Applied Mathematics & Information Sciences **5**(2) (2011), 148-170
– An International Journal

An approach to RAID-6 based on cyclic groups

Robert Jackson[1], Dmitriy Rumynin[2] and Oleg V. Zaboronski[3]

[1]Arithmatica, ltd., Haseley Business Centre, Warwick, CV35 7LS, UK

Email Address: robert.jackson@arithmatica.com

[2]Department of Mathematics, University of Warwick, Coventry, CV4 7AL, UK

Email Address: D.Rumynin@warwick.ac.uk

[3]Department of Mathematics, University of Warwick, Coventry, CV4 7AL, UK

Email Address: O.V.Zaboronski@warwick.ac.uk

Received June 22, 200x; Revised March 21, 200x

As the size of data storing arrays of disks grows, it becomes vital to protect data against double disk failures. A popular method of protection is via the Reed-Solomon (RS) code with two parity check symbols. In the present paper we construct alternative examples of linear block codes protecting against two erasures. Our construction is based on an abstract notion of cone. Concrete cones are constructed via matrix representations of cyclic groups of prime order. In particular, this construction produces EVENODD code. Interesting conditions on the prime number arise in our analysis of these codes. At the end, we analyse an assembly implementation of the corresponding system on a general purpose processor and compare its write and recovery speed with the standard DP-RAID system.

Keywords: RAID-6, Artin's conjecture, Mersenne prime, Reed-Solomon Code

2000 Mathematics Subject Classification: 94B60

1 Introduction

A typical storage solution targeting a small-to-medium size enterprise is a networked unit with 12 disk drives with total capacity of around 20 TB [9, 13]. The volume of information accumulated and stored by a typical small-size information technology company amounts to fifty 100-gigabyte drives. The specified mean time between failures (MTBF) for a modern desktop drive is about 500,000 hours [14]. Assuming that such an MTBF is actually achieved and that the drives fail independently, the probability of a disk failure in

Oleg V. Zaboronski was partially supported by Royal Society Industrial Fellowship

the course of a year is $1 - e^{-12/57} \approx 0.2$. Therefore, even a small company can no longer avoid the necessity of protecting its data against disk failures. The use of redundant arrays of independent disks (RAIDs) enables such a protection in a cost efficient manner.

To protect an array of K disks against a single disk failure it is sufficient to add one more disk to the array. For every K bits of user data written on K disks of the array, a parity bit equal to an exclusive OR (XOR) of these bits is written on the $(K+1)$-st disk. Binary content of any disk can be then recovered as a bitwise XOR of contents of remaining K disks. The corresponding system for storing data and distributing parity between disks of the array is referred to as RAID-5 [8]. Today, RAID-5 constitutes the most popular solution for protected storage.

As the amount of data stored by humanity on magnetic media grows, the danger of $multiple$ disk failures within a single array becomes real. Maddock, Hart and Kean argue that for a storage system consisting of one hundred $8+P$ RAID-5 arrays the rate of failures amounts to losing one array every six months [8]. Because of this danger, RAID-5 is currently being replaced with RAID-6, which offers protection against double failure of drives within the array. RAID-6 refers to any technique where two strips of redundant data are added to the strips of user data, in such a way that all the information can be restored if any two strips are lost.

A number of RAID-6 techniques are known [4, 8, 10].A well-known RAID-6 scheme is based on the rate-$255/257$ Reed-Solomon code [1]. In this scheme two extra disks are introduced for up to 255 disks of data and two parity bytes are computed per 255 data bytes. Hardware implementation of RS-based RAID-6 is as simple as operations in $\widetilde{\mathbb{F}} = GF(256)$, which are byte-based. Addition of bytes is just a bitwise XOR. Multiplication of bytes corresponds to multiplication of boolean polynomials modulo an irreducible polynomial. Multiplication can be implemented using XOR-gates, AND-gates and shifts.

