execution and 3D Printing Delay: To measure the execution delay of SIDE's main components, including the codec, G-Code generation (involving both layer and toolpath generation), and firmware, we record timestamps at the start and end of each component's execution and calculate the average delays over 10 printing processes. To evaluate the overall 3D printing delay, we record timestamps marking the start and completion of the printing application execution and calculate the difference to determine the delay. As shown in Table 3, SIDE introduces no runtime overhead for the execution delays of individual software components under protection of TEE, as the printing binaries remain the same on the same architecture regardless of the CPU security state. However, SIDE reduces peak memory usage by employing the progressive slicing strategy, which introduces multiple context switches between the REE and TEE. This results in additional execution delay for the overall printing process. Nonetheless, when compared to the delay inherent in physical printing, this runtime overhead is negligible.

Memory Overhead: To measure the effectiveness of SIDE in automatically splitting 3D object models to reduce heap consumption, we instrument dynamic memory allocation and deallocation APIs within the 3D printing software to monitor the size of dynamically allocated memory, both with and without SIDE. As shown in Table 3, SIDE reduces the peak heap memory usage to 14.99%, 15.47%, and 9.67% on Glock, AR-15, and 3DBenchy respectively through progressive slicing, effectively addressing the memory limitations of TEE.

Printed Object Quality: To measure the quality impact on printed object from TEE implementation, we calculate the root mean square (RMS) roughness of printed object with observed height from optical coherence tomography with and without TEE implementation. We observe that the difference in the roughness of both objects is negligible.

7 Security Analysis and Discussion

This section analyzes the security of SIDE against various potential attacks.

Excessive breaking and hiding: In order to jeopardize fingerprint extraction, the adversary may attempt to compromise the availability of embedded information by excessively breaking the printed tool and hide a great amount of fragments. However, as shown in the simulation results in Appendix C, SIDE provide an exceptionally high success rate of fingerprinting recovery even in the extreme case that the tool is broken into 100 pieces, with 75% of them being missing from the decoder.

This is attributed to the break-resiliency and loss-tolerance properties of α-BRC. First, as stated in Remark 1 and Section 4.3, fragments can be reassembled if they retain sufficient overlapping bits. Thus, a break is repairable in the preprocessing stage unless it is perpendicular, or nearly perpendicular, to the printing direction. Additionally, concealing fragments does not necessarily result in the loss of information, as their content may also exist in confiscated fragments. Finally, the breaks and missing bits that cannot be fixed in the preprocessing stage are addressed in BRC decoding (Section 4.4).

Forging Attacks: Attackers may attempt forging attacks by embedding incorrect information into a printed object to impersonate another printer. We propose two tiered defense against such attacks. First, in order to impersonate another printer, one would have to breach the TEE of one's own printer, contradicting our security assupmtion. Second, even if the TEE is broken successfully, impersonation can still be prevented via exploiting intrinsic printer properties as follows. It was shown in the literature that much like firearms, 3D printers carry a unique signature that is manifested in various minor defects in the resulting prints [16], [17]. Hence, SIDE

can embed a hash of these imperfections inside the object. Then, forging attacks would fail since the de-facto defects of the object would not match the hash of the printer's imperfection. However, manipulating the embedding mechanisms in SIDE requires significant additional effort, such as breaching the TEE-protected 3D printing software or employing specialized hardware and expertise to modify the object's surface. Thus, SIDE raises the bar for this attack. Furthermore, SIDE can authenticate the print by embedding the hash of intrinsic printer properties, such as manufacturing imperfections [16] or thermodynamic characteristics [17], enabling detection of mismatches between the print and its originating printer.

Hardware Attacks: Adversaries may attempt hardware attacks, such as replacing the control board or injecting signals between the control board and printers, to bypass fingerprint embedding. While SIDE does not defend against these hardware attacks, which require specialized expertise and significant cost, establishing secure communication channels and authentication mechanisms between the control board and printer component controllers can help mitigate such attacks, further raising the barriers to bypassing 3D fingerprinting.

Surface Altering Attack: Attackers may attempt to alter the surface of 3D prints to tamping with the 3D fingerprinting information. SIDE is inherently immune to this type of attack by its design. Recall that SIDE embeds information by altering physical elements on the printing direction (e.g., layer thickness), which is an intrinsic property of the printed object and cannot be altered by post processings on the surface. Indeed, the reading of bits does not require the surface of the fragment to be free of adversarial tampering, as less-economical solutions, such as an industrial CT scanner, can be used to infer the layer thickness via tomographic analysis.

8 Related Work

8.1 Existing 3D Fingerprinting Methods

Several methods for embedding bits in 3D-printed objects have recently been proposed in the literature. These technologies allow the printer to vary either the orientation of the nozzle, the thickness of the layer, or the printing speed. Within reasonable bounds, varying either of those has a marginal effect on the functionality of the object. By varying layer thickness, for example, the printer can embed a 0 by printing a layer that is slightly thinner, and a 1 by printing a layer that is slightly thicker, than some reference thickness. By varying the orientation of the nozzle, bits can be embedded by the relative orientation of adjacent layers; for example, if two adjacent layers are oriented similarly, the embedded bit is 0, and otherwise it is 1. Both methods are illustrated in Figure 13.

Similar ideas have been implemented successfully in several recent works. Delmotte *et al.* vary the thickness of each layer across several adjacent layers to create a matrix of bits

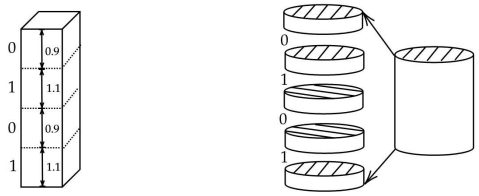


Figure 13: Two possible methods for embedding bits in a 3D printed object with little to no effect on functionality: (a) Embedding by layer thickness; thicker layers represent 1 and thinner layers represent 0. (b) Embedding bits using the orientation of adjacent layers; if two adjacent layers are oriented similarly, it is a 0, and if oriented differently, it is a 1. Both left and right figures contain the bits 0101.

that is visible to the naked eye on the surface of the object [8]. Parity bits were then added to resolve reliability issues in some cases, and additional noise patterns were discussed, such as orientation issues and sanding. In the method LayerCode [50], variations in color and thickness were used to embed a barcode on the surface of the printed object, that can be retrieved using a smartphone camera. An orientation-based method has been implemented in [51], where the authors print a reference layer that is circularly grooved by a sequence with low auto-correlation. Data is embedded in all other layers by the respective angle of the layer to the reference layer; this enables encoding with alphabet size larger than two.

