One Code Fits All: Strong stuck-at codes for versatile memory encoding

Roni Con[*]   Ryan Gabrys[†]   Eitan Yaakobi[‡]

January 29, 2024

Abstract

In this work we consider a generalization of the well-studied problem of coding for "stuck-at" errors, which we refer to as "strong stuck-at" codes. In the traditional framework of stuck-at codes, the task involves encoding a message into a one-dimensional binary vector. However, a certain number of the bits in this vector are 'frozen', meaning they are fixed at a predetermined value and cannot be altered by the encoder. The decoder, aware of the proportion of frozen bits but not their specific positions, is responsible for deciphering the intended message. We consider a more challenging version of this problem where the decoder does not know also the fraction of frozen bits. We construct explicit and efficient encoding and decoding algorithms that get arbitrarily close to capacity in this scenario. Furthermore, to the best of our knowledge, our construction is the first, fully explicit construction of stuck-at codes that approach capacity.

---
[*]Technion - Israel Institute of Technology, Haifa, Israel, roni.con93@gmail.com
[†]University of California San Diego, Naval Information Warfare Center, CA, rgabrys@ucsd.edu
[‡]Technion - Israel Institute of Technology, Haifa, Israel, yaakobi@cs.technion.ac.il

Contents

1 Introduction

In this research, we initiate the development of *strong-stuck-at codes*, an advanced version of traditional codes that have applications to *stuck-at memories*. Our approach considers a storage medium analogous to a one-dimensional vector with a fixed length, containing a certain proportion of 'frozen' components that cannot be altered during encoding. The objective is to create a coding system capable of encoding the greatest possible amount of information while ensuring the frozen components' values and positions, known during encoding but unknown during decoding, remain intact. Previous studies typically assume knowledge of the maximum size of the set of frozen components at the time of encoding and decoding, even if the set itself is not known. Our study addresses the more flexible (yet challenging) scenario where both the specific set and the maximum size of the frozen components are unknown at the decoding stage.

The problem of constructing codes for stuck-at memories has its roots in the early work of Kuznetsov and Tsybakov [KT74]. Building on this, Tsybakov expanded the scope by considering scenarios where, apart from the frozen components, the memory might incur additional errors post-encoding [Tsy75]. This led Heegard to innovate a new class of codes, termed partitioned linear block codes [Hee83], which he demonstrated to meet the Shannon capacity in specific conditions [Hee85]. However, these findings are not applicable to scenarios involving binary alphabet codes, which is the primary focus of our study. It's noteworthy that this issue has evolved with the advent of newer technologies like Flash and Phase-Change Memory (PCM) and some new works on this (and similar) settings include [LMJF10, KK13, WZY15, MV15].

A strongly related area of work is in the setting of coding for "Write-Once-Memories" or WOM, which was originally introduced by Rivest and Shamir in 1982 [RS82]. In this setting, memory cells are initialized to each have value 0 and, at each round of the encoding, one is allowed to change some fraction of the cells only from 0 to 1. For the case of two-write WOM-codes, in the first round the encoder is permitted to change any fraction of the cells to 1. The decoding in the first round is straightforward. In the second round, the encoder has access to the state of the memory after the first round so that it knows which cells were set to one in the first round, but the decoder only has access to the state of the memory after the second write and so it does not know what bits were set to 1 in the first round. Thus, the second round of encoding/decoding represents an instance of the defective memory with stuck-at components.

Capacity-achieving two-write WOM-codes have been known for some time starting with the seminal work by Sphilka [Shp13] and later by Chee et al. [CKVY19]. In fact, using this connection between two-write WOM-codes and coding for stuck at errors, it was noted in [Shp13] that if the encoder is allowed to transmit a small amount of side information directly to the decoder that cannot be corrupted by stuck-at errors, then a slight variation of the encoder/decoder for his two-write WOM is equivalent to a stuck-at code. In the work by Chee et al. [CKVY19], which leverages spreads in projective geometry in order to guide the encoding function for the second round write, the value of the cells matters so that it is not clear how to make their approach account for frozen or stuck-at cells that can have value 0 or 1.

Perhaps the closest existing work to the problem of designing strong-stuck-at codes is the work Gabizon and Shaltiel [GS12], who designed capacity-achieving stuck-at codes

for the case where the maximum number of frozen components is known ahead of time. Although their constructions provided the first explicit scheme with asymptotically optimal rate, their model permitted a randomized encoding function which was allowed to succeed with randomized polynomial time (with respect to the block length of the memory).

In this work, we develop almost capacity-achieving strong-stuck-at codes where the number of frozen components is not known beforehand. Although our primary goal is the design of explicit and efficient codes for this generalized model, our codes also have several properties for the classical stuck-at model. Unlike the work of Gabizon and Shaltiel, our encoding procedure is completely deterministic (see Theorem 1.5). Furthermore, we show that in the randomized version of our algorithm which is presented in Section 3, we are able to construct codes using fewer random bits than in previous constructions.

The rest of this paper is organized as follows. In the remainder of this section, we formally introduce our problem setup and highlight our results. In Section 2, we present an existential result showing that, perhaps surprisingly, it is possible to encode at virtually the same rate as a conventional stuck-at code even when the size (or a bound on the size) of the set of frozen components is not available to the decoder. Section 3 presents a simplified version of our construction where we assume the encoder is provided a side channel to convey a small amount of information to the decoder in a manner analogous to the setting originally studied in [Shp13]. Finally Section 4 presents our main construction.

1.1 Problem setup

Denote $\mathbb{P}^{(i)}([N]) := \{\mathcal{F} \subseteq [N] \mid |\mathcal{F}| = i\}$, i.e., all the subsets of $[N]$ of size i. Formally, our goal is to design

1. A sequence of pairs $E := (E_1, \mathcal{M}_1), (E_2, \mathcal{M}_2), \ldots, (E_N, \mathcal{M}_N)$ where the $\mathcal{M}_i$'s are sets of messages and

$$E_i : \{0,1\}^N \times \mathbb{P}^{(i)}([N]) \times \mathcal{M}_i \to \{0,1\}^N$$

   are encoding maps that get as input a cover vector $\boldsymbol{v} \in \{0,1\}^N$, a set of frozen indices of size i, and a message to encode.

2. A decoder

$$D : \{0,1\}^N \to \bigcup_{i=1}^N \mathcal{M}_i$$

   that maps vectors to messages.

A *strong-stuck-at-code* of *length* N is a pair (E, D) such that for every $i \in [N]$, $\boldsymbol{v} \in \{0,1\}^N$, $\mathcal{F} \in \mathbb{P}^i([N])$, and $\boldsymbol{m} \in \mathcal{M}_i$, the following two conditions hold:

1. Consistency:
$$(E_i(\boldsymbol{v}, \mathcal{F}, \boldsymbol{m}))_j = \boldsymbol{v}_j, \quad \forall j \in \mathcal{F} .$$

   Namely, the encoders are allowed to change only coordinates of $\boldsymbol{v}$ whose indices are outside of $\mathcal{F}$.

4

2. Unique-decodability:
$$D(E_i(\boldsymbol{v}, \mathcal{F}, \boldsymbol{m})) = \boldsymbol{m} \ .$$

Definition 1.1. *The rate of a strong-stuck-at-code at ρ-fraction defect is defined as*

$$\frac{\log\left(|\mathcal{M}_{\rho N}|\right)}{N} \ .$$

Naturally, given that ρN of the bits are frozen, we can encode up to $1 - \rho$ fraction of information bits. Our goal in this paper is to design codes that approach this bound. This goal motivates the following code definition.

Definition 1.2. *Let $\varepsilon > 0$. An ε-gapped strong-stuck-at-code of length N is a strong stuck-at code such that for every defect fraction $\rho \in (0, 1 - \varepsilon)$, the rate of the code is at least $1 - \rho - \varepsilon$.*

1.2 Our results

In the following theorem, we show that there are ε-gapped strong-stuck-at-code.

Theorem 1.3. *For every $\varepsilon > 0$, there exists an $N(\varepsilon)$ such that for every $N > N(\varepsilon)$, there exists ε-gapped strong-stuck-at-code of length N.*

Our next theorem presents a randomized construction of ε-gapped strong-stuck-at-code.

