Read-Write-Codes
An Erasure Resilient Encoding System for Flexible Reading and Writing in Storage Networks

Mario Mense[1] and Christian Schindelhauer[2]

[1] Heinz Nixdorf Institute,
University of Paderborn, Germany
vodisek@upb.de
[2] Computer Networks and Telematics,
University of Freiburg, Germany
schindel@informatik.uni-freiburg.de

Abstract. We introduce the Read-Write-Coding-System (RWC) – a very flexible class of linear block codes that generate efficient and flexible erasure codes for storage networks. In particular, given a message x of k symbols and a codeword y of n symbols, an RW code defines additional parameters $k \leq r, w \leq n$ that offer enhanced possibilities to adjust the fault-tolerance capability of the code. More precisely, an RWC provides linear (n, k, d)-codes that have (a) minimum distance $d = n-r+1$ for any two codewords, and (b) for each codeword there exists a codeword for each other message with distance of at most w. Furthermore, depending on the values r, w and the code alphabet, different block codes such as parity codes (e.g. RAID 4/5) or Reed-Solomon (RS) codes (if $r = k$ and thus, $w = n$) can be generated. In storage networks in which I/O accesses are very costly and redundancy is crucial, this flexibility has considerable advantages as r and w can optimally be adapted to read or write intensive applications; only w symbols must be updated if the message x changes completely, what is different from other codes which always need to rewrite y completely as x changes. In this paper, we first state a tight lower bound and basic conditions for all RW codes. Furthermore, we introduce special RW codes in which all mentioned parameters are adjustable even online, that is, those RW codes are adaptive to changing demands. At last, we point out some useful properties regarding safety and security of the stored data.

1 Introduction

An erasure (resilient) code maps a word x of k symbols drawn from an alphabet Σ into a codeword y of $n > k$ symbols from the same alphabet, and in the optimal case, any k symbols from the n codeword symbols are sufficient to recover x. This property has made erasure codes become very prominent in many application areas [1, 7, 16]. In storage networks such as RAID-*arrays* [13, 14] and modern

storage area networks (SAN) [10] access to hard disks is comparably slow, and thus, data is scattered into fixed sized blocks which are evenly distributed among the storage devices to exploit access parallelism. If then some disks fail for reading (erasures), in the optimal case, any k symbols from y are sufficient to recover x, i.e. such codes can tolerate up to $n-k$ erasures which may be caused by failed, respectively temporarily not accessible disks. Since the number n of blocks (symbols) in a codeword y is fixed (called *data stripe*) and all blocks in a stripe are hosted by n different disk, *linear block codes* are mainly applied [13, 5, 14, 8]. More importantly, linear block codes are *optimal codes*, i.e. they only require any k blocks from y to recover x what is important in a scenario that suffers from expensive I/O operations.

However, most codes applied in RAID-like storage networks almost aim at providing a (near-)optimal recovery behavior but what implies a serious drawback as any code that is able to reconstruct x from up to $n-k$ erasures suffers from a bad update behavior. In particular, if one information symbol changes from x_i to x_i', any codesymbol y_i must also be modified. If then any of the disk keeping y_i is not accessible, there is no chance to store the modified codeword appropriately (except merely the plain information word if a systematic code is applied such as given, for example, with the RAID 4/5 encoding).

In order to face this negative update behavior that is inherent to usual linear blocks codes and which turns to be pretty costly when such codes are applied in RAID-like storage environments, we introduce the *Read-Write-Coding-System (RWC)* – a very flexible framework for generating different linear block codes, called *Read-Write (RW) codes* in the following, which feature enhanced update properties for given codewords by simultaneously offering different degrees of fault-tolerance. In contrast to common linear block codes, an RWC defines further parameters $k \leq r, w \leq n$ which offer enhanced possibilities to adjust the redundancy, and thus, the fault-tolerance capability of an RW code. In the language of coding theory, for any fixed r, an RWC provides linear (n, r, d)-codes over some finite field $\mathbb{F}_q$ that have (a) minimum (Hamming) distance $d = n-r+1$ (thus, are MDS codes if $r = k$), and (b) for each codeword there exists a codeword for each other message with distance of at most w, i.e. the two code words differ in at most w symbols. More specific, an RWC generates appropriate sub-codes of Reed-Solomon (RS) codes (see e.g. [9] for details on RS codes) of dimension r and length n in which any two codewords have distance at least w. Depending on the values r, w and the field $\mathbb{F}_q$ chosen, different block codes can be generated, e.g. parity codes (if $q = 2$).

The ensured degree of redundancy mixed up with the improved update behavior offered by an Read-Write code provides significant benefits for the observed storage systems which, driven by the application's read and write behavior, on one hand, suffer from very costly I/O operations, and on the other hand, have to ensure some defined level of fault-tolerance at any time. Clearly, a Read-Write code provides best improvements for write-intensive applications because, given an n-sized codeword y and parameters r, w, it can decode the information from **any** r symbols of y whereas only **any** w symbols of y must be updated

Contents		Code				Line
x_1	x_2	y_1	y_2	y_3	y_4	v
0	0	0	0	0	0	0
0	0	1	1	1	1	1
0	1	0	1	0	1	0
0	1	1	0	1	0	1
1	0	0	0	1	1	0
1	0	1	1	0	0	1
1	1	0	1	1	0	0
1	1	1	0	0	1	1

Table 1. A $(2,3,3,4)_2$-Read-Write-Code for contents x_1, x_2 and code y_1, y_2, y_3, y_4. Every information vector has two possible code words. Even if only three of the four code words are available for reading and writing, the system can perform read and write operations (see Figure 1 for the encoding). Variable v is an internal variable of our RW-code introduced in Chapter 5.

whenever the information word x changes completely (recall that we can choose $w < n$). Again, this is different to other linear codes, like e.g. Reed-Solomon codes, which always must rewrite the codeword y completely as x changes. This novel redundancy property comes with cost of additional necessary disk space for compensating the absence of writable disks, i.e. the data rate of RW-codes decreases with increasing w.