Other creative ideas have been explored, including embedding information-carrying cavities within the object [52], [53], water-marking the 3D-mesh of the surface of the object [54], inserting RFID tags [12], inserting a series of notches which create an acoustic barcode when tapped [15], etc. In the data extraction phase, most existing methods rely on an RGB camera, a 3D scanner, or an ordinary scanner. Future technologies however, such as the ones in Figure 13, might require an industrial CT scanner. However, none of these approaches is suitable for forensic applications. First, they all implicitly assume that a mechanism ensuring correct information embedding is in place. Such a mechanism is crucial since in most scenarios the adversary owns the printer and/or the file, and might potentially remove the embedded bits altogether. Second, none of the methods is provably resilient to adversarial tampering; they can be easily breached by an adaptive adversary that can scrape the object or break it apart.

PrinTracker [16] represents a different line of works, which apply machine-learning based methods to identify the intrinsic fingerprint resulting from printer hardware imperfections. Our proposed solution allows for embedding arbitrary bits, which enables a variety of forensic applications, including but not limited to fingerprinting. For example, certain applications may require the embedding of timestamp or geoposition of the printing into the printed object in order to check whether the

printer has been misused in unauthorized time and location.

8.2 Coding Methods

Secure information extraction is essentially a problem of communicating in the presence of (potentially adversarial) noise. Such problems are studied in the information-theoretic literature under the title of coding theory [18]. A typical problem setup in coding theory includes a sender (e.g., the printer), which would like to send a message (e.g., printer ID) to a receiver (e.g., law-enforcement). This must be accomplished successfully even if an adversary (e.g., a criminal) injects adversarial noise that is limited by some security parameter.

Coding problems of information extraction from fragments have been previously studied in the literature, motivated by applications in distributed systems [55]–[58], and DNA storage. In particular, several variations of the *torn-paper channel* were studied in [59]–[62], where [59]–[61] focused on a probabilistic error model which is incompatible with our adversarial setting, and [62] studied an adversarial model in which fragment are restricted in length. SIDE complements these probabilistic-based solutions by relaxing the assumptions. In fact, we assume an adaptive adversary who is fully aware of the coding scheme, and is constrained only by the parameters t and s. Consequently, previous methods fail to ensure secure information embedding facing such an adversary, as they can be exploited by strategically selecting break locations and concealing specific fragments.

Similarly to our scenario, a video watermarking solution for IP protection [63] employs *cyclically permutable codes* (CPC). As the name suggests, CPC codewords are cyclically distinct, i.e., one cannot obtain a codeword by cyclically shifting another codeword. In CPC-based solutions, the watermark is encoded to a CPC codeword, and then iteratively spread over consecutive video frames. Due to its cyclically distinct feature, the embedded CPC codeword can be obtained from every video clip that has more frames than the codeword length. Although this method may serve as a simple solution to the 3D fingerprinting problem, it requires to repeat the CPC codeword at least $t+1$ times to guarantee the existence of such a video clip in every possible way to cut the video t times. Hence, it leads to a code rate of at most $1/(t+1)$, i.e., $k(t+1)$ bits needs to be actually embedded in the video to represent k bits of information, which hinders its applicability in real-world scenario; see Figure 11 for rate comparison.

Most closely related, [64] studied t-break codes, which concerns the recovery of information from *arbitrarily* broken codewords, which is fundamentally different from probabilistic error models mentioned earlier. They provided a theoretical analysis of the fundamental limits, and an (almost) matching code construction. However, the scheme described in [64] involves random encoding, is only effective for a very large number of embedded bits, and only tolerates a small amount of lost bits. SIDE complements these works by offering a coding scheme with tolerance to fragment loss, explicit and efficient encoding function, and high code rate.

Coding Method	Model	Minimum Length of Fragments	Loss Tolerance
Shomorony et al. [59], [60]	probability	Not Assumed	No
Ravi et al. [61]	probability	Not Assumed	Yes
Bar-Lev et al. [62]	adversarial	Assumed	No
Kuribayashi et al. [63]	adversarial	Not Assumed	Yes
Wang et al. [64]	adversarial	Not Assumed	No
α-BRC	adversarial	Not Assumed	Yes

Table 4: Coding Method Comparison

9 Conclusion

This paper introduces SIDE, a forensic fingerprinting mechanism for 3D prints that ensures secure information embedding and extraction against adversarial threats. To enable fingerprint recovery despite malicious breaks and fragment hiding, SIDE employs break-resilient, loss-tolerant embedding techniques. It safeguards the embedding process using a TEE and a progressive slicing mechanism. SIDE's efficiency and effectiveness are validated through simulations and a prototype on a Creality Ender 3 printer with a Raspberry Pi 3B.

Acknowledgments

This research was supported by the National Science Foundation (Grants CNS-2223032, CNS-2038995, and CNS-223863) and the Army Research Office (Grant W911NF-24-1-0155).

Ethics Considerations

This work focuses on advancing 3D printing forensics by proposing a novel fingerprinting solution leveraging Trusted Execution Environments (TEEs) and coding techniques. The proposed method allows reconstructing fingerprints even if the print is fragmented and with some parts missing.

This work does not reveal new vulnerabilities or introduce new attack vectors. Additionally, we do not anticipate that this solution could be misused to facilitate new types of attacks. The experiments conducted in this work do not involve live systems, and do not affect the well-being of team members.

Open Science

The research artifacts generated during this research include:

- Source code for BRC: This includes the implementation of our break-resilient coding (BRC) scheme, including the encoding and decoding algorithms.
- Source code for the slicer software: Slicer software specifically designed to ensure compatibility with TEE.

- Scripts for simulation: The script that performs random fragmentation of a 3D mesh, and the script that parses the simulation results.
- Microscope images: For each sample in the experiments, we provide a collection of images collected using Leica S9D microscope and the SD-OCT device.
- Script for bit extraction: The script extracts bits from the the aforementioned microscope images, and bit extraction results.