Some RAID-6 schemes use only bitwise XOR for the computation of parity bits by exploiting a two-dimensional striping of disks of the array. Examples are a proprietary RAID-DP developed by Network Appliances [7] and EVENODD [2]. Some other RAID-6 methods use a non-trivial striping and employ only XOR operation for parity calculation and reconstruction. Examples include X-code, ZZS-code and Park-code [8, 11, 12].

In all the cases mentioned above, the problem dealt with is inventing an error correcting block code capable of correcting up to two erasures (we assume that it is always known which disks have failed). In the present paper we describe a general approach to the solution of this problem, which allows one to develop an optimal RAID-6 scheme for given technological constraints (e.g. available hardware, the number of disks in the array, the required read and write performance). We also consider an assembly implementation of an exemplary RAID-6 system built using our method and show that it outperforms the Linux kernel implementation of RS-based RAID-6.

The paper is organised as follows. In Section 2 we discuss RAID-6 in the context of

systematic linear block codes and construct simple examples of codes capable of correcting two errors in known positions. In Section 3 we identify an algebraic structure (cone) common to all such codes and use it to construct RAID-6 schemes starting with elements of a cyclic group of a prime order. Section 3.3 is of particular interest to number theorists where we discuss a new condition on the prime numbers arising in the context of RAID-6 schemes. In Section 4 we compare encoding and decoding performances of an assembly implementation of RAID-6 based on Z_{17} with its RS-based counterpart implemented as a part of Linux kernel.

Let us comment on the relation of the presented material to other modern research efforts. Section 2 is rather standard [3]. All original theoretical material of this paper is in Section 3. The notion of a cone is somewhat related to a non-singular difference set of Blaum and Roth [3] but there are essential differences between them. The cone from a cyclic group of prime order as in Lemma 3.4.1 gives EVENODD code [2]. Its extended versions and connections to number theoretic conditions are new.

2 RAID-6 from the viewpoint of linear block codes.

Suppose that information to be written on the array of disks is broken into blocks of length n bits. What is the best rate linear block code, which can protect data against the loss of two blocks?

Altogether, there are 2^{2n} possible pairs of n-bit blocks. In order to distinguish between them, one needs at least $2n$ distinct syndromes. Therefore, any linear block code capable of restoring 2 lost symbols in known locations must have at least $2n$ parity check bits. Suppose the size of the information is Kn bits or K blocks. In the context of RAID, K is the number of information disks to be protected against the failure. Then the code's block size must be at least $(2+K)n$ and the rate is

$$R \leq \frac{K}{K+2}$$

This result is intuitively clear: to protect K information disks against double failure, we need at least 2 parity disks.

In the following subsections we construct explicit examples of linear codes for RAID-6 for small values of n and K. These examples both guide and illustrate our general construction of RAID-6 codes presented in Section 3.

2.1 Redundant array of four independent disks, which protects against the failure of any two disks.

We restrict our attention to *systematic* linear block codes. These are determined by the parity matrix. To preserve a backward compatibility with RAID-5 schemes, we require half

of the parity bits to be the straight XOR of the information bits. Hence the general form of the parity check matrix for $K = 2$ is

$$P = \begin{pmatrix} I_{n\times n} & I_{n\times n} & I_{n\times n} & 0_{n\times n} \\ H & G & 0_{n\times n} & I_{n\times n}, \end{pmatrix} \tag{2.1}$$

where $I_{n\times n}$ and $0_{n\times n}$ are $n \times n$ identity and zero matrix correspondingly; G and H are some $n \times n$ binary matrices. The corresponding parity check equations are

$$d_1 + d_2 + \pi_1 = 0 \qquad \text{and} \qquad H \cdot d_1 + G \cdot d_2 + \pi_2 = 0. \tag{2.2}$$

Here d_1, d_2 are n-bit blocks written on disks 1 and 2, π_1 and π_2 are n-bit parity check blocks written on disks 3 and 4; "$\cdot$" stands for binary matrix multiplication.

Matrices G and H defining the code are constrained by the condition that the system of parity check equations must have a unique solution with respect to any pair of variables. To determine these constraints we need to consider the following particular cases.

(π_1, π_2) *are lost.* The system (2.2) always has a unique solution with respect to lost variables: we can compute parity bits in terms of information bits.

