Erasure Codes for Reading and Writing (Technical Report tr-ri-06-274)

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 MDS codes using a variation of Reed-Solomon (RS) codes, but offer enhanced possibilities to update any codeword, and to adjust the redundancy and thus, the fault-tolerance capability of the code. In particular, given a message x of length k, a block code y of length n and arbitrarily chosen parameters $k \leq r, w \leq n$, our RW codes provide a linear (n, k, d)-code such that (a) the minimum distance $d = n - r + 1$ for any two codewords, and (b) any two codewords are within a distance of at most w from each other. The parameter $r \geq k$ determines the rate of the code and thus, the minimum fraction of encoded blocks needed to recover x, and $w \leq n$ determines the maximum number of blocks to update y if x is modified. This is different from RS codes which always need to rewrite y completely if x changes. In addition, for a subset of the RW codes, all mentioned parameters are adjustable even online such that the code becomes adaptive to changing demands. RW codes are very attractive for e.g. storage networks in which, on one hand, I/O accesses are very costly, and on the other hand, redundancy is crucial. With an RW code, this trade-off can then be better adjusted than with RS codes. We state a tight lower bound for all RWC and show under which conditions of the symbol alphabet size RW codes do exist. Furthermore, we point out some useful properties regarding safety and security of the data encoded.

1 Introduction

Erasure resilient coding have indisputable importance concerning data availability in many application areas (see e.g. [1, 6, 19, 17]). In storage networks, encoding is, on one hand, utilized for efficiency reason, and on the other hand, to be safe from data loss. A typical example are RAID-*arrays* [14, 16] and modern *storage area networks* (SAN) [12]. A SAN is a dedicated network into which companies started to consolidate their storage capacity, in the last couple of years, to face the dramatic growth of enterprise data storage capacity. To provide the efficient operation of a SAN, two major requirements must at least be accomplished: high-level *serviceability*, i.e. always serve given I/O requests quick and efficiently, and high *data availability*, i.e. to feature redundancy mechanisms to be safe from disk failures whose probability to occur grows with the number of connected disks (see [14, 7]). Since access to hard disks is comparably slow, the data is distributed in blocks evenly among the storage devices to exploit access parallelism, and advanced data availability is usually achieved by introducing redundancy into the system. Both can be obtained by an efficient encoding of data using erasure (resilient) block codes. Typical examples applied are e.g. RAID, EVENODD or MDS codes to which RS codes belong, too (see e.g. [14, 4, 16, 9]).

An erasure (resilient) block encoding maps of a word x of k symbols drawn from an alphabet Σ into a codeword y of $n > k$ symbols from the same alphabet. Often, the set $C = \{c_1, \ldots, c_N\}$ of all codewords is referred to as the code. The ratio n/k is the *stretch factor* denoting the storage consumption of the code. To satisfy prallel access requirements in a SAN, the codewords are partitioned into fix sized blocks and evenly distributed among the connected devices. For this, parity-based schemes, like RAID or EVENODD, have become very popular because they have low storage consumption and base on simple but efficient XOR operations. Furthermore, parity and all other mentioned block codes are optimal codes, i.e. they only require any k blocks from y to recover x. Generally, for a code, two parameters are basically important. The first is the ratio k/n, called the *rate* of the code, which indicates the share of information of the original message each encoded symbol maintains. The second paramter is the *distance* $d(C)$ that is defined as the minimum Hamming distance between two codewords $c_i = (c_{i1}, \ldots, c_{in})$, $c_j = (c_{j1}, \ldots, c_{jn})$ in C while the *Hamming distance* is defined as $d(c_i, c_j) = |\{1 \leq l \leq n | c_{il} \neq c_{jl}\}|$. The goal now is to find a code that has both, high distance and a rate close to 1. Unfortunately, there is a tradeoff between these two objectives. A rate close to 1 implies less storage consumption and a high share of information in each encoded symbol wich has great advantages for the data recovery. On the other hand, clearly, a rate close to 1 implies less redundancy and thus, small relative distance which in turn means bad fault-tolerance. Therefore, the goal is to maximize the minimum distance between any two

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).

codewords. In an MDS code (Maximum Distance Separable) $d(C)$ is maximized with $d(C) = n - k + 1$ (see e.g. [9, 11]).