All artifacts are made publicly available at [65].

References

[1] H. Jiang and T. Hunt, *Untraceable 'ghost guns' are easier than ever to 3d print — we went inside a company that helps people do it*, 2021. [Online]. Available: https://news.yahoo.com/untraceable-ghost-guns-easier-ever-150015562.html.

[2] C. M. McNulty, N. Arnas, and T. A. Campbell, "Toward the printed world: Additive manufacturing and implications for national security," *Defense Horizons*, no. 73, p. 1, 2012.

[3] T. Gibbons-Neff and A. Toler, *When a glock isn't a glock: The history of the pistol found with luigi mangione*, Dec. 2024. [Online]. Available: https://www.nytimes.com/2024/12/12/us/unitedhealthcare-luigi-mangione-gun.html.

[4] A. Greenberg, *The "ghost gun" linked to luigi mangione shows just how far 3d-printed weapons have come*, Dec. 2024. [Online]. Available: https://www.wired.com/story/luigi-mangione-united-healthcare-3d-printed-gun-fmda-chairmanwon-v1/.

[5] D. Schoen, "Investigating machine identification code technology in color laser printers," *The Electronic Frontier Foundation*, 2005.

[6] P.-J. Chiang, A. Mikkilineni, E. Delp, J. Allebach, and G. Chiu, "Extrinsic signatures embedding and detection in electrophotographic halftone images through laser intensity modulation," in *NIP & Digital Fabrication Conference*, vol. 22, 2006, pp. 432–435.

[7] G. Ali, A. Mikkilineni, J. Allebach, E. Delp, P.-J. Chiang, and G. Chiu, "Intrinsic and extrinsic signatures for information hiding and secure printing with electrophotographic devices," in *NIP & Digital Fabrication Conference*, vol. 19, 2003, pp. 511–515.

[8] 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. Multimed.*, vol. 22, no. 11, pp. 2780–2791, 2019.

[9] K. ElSayed, A. Dachowicz, and J. Panchal, "Information embedding in additive manufacturing through printing speed control," in *ACM CCS Workshop on Additive Manufacturing Security*, 2021, pp. 31–37.

[10] D. Salas, D. Ebeperi, M. Elverud, R. Arróyave, R. Malak, and I. Karaman, "Embedding hidden information in additively manufactured metals via magnetic property grading for traceability," *Additive Manufacturing*, vol. 60, p. 103 261, 2022.

[11] 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. 29, pp. 6–9, 2017.

[12] 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.

[13] 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.

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

[15] C. Harrison, R. Xiao, and S. Hudson, "Acoustic barcodes: Passive, durable and inexpensive notched identification tags," in *ACM UIST*, 2012, pp. 563–568.

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

[17] Y. Gao, W. Wang, Y. Jin, C. Zhou, W. Xu, and Z. Jin, "Thermotag: A hidden id of 3d printers for fingerprinting and watermarking," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 2805–2820, 2021.

[18] R. Roth, *Introduction to Coding Theory*. Cambridge University Press, 2006.

[19] Z. Yu, Y. Chang, S. Zhai, *et al.*, "XCheck: Verifying integrity of 3d printed Patient-Specific devices via computing tomography," in *USENIX Security*, 2023, pp. 2815–2832.

[20] K. Krawiecka, A. Kurnikov, A. Paverd, M. Mannan, and N. Asokan, "Safekeeper: Protecting web passwords using trusted execution environments," in *TheWebConf*, 2018, pp. 349–358.

[21] F. Schwarz and C. Rossow, "SENG, the SGX-Enforcing network gateway: Authorizing communication from shielded clients," in *USENIX Security*, 2020, pp. 753–770.

[22] J. B. Djoko, J. Lange, and A. J. Lee, "Nexus: Practical and secure access control on untrusted storage platforms using client-side sgx," in *IEEE/IFIP DSN*, IEEE, 2019, pp. 401–413.

[23] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *ACM Trans. Comput. Syst.*, vol. 33, no. 3, pp. 1–26, 2015.

[24] F. Schuster, M. Costa, C. Fournet, *et al.*, "Vc3: Trustworthy data analytics in the cloud using sgx," in *IEEE S&P*, IEEE, 2015, pp. 38–54.

[25] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "Safebricks: Shielding network functions in the cloud," in *USENIX NSDI*, 2018, pp. 201–216.

[26] T. Hunt, Z. Jia, V. Miller, *et al.*, "Telekine: Secure computing with cloud gpus," in *USENIX NSDI*, 2020, pp. 817–833.

[27] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis, "Ppfl: Privacy-preserving federated learning with trusted execution environments," in *ACM MobiSys*, 2021, pp. 94–108.

[28] J. Wang, Y. Cheng, Q. Li, and Y. Jiang, "Interface-based side channel attack against intel sgx," *arXiv preprint arXiv:1811.05378*, 2018.

[29] J. Wang, A. Li, H. Li, C. Lu, and N. Zhang, "Rt-tee: Real-time system availability for cyber-physical systems using arm trustzone," in *IEEE S&P*, IEEE, 2022, pp. 352–369.

[30] S. Pinto, H. Araujo, D. Oliveira, J. Martins, and A. Tavares, "Virtualization on trustzone-enabled microcontrollers? voilà!" In *IEEE RTAS*, IEEE, 2019, pp. 293–304.

[31] J. Wang, Y. Wang, A. Li, *et al.*, "ARI: Attestation of real-time mission execution integrity," in *USENIX Security*, 2023, pp. 2761–2778.

[32] J. Wang, Y. Wang, and N. Zhang, "Secure and timely gpu execution in cyber-physical systems," in *ACM CCS*, 2023, pp. 2591–2605.

[33] J. Wang, Y. Wu, H. Liu, B. Yuan, R. Chamberlain, and N. Zhang, "Ip protection in tinyml," in *ACM/IEEE DAC*, IEEE, 2023, pp. 1–6.

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

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

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

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

[38] D. Bajic and J. Stojanovic, "Distributed sequences and search process," in *IEEE ICC*, vol. 1, 2004, pp. 514–518.

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

[40] 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.

[41] 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.

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

[43] 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.

[44] O. Elishco, R. Gabrys, E. Yaakobi, and M. Médard, "Repeat-free codes," *IEEE Trans. Inf. Theory*, vol. 67, no. 9, pp. 5749–5764, 2021.