(d_1, π_2) *are lost.* The system (2.2) always has a unique solution with respect to lost variables: compute d_1 in terms of π_1 and d_2 using the first equation of (2.2) as in RAID-5. Then compute π_2 using the second equation.

(d_2, π_2) *are lost.* The system (2.2) always has a unique solution with respect to lost variables: compute d_2 using π_1 and d_1 as in RAID-5. Then compute π_1 using (2.2).

(π_1, d_1) *are lost.* The system (2.2) always has a unique solution with respect to lost variables provided the matrix H is invertible.

(π_1, d_2) *are lost.* The system (2.2) always has a unique solution with respect to lost variables provided the matrix G is invertible.

(d_1, d_2) *are lost.* The system (2.2) always has a unique solution with respect to lost variables provided the matrix $\begin{pmatrix} I_{n\times n} & I_{n\times n} \\ H_{n\times n} & G_{n\times n} \end{pmatrix}$ is invertible.

As it turns out, one can build a parity check matrix satisfying all the non-degeneracy requirements listed above for $n = 2$. The simplest choice is

$$H = I_{2\times 2},\ G = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}. \tag{2.3}$$

Non-degeneracy of the three matrices H, G and (2.1) is evident. For instance[1],

$$\det\begin{pmatrix} I_{n\times n} & I_{n\times n} \\ H_{n\times n} & G_{n\times n} \end{pmatrix} = -1 = 1.$$

[1]The reader is aware that $-1 = 1 \neq 0$ in characteristic 2

We conclude that the linear block code with a 4×8 parity check matrix (2.1, 2.3) gives rise to RAID-6 consisting of four disks. The computation of parity dibits $\pi_1,\ \pi_2$ in the described DP RAID is almost as simple as the computation of regular parity bits: Let $d_1 = (d_{11}, d_{12})$ and $d_2 = (d_{21}, d_{22})$ be the dibits to be written on disks one and two correspondingly. Then

$$\begin{aligned} \pi_{11} &= d_{11} + d_{21}, & \pi_{12} &= d_{12} + d_{22}, \\ \pi_{21} &= d_{11} + d_{22}, & \pi_{22} &= d_{12} + d_{21} + d_{22}. \end{aligned} \quad (2.4)$$

The computations involved in the recovery of lost data are bitwise XOR only. As an illustration, let us write down expressions for lost data bits in terms of parity bits explicitly:

$$\begin{aligned} d_{22} &= \pi_{11} + \pi_{12} + \pi_{21} + \pi_{22}, & d_{12} &= \pi_{11} + \pi_{21} + \pi_{22}, \\ d_{11} &= \pi_{11} + \pi_{12} + \pi_{22}, & d_{21} &= \pi_{12} + \pi_{22}. \end{aligned}$$

It is interesting to note that RAID-6 code described here is equivalent to Network Appliances' horizontal-diagonal parity RAID-DP^{TM} with two data disks [7]. Really, diagonal-horizontal parity system for two info disks is

$$\begin{array}{cccc} A & B & HP & DP1 \\ C & D & HP2 & DP2, \end{array}$$

where strings (A, C) are written on information disk 1, strings (B, D) are written on disk 2, $(HP, HP2)$ is horizontal parity, $(DP1, DP2)$ is diagonal parity. By definition, $HP = A + B$, $HP2 = C + D$, $DP1 = A + D$, $DP2 = B + C + D$, which coincides with parity check equations (2.4).

On the other hand, the code (2.1) is a reduction of the RS code based on $GF(4)$ which we will describe in the next subsection.

