Rank-Modulation Rewrite Coding for Flash Memories

Eyal En Gad, Eitan Yaakobi, *Member, IEEE,* Anxiao (Andrew) Jiang, *Senior Member, IEEE,* and Jehoshua Bruck, *Fellow, IEEE*

Abstract

The current flash memory technology focuses on the cost minimization of its static storage capacity. However, the resulting approach supports a relatively small number of program-erase cycles. This technology is effective for consumer devices (e.g., smartphones and cameras) where the number of program-erase cycles is small. However, it is not economical for enterprise storage systems that require a large number of lifetime writes.

The proposed approach in this paper for alleviating this problem consists of the efficient integration of two key ideas: (i) improving reliability and endurance by representing the information using relative values via the rank modulation scheme and (ii) increasing the overall (lifetime) capacity of the flash device via rewriting codes, namely, performing multiple writes per cell before erasure.

This paper presents a new coding scheme that combines rank-modulation with rewriting. The key benefits of the new scheme include: (i) the ability to store close to 2 bits per cell on each write with minimal impact on the lifetime of the memory, and (ii) efficient encoding and decoding algorithms that make use of capacity-achieving write-once-memory (WOM) codes that were proposed recently.

Index Terms

rank modulation, permutations of multisets, flash memories, WOM codes, side-information coding.

I. Introduction

The term *rank modulation* refers to the representation of information by *permutations*. This method was initially suggested by Slepian in 1965 for the application of bandlimited digital communication [28]. In 2009, Jiang *et al.* suggested that rank modulation could be useful for information storage in non-volatile devices such as flash memories [19], especially considering their asymmetric write properties. The motivation for this application comes from the physical and architectural properties of flash memories. Flash memories are composed of cells which store electric charge, where the amount of charge is quantized to represent information. When a cell is quantized into two states, it can store a single bit, and it is called a single-level cell (SLC). When it is quantized into more states, typically 4 or 8, it is called a multi-level cell (MLC)[1]. The quantization is performed by a comparison of the charge level with a set of threshold levels.

While injecting charge into a flash cell is a simple operation, removing it is difficult, and can be done only by the removal of the *entire* charge from a large block of cells, a process called *block erasure*. It is desirable to minimize the usage of block erasures, since they are slow, power consuming and reduce the device reliability. The high cost of erasure also obstructs the operation of charge injection in MLC flash. Since charge injection is a noisy process, an overshooting may occur, where a cell is programmed to a higher discrete level than desired. And since the excessive charge cannot be removed, the cell value must

The material in this paper was presented in part at the IEEE Int. Symp. on Inform. Theory (ISIT), Saint Petersburg, Russia, August 2011 [8], at the IEEE Int. Symp. on Inform. Theory (ISIT), Cambridge, MA, U.S.A., July 2012 [9], and at the IEEE Int. Symp. on Inform. Theory (ISIT), Istanbul, Turkey, July 2013 [7].

E. En Gad, E. Yaakobi and J. Bruck are with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA, 91125 USA e-mail: (eengad@caltech.edu; yaakobi@caltech.edu; bruck@caltech.edu).

A. Jiang is with the Department of Computer Science and Engineering and the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (email:ajiang@cse.tamu.edu).

[1]Here we are using the term MLC in its broad sense. In the industry, cells of 8 levels are often referred to as TLC.

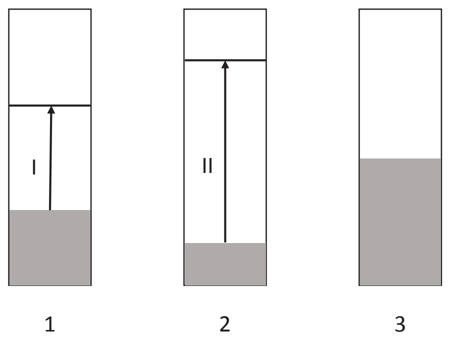


Fig. 1. Rewriting a new permutation in two stages. In stage I, cell 1 is pushed above cell 3, and in stage II, cell 2 is pushed above cell 1.

be treated as an error. To minimize the chances of such overshooting errors, the charge injection is done iteratively, and therefore slowly. An additional issue in flash memories is the leakage of charge from the cells over time, which introduces additional errors.

The rank-modulation scheme tackles both the overshooting and the charge leakage problems. In rank modulation, a set of n memory cells represents information according to the *permutation* induced by the cell levels. For example, we can use a set of 3 cells, labeled from 1 to 3, such that each cell has a distinct charge level. We then rank the cells according to their charge levels, and obtain one of $3! = 6$ possible permutations of the set $\{1, 2, 3\}$. A possible permutation could have, for example, cell 3 with the highest level, then cell 1 and then cell 2 with the lowest level, as shown in Figure 1. Each permutation can represent a distinct message, and so the 3 cells in this example store together $\log_2 6$ bits. The charge leakage is tolerated well with permutations, since it behaves similarly in spatially close cells, and thus the order of the cell levels is not likely to change. In comparison, when the information is represented by a quantization of absolute values, the leakage is more likely to introduce errors. This phenomenon was demonstrated in phase-change memories, a different type of non-volatile memories with a similar leakage property [24].

When a user wishes to write a message to the memory, she needs to increase the cell levels such that they will represent the permutation that corresponds to the desired message. For example, suppose that the user wishes to write the message that corresponds to the permutation in which cell 2 is the highest cell, then cell 1 and then cell 3. The user first increases the level of cell 1 above that of cell 3, and then she increases the level of cell 2 above that of cell 1, as shown in Figure 1. Notice that in each step a cell level is increased to be above a certain value, but there is no need to carefully keep it from exceeding some maximal value. Therefore, the cell programming can be accelerated significantly.

Following the flash memory application, research interest in the rank-modulation scheme increased in recent years. Error-correcting codes for a variety of error models were studied in [3], [11], [20], [29]. In addition, other variations of rank modulation were proposed and studied in [10], [31]. A hardware implementation of the scheme was recently designed on flash memories [21]. In this paper we focus on the notion of *rewriting codes*, that were proposed for the rank-modulation scheme in [19]. Rewriting codes were studied for traditional representation schemes since the work of Rivest and Shamir on write-once memories [26], and are closely related to the side-information coding problem that was studied by Gelfand and Pinsker [13]. Other models of rewrite coding were proposed by Jiang, Bohossian and Bruck in [17].

In the context of flash memories, the aim of rewriting codes is to increase the lifetime of the memory. This is especially important in applications that require a large number of lifetime writes, such as enterprise storage systems. In conventional flash systems, a block of cells is always erased before data is written to the cells. The approach of rewriting codes is to write a set of cells multiple times between block erasures. On each write, additional charge is injected to the cells such that they induce a desired new permutation, and thus represent a new information message. After a certain number of rewriting cycles, some of the cells will reach their maximal level, at which point a block erasure is unavoidable. The aim of rewriting codes is to increase the number of rewriting cycles before a block erasure is necessary.

In rank-modulation, a certain gap is needed between the charge levels of cells of adjacent ranks, in order to reduce the chance of errors. Therefore, we use a discrete model for the design and analysis of rewriting codes, despite the fact that the information is represented by the relative *analog* levels of the cells. Our approach is to focus, on each rewrite, on the difference between the level of the *highest* cell before and after the write. We refer to this difference as the *cost* of rewriting. For example, the cost of the rewriting in Figure 1 is 2 "gaps". However, if we were to write the permutation in which cell 1 is the highest, then cell 3 and then cell 2, then we would only need to increase cell 1 above cell 3 (stage I in Figure 1), and the cost would only be a single "gap". The reason for this definition of cost is that writing with high cost gets the highest cell closer to its maximal levels, when block erasure is required for additional writes. Under this model, the goal in this paper is to study codes that limit the rewriting cost, and in that way increase the number of writes between erasures.

The main results of this paper are the following:

1) We derive an expression for the maximal amount of information that can be stored with a limited rewriting cost. When the cost is limited to its minimal amount, in which the number of writes is maximized, the maximal amount of information that can be stored on each write is 2 bits per cell.
2) We describe a construction of a rewrite coding scheme, which allows for the writing of asymptotically optimal amount of information for any limit on the writing cost. The construction is composed of efficient encoding and decoding algorithms that make use of capacity-achieving write-once-memory (WOM) codes that were proposed recently.

The rest of the paper is organized as follows: In Section II we formally define the notion of rank-modultion rewriting codes and study their information limits. Section III describes the higher level of the construction we propose in this paper, and Sections IV and V describe two alternative implementations of the building blocks of the construction. Finally, concluding remarks are provided in Section VI.

II. Definition and Limits of Rank-Modulation Rewriting Codes

In this section we define the rank-modulation scheme and the associated rewriting codes, and study the limits of those codes.

A. The Rank-Modulation Scheme

In this work we consider a modulation scheme that represents information by permutations of multisets. Let $M = \{a_1^{z_1}, \dots, a_q^{z_q}\}$ be a multiset of q distinct elements, where each element a_i appears z_i times. The positive integer z_i is called the multiplicity of the element a_i, and the cardinality of the multiset is $n = \sum_{i=1}^{q} z_i$. For a positive integer n, the set $\{1, 2, \dots, n\}$ is labeled by $[n]$. We think of a permutation σ of the multiset M as a partition of the set $[n]$ into q disjoint subsets, $\sigma = (\sigma(1), \sigma(2), \dots, \sigma(q))$, such that $|\sigma(i)| = z_i$ for each $i \in [q]$, and $\cup_{i \in [q]} \sigma(i) = [n]$. We also define the inverse permutation σ^{-1} such that for each $i \in [q]$ and $j \in [n]$, $\sigma^{-1}(j) = i$ if j is a member of the subset $\sigma(i)$. We label σ^{-1} as the length-n vector $\sigma^{-1} = (\sigma^{-1}(1), \sigma^{-1}(2), \dots, \sigma^{-1}(n))$. For example, if $M = \{1, 1, 2, 2\}$ and $\sigma = (\{1, 3\}, \{2, 4\})$, then $\sigma^{-1} = (1, 2, 1, 2)$. We refer to both σ and σ^{-1} as a permutation, since they represent the same object.

Let $\mathfrak{S}_M$ be the set of all permutations of the multiset M. By abuse of notation, we view $\mathfrak{S}_M$ also as the set of the inverse permutations of the multiset M. For a given cardinality n and number of elements q, it is easy to show that the number of multiset permutations is maximized if the multiplicities of all of the elements are equal. Therefore, to simplify the presentation, we take most of the multisets in this paper to be of the form $M = \{1^z, 2^z, \dots, q^z\}$, and label the set $\mathfrak{S}_M$ by $\mathfrak{S}_{q,z}$.

Consider a set of n memory cells, and denote $\boldsymbol{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ as the *cell-state vector*. We represent information on the cells according to the mutiset permutation that their values induce. This permutation is derived by a demodulation process.

Demodulation: Given positive integers q and z, a cell-state vector $\boldsymbol{x}$ of length $n = qz$ is demodulated into a permutation $\pi_{\boldsymbol{x}}^{-1} = (\pi_{\boldsymbol{x}}^{-1}(1), \pi_{\boldsymbol{x}}^{-1}(2), \dots, \pi_{\boldsymbol{x}}^{-1}(n))$. Note that while $\pi_{\boldsymbol{x}}^{-1}$ is a function of q, z and $\boldsymbol{x}$,

q and z are not specified in the notation since they will be clear from the context. The demodulation is performed as follows: First, let $k_1, \dots, k_n$ be an order of the cells such that $x_{k_1} \le x_{k_2} \le \cdots \le x_{k_n}$. Then, for each $j \in [n]$, assign $\pi_{\boldsymbol{x}}^{-1}(k_j) = \lceil j/z \rceil$.

Example 1. *Let* $q = 3$, $z = 2$ *and so* $n = qz = 6$. *Assume that we wish to demodulate the cell-state vector* $\boldsymbol{x} = (1, 1.5, 0.3, 0.5, 2, 0.3)$. *We first order the cells according to their values:* $(k_1, k_2, \dots, k_6) = (3, 6, 4, 1, 2, 5)$, *since the third and sixth cells have the smallest value, and so on. Then we assign*

$$\pi_{\boldsymbol{x}}^{-1}(k_1 = 3) = \lceil 1/2 \rceil = 1,$$
$$\pi_{\boldsymbol{x}}^{-1}(k_2 = 6) = \lceil 2/2 \rceil = 1,$$
$$\pi_{\boldsymbol{x}}^{-1}(k_3 = 4) = \lceil 3/2 \rceil = 2,$$

and so on, and get the permutation $\pi_{\boldsymbol{x}}^{-1} = (2, 3, 1, 2, 3, 1)$. *Note that* $\pi_{\boldsymbol{x}}^{-1}$ *is in* $\mathfrak{S}_{3,2}$.

Note that $\pi_{\boldsymbol{x}}$ is not unique if for some $i \in [q]$, $x_{k_{zi}} = x_{k_{zi+1}}$. In this case, we define $\pi_{\boldsymbol{x}}$ to be illegal and denote $\pi_{\boldsymbol{x}} = F$. We label Q_M as the set of all cell-state vectors that demodulate into a valid permutation of M. That is, $Q_M = \{\boldsymbol{x} \in \mathbb{R}^n \,|\pi_{\boldsymbol{x}} \ne F\}$. So for all $\boldsymbol{x} \in Q_M$ and $i \in [q]$, we have $x_{k_{zi}} < x_{k_{zi+1}}$.

For $j \in [q]$, the value $\pi^{-1}(j)$ is called the *rank* of cell j in the permutation π. In order to reduce the possibility of error in the demodulation process, a certain gap must be placed between the levels of cells with different ranks. We normalize that gap, and without loss of generality constrain the cell levels to be natural numbers. So a cell-state vector must be a member of $\mathbb{Z}^n$.

We model the flash memory such that when a user wishes to store a message on the memory, the cell levels can only increase. When the cells reach their maximal levels, an expensive erasure operation is required. Therefore, in order to maximize the number of writes between erasures, it is desirable to raise the cell levels as little as possible on each write. For a cell-state vector $\boldsymbol{x} \in Q_M$, denote by $\Gamma_{\boldsymbol{x}}(i)$ the highest level among the cells with rank i in $\pi_{\boldsymbol{x}}$. That is,

$$\Gamma_{\boldsymbol{x}}(i) = \max_{j \in \pi_{\boldsymbol{x}}(i)} \{x_j\}.$$

Let $\boldsymbol{s}$ be the cell-state vector of the memory before the writing process takes place, and let $\boldsymbol{x}$ be the cell-state vector after the write. The following modulation method minimizes the increase in the cell levels.