[45] L. Huang, *T43*, 2024. [Online]. Available: https://github.com/LingDong-/t43.

[46] Klipper3d, *Klipper*, 2024. [Online]. Available: https://github.com/Klipper3d/klipper.

[47] B. He, Y. Zhang, L. Zhao, *et al.*, "Robotic-oct guided inspection and microsurgery of monolithic storage devices," *Nat. Commun.*, vol. 14, no. 1, p. 5701, 2023.

[48] H. Liang, M. G. Cid, R. G. Cucu, *et al.*, "En-face optical coherence tomography-a novel application of non-invasive imaging to art conservation," *Optics Express*, vol. 13, no. 16, pp. 6133–6144, 2005.

[49] S. Campello, W. Dos Santos, V. Machado, C. Mota, A. Gomes, and R. de Souza, "Micro-structural information of porous materials by optical coherence tomography," *Microporous and mesoporous materials*, vol. 198, pp. 50–54, 2014.

[50] H. T. Maia, D. Li, Y. Yang, and C. Zheng, "Layercode: Optical barcodes for 3d printed shapes," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 1–14, 2019.

[51] J.-U. Hou, D.-G. Kim, S. Choi, and H.-K. Lee, "3d print-scan resilient watermarking using a histogram-based circular shift coding structure," in *ACM IH&MMSec*, 2015, pp. 115–121.

[52] K. D. Willis and A. D. Wilson, "Infrastructs: Fabricating information inside physical objects for imaging in the terahertz region," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 1–10, 2013.

[53] D. Li, A. S. Nair, S. K. Nayar, and C. Zheng, "Aircode: Unobtrusive physical tags for digital fabrication," in *ACM UIST*, 2017, pp. 449–460.

[54] J.-W. Cho, R. Prost, and H.-Y. Jung, "An oblivious watermarking for 3-d polygonal meshes using distribution of vertex norms," *IEEE Trans. Signal Process.*, vol. 55, no. 1, pp. 142–155, 2007.

[55] C. Wang and N. Raviv, "Low latency cross-shard transactions in coded blockchain," in *IEEE ISIT*, IEEE, 2021, pp. 2678–2683.

[56] C. Wang and N. Raviv, "Breaking blockchain's communication barrier with coded computation," *IEEE Journal on Selected Areas in Information Theory*, vol. 3, no. 2, pp. 405–421, 2022.

[57] C. Wang and N. Raviv, "All-to-all encode in synchronous systems," in *IEEE ITW*, 2022, pp. 738–743.

[58] C. Wang and N. Raviv, "On the encoding process in decentralized systems," *arXiv preprint arXiv:2408.15203*, 2024.

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

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

[61] A. N. Ravi, A. Vahid, and I. Shomorony, "Capacity of the torn paper channel with lost pieces," in *IEEE ISIT*, IEEE, 2021, pp. 1937–1942.

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

[63] M. Kuribayashi and H. Tanaka, "How to generate cyclically permutable codes from cyclic codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4660–4663, 2006.

[64] C. Wang, J. Sima, and N. Raviv, "Break-resilient codes for forensic 3d fingerprinting," in *IEEE ISIT*, IEEE, 2024, pp. 3148–3153.

[65] C. Wang, J. Wang, M. Zhou, *et al.*, *Artifects*, Jan. 2025. [Online]. Available: https://doi.org/10.5281/zenodo.14737894.

A Distinct Strings

A.1 Encoding

Alg. 3 offers a procedure that maps an input word $\mathbf{w} \in \{0,1\}^{l\cdot m-1}$ to `dStrings`, which is an array of l pairwise distinct binary strings of length m. Besides, the inverse operation, as given in Alg. 4, outputs $\mathbf{w}$ given the array `dStrings`.

The encoding algorithm first appends a 1 to $\mathbf{w}$ (line 1), and as a result, $|\mathbf{w}| = l \cdot m$. The word $\mathbf{w}$ is segmented to l intervals of length m and placed in a tentative array `dStrings` (line 1). The segments are not necessarily pairwise distinct at this point, and they are referred as the *original*.

The encoding algorithm continues to look for identical original segments, deletes one of them, and appended a *new* binary string to `dStrings`; this new string contains the indexing information of the two identical binary strings which allows for decoding at a later point in time. Moreover, we end every new string with a 0 to distinguish it from the rightmost original segment which ends with a 1.

With indices i and i_{end} initially set to 0 and $l-1$, the encoder enumerates every index $j \in [i+1, i_{\text{end}}]$ for a match, i.e., `dStrings`$[i]$ is identical to `dStrings`$[j]$ (line 4). Once a match is found, the latter is deleted from `dStrings` (line 6) and the indices i, j are recorded in a binary string to be placed at the end of `dStrings`, which will be used for recovering the deleted entry during decoding.

Note that, naïvely defining the new strings as the concatenation of binary representations of i and j may introduce more repeated strings; it may coincide with existing elements in `dStrings`. To this end, the algorithm looks for an *alternative* binary representation of j which is not identical to the first $\lceil \log l \rceil + 1$ bits of *every* existing element in `dStrings`.

Starting from $j' = 0$, the following procedure is repeated j times. In each time, j' is increased by 1 (line 10). Then, it continues to increase until its binary representation in $\lceil \log l \rceil + 1$ bits does not coincide with the first $\lceil \log l \rceil + 1$ bits of every existing element in `dStrings` (line 11). One may imagine this process as looking for the j-th available parking slot in a row, in which some have been occupied (unavailable). A slot is unavailable if the binary representation of its index coincides with the first $\lceil \log l \rceil + 1$ bits of any element in `dStrings`. Otherwise, it is available. Starting for index 0, j' is indeed the index of the j-th available slot.

Note that when the repetition stops, j' equals to the sum of j and the number of times the condition in line 11 was true. Recall that $1 \le j \le l-1$, and the latter equals to the number of unavailable slots that may be encountered during the increment of j', which is at most $l-1$ since there are only $l-1$ elements in `dStrings`. Therefore, $j' \le 2l-2$ and can be represented by $\lceil \log l \rceil + 1$ bits.