Regarding fault-tolerance parity-based codes can only tolerate a very small number of failures, since they have high rate ($k/(k+1)$ with RAID encoding and $k/(k+2)$ with EVENODD) and thus, low redundancy. The same holds for many variants of MDS codes, like MDS array codes [3] or *X-codes* [2] which are $(n, n-2)$ MDS array codes. Hamming codes have good rate but distance of at most 3 and Reed-Muller codes have high distance but bad rate (c.f. [11]). Thus, in the recent years, RS codes have been utilized efficiently in distributed storage systems [18, 13] and disk arrays [5, 16] since they combine a good rate of $(n-k)/k$ and good distance $d(C) = n-k+1$ because they are also MDS codes. As a drawback, RS codes suffer from high computational costs (see e.g. [15]), thus, near-optimal codes (e.g. [7, 8, 10, 16, 17]) have been introduced which, on one hand, require $(1+\epsilon)n$ encoding symbols to recover the original message, where $\epsilon > 0$ is a parameter, but on the other hand, provide increased computation. Some of them have been utilized for distributed storage applications (see e.g. [7, 8, 10, 16, 17]). However, since I/O accesses in storage networks are comparably costly, optimal codes have become most attractive in e.g. peer-to-peer networks or SANs where applications have high read frequency. On the other hand, if the access behavior changes to increased write frequency, all the employed codes, like parity schemes or RS codes, suffer from an undesired update overhead because if x is modified, all blocks of y must be rewritten.

In this paper, we overcome this problem by the Read-Write-Coding System (RWC) – a very flexible class of linear block codes that are closely related to the conception of RS codes, but offer enhanced possibilities to adjust the redundancy and thus, the fault-tolerance capability of the code. In particular, given a message x of length k, a block code y of length n and arbitrarily chosen parameters $k \leq r, w \leq n$, our Read-Write codes (RW codes) provide a linear (n, k, d)-code such that (a) the minimum distance $d(C) = n-r+1$ for any two codewords, and (b) any two codewords are within a distance of at most w from each other. The parameters r, w are fixed but can be arbitrarily chosen for any RW code. The parameter $r \geq k$ determines the rate of the code, and thus, the minimum fraction of encoded blocks needed to recover x, and $w \leq n$ determines the maximum number of blocks to update y if x is modified. While the first condition is a standard one in coding theory, the second condition has received less attention. RW codes provide further advantages when employed in a SAN because additional information can be exploited at encoding or decoding time. For example, failed or blocked disks can be identified in advance and since only any $w \leq n$ blocks of the codeword need to be modified, the placement of updated blocks can be controlled more flexible. Consider for instance a codeword y distributed among n disk which must be updated to y'. Consider further a subset E, $|E| \leq n - w$, of failed or blocked disks at modification time of y. Although E is known, when using an RW code y' can be updated and is still consistent with the encoding. If then a user wants to read y' after a while and meanwhile, the set E of failed disks has changed to E', $|E'| \leq n - w$, the user can simply select any other w of the $n - |E'|$ remaining disks.

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 $y_1 = x_1, y_2 = x_2, y_3 = x_3$ and $y_4 = x_1 + x_2 + x_3$, where addition denotes the XOR-operation and the y_i, $1 \leq i \leq 4$, are stored separately on disjoint 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 one disks 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 bits y_1, y_2, y_3, y_4. We now encode $k = 2$ 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 further, only any of such $w = 3$ code bits $y_{i'}, y_{j'}, y_{k'}$ need to be changed to encode a completely new information. E.g. start with code

$(0,1,1,?)$. According to the 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. So, we choose line 0 for information $(0,1)$ and get code $(0,1,0,1)$. This example code shows that such *Read-Write codes* exist. 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 b, and a parameter r for recovering the message 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. We present a general scheme to produce any $(k,r,w,n)_b$-RW code if $k+n \leq r+w$, 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, we investigate the question for which choices of (k,r,w,n) a coding system exists over the binary alphabet, i.e. $b=2$, and discuss how RW codes can be combined. This paper is written more operational rather than conceptual since our RW codes base upon algebraic principles similar to RS codes.