Modulation: Writing a permutation π on a memory with state $\boldsymbol{s}$. The output is the new memory state, denoted by $\boldsymbol{x}$.

1) For each $j \in \pi(1)$, assign $x_j \Leftarrow s_j$.
2) For $i = 2, 3, \dots, q$, for each $j \in \pi(i)$, assign

$$x_j \Leftarrow \max\{s_j, \Gamma_{\boldsymbol{x}}(i-1) + 1\}.$$

Example 2. *Let* $q = 3$, $z = 2$ *and so* $n = qz = 6$. *Let the state be* $\boldsymbol{s} = (3, 4, 2, 3, 4, 1)$ *and the target permutation be* $\pi^{-1} = (1, 1, 2, 2, 3, 3)$. *In step 1 of the modulation process, we notice that* $\pi(1) = \{1, 2\}$ *and so we set*

$$x_1 \Leftarrow s_1 = 3$$

and

$$x_2 \Leftarrow s_2 = 4.$$

In step 2 we have $\pi(2) = \{3, 4\}$ *and* $\Gamma_{\boldsymbol{x}}(1) = \max\{x_1, x_2\} = \max\{3, 4\} = 4$, *so we set*

$$x_3 \Leftarrow \max\{s_3, \Gamma_{\boldsymbol{x}}(1) + 1\} = \max\{2, 5\} = 5$$

and

$$x_4 \Leftarrow \max\{s_4, \Gamma_{\boldsymbol{x}}(1) + 1\} = \max\{3, 5\} = 5.$$

And in the last step we have $\pi(3) = \{5, 6\}$ *and* $\Gamma_{\boldsymbol{x}}(2) = 5$, *so we set*

$$x_5 \Leftarrow \max\{4, 6\} = 6$$

and

$$x_6 \Leftarrow \max\{1, 6\} = 6.$$

In summary, we get $\boldsymbol{x} = (3, 4, 5, 5, 6, 6)$, *which demodulates into* $\pi_{\boldsymbol{x}}^{-1} = (1, 1, 2, 2, 3, 3) = \pi^{-1}$, *as required.*

Since the cell levels cannot decrease, we must have $x_j \geq s_j$ for each $j \in [n]$. In addition, for each j_1 and j_2 in $[n]$ for which $\pi^{-1}(j_1) > \pi^{-1}(j_2)$, we must have $x_{j_1} > x_{j_2}$. Therefore, the proposed modulation process minimizes the increase in the levels of all the cells. Remember also that the level x_j of each cell is upper bounded by a certain value. Therefore, given a state $\boldsymbol{s}$, certain permutations π might require a block erasure before writing, while others might not. In addition, some permutations might get the memory state *closer* to a state in which an erasure is required than other permutations. In order to maximize the number of writes between block erasures, we add redundancy by letting *multiple* permutations represent the *same* information message. This way, when a user wishes to store a certain message, she could chose one of the permutations that represent the required message such that the chosen permutation will increase the cell levels in the least amount. Such a method can increase the longevity of the memory in the expense of the amount of information stored on each write. The mapping between the permutations and the messages they represent is called a rewriting code.

To analyze and design rewriting codes, we focus on the difference between $\Gamma_{\boldsymbol{x}}(q)$ and $\Gamma_{\boldsymbol{s}}(q)$. Using the modulation process we defined above, the vector $\boldsymbol{x}$ is a function of $\boldsymbol{s}$ and π, and therefore the difference $\Gamma_{\boldsymbol{x}}(q) - \Gamma_{\boldsymbol{s}}(q)$ is also a function of $\boldsymbol{s}$ and π. We label this difference by $\alpha(\boldsymbol{s} \to \pi) = \Gamma_{\boldsymbol{x}}(q) - \Gamma_{\boldsymbol{s}}(q)$ and call it the *rewriting cost*, or simply the cost. We motivate this choise by the following example. Assume that the difference between the maximum level of the cells and $\Gamma_{\boldsymbol{s}}(q)$ is 10 levels. Then only the permutations π which satisfy $\alpha(\boldsymbol{s} \to \pi) \leq 10$ can be written to the memory without erasure. Alternatively, if we use a rewriting code that guarantees that for any state $\boldsymbol{s}$, any message can be stored with, say, cost no greater than 1, then we can guarantee to write 10 more times to the memory before an erasure will be required. Such rewriting codes are the focus of this paper.

B. Definition of Rank-Modulation Rewriting Codes

The cost $\alpha(\boldsymbol{s} \to \pi)$ was defined according to the vectors $\boldsymbol{s}$ and $\boldsymbol{x}$. However, it will be helpful for the study of rewriting codes to have some understanding of the cost in terms of the demodulation of the state $\boldsymbol{s}$ and the permutation π. Let $\sigma_{\boldsymbol{s}}$ be the permutation obtained from the demodulation of the state $\boldsymbol{s}$. We present such connection in the following lemma.

Lemma 3. *Let* M *be a multiset of cardinality* n. *If* $\boldsymbol{s}$ *is in* Q_M *and* π *is in* $\mathfrak{S}_M$, *then*

$$\alpha(\boldsymbol{s} \to \pi) \leq \max_{j \in [n]}\{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\} \tag{1}$$

with equality if $\Gamma_q(\boldsymbol{s}) - \Gamma_1(\boldsymbol{s}) = q - 1$.

The proof of Lemma 3 is brought in Appendix A. We would take a worst-case approach, and opt to design codes that guarantee that on each rewriting, the value $\max_{j \in [n]}\{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}$ is bounded. For permutations σ and π in $\mathfrak{S}_{q,z}$, the rewriting cost $\alpha(\sigma \to \pi)$ is defined as

$$\alpha(\sigma \to \pi) = \max_{j \in [n]}\{\sigma^{-1}(j) - \pi^{-1}(j)\}.$$

This expression is an asymmetric version of the Chebyshev distance (also known as the L_∞ distance). For simplicity, we assume that the channel is noiseless and don't consider the error-correction capability of the codes. However, such consideration would be essential for practical applications. Since the channel is noiseless, the code is uniquely defined by the decoding map.

Definition 4. (Rank-modulation rewriting codes) *Let q, z, r and K_R be positive integers, and let $\mathcal{C}$ be a subset of $\mathfrak{S}_{q,z}$ called the codebook. Then a surjective function $D_R : \mathcal{C} \to [K_R]$ is a (q, z, K_R, r)* ***rank-modulation rewriting code (RM rewriting code)*** *if for each message $m \in [K_R]$ and state $\sigma \in \mathcal{C}$, there exists a permutation π in $D_R^{-1}(m) \subseteq \mathcal{C}$ such that $\alpha(\sigma \to \pi) \le r$.*

$D_R^{-1}(m)$ is the set of permutations that represent the message m. It could also be insightful to study rewriting codes according to an average cost constraint, assuming some distribution on the source and/or the state. However, we use the wort-case constraint since it is easier to analyze. The amount of information stored with a (q, z, K_R, r) RM rewriting code is $\log K_R$ bits (all of the logarithms in this paper are binary). Since it is possible to store up to $\log |\mathfrak{S}_{q,z}|$ bits with permutations of a multiset $\{1^z, \dots, q^z\}$, it could be natural to define the code rate as:

$$R' = \frac{\log K_R}{\log |\mathfrak{S}_{q,z}|}.$$

However, this definition doesn't give much engineering insight into the amount of information stored in a set of memory cells. Therefore, we define the rate of the code as the amount of information stored per memory cell:

$$R = \frac{\log K_R}{qz}.$$

An encoding function E_R for a code D_R maps each pair of message m and state σ into a permutation π such that $D_R(\pi) = m$ and $\alpha(\sigma \to \pi) \le r$. By abuse of notation, let the symbols E_R and D_R represent both the functions and the algorithms that compute those functions. If D_R is a RM rewriting code and E_R is its associated encoding function, we call the pair (E_R, D_R) a rank-modulation rewrite coding scheme.

Rank-modulation rewriting codes were proposed by Jiang *et al.* in [19], in a more restrictive model than the one we defined above. The model in [19] is more restrictive in two senses. First, the mentioned model used the rank-modulation scheme with permutations of sets only, while here we also consider permutations of multisets. And second, the rewriting operation in the mentioned model was composed only of a cell programming operation called "push to the top", while here we allow a more opportunistic programming approach. A push-to-the-top operation raises the charge level of a single cell above the rest of the cells in the set. As described above, the model of this paper allows to raise a cell level above a subset of the rest of the cells. The rate of RM rewriting codes with push-to-the-top operations and cost of $r = 1$ tends to zero with the increase in the block length n. On the contrary, we will show that the rate of RM rewriting codes with cost $r = 1$ and the model of this paper tends to 1 bit per cell with permutations of sets, and 2 bits per cell with permutations of multisets.

C. Limits of Rank-Modulation Rewriting Codes

For the purpose of studying the limits of RM rewriting codes, we define the ball of radius r around a permutation σ in $\mathfrak{S}_{q,z}$ by

$$B_{q,z,r}(\sigma) = \{\pi \in \mathfrak{S}_{q,z} | \alpha(\sigma \to \pi) \le r\},$$

and derive its size in the following lemma.

Lemma 5. *For positive integers q and z, if σ is in $\mathfrak{S}_{q,z}$ then*

$$|B_{q,z,r}(\sigma)| = \binom{(r+1)z}{z}^{q-r} \prod_{i=1}^{r} \binom{iz}{z}.$$

Proof: Let $\pi \in B_{q,z,r}(\sigma)$. By the definition of $B_{q,z,r}(\sigma)$, for any $j \in \pi(1)$, $\sigma^{-1}(j) - 1 \le r$, and thus $\sigma^{-1}(j) \le r + 1$. Therefore, there are $\binom{(r+1)z}{z}$ possibilities for the set $\pi(1)$ of cardinality z. Similarly, for any $i \in \pi(2)$, $\sigma(i)^{-1} \le r + 2$. So for each fixed set $\pi(1)$, there are $\binom{(r+1)z}{z}$ possibilities for $\pi(2)$, and in total $\binom{(r+1)z}{z}^2$ possibilities for the pair of sets $(\pi(1), \pi(2))$. The same argument follows for all $i \in [q - r]$, so there are $\binom{(r+1)z}{z}^{q-r}$ possibilities for the sets $(\pi(1),\dots,\pi(q-r))$. The rest of the sets of π: $\pi(q-r+1),\pi(q-r+2),\dots,\pi(q)$, can take any permutation of the multiset $\{(q-r+1)^z,(q-r+2)^z,\dots,q^z\}$, giving the statement of the lemma. ■

Note that the size of $B_{q,z,r}(\sigma)$ is actually *not* a function of σ. Therefore we denote it by $|B_{q,z,r}|$.

Proposition 6. *Let D_R be a (q,z,K_R,r) RM rewriting code. Then*

$$K_R \le |B_{q,z,r}|.$$

Proof: Fix a state $\sigma \in \mathcal{C}$. By Definition 4 of RM rewriting codes, for any message $m \in [K_R]$ there exists a permutation π such that $D_R(\pi)=m$ and π is in $B_{q,z,r}(\sigma)$. It follows that $B_{q,z,r}(\sigma)$ must contain K_R different permutations, and so its size must be at least K_R. ■

Corollary 7. *Let $R(r)$ be the rate of an (q,z,K_R,r)-RM rewriting code. Then*

$$R(r) < (r+1)H\left(\frac{1}{r+1}\right),$$

where $H(p)=-p\log p-(1-p)\log(1-p)$. In particular, $R(1)<2$.

Proof:

$$\begin{aligned}\log|B_{q,z,r}| =& \sum_{i=1}^{r}\log\binom{iz}{z}+(q-r)\log\binom{(r+1)z}{z}\\ <& r\log\binom{(r+1)z}{z}+(q-r)\log\binom{(r+1)z}{z}\\ =& q\log\binom{(r+1)z}{z}\\ <& q\cdot(r+1)zH\left(\frac{1}{r+1}\right),\end{aligned}$$

where the last inequality follows from Stirling's formula. So we have

$$R(r)=\frac{\log K_R}{qz}\le\frac{\log|B_{q,z,r}|}{qz}<(r+1)H\left(\frac{1}{r+1}\right).$$

The case of $r=1$ follows immediately. ■

We will later show that this bound is in fact tight, and therefore we call it the *capacity* of RM rewriting codes and denote it as

$$C_R(r)=(r+1)H\left(\frac{1}{r+1}\right).$$

Henceforth we omit the radius r from the capacity notation and denote it by C_R. To further motivate the use of multiset permutations rather than set permutation, we can observe the following corollary.

Corollary 8. *Let $R(r)$ be the rate of an $(q,1,K_R,r)$-RM rewriting code. Then $R(r)<\log(r+1)$, and in particular, $R(1)<1$.*

Proof: Note first that $|B_{q,z,r}|=(r+1)^{q-r}r!$. So we have

$$\begin{aligned}\log|B_{q,z,r}| =& \log r!+(q-r)\log(r+1)\\ <& r\log(r+1)+(q-r)\log(r+1)\\ =& q\log(r+1).\end{aligned}$$

Therefore,

$$R(r)\le\frac{\log|B_{q,z,r}|}{q}<\log(r+1),$$

and the case of $r = 1$ follows immediately. ■

In the case of $r = 1$, codes with multiset permutations could approach a rate close to 2 bits per cell, while there are no codes with set permutations and rate greater than 1 bit per cell. The constructions we present in this paper are analyzed only for the case of multiset permutations with a large value of z. We now define two properties that we would like to have in a family of RM rewrite coding schemes. First, we would like the rate of the codes to approach the upper bound of Corollary 7. We call this property *capacity achieving*.