2.2 Redundant array of five independent disks, which protects against the failure of any two disks.

The code (2.1) can be extended to a scheme providing double protection of user data written on three disks [3, Example 1.1]. The parity check matrix is

$$P = \begin{pmatrix} I_{2\times 2} & I_{2\times 2} & I_{2\times 2} & I_{2\times 2} & 0_{2\times 2} \\ I_{2\times 2} & G & G^2 & 0_{2\times 2} & I_{2\times 2}, \end{pmatrix} \quad (2.5)$$

where 2×2 matrix G was defined in the previous subsection. The corresponding parity check equations are

$$d_1 + d_2 + d_3 + \pi_1 = 0, \qquad d_1 + G \cdot d_2 + G^2 \cdot d_3 + \pi_2 = 0. \quad (2.6)$$

The solubility of these equations with respect to any pair of variables from the set $\{d_1, d_2, d_3, \pi_1, \pi_2\}$ requires two extra conditions of non-degeneracy in addition to non-degeneracy conditions listed in the previous subsection. Namely, matrices $\begin{pmatrix} I_{2\times 2} & I_{2\times 2} \\ I_{2\times 2} & G^2 \end{pmatrix}$ and $\begin{pmatrix} I_{2\times 2} & I_{2\times 2} \\ G & G^2 \end{pmatrix}$ must be invertible. It is possible to check the invertibility of these matrices via a direct computation. However, in the next section we will construct a generalisation of the above example and find an elegant way of proving non-degeneracy.

The code (2.1) is a reduction of (2.5) corresponding to $d_3 = 0$. Note also that the code (2.5) is equivalent to rate-$3/5$ Reed-Solomon code based on $GF(4)$: a direct check shows that the set of 2×2 matrices $0, 1, G, G^2$ is closed under multiplication and addition and all non-zero matrices are invertible. Thus this set forms a field isomorphic to $GF(4)$. On the other hand, as we established in the previous subsection, the code (2.1) is equivalent to RAID-DP^{TM} with four disks. Therefore, RAID-DP^{TM} with four disks is a particular case of the RS-based RAID-6. It would be interesting to see if RAID-DP^{TM} can be reduced to the RS-based RAID-6 in general.

We are now ready to formulate general properties of linear block codes suitable for RAID-6 and construct a new class of such codes.

3 RAID-6 based on the cyclic group of a prime order.

3.1 RAID-6 and cones of $GL_n(\mathbb{F})$.

In this subsection we will define a general mathematical object underlying all existing algebraic RAID-6 schemes. We recall that $\mathbb{F} = GF(2)$ is the field of two elements and $GL_n(\mathbb{F})$ is the set of $n \times n$ invertible matrices.

Definition 3.1.1. A cone[2] C is a subset of $GL_n(\mathbb{F})$ such that $g + h \in GL_n(\mathbb{F})$ for all $g \neq h \in C$.

This notion is related to *non-singular difference sets* of Blaum and Roth [3]. The cone satisfies the axioms P1 and P2 of Blaum and Roth but the final axiom P3 or P3' is too restrictive for our ends. On the other hand, we consider only binary codes while Blaum and Roth consider codes over any finite field.

A standard example of a cone appears in the context of Galois fields. If we choose a basis of $GF(2^n)$ as a vector space over $\mathbb{F}$ then we can think of $GL_n(\mathbb{F})$ as the group of all $\mathbb{F}$-linear transformations of $GF(2^n)$. Multiplications by non-zero elements of $GF(2^n)$ form a cone. If $\alpha \in GF(2^m)$ is a primitive generator and g is the matrix of multiplication

[2]This terminology is slightly questionable. If one asks $g + h \in C$ then $C \cup \{0\}$ is a convex cone in the usual mathematical sense. Our choice of the term is influenced by this analogy. Non-singular difference set or quasicone or RAID-cone could be more appropriate scientifically but would pay a heavy linguistic toll.

by α then this cone is $\{g^m \mid 0 \leq m \leq 2^n - 2\}$. This cone gives the RS-code with two parity blocks.

The usefulness of cones for RAID-6 is explained by the following

Lemma 3.1.2. *Let $C = \{g_1, g_2, \dots g_K\} \subseteq GL_n(\mathbb{F})$ be a cone of K elements. Then the system of parity equations*

$$d_{K+1} = \sum_{t=1}^{K} d_t \qquad and \qquad d_{K+2} = \sum_{t=1}^{K} g_t d_t \tag{3.1}$$

has a unique solution with respect to any pair of variables $(d_i, d_j) \in \mathbb{F}^n \times \mathbb{F}^n$, $1 \leq i < j \leq K+2$. Here d_i are binary n-dimensional vectors.