Therefore, we use the binary representation of j' in $\lceil \log l \rceil + 1$ to serve as the alternative representation of j. It is concatenated with the binary representation of i (in $\lceil \log l \rceil$ bits since $i < l$) and sufficiently many 0's to make a *new* string (line 12). The new string is appended to `dStrings`, and is different from every other element in the first $\lceil \log l \rceil + 1$ bits; this fact gives the following lemma.

Lemma 1. The new binary string being appended in line 12 is different from every existing elements in `dStrings`.

Lemma 1 allows us to decrease i_{end} by one (line 13) since there is no need to compare `dStrings`$[j]$ with element whose index is greater than $i_{\text{end}} - 1$. The algorithm terminates when

Algorithm 3 D-ENCODE (distinct strings encoding)

Input: A binary string $\mathbf{w} \in \{0,1\}^{l \cdot m - 1}$, where l, m are positive integers such that $m \geq \lceil 2 \log l \rceil + 2$.
Output: Array `dStrings` of l length-m pairwise distinct binary strings.

1: Let $\mathbf{w} \leftarrow \mathbf{w} \circ 1$, $i \leftarrow 0, i_{\text{end}} \leftarrow l-1$, and $\texttt{dStrings}[a] \leftarrow \mathbf{w}[a \cdot m : (a+1) \cdot m)$ for $a \in [0, l-1]$.
2: **while** $i < i_{\text{end}}$ **do**
3: $\quad j \leftarrow i+1$
4: $\quad$ **while** $j \leq i_{\text{end}}$ **do**
5: $\quad\quad$ **if** $\texttt{dStrings}[i] = \texttt{dStrings}[j]$ **then**
6: $\quad\quad\quad \texttt{dStrings} \leftarrow (\texttt{dStrings}[0 : j-1], \texttt{dStrings}[j+1 : l-1])$
7: $\quad\quad\quad j' \leftarrow 0$
8: $\quad\quad\quad$ **repeat** j **times:**
9: $\quad\quad\quad\quad$ **do**
10: $\quad\quad\quad\quad\quad j' \leftarrow j'+1$
11: $\quad\quad\quad\quad$ **while** $\exists r \in [l-2]$ s.t. $\texttt{dStrings}[r-1][0 : \lceil \log l \rceil] = \text{BINARY}(j', \lceil \log l \rceil + 1)$
12: $\quad\quad\quad \texttt{dStrings}.\text{APPEND}(\text{BINARY}(j', \lceil \log l \rceil + 1) \circ \text{BINARY}(i, \lceil \log l \rceil) \circ 0^{m-2\lceil \log l \rceil - 1})$
13: $\quad\quad\quad i_{\text{end}} \leftarrow i_{\text{end}} - 1$
14: $\quad\quad$ **else**
15: $\quad\quad\quad j \leftarrow j+1$
16: $\quad i \leftarrow i+1$
17: **return** `dStrings`

Algorithm 4 D-DECODE (distinct strings decoding)

Input: Array `dStrings` of l length-m pairwise distinct binary strings.
Output: The information word $\mathbf{w} \in \{0,1\}^k$ used to generate `dStrings`.

1: **while** $\texttt{dStrings}[-1][-1] = 0$ **do**
2: $\quad i \leftarrow \text{INTEGER}(\texttt{dStrings}[l][\lceil \log l \rceil + 2 : 2\lceil \log l \rceil + 1])$
3: $\quad j, j' \leftarrow \text{INTEGER}(\texttt{dStrings}[l-1][1 : \lceil \log l \rceil + 1])$
4: $\quad$ **for all** $r \in [0, l-2]$ and $s \in [j'-1]$ **do**
5: $\quad\quad$ **if** $\texttt{dStrings}[r][1 : \lceil \log l \rceil + 1] = \text{BINARY}(s, \lceil \log l \rceil + 1)$ **then** $j \leftarrow j-1$
6: $\quad \texttt{dStrings} \leftarrow (\texttt{dStrings}[0 : j-1], \texttt{dStrings}[i], \texttt{dStrings}[j : l-2])$
7: $\mathbf{w} \leftarrow \texttt{dStrings}[0] \circ \ldots \circ \texttt{dStrings}[l-1]$
8: **return** $\mathbf{w}[:-1]$

there are no elements of `dStrings` remains to be compared (line 2), and its output satisfies the following.

Theorem 2. Algorithm 3 outputs an array `dStrings` of l pairwise distinct binary strings of length m.

Proof. Assume, for sake of contradiction, that there exist $a, b \in [0, l-1]$ and $a < b$ such that $\texttt{dStrings}[a] = \texttt{dStrings}[b]$. There are two possible cases for $\texttt{dStrings}[b]$.

If $\texttt{dStrings}[b]$ is a new string constructed in line 12, then it is distinct from every other elements on its left by Lemma 1, a contradiction. If $\texttt{dStrings}[b]$ is not a new string, then the $\texttt{dStrings}[a]$ on its left is not as well. As such, $\texttt{dStrings}[b]$ would have been deleted in line 6 when $i = a$ and $j = b$, a contradiction. □

A.2 Decoding

We proceed to introduce the decoding procedure in Algorithm 4, which is essentially the inverse operation of the encoding process. Given the array `dStrings`, the decoding algorithm reads i, j' from the rightmost element if it is a constructed new string, i.e., if last bit of which is 0 (line 1). Recall that a new string is created when two identical strings is found, with i being the index of the first one (the reference) and j' being the *alternative* index of the second (the referent).

Line 2 reads the value of i, and line 3 reads the value of j'. Recall that j' is the index of the j-th available slots in a row. Hence, the variable j is initially set to j', and then subtracted by the number of unavailable slots with indices less than j' to reaches the actual index of the referent (line 4–5). Together, i, j enable the recovery of the referent, and the rightmost element is deleted (line 6). Once all new strings have been consumed in the aforementioned process, the decoding

is concluded and $\mathbf{w}$ is returned (line 8).