Definition 9. (Capacity-achieving family of RM rewriting codes) *For a positive integer i, let the positive integers q_i, z_i and K_i be some functions of i, and let $n_i = q_i z_i$ and $R_i = (1/n_i)\log K_i$. Then an infinite family of (q_i, z_i, K_i, r) RM rewriting codes is called* ***capacity achieving*** *if*

$$\lim_{i\to\infty} R_i = C_R.$$

The second desired property is computational efficiency. We say that a family of RM rewrite coding schemes $(E_{R,i}, D_{R,i})$ is *efficient* if the algorithms $E_{R,i}$ and $D_{R,i}$ run in polynomial time in $n_i = q_i z_i$. The main result of this paper is a construction of an efficient capacity-achieving family of RM rewrite coding schemes.

III. High-Level Construction

The proposed construction is composed of two layers. The higher layer of the construction is described in this section, and two alternative implementations of the lower layer are described in the following two sections. We will present two versions of the higher layer in this section. The first version is presented mainly for the purpose of explaining the main ideas, and the second version will introduce a few additional ideas.

A. Basic Construction

A few additional definitions are needed for the description of the construction. First, let $2^{[n]}$ denote the set of all subsets of $[n]$. Next, let the function $\theta_n : 2^{[n]} \to \{0,1\}^n$ be defined such that for a subset $S \subseteq [n]$, $\theta_n(S) = (\theta_{n,1}, \theta_{n,2}, \ldots, \theta_{n,n})$ is its characteristic vector, where

$$\theta_{n,j} = \begin{cases} 0 & \text{if } j \notin S \\ 1 & \text{if } j \in S. \end{cases}$$

For a vector $\boldsymbol{x}$ of length n and a subset $S \subseteq [n]$, we denote by $\boldsymbol{x}_S$ the vector of length $|S|$ which is obtained by "throwing away" all the positions of $\boldsymbol{x}$ outside of S. For positive integers $n_1 \le n_2$, the set $\{n_1, n_1+1, \ldots, n_2\}$ is labeled by $[n_1 : n_2]$. Finally, for a permutation $\sigma \in \mathfrak{S}_{q,z}$, we define the set $U_{i_1,i_2}(\sigma)$ as the union of the sets $\{\sigma(i)\}_{i\in[i_1:i_2]}$ if $i_1 \le i_2$. If $i_1 > i_2$, we define $U_{i_1,i_2}(\sigma)$ to be the empty set.

To construct a (q, z, K_R, r) RM rewriting scheme $\{E_R, D_R\}$, we will think first of the encoder E_R. Given a message $m \in [K_R]$ and a state σ in the codebook $\mathcal{C}$, the encoder needs to find a permutation π in $B_{q,z,r}(\sigma)$, such that $D_R(\pi) = m$. Remember that for π to be in the set $B_{q,z,r}(\sigma)$, the cost $\alpha(\sigma \to \pi)$ must not be greater that r. And by the definition of the cost, this implies that for each $j \in [n]$ we must have $\sigma^{-1}(j) - \pi^{-1}(j) \le r$. Specifically, for each cell j that the encoder might choose for the set $\pi(1)$, we have $\pi^{-1}(j) = 1$ and so we must have $\sigma^{-1}(j) \le r + 1$. According to the notation we defined above, the set of cells j for which $\sigma^{-1}(j) \le r + 1$ is denoted by $U_{1,r+1}(\sigma)$. So in short, the encoder must chose the set $\pi(1)$ as a subset of $U_{1,r+1}(\sigma)$. Remember also that the codeword π must be in the set $D_R^{-1}(m)$, and so for each message m and state $\sigma \in \mathcal{C}$, the set $D_R^{-1}(m)$ must contain at least one codeword π with a set $\pi(1)$ that is a subset of $U_{1,r+1}(\sigma)$. The main observation of this paper is that this constraint is very similar to a property of a class of codes known in the literature as *write-once-memory codes*, or WOM codes for short.

WOM codes were proposed by Rivest and Shamir in 1982 for the rewriting of memories in which each cell can only be written once [26]. Several variations of write-once memories were considered, but here we only consider the case in which the cells take only values of 0 or 1. We can think of the characteristic vector of $U_{1,r+1}(\sigma)$, denoted as $\boldsymbol{s} = \theta_n(U_{1,r+1}(\sigma))$ (where $n = qz$), as the state of the memory before it is rewritten, and of the vector $\boldsymbol{x} = \theta_n(\pi(1))$ as the codeword written to the memory. The constraint $\pi(1) \subset U_{1,r+1}(\sigma)$ is then translated to the constraint $\boldsymbol{x} \leq \boldsymbol{s}$, which means that for each $j \in [n]$ we must have $x_j \leq s_j$. Write-once memories are typically defined such that each cell can be written once from 0 to 1. However, here it is more convenient to consider cells that can be written once from 1 to 0, since then a cell in state $s_j = 0$ cannot be changed anymore, and we must have $x_j = 0$. This is equivalent to the notation $x_j \leq s_j$. The fact that we "reverse" the definition of write-once memories will of course not prevent us from using any result on WOM codes from the literature.

Remember that the cardinality of the set $\pi(1)$ is equal to z. That means that the "WOM codeword" $\boldsymbol{x}$ that we write to the memory must have a Hamming weight of z (number of non-zero bits). Therefore, we will be interested in WOM codes in which all of the codewords have the same weight (henceforth the term weight refers to the Hamming weight). Such codes were not considered in the literature, and we naturally call them *constant-weight WOM codes*. For the definition of constant-weight WOM codes we need an additional notation. For a positive integer n and a real number $w \in [0,1]$, we let $J_w(n) \subset \{0,1\}^n$ be the set of all vectors of n bits whose weight equals $\lfloor wn \rfloor$. Since we will need to use a weaker version of this definition later, we add the word "strong" to the definition here.

Definition 10. (Constant-weight strong WOM codes) *Let K_W and n be positive integers and let w_s be a real number in $[0,1]$ and w_x be a real number in $[0,w_s]$. A surjective function $D_W : J_{w_x}(n) \to [K_W]$ is an (n, K_W, w_s, w_x)* ***constant-weight strong WOM code*** *if for each message $m \in [K_W]$ and state vector $\boldsymbol{s} \in J_{w_s}(n)$, there exists a codeword vector $\boldsymbol{x} \leq \boldsymbol{s}$ in the subset $D_W^{-1}(m) \subseteq J_{w_x}(n)$. The rate of a constant-weight strong WOM code is defined as $R_W = (1/n)\log K_W$.*

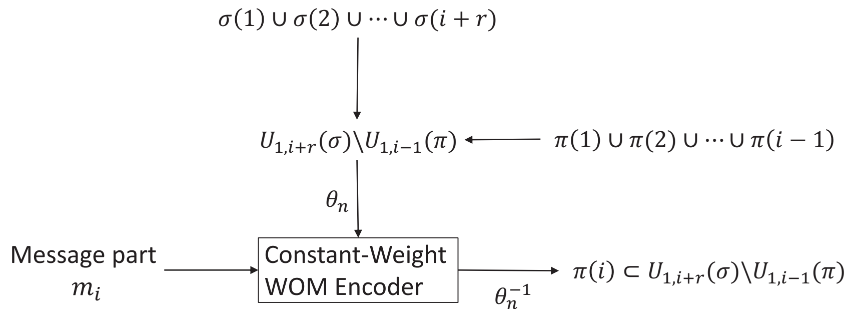
It is useful to know the tightest upper bound on the rate of constant-weight strong WOM codes, which we call the capacity of those codes.

Lemma 11. *Let w_s and w_x be as defined in Definition 10. Then the capacity of constant-weight strong WOM codes is*

$$C_W = w_s H(w_x/w_s).$$

The proof of Lemma 11 is brought in Appendix B. We also define the notions of coding scheme, capacity achieving and efficient family for constant-weight strong WOM codes in the same way we defined it for RM rewriting codes. The main property of the basic construction we present now is that using an efficient capacity-achieving family of constant-weight strong WOM coding schemes, the constructed family of RM rewrite coding schemes is also efficient and capacity achieving. Unfortunately, we do not know how to construct an efficient capacity-achieving family of constant-weight strong WOM coding schemes. For that reason, we will later present modifications to this construction that uses weaker notions of constant-weight WOM codes.

We now see how WOM codes can help for the construction of RM rewriting codes. Think of m_1 as some "part" of the message, say the first few bits of its binary representation. Then the encoder can think of $U_{1,r+1}(\sigma)$ as a subset of $[n]$, and think of its characteristic vector $\theta_n(U_{1,r+1}(\sigma))$ as the *state* vector $\boldsymbol{s}_1$ of a write-once memory. In this view, an $(n, K_1, (r+1)/q, 1/q)$ constant-weight strong WOM code D_W can be used for representing m_1 by the set $\pi(1)$. First, notice that the weight of $\boldsymbol{s}_1 = \theta_n(U_{1,r+1}(\sigma))$ is $(r+1)z = ((r+1)/q)n$, as required by the WOM code, since $U_{1,r+1}(\sigma)$ has $(r+1)z$ members. Second, the encoder wishes to find a subset $\pi(1) \subset U_{1,r+1}(\sigma)$ that represents m_1. In terms of the characteristic vector, that means that the encoder wishes to find a vector $\boldsymbol{x}_1$ of weight $z = (1/q)n$, such that $\boldsymbol{x}_1 \leq \boldsymbol{s}_1$, and $\boldsymbol{x}_1$ represents m_1. Since we let m_1 be represented by a WOM code, Definition 10 tells us that such a vector $\boldsymbol{x}_1$ always exists. Furthermore, if the WOM code has an efficient encoding algorithm E_W, then we can find $\boldsymbol{x}_1$ fast. Once we find $\boldsymbol{x}_1$, we let $\pi(1)$ be the set with characteristic vector $\boldsymbol{x}_1$, that is $\pi(1) = \theta_n^{-1}(\boldsymbol{x}_1)$.



$\sigma(1) \cup \sigma(2) \cup \cdots \cup \sigma(i+r)$
$U_{1,i+r}(\sigma) \setminus U_{1,i-1}(\pi)$
$\pi(1) \cup \pi(2) \cup \cdots \cup \pi(i-1)$
θ_n
Message part m_i
Constant-Weight WOM Encoder
θ_n^{-1}
$\pi(i) \subset U_{1,i+r}(\sigma) \setminus U_{1,i-1}(\pi)$

Fig. 2. Iteration i of the encoding algorithm, where $1 \leq i \leq q-r-1$.

When the decoder will read the permutation π, she could observe the characteristic vector $\theta_n(\pi(1))$ and find m_1 according to the decoding map of the WOM code.

We now apply the same idea to the rest of the sets of π, iteratively. On each iteration i from 1 to $q-r-1$, the set $\pi(i)$ must be a subset of $U_{1,i+r}(\sigma)$, otherwise π wouldn't be in $B_{q,z,r}(\sigma)$. However, here the sets $\{\pi(1), \ldots, \pi(i-1)\}$ were already determined in previous iterations, and thus their members cannot belong to $\pi(i)$. The set $U_{1,i-1}(\pi)$ contains the members of those sets, where $U_{1,0}(\pi)$ is the empty set. So we can say that the set $\pi(i)$ must be a subset of $U_{1,i+r}(\sigma) \setminus U_{1,i-1}(\pi)$ (the set of cells in $U_{1,i+r}(\sigma)$ that are not in $U_{1,i-1}(\pi)$). As before, we use a WOM code to represent a message "part" m_i by the set $\pi(i)$, and we let the state vector of the WOM code to be $\boldsymbol{s}_i = \theta_n(U_{1,i+r}(\sigma) \setminus U_{1,i-1}(\pi))$. We then use the WOM encoder $E_W(m_i, \boldsymbol{s}_i)$ to find an appropriate vector $\boldsymbol{x}_i \leq \boldsymbol{s}_i$ that represents m_i, and assign $\pi(i) = \theta_n^{-1}(\boldsymbol{x}_i)$, such that $\pi(i)$ represents m_i. This algorithm is shown in Figure 2. If we use a capacity achieving family of constant-weight strong WOM codes, we store close to $w_s H(w_x/w_s) = (r+1)(1/q)H(\frac{1}{r+1})$ bits per cell on each rank. This amount is multiplied by $q-r-1$ ranks, approaching the capacity of RM rewriting codes when q is large compared to r.

To complete the encoding function, we are left with determining the sets $\{\pi(q-r), \ldots, \pi(q)\}$. These sets can in fact take any permutation of the multiset $M = \{(q-r)^z, (q-r+1)^z, \ldots, q^z\}$ without taking the cost of rewrite above r. That is because for any cell $j \in U_{q-r,q}$, we have $\sigma^{-1}(j) - \pi^{-1}(j) \leq q-(q-r) = r$. For that reason, we use an enumerative code for those ranks. Let $h_M : [|\mathfrak{S}_M|] \to \mathfrak{S}_M$ be a bijective mapping from indices to multiset pemutations, and let h_M^{-1} be the inverse map. There are several efficient implementations of such maps, see for example [23]. The part of the message that will be stored in the ranks $[q-r:q]$ is denoted by m_{q-r}, and the sequence of sets $(\pi(q-r), \ldots, \pi(q))$ is denoted by $\pi_{[q-r:q]}$, where $\pi_{[q-r:q]}$ is in fact a permutation of the multiset M. Then the last step of the encoding algorithm is to assign $\pi_{[q-r:q]} = h_M(m_{q-r})$.

To decode a message vector $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r})$ from the stored permutation π, we can just decode each of the bottom $q-r-1$ ranks separately, and the top $r+1$ ranks together. For each rank $i \in [q-r-1]$, the decoder finds the vector $\boldsymbol{x}_i = \theta_n(\pi(i))$, and then the message part m_i is calculated by the WOM decoder, $m_i \Leftarrow D_W(\boldsymbol{x}_i)$. The message part m_{q-r} is found by the decoder of the enumerative code, $m_{q-r} = h_M^{-1}(\pi_{[q-r:q]})$. The different message parts can in fact be decoded in parallel.