2 The Operational Model

Read-Write-Codes encode information words into code words. The information is given by a k-tuple over a finite alphabet Σ. The code is an n-tuple over the same alphabet Σ, $k<n$. Let $b=|\Sigma|$ and $\mathbf{P}(M)$ denotes the power set of a set M. Throughout this paper, we define $[n] := \{1,2,\ldots,n\}$ and let $\mathbf{P}_\ell(M) := \{S \in \mathbf{P}(M) \mid |S| = \ell\}$.

A $(k,r,w,n)_b$ Read-Write-Codings-System (RWC) consists of the following elements.

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 code Y_0.
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 code word with known positions of the symbols. The first parameter shows the positions of the symbols in the code, the second parameter gives the corresponding code symbols. The outcome is the decoded information.

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

3b. **Differential write function** $\delta : \mathbf{P}_w([n]) \times \Sigma^k \to \Sigma^w$
   This is a restricted alternative to the write function which has as parameters the positions S of symbols available for write and the difference of the original and new information, but without reading the w code entries. The result is the difference of the available old and the new code word symbols. This means for two functions $\Delta_1 : \Sigma^k \times \Sigma^k \to \Sigma^k$ and $\Delta_2 : \Sigma^w \times \Sigma^w \to \Sigma^w$ that the write functions g can be described by the differential write function via $Y' = \Delta_2(Y, \delta(S, \Delta_1(X, X')))$. All RW-codes, presented here, have such differential write functions where Δ_1, Δ_2 are the 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. $y_i' = y_i + \gamma$. This makes sense, since a device could be 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 $S \in \mathbf{P}_w([n])$, $X' \in \Sigma^k$ the write operation

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

Define the set of possible codes $\mathcal{C}$ 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.

An RWC 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*: $\forall S, S' \in \mathbf{P}_r([n])$
   $\forall Y \in \mathcal{C}$: $\mathbf{Read}(S, Y) = \mathbf{Read}(S', Y)$.
3. *Correctness of write operation*:
   $\forall S \in \mathbf{P}_w([n]), \forall S' \in \mathbf{P}_r([n]), \forall Y \in \mathcal{C}$,
   $\forall X \in \Sigma^n : \mathbf{Read}(S', \mathbf{Write}(S, S', Y, X)) = X$.

3 Lower Bounds

The example of a $(2,3,3,4)_2$-RW code stores two symbols of information in a four symbol code. This storage overhead of a factor two is unavoidable, as the following theorem shows that e.g. no $(3,3,3,4)_b$-RWC exist.

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 with a subsequent read operation where the index set W of the write operation ($|W|=w$) and the index set of the read operation R with $|R|=r$ have an intersection: $W\cap R=S$ with $|S|=r+w-n$. There are b^k possible change vectors that need to be encoded by the write operation into this intersection symbols, since this is the only base of information for the read operation. The reason is that all $R\setminus S$ code words 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 $r<k$ then only b^r different messages can be distinguished while b^k different messages exist. From the pigeonhole principle it follows that such a code does not exist. For the case $w<k$, this is analogous. □

So, 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 the read operation on y_1,y_2 and the write operation on y_2,y_3. Then, y_2 is the only intersecting bit and it has to be inverted if the information vector changes. Now the same holds for the read operation on y_1,y_3, the write operation on y_2,y_3 and bit y_3. So, y_2 and y_3 have to be inverted if the information vector changes. 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. The information bit has been inverted thrice and is thus inverted. So, all read operation lead to wrong results. □

Yet, if we allow a larger symbol alphabet we can provide an RWC, as the following lemma shows.

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

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

Notice, since the definition of XOR-based parity-RAID (RAID 4/5) corresponds to a $(k,k,k+1,k+1)_2$-RWC, $k\geq 1$, the following lemma shows that there is no RAID-system with improved access properties.