B Proof of Theorem 1

The crux of proving Theorem 1 is showing that the decoder is able to obtain the key-value store `next` from unordered and partially missing fragments which result from breaking a codeword $\mathbf{c}$ at t arbitrary places and hiding fragments whose total length is at most s bits. That is, the Reed-Solomon decoding in line 9 of Algorithm 2 concludes successfully—it is well known that this requires the number of erasures plus twice the number of errors to be at most 4α in the RS codeword

$$(\texttt{approxNext}, \mathbf{r}_1, \ldots, \mathbf{r}_{4\alpha}). \tag{6}$$

Recall that every α-BRC codeword $\mathbf{c}$ is a concatenation of $\mathcal{C}_{\mathrm{MU}}$ codewords, which start with $\lceil \log m \rceil + 1$ consecutive zeros (see Section 4.1), and redundancy packets, which are free of zero runs of length $\lceil \log m \rceil + 1$ or more thanks to the RLL encoding. Hence, every discernible $\mathcal{C}_{\mathrm{MU}}$ codeword in $\mathbf{c}$ does not overlap with redundancy packets, and the decoder does not confuse the two.

The following lemma counts the number of erasures in (6), which equals to the number of redundancy strings that the decoder fails to obtain from the fragments.

Lemma 2. Let t_2 be the number of breaks that fall in the redundancy region, or separate the redundancy regions from the information region, and let s_2 be the number of missing bits that originally reside in the redundancy region. Then, the number of redundancy strings that the decoder fails to obtain is at most $4[t_2 + s_2/(5m + \lceil \log m \rceil + 15)]$.

Proof. The decoding algorithm may fail to obtain a redundancy packet due to exactly one of the following reasons.

1. There exists a break either in the packet itself, or in its preceding marker.
2. The packet, as well as its preceding marker, wholly resides in a missing fragment.

Since t_2 breaks occur in the redundancy region, there are at most t_2 missing redundant packets due to the first reason.

Recall that a redundancy packet and a marker add up to $5m + \lceil \log m \rceil + 15$ bits. Then, for a missing fragment $\mathbf{f}$ that resides in the redundancy region (if $\mathbf{f}$ cross both regions, we only consider the part in the redundancy region), at most

$$|\mathbf{f}|/(5m + \lceil \log m \rceil + 15)$$

packets, together with their preceding markers, are lost due to the second reason. Together, the number of missing redundant packets is no more than

$$\begin{aligned} &t_2 + \sum_{\mathbf{f} \text{ in redundancy region}} \frac{|\mathbf{f}|}{5m + \lceil \log m \rceil + 15} \\ &\leq t_2 + s_2/(5m + \lceil \log m \rceil + 15). \end{aligned}$$

Finally, since one redundancy packet contains four redundancy strings, it follows that the decoder fails to obtain at most $4(t_2 + s_2/(5m + \lceil \log m \rceil + 15))$ redundancy strings. □

We continue to count the number of errors in (6), i.e., the number of entries in which `approxNext` and `next` differ.

Lemma 3. Let t_1 be the number of breaks that occur in the information region, and s_1 be the number of missing bits that originally reside in the information region. Then, the number of entries in which `approxNext` and `next` differ is at most

$$2 \cdot t_1 + s_1/(m + \lceil \log m \rceil + 4). \tag{7}$$

Proof. Observe that the decoding algorithm may fail to find a non-marker (i.e., $\mathbf{u}_i$ for $i \geq t$) in the information region due to exactly one of the following reasons.

1. There exists a break in it.
2. It wholly resides in a missing fragment.

Notice that, failing to capture a non-marker $\mathbf{u}_u$ affects at most two key-value pairs of `next`, i.e., $\texttt{next}[\mathbf{u}_{u-1}]$ and $\texttt{next}[\mathbf{u}_{u+1}]$, and hence `next` and `approxNext` differ in at most $2t_1$ positions due to the first reason.

Recall that the code length of $\mathcal{C}_{\mathrm{MU}}$ is $m + \lceil \log m \rceil + 4$. Hence, for a missing fragment $\mathbf{f}$ that resides in the information region (if $\mathbf{f}$ cross both regions, we only consider the part in the information region), there are at most

$$|\mathbf{f}|/(m + \lceil \log m \rceil + 4)$$

consecutive non-markers being lost due to the second reason. This leads to at most

$$|\mathbf{f}|/(m + \lceil \log m \rceil + 4) - 1$$

different entries in `approxNext` in addition to the ones caused by breaks. Together, the number of entries that `approxNext` and `next` differ is at most

$$\begin{aligned} &2t_1 + \sum_{\mathbf{f} \text{ in information region}} \frac{|\mathbf{f}|}{m + \lceil \log m \rceil + 4} - 1 \\ &\leq 2t_1 + s_1/(m + \lceil \log m \rceil + 4). \end{aligned}$$ □

Now, by Lemma 2 and Lemma 3, the sum of the number of errors and twice the number of erasures in (6) is no more than

$$\begin{aligned} &4(t_1 + t_2) + 4s_2/(5m + \lceil \log m \rceil + 15) + 2s_1/(m + \lceil \log m \rceil + 4) \\ &\leq 4t + 2s_2(m + \lceil \log m \rceil + 4) + 2s_1/(m + \lceil \log m \rceil + 4) \\ &\leq 4t + 2s/(m + \lceil \log m \rceil + 4) \leq 4\alpha, \end{aligned}$$

where the last inequality follows from (3). The proof is concluded since line 9 outputs the correct key-value store `next`, given that a $(2^m + 4\alpha, 2^m)$ Reed-Solomon code can simultaneously correct x errors and y erasures provided that $2x + y \leq 4\alpha$.

C Simulation Results

This section presents simulation results on the success rate of fingerprint recovery using α-BRC across three models.