Construction 12. (A RM rewriting code from a constant-weight strong WOM code) *Let K_W, q, r, z be positive integers, let $n = qz$ and let (E_W, D_W) be an $(n, K_W, (r+1)/q, 1/q)$ constant-weight strong WOM coding scheme. Define the multiset $M = \{(q-r)^z, (q-r+1)^z, \ldots, q^z\}$ and let $K_M = |\mathfrak{S}_M|$ and $K_R = K_W^{q-r-1} \cdot K_M$. The codebook $\mathcal{C}$ is defined to be the entire set $\mathfrak{S}_{q,z}$. A (q, z, K_R, r) RM rewrite coding scheme $\{E_R, D_R\}$ is constructed as follows:*

*The **encoding** algorithm E_R receives a message $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$ and a state permutation $\sigma \in \mathfrak{S}_{q,z}$, and returns a permutation π in $B_{q,z,r}(\sigma) \cap D_R^{-1}(\boldsymbol{m})$ to store in the memory. It is constructed as follows:*

```
1: for $i = 1$ to $q - r - 1$ do
```

```
2:     $\boldsymbol{s}_i \Leftarrow \theta_n(U_{1,i+r}(\sigma) \setminus U_{1,i-1}(\pi))$
3:     $\boldsymbol{x}_i \Leftarrow E_W(m_i, \boldsymbol{s}_i)$
4:     $\pi(i) \Leftarrow \theta_n^{-1}(\boldsymbol{x}_i)$
5: end for
6: $\pi_{[q-r:q]} \Leftarrow h_M(m_{q-r})$
```

*The **decoding** algorithm D_R receives the stored permutation $\pi \in \mathfrak{S}_{q,z}$, and returns the stored message $\boldsymbol{m} = (m_1, m_2, \dots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$. It is constructed as follows:*

```
1: for $i = 1$ to $q - r - 1$ do
2:     $\boldsymbol{x}_i \Leftarrow \theta_n(\pi(i))$
3:     $m_i \Leftarrow D_W(\boldsymbol{x}_i)$
4: end for
5: $m_{q-r} \Leftarrow h_M^{-1}(\pi_{[q-r:q]})$
```

Remark: On each rank $i \in [2 : q - r - 1]$, we can in fact use a WOM code with shorter block length, since the cells in the ranks lower than i do not need to be used. This slightly improves the rate and computation complexity of the coding scheme. However, this improvement does not affect the asymptotic analysis we make in this paper. Therefore, for the ease of presentation, we did not use this improvement.

Theorem 13. *Let $\{E_W, D_W\}$ be a member of an efficient capacity-achieving family of constant-weight strong WOM coding schemes. Then the family of RM rewrite coding schemes in Construction 12 is efficient and capacity-achieving.*

Proof: The decoded message is equal to the encoded message by the property of the WOM code in Definition 10. By the explanation above the construction, it is clear that the cost is bounded by r, and therefore $\{E_R, D_R\}$ is a RM rewrite coding scheme. We will first show that $\{E_R, D_R\}$ is capacity achieving, and then show that it is efficient. Let $R_R = (1/n)\log K_R$ be the rate of a RM rewriting code. To show that $\{E_R, D_R\}$ is capacity achieving, we need to show that for any $\epsilon_R > 0$, $R_R > C_R - \epsilon_R$, for some q and z.

Since $\{E_W, D_W\}$ is capacity achieving, $R_W > C_W - \epsilon_W$ for any $\epsilon_W > 0$ and large enough n. Remember that $C_W = w_s H(w_x/w_s)$. In $\{E_R, D_R\}$ we use $w_s = (r+1)/q$ and $w_x = 1/q$, and so $C_W = \frac{r+1}{q} H\left(\frac{1}{r+1}\right)$. We have

$$\begin{aligned}
R_R &= (1/n)\log K_R \\
&= (1/n)\log(K_M \cdot K_W^{q-r-1}) \\
&> (q-r-1)(1/n)\log K_W \\
&> (q-r-1)(C_W - \epsilon_W) \\
&= (q-r-1)\left(\frac{r+1}{q} H\left(\frac{1}{r+1}\right) - \epsilon_W\right) \\
&= \frac{q-r-1}{q}(C_R - q\epsilon_W) \\
&= (C_R - q\epsilon_W)(1 - (r+1)/q) \\
&> C_R - (r+1)^2/q - q\epsilon_W
\end{aligned} \tag{2}$$

The idea is to take $q = \lfloor (r+1)/\sqrt{\epsilon_W} \rfloor$ and $\epsilon_R = 3(r+1)\sqrt{\epsilon_W}$, and get that

$$R_R > C_R - \frac{(r+1)^2}{\lfloor (r+1)/\sqrt{\epsilon_W} \rfloor} - \lfloor (r+1)/\sqrt{\epsilon_W} \rfloor \epsilon_W > C_R - 2(r+1)\sqrt{\epsilon_W} - (r+1)\sqrt{\epsilon_W} = C_R - \epsilon_R.$$

Formally, we say: for any $\epsilon_R > 0$ and integer r, we set $\epsilon_W = \frac{\epsilon_R^2}{9(r+1)^2}$ and $q = \lfloor (r+1)/\sqrt{\epsilon_W} \rfloor$. Now if z is large enough then $n = qz$ is also large enough so that $R_W > C_W - \epsilon_W$, and then Equation 2 holds and we have $R_R > C_R - \epsilon_R$, proving that the construction is capacity achieving. Note that the family of coding schemes has a constant value of q and a growing value of z, as permitted by Definition 9 of capacity-achieving code families.

Next we show that $\{E_R, D_R\}$ is efficient. If the scheme (h_M, h_M^{-1}) is implemented as described in [23], then the time complexity of h_M and h_M^{-1} is polynomial in n. In addition, we assumed that E_W and D_W run in polynimial time in n. So since h_M and h_M^{-1} are executed only once in E_R and D_R, and E_W and D_W are executed less than q times in E_R and D_R, where $q < n$, we get that the time complexity of E_R and D_R is polynomial in n. ■

Remark: Since the proof of Theorem 13 is asymptotic, the $\log K_E$ bits stored in the top $r+1$ ranks were not used in the rate analysis. Nonetheless, they could contribute to the performance of practical systems.

B. Rank-Modulation Rewriting Schemes from Weak WOM Schemes

As mentioned earlier, we are not familiar with a family of efficient capacity-achieving constant-weight strong WOM coding schemes. Nonetheless, it turns out that we can construct efficient capacity-achieving WOM coding schemes that meet a slightly weaker definition, and use them to construct capacity-achieving RM rewriting codes. In this subsection we will define a weak notion of constant-weight WOM codes, and describe an associated RM rewriting coding scheme. In Sections IV and V we will present yet weaker definition of WOM codes, together with constructions of appropriate WOM schemes and associated RM rewriting schemes.

In the weak definition of WOM codes, each codeword is a *pair*, composed of a constant-weight binary vector $\boldsymbol{x}$ and an index integer m_a. Meanwhile, the state is still a single vector $\boldsymbol{s}$, and the vector $\boldsymbol{x}$ in the codeword is required to be smaller than the state vector. We say that these codes are weaker since there is no restriction on the index integer in the codeword. This allows the encoder to communicate some information to the decoder without restrictions.

Definition 14. (Constant-weight weak WOM codes) *Let K_W, K_a, n be positive integers and let w_s be a real number in $[0,1]$ and w_x be a real number in $[0, w_s]$. A surjective function $D_W : J_{w_x}(n) \times [K_a] \to [K_W]$ is an (n, K_W, K_a, w_s, w_x)* ***constant-weight weak WOM code*** *if for each message $m \in [K_W]$ and state vector $\boldsymbol{s} \in J_{w_s}(n)$, there exists a pair $(\boldsymbol{x}, m_a)$ in the subset $D_W^{-1}(m) \subseteq J_{w_x}(n) \times [K_b]$ such that $\boldsymbol{x} \leq \boldsymbol{s}$. The rate of a constant-weight weak WOM code is defined to be $R_W = (1/n)\log(K_W/K_a)$.*

If $K_a = 1$ the code is in fact a constant-weight strong WOM code. Since R_W is a decreasing function of K_a, it follows that the capacity of constant-weight weak WOM code is also $C_W = w_s H(w_x/w_s)$. Consider now the encoder E_R of a (q, z, K_R, r) RM rewriting code D_R with a codebook $\mathcal{C}$. For a message $m \in [K_R]$ and a state permutation $\sigma \in \mathcal{C}$, the encoder needs to find a permutation π in the intersection $B_{q,z,r}(\sigma) \cap D_R^{-1}(m)$. As before, we let the encoder determine the sets $\pi(1), \pi(2), \ldots, \pi(q-r-1)$ sequentially, such that each set $\pi(i)$ represents a message part m_i. If we were to use the previous encoding algorithm (in Construction 12) with a weak WOM code, the WOM encoding would find a pair $(\boldsymbol{x}_i, m_{a,i})$, and we could store the vector $\boldsymbol{x}_i$ by the set $\pi(i)$. However, we would not have a way to store the index $m_{a,i}$ that is also required for the decoding. To solve this, we will *add* some cells that will serve for the sole purpose of storing the index $m_{a,i}$.

Since we use the WOM code $q-r-1$ times, once for each rank $i \in [q-1]$, it follows that we need to add $q-r-1$ different sets of cells. The added sets will take part in a larger permutation, such that the code will still meet Definition 4 of RM rewriting codes. To achieve that property, we let each added set of cells to represent a permutation. That way the number of cells in each rank is constant, and a concatenation (in the sense of sting concatenation) of those permutations together results in a larger permutation. To keep the cost of rewriting bounded by r, we let each added set to represent a permutation with $r+1$ ranks. That way each added set could be rewritten arbitrarily with a cost no greater than r. We also let the number of cells in each rank in those added sets to be equal, in order to maximize the amount of stored information. Denote the number of cells in each rank in each of the added set as a. Since each added set needs to store an index from the set $[K_a]$ with $r+1$ ranks, it follows that a must satisfy the inequality $|\mathfrak{S}_{r+1,a}| \geq K_a$. So to be economical with our resources, we set a to be the smallest integer that satisfies this inequality. We denote each of these additional permutations as $\pi_{a,i} \in \mathfrak{S}_{r+1,a}$. The main permutation is denoted by π_W, and the number of cells in each rank in π_W is denoted by z_W. The permutation π will be a string concatenation of the main permutation together with the $q-r-1$ added permutations. Note that this way the number of cells in each rank is not equal (there are more cells in the lowest $r+1$ ranks). This is actually not a problem, but it will be cleaner to present the construction if we add yet another permutation that "balances" the code. Specifically, we let π_b be a permutation of the multiset $\left\{(r+2)^{(q-r-1)a}, (r+3)^{(q-r-1)a}, \ldots, q^{(q-r-1)a}\right\}$ and let π^{-1} be the string concatenation $(\pi_{a,1}^{-1}, \ldots, \pi_{a,q-r-1}^{-1}, \pi_b^{-1}, \pi_W^{-1})$. This way in each rank there are exactly $z_W + (q-r-1)a$ cells. We denote $z = z_W + (q-r-1)a$, and then we get that π is a member of $\mathfrak{S}_{q,z}$.

On each iteration i from 1 to $q-r-1$ we use a constant-weight weak WOM code. The vectors $\boldsymbol{s}_i$ and $\boldsymbol{x}_i$ of the WOM code are now corresponding only to the main part of the permutation, and we denote their length by $n_W = qz_W$. We assign the state vector to be $\boldsymbol{s}_i = \theta_{n_W}(U_{1,i+r}(\sigma_W) \setminus U_{1,i-1}(\pi_W))$, where σ_W and π_W are the main parts of σ and π, accordingly. Note that $U_{1,i+r}(\sigma_W)$ and $U_{1,i-1}(\pi_W)$ are subsets of $[n_W]$ and that the characteristic vector θ_{n_W} is taken according to n_W as well. The message part m_i and the state vector $\boldsymbol{s}_i$ are used by the encoder E_W of an $(n_W, K_W, K_b, (r+1)/q, 1/q)$ constant-weight weak WOM code D_W. The result of the encoding is the pair $(\boldsymbol{x}_i, m_{a,i}) = E_W(m_i, \boldsymbol{s}_i)$. The vector $\boldsymbol{x}_i$ is stored on the main part of π, by assigning $\pi_W(i) = \theta_{n_W}^{-1}(\boldsymbol{x}_i)$. The additional index $m_{a,i}$ is stored on the additional cells corresponding to rank i. Using an enumerative code $h_{r+1,a} : [|\mathfrak{S}_{r+1,a}|] \to \mathfrak{S}_{r+1,a}$, we assign $\pi_{a,i} = h_{r+1,a}(m_{a,i})$. After the lowest $q-r-1$ ranks of π_W are determined, we determine the highest $r+1$ ranks by setting $\pi_{W,[q-r,q]} = h_M(m_{q-r})$ where $M = \{(q-r)^{z_W}, (q-r+1)^{z_W}, \ldots, q^{z_W}\}$. Finally, the permutation π_b can be set arbitrarily, say, to σ_b.

The decoding is performed in accordance with the encoding. For each rank $i \in [q-r-1]$, we first find $\boldsymbol{x}_i = \theta_{n_W}(\pi_W(i))$ and $m_{a,i} = h_{r+1,a}^{-1}(\pi_{a,i})$, and then assign $m_i = D_W(\boldsymbol{x}_i, m_{a,i})$. Finally, we assign $m_{q-r} = h_M^{-1}(\pi_{W,[q-r:q]})$.