Lemma 3. *For $k\geq 1$, there is no $(k,k,k,k+1)_2$-RWC.*

Proof: The proof follows directly from theorem 1. □

4 The Matrix-Approach

We show that perfect RW codes always exist if the symbol alphabet is large enough. Since RW codes realize a variation of Reed-Solomon codes, they can also be constructed by matrix operations over finite fields. Briefly speaking, we examine codes of dimension r and length n in which every codeword has weight w. Let $x=x_1,\ldots,x_k\in F[b]$ be the information vector and let $y=y_1,\ldots y_n\in F[b]$. The matrix based approach uses some internal slack variables $v=v_1,\ldots,v_l$ for $l=n-w=r-k$ carrying no particular information, and an appropriate $n\times r$ generator matrix M with $M_{i,j}\in F[b]$. The sub-matrix $(M_{i,j})_{i\in[n],j\in\{k+1,\ldots,r\}}$ is called the variable matrix. The code relies on the following equation.

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

Operations

– **Initialization**.
Starting with any information vector $x_1,\ldots,x_k$ the variables $v_1,\ldots,v_\ell$ can be set to arbitrary values. If one want to benefit from the security features of this coding system (see section 5), these variables must be chosen uniformly at random. Now, compute the code $y=y_1,\ldots,y_n$ by using Equation (1).

– **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''. We now use the property that M'' is invertible and the information vector x (and the variable vector v) is now given 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, according to Equation (1), given the new k-dimensional information vector x', the taks is to find another $(r-k)$-dimensional vector ρ with $v' = v + \rho$ such that the new codeword $y' = M\left(x' \mid v'\right)^T = M\left(x + \delta \mid v + \rho\right)^T$ 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 $M\left(x' \mid v'\right)^T = \left(y'_w \mid 0\right)^T$, with y'_1 of length w, and 0 of length $n - w$. Clearly, we must rearrange the rows of the matrix M due to the vector $\left(y'_w \mid 0\right)^T$. After that, we partition M according to the lengths of the sub-vectors involved, and obtain

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

.
A precondition of the write operation is the invertibility of $M^{\downarrow\rightarrow}$. The code symbol vector is now 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, as shown in figure 1 in the Appendix (compare also in the Appendix $(1,2,2,3)_3$-RWC in Table 2 and Fig. 2).

Definition 1. *An $n \times k$-matrix A over the 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 is row-wise invertible and 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 Equation 1 is valid. This is straight-forward 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 now follows by combining:

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

This equation is equivalent to the following.

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

Since δ is given, 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 $n \geq k$ there exists a basis $b \geq 2^{\lceil \log_2 n+1 \rceil}$ with a row-wise $n \times k$-matrix over the finite field $F[b]$. Furthermore, each submatrix is also row-wise invertible.*

Proof:

Define an $n \times k$ Vandermonde like matrix V for non-zero distinct elements $c_1, \ldots, c_n \in F[2^{\lceil \log_2 m+1 \rceil}]$. 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 this proves the claim. □

5 Security and Redundancy

In this section, we show some additional features of our RW codes concerning data security and highly dynamic storage networks. Consider the very extreme scenario of a combination of hard disks of n portable (laptop) computers in a storage network. Using an $(k, r, w, n)_b$-RWC for at most n laptops, it is sufficient if at least $\max\{r, w\}$ computers are at the office to access and change data. If only r computers are connected, then 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 any $(k, r, w, k)_b$-RWC allows to point out the number of wrong data and repair it (to some extent).

A different problem occurs if computers are stolen to achieve knowledge about company data. The good news is that in every matrix based RWC one can give away any $n - w$ hard disks without revealing any information to an adversary. The attacker will receive hard disks with perfect random sequences, absolutely useless without the other hard disks. As a surplus, this redundatizes the need for komplex storage encryption algorithms.

Redundancy

Theorem 4. *Every $(k, r, w, n)_b$-RWC system can detect and repair ℓ faulty code symbols if $\frac{n!(r+\ell)!}{(n-\ell)!r!} < \frac{1}{2}$. 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. If ℓ code symbols are faulty, then we 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 $\binom{n-\ell}{r} \geq \frac{1}{2}\binom{n}{r}$. This inequality is equivalent to $\frac{n!(r+\ell)!}{(n-\ell)!r!} < \frac{1}{2}$. □

Security

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 are 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. □

6 Adaptive RW Codes

In a storage area network (SAN), adding and removing hard disks are the most delicate maneuvers. Using RW codes allows to seamlessly continue all operations. 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. If the 10th disk is permanently lost, then 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. *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 with $n, n' \leq M$ 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. 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 by the content and slack symbols at any time. The initial vector, read and write function are chosen as in the matrix based approach. For the switch operation, we switch from a (k, r, w, n)-RWC to a (k', r', w', n')-RWC as follows.