For most applications it is necessary to read the data and then to update the information such that the condition $w \geq r$ seems natural. Our encoding systems allows the update without prior reading, i.e. the difference vector δ of the old and new message is needed. E.g. if an empty file is overwritten with the first data entries then only w disks need to be written. In such cases RW-codes with $w < r$ become interesting.

An example. Consider a RAID 4-parity code with $n = 4$ hard disks storing a data file bit by bit, $\Sigma = \{0, 1\}$. We encode $k = 3$ bits x_1, x_2, x_3 to symbols $y_1 = x_1, y_2 = x_2, y_3 = x_3$ and $y_4 = x_1 + x_2 + x_3$, where addition denotes the XOR-operation and the code symbols y_i, $1 \leq i \leq 4$, are stored separately on distinct disks. The XOR-operation enables to recover the original three bits from any combination of $r = 3$ hard disks, e.g. giving y_2, y_3, y_4 we have $x_1 = y_2 + y_3 + y_4$, $x_2 = y_2$, and $x_3 = y_3$. Thus, if any one disk is temporarily not available, reading data is still possible. However then, writing data is not possible since a complete change of the original information involves the change of the entire code; we call this *code consistency* (this also holds for any other erasure code applied in storage environments). The second example shows an RW-code. Again, consider $n = 4$ hard disks with code bits y_1, y_2, y_3, y_4. Now, we encode $k = 2$ information bits x_1, x_2 such that any $r = 3$ code bits y_i, y_j, y_k can be used to recover the original message, and furthermore, only any of such $w = 3$ code bits $y_{i'}, y_{j'}, y_{k'}$ need to be changed to encode a completely new information. For instance, start with the codeword $(0, 1, 1, ?)$. According to Table 1, the information is $(1, 1)$ and therefore the complete code is $(0, 1, 1, 0)$. Now, we want to encode $(0, 1)$ without changing the second entry. For this, we choose line 0 for information $(0, 1)$ and get code $(0, 1, 0, 1)$.

Moreover, RW codes can exploit system information about existing erasures, which are caused by failed or blocked disks and that have rather long-term character in a SAN, for encoding and decoding. For instance, consider a codeword y with symbols stored on n disks from which $b \leq n - w$ disks are unreachable (e.g. failed or blocked). Then, using an RW code, y can still be updated to the codeword y' in a code consistent manner. Furthermore, if then some of the formerly blocked disks become available again while some other $b' \leq n - r$ disks turn to be unreachable, we can still recover the new information word x' by simply selecting any r of the remaining $n - b'$ accessible disks. Therefore, as long as sufficient disks are accessible, an RW code provides code consistent operations by circumventing blocked disks.

At last, some RW codes offer the possibility to change any of the parameters k, r, w and n during runtime, that is, a (k, r, w, n)-RW code can be changed to (nearly) any choice of (k', r', w', n') giving such codes the ability to adapt to changing system conditions.

2 Related Work

The most popular codes used are parity-based schemes, like RAID [13] or EVEN-ODD [4] that have low storage consumption which is given by a factor $(k+1)/k$ and $(k+2)/k$, respectively, and that base on simple but efficient XOR-operations. Unfortunately, parity codes are able to tolerate only one or two erasures at a time, what is often not sufficient, even in large SANs. Therefore, since in large SANs an increased fault-tolerance is often the major focus, a code should be used that provides a high minimum distance between any two codewords. Codes providing this feature are called *MDS codes* (Maximum Distance Separable), which ensure distance $d(Y) = n - k + 1$ for any two codewords [8, 9]. Nevertheless, many variants of MDS codes, like *MDS array codes* [3] or *X-codes* [2] also suffer from high rates. Alternatively, *Hamming codes* have good rate but distance of at most 3, and *Reed-Muller codes* have high distance but bad rate (c.f. [9]). Therefore, *Reed-Solomon (RS) codes* have become very popular in distributed storage systems [15, 11] and disk arrays [6, 14] since they combine a good rate of $(n - k)/k$ with distance $d(Y) = n - k + 1$. Unfortunately, as RS codes are MDS codes, they also suffer from an undesired update overhead because if x is modified, all blocks of y must be rewritten, what is dismal in a SAN suffering from expensive I/O accesses.