Construction 15. (A RM rewriting code from a constant-weight weak WOM code) *Let K_W, K_a, q, r and z_W be positive integers, and let $n_W = qz_W$. Let D_W be an $(n_W, K_W, K_a, (r+1)/q, 1/q)$ constant-weight weak WOM code with encoding algorithm E_W, and let a be the smallest integer for which $|\mathfrak{S}_{r+1,a}| \geq K_a$. Define the multiset $M = \{(q-r)^{z_W}, (q-r+1)^{z_W}, \ldots, q^{z_W}\}$ and let $K_M = |\mathfrak{S}_M|$ and $K = K_M \cdot K_W^{q-r-1}$.*

Let $z = z_W + (q-r-1)a$ and $n = qz$. Define a codebook $\mathcal{C} \subset \mathfrak{S}_{q,z}$ as a set of permutations $\pi \in \mathcal{C}$ in which π^{-1} is a string concatenation $(\pi_W^{-1}, \pi_{a,1}^{-1}, \ldots, \pi_{a,q-r-1}^{-1}, \pi_b^{-1})$ such that the following conditions hold:

1) $\pi_W \in \mathfrak{S}_{q,z_W}$.
2) *For each rank $i \in [q-r-1]$, $\pi_{a,i} \in \mathfrak{S}_{r+1,a}$.*
3) *π_b is a permutation of the multiset $\left\{(r+2)^{(q-r-1)a}, (r+3)^{(q-r-1)a}, \ldots, q^{(q-r-1)a}\right\}$.*

A (q, z, K_R, r) RM rewrite coding scheme $\{E_R, D_R\}$ is constructed as follows:

*The **encoding** function E_R receives a message $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$ and a state permutation $\sigma \in \mathcal{C}$, and finds a permutation π in $B_{q,z,r}(\sigma) \cap D_R^{-1}(\boldsymbol{m})$ to store in the memory. It is constructed as follows:*

```
1: for $i = 1$ to $q - r - 1$ do
2:     $\boldsymbol{s}_i \Leftarrow \theta_{n_W}(U_{1,i+r}(\sigma_W) \setminus U_{1,i-1}(\pi_W))$
3:     $(\boldsymbol{x}_i, m_{a,i}) \Leftarrow E_W(m_i, \boldsymbol{s}_i)$
4:     $\pi_W(i) \Leftarrow \theta_{n_W}^{-1}(\boldsymbol{x}_i)$
5:     $\pi_{a,i} \Leftarrow h_{r+1,a}(m_{a,i})$
6: end for
7: $\pi_{W,[q-r:q]} \Leftarrow h_M(m_{q-r})$
8: $\pi_b \Leftarrow \sigma_b$
```

*The **decoding** function D_R receives the stored permutation $\pi \in \mathcal{C}$, and finds the stored message $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$. It is constructed as follows:*

```
1: for $i = 1$ to $q - r - 1$ do
```

```
2:     $\boldsymbol{x}_i \Leftarrow \theta_{n_W}(\pi_W(i))$
3:     $m_{a,i} \Leftarrow h^{-1}_{r+1,a}(\pi_{a,i})$
4:     $m_i \Leftarrow D_W(\boldsymbol{x}_i, m_{a,i})$
5: end for
6: $m_{q-r} \Leftarrow h^{-1}_M(\pi_{W,[q-r:q]})$
```

Remark: To be more economical with our resources, we could use the added sets "on top of each other", such that the $r+1$ lowest ranks store one added set, the next $r+1$ ranks store another added set, and so on. To ease the presentation, we did not describe the construction this way, since the asymptotic performance is not affected. However, such a method could increase the performance of practical systems.

Theorem 16. *Let $\{E_W, D_W\}$ be a member of an efficient capacity-achieving family of constant-weight weak WOM coding schemes. Then the family of RM rewrite coding schemes in Construction 15 is efficient and capacity-achieving.*

The proof of Theorem 16 is similar to that of Theorem 13 and is brought in Appendix C.

IV. Constant-Weight Polar WOM Codes

In this section we consider the use of polar WOM schemes [6] for the construction of constant-weight weak WOM schemes. Polar WOM codes do not have a constant weight, and thus require a modification in order to be used in Construction 15 of RM rewriting codes. The modification we propose in this section is exploiting the fact that while polar WOM codes do not have a constant weight, their weight is still *concentrated* around a constant value. This section is composed of two subsections. In the first, we show a general method to construct constant-weight weak WOM codes from WOM codes with concentrated weight. The second subsection describes the construction of polar WOM schemes of Burshtein and Strugatski [6], and explains how they could be used as concentrated-weight WOM schemes.

A. Constant-Weight Weak WOM Schemes from Concentrated-Weight Strong Schemes

We first introduce additional notation. Label the weight of a vector $\boldsymbol{x}$ by $w_H(\boldsymbol{x})$. For $\delta > 0$, let $J_{w_x}(n, \delta)$ be the set of all n-bit vectors $\boldsymbol{x}$ such that $|w_x - w_H(\boldsymbol{x})/n| \leq \delta$.

Definition 17. (Concentrated-weight WOM codes) *Let K_C and n be positive integers and let w_s be in $[0,1]$, w_x be in $[0, w_s]$ and δ in $[0,1]$. A surjective function $D_C : J_{w_x}(n,\delta) \to [K_C]$ is an $(n, K_C, w_s, w_x, \delta)$* ***concentrated-weight WOM code*** *if for each message $m \in [K_C]$ and state vector $\boldsymbol{s} \in J_{w_s}(n)$, there exists a vector $\boldsymbol{x} \leq \boldsymbol{s}$ in the subset $D_C^{-1}(m) \subseteq J_{w_x}(n,\delta)$.*

From Theorem 1 in [15] and Lemma 11 we get that the capacity of concentrated-weight WOM codes in $C_W = w_s H(w_x/w_s)$. We define the notion of efficient capacity-achieving family of concentrated-weight WOM coding schemes accordingly. For the construction of constant-weight weak WOM codes from concentrated-weight WOM codes, we will use another type of enumerative coding schemes. For an integer n and δ in $[0, 1/2]$, let $J_{\leq\delta}(n)$ be the set of all n-bit vectors of weight at most δn, and define some bijective function $h_{\leq\delta} : \left[\sum_{i=1}^{\lfloor \delta n \rfloor} \binom{n}{j}\right] \to J_{\leq\delta}(n)$ with an inverse function $h^{-1}_{\leq\delta}$. The enumeration scheme $(h_{\leq\delta}, h^{-1}_{\leq\delta})$ can be implemented with computational complexity polynomial in n by methods such as [4, pp. 27-30], [25], [30].

We will now describe a construction of a constant-weight weak WOM coding scheme from a concentrated-weight WOM coding scheme. We start with the encoder E_W of the constant-weight weak WOM codes. According to Definition 14, given a message $m \in [K_W]$ and a state $\boldsymbol{s} \in J_{w_s}(n)$, the encoder needs to find a pair $(\boldsymbol{x}, m_a)$ in the set $D_W^{-1}(m)$ such that $\boldsymbol{x} \leq \boldsymbol{s}$. We start the encoding by finding the vector $\boldsymbol{x}_C = E_C(m, \boldsymbol{s})$ by the encoder of an $(n, K_C, w_s, w_x, \delta)$ concentrated-weight WOM code. We know that the weight of $\boldsymbol{x}_C$ is "δ-close" to $w_x n$, but we need to find a vector with weight exactly $\lfloor w_x n \rfloor$. To do this, the main idea is to "flip" $|\lfloor w_x n \rfloor - w_H(\boldsymbol{x}_C)|$ bits in $\boldsymbol{x}_C$ to get a vector $\boldsymbol{x} \leq \boldsymbol{s}$ of weight $\lfloor w_x n \rfloor$, and store the location of the flipped bits in m_a. Let $\boldsymbol{a}$ be the n-bit vector of the flipped locations, such that $\boldsymbol{x} = \boldsymbol{x}_C \oplus \boldsymbol{a}$

where $\oplus$ is the bitwise XOR operation. It is clear that the weight of $\boldsymbol{a}$ must be $|\lfloor w_x n \rfloor - w_H(\boldsymbol{x}_C)|$. Let $\boldsymbol{x}_C = (x_{C,1}, x_{C,2}, \ldots, x_{C,n})$. If $w_H(\boldsymbol{x}_C) < w_x n$, we also must have $a_i = 0$ wherever $x_{C,i} = 1$, since we only want to flip 0's to 1's to increase the weight. In addition, we must have $a_i = 0$ wherever $s_i = 0$, since in those locations we have $x_{C,i} = 0$ and we want to get $x_i \leq s_i$. We can summarize those conditions by requiring that $\boldsymbol{a} \leq \boldsymbol{s} \oplus \boldsymbol{x}_C$ if $w_H(\boldsymbol{x}_C) < w_x n$. In the case that $w_H(\boldsymbol{x}_C) > w_x n$, we should require that $\boldsymbol{a} \leq \boldsymbol{x}_C$, since a_i can be 1 only where $x_{C,i} = 1$. In both cases we have the desired properties $\boldsymbol{x} \leq \boldsymbol{s}$, $w_H(\boldsymbol{x}) = \lfloor w_x n \rfloor$ and $w_H(\boldsymbol{a}) \leq \delta n$.

To complete the encoding, we let m_a be the index of the vector $\boldsymbol{a}$ in an enumeration of the n-bit vectors of weight at most δn. That will minimize the space required to store $\boldsymbol{a}$. Using an enumerative coding scheme, we assign $m_a = h_{\leq\delta}^{-1}(\boldsymbol{a})$. The decoding is now straight forward, and is described in the following formal description of the construction.

Construction 18. (A constant-weight weak WOM code from a concentrated-weight WOM code) *Let K_C and n be positive integers and let w_s be in $[0,1]$, w_x be in $[0, w_s]$ and δ in $[0, 1/2]$. Let D_C be an $(n, K_C, w_s, w_x, \delta)$ concentrated-weight WOM code, and define $K_W = K_C$ and $K_a = \sum_{i=0}^{\lfloor \delta n \rfloor} \binom{n}{i}$.*

An (n, K_W, K_a, w_s, w_x) constant-weight weak WOM coding scheme $\{E_W, D_W\}$ is defined as follows:

The ***encoding*** *function E_W receives a message $m \in [K_W]$ and a state vector $\boldsymbol{s} \in J_{w_x}(n)$, and finds a pair $(\boldsymbol{x}, m_a)$ in $D_W^{-1}(m) \subseteq J_{w_x}(n) \times [K_a]$ such that $\boldsymbol{x} \leq \boldsymbol{s}$. It is constructed as follows:*

1) *Let $\boldsymbol{x}_C \Leftarrow E_C(\boldsymbol{s}, m)$.*
2) *Let $\boldsymbol{a}$ be an arbitrary vector of weight $|\lfloor w_x n \rfloor - w_H(\boldsymbol{x}_C)|$ such that $\boldsymbol{a} \leq \boldsymbol{s} \oplus \boldsymbol{x}_C$ if $w_H(\boldsymbol{x}_C) \leq w_x n$ and $\boldsymbol{a} \leq \boldsymbol{x}_C$ otherwise.*
3) *Return the pair $(\boldsymbol{x}, m_a) \Leftarrow (\boldsymbol{x}_C \oplus \boldsymbol{a}, h_{\leq\delta}^{-1}(\boldsymbol{a}))$.*

The ***decoding*** *function D_W receives the stored pair $(\boldsymbol{x}, m_a) \in J_{w_x}(n) \times [K_a]$, and finds the stored message $m \in [K_W]$. It is constructed as follows:*

1) *Let $\boldsymbol{a} \Leftarrow h_{\leq\delta}(m_a)$.*
2) *Let $\boldsymbol{x}_C \Leftarrow \boldsymbol{x} \oplus \boldsymbol{a}$.*
3) *Return $m \Leftarrow D_C(\boldsymbol{x}_C)$.*

Theorem 19. *Let $\{E_C, D_C\}$ be a member of an efficient capacity-achieving family of concentrated-weight WOM coding schemes. Then Construction 18 describes an efficient capacity-achieving family of constant-weight weak WOM coding schemes.*

Proof: First, since $E_C, D_C, h_{\leq\delta}$ and $h_{\leq\delta}^{-1}$ can be performed in polynomial time in n, it follows directly that E_W and D_W can also be performed in polynomial time in n. Next, we show that the family of coding schemes is capacity achieving. For large enough n we have $(1/n)\log K_W > C_W - \epsilon_C$. So

$$R_W = (1/n)\log(K_W/K_a) > C_W - \epsilon_C - H(\delta),$$

since $\log K_a = \log \sum_{i=0}^{\lfloor \delta n \rfloor} \binom{n}{i} \leq nH(\delta)$ by Stirling's formula. Now, given $\epsilon_W > 0$, we let $\epsilon_C = \epsilon_W/2$ and $\delta = H^{-1}(\epsilon_W/2)$ such that $\epsilon_C + H(\delta) = \epsilon_W$. So for large enough n we have $R_W > C_W - \epsilon_C - H(\delta) = C_W - \epsilon_W$. ■

B. Polar WOM Codes

There are two properties of polar WOM coding schemes that do not fit well in our model. First, the scheme requires the presence of common randomness, known both to the encoder and to the decoder. Such an assumption brings some weakness to the construction, but can find some justification in a practical applications such as flash memory devices. For example, the common randomness can be the address of the storage location within the device. Second, the proposed encoding algorithm for polar WOM coding schemes does not always succeed in finding a correct codeword for the encoded message. In particular the algorithm is randomized, and it only guarantees to succeed with high probability, over the algorithm randomness, the common randomness, and a probabilistic assumption of a uniformly distributed message.

Nonetheless, for flash memory application, this assumption can be justified by the fact that such failure probability is much smaller than the unreliable nature of the devices. Therefore, some error-correction capability must be included in the construction for such practical implementation, and a failure of the encoding algorithm will not significantly affect the decoding failure rate. More approaches to tackle this issue are described in [6].

The construction is based on the method of channel polarization, which was first proposed by Arikan in his seminal paper [1] in the context of channel coding. We describe it here briefly by its application for WOM coding. This application is based on the use of polar coding for lossy source coding, that was proposed by Korada and Urbanke [22].