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 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 $r' > r$ then the content/slack-variable vector $(x|v)^T$ is extended by $(r' - r)$ 0-entries. We assume new contents to be written during the switch-operation (especially if $k \neq k'$). Let $v'_1, \ldots, v'_{r'-k'}$ be the new set of slack variables. Furthermore, w' code symbols are only available for writing.

First, we erase the rows $n' + 1, \ldots, M$ in y and int the Vandermonde matrix, since they are of no interest for this operation. Like in section (4), we then rearrange the residual matrix and the residual code vector such that the first w positions are the writable variables. Furthermore, we 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 matrix $\tilde{M}$. The original vector x is then 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 variable vector with $r' - k'$ entries. If $r' \geq r$, x has k' entries. If $r' < r$, 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 a matrix based RWC write operation changing w' code symbols. Let $\ell' = r' - k' = n' - w'$. For this, partition M' like in section (4). 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 $\ell' \times \ell'$-sub-matrix of M'. Again, the new (rearranged) code vector y' is obtained by $y' = y + \left[(M^{\leftarrow\uparrow}) - (M^{\uparrow\rightarrow})(M^{\downarrow\rightarrow})^{-1}(M^{\leftarrow\downarrow})\right] \cdot (x' - x)$ by using the old (rearranged) writable symbol vector y. The proof of correctness is analogous to section (4).

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, the matrix has $r - r'$ columns more than in the previous case. This is no problem since we only have to adapt the sub-matrices. Let $\ell' = r' - k' = n' - w'$ and let $\tilde{w} = w + r - r'$. Now 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 $\ell' \times \ell'$-sub-matrix of M'. Again, y denote the old and y' the new writeable symbols. The new resulting vector y' is then obtained by $y' = y + \left[(M^{\leftarrow\uparrow}) - (M^{\uparrow\rightarrow})(M^{\downarrow\rightarrow})^{-1}(M^{\leftarrow\downarrow})\right] \cdot (x' - x)$.

Again, the proof is analogous to the proof for the matrix based approached. □

7 Boolean Read-Write-Codes

Since computation speed of RW codes strongly depend on the finite field used (c.f. RS codes), the most interesting case for the choice of the alphabet is the binary case $\Sigma = \{0, 1\}$. We have already shown there are no Boolean $(1, 2, 2, 3)$-

RW codes, and also for the matrix based RW codes, the Boolean basis poses a severe restriction. The reason is that only for some dimensions Boolean matrices can be row-wise invertible.

Lemma 5. *For each n there is one $n \times 1$ row-wise invertible Boolean matrix. For $k \geq 2$, there exist $n \times k$ row-wise invertible Boolean matrices if and only if $n = k$ or $n = k + 1$.*

Proof: The first claim is trivial. For the second, note there is a $(k + 1) \times n$ row-wise invertible Boolean matrix, e.g. $M = (M_{i,j})_{i \in [k+1], j \in [n]}$ such that

$$M_{i,j} = \begin{cases} 1 & : \quad i = n \vee i = j \\ 0 & : \quad else \end{cases}$$

Now, we prove that there are no $(k + 2) \times k$ row-wise invertible matrices. We show that given a $k \times k$ full rank Boolean matrix M, there is always a unique vector leading to a $(k + 1) \times n$ row-wise Boolean matrix.

Consider M and remove row i. Then, there are 2^{k-1} possibilities to add a row receiving a matrix with full rank. Such a row is described by the vector set $M(x_1, \ldots, x_k)^T$ where $x_i = 1$ and the rest is chosen arbitrarily. The only vector that can be added to each of these combinations is $M(1, \ldots, 1)^T$. After adding this vector, there is no other vector which is linearly independent from the other vectors. □

This lemma has severe implications on the matrix based method for Boolean bases, as we show in the following.