Thus, due to their beneficial properties, applying RW-codes in a SAN seems quite self-evident. From now on, we call a $(k, r, w, n)_b$-Read-Write code a coding system with a k-symbol message and an n-symbol code with symbols drawn from a b-symbol alphabet, and a parameter r for recovering the message and w for modification with $k \leq r, w \leq n$. In the next section, we state the operations of the RWC formally. After that, we prove general bounds for the parameters of RW codes and present a general scheme to generate $(k, r, w, n)_b$-RW codes as long as $k + n \leq r + w$ holds for an appropriate choice of b. Then, we introduce adaptive RW codes called *Chameleon codes*, where any of the given parameters

can be subject to changes. At last, notice that the following description is given in more operation-based terms rather than conceptual since RW codes base on the same well-studied algebraic principles as RS codes.

3 The Operational Model

Read-Write codes encode information words into codewords. The information is given by a k-tuple over some finite alphabet Σ, and since Read-Write codes are linear block codes, the codeword is an n-tuple over the same alphabet Σ, $k < n$. Now, for what follows, let $b = |\Sigma|$ and $\mathbf{P}(M)$ denotes the power set of some set M. Moreover, let $\mathbf{P}_\ell(M) := \{S \in \mathbf{P}(M) \mid |S| = \ell\}$.

Then, the following operations are provided by a $(k, r, w, n)_b$ Read-Write-Coding-System (RWC):

1. **Initial state** $x_0 \in \Sigma^k$, $y_0 \in \Sigma^n$
   This is the initial state of the system with information x_0 and codeword y_0. This state is crucial because all further operations ensuring the beneficial features of an RW code depend on this initial state.
2. **Read function** $f : \mathbf{P}_r([n]) \times \Sigma^r \to \Sigma^k$
   This function reconstructs the information by reading r symbols of the codeword whose positions are known. The first parameter shows the positions (indices) of the symbols in the code, and the second parameter gives the corresponding code symbols. The outcome is the decoded information.

3a. **Write function** g :
   $\mathbf{P}_r([n]) \times \Sigma^r \times \Sigma^k \times \mathbf{P}_w([n]) \to \Sigma^w$
   This function adapts the codeword to a changed information by changing w symbols of the codeword at w given positions. The first two parameters describe the reading of the original information. Then, we have the new information as a parameter, and the last parameter indicates which code symbols to change in the codeword. The outcome are the values of the new w code symbols.

3b. **Differential write function** $\delta : \mathbf{P}_w([n]) \times \Sigma^k \to \Sigma^w$
   This is a restricted alternative to the write function whose parameters are the positions S of symbols available for writing as well as the difference of the original information x and the new information x' but without reading the w code entries. The outcome is the difference of the available old and the new codeword symbols. Thus, for two functions $\Delta_1 : \Sigma^k \times \Sigma^k \to \Sigma^k$ and $\Delta_2 : \Sigma^w \times \Sigma^w \to \Sigma^w$ and w given positions $\nu_1, \ldots, \nu_w \in [n]$ from y, we can describe the write function g above by the differential write function as

$$(y'_{\nu_1}, \ldots, y'_{\nu_w}) \;=\; \Delta_2((y_{\nu_1}, \ldots, y_{\nu_w}), \delta(S, \Delta_1(x, x'))).$$

   while the original write function needs to read at positions $\rho_1, \ldots, \rho_w \in [n]$ and produces the same result by the following.

$$(y'_{\nu_1}, \ldots, y'_{\nu_w}) = g(\{\rho_1, \ldots, \rho_r\}, (y_{\rho_1}, \ldots, y_{\rho_r}), x', \{\nu_1, \ldots, \nu_w\})$$

All RW codes presented here have such differential write functions where Δ_1, Δ_2 denote the bit-wise XOR-operations. The goal is that e.g. a controller in a storage device i can, by itself, update its kept block y_i by simply adding (XOR) the received difference γ of y_i and y_i', i.e. $y_i' = y_i + \gamma$, if, for example, the device is blocked between reading the old and writing the modified block.

For a tuple $y = (y_1, \ldots, y_n)$ and a subset $S \in \mathbf{P}_\ell([n])$, let $\textsc{Choose}(S, y)$ be the tuple $(y_{i_1}, y_{i_2}, \ldots, y_{i_\ell})$ where $i_1, \ldots, i_\ell$ are the ordered elements of S. Furthermore, for an ℓ-tuple d, let $\textsc{Subst}(S, y, d)$ be the tuple where according to S each indexed element $y_{i_1}, y_{i_2}, \ldots, y_{i_\ell}$ of y is replaced by the element taken from d such that $\textsc{Choose}(S, \textsc{Subst}(S, y, d)) = d$ and all other elements in y remain unchanged in the outcome.
Now, for $S' \in \mathbf{P}_r([n])$, define the read operation

$$\mathbf{Read}(S', y) := f(S', \textsc{Choose}(S', y))$$

and for $S \in \mathbf{P}_w([n])$ and $x' \in \Sigma^k$, the write operation

$$\mathbf{Write}(S, S', y, x') := \textsc{Subst}(S, y, g(S', \mathbf{Read}(S', y), x', S)).$$

Since any Read-Write code needs to start at some initial state, we define the set of possible codewords Y as the *transitive closure* of the function $y \mapsto \mathbf{Write}(S, S', y, x')$ starting with $y = y_0$ and allowing all values S, S', x. Then, an RW code is correct if the following statements are satisfied.