Theorem 1.4. *For every $\varepsilon > 0$, there exists an $N(\varepsilon)$ such that for every $N > N(\varepsilon)$, there exists a randomized ε-gapped strong-stuck-at-code of length N such that*

1. *The encoder and the decoder run in $\mathcal{O}\left(N \cdot \mathrm{poly}(\log n) \cdot \mathrm{poly}(1/\varepsilon)\right)$.*

2. *The number of random bits that are used by the encoder is $\mathcal{O}\left(\frac{1}{\varepsilon} \log N\right)$ and the encoder succeeds with probability $1 - o(1)$.*

Our next theorem is a version of Theorem 1.4 that is fully deterministic. We note that the cost of making the encoder deterministic results in much higher encoding complexity.

Theorem 1.5. *For every $\varepsilon > 0$, there exists an $N(\varepsilon)$ such that for every $N > N(\varepsilon)$, there exists an explicit ε-gapped strong-stuck-at-code of length N such that the encoder runs in time $N^{\mathcal{O}(1/\varepsilon)}$ and the decoder runs in time $\mathcal{O}\left(N \cdot \mathrm{poly}(\log n) \cdot \mathrm{poly}(1/\varepsilon)\right)$.*

1.3 Preliminaries

For an integer k, we denote $[k] := \{1, 2, \ldots, k\}$. We shall denote vectors by boldface letters $\boldsymbol{u}$, sets by calligraphic letters such as $\mathcal{F}$. Note that we denote an interval of positive integers by $[a, b]$ and a vector restricted to a set of coordinates will be denoted by $\boldsymbol{v}_{\mathcal{F}}$. In particular, a subvector of $\boldsymbol{v}$ starting from index i up to an index j will be denoted as $\boldsymbol{v}_{[a,b]}$. We shall denote a concatenation of two vectors, $\boldsymbol{u}$ and $\boldsymbol{v}$ by $\boldsymbol{u} \circ \boldsymbol{v}$. Throughout this paper, $\log x$ will refer to the base-2 logarithm.

We note here that in many places we drop all floors and ceilings in order to ease notation and the analysis of the codes. However, the loss in the rate due to these roundings is negligible and does not affect the asymptotic results.

A concept that will be useful in our construction is that of *almost k-wise independent* random variables.

Definition 1.6. *A random variable* $X = (X_1, X_2, \ldots, X_r) \in \{0, 1\}^r$ *is said to be μ-almost k-wise independent if for all sets of k distinct indices $\{i_1, \ldots, i_k\} \subseteq [r]$ and for all $(x_1, x_2, \ldots, x_k) \in \{0, 1\}^r$, we have*

$$\left| \Pr[X_{i_1} = x_1, \ldots, X_{i_k} = x_k] - 2^{-k} \right| \leq \mu .$$

The following well-known result gives an efficient construction of a collection of μ-almost k-wise independent random variables which can be generated from a small number of random bits.

Lemma 1.7 ([AGHP92]). *For every two positive integers r, k and every $\mu > 0$, there exists a function $g : \{0, 1\}^t \to \{0, 1\}^r$ with $t = \mathcal{O}\left(\log\left(\frac{k \log r}{\mu} \right) \right)$, such that $g(U_t)$ is a μ-almost k-wise independent variable over $\{0, 1\}^r$, where U_t denotes the uniform distribution over $\{0, 1\}^t$. Moreover, $g(\boldsymbol{u})$ can be computed in time $\mathrm{poly}(r, 1/\mu)$.*

Remark 1.8. *We shall use Lemma 1.7 with $r = \mathcal{O}(N \log N)$, $k = \mathcal{O}(\log N)$, and $\mu = N^{\mathcal{O}(1)}$. In this case, we have that $t = \mathcal{O}(\log N)$. Furthermore, it can be verified that in this case, the running time of g on an input $\boldsymbol{u} \in \{0, 1\}^t$ is $\mathcal{O}(N \cdot \mathrm{poly}(\log N))$. The details are given in the appendix.*

We have the following simple claim whose proof is deferred to the appendix.

Claim 1.9. *Let $m < n$ be positive integers and*

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{pmatrix}$$

where $(A_{i,j})_{1 \leq i \leq m, 1 \leq j \leq n}$ is a μ-almost n-wise independent variable. Then, the probability that A does not have full rank is most $2^{m-n} + \mu 2^m$.

2 Existential result

In this section, we prove Theorem 1.3 which is restated for convenience

Theorem 1.3. *For every $\varepsilon > 0$, there exists an $N(\varepsilon)$ such that for every $N > N(\varepsilon)$, there exists ε-gapped strong-stuck-at-code of length N.*

Proof. Let L be an integer such that $L \leq 2\varepsilon^{-1} \leq L + 1$ and let N be an integer such that $L + 1$ divides N. For every $i \in [L]$, define $\mathcal{M}_i := \left\{ (i, \boldsymbol{m}) \mid \boldsymbol{m} \in \{0, 1\}^{\frac{N}{L+1} \cdot i} \right\}$. Every $\mathcal{M}_i, i \in [L]$ can be seen as a message space of a specific length, and our encoder, based on

the fraction of frozen symbols, will encode a message from the largest possible message space.

Our strategy will be to randomly assign vectors from $\{0,1\}^N$ into $\left|\cup_{i=1}^L \mathcal{M}_i\right|$ bins where each bin will be labeled $B_{i,\boldsymbol{m}}$. Formally, every $\boldsymbol{v} \in \{0,1\}^N$,

$$\Pr\left[\boldsymbol{v} \text{ is assigned to } B_{i,\boldsymbol{m}}\right] = \frac{1}{L \cdot 2^{\frac{N}{L+1} \cdot i}} .$$

Our encoder, which receives as input a vector $\boldsymbol{v} \in \{0,1\}^N$, a set $\mathcal{F} \subseteq [N]$ of size ρN performs the following:

1. Sets j to be the largest integer such that $(1-\rho)N \geq \frac{j}{L+1}N + \frac{\varepsilon}{2}N$.

2. Encodes a message $\boldsymbol{m} \in \{0,1\}^{\frac{j}{L+1}N}$ by choosing a vector $\boldsymbol{u} \in B_{j,\boldsymbol{m}}$ such that $\boldsymbol{v}_{\mathcal{F}} = \boldsymbol{u}_{\mathcal{F}}$ and will store this vector in the memory.

Note that by the choice of j, we have ensured that the gap between the length of the message and the number of unfrozen bits is at least $\varepsilon/2 \cdot N$. Clearly, the decoder who knows the partition of $\{0,1\}^N$ to the sets $B_{i,\boldsymbol{m}}$ will correctly identify the message. Thus, it remains to show that the consistency condition holds with high probability. Namely, that with high probability the second step of our encoder always succeeds.

We first compute the probability for a specific $\mathcal{F}$ of size ρN and a cover vector $\boldsymbol{v}$, there is no $\boldsymbol{u} \in B_{j,\boldsymbol{m}}$ for which $\boldsymbol{u}_{\mathcal{F}} = \boldsymbol{v}_{\mathcal{F}}$. Since there are $2^{N-|\mathcal{F}|}$ vectors $\boldsymbol{u} \in \{0,1\}^N$ such that $\boldsymbol{u}_{\mathcal{F}} = \boldsymbol{v}_{\mathcal{F}}$, the probability that none of them falls in $B_{j,\boldsymbol{m}}$ is at most

$$\left(1 - \frac{1}{L \cdot 2^{\frac{N}{L+1} \cdot j}}\right)^{2^{N-|\mathcal{F}|}} \leq \left(1 - \frac{1}{L \cdot 2^{\frac{N}{L+1} \cdot j}}\right)^{2^{\frac{j}{L+1}N + \frac{\varepsilon}{2}N}} \leq \exp\left(-\frac{1}{L}\right)^{2^{\frac{\varepsilon}{2}N}} \leq \exp\left(-\varepsilon\right)^{2^{\frac{\varepsilon}{2}N}} .$$

Now, the probability that there exists a vector $\boldsymbol{v} \in \{0,1\}^N$, a set $\mathcal{F} \subseteq [N]$ and a message $\boldsymbol{m}$ (of suitable length) such that the respective set $B_{j,\boldsymbol{m}}$ does not contain a vector that agrees with $\boldsymbol{v}$ on the coordinates specified by $\mathcal{F}$ is at most,

$$2^N \cdot 2^N \cdot 2^N \cdot \exp\left(-\varepsilon\right)^{2^{\frac{\varepsilon}{2}N}} = \exp\left(\ln 2^{3N} - \varepsilon 2^{\frac{\varepsilon}{2}N}\right) .$$