Let n be a power of 2, and let $G_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $G_2^{\otimes \log n}$ be its $\log n$-th Kronecker product. Consider a memoryless channel with a binary-input and transition probability $W(y|x)$. Define a vector $\boldsymbol{u} \in \{0,1\}^n$, and let $\boldsymbol{x} = \boldsymbol{u}G_2^{\otimes \log n}$, where the matrix multiplication is over $\mathbb{F}_2$. The vector $\boldsymbol{x}$ is the input to the channel, and $\boldsymbol{y}$ is the output vector. The main idea of polar coding is to define n sub-channels

$$W_n^{(i)}(\boldsymbol{y}, \boldsymbol{u}_{[i-1]}|u_i) = P(\boldsymbol{y}, \boldsymbol{u}_{[i-1]}|u_i) = \frac{1}{2^{n-1}} \sum_{\boldsymbol{u}_{[i+1:n]}} P(\boldsymbol{y}|\boldsymbol{u}).$$

For large n, each sub-channel is either very reliable or very noisy, and therefore it is said that the channel is polarized. A useful measure for the reliability of a sub-channel $W_n^{(i)}$ is its Bhattacharyya parameter, defined by

$$Z(W_n^{(i)}) = \sum_{y \in \mathcal{Y}} \sqrt{W_n^{(i)}(y|0)W_n^{(i)}(y|1)}. \tag{3}$$

Consider now a write-once memory. Let $\boldsymbol{s} \in \{0,1\}^n$ be the state vector, and let w_s be the fraction of 1's in $\boldsymbol{s}$. In addition, assume that a user wishes to store the message $m \in K_C$ with a codeword $\boldsymbol{x} \in J_{w_x}(n, \delta)$. The following scheme allows a rate arbitrarily close to C_W for n sufficiently large. The construction uses a compression scheme, based on a *test channel*. Let x be a binary input to the channel, and (s, g) be the output, where s and g are binary variables as well. Denote $x = g \oplus v$. The probability transition function of the channel is given by

$$W(s, g|v) = \begin{cases} w_s - w_x & \text{if } (s,x) = (1,0), \\ w_x & \text{if } (s,x) = (1,1), \\ 1 - w_s & \text{if } (s,x) = (0,0), \\ 0 & \text{if } (s,x) = (0,1). \end{cases}$$

The channel is polarized by the sub-channels $W_n^{(i)}$ of Equation 3, and a *frozen set* F is defined by

$$F = \left\{ i \in [n] : Z(W_n^{(i)}) \geq 1 - 2\delta_n^2 \right\},$$

where $\delta_n = 2^{-n^\beta}/(2n)$, for $0 < \beta < 1/2$. It is easy to show that the capacity of the test channel is $C_T = 1 - C_W$. It was shown in [22] that $|F| = n(C_T + \epsilon_C) = n(1 - C_W + \epsilon_C)$, where ϵ_C is arbitrarily small for n sufficiently large. Let $\boldsymbol{g}$ be a common randomness source from an n dimensional uniformly distributed random binary vector. The coding scheme is the following:

Construction 20. (A Polar WOM code [6]) *Let n be a positive integer and let w_s be in $[0,1]$, w_x be in $[0, w_s]$ and δ in $[0, 1/2]$. Let ϵ_C be in $[0, 1/2]$ such that $K_C = 2^{n(C_W - \epsilon_C)}$ is an integer.*

*The **encoding** function E_C receives a message $\boldsymbol{m} \in \{0,1\}^{\lceil \log K_C \rceil}$, a state vector $\boldsymbol{s} \in J_{w_s}(n)$ and the dither vector $\boldsymbol{g} \in \{0,1\}^n$, and returns a vector $\boldsymbol{x} \leq \boldsymbol{s}$ in $D_C^{-1}(\boldsymbol{m}) \subseteq J_{w_x}(n, \delta)$ with high probability. It is constructed as follows:*

1) *Assign $y_j = (s_j, g_j)$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$.*
2) *Define a vector $\boldsymbol{u} \in \{0,1\}^n$ such that $\boldsymbol{u}_F = \boldsymbol{m}$.*

3) *Create a vector $\hat{\boldsymbol{u}} \in \{0,1\}^n$ by compressing the vector $\boldsymbol{y}$ according to the following successive cancellation scheme: For $i = 1, 2, \ldots, n$, let $\hat{u}_i = u_i$ if $i \in F$. Otherwise, let*

$$\hat{u}_i = \begin{cases} 0 & \text{w.p. } L_n^{(i)}/(L_n^{(i)}+1) \\ 1 & \text{w.p. } 1/(L_n^{(i)}+1) \end{cases},$$

where w.p. denotes with probability and

$$L_n^{(i)} = L_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_{[i-1]}) = \frac{W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_{[i-1]}|u_i = 0)}{W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_{[i-1]}|u_i = 1)}.$$

4) *Assign $\boldsymbol{v} \Leftarrow \hat{\boldsymbol{u}} G_2^{\otimes \log n}$.*
5) *Return $\boldsymbol{x} \Leftarrow \boldsymbol{v} \oplus \boldsymbol{g}$.*

The ***decoding*** *function D_C receives the stored vector $\boldsymbol{x} \in J_{w_x}(n, \delta)$ and the dither vector $\boldsymbol{g} \in \{0,1\}^n$, and finds the stored message $\boldsymbol{m} \in \{0,1\}^{\lceil \log K_C \rceil}$. It is constructed as follows:*

1) *Assign $\boldsymbol{v} \Leftarrow \boldsymbol{x} \oplus \boldsymbol{g}$.*
2) *Assign $\hat{\boldsymbol{u}} \Leftarrow \boldsymbol{v}(G_2^{\otimes \log n})^{-1}$.*
3) *Return $\boldsymbol{m} \Leftarrow \hat{\boldsymbol{u}}_F$.*

In [6], a few slight modifications for this scheme are described, for the sake of the proof. We use the coding scheme (E_C, D_C) of Construction 20 as an $(N, K_C, w_s, w_x, \delta)$ concentrated-weight WOM coding scheme, even though it does not meet the definition precisely.

By the proof of Lemma 1 of [6], for $0 < \beta < 1/2$, the vector $\boldsymbol{x}$ found by the above encoding algorithm is in $D_C^{-1}(\boldsymbol{m})$ and in $J_{w_x}(n, \delta)$ w.p. at least $1 - 2^{-n^\beta}$ for n sufficiently large. Therefore, the polar WOM scheme of Construction 20 can be used as a practical concentrated-weight WOM coding scheme for the construction of RM rewriting codes by Constructions 15 and 18. Lemma 1 of [6] also proves that this scheme is capacity achieving. By the results in [22], the encoding and the decoding complexities are $O(n \log n)$, and therefore the scheme is efficient. This completes our first full description of a RM rewrite coding scheme in this paper, although it does not meet the definitions of Section II precisely. In the next section we describe a construction of efficient capacity-achieving RM rewrite coding schemes that meet the definitions of Section II.

V. Rank-Modulation Schemes from Hash WOM Schemes

The construction in this section is based on a recent construction of WOM codes by Shpilka [27]. This will require an additional modification to Construction 15 of RM rewrite coding schemes.

A. Rank-Modulation Schemes from Concatenated WOM Schemes

The construction of Shpilka does not meet any of our previous definitions of WOM codes. Therefore, we define yet another type of WOM codes, called ”constant-weight concatenated WOM codes”. As the name implies, the definition is a string concatenation of constant-weight WOM codes.

Definition 21. (Constant-weight concatenated WOM codes) *Let K_W, K_a, n and t be positive integers and let w_s be a real number in $[0,1]$ and w_x be a real number in $[0, w_s]$. A surjective function $D_W : (J_{w_x}(n))^t \times [K_a] \to [K_W]$ is an $(n, t, K_W, K_a, w_s, w_x)$* ***constant-weight concatenated WOM code*** *if for each message $m \in [K_W]$ and state vector $\boldsymbol{s} \in (J_{w_s}(n))^t$, there exists a pair $(\boldsymbol{x}, m_a)$ in the subset $D_W^{-1}(m) \subseteq (J_{w_x}(n))^t \times [K_b]$ such that $\boldsymbol{x} \leq \boldsymbol{s}$.*

Note that the block length of constant-weight concatenated WOM codes is nt, and therefore their rate is defined to be $R_W = \frac{1}{nt} \log K_W$. Since concatenation does not change the code rate, the capacity of constant-weight concatenated WOM codes is $C_W = w_s H(w_x/w_s)$. We define the notion of coding schemes, capacity achieving and efficient family of schemes accordingly. Next, we use constant-weight concatenated WOM coding schemes to construct RM rewrite coding schemes by a similar concatenation.

Construction 22. (A RM rewriting scheme from a constant-weight concatenated WOM scheme) *Let K_W, K_a, q, r, t and z_W be positive integers, and let $n_W = qz_W$. Let D_W be an $(n_W, t, K_W, K_a, (r+1)/q, 1/q)$ constant-weight concatenated WOM code with encoding algorithm E_W, and let a be the smallest integer for which $|\mathfrak{S}_{r+1,a}| \geq K_b$. Define the multiset $M = \{(q-r)^{z_W}, (q-r+1)^{z_W}, \ldots, q^{z_W}\}$ and let $K_M = |\mathfrak{S}_M|$ and $K_R = K_M \cdot K_W^{q-r-1}$.*

Let $z = tz_W + (q-r-1)a$ and $n = qz$. Define a codebook $\mathcal{C} \subset \mathfrak{S}_{q,z}$ as a set of permutations $\pi \in \mathcal{C}$ in which π^{-1} is a string concatenation $(\pi_{a,1}^{-1}, \ldots, \pi_{a,q-r-1}^{-1}, \pi_b^{-1}, \pi_{x,1}^{-1}, \ldots, \pi_{x,t}^{-1})$ such that the following conditions hold:

1) *$\pi_{x,i} \in \mathfrak{S}_{q,z_W}$ for each $i \in [t]$.*
2) *$\pi_{a,i} \in \mathfrak{S}_{r+1,a}$ for each rank $i \in [q-r-1]$.*
3) *π_b is a permutation of the multiset $\left\{(r+2)^{(q-r-1)a}, (r+3)^{(q-r-1)a}, \ldots, q^{(q-r-1)a}\right\}$.*

Denote the string concatenation $(\pi_{x,1}^{-1}, \ldots, \pi_{x,t}^{-1})$ by π_W^{-1}, and denote σ_W in the same way. A (q, z, K_R, r) RM rewrite coding scheme $\{E_R, D_R\}$ is constructed as follows:

*The **encoding** function E_R receives a message $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$ and a state permutation $\sigma \in \mathcal{C}$, and finds a permutation π in $B_{q,z,r}(\sigma) \cap D_R^{-1}(\boldsymbol{m})$ to store in the memory. It is constructed as follows:*

```
1: for $i = 1$ to $q - r - 1$ do
2:     $\boldsymbol{s}_i \Leftarrow \theta_{n_W}(U_{1,i+r}(\sigma_W) \setminus U_{1,i-1}(\pi_W))$
3:     $(\boldsymbol{x}_i, m_{a,i}) \Leftarrow E_W(m_i, \boldsymbol{s}_i)$
4:     $\pi_W(i) \Leftarrow \theta_{n_W}^{-1}(\boldsymbol{x}_i)$
5:     $\pi_{a,i} \Leftarrow h_{r+1,a}(m_{a,i})$
6: end for
7: $\pi_{W,[q-r,q]} \Leftarrow h_M(m_{q-r})$
8: $\pi_b = \sigma_b$
```

*The **decoding** function D_R receives the stored permutation $\pi \in \mathcal{C}$, and finds the stored message $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$. It is constructed as follows:*

```
1: for $i = 1$ to $q - r - 1$ do
2:     $\boldsymbol{x}_i \Leftarrow \theta_{n_W}(\pi_W(i))$
3:     $m_{a,i} \Leftarrow h_{r+1,a}^{-1}(\pi_{a,i})$
4:     $m_i \Leftarrow D_W(\boldsymbol{x}_i, m_{a,i})$
5: end for
6: $m_{q-r} \Leftarrow h_M^{-1}(\pi_{W,[q-r,q]})$
```

Since again concatenation does not affect the rate of the code, the argument of the proof of Theorem 16 gives the following statement:

Theorem 23. *Let $\{E_W, D_W\}$ be a member of an efficient capacity-achieving family of constant-weight concatenated WOM coding schemes. Then the family of RM rewrite coding schemes in Construction 22 is efficient and capacity-achieving.*

B. Hash WOM Codes

In [27] Shpilka proposed a construction of efficient capacity-achieving WOM coding scheme. The proposed scheme follows the concatenated structure of Definition 21, but does not have a constant weight. In this subsection we describe a slightly modified version of the construction of Shpilka, that does exhibit a constant weight.

To describe the construction, we follow the definitions of Shpilka [27]. The construction is based on a set of hash functions. For positive integers n, k, l and field members $a, b \in \mathbb{F}_{2^n}$, define a map $H_{a,b}^{n,k,l} : \{0,1\}^n \to \{0,1\}^{k-l}$ as $H_{a,b}^{n,k,l}(\boldsymbol{x}) = (ax+b)_{[k-l]}$. This notation means that we compute the affine transformation $ax + b$ in $\mathbb{F}_{2^n}$, represent it as a vector of n bits using the natural map and then keep the first $k - l$ bits of this vector. We represent this family of maps by $\mathcal{H}^{n,k,l}$, namely

$$\mathcal{H}^{n,k,l} = \left\{ H_{a,b}^{n,k,l} | a, b \in \mathbb{F}_{2^n} \right\}.$$

The family $\mathcal{H}^{n,k,l}$ contains 2^{2n} functions. For an integer $m_a \in [2^{2n}]$, we let H_{m_a} be the m_a-th function in $\mathcal{H}^{n,k,l}$.

Construction 24. (A constant-weight concatenated WOM coding scheme from hash functions) *Let ϵ, δ be in $[0, 1/2]$, w_s in $[0, 1]$, w_x in $[0, w_s]$ and $c > 20$. Let $n = \lceil (c/\epsilon) \log(1/\epsilon) \rceil$, $k = \lfloor n(C_W - 2\epsilon/3) \rfloor$, $t_1 = \lfloor (1/\epsilon)^{c/12} - 1 \rfloor$ and $t_2 = 2^{\frac{4n}{\delta}}$. Finally, Let $t = t_1 t_2$, $K_b = 2^k$ and $K_a = 2^{2n}$. An $(n, t, K_b^t, K_a^{t_2}, w_s, w_x)$ constant-weight concatenated WOM code is defined as follows:*

*The **encoding** function E_W receives a message matrix $\boldsymbol{m} \in [K_b]^{t_1 \times t_2}$, a state matrix of vectors $\boldsymbol{s} \in (J_{w_s}(n))^{t_1 \times t_2}$, and returns a pair $(\boldsymbol{x}, \boldsymbol{m}_a)$ in $D_W^{-1}(\boldsymbol{m}) \subseteq (J_{w_x}(n))^{t_1 \times t_2} \times [K_a]^{t_2}$ such that for each $(i, j) \in [t_1] \times [t_2]$ we have $\boldsymbol{x}_{i,j} \le \boldsymbol{s}_{i,j}$. It is constructed as follows: For each $j \in [t_2]$, use a brute force search to find an index $m_{a,j} \in [K_a]$ and a vector $\boldsymbol{x}_j = (\boldsymbol{x}_{1,j}, \ldots, \boldsymbol{x}_{t_1,j})$ such that for all $i \in [t_1]$, the following conditions hold:*

1) $\boldsymbol{x}_{i,j} \le \boldsymbol{s}_{i,j}$.
2) $\boldsymbol{x}_{i,j} \in J_{w_x}(n)$.
3) $H_{m_{a,j}}(\boldsymbol{x}_{i,j}) = m_{i,j}$.