1. *Correctness of the initial state:*

$$\forall S \in \mathbf{P}_r([n]) \ : \ \mathbf{Read}(S, y_0) = x_0 \ .$$

2. *Consistency of read operation*:

$$\forall S, S' \in \mathbf{P}_r([n]) \ \forall y \in Y \ : \ \mathbf{Read}(S, y) = \mathbf{Read}(S', y) \ .$$

3. *Correctness of write operation*:

$$\forall S \in \mathbf{P}_w([n]), \forall S' \in \mathbf{P}_r([n]), \forall y \in Y, \forall x \in \Sigma^k : \mathbf{Read}(S', \mathbf{Write}(S, S', y, x)) = x \ .$$

4 Lower Bounds

The example of a $(2, 3, 3, 4)_2$-RW code in the previous section stores two symbols of information in a four symbol code (c.f. Table 1). Unfortunately, this storage overhead of a factor two is unavoidable, as the following theorem shows (moreover, this implies that e.g. no $(3, 3, 3, 4)_b$-RWC exists).

Theorem 1. *For $r + w < k + n$ or $r, w < k$ and any base b, there does not exist any $(k, r, w, n)_b$-RWC.*

Proof: Consider a write operation and a subsequent read operation where the index set W of the write operation ($|W| = w$) and the index set R of the read operation ($|R| = r$) have an intersection: $W \cap R = S$ with $|S| = r + w - n$. Then, there are b^k possible change vectors with symbols in S that need to be encoded by the write operation since this is the only base of information for the subsequent read operation. This holds because all further $R \setminus S$ code symbols remain unchanged. Now, assume that $|S| < k$. Then, at most b^{k-1} possible changes can be encoded, and therefore, the read operation will produce faulty outputs for some write operations. Thus, $r + w - n \geq k$ and the claim follows.

If $r < k$, only b^r different messages can be distinguished while b^k different messages exist. Then, from the pigeonhole principle, it follows that such a code does not exist. For the case $w < k$, this is analogous. □

Thus, in the best case $(k, r, w, n)_b$-RW codes have parameters $r + w = k + n$. We call such RWC codes *perfect*. Unfortunately, such perfect codes do not always exist as the following lemma shows.

Lemma 1. *There is no* $(1, 2, 2, 3)_2$*-RWC.*

Proof: Consider a read operation on the code bits y_1, y_2 and a write operation on y_2, y_3. Then, y_2 is the only intersecting bit which must be inverted in case of an information bit flip. The same holds for bit y_3 when considering a read operation on y_1, y_3 and a write operation on y_2, y_3. Thus, together, both y_2 and y_3 have to be inverted if the information bit x_1 flips. Now, consider a sequence of three write operations on bits $(1, 2), (2, 3), (1, 3)$ each inverting the information bit x_1. After these operations, all code bits have been inverted twice bringing it back to the original state. In contrast, the information bit has been inverted thrice and is thus inverted. Therefore, all read operations lead to wrong results. □

However, if we allow a larger symbol alphabet, we can find an RW code.

Lemma 2. *There exists a* $(1, 2, 2, 3)_3$*-RWC.*

Proof: See Table 2 for an example. The correctness is straight-forward. □

Clearly, concerning operational complexity, $b = 2$ (i.e. $\mathbb{F}_2$) is the best choice for codes applied in SANs because XOR-based I/O operations can often efficiently be realized in hardware. However, as common RAID 4/5 schemes as well as parity-based Reed-Solomon codes correspond to an $(n, n, n + 1, n + 1)_2$-RW code, $n \geq 1$, the following lemma shows that there is no parity-based placement scheme offering better update properties.

Lemma 3. *For* $n \geq 1$*, there is no* $(n, n, n, n + 1)_2$*-RW code.*

Proof: The proof follows directly from Theorem 1. □

5 Encoding and Decoding

We show that perfect RW codes always exist if the symbol alphabet is large enough, and as being closely related to Reed-Solomon codes, RW codes can also

Contents	Code			Line
x	y_1	y_2	y_3	v
0	0	0	0	0
0	1	1	1	1
0	2	2	2	2
1	0	1	2	0
1	1	2	0	1
1	2	0	1	2
2	0	2	1	0
2	1	0	2	1
2	2	1	0	2

Table 2. A $(1, 2, 2, 3)_3$-Read-Write code for an information word x and codeword y consisting of symbols y_1, y_2, y_3. For every information, there are three possible codewords, and if only any two of them three are available for reading and writing, the system can perform read and write operations (see Figure 2 in the next section for the encoding

be constructed by matrix operations over finite fields. More formally, for given information tuples $x = (x_1, \ldots, x_k) \in \Sigma^k$ whose underlying alphabet Σ is a sufficiently large finite field $\mathbb{F}_q$ (and thus, x is a k-dimensional vector in the vector space $\mathbb{F}_q^k$) and additional parameters $k \leq r, w \leq n$, we examine special subcodes of larger Reed-Solomon codes that have dimension r and length n and in which for each codeword $y = (y_1, \ldots, y_n) \in \Sigma^n$ there exists a codeword for each other message with distance of at most w.