Thus, since ε is constant, the probability that our partition of $\{0,1\}^N$ to the sets $B_{i,\boldsymbol{m}}$ indeed yields a strong-stuck-at-code is at least $1 - o(1)$ (the term $o(1)$ goes to zero as N tends to infinity). For every ρ, the rate of our probabilistic construction at ρ-fraction of defect is at least $1 - \rho - \frac{1}{L+1} - \frac{\varepsilon}{2} \leq 1 - \rho - \varepsilon$. $\qquad\square$

3 Construction with clean transmission assumption

In this section, we will assume that the encoder can transmit $\mathcal{O}\left(\frac{1}{\varepsilon}\log N\right)$ bits to the decoder where this transmission is errorless. The decoder will use this clean metadata to decode the original message. This construction is a first step towards our final construction which does not assume that there is a clean transmission of bits between the encoder and the decoder. Throughout this section, we assume that C is a universal constant (independent of N) that is known both to the encoder and the decoder. Also, we denote

by $h_b(d)$ the function that takes as input an integer $d \in [0, b-1]$ and outputs its binary representation using $\lceil \log b \rceil$ bits.

Our encoding algorithm is given in Algorithm 1 and the decoding algorithm is given in Algorithm 2. In the rest of the section, we prove that

Theorem 3.1. *Let $\varepsilon > 0$. there exists an $N(\varepsilon)$ such that for every $N > N(\varepsilon)$, there exists a randomized ε-gapped strong-stuck-at-code of length N such that*

1. *The encoder uses $\mathcal{O}\left(\frac{1}{\varepsilon} \log N\right)$ random bits and succeeds with probability $1 - \mathcal{O}\left(1/\log N\right)$*

2. *The encoder can transmit $\mathcal{O}\left(\frac{1}{\varepsilon} \log N\right)$ bits to the decoder in an errorless transmission.*

3. *The encoder and the decoder run in time $\mathcal{O}\left(N \cdot \mathrm{poly}(\log N) \cdot \mathrm{poly}(1/\varepsilon)\right)$*

Comparison with [Shp13, Theorem 7.1] Note that although our primary aim is to design efficiently strong-stuck-at codes, our work represents an improvement over the setup previously studied by Shpilka [Shp13, Theorem 7.1] where we assume we have access to a small area of clean memory (equivalently, we have an errorless transmission between the encoder and the decoder) and also the decoder knows the number of stuck-at bits. The next theorem more precisely states the previous work by Shpilka, which will be useful as a basis for comparison.

Theorem 3.2. *[Shp13, Theorem 7.1] Let $\rho < 1$ and let $\boldsymbol{v} \in \{0,1\}^N$ containing ρN frozen bits. There is a randomized encoder and a deterministic decoder such that*

1. *The encoder can encode $(1 - p - \varepsilon)N$ bits for any constant $\varepsilon > 0$.*

2. *The encoder transmits $\mathcal{O}\left(\log^3 N\right)$ bits to the decoder using an errorless transmission.*

3. *The encoder runs in polynomial time in N and $1/\varepsilon$.*

Note that the construction presented in this section requires only $\mathcal{O}\left(\varepsilon^{-1} \cdot \log(N)\right)$ bits to be transmitted to the decoder in the errorless transmission compared to the $\mathcal{O}\left(\log^3 N\right)$ bits required by [Shp13]. [1]

We present also a deterministic version of Theorem 3.1

Theorem 3.3. *Let $\varepsilon > 0$. there exists an $N(\varepsilon)$ such that for every $N > N(\varepsilon)$, there exists a explicit ε-gapped strong-stuck-at-code of length N such that*

1. *The encoder can transmit $\mathcal{O}\left(\frac{1}{\varepsilon} \log N\right)$ bits to the decoder in an errorless transmission.*

2. *The encoder runs in time $N^{\mathcal{O}(1/\varepsilon)}$ and the decoder runs $\mathcal{O}\left(N \cdot \mathrm{poly}(\log N)\right)$.*

---

[1] We note that in fact, we could have defined $B = C \cdot \log(N/\log N)$. In that case, the number of random bits is $\mathcal{O}\left(\frac{1}{\varepsilon} \cdot \log(N/\log N)\right)$ at the expanse of failure probability which increases to $1 - O(1/C)$. We chose to present the first version for the sake of notations.

Notations and preliminaries for Algorithm 1 The following notations are used in Algorithm 1

- Let $B = C \cdot \log N$

- We divide $[N]$ into $M := N/B$ contiguous blocks

- Let $\mathcal{F}_i \subseteq \mathcal{F}$ denote the frozen elements that appear in the ith block and $\overline{\mathcal{F}}_i$ the nonfrozen elements in the ith block.

- Denote by $\rho_i = |F_i|/B$ the fraction of frozen symbols in ith block

We proceed with a high-level description of Algorithm 1, which consists of three steps each of which is described in the next three paragraphs.

Our encoding algorithm will encode the message into M blocks, each of length B. At Step 1, for each block i, we compute m_i, the number of message bits we will encode in the ith block. Note that some of the m_is can be zero as it can be the case that (almost) all the bits of a block are frozen. The total number of bits that are going to be encoded in the ith block is denoted by $\overline{m_i}$ and will contain m_i, another $\log N$ bits for the position of the next block to be decoded, and another $\log B$ bits that denote the number of encoded message bits in the next block. If we cannot encode message bits in the ith block (this happens if we have at most $2\log N + \log B$ unfrozen bits in the block), then we set $\overline{m_i} = 0$.

At Step 2, we generate $B \cdot N$ bits that are ε-almost B-wise independent. The first $\overline{m_1} B$ bits will form the matrix A_1, then the next $\overline{m_2} B$ bits will form the matrix A_2, etc. Overall, at the end of this step, we have M matrices $A_1, \ldots, A_M$.

At step 3, we perform the actual encoding. We only encode bits of our message in blocks for which $\overline{m_i} \neq 0$. For each such block, we solve the linear system

$$(A_i)_{\overline{\mathcal{F}}_i} \cdot \boldsymbol{w}_i = \boldsymbol{m}_i \circ h_{\log N}(i') \circ h_{\log B}(m_{i'})$$

where i' is the next block index for which $\overline{m_{i'}} \neq 0$. We note that this step might fail since it can be that $(A_i)_{\overline{\mathcal{F}}_i}$ does not have full rank. We will prove that this happens with small probability. Finally, we concatenate all the blocks to produce our encoded cover object. Also, we transmit to the decoder the metadata that he needs to decode the message (recall that we assume that this transmission is errorless). This metadata includes the string that generates the matrices $A_1, \ldots, A_M$, the position of the first block that encodes message bits, and the number of message bits that are encoded in that block.

3.1   Analysis

Rate. The number of message bits that we encode in each block i, is $m_i = \max(B(1 - \rho_i) - 2\log N - \log B, 0)$. Thus, the number of bits that we can encode is at least

$$\sum_{i=1}^{M} m_i \geq \sum_{i=1}^{M} B(1 - \rho_i) - 2\log N - \log B$$
$$= N - |\mathcal{F}| - 2M\log N - M\log B$$

Algorithm 1: Encoding with assumption

> **input** : A vector $\boldsymbol{v} \in \{0,1\}^N$, a set of frozen indices $\mathcal{F} \subseteq [N], |\mathcal{F}| = \rho N$, and message $\boldsymbol{m} \in \{0,1\}^m$ where $m \leq N\left(1 - \rho - \frac{2}{C} - \frac{\log(C\log N)}{C\log N}\right)$
>
> **output** : A vector $\boldsymbol{w} \in \{0,1\}^N$ and $\boldsymbol{u} \in \{0,1\}^*$.