*The **decoding** function D_W receives the stored pair $(\boldsymbol{x}, \boldsymbol{m}_a) \in (J_{w_x}(n))^{t_1 \times t_2} \times [K_a]$, and returns the stored message $\boldsymbol{m} \in [K_b]^{t_1 \times t_2}$. It is constructed as follows: For each pair $(i, j) \in [t_1] \times [t_2]$, assign $m_{i,j} \Leftarrow H_{m_{a,j}}(\boldsymbol{x}_{i,j})$.*

The only conceptual difference between Construction 24 and the construction in [27] is that here we require the vectors $\boldsymbol{x}_{i,j}$ to have a constant weight of $\lfloor w_x n \rfloor$, while the construction in [27] requires the weight of those vectors to be only bounded by $w_x n$. This difference is crucial for the rank-modulation application, but in fact it has almost no effect on the proofs of the properties of the construction.

To prove that the code in Construction 24 is a constant-weight concatenated WOM code, we will need the following lemma from [27]:

Lemma 25. *[27, Corollary 2.3]: Let k', ℓ, t_1 and n be positive integers such that $\ell \le k' \le n$ and $t_1 < 2^{\ell/4}$. Let $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{t_1} \subseteq \{0,1\}^n$ be sets of size $|\boldsymbol{X}_1|, \ldots, |\boldsymbol{X}_{t_1}| \ge 2^{k'}$. Then, for any $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{t_1} \in \{0,1\}^{k'-\ell}$ there exists $H_m \in \mathcal{H}^{n,k',\ell}$ and $\{\boldsymbol{x}_i \in \boldsymbol{X}_i\}$ such that for all $i \in [t_1]$, $H_m(\boldsymbol{x}_i) = \boldsymbol{m}_i$.*

Lemma 25 is proven using the leftover hash lemma [2, pp. 445], [5], [16] and the probabilistic method.

Proposition 26. *The code D_W of Construction 24 is an $(n, t, K_b^t, K_a^{t_2}, w_s, w_x)$ constant-weight concatenated WOM code.*

Proof: The proof is almost the same as the proof of Lemma 2.4 in [27], except that here the codewords' weight is constant. Let $\ell = \lceil \epsilon n/3 \rceil$, $k' = k + \ell$ and

$$\boldsymbol{X}_i = \{\boldsymbol{x} \in \{0,1\}^n | \boldsymbol{x} \le \boldsymbol{s}_i \text{ and } \boldsymbol{x} \in J_{w_x}(n)\}.$$

Since $\boldsymbol{x} \in J_{w_x}(n)$, we have that

$$|\boldsymbol{X}_i| = \binom{\lfloor w_s n \rfloor}{\lfloor w_x n \rfloor}$$

which by Stirling's formula can be lower bounded by

$$\begin{aligned} &\ge 2^{w_s n H(w_x/w_s) - \log(w_s n)} \ge 2^{n C_W - \log n} \\ &\qquad \ge 2^{n C_W - \epsilon n/3} = 2^{k'} \end{aligned}$$

For the last inequality we need $\epsilon n \ge 3 \log n$, which follows from

$$\frac{3 \log n}{\epsilon n} < \frac{3 \log[(2c/\epsilon) \log(1/\epsilon)]}{c \log(1/\epsilon)} < \frac{3 \log[(40/\epsilon) \log(1/\epsilon)]}{20 \log(1/\epsilon)} < 1.$$

Notice also that

$$t_1 = \lfloor (1/\epsilon)^{c/12} - 1 \rfloor < (1/\epsilon)^{c/12} = 2^{\frac{1}{4} \frac{\epsilon}{3} \frac{c}{\epsilon} \log(1/\epsilon)} \le 2^{\frac{1}{4} \frac{\epsilon n}{3}} \le 2^{\ell/4}.$$

So all of the conditions of Lemma 25 are met, which implies that the encoding of Construction 24 is always successful, and thus that D_W is a constant-weight concatenated WOM code. ■

Theorem 27. *Construction 24 describes an efficient capacity-achieving family of concatenated WOM coding schemes.*

Proof: We first show that the family is capacity achieving. We will need the following inequality:

$$\frac{2}{t_1} = \frac{2}{\lfloor (1/\epsilon)^{c/12} - 1 \rfloor} < 4\epsilon^{5/3} < \epsilon/3.$$

Now the rate can be bounded bellow as follows:

$$\begin{aligned} R_W &= \frac{t \log K_b - t_2 \log K_a}{nt} \\ &= \frac{t_1 \log K_b - \log K_a}{nt_1} \\ &= \frac{t_1 k - 2n}{nt_1} \\ &\geq \frac{t_1(C_W - 2\epsilon/3) - 2}{t_1} \\ &> C_W - 2\epsilon/3 - \epsilon/3 \\ &= C_W - \epsilon, \end{aligned}$$

and therefore the family is capacity achieving.

To show that the family is efficient, denote the block length of the code as $N = nt$. The encoding time is

$$t_2 |\mathcal{H}^{n,k,\ell}| \cdot \sum_{i=1}^{t_1} |\boldsymbol{X}_i| \leq t_2 t_1 2^{3n} < t_2 2^{4n} = t_2^{1+\delta} < N^{1+\delta},$$

and the decoding time is

$$t_2 \cdot \text{poly}(kt_1 n) = 2^{4n/\delta}(2/\epsilon)^{O(c)} < N \cdot 2^{O(n\epsilon)} = N \cdot N^{O(\delta\epsilon)} = N^{1+O(\delta\epsilon)}.$$

This completes the proof of the theorem. ■

Remark: Note that t_2 is exponential in $1/\epsilon$, and therefore the block length N is exponential in $(1/\epsilon)$. This can be an important disadvantage for these codes. In comparison, it is likely that the block length of polar WOM codes is only polynomial in $(1/\epsilon)$, since a similar results was recently shown in [14] for the case of polar lossy source codes, on which polar WOM codes are based.

We also note here that it is possible that the WOM codes of Gabizon and Shaltiel [12] could be modified for constant weight, to give RM rewriting codes with short block length without the dither and error probability of polar WOM codes.

VI. Conclusions

In this paper we studied the limits of rank-modulation rewriting codes, and presented two capacity-achieving code constructions. The construction of Section V, based on hash functions, has no possibility of error, but require a long block length that might not be considered practical. On the other hand, the construction of section IV, based on polar codes, appears to have a shorter block length, but requires the use of common randomness and exhibit a small probability of error. Important open problems in this area include the rate of convergence of polar WOM codes and the study of error-correcting rewriting codes. Initial results regarding error-correcting polar WOM codes were proposed in [18].

VII. Acknowledgments

This work was partially supported by the NSF grants ECCS-0801795 and CCF-1217944, NSF CAREER Award CCF-0747415, BSF grant 2010075 and a grant from Intellectual Ventures.

Appendix A

Proof of Lemma 3: We want to prove that if $\boldsymbol{s}$ is in Q_M and π is in $\mathfrak{S}_M$, then

$$\alpha(\boldsymbol{s} \to \pi) \leq \max_{j\in[n]}\{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}$$

with equality if $\Gamma_{\boldsymbol{s}}(q) - \Gamma_{\boldsymbol{s}}(1) = q-1$.

First, we claim by induction on $i \in [q]$ that

$$\Gamma_{\boldsymbol{s}}(i) \leq \Gamma_{\boldsymbol{s}}(q) + i - q. \tag{4}$$

We take $i = q$ as the base case, which holds trivially. Note also that since the cell-state vector $\boldsymbol{s}$ is in Q_M, we have $\Gamma_{\boldsymbol{s}}(i) < \Gamma_{\boldsymbol{s}}(i+1)$ for all $i \in [q-1]$. Now for the inductive step we use $\Gamma_{\boldsymbol{s}}(i-1) \leq \Gamma_{\boldsymbol{s}}(i) - 1$ and the inductive hypothesis and get

$$\Gamma_{\boldsymbol{s}}(i-1) \leq \Gamma_{\boldsymbol{s}}(i) - 1 \leq \Gamma_{\boldsymbol{s}}(q) + i - q - 1,$$

which proves the induction claim. Note that the condition $\Gamma_{\boldsymbol{s}}(q) - \Gamma_{\boldsymbol{s}}(1) = q-1$ in the statement of the lemma implies that $\Gamma_{\boldsymbol{s}}(i) = \Gamma_{\boldsymbol{s}}(i+1) - 1$ for all $i \in [q-1]$, and therefore equality in Equation 4.

Next, define a set $U_{i_1,i_2}(\sigma_{\boldsymbol{s}})$ to be the union of the sets $\{\sigma_{\boldsymbol{s}}(i)\}_{i\in[i_1:i_2]}$, and remember that the writing process sets $x_j = s_j$ if $\pi^{-1}(j) = 1$, and otherwise

$$x_j = \max\{s_j, \Gamma_{\boldsymbol{x}}(\pi^{-1}(j) - 1) + 1\}.$$

Now we claim by induction on $i \in [q]$ that

$$\Gamma_{\boldsymbol{x}}(i) \leq i + \Gamma_{\boldsymbol{s}}(q) - q + \max_{j\in U_{1,i}(\pi)}\{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}. \tag{5}$$

In the base case, $i = 1$, and

$$\begin{aligned}\Gamma_{\boldsymbol{x}}(1) &\overset{(a)}{=} \max_{j\in\pi(1)}\{x_j\} \overset{(b)}{=} \max_{j\in\pi(1)}\{s_j\} \overset{(c)}{\leq} \max_{j\in\pi(1)}\{\Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j))\} \overset{(d)}{\leq} \max_{j\in\pi(1)}\{\Gamma_{\boldsymbol{s}}(q) - q + \sigma_{\boldsymbol{s}}^{-1}(j)\}\\ &\overset{(e)}{=}\Gamma_{\boldsymbol{s}}(q) - q + \max_{j\in\pi(1)}\{\sigma_{\boldsymbol{s}}^{-1}(j) + (1 - \pi^{-1}(j))\} \overset{(f)}{=} 1 + \Gamma_{\boldsymbol{s}}(q) - q + \max_{j\in U_{1,i}(\pi)}\{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}\end{aligned}$$

Where (a) follows from the definition of $\Gamma_{\boldsymbol{x}}(1)$, (b) follows from the modulation process, (c) follows since $\Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j)) = \max_{j'\in\sigma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j))}\{s_{j'}\}$, and therefore $\Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j)) \geq s_j$ for all $j \in [n]$, (d) follows from Equation 4, (e) follows since $j \in \pi(1)$, and therefore $\pi^{-1}(j) = 1$, and (f) is just a rewriting of the terms. Note that the condition $\Gamma_{\boldsymbol{s}}(q) - \Gamma_{\boldsymbol{s}}(1) = q-1$ implies that $s_j = \Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j))$ and $\Gamma_{\boldsymbol{s}}(i) \leq \Gamma_{\boldsymbol{s}}(q) + i - q$, and therefore equality in (c) and (d).

For the inductive step, we have

$$\begin{aligned}\Gamma_{\boldsymbol{x}}(i) &\overset{(a)}{=} \max_{j\in\pi(i)}\{x_i\}\\ &\overset{(b)}{=} \max_{j\in\pi(i)}\{\max\{s_j, \Gamma_{\boldsymbol{x}}(i-1) + 1\}\}\\ &\overset{(c)}{\leq} \max\{\max_{j\in\pi(i)}\{s_j\}, (i-1) + \Gamma_{\boldsymbol{s}}(q) - q + \max_{j\in U_{1,i-1}(\pi)}\{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\} + 1\}\\ &\overset{(d)}{\leq} \max\{\max_{j\in\pi(i)}\{\Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j))\}, i + \Gamma_{\boldsymbol{s}}(q) - q + \max_{j\in U_{1,i-1}(\pi)}\{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}\}\end{aligned}$$

$$\begin{aligned}
&\overset{\text{(e)}}{\leq} \max\{\max_{j\in\pi(i)}\{\Gamma_{\boldsymbol{s}}(q)-q+\sigma_{\boldsymbol{s}}^{-1}(j)\}, i+\Gamma_{\boldsymbol{s}}(q)-q+\max_{j\in U_{1,i-1}(\pi)}\{\sigma_{\boldsymbol{s}}^{-1}(j)-\pi^{-1}(j)\}\} \\
&\overset{\text{(f)}}{=}\Gamma_{\boldsymbol{s}}(q)-q+\max\{\max_{j\in\pi(i)}\{\sigma_{\boldsymbol{s}}^{-1}(j)+(i-\pi^{-1}(j))\}, i+\max_{j\in U_{1,i-1}(\pi)}\{\sigma_{\boldsymbol{s}}^{-1}(j)-\pi^{-1}(j)\}\} \\
&\overset{\text{(g)}}{=}i+\Gamma_{\boldsymbol{s}}(q)-q+\max\{\max_{j\in\pi(i)}\{\sigma_{\boldsymbol{s}}^{-1}(j)-\pi^{-1}(j)\}, \max_{j\in U_{1,i-1}(\pi)}\{\sigma_{\boldsymbol{s}}^{-1}(j)-\pi^{-1}(j)\}\} \\
&\overset{\text{(h)}}{=}i+\Gamma_{\boldsymbol{s}}(q)-q+\max_{j\in U_{1,i}(\pi)}\{\sigma_{\boldsymbol{s}}^{-1}(j)-\pi^{-1}(j)\}
\end{aligned}$$

Where (a) follows from the definition of $\Gamma_{\boldsymbol{x}}(i)$, (b) follows from the modulation process, (c) follows from the induction hypothesis, (d) follows from the definition of $\Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j))$, (e) follows from Equation 4, (f) follows since $\pi^{-1}(j)=i$, and (g) and (h) are just rearrangements of the terms. This completes the proof of the induction claim. As in the base case, we see that if $\Gamma_{\boldsymbol{s}}(q)-\Gamma_{\boldsymbol{s}}(1)=q-1$ then the inequality in Equation 5 becomes an equality.