For what follows, we consider the information vector $x = (x_1, \ldots, x_k) \in \mathbb{F}_q^k$, the corresponding codeword $y = (y_1, \ldots y_n) \in \mathbb{F}_q^n$, and for any modification in x, let $\delta = \Delta x$ be the information change vector. Moreover, let $v = (v_1, \ldots, v_\ell)$ denote the vector of internal slack variables with $\ell = n - w = r - k$ and which carry no particular information. Then, the aforementioned operations are realized by the following linear mapping using an appropriate $n \times r$ generator matrix M with $M_{i,j} \in \mathbb{F}_q$; the sub-matrix $(M_{i,j})_{i \in [n], j \in \{k+1, \ldots, r\}}$ is called the *variable matrix*. In particular, an RW code relies on the following *matrix approach*:

$$\begin{pmatrix} M_{1,1} & M_{1,2} & \cdots & M_{1,r} \\ M_{2,1} & M_{2,2} & \cdots & M_{2,r} \\ \vdots & \vdots & & \vdots \\ M_{n,1} & M_{n,2} & \cdots & M_{n,r} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_k \\ v_1 \\ \vdots \\ v_l \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Operations

– **Initialization**:
We start with an arbitrary given information vector $x_0 = (x_1, \ldots, x_k)$, for which the variables $(v_1, \ldots, v_\ell)$ can be set to arbitrary values (if one wants to benefit from the security features of this coding system (see section 6), these slack variables must be chosen uniformly at random). Then, compute the codeword $y_0 = (y_1, \ldots, y_n)$ using the matrix approach above.

– **Read**: Given r code entries from y, compute x
  We rearrange the rows of M and the rows of y such that the first r entries of y are available for reading. Let y' and M' denote these rearranged vector and matrix. The first r rows of M' describe the $r \times r$ matrix M'' that we assume to be invertible. Then, the information vector x (and the variable vector v) is obtained by: $(x \mid v)^T = (M'')^{-1} y$.
– **Differential write**: Given the information change vector δ and w code entries from y, compute the difference vector γ for the w code entries. Recall that y is updated by γ without first reading the information of y at the w code positions.
  The new information vector x' is given by $x'_i = x_i + \delta$. This notation allows to change the vector x' without reading its entries. Clearly, only the choices $w < r$ make sense. Now, due to the matrix approach, given the new k-dimensional information vector x', the task is to find another $(r-k)$-dimensional vector ρ with $v' = v + \rho$ such that the new codeword $y' = M\,(x' \mid v')^T = M\,(x + \delta \mid v + \rho)^T$ is a vector of weight at most w. Since we only consider at most w positions of y', we may, without loss of generality, assume that the last $n-w$ positions are zero, so that $M\,(x' \mid v')^T = (y'_w \mid 0)^T$, with y'_w of length w, and the vector $0 = (0, \ldots, 0)^T$ is of length $n-w$. Clearly, we must rearrange the rows of the matrix M due to the vector $(y'_w \mid 0)^T$. After that, we partition M according to the lengths of the sub-vectors involved, and obtain

$$\begin{aligned} \left(M^{\leftarrow\uparrow}\right) x' + \left(M^{\uparrow\rightarrow}\right) v' &= y'_w \\ \left(M^{\leftarrow\downarrow}\right) x' + \left(M^{\downarrow\rightarrow}\right) v' &= 0\ . \end{aligned}$$

  An important precondition of the write operation is the invertibility of the submatrix $M^{\downarrow\rightarrow}$. The code symbol vector is then updated by the w-dimensional vector $\gamma = ((M^{\leftarrow\uparrow}) - (M^{\uparrow\rightarrow})(M^{\downarrow\rightarrow})^{-1}(M^{\leftarrow\downarrow}))\ \delta$, such that the new w codeword y' is derived from the former code symbols at the w given positions by simple addition, that is, $y' = y + \gamma$.

In fact, the $(2,3,3,4)_2$-RWC in Table 1 can be generated by this matrix based approach whose encoding is given in Figure 1 (compare also the $(1,2,2,3)_3$-RWC in Table 2 and Fig. 2).

Definition 1. *An $n \times k$-matrix A over any base b with $n \geq k$ is* row-wise invertible *if each $k \times k$ matrix constructed by combining k distinct rows of A has full rank (and therefore is invertible).*

Theorem 2. *The matrix based RWC is correct and well-defined if the $n \times r$ generator matrix M as well as the $n \times (r-k)$ variable sub-matrix M' is row-wise invertible.*

Proof: Follows from the definition of row-wise invertibility and the description of the operations. To prove the correctness of the coding system we show that after each operation the matrix based mapping is valid. This is straight-forward

$$\begin{pmatrix} 0\,0\,1 \\ 0\,1\,1 \\ 1\,0\,1 \\ 1\,1\,1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ v_1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

Readable code symbols	x_1	x_2
y_1, y_2, y_3	$y_1 + y_3$	$y_1 + y_2$
y_1, y_2, y_4	$y_2 + y_4$	$y_1 + y_2$
y_1, y_3, y_4	$y_1 + y_3$	$y_3 + y_4$
y_2, y_3, y_4	$y_2 + y_4$	$y_3 + y_4$

Write code	$(x'_1, x'_2) = (x_1 + \delta_1, x_2 + \delta_2)$			
	$y'_1 = y_1 +$	$y'_2 = y_2 +$	$y'_3 = y_3 +$	$y'_4 = y_4 +$
$1, 2, 3$	$\delta_1 + \delta_2$	δ_1	δ_2	0
$1, 2, 4$	δ_1	$\delta_1 + \delta_2$	0	δ_2
$1, 3, 4$	δ_2	0	$\delta_1 + \delta_2$	δ_1
$2, 3, 4$	0	δ_2	δ_1	$\delta_1 + \delta_2$