[1] **for** *every* $i \in [M]$ **do**

> **if** $B(1 - \rho_i) > 2\log N + \log B$ **then**
>
> > Set $m_i := \min\left(B(1 - \rho_i) - 2\log N - \log B, m\right)$
> > Set $\overline{m_i} := m_i + \log N + \log B$
> > Update $m = m - m_i$
>
> **else**
>
> > Set $\overline{m_i} := 0$

end

[2] Let $r = B \cdot N$, $\mu = N^{-C}$, $k = B$ and let t be as given in Lemma 1.7. Sample $\boldsymbol{u}_t$ uniformly at random from $\{0,1\}^t$ and apply the function g (given in Lemma 1.7) to get $\boldsymbol{a} \in \{0,1\}^{B \cdot N}$. Use the first $B \cdot m$ bits of $\boldsymbol{a}$ to construct M matrices

$$A_1 \in \{0,1\}^{\overline{m_1} \times B}, \ldots, A_M \in \{0,1\}^{\overline{m_M} \times B}$$

[3] Let $1 \leq i_1 < \cdots < i_{M'} \leq M$ be all the indices for which $\overline{m_{i_j}} \neq 0$. Also let $i_{M'+1} = 0$ and $m_{i_{M'+1}} = 0$

for *every* $j \in [M']$ **do**

> **if** $(A_{i_j})_{\overline{\mathcal{F}}_{i_j}}$ *is not full rank* **then**
>
> > Declare failure and exit
>
> **end**
>
> Compute $\boldsymbol{w}_{i_j} \in \{0,1\}^B$ such that
>
> > 1. $A_{i_j} \cdot \boldsymbol{w}_{i_j} = \boldsymbol{m}_{i_j} \circ h_{N/\log N}(i_{j+1}) \circ h_B(m_{i_{j+1}})$
> >
> > 2. $(\boldsymbol{w}_{i_j})_{F_{i_j}} = (\boldsymbol{v}_{i_j})_{F_{i_j}}$

end

[4] Return the string $\boldsymbol{w} = \boldsymbol{w}_1 \circ \boldsymbol{w}_2 \circ \cdots \circ \boldsymbol{w}_M$ and the string $\boldsymbol{u} = \boldsymbol{u}_t \circ h_{N/\log N}(i_1) \circ h_B(m_1)$

Algorithm 2: Decoding with assumption

> **input** : A vector $\boldsymbol{v} \in \{0,1\}^N$ and $\boldsymbol{u} \in \{0,1\}^{t+\log B}$
>
> **output** : A message $\boldsymbol{m} \in \{0,1\}^*$

[1] Identify from $\boldsymbol{u}$ the vector $\boldsymbol{u}_t$, and the values i and m_i

[2] Compute $g(\boldsymbol{u}_t)$ to get a string $\boldsymbol{a} \in \{0,1\}^{B \cdot N}$

[3] **while** $i \neq 0$ **do**

> Identify the matrix $A_i \in \{0,1\}^{m_i \times B}$ from the string $\boldsymbol{a}$
> Compute $A_i \boldsymbol{v}_i$ to get $\boldsymbol{m}_i$ and update the next i and m_i

end

[4] Return $\boldsymbol{m} = \boldsymbol{m}_1 \circ \cdots \circ \boldsymbol{m}_M$

---

and therefore, the rate of the scheme is at least

$$1 - \rho - \frac{2}{C} - \frac{\log(C \log N)}{C \log N} \geq 1 - \rho - \frac{3}{C} \ , \tag{1}$$

where the inequality follows for large enough N.

The following proposition proves the correctness of the algorithm.

Proposition 3.4. *Let $\boldsymbol{v} \in \{0,1\}^N$, $\mathcal{F} \subseteq [N]$ where $|\mathcal{F}| = \rho N$. Let $\boldsymbol{m} \in \{0,1\}^m$ where $m \leq N(1 - \rho - \frac{3}{C})$ be a message to be encoded. If we execute Algorithm 1 on $\boldsymbol{v}$, $\mathcal{F}$, and $\boldsymbol{m}$, then the following holds*

1. *The algorithm succeeds with probability at least $1 - \mathcal{O}\left(1/(C \log N)\right)$. Specifically, the only step that might cause the algorithm to fail and abort is Step 3.*

2. *If the algorithm succeeds and outputs the vector $\boldsymbol{w}$ and the metadata $\boldsymbol{u}$, then the decoding algorithm, Algorithm 2, which receives $\boldsymbol{w}$ and $\boldsymbol{u}$ as input, will output $\boldsymbol{m}$.*

Proof. Recall that we denote by $\mathcal{F}_i \subseteq \mathcal{F}$ the frozen elements that appear in the ith block and by $\overline{\mathcal{F}}_i$ the nonfrozen elements that are in the ith block of $\boldsymbol{v}$. The step that might fail in Algorithm 1 and cause the algorithm to abort is Step 3. If one of the matrices $(A_1)_{\overline{\mathcal{F}}_1}, \ldots, (A_M)_{\overline{\mathcal{F}}_M}$ does not have full rank, say $(A_1)_{\overline{\mathcal{F}}_1}$, then clearly we have that $\{(A_1)_{\overline{\mathcal{F}}_1} \cdot \boldsymbol{w} \mid \boldsymbol{w} \in \{0,1\}^{B - \rho_1 B}\} \subsetneq \{0,1\}^{\overline{m_1}}$. Thus, there exists a vector in $\{0,1\}^{\overline{m_1}}$ that cannot be encoded using this procedure. Therefore, in order to be able to encode any message, we must require that the matrices $(A_i)_{\overline{\mathcal{F}}_i}$ each have full rank.

We compute the probability that $(A_1)_{\overline{\mathcal{F}}_1}$ does not have full rank. Note that $(A_1)_{\overline{\mathcal{F}}_1} \in \{0,1\}^{\overline{m_1} \times (B - \rho_1 B)}$ where $\overline{m_1} \leq B - \rho_1 B - \log N$. Also, it is easy to see that a random variable that is ε-almost k-wise independent, is also $(2^{k-k'} \varepsilon)$-almost k'-independent for every $k' < k$. Thus, according to Claim 1.9, the probability that $(A_1)_{\overline{\mathcal{F}}_1}$ does not have full rank is at most

$$2^{-\log N} + \mu \cdot 2^{B \rho_1} \cdot 2^{B(1 - \rho_1) - \log N} = \frac{2}{N} \ .$$

Now by union bound, the probability that there exists a matrix among the matrices $(A_1)_{\overline{\mathcal{F}}_1}, (A_2)_{\overline{\mathcal{F}}_2}, \ldots, (A_M)_{\overline{\mathcal{F}}_M}$ that does not have full rank is at most $M \cdot (2/N) = 2/(C \log N) = \mathcal{O}\left(1/(C \log N)\right)$. Therefore, Step 3 can fail with probability at most $1 - \mathcal{O}\left(1/(C \log N)\right)$. If all the matrices are indeed full rank, then the linear equations at Step 3 all have solutions and therefore, the encoded vector $\boldsymbol{w} \in \{0,1\}^N$ is just the concatenation of all the $\boldsymbol{w}_i$s. The second output of the encoder is a vector $\boldsymbol{u}$ which concatenates the string $\boldsymbol{u}_t$ generated in Step 2 with the position of the minimal $i \in [M]$ for which $\overline{m_i} \neq 0$ and the corresponding m_i. The last two values correspond to the position of the first block that encodes message bits and the number of message bits that are encoded in this block, respectively.

Note that the decoder, which has access to $\boldsymbol{u}$ and knows the value C can read the first t bits to identify $\boldsymbol{u}_t$. Then, reading the following $\log N + \log B$ bits, the decoder knows the identity of the first block, $i \in [M]$, that encodes message bits and the exact number of bits m_i the block encodes. Note here that $i \in [M]$ where $M = N/(C \log N)$ and that $m_i \leq B - 2 \log N - \log B$, therefore, $\log N$ followed by $\log B$ bits suffice in order to save i and m_i, respectively.

11

Computing $g(\boldsymbol{u}_t)$, the decoder identifies the matrix A_i and then simply computes $A_i \boldsymbol{w}_i$ to get $\boldsymbol{m}_i$ and the position of the next block that contains information, $i < j$, and the number of encoded bits in $\boldsymbol{w}_j$. The decoder continues until he reaches the last block containing information. Note that this process stops when the decoder decodes the last block, he encounters that 0 is encoded as the position of the next block that contains information. □

We are now ready to prove Theorem 3.1.