Theorem 7. *For $r + w = k + n$ there are Boolean $n \times k$ generator matrices V for $(k, r, w, n)_2$-RWC-Matrix-Coding for only these cases:*

1. $(1, 1, n, n)_2$, $(1, n, 1, n)_2$, $(k, k, k, k)_2$
2. *For $k \geq 2$:* $(k, k + 1, k + 1, k + 2)_2$
3. *For $k \geq 1$:* $(k, k, k + 1, k + 1)_2$
4. *For $k \geq 1$:* $(k, k + 1, k, k + 1)_2$

Proof: Follows by combining the prerequisites of the matrix based RWC method with Lemma 5. □

If only one information bit needs to be encoded, then, the following lemma holds.

Lemma 6. *Every $(1, r, w, r + w - 1)_2$-RW code is a matrix based RW code.*

Proof: Consider a write operation on the set W with $|W| = w$ and a read operation on the index set $R \subseteq [k]$ with $|R| = r$ such that $|R \cap W| = 1$. Let i be the index of this element in the intersection. Now, if the data entry $x \in \{0, 1\}$ changes to x', then this bit must change as well. Otherwise the result of the read operation would be the same as before which is incorrect. The operation on this intersected code bit y_i can be described by $y_i' \equiv y_i + x' + x \pmod 2$. Note that for any index $i \in W$ there is a set R with $|R| = r$ and $|R \cap W| = 1$. So, the above equivalence is valid for all bits leading to the matrix representation of RW codes. □

Notice, there is no technique beyond matrix based RW codes that produces perfect Boolean RW codes.

Lemma 7. *There is no $(2, r, w, n)_2$-RW code for $r + w = 2 + n$ and $w \geq 4$.*

Proof: Consider any 4 bits of the code word y. The index set is denoted by F. Now, we partition the residual $n-4$ code bits into two disjoint index sets R with $|R| = r-2$ and W with $|W| = w-4$. W.l.o.g. consider the set $F = \{1, 2, 3, 4\}$ describing the the code bits y_1, y_2, y_3, y_4. Furthermore, consider write operations on the index sets $W_{i,j} = W \cup F$ for all distinct $i, j \in F$ and read operations on the index set $R_{i,j} = R \cup \{i, j\}$, again for all $i, j \in W$. Now, what is the number of bits that need to be changed, if the information x_1, x_2 changes to x_1', x_2' (there are three posibilities)? Let p_i be a predicate which is only true if y_i must be changed, i.e. inverted. If we consider the read operation on $R_{i,j}$ then all bits in R remain the same. The only way the write operation induces a change is that at least one of the code bits y_i, y_j is changed. Considering all such read operations we have to fulfill the following term:

$$\bigwedge_{i,j \in [4], i \neq j} (p_i \vee p_j) = \left[(p_1 \wedge p_2 \wedge p_3) \vee (p_1 \wedge p_2 \wedge p_4) \vee (p_1 \wedge p_3 \wedge p_4) \vee (p_2 \wedge p_3 \wedge p_4) \right].$$

All but (at most) one bit have to be inverted. Hence, there are five possibilities to encode all three possible changes to the information vector x_1, x_2. Now, each read operation can only observe two out of these four bits. We start with $R_{1,2}$. Then the following cases occur: (A) y_1 and y_2 are inverted, (B) only y_1 is inverted, (C) only y_2 is inverted. A, B, and C can be mapped to all three possible changes of x_1, x_2 to x_1', x_2'. Yet, there is a conflict with the read operation $R_{3,4}$ in the situations B and C which are indistinguishable for the read operation because all bits y_3 and y_4 must be inverted in both situations. Then, the read operation of $R_{3,4}$ cannot distinguish the different change in the information vector. □

This lemma can be generalized to the following theorem.