	1-BRC, 0.613 mm/bit				2-BRC, 0.485 mm/bit				3-BRC, 0.401 mm/bit				4-BRC, 0.342 mm/bit			
β \ ρ	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%
10	81.25%	45.41%	13.87%	0.78%	100.00%	83.15%	43.24%	6.67%	100.00%	97.27%	76.71%	21.07%	100.00%	99.54%	89.67%	37.26%
20	60.16%	25.98%	5.10%	0.17%	99.22%	85.82%	46.24%	4.08%	100.00%	99.71%	89.36%	24.80%	100.00%	99.90%	97.05%	48.36%
30	32.03%	11.06%	1.27%	0.00%	100.00%	85.35%	40.38%	2.66%	100.00%	99.68%	90.11%	24.41%	100.00%	100.00%	98.56%	55.25%
40	10.94%	3.34%	0.56%	0.00%	96.09%	75.46%	32.71%	1.64%	100.00%	99.49%	88.13%	23.97%	100.00%	99.98%	99.00%	57.52%
50	5.47%	1.17%	0.07%	0.00%	86.72%	61.62%	20.43%	0.32%	100.00%	99.58%	88.11%	18.99%	100.00%	100.00%	99.29%	58.30%
60	0.00%	0.00%	0.00%	0.00%	78.91%	43.09%	10.52%	0.27%	100.00%	99.44%	83.54%	14.77%	100.00%	99.98%	98.80%	53.00%
70	0.00%	0.00%	0.00%	0.00%	54.69%	26.05%	4.83%	0.20%	100.00%	98.90%	79.42%	12.13%	100.00%	100.00%	99.02%	50.34%
80	0.00%	0.00%	0.00%	0.00%	50.78%	20.95%	3.56%	0.05%	100.00%	97.68%	71.78%	8.11%	100.00%	99.98%	98.17%	46.80%
90	0.00%	0.00%	0.00%	0.00%	37.50%	13.04%	2.17%	0.02%	100.00%	95.97%	64.60%	5.27%	100.00%	99.98%	98.24%	43.12%
100	0.00%	0.00%	0.00%	0.00%	14.06%	4.25%	0.29%	0.00%	100.00%	90.70%	53.10%	3.64%	100.00%	99.93%	96.75%	33.59%
	5-BRC, 0.298 mm/bit				6-BRC, 0.264 mm/bit				7-BRC, 0.237 mm/bit				8-BRC, 0.215 mm/bit			
β \ ρ	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%
10	100.00%	99.80%	96.00%	56.30%	100.00%	100.00%	97.63%	66.60%	100.00%	99.95%	99.05%	74.51%	100.00%	100.00%	99.61%	81.74%
20	100.00%	100.00%	99.51%	74.41%	100.00%	100.00%	99.73%	84.30%	100.00%	100.00%	99.95%	92.19%	100.00%	100.00%	100.00%	95.95%
30	100.00%	100.00%	99.93%	82.54%	100.00%	100.00%	99.98%	93.16%	100.00%	100.00%	100.00%	97.53%	100.00%	100.00%	100.00%	99.17%
40	100.00%	100.00%	99.98%	87.52%	100.00%	100.00%	100.00%	96.29%	100.00%	100.00%	100.00%	98.58%	100.00%	100.00%	100.00%	99.80%
50	100.00%	100.00%	99.93%	88.94%	100.00%	100.00%	100.00%	97.53%	100.00%	100.00%	100.00%	99.49%	100.00%	100.00%	100.00%	99.95%
60	100.00%	100.00%	99.98%	87.99%	100.00%	100.00%	100.00%	98.24%	100.00%	100.00%	100.00%	99.71%	100.00%	100.00%	100.00%	99.95%
70	100.00%	100.00%	100.00%	90.04%	100.00%	100.00%	100.00%	98.95%	100.00%	100.00%	100.00%	99.95%	100.00%	100.00%	100.00%	100.00%
80	100.00%	100.00%	100.00%	89.70%	100.00%	100.00%	100.00%	99.19%	100.00%	100.00%	100.00%	99.95%	100.00%	100.00%	100.00%	99.98%
90	100.00%	100.00%	99.98%	87.94%	100.00%	100.00%	100.00%	98.80%	100.00%	100.00%	100.00%	99.88%	100.00%	100.00%	100.00%	100.00%
100	100.00%	100.00%	99.93%	85.28%	100.00%	100.00%	100.00%	99.07%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

Table 5: Simulation results of the FMDA Glock Frame.

	1-BRC, 0.653 mm/bit				2-BRC, 0.5168 mm/bit				3-BRC, 0.427 mm/bit				4-BRC, 0.364 mm/bit			
β \ ρ	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%
10	80.47%	42.77%	10.55%	0.44%	100.00%	89.14%	53.91%	8.94%	100.00%	97.19%	78.37%	23.56%	100.00%	99.10%	88.96%	41.82%
20	67.97%	31.69%	6.71%	0.20%	100.00%	95.39%	63.92%	8.52%	100.00%	99.73%	92.99%	36.82%	100.00%	99.98%	98.29%	60.50%
30	44.53%	16.70%	2.10%	0.02%	100.00%	93.92%	59.42%	5.30%	100.00%	99.95%	95.92%	36.35%	100.00%	100.00%	99.44%	68.36%
40	30.47%	10.55%	1.20%	0.00%	99.22%	91.26%	50.76%	3.71%	100.00%	100.00%	96.95%	36.38%	100.00%	100.00%	99.85%	75.17%
50	15.62%	4.88%	0.44%	0.02%	98.44%	85.33%	40.62%	1.93%	100.00%	99.95%	95.53%	31.62%	100.00%	100.00%	99.95%	73.32%
60	7.81%	1.49%	0.10%	0.00%	96.09%	78.15%	30.62%	1.15%	100.00%	99.95%	96.39%	31.05%	100.00%	100.00%	99.78%	72.36%
70	2.34%	0.51%	0.00%	0.00%	95.31%	69.09%	21.56%	0.29%	100.00%	100.00%	94.78%	26.10%	100.00%	100.00%	99.83%	70.70%
80	0.00%	0.00%	0.00%	0.00%	88.28%	55.64%	13.94%	0.27%	100.00%	99.98%	91.85%	20.14%	100.00%	100.00%	99.93%	67.07%
90	0.00%	0.00%	0.00%	0.00%	75.78%	42.65%	7.89%	0.00%	100.00%	99.63%	88.13%	15.84%	100.00%	100.00%	99.68%	63.35%
100	0.00%	0.00%	0.00%	0.00%	67.97%	31.10%	4.54%	0.02%	100.00%	99.63%	84.64%	11.47%	100.00%	100.00%	99.78%	61.60%
	5-BRC, 0.318 mm/bit				6-BRC, 0.281 mm/bit				7-BRC, 0.253 mm/bit				8-BRC, 0.229 mm/bit			
β \ ρ	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%
10	100.00%	99.39%	93.92%	53.44%	100.00%	99.66%	96.75%	62.30%	100.00%	99.85%	97.63%	70.87%	100.00%	99.61%	98.34%	76.86%
20	100.00%	99.98%	99.68%	78.27%	100.00%	100.00%	99.98%	87.48%	100.00%	100.00%	99.98%	92.41%	100.00%	100.00%	100.00%	95.48%
30	100.00%	100.00%	99.95%	86.04%	100.00%	100.00%	99.98%	94.41%	100.00%	100.00%	100.00%	97.49%	100.00%	100.00%	100.00%	98.75%
40	100.00%	100.00%	100.00%	90.58%	100.00%	100.00%	100.00%	97.53%	100.00%	100.00%	100.00%	99.54%	100.00%	100.00%	100.00%	99.83%
50	100.00%	100.00%	100.00%	91.80%	100.00%	100.00%	100.00%	98.12%	100.00%	100.00%	100.00%	99.66%	100.00%	100.00%	100.00%	99.88%
60	100.00%	100.00%	100.00%	93.77%	100.00%	100.00%	100.00%	98.54%	100.00%	100.00%	100.00%	99.90%	100.00%	100.00%	100.00%	99.98%
70	100.00%	100.00%	100.00%	92.77%	100.00%	100.00%	100.00%	98.97%	100.00%	100.00%	100.00%	99.80%	100.00%	100.00%	100.00%	99.98%
80	100.00%	100.00%	100.00%	93.48%	100.00%	100.00%	100.00%	99.10%	100.00%	100.00%	100.00%	99.88%	100.00%	100.00%	100.00%	100.00%
90	100.00%	100.00%	100.00%	93.31%	100.00%	100.00%	100.00%	99.44%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
100	100.00%	100.00%	100.00%	93.09%	100.00%	100.00%	100.00%	99.24%	100.00%	100.00%	100.00%	99.98%	100.00%	100.00%	100.00%	100.00%