Proof of Theorem 3.1. Let C be a universal constant and define $\varepsilon = 3/C$. Let N be a large enough integer (depending only on ε) and let $\boldsymbol{v} \in \{0,1\}^N$ be a cover vector, and $\mathcal{F} \subset [N]$ be a set of frozen sets of size ρN where $\rho \in (1 - \varepsilon, 0)$. Then, according to Proposition 3.4, we can encode any $m \in \{0,1\}^{(1-\rho-\varepsilon)}$. Note that regardless of ρ, the fraction of stuck-at bits, our decoding algorithm always succeeds in decoding the encoded message given an encoded vector and the metadata that was generated by Algorithm 2.

The rest of the proof analyzes the complexity and the metadata size that is transferred to the decoder using an errorless transmission. Clearly, 1 of the encoding algorithm takes $\mathcal{O}(N)$ as we just scan the input cover vector and identify the sets of frozen components in each block. The time complexity of Step 2 is the time that it takes to compute a value of the function g which is $\mathcal{O}(N \cdot \mathrm{poly}(\log N))$. In Step 3, we solve $M = \mathcal{O}(\varepsilon \cdot N/\log N)$ linear equation systems, each contains at most $B = \mathcal{O}(\varepsilon^{-1}\log N)$ equations. Thus, this step can be performed in $\mathcal{O}(M \cdot B^3) = \mathcal{O}(\varepsilon^{-2} \cdot N \cdot \log^2 N)$ time. Therefore, overall, the encoding algorithm, Algorithm 1, runs in at most $\mathcal{O}(\varepsilon^{-2} \cdot N \cdot \mathrm{poly}(\log N))$ time.

The decoding algorithm reads the value $\boldsymbol{u}_t$ and the value m_1 and by applying g on $\boldsymbol{u}_t$, it retrieves the matrices $A_1, \ldots, A_M$. This step takes the time of computing g, i.e., $\mathcal{O}(N \cdot \mathrm{poly}(\log N))$. Then, recovering each portion of the message $\boldsymbol{m}_i$ is done in $\mathcal{O}(B^2)$ (simple multiplication of a vector of length B with a matrix of both dimensions $\leq B$). Thus, recovering the entire message $\boldsymbol{m}$ is done in $\mathcal{O}(M \cdot B^2) = \mathcal{O}(\varepsilon^{-1} \cdot N \cdot \log N)$. Thus, the overall running time of the decoder is $\mathcal{O}(\varepsilon^{-1} \cdot N \cdot \mathrm{poly}(\log N))$.

The number of random bits the algorithm needs is

$$\mathcal{O}\left(\log\left(\frac{B\log(B \cdot N)}{N^{-C}}\right)\right) = \mathcal{O}(C\log(N)) \ ,$$

and since $\varepsilon = 3/C$, the total number of random bits is $\mathcal{O}(\varepsilon^{-1}\log N)$. Note that metadata, $\boldsymbol{u}$, that is generated by Algorithm 1 is of length

$$\mathcal{O}(C\log N) + \log N + \log B = \mathcal{O}(C\log N) = \mathcal{O}(\varepsilon^{-1}\log N) \ . \tag{2}$$

□

Remark 3.5. *Note that to prove Theorem 3.3, one simply needs to change Step 2 from sampling to a brute force search. Namely, for each one of the $N^{\mathcal{O}(C)}$ vectors in the space $\{0,1\}^t$ (recall that $t = \mathcal{O}(C\log N)$) we will compute the function g and check if all the matrices $(A_i)_{\overline{F_i}}, i \in [M]$ are full rank. Doing this step now takes $N^{\mathcal{O}(1/\varepsilon)}\mathrm{poly}(\log N)$ time. Clearly, the complexity of this step dominates the complexity of the algorithm. The rest of proof is identical to that of Theorem 3.3.*

4 Final construction

Note that the main issue with the previous construction is that it assumes that we can transmit the decoder the metadata $\boldsymbol{u}$ that contains the data required by the decoder to perform the decoding. In this section, we shall overcome this problem. Intuitively speaking, our solution will encode the message but also the metadata and location of the metadata in our cover object. In this section, we prove Theorem 1.4 which is restated next.

Theorem 1.4. *For every $\varepsilon > 0$, there exists an $N(\varepsilon)$ such that for every $N > N(\varepsilon)$, there exists a randomized ε-gapped strong-stuck-at-code of length N such that*

1. *The encoder and the decoder run in $\mathcal{O}\left(N \cdot \operatorname{poly}(\log n) \cdot \operatorname{poly}(1/\varepsilon)\right)$.*

2. *The number of random bits that are used by the encoder is $\mathcal{O}\left(\frac{1}{\varepsilon} \log N\right)$ and the encoder succeeds with probability $1 - o(1)$.*

4.1 Auxiliary claims

We start by proving two auxiliary claims that will be useful. We shall divide our cover vector into four subvectors, $\boldsymbol{v} = \boldsymbol{v}_1 \circ \boldsymbol{v}_2 \circ \boldsymbol{v}_3 \circ \boldsymbol{v}_4$. In $\boldsymbol{v}_1$ and $\boldsymbol{v}_3$ we will encode our message and in $\boldsymbol{v}_2$ and $\boldsymbol{v}_4$ we will make sure that we have enough unfrozen bits to encode the metadata. The following claim makes sure that such a partition is indeed feasible. Specifically, we will show that for any small enough δ, there exists an interval $[i \cdot \delta N, (i+1)\delta N - 1]$ (we will define $\boldsymbol{v}_2 = \boldsymbol{v}_{[i \cdot \delta N, (i+1)\delta N - 1]}$) such that it contains at most $(\rho + 2\delta)\delta N$ frozen elements and that this interval does not intersect the subvector $\boldsymbol{v}_4$ which contains exactly $N/\log N$ unfrozen bits.

Claim 4.1. *Let $\rho, \delta \in (0,1)$ such that $2\delta < 1 - \rho$ and let $\mathcal{F} \subseteq [N]$ be of size ρN. Let $j \in [N]$ be the largest such that $\left|[j+1, N] \cap \overline{\mathcal{F}}\right| = N/\log N$. Then, there exists an integer $i \in [\lfloor 1/\delta \rfloor]$ such that*

1. *$i \cdot \delta N + \delta N - 1 \leq j$*

2. *$|[i \cdot \delta N, (i+1) \cdot \delta N - 1] \cap F| \leq (\rho + 2\delta)\delta N$*

In order to encode a small number of bits, say ℓ bits, one could do the following simple trick. Let x represent the decimal number that corresponds to our ℓ bits of information, then, one can just flip unfrozen bits such that the weight of the resulting vector is $x(\bmod 2^{\ell})$. The following simple claim shows how many unfrozen bits are needed to encode using this method.

Claim 4.2. *Let $\boldsymbol{v} \in \{0,1\}^N$ and let $d < N$ be an integer so that there are at least $2d$ unfrozen bits in $\boldsymbol{v}$. Then, we can flip at most d unfrozen bits of $\boldsymbol{v}$ to get a vector $\boldsymbol{w}$ such that $wt(\boldsymbol{w}) \equiv x(\bmod d)$ for any $x < d$.*

We can see that the rate of this encoding method is very small. Indeed, we can hope to encode more than $\log N$ bits using this method. We will use this method to encode the location of the specific intervals that contain the metadata needed for decoding the message.
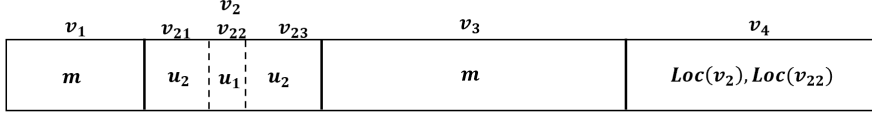
v_1		v_2			v_3	v_4
	v_{21}	v_{22}	v_{23}			
m	u_2	u_1	u_2		m	$Loc(v_2), Loc(v_{22})$

Figure 1: The message m is encoded using Algorithm 1 in v_1 and v_3. Then the metadata u_1 that is needed to decode m is encoded in v_{22}. The metadata u_2 that is needed to decode u_2 is encoded using Claim 4.2 in v_{21} and v_{23} and the locations of v_2 and v_{22} are encoded in v_4

4.2 Encoding and decoding algorithms

We shall use the following notations throughout this section. Again, C is some universal constant known to the encoder and the decoder.

- Let $\delta = 1/C$ and let $B' = \delta N$. Also, assume that $2\delta < 1 - \rho$.

- Let K be the constant implied by (2). Namely, the vector u, that is returned from Algorithm 1 is of length $K \cdot C \log N$ where N is the length of the cover object.