Theorem 8. *There is no $(k, r, w, n)_2$-RW code for $r + w = k + n$ and $w \geq k + 2$.*

Proof: The proof is analogous to the proof of lemma (7), but we now have to consider $k + 2$ bits of y, $F = [k + 2]$ describing the code bits $y_1, y_2, \ldots, y_{k+2}$, and the $n - k$ residual code bits are partitioned into the sets R and W with $|R| = r - 2$ and $|W| = w - k - 2$. The considered read and write operations on the index sets $W_{i,j}$ and $R_{i,j}$ are then defined as in lemma (7). Again, we are interested in the number of bits that need to be changed, if the information $x_{i_1}, \ldots, x_{i_k}$ changes to $x_{i_1}, \ldots, x_{i_{k+2}}$ (there are $2^k - 1$ posibilities). Let the predicate p_i also be defined as before. If we consider the read operation on $R \cup \{i_1, \ldots, i_k\}$ then all bits in R remained the same. So, the only way, the write operation induces a change is that at least one of the code bits $y_{i_1}, \ldots, y_{i_k}$ are changed (the read operation will read only two bits from this intersection). Considering all such read operations we have to fulfill the following term:

$$\bigwedge_{S \in [k+2]:|S|=k} \bigvee_{i \in S} p_i = \bigvee_{S \in [k+2]:|S| \geq 3} \bigwedge_{i \in S} p_i \, .$$

Thus, at least 3 bits must be inverted in every write operation, and there are $2^{k+2} - \sum_{i=0}^{2} \binom{k+2}{i}$ possibilities to encode all $2^k - 1$ possible changes to the vector $x_1, \ldots, x_k$. However, each of the read operations accesses only k bits (in the intersection) needing $2^k - 1$ possible outcomes, and if we combine two of the read operations, this implies for the whole set that they need at least $2^{k+2} - 7$ possible outcomes. Consider the intersection of read operations that have only $k - 2$ positions in common. If these bits are the same, then each of the residual pairs of two bits must differ, leaving 9 possibilities. In the other case we have $2^{k-2} - 1$ possibilities for the common bit vector and 16 possibilities for the 'private' bit pairs. So, there are $\sum_{i=0}^{2} \binom{k+2}{i} - 7 > 1$ codes missing, for all $k \geq 1$. □

8 General RW-Codes

Since only few perfect Boolean RW codes exist, we consider more general RW codes with $r + w > n + k$. For this, we use a modified matrix based approach and choose all matrix entries randomly. Then, we prove that with positive probability the read and write operations work. We call the following approach a random matrix $(k, \ell, n)_b$-RW code, where the information vector consists of k symbols, ℓ used slack variables, and an n-symbol code is generated. Again, $|\Sigma| = b$. Let M be a (uniformly) random $(k + \ell) \times n$-matrix over Σ. We use the matrix based representation $M(x \mid v)^T = y$. The read and write operation can be derived as follows:

Read Given the $r \geq \ell + k$ readable positions in the code vector, we derive the matrix M' by choosing all rows of M corresponding to the rows available for reading. In M', we choose $\ell + k$ linear independent row vectors yielding matrix M''. If such many linear independent rows do not exist, then the read operation fails. Corresponding to the rows, we reduce the code vector, yielding y' and compute the content vector and the slack variable vector by $(M'')^{-1} y'$.

Write Consider the variable sub-matrix $M^{\rightarrow}$ of n of the ℓ rightmost columns. Now, there are $n - w$ non-writable positions corresponding to rows of the code vector y and variable sub-matrix $M^{\rightarrow}$. If these $n - w$ rows of $M^{\rightarrow}$ are not linear independent, then the write operation fails. Otherwise, we add some other $\ell - n + w$ linear independent rows of $M^{\rightarrow}$ (Again, if these do not exist, the operation fails). These rows now correspond to non-writable positions. Using the complement of this rows as writable positions we can directly apply the matrix based RW code write operation.

We now investigate the success probability of read and write.

Lemma 8. *A $(k + \ell) \times k$ base $b \geq 2$ random matrix has a $k \times k$-sub-matrix (by choosing k of the rows) which is invertible with probability larger than $1 - b^{-\ell+2}$.*

Proof: The probability that a random $(k + \ell) \times k$ matrix over $F[b]$, $b \geq 2$, does not have rank k is the probability that all the $k + \ell$ rows of the matrix lie in some $(n - 1)$-dimensional subspace of $F^k[b]$. Enumerating these subspaces by their dual, we see that there are $(bk - 1) / (b - 1)$ such subspaces, and for each of such subspaces the probability that all the rows of the matrix are in this subspace is $b^{-(k+\ell)}$. By the union bound, the probability that all the rows are in the same subspace is upper bounded by

$$b^{-(k+\ell)} \frac{b^n - 1}{b - 1} \leq \frac{b^{-\ell}}{b - 1} \leq b^{-\ell}.$$

Thus, the probability that the matrix has full rank is at least $1 - b^{-\ell}$. □

Hence, if only few combinations of read and write index sets occur, then the overhead for general codes is small.