Fig. 1. A $(2, 3, 3, 4)_2$-Read-Write-Code over the alphabet $\mathbb{F}_2 = \{0, 1\}$ modulo 2.

$$\begin{pmatrix} 0\,1 \\ 1\,1 \\ 2\,1 \end{pmatrix} \begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Readable code symbols	x
y_1, y_2	$2y_1 + y_2$
y_1, y_3	$y_1 + 2y_3$
y_2, y_3	$2y_2 + y_3$

Writable symbols	$x' = x + \delta$		
	$y'_1 =$	$y'_2 =$	$y'_3 =$
y_1, y_2	$\delta + y_1$	$2\delta + y_2$	y_3
y_1, y_3	$2\delta + y_1$	y_2	$\delta + y_3$
y_2, y_3	y_1	$\delta + y_2$	$2\delta + y_3$

Fig. 2. A $(1, 2, 2, 3)_3$-Read-Write-Code over the field $\mathbb{F}_3 = \{0, 1, 2\}$ modulo 3.

for the initialization and read operations. It remains to prove the correctness of the write operation.

Again, consider the additive vector $(\rho_1, \ldots, \rho_\ell)$ denoting the change of the variable vector v and the vector $(\gamma_1, \ldots, \gamma_w)$. With this and the information change vector δ, we obtain $x' = x + \delta$, $v' = v + \rho$ and $y' = y + \gamma$. The correctness of the write operation then follows by combining:

$$M \begin{pmatrix} x' \\ v' \end{pmatrix} = M \begin{pmatrix} x + \delta \\ v + \rho \end{pmatrix} = M \begin{pmatrix} x \\ v \end{pmatrix} + M \begin{pmatrix} \delta \\ \rho \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_w \\ y_{w+1} \\ \vdots \\ y_n \end{pmatrix} + \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_w \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

This equation is equivalent to the following.

$$(M^{\leftarrow\uparrow})\delta + (M^{\uparrow\rightarrow})\rho = \gamma$$
$$(M^{\downarrow\rightarrow})\rho + (M^{\leftarrow\downarrow})\delta = \mathbf{0}\ .$$

Since δ is given, the variable vector ρ can be computed as

$$\rho = \left(M^{\downarrow\rightarrow}\right)^{-1}\left(-M^{\leftarrow\downarrow}\right)\delta\ ,$$

and γ by the last upper equation. If ρ is known, then the product $M \cdot (\delta \mid \rho)^T$ (reduced to the first w rows) gives the difference vector γ which provides the new code entries of y' by $y' = y + \gamma$. □

Theorem 3. *For any $k \leq r, w \leq n$ with $r+w = k+n$ there exists an $(k, r, w, n)_b$-RWC for an appropriate base b. Furthermore, this coding system can be computed in polynomial time.*

Proof: Follows from the following lemma and the fact that we use standard Gaussian elimination for recovery. □

Lemma 4. *For each $n \geq k$ and basis $b \geq 2^{\lceil \log_2 n+1 \rceil}$, there is a row-wise-invertible $n \times k$-matrix over the finite field $\mathbb{F}_b$. Furthermore, each submatrix is also row-wise invertible.*

Proof:

Define an $n \times k$ Vandermonde like matrix V for non-zero distinct elements $(c_1, \ldots, c_n) \in \mathbb{F}_{[2^{\lceil \log_2 n+1 \rceil}]}$.

$$V = \begin{pmatrix} c_1^1 & c_1^2 & \ldots & c_1^k \\ c_2^1 & c_2^2 & \ldots & c_2^k \\ c_3^1 & c_3^2 & \ldots & c_3^k \\ \vdots & & \ddots & \vdots \\ c_n^1 & c_n^2 & \ldots & c_n^k \end{pmatrix}$$

Then, erase any $n - k$ rows resulting in an $k \times k$ matrix V'. This submatrix is also a Vandermonde-matrix. Since all Vandermonde-matrices are invertible, the lemma follows. □

6 Security and Redundancy

Depending on the usage of additional slack variables, in this section, we show that Read-Write codes furthermore offer useful properties concerning data availability and security. Consider, for instance, the very extreme scenario of a combination of hard disks of n portable (laptop) computers in an office. If then a $(k, r, w, n)_b$-RWC is used for encoding for at most n laptops, it is sufficient if at least $\max\{r, w\}$ computers are accessible at the office at any time for data access

and changes. If merely r computers are connected, at least the read operations can be performed. Now, what happens if computer hard disks are broken or information on some hard disks has changed ? Then, the inherent redundancy of the $(k, r, w, k)_b$-RWC allows to point out the number of wrong data and repair it (to some extent).

A different problem occurs if computers are stolen by some adversary to achieve knowledge about company data. The good news is that, for every matrix based RWC, it holds that one can give away any $n - w$ hard disks without revealing any information to the adversary. If the slack variables are chosen uniformly at random from Σ, the attacker will receive hard disks with perfect random sequences, absolutely useless without the other hard disks. As a surplus, this redundantizes the need for complex encryption algorithms.