The encoding algorithm is given in Algorithm 3 and the decoding algorithm in Algorithm 4.

Before delving into the details, we give a high-level overview of the encoding algorithm. At the first step, we divide v into four contiguous parts, i.e., $v = v_1 \circ v_2 \circ v_3 \circ v_4$ with the premise given in Claim 4.1. Namely, $|v_2| = B'$ where the number of frozen bits in v_2 is at most $(\rho + \delta)B'$ and v_4 contains exactly $N/\log N$ unfrozen bits.

The second step invokes Algorithm 1 in order to encode the message m into v_1 and v_3. We guarantee that the algorithm will encode only at these parts by adding to the set of frozen bits all the unfrozen bits in v_2 and v_4. Note that this step produces a metadata vector u_1 of length $KC \log N$ that contains the information needed to decode the message.

The third step first divides v_2 into three contiguous parts, $v_2 = v_{21} \circ v_{22} \circ v_{23}$ such that v_{22} is of length $N^{1/2KC}$ and contains at most $(\rho + 2\delta)N^{1/2KC}$ frozen bits. Then, we encode u_1 in v_{22} using Algorithm 1. This produces a metadata vector u_2 of length $KC \cdot \log N^{1/2KC} = \log \sqrt{N}$ which can be represented by a decimal number $U \leq \sqrt{N}$. We shall encode u_2 in v_{21} and v_{23} by flipping at most $2\sqrt{N}$ bits to make sure that $\mathrm{wt}\,(v_2) \equiv U \,(\mathrm{mod}\,\sqrt{N})$ (see Claim 4.2).

Finally, at the fourth step, we encode the starting position of v_2 and v_{22}. We will see that both positions can be identified using only $\log(N^{1-1/2KC})$ bits. Therefore, by Claim 4.2 we can encode this information in v_4 by flipping $2N^{1-1/2KC}$ bits and recall that we have $N/\log N$ unfrozen bits there.

4.3 Analyses

We start by analyzing the rate and then proceed to show the correctness of the algorithms.

14

Algorithm 3: Encode

    input   : A vector $\boldsymbol{v} \in \{0,1\}^N$, a set of frozen indices $\mathcal{F} \subseteq [N], |\mathcal{F}| = \rho N$, and
                  a message $\boldsymbol{m}$, of length $< (1 - \rho - \frac{5}{C})N$

    output   : A vector $\boldsymbol{w} \in \{0,1\}^N$.

[1]   Find the maximal $j \in [N]$ such that there are at least $N/\log N$ unfrozen coordinates in $\boldsymbol{v}$ to the right of j. Find $i \in [j/\delta N]$ such that

$$|[i \cdot B', (i+1) \cdot B' - 1] \cap F| \leq (\rho + \delta) B'$$

and $(i+1) \cdot \delta N \leq j$. Denote $\boldsymbol{v} = \boldsymbol{v}_1 \circ \boldsymbol{v}_2 \circ \boldsymbol{v}_3 \circ \boldsymbol{v}_4$ and $\mathcal{F} = F_1 \cup F_2 \cup F_3 \cup F_4$ where

$$
\begin{aligned}
\boldsymbol{v}_1 &= \boldsymbol{v}_{[1, iB'-1]} & F_1 &= [1, iB' - 1] \cap F \\
\boldsymbol{v}_2 &= \boldsymbol{v}_{[i \cdot B', (i+1) \cdot B' - 1]} & F_2 &= [i \cdot B', (i+1) \cdot B' - 1] \cap F \\
\boldsymbol{v}_3 &= \boldsymbol{v}_{[(i+1) \cdot B', j-1]} & F_3 &= [(i+1) \cdot B', j-1] \cap F \\
\boldsymbol{v}_4 &= \boldsymbol{v}_{[j:N]} & F_4 &= [j : N] \cap F
\end{aligned}
$$

[2]   Run Algorithm 1 with $\boldsymbol{v}, \mathcal{F}_1 \cup [i \cdot B', (i+1) \cdot B' - 1] \cup F_3 \cup [j, N]$, and $\boldsymbol{m} = \boldsymbol{m}$. Denote the first output by $\boldsymbol{w}_1 \circ \boldsymbol{v}_2 \circ \boldsymbol{w}_3 \circ \boldsymbol{v}_4$ where $|\boldsymbol{w}_1| = |\boldsymbol{v}_1|$ and $|\boldsymbol{w}_3| = |\boldsymbol{v}_3|$ and second output as $\boldsymbol{u}_1$

[3]   Find $i' \in [i \cdot B', (i+1) \cdot B' - 1]$ such that i is a multiple of $N^{1/2KC}$ and $|[i', i' + N^{1/2KC} - 1] \cap \mathcal{F}_2| \leq (\rho + 2\delta)N^{1/2KC}$
    Denote $\mathcal{F}_2' = |[i', i' + N^{1/2KC} - 1] \cap \mathcal{F}_2$ and $\boldsymbol{v}_2 = \boldsymbol{v}_{21} \circ \boldsymbol{v}_{22} \circ \boldsymbol{v}_{23}$ where

$$
\begin{aligned}
\boldsymbol{v}_{21} &= \boldsymbol{v}_{[i \cdot B', i'-1]} \\
\boldsymbol{v}_{22} &= \boldsymbol{v}_{[i', i' + N^{1/2KC}-1]} \\
\boldsymbol{v}_{23} &= \boldsymbol{v}_{[i' + N^{1/2KC}, (i+1)B'-1]}
\end{aligned}
$$

  [3.1]   Run Algorithm 1 with $\boldsymbol{v} = \boldsymbol{v}_{22}, \mathcal{F} = F_2'$, and $\boldsymbol{m} = \boldsymbol{u}_1$. Denote the output by $\boldsymbol{w}_{22}$ and $\boldsymbol{u}_2$. Let $U \in [\sqrt{N}]$ be the decimal number that corresponds to $\boldsymbol{u}_2$
  [3.2]   Flip unfrozen bits in $\boldsymbol{v}_{21}$ and $\boldsymbol{v}_{23}$ to get $\boldsymbol{w}_{21}$ and $\boldsymbol{w}_{23}$ so that for
$\boldsymbol{w}_2 := \boldsymbol{w}_{21} \circ \boldsymbol{w}_{22} \circ \boldsymbol{w}_{23}$ it holds that $\text{wt}(\boldsymbol{w}_2) \equiv U \pmod{\sqrt{N}}$

[4]   Let $\boldsymbol{d} = h_{1/\delta}(i) \circ h_{\delta \cdot N^{1 - \frac{1}{2KC}}}(i')$ and denote by $d \in \left[N^{1 - \frac{1}{2KC}}\right]$ the integer whose binary representation is $\boldsymbol{d}$
    Flip unfrozen bits in $\boldsymbol{v}_4$ to get the vector $\boldsymbol{w}_4$ where it holds that
$\text{wt}(\boldsymbol{w}_1 \circ \boldsymbol{w}_2 \circ \boldsymbol{w}_3 \circ \boldsymbol{w}_4) \equiv d \pmod{N^{1 - \frac{1}{2KC}}}$

Algorithm 4: Decode
---
    input   : A vector $\boldsymbol{v} \in \{0,1\}^N$

    output   : A message $\boldsymbol{m} \in \{0,1\}^*$

[1] Let $d \equiv \mathrm{wt}\,(\boldsymbol{v}) \,(\mathrm{mod}\ N^{1-\frac{1}{2KC}})$ and from $h_{N^{\frac{1}{2KC}}}(d)$ identify i, the starting of $\boldsymbol{v}_2$,
    and i', the position inside $\boldsymbol{v}_2$ where $\boldsymbol{u}_1$ is encoded

[2] Let $U \equiv \mathrm{wt}\left(\boldsymbol{v}_{[i,(i+1)\delta N - 1]}\right)(\mathrm{mod}\ \sqrt{N})$

[3] Run Algorithm 2 with input $\boldsymbol{v}_{[i',(i'+1)N^{1/2KC}-1]}$ and $h_{\sqrt{N}}(U)$ to get $\boldsymbol{u}_1$

[4] Run Algorithm 2 with input $\boldsymbol{v}$ and $\boldsymbol{u}_1$ to get $\boldsymbol{m}$

Rate. Our message $\boldsymbol{m}$ is encoded in Step 2 by invoking Algorithm 1 with $\boldsymbol{v} = \boldsymbol{v}_1 \circ \boldsymbol{v}_2 \circ \boldsymbol{v}_3 \circ \boldsymbol{v}_4$ and a set of frozen elements of size at most $\rho N + \delta N + \frac{N}{\log N}$ (We enlarge the set of frozen elements by adding to $\mathcal{F}$ all the coordinates of $\boldsymbol{v}_2$ and $\boldsymbol{v}_4$). Thus, by the premises of Algorithm 1 (see inequality (1)), for large enough N, we can encode up to

$$(1 - \rho - \delta)N - \frac{N}{\log N} - \frac{3N}{C} \tag{3}$$

bits which implies that the rate is

$$1 - \rho - \delta - \frac{3}{C} - \frac{1}{\log N} \geq 1 - \rho - \frac{5}{C}$$

where the inequality holds for large enough N and by recalling that $\delta = 1/C$.