Theorem 9. *Consider a general RWC with read index sets $\mathcal{R} = \{R_1, \ldots, R_{|\mathcal{R}|}\}$, $|R_i| \geq r$, and write index sets $\mathcal{W} = \{W_1, \ldots, W_{|\mathcal{W}|}\}$, $|W_i| \geq w$. Then, there is a restricted $(k, r, w, n)_b$ RWC for any $r > k + \log_b |\mathcal{R}| + 2$ and $w + r > n + k + \log_b |\mathcal{W}| + 2$.*

Proof: The probability of having a valid code for reading using r rows is at least $1 - b^{-(r-k)+2}$. Summing up the error probabilities of all possible read operations, we end up with an error probability less than $|\mathcal{R}|b^{-(r-k)+2} < 1$ if $r > k + \log_b |\mathcal{R}| + 3$. For the write operation, we consider an $\ell \times (n - w)$ sub-matrix for $\ell = r - k$. Here, we need a positive probability to find $n - w \leq \ell$ independent rows. The other failure case can be omitted. Hence, with a success probability higher than $1 - b^{n-w-\ell+2}$ we succeed. Thus, for $n - w - r + k + 2 > \log_b |\mathcal{W}|$ the summed error probability is smaller than 1. Combining both error probabilities gives an overall error probability of less than 1. So, there exists a matrix allowing these restricted read and write index sets. □

Yet, the overall number of possible read and write index sets is quite high. The following theorem shows that random Boolean matrices perform quite well for most of the read and write index sets.

Theorem 10. *For any base b, a random matrix based $(\ell, k, n)_b$-RW code successfully performs a read operation for a random choice of $k + \ell + q$ code symbols with probability $1 - b^{-q+2}$, and successfully performs a write operation for a random choice of $n - \ell + q$ writable code symbols with probability $1 - b^{-q+2}$.*

Proof: Follows directly from Lemma 8 and from the definition of the Read and Write operation of random matrix based $(\ell, k, n)_b$-RW-Codes. □

9 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.

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, 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.
5. W. A. Burkhard and J. Menon. Disk array storage system reliability. In *Symposium on Fault-Tolerant Computing*, pages 432–441, 1993.
6. 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.
7. Y. M. Chee, C. J. Colbourn, and A. C. H. Ling. Asymptotically optimal erasure-resilient codes for large disk arrays. *Discrete Appl. Math.*, 102(1-2):3–36, 2000.
8. J. A. Cooley, J. L. Mineweaser, L. D. Servi, and E. T. Tsung. Software-based erasure codes for scalable distributed storage. In *IEEE Symposium on Mass Storage Systems*, pages 157–164, 2003.
9. N. S. F.J. Mac Williams. *The Theory of Error Correcting Codes*. North-Holland Mathematical Library, 1977.
10. A. C. H. Xia. Robustore: Robust performance for distributed storage systems. Technical Report CS2005-0838, University of California, San Diego, October 2005.
11. C. Huffmann and V. Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, Cambridge, UK, 2003.
12. A. J.Tate, R.Kanth. Introduction to Storage Area Networks. Technical report, IBM, May 2005.
13. 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.
14. 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.
15. W. Peterson. *Error Correcting Codes*. The MIT Press, Wiley and Sons, 1961.
16. 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.
17. J. S. Plank and M. G. Thomason. On the practical use of ldpc erasure codes for distributed storage applications. Technical Report CS-03-510, University of Tennessee, September 2003.
18. S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage, 2001.
19. L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2):24–36, Apr. 1997.

Appendix

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

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 $F[2] = \{0, 1\}$ modulo 2.

Contents x	Code y_1	y_2	y_3	Line 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 contents x and code y_1, y_2, y_3.

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

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 $F[3] = \{0, 1, 2\}$ modulo 3.