Theorem 4. *Every $(k, r, w, n)_b$-RWC system can detect and repair ℓ faulty code symbols if $\frac{n!(r+\ell)!}{(n-\ell)!r!} < \frac{1}{2}$. Additionally, it can reconstruct $n - r$ missing code symbols.*

Proof: If $n - r$ code symbols are missing, then by the definition of a RWC system the complete information can be recovered from any r code symbols. Furthermore, if then ℓ out of these r code symbols are faulty, we simply test any combination of the $\binom{n}{r}$ combinations of r code symbols and take a majority vote over the information vector. In this vote, at least $\binom{n-\ell}{r}$ produce the correct result. This results in a majority if $\binom{n-\ell}{r} > \frac{1}{2}\binom{n}{r}$ which, by transformation, is equivalent to $\frac{n!(r+\ell)!}{(n-\ell)!r!} < \frac{1}{2}$. □

If the coded symbols are stored on distinct storage devices, with an (k, r, w, n)-RWC the loss of at most $n - \max\{r, w\}$ device can be tolerated. For instance, if these storage devices were stolen, then the following theorem shows that the thief cannot reveal any information whatsoever from the encoded information: the attacker sees only a completely random sequence vector.

Theorem 5. *Every matrix based $(k, r, w, n)_b$-RWC with $k + n = r + w$ can be used such that every choice of $n-w$ coded symbols does not reveal any information about the original information vector.*

Proof: Choose random vectors $v_1, \ldots, v_\ell$ for the initialization. Then, there is an isomorphism between these slack variables and the stolen coded symbols leading b^ℓ possibilities for the stolen coded symbol to be changed. If more symbols are added, this starts to reveal some information. □

7 Adaptive Read-Write Codes

In a SAN, adding and removing hard disks are the most delicate maneuvers, and provided that the size of the underlying symbol alphabet is chosen appropriately, we show in the following that perfect RW codes exist which allow to seamlessly continue all operations without forcing the system to be in some intermediate and, more importantly, invalid state. For instance, assume 10 disk in a SAN using

an $(8, 9, 9, 10)$-RW code. Then, for better space utilization, the system administrator wants to switch to a $(4, 7, 7, 10)$-RW code. In a usual encoding, all disks have to be available for such a switch. In this section, we show that there exist some special RW codes, called *Chameleon codes*, that allow to switch while only 9 disks are accessible for read and write. If the 10th disk returns after computing the re-encoding of all data on the 9 disks, it can immediately participate in the new $(4, 7, 7, 10)$-RW code. Moreover, if the 10th disk is permanently lost, it can be reconstructed from the new $(4, 7, 7, 10)$-RW code. In particular, a *Chameleon code* is a set of RW-codes $(k, r, w, n)_b$ with fixed alphabet, and, unlike the initial codes, has a switch function. If the code is switched, all parameters k, r, w, n can be subject to change. Regarding the codeword y, not all of the code symbols have to be read or changed.

Theorem 6. *For a sufficiently large constant M, there is an (M, b)-Chameleon-RWC with $b \geq 2^{\lceil \log_2 M+1 \rceil}$. In this system, it is possible to switch at any time from a $(k, r, w, n)_b$-RWC to any $(k', r', w', n')_b$-RWC provided that $n, n' \leq M$ and $k' + n' = r' + w'$ only by reading any set of r encoded symbols and changing any set of w' encoded symbols.*

Proof: First, we select a base $b \geq 2^{\lceil \log_2 M+1 \rceil}$ and take the Vandermonde Matrix based approach as shown in Section 5. We change the main equation to the following.

$$\begin{pmatrix} c_1^1 & c_1^2 & \cdots & c_1^M \\ c_2^1 & c_2^2 & \cdots & c_2^M \\ \vdots & \vdots & & \vdots \\ c_M^1 & c_M^2 & \cdots & c_M^M \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_k \\ v_1 \\ \vdots \\ v_{r-k} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \\ z_1 \\ \vdots \\ z_{M-n} \end{pmatrix}$$

Again, $x_1, \ldots, x_k$ are the content symbols, $v_1, \cdots, v_{r-k}$ are the slack variables and $y_1, \ldots y_n$ are the code symbols. The variables $z_1, \ldots, z_{M-n}$ can be ignored for the beginning; they are neither contents, slack nor code symbols and can be generated from the content and slack symbols at any time. The initial vector as well as the read and write function are chosen as in the matrix based approach. Then, the switch operation, that is, switching from a (k, r, w, n)-RWC to a (k', r', w', n')-RWC, works as follows.

First, we read r code symbols at given positions and decode the vectors x and v according to the matrix based approach. Then, we adapt the size of the former code to the new code size. If $n' > n$, we compute the corresponding variables z_i from x and v. If $n' < n$, we rename $n - n'$ code variables to z-variables, and thus, reduce the code size. If $r' > r$, the content/slack-variable vector $(x \,|\, v)^T$ is extended by $(r' - r)$ 0-entries. We assume that new contents are written during the switch-operation (especially, if $k \neq k'$). For this, let $v'_1, \ldots, v'_{r'-k'}$ be the new set of slack variables. Furthermore, we suppose at most w' code symbols (positions) available for writing.