The correctness is given in the following proposition

Proposition 4.3. *Let $\boldsymbol{v} \in \{0,1\}^N$, $\mathcal{F} \subseteq [N]$ where $|\mathcal{F}| = \rho N$. Let $\boldsymbol{m} \in \{0,1\}^m$ where $m \leq N(1 - \rho - \frac{5}{C})$ be a message to be encoded. Then, applying Algorithm 3 on $\boldsymbol{v}$, $\mathcal{F}$, and $\boldsymbol{m}$ succeeds with probability at least $1 - \mathcal{O}\left(1/\log(N)\right)$. Furthermore, if Algorithm 3 succeeds and outputs the vector $\boldsymbol{w}$ then the decoding algorithm, Algorithm 4, which receives $\boldsymbol{w}$ as input, will output $\boldsymbol{m}$.*

Proof. First note that Claim 4.1 guarantees that the partition that we perform in Step 1 is indeed possible.

In Steps 2, we invoke Algorithm 1. In doing that, we have to make sure that the input we give the algorithm is valid. Specifically, if one wishes to encode a message $\boldsymbol{m}$ of length m in a vector $\boldsymbol{v}$ of length $\overline{N}$ with a set of frozen indices $\mathcal{F}$, then by (1) we need that

$$|\boldsymbol{m}| \leq \overline{N} - |\mathcal{F}| - \frac{3\overline{N}}{C}\ . \tag{4}$$

We already showed in (3) what is the maximal message length that can be encoded in Step 2 and that our message length is below that threshold for large enough N.

In Step 3, we focus just on $\boldsymbol{v}_2 = \boldsymbol{v}_{[i \cdot B', i \cdot B' - 1]}$. We first find an index $i' \in [i \cdot B', i \cdot B' - 1]$ that is a multiple of $N^{1/2KC}$ such that $\boldsymbol{v}_{22} = \boldsymbol{v}_{[i', i' + N^{1/2KC} - 1]}$ has at most $(p + \delta)N^{1/2KC}$ frozen bits. Since $\boldsymbol{v}_2$ contains at most $(\rho + \delta)B'$ frozen bits, such an index must exist by a simple averaging argument. Then, in Step 3.1, our goal is to encode $\boldsymbol{u}_1$, the metadata that was returned at the previous step, in $\boldsymbol{v}_{22}$. By the proof of Theorem 3.1, the length of

$\boldsymbol{u}_1$ is $KC \log N$ (recall that K is the constant implied by (2)). Now, since $|\boldsymbol{v}_{22}| = N^{1/2KC}$ and $|\mathcal{F}_2'| \leq (\rho + \delta)N^{1/2KC}$ ($\mathcal{F}_2'$ is the set of frozen coordinates in $\boldsymbol{v}_2 2$), then, for large enough N, inequality (4), holds with $\boldsymbol{m} = \boldsymbol{u}_1$, $\mathcal{F} = \mathcal{F}_2'$, and $\overline{N} = N^{1/2KC}$. Recall that Algorithm 1 returns the encoded vector, which we call $\boldsymbol{w}_2 2$, and a metadata vector, $\boldsymbol{u}_2$, that is needed to decode $\boldsymbol{w}_2 2$. The length of $\boldsymbol{u}_2$ is $KC \log(N^{1/2KC}) = \log(\sqrt{N})$ and our next goal it using Claim 4.1.

To encode $\boldsymbol{u}_2$ in Step 3.2, we represent it using a decimal number U which is at most $\sqrt{N}$. Observe that the number of unfrozen bits in $\boldsymbol{v}_{21}$ and $\boldsymbol{v}_{23}$ is at least $(1 - \rho - 2\delta)\delta N - N^{1/2KC}$ which is greater than $2\sqrt{N}$, for large enough N. Therefore, by Claim 4.2, we can flip $\sqrt{N}$ unfrozen bits in $\boldsymbol{v}_{21}$ and $\boldsymbol{v}_{23}$ and make sure that the resulting weight of $\boldsymbol{w}_2 = \boldsymbol{w}_{21} \circ \boldsymbol{w}_{22} \circ \boldsymbol{w}_{23}$ will be $U(\mathrm{mod} \sqrt{N})$. We now compute what is the failure probability of Steps 2 and 3. Recall that Algorithm 1 can fail with probability $\mathcal{O}\left(1/(C \log N)\right)$, therefore, the failure probabilities of Step 2 and 3 are $\mathcal{O}\left(1/(C \log N)\right)$ and $\mathcal{O}\left(2K/\log N\right)$.

To convince ourselves that Step 4 is feasible, we note that the maximal value i can take is upper bounded by C and that the value of i' is upper bounded by $(N/C) \cdot N^{1/2KC} = N^{1-1/2KC}/C$. Therefore, $|h_C(i) \circ h_{N^{1-1/2KC}/C}(i')| = \log N^{1-1/2KC}$ which implies that there exists a decimal number $U' \leq N^{1-1/2KC}$ that corresponds uniquely to the values i and i'. Note that as the number of unfrozen bits in $\boldsymbol{v}_4$ is $N/\log N > 2N^{1-1/2KC}$ (where the inequality is for large enough N), by Claim 4.2, we can flip these unfrozen bits in $\boldsymbol{v}_4$ to make sure that the weight of $\boldsymbol{w}$ is $U'(\mathrm{mod}\, N^{1-1/2KC})$.

As for the decoder. Note that by computing the weight of $\boldsymbol{v}$ (in Step 1), the decoder knows the values of i and i'. Thus, he knows that he needs to compute $\mathrm{wt}\left(v_{[i,i+B'-1]}\right)$ in order to get the metadata that is needed for decoding $\boldsymbol{u}_1$ from $\boldsymbol{v}_{[i',i'+N^{1/2KC}-1]}$. Once he extracts $\boldsymbol{u}_1$ from $\boldsymbol{v}_{[i',i'+N^{1/2KC}-1]}$ in Step 3, he can proceed to Step 4 and decode the message. $\qquad\square$

Proof of Theorem 1.4. Let C be a universal constant and define $\varepsilon = 5/C$. Note here that $\delta = 1/C < (1 - \rho)/5$ and thus our assumption that $2\delta < 1 - \rho$ holds. Let N be a large enough integer (depending only on ε) and let $\boldsymbol{v} \in \{0,1\}^N$ be a cover vector, and $\mathcal{F} \subset [N]$ be a set of frozen sets of size ρN where $\rho \in (1-\varepsilon, 0)$. Then, according to Proposition 4.3, we can encode any $m \in \{0,1\}^{(1-\rho-\varepsilon)}$. Note that regardless of ρ, the fraction of stuck-at bits, our decoding algorithm always succeeds in decoding the encoded message given an encoded vector Algorithm 3.

We are left to show that the complexity is $N \cdot \mathrm{poly}(N) \cdot \mathrm{poly}(1/\varepsilon)$ for both the encoder and the decoder. Clearly, Step 1 can be done in $\mathcal{O}(N)$. Indeed, identifying a subvector of a specific length with a maximal number of unfrozen bits requires a single scan of the entire input vector. Steps 2 and 3 both invoke Algorithm 1, and thus their running time is $N \cdot \mathrm{poly}(N) \cdot \mathrm{poly}(1/\varepsilon)$. Steps 4 requires at most $\mathcal{O}(N)$ as we need just to compute the weight of a vector and then flip at most $N^{1-1/2KC}$ bits. Note that finding the bits that need to be flipped takes also $\mathcal{O}(N)$ time since we need to find the dominant symbol in the unfrozen bits (0 or 1) and then flip the first unfrozen occurrences of that symbol. Thus, the encoder runs in $N \cdot \mathrm{poly}(N) \cdot \mathrm{poly}(1/\varepsilon)$ time, as desired.