Finally, taking $i=q$ in Equation 5 gives

$$\Gamma_{\boldsymbol{x}}(q)\leq q+\Gamma_{\boldsymbol{s}}(q)-q+\max_{j\in U_{1,q}(\pi)}\{\sigma_{\boldsymbol{s}}^{-1}(j)-\pi^{-1}(j)\}=\Gamma_{\boldsymbol{s}}(q)+\max_{j\in[n]}\{\sigma_{\boldsymbol{s}}^{-1}(j)-\pi^{-1}(j)\}$$

with equality if $\Gamma_{\boldsymbol{s}}(q)-\Gamma_{\boldsymbol{s}}(1)=q-1$, which completes the proof of the lemma, since $\alpha(\boldsymbol{s}\to\pi)$ was defined as $\Gamma_{\boldsymbol{x}}(q)-\Gamma_{\boldsymbol{s}}(q)$. ■

Appendix B

Proof of Lemma 11: The proof follows a similar proof by Heegard [15], for the case where the codewords' weight is not necessarily constant. Given a state $\boldsymbol{s}$, the number of vectors $\boldsymbol{x}$ of weight $\lfloor w_x n\rfloor$ such that $\boldsymbol{x}\leq\boldsymbol{s}$ is at most $\binom{\lfloor w_s n\rfloor}{\lfloor w_x n\rfloor}$. Since K_W cannot be greater than this number, we have

$$R_W=(1/n)\log K_W\leq(1/n)\log\binom{\lfloor w_s n\rfloor}{\lfloor w_x n\rfloor}\leq(1/n)\log 2^{w_s nH(w_x/w_s)}=C_W,$$

where the last inequality follows from Stirling's formula. Therefore, the capacity is at most C_W.

The lower bound on the capacity is proven by the probabilistic method. Randomly and uniformly partition $J_{w_x}(n)$ into K_W subsets of equal size,

$$|D_W^{-1}(m)|=|J_{w_x}|/2^{-nR_W}.$$

Fix $m\in[K_W]$ and $\boldsymbol{s}\in J_{w_s}(n)$, and let $\beta(\boldsymbol{s})$ be the set of vectors $\boldsymbol{x}\in J_{w_x}(n)$ such that $\boldsymbol{x}\leq\boldsymbol{s}$. Then

$$\begin{aligned}
P(D_W^{-1}(m)\cap\beta(\boldsymbol{s})=\emptyset)&=\prod_{i=0}^{|D_W^{-1}(m)|-1}\frac{|J_{w_x}(n)|-|\beta(\boldsymbol{s})|-i}{|J_{w_x}(n)|-i} \\
&\leq\left(\frac{|J_{w_x}(n)|-|\beta(\boldsymbol{s})|}{|J_{w_x}(n)|}\right)^{|D_W^{-1}(m)|}.
\end{aligned}$$

$|\beta(\boldsymbol{s})|\geq 2^{nC_W-\log(w_s n)}$, and thus

$$\begin{aligned}
P(D_W^{-1}(m)\cap\beta(\boldsymbol{s})=\emptyset)&\leq(1-|J_{w_x}(n)|^{-1}2^{nC_W-\log(w_s n)})^{|J_{w_x}|2^{-nR_W}} \\
&<e^{-(2^{n(C_W-R_W)-\log(w_s n)})},
\end{aligned}$$

where the last inequality follows from the fact that $(1-x)^y < e^{-xy}$ for $y > 0$. If $R_W < C_W$, this probability vanishes for large n. In addition,

$$\begin{aligned}P(\exists m \in [K_W] \text{ and } \boldsymbol{s} \in J_{w_s}(n) \text{ s.t. } D_W^{-1}(m) \cap \beta(\boldsymbol{s}) = \emptyset)&\\ = P\left(\cup_{m\in[K_W]} \cup_{\boldsymbol{s}\in J_{w_s}(n)} \left\{D_W^{-1}(m) \cap \beta(\boldsymbol{s}) = \emptyset\right\}\right)&\\ \le \sum_{m\in[K_W]} \sum_{\boldsymbol{s}\in J_{w_s}(n)} P(D_W^{-1}(m) \cap \beta(\boldsymbol{s}) = \emptyset)&\\ \le 2^{n(R_W+H(w_s))} e^{-(2^{n(C_W-R_W)-\log(w_s n)})}&\end{aligned}$$

This means that if $R_W < C_W$ and n is large enough, the probability that the partition is not a constant-weight strong WOM code approaches 0, and therefore there exists such a code, completing the proof. ■

APPENDIX C

Proof of Theorem 16: We will first show that $\{E_R, D_R\}$ is capacity achieving, and then show that it is efficient. Let $R_R = (1/n)\log K_R$ be the rate of a RM rewriting code. To show that $\{E_R, D_R\}$ is capacity achieving, we need to show that for any $\epsilon_R > 0$, $R_R > C_R - \epsilon_R$, for some q and z.

Since $\{E_W, D_W\}$ is capacity achieving, $R_W > C_W - \epsilon_W$ for any $\epsilon_W > 0$ and large enough n. Remember that $C_W = w_s H(w_x/w_s)$. In $\{E_R, D_R\}$ we use $w_s = (r+1)/q$ and $w_x = 1/q$, and so $C_W = \frac{r+1}{q} H\left(\frac{1}{r+1}\right)$. We will need to use the inequality $\log K_a > a$, which follows from:

$$\log K_a > \log|\mathfrak{S}_{r+1,a-1}| > \log|\mathfrak{S}_{2,a-1}| > 2a - 2 - \log 2a > a$$

Where the last inequality requires a to be at least 6. In addition, we will need the inequality $n_W/n > 1 - q^2\epsilon_W$, which follows form:

$$\begin{aligned}\frac{n_W}{n} &= \frac{n_W}{n_W + q(q-r-1)a} > \frac{n_W}{n_W + q^2 a} > 1 - \frac{q^2 a}{n_W} > 1 - \frac{q^2 \log K_a}{n_W}\\ &= 1 - q^2\left(\frac{\log K_W}{n_W} - \frac{\log(K_W/K_a)}{n_W}\right) > 1 - q^2(C_W - (C_W - \epsilon_W)) = 1 - q^2\epsilon_W.\end{aligned}$$

Now we can bound the rate from below, as follows:

$$\begin{aligned}R_R &= (1/n)\log K_R\\ &= (1/n)\log(K_M \cdot K_W^{q-r-1})\\ &> (q-r-1)(1/n)\log K_W\\ &> (q-r-1)(C_W - \epsilon_W)(n_W/n) \qquad (6)\\ &= (q-r-1)\left(\frac{r+1}{q}H\left(\frac{1}{r+1}\right) - \epsilon_W\right)(1 - q^2\epsilon_W)\\ &= \frac{q-r-1}{q}(C_R - q\epsilon_W)(1 - q^2\epsilon_W)\\ &= (C_R - q\epsilon_W)(1 - (r+1)/q)(1 - q^2\epsilon_W)\\ &> C_R - C_R q^2\epsilon_W - C_R(r+1)/q + (C_R(r+1)q\epsilon_W - q\epsilon_W) + (q^3\epsilon^2 - (r+1)q^2\epsilon_W^2)\\ &> C_R - (r+1)q^2\epsilon_W - (r+1)^2/q\end{aligned}$$

The idea is to take $q = \left\lfloor\left(\frac{r+1}{\epsilon_W}\right)^{1/3}\right\rfloor$ and $\epsilon_R = 3(r+1)^{2/3}\epsilon_W^{1/3}$ and get that

$$R_R > C_R - (r+1)\left\lfloor\left(\frac{r+1}{\epsilon_W}\right)^{1/3}\right\rfloor^2 \epsilon_W - \frac{(r+1)^2}{\left\lfloor\left(\frac{r+1}{\epsilon_W}\right)^{1/3}\right\rfloor} > C_R - (r+1)^{2/3}\epsilon_W^{1/3} - 2(r+1)^{2/3}\epsilon_W^{1/3} = C_R - \epsilon_R.$$

So we can say that for any $\epsilon_R > 0$ and integer r, we set $\epsilon_W = \frac{\epsilon_R^2}{9(r+1)^2}$ and $q = \lfloor (r+1)/\sqrt{\epsilon_W} \rfloor$. Now if z is large enough then $n = qz$ is also large enough so that $R_W > C_W - \epsilon_W$, and then Equation 2 holds and we have $R_R > C_R - \epsilon_R$.

Finally, we show that $\{E_R, D_R\}$ is efficient. If the scheme (h_M, h_M^{-1}) is implemented as described in [23], then the time complexity of h_M and h_M^{-1} is polynomial in n. In addition, we assumed that E_W and D_W run in polynomial time in n. So since h_M and h_M^{-1} are executed only once in E_R and D_R, and E_W and D_W are executed less than q times in E_R and D_R, where $q < n$, we get that the time complexity of E_R and D_R is polynomial in n. ■

References

[1] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. on Inform. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

[2] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*, 1st ed. New York, NY, USA: Cambridge University Press, 2009.

[3] A. Barg and A. Mazumdar, "Codes in permutations and error correction for rank modulation," *IEEE Trans. on Inform. Theory*, vol. 56, no. 7, pp. 3158–3165, Jul. 2010.

[4] E. F. Beckenbach (Editor), *Applied Combinatorial Mathematics*. New York, J. Wiley, 1964.

[5] C. H. Bennett, G. Brassard, and J.-M. Robert, "Privacy amplification by public discussion." *SIAM J. Comput.*, vol. 17, no. 2, pp. 210–229, Apr. 1988.

[6] D. Burshtein and A. Strugatski, "Polar write once memory codes," *IEEE Trans. on Inform. Theory*, vol. 59, no. 8, pp. 5088–5101, Aug. 2013.

[7] E. En Gad, E. Yaakobi, A. Jiang, and J. Bruck, "Rank-modulation rewriting codes for flash memories," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2013, pp. 704–708.

[8] E. En Gad, A. Jiang, and J. Bruck, "Compressed encoding for rank modulation," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aug. 2011, pp. 884–888.

[9] ——, "Trade-offs between instantaneous and total capacity in multi-cell flash memories," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2012, pp. 990–994.

[10] E. En Gad, M. Langberg, M. Schwartz, and J. Bruck, "Constant-weight gray codes for local rank modulation," *IEEE Trans. on Inform. Theory*, vol. 57, no. 11, pp. 7431–7442, Nov. 2011.

[11] F. Farnoud, V. Skachek, and O. Milenkovic, "Error-correction in flash memories via codes in the ulam metric," *IEEE Trans. on Inform. Theory*, vol. 59, no. 5, pp. 3003–3020, May 2013.

[12] A. Gabizon and R. Shaltiel, "Invertible zero-error dispersers and defective memory with stuck-at errors," in *APPROX-RANDOM*, 2012, pp. 553–564.

[13] S. I. Gelfand and M. S. Pinsker, "Coding for channels with random parameters," *Probl. Control and Info. Theory*, vol. 9, no. 1, pp. 19–31, 1980.

[14] D. Goldin and D. Burshtein, "Improved bounds on the finite length scaling of polar codes," in *arXiv:1307.5510 [cs.IT]*, 2013.

[15] C. D. Heegard, "On the capacity of permanent memory," *IEEE Trans. on Inform. Theory*, vol. 31, no. 1, pp. 34–42, Jan. 1985.

[16] R. Impagliazzo, L. A. Levin, and M. Luby, "Pseudo-random generation from one-way functions," in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, ser. STOC '89. New York, NY, USA: ACM, 1989, pp. 12–24.

[17] A. Jiang, V. Bohossian, and J. Bruck, "Rewriting codes for joint information storage in flash memories," *IEEE Trans. on Inform. Theory*, vol. 56, no. 10, pp. 5300–5313, Oct. 2010.

[18] A. Jiang, Y. Li, E. En Gad, M. Langberg, and J. Bruck, "Joint rewriting and error correction in write-once memories," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT),*, Jul. 2013, pp. 1067–1071.

[19] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," *IEEE Trans. on Inform. Theory*, vol. 55, no. 6, pp. 2659–2673, Jun. 2009.

[20] A. Jiang, M. Schwartz, and J. Bruck, "Correcting charge-constrained errors in the rank-modulation scheme," *IEEE Trans. on Inform. Theory*, vol. 56, no. 5, pp. 2112–2120, May 2010.

[21] M. Kim, J. K. Park, and C. Twigg, "Rank modulation hardware for flash memories," in *IEEE Int. Midwest Symp. on Circuits and Systems (MWSCAS)*, aug. 2012, pp. 294 –297.

[22] S. B. Korada and R. Urbanke, "Polar codes are optimal for lossy source coding," *IEEE Trans. on Inform. Theory*, vol. 56, no. 4, pp. 1751–1768, Apr. 2010.

[23] O. Milenkovic and B. Vasic, "Permutation (d,k) codes: efficient enumerative coding and phrase length distribution shaping," *IEEE Trans. on Inform. Theory*, vol. 46, no. 7, pp. 2671–2675, Jul. 2000.

[24] N. Papandreou, H. Pozidis, T. Mittelholzer, G. F. Close, M. Breitwisch, C. Lam, and E. Eleftheriou, "Drift-tolerant multilevel phase-change memory," in *IEEE Int. Memory Workshop (IMW)*, 2011, pp. 1–4.

[25] T. Ramabadran, "A coding scheme for m-out-of-n codes," *IEEE Trans. on Communications*, vol. 38, no. 8, pp. 1156–1163, Aug. 1990.

[26] R. L. Rivest and A. Shamir, "How to reuse a "write-once" memory," *Inform. and Control*, vol. 55, pp. 1–19, 1982.

[27] A. Shpilka, "Capacity achieving multiwrite wom codes," in *arXiv:1209.1128 [cs.IT]*, 2012.

[28] D. Slepian, "Permutation modulation," in *Proc. of the IEEE*, vol. 53, no. 3, 1965, pp. 228–236.

[29] I. Tamo and M. Schwartz, "Correcting limited-magnitude errors in the rank-modulation scheme," *IEEE Trans. on Inform. Theory*, vol. 56, no. 6, pp. 2551–2560, Jun. 2010.

[30] C. Tian, V. Vaishampayan, and N. Sloane, “A coding algorithm for constant weight vectors: A geometric approach based on dissections,” *IEEE Trans. on Inform. Theory*, vol. 55, no. 3, pp. 1051–1060, Mar. 2009.

[31] Z. Wang and J. Bruck, “Partial rank modulation for flash memories,” in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2010, pp. 864–868.