Table 6: Simulation results of AR-15 lower receiver.

	1-BRC, 0.161 mm/bit				2-BRC, 0.127 mm/bit				3-BRC, 0.105 mm/bit				4-BRC, 0.090 mm/bit			
β \ ρ	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%
10	100.00%	78.34%	53.69%	20.41%	100.00%	92.63%	74.68%	39.60%	100.00%	94.19%	77.69%	44.75%	100.00%	95.65%	81.69%	49.51%
20	100.00%	79.91%	56.13%	19.97%	100.00%	96.78%	81.64%	43.51%	100.00%	97.80%	88.70%	57.37%	100.00%	98.27%	90.84%	62.26%
30	100.00%	83.23%	57.28%	18.53%	100.00%	98.85%	89.18%	48.44%	100.00%	99.29%	93.87%	66.09%	100.00%	99.76%	95.70%	73.36%
40	100.00%	84.23%	57.32%	19.60%	100.00%	99.41%	91.43%	49.66%	100.00%	99.83%	95.95%	70.80%	100.00%	99.93%	97.53%	79.71%
50	99.22%	84.03%	56.35%	16.21%	100.00%	99.80%	92.75%	50.88%	100.00%	99.98%	97.78%	74.83%	100.00%	99.93%	98.68%	83.67%
60	98.44%	83.98%	53.76%	14.60%	100.00%	99.76%	93.02%	51.44%	100.00%	99.93%	98.41%	77.10%	100.00%	99.98%	99.19%	86.33%
70	96.09%	79.42%	48.00%	11.52%	100.00%	99.83%	93.41%	49.95%	100.00%	99.98%	98.90%	79.66%	100.00%	100.00%	99.61%	88.92%
80	96.09%	78.74%	50.46%	10.57%	100.00%	99.63%	92.85%	47.73%	100.00%	100.00%	98.97%	79.17%	100.00%	100.00%	99.83%	91.33%
90	92.97%	76.07%	43.09%	8.79%	100.00%	99.58%	92.53%	46.17%	100.00%	100.00%	99.39%	81.54%	100.00%	100.00%	99.90%	92.72%
100	91.41%	70.19%	37.35%	6.54%	100.00%	99.41%	91.14%	43.51%	100.00%	100.00%	99.32%	80.52%	100.00%	100.00%	99.95%	92.65%
	5-BRC, 0.078 mm/bit				6-BRC, 0.062 mm/bit				7-BRC, 0.062 mm/bit				8-BRC, 0.056 mm/bit			
β \ ρ	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%
10	100.00%	97.31%	87.40%	58.37%	100.00%	99.73%	95.51%	71.73%	100.00%	99.98%	98.22%	82.42%	100.00%	99.98%	99.46%	89.50%
20	100.00%	99.49%	95.39%	72.34%	100.00%	100.00%	99.15%	86.82%	100.00%	100.00%	99.90%	93.21%	100.00%	100.00%	99.98%	96.36%
30	100.00%	99.80%	97.88%	81.01%	100.00%	99.98%	99.61%	90.84%	100.00%	100.00%	100.00%	97.05%	100.00%	100.00%	100.00%	98.44%
40	100.00%	99.95%	98.97%	86.82%	100.00%	100.00%	99.80%	94.87%	100.00%	100.00%	99.98%	98.56%	100.00%	100.00%	100.00%	99.46%
50	100.00%	100.00%	99.37%	90.80%	100.00%	100.00%	99.88%	96.02%	100.00%	100.00%	99.98%	99.10%	100.00%	100.00%	100.00%	99.78%
60	100.00%	100.00%	99.73%	93.92%	100.00%	100.00%	99.95%	97.73%	100.00%	100.00%	100.00%	99.41%	100.00%	100.00%	100.00%	99.90%
70	100.00%	100.00%	99.83%	95.61%	100.00%	100.00%	99.98%	98.39%	100.00%	100.00%	100.00%	99.46%	100.00%	100.00%	100.00%	99.98%
80	100.00%	100.00%	99.95%	95.95%	100.00%	100.00%	100.00%	98.97%	100.00%	100.00%	100.00%	99.78%	100.00%	100.00%	100.00%	99.93%
90	100.00%	100.00%	100.00%	97.41%	100.00%	100.00%	100.00%	99.19%	100.00%	100.00%	100.00%	99.88%	100.00%	100.00%	100.00%	100.00%
100	100.00%	100.00%	99.95%	97.63%	100.00%	100.00%	100.00%	99.44%	100.00%	100.00%	100.00%	99.90%	100.00%	100.00%	100.00%	100.00%

Table 7: Simulation results of 3DBenchy.