We analyze now the decoder. Steps 1 and 2 run in time $\mathcal{O}(N)$ as we compute the weight of a vector and perform a casting of an integer in decimal representation to binary representation. Steps 3 and 4 invoke Algorithm 2 whose running time is $N \cdot \mathrm{poly}(N) \cdot$

poly$(1/\varepsilon)$. Thus, the total running time is again $N \cdot \text{poly}(N) \cdot \text{poly}(1/\varepsilon)$. $\qquad\square$

Remark 4.4. *As discussed in Remark 3.5, to prove Theorem 1.5, which is the deterministic version of Theorem 1.4, we change the second step in Algorithm 1 to a brute force step. This change affects the complexity of Step 2 and 3 which now becomes $N^{\mathcal{O}(1/\varepsilon)}$. The rest of the proof is identical to the one of the randomized construction.*

References

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

[CKVY19] Yeow Meng Chee, Han Mao Kiah, Alexander Vardy, and Eitan Yaakobi. Explicit and efficient wom codes of finite length. *IEEE Transactions on Information Theory*, 66(5):2669–2682, 2019.

[GS12] Ariel Gabizon and Ronen Shaltiel. Invertible zero-error dispersers and defective memory with stuck-at errors. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 553–564. Springer, 2012.

[Hee83] Chris Heegard. Partitioned linear block codes for computer memory with'stuck-at'defects. *IEEE Transactions on Information Theory*, 29(6):831–842, 1983.

[Hee85] Chris Heegard. On the capacity of permanent memory. *IEEE Transactions on Information Theory*, 31(1):34–42, 1985.

[KK13] Yongjune Kim and BVK Vijaya Kumar. Coding for memory with stuck-at defects. In *2013 IEEE International Conference on Communications (ICC)*, pages 4347–4352. IEEE, 2013.

[KT74] Aleksandr Vasil'evich Kuznetsov and Boris Solomonovich Tsybakov. Coding in a memory with defective cells. *Problemy peredachi informatsii*, 10(2):52–60, 1974.

[LMJF10] Luis Alfonso Lastras-Montaño, Ashish Jagmohan, and MM Franceschini. Algorithms for memories with stuck cells. In *2010 IEEE International Symposium on Information Theory*, pages 968–972. IEEE, 2010.

[MV15] Hessam Mahdavifar and Alexander Vardy. Explicit capacity achieving codes for defective memories. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 641–645. IEEE, 2015.

[RS82] Ronald L Rivest and Adi Shamir. How to reuse a "write-once" memory. *Information and control*, 55(1-3):1–19, 1982.

[Shp13] Amir Shpilka. New constructions of wom codes using the wozencraft ensemble. *IEEE Transactions on Information Theory*, 59(7):4520–4529, 2013.

[Tsy75]   Boris Solomonovich Tsybakov. Defect and error correction. *Problemy peredachi informatsii*, 11(3):21–30, 1975.

[WZY15]   Antonia Wachter-Zeh and Eitan Yaakobi. Codes for partially stuck-at memory cells. *IEEE Transactions on Information Theory*, 62(2):639–654, 2015.

A Appendix

A.1 The complexity of g from Lemma 1.7

We briefly recall the third construction given in [AGHP92].

- Let $h := k \log r$ and let $A \in \mathbb{F}_2^{r \times h}$ be a generating of binary code whose dual distance is exactly k.

- Let $t := \log \frac{h}{\mu}$

- Let $x, y \in \mathbb{F}_{2^{t/2}}$, where $\mathbb{F}_{2^{t/2}}$ is the finite field with $2^{t/2}$ elements. Note that x and y can be viewed also as elements in $\{0,1\}^{t/2}$ as $\mathbb{F}_{2^{t/2}} \cong \mathbb{F}_2^{t/2}$.

The function $g : \{0,1\}^t \to \{0,1\}^r$ is defined by

$$g(x,y) = A \cdot \left(\langle x^0, y \rangle, \langle x^1, y \rangle, \ldots, \langle x^{h-1}, y \rangle \right)$$

where $\langle \cdot, \cdot \rangle$ is the mod two inner product. By [AGHP92], $g(U_t)$ is μ-wise k independent variable over $\{0,1\}^r$ where U_t is the uniform distribution over U_t.

As for the complexity of computing g. Note that $\langle x^i, y \rangle$ can be performed in $\mathrm{poly}(t)$ time and we perform this operation h times. The multiplication of the vactor by the matrix takes $\mathcal{O}(h \cdot r)$ operations. In total, we perform,

$$\mathcal{O}(h \cdot \mathrm{poly}(t) + h \cdot r) = \mathcal{O}\left(k \log r \cdot \mathrm{poly}(\log \frac{k \log r}{\mu}) + r \cdot k \log r \right) \ ,$$

and since in our settings, $r = \mathcal{O}(N \log N)$, $k = \mathcal{O}(\log N)$, and $\mu = N^{\mathcal{O}(1)}$, we get that the complexity of g is

$$\mathcal{O}(\mathrm{poly}(\log N) + N \cdot \mathrm{poly}(\log N)) = \mathcal{O}(N \cdot \mathrm{poly}(\log N)) \ .$$

A.2 Missing proofs

Proof of Claim 1.9. Denote by r_i the ith row. The matrix A has full rank if and only if for any $i \in [m]$,

- The rows $r_1, \ldots, r_{i-1}$ are linearly independent and,

- $r_i \notin \mathrm{span}\{r_1, \ldots, r_{i-1}\}$.

Thus,

$$\Pr[A \text{ has full rank}] = \prod_{i=1}^{m} \Pr[r_i \notin \mathrm{span}\{r_1, \ldots, r_{i-1}\}]$$

$$\geq \prod_{i=1}^{m} \left(1 - 2^{i-1}(2^{-n} + \varepsilon) \right)$$

$$\geq 1 - (2^{-n} + \varepsilon) \cdot \sum_{i=1}^{m} 2^{i-1}$$

$$= 1 - 2^{m-n} - \varepsilon \cdot 2^m$$

where the first inequality holds since $r_i = (A_{r,1}, A_{r,2}, \ldots, A_{r,n})$ is an ε-almost n-wise random variable and the second inequality is a standard union bound. $\square$

Proof of Claim 4.1. Denote $x = |[j+1, N] \cap \mathcal{F}|$ (the number of frozen bits in $\boldsymbol{v}_4$) and note that $j = N - x - \frac{N}{\log N}$. Therefore, we have $|[j] \cap \mathcal{F}| = \rho N - x$. Denote $I_i := [i \cdot \delta N, (i+1) \cdot \delta N - 1]$ for all $i \in [\lfloor j/\delta N \rfloor - 1]$. Assume that for all $i \in [\lfloor j/\delta N \rfloor - 1]$, it holds that $|I_i \cap F| > (\rho + 2\delta)\delta N$. Then, $|[\delta N \cdot \lfloor j/\delta N \rfloor] \cap F| > (\rho + 2\delta)j - \delta N$. Therefore,

$$
\begin{aligned}
|\mathcal{F}| &> (\rho + 2\delta)\, j - \delta N + x \\
&= (\rho + 2\delta)N + (1 - \rho - 2\delta)x - \frac{(\rho + 2\delta)N}{\log N} - \delta N \\
&= (\rho + \delta)N + (1 - \rho - 2\delta)x - \mathcal{O}\left(\frac{N}{\log N}\right) \\
&> \rho N
\end{aligned}
$$

where the last inequality follows since $2\delta < 1 - \rho$ and for large enough N. $\square$

Proof of Claim 4.2. Assume that $\mathrm{wt}\,(\boldsymbol{v}) \equiv y \pmod d$ for some $y < d$. Since there are at least $2d$ unfrozen bits, at least d of them are either 1 or 0. Assume w.l.o.g., that at least d of them are zero. Then, we need to flip exactly $x - y \pmod d$ bits in order to get a vector $\boldsymbol{w}$ with $\mathrm{wt}\,(\boldsymbol{w}) \equiv x \pmod d$. $\square$