We start by erasing the rows $n'+1,\ldots,M$ in y and in the Vandermonde matrix since they are of no interest for this operation. Then, like in Section 5, we rearrange the residual matrix and the residual code vector such that the first w positions are the writable variables. We additionally rearrange the columns of the Vandermonde matrix and the contents/slack vector such that the new slack variables are on the rightmost columns, respectively lowermost lines. This results in the generator matrix M' (c.f. Section 5), and the original vector x is rearranged up to the lowest $r'-k'$ entries (possibly containing a mixture of old contents, old slack variables, and 0-entries). Let x' be the vector of the new contents (adequately rearranged), and let v' be the new slack vector with $r'-k'$ entries. If $r' \geq r$, x has k' entries, and otherwise, x (x') has $r-r'$ additional entries resulting from former slack or content variables that must to be set to 0.

We first consider the case $r' \geq r$. The number of entries in x is k'. Then, we can perform an RWC write operation changing w' code symbols. Let $\ell' = r'-k' = n'-w'$ and partition M' like in Section 5. That is, let $M^{\leftarrow\uparrow}$ be a $w' \times k'$-sub-matrix of M', $M^{\uparrow\rightarrow}$ a $w' \times k'$-sub-matrix, $M^{\leftarrow\downarrow}$ an $\ell' \times n'$-sub-matrix and $M^{\downarrow\rightarrow}$ an invertible $\ell' \times \ell'$-sub-matrix of M'. Again, according to the matrix based approach, the new (rearranged) code vector y' is obtained by

$$y' = y + \left[(M^{\leftarrow\uparrow}) - (M^{\uparrow\rightarrow})(M^{\downarrow\rightarrow})^{-1}(M^{\leftarrow\downarrow})\right] \cdot (x'-x) \qquad (1)$$

using the old (rearranged) writable symbol vector y. The proof of correctness is analogous to Section 5.

Now, consider the case $r' < r$. Then, the number of entries in x and x' is $\tilde{k} = k' + r - r'$. Again, let x' be the new (adequately rearranged) vector containing the $\tilde{k}$ new symbols, and v' is the new slack variable vector with $r'-k'$ entries. Note that $x'-x$ can be computed at this stage. We now perform a slightly adapted matrix based RWC write operation that changes w' code symbols. Clearly, compared to the previous case, that matrix consists of additional $r-r'$ columns but what does not cause any problem since we only have to adapt the sub-matrices. Furthermore, let $\ell' = r'-k' = n'-w'$ and $\tilde{w} = w + r - r'$. Then, let $M^{\leftarrow\uparrow}$ be a $\tilde{w} \times \tilde{k}$-sub-matrix of M', $M^{\uparrow\rightarrow}$ a $\tilde{w} \times \ell'$-sub-matrix, $M^{\leftarrow\downarrow}$ an $\ell' \times \tilde{k}$-sub-matrix and $M^{\downarrow\rightarrow}$ an invertible $\ell' \times \ell'$-sub-matrix of M'. As usual, y denotes the old and y' the new writeable symbols. Then, applying the defined matrices, the new vector y' is obtained as given with Equation 1, and again, the proof is analogous to the proof given in Section 5. □

8 Conclusions

The Read-Write codes, presented here, provide linear block codes that, in contrast to commonly applied strategies, such as parity schemes or RS codes, feature advanced possibilities to update any codeword, and to adjust the redundancy and thus, the fault-tolerance capability of the code. In general, RW codes seem to be well-designed to any setting in which I/O operations are very costly, that feature high frequencies of write operations, and that are of dynamic behavior, like modern storage area networks. Further results and applications of RW codes can be found in [12].

References

1. M. Adler, Y. Bartal, J. W. Byers, M. Luby, and D. Raz. A modular analysis of network transmission protocols. In *Israel Symposium on Theory of Computing Systems*, pages 54–62, 1997.
2. M. K. Aguilera, R. Janakiraman, and L. Xu. Reliable and secure distributed storage using erasure codes.
3. M. Blaum, J. Brady, F. Bruck, and H. van Tilborg. Array codes. In *Handbook of Coding Theory*, volume 2, chapter 22. V.S. Pless and W.C. Huffman, 1999.
4. M. Blaum, J. Brady, J. Bruck, and J. Menon. Evenodd: an optimal scheme for tolerating double disk failures in raid architectures. In *ISCA '94: Proceedings of the 21st annual international symposium on Computer architecture*, pages 245–254, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
5. M. Blaum, J. Brady, J. Bruck, J. Menon, and A. Vardy. The EVENODD code and its generalization: An efficient scheme for tolerating multiple disk failures in RAID architectures. In H. Jin, T. Cortes, and R. Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, chapter 14, pages 187–208. IEEE Computer Society Press and Wiley, New York, NY, 2001.
6. W. A. Burkhard and J. Menon. Disk array storage system reliability. In *Symposium on Fault-Tolerant Computing*, pages 432–441, 1993.
7. J. Byers, M. Luby, and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas in Communications, 20(8), Oct*, 2002.
8. N. S. F.J. Mac Williams. *The Theory of Error Correcting Codes*. North-Holland Mathematical Library, 1977.
9. C. Huffmann and V. Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, Cambridge, UK, 2003.
10. A. J.Tate, R.Kanth. Introduction to Storage Area Networks. Technical report, IBM, May 2005.
11. J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.
12. M. Mense. *On Fault-Tolerant Data Placement in Storage Networks*. PhD thesis, University of Paderborn, January 2009.
13. D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD)*, pages 109–116, June 1988.
14. J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software, Practice and Experience*, 27(9):995–1012, 1997.
15. S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage, 2001.
16. L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2):24–36, Apr. 1997.