Proof. The fact that system (3.1) has a unique solution with respect to (d_{K+1}, d_{K+2}) is obvious.

The system has a unique solution with respect to (d_{K+1}, d_j) for any $j \leq K$: from the second of equations (3.1), $d_j = g_j^{-1}(d_{K+2} + \sum_{t \neq j}^{K} g_t d_t)$, where we used invertibility of $g_j \in GL_n(\mathbb{F})$. With d_j known, d_{K+1} can be computed from the first of equations (3.1).

The system has a unique solution with respect to (d_{K+2}, d_j) for any $j \leq K$: from the first of equations (3.1), $d_j = d_{K+1} + \sum_{t \neq j}^{K} d_t$. With d_j known, d_{K+2} can be computed from the second of equations (3.1).

The system has a unique solution with respect to any pair of variables d_i, d_j for $1 \leq i < j \leq K$: multiplying the first of equations (3.1) with g_i and adding the first and second equations, we get $d_j = (g_i + g_j)^{-1}(g_i d_{K+1} + d_{K+2} + \sum_{t \neq i,j}^{K}(g_t + g_i)d_t)$. Here we used the invertibility of the sum $g_i + g_j$ for any $i \neq j$, which follows from the definition of the cone. With d_j known, d_i can be determined from any of the equations (3.1). **QED**

In the context of RAID-6, d_i for $1 \leq i \leq K$ can be thought of as n-bit strings of user data, d_{K+1}, d_{K+2} - as n-bit parity strings. The lemma proved above ensures that any two strings can be restored from the remaining K strings.

We conclude that any cone can be used to build RAID-6. The following lemma gives some necessary conditions for a cone.

Lemma 3.1.3. *Let $C \subset GL_n(\mathbb{F})$ be a cone.*

(i) For all $g,\ h \in C$ such that $g \neq h$ and for all $x \in GF(2^m)^n$, $gx = hx$ if and only if $x = 0$.

(ii) No two elements of the same cone can share an eigenvector in $\mathbb{F}^n$.

(iii) The cone C can contain no more than one permutation matrix.

Proof. To prove (i), assume that there is $x \neq 0 : gx = hx$. Then $(g + h)x = 0$, which contradicts the fact that $g + h$ is non-degenerate. Therefore, $x = 0$. Let us prove (ii) now. As elements of C are non-degenerate, the only possible eigenvalue in $\mathbb{F}$ is 1, thus for any two elements sharing an eigenvector x, $x = hx = gx$, which again would imply degeneracy of $h + g$ unless $x = 0$. The statement (iii) follows from (ii) if one notices that

any two permutation matrices share an eigenvector whose components are all equal to one. **QED**

The notion of the cone is convenient for restating well understood conditions for a linear block code to be capable of recovering up to two lost blocks. Our main challenge is to find examples of cones with sufficiently many elements, which lead to easily implementable RAID-6 systems. We will now construct a class of cones starting with elements of a cyclic subgroup of $GL_n(\mathbb{F})$ of a prime order.

3.2 RAID-6 based on matrix generators of Z_K.

We start with the following

Theorem 3.2.1. *Let K be an odd number. Let g be an $n \times n$ binary matrix such that $g^K = Id$ and $Id + g^m$ is non-degenerate for each proper*[3] *divisor m of K. Then the elements of cyclic group $Z_K = \{Id, g, g^2, \ldots, g^{K-1}\}$ form a cone.*

The proof of the Theorem 3.2.1 is based on the following two lemmas.

Lemma 3.2.2. *Let g be a binary matrix such that $Id + g$ is non-degenerate and $g^K = Id$, where K is an integer. Then*

$$\sum_{t=0}^{K-1} g^t = 0 \tag{3.2}$$

Proof. Let us multiply the left hand side of (3.2) with $(Id + g)$ and simplify the result using that $h + h = 0$ for any binary matrix:

$$\begin{aligned}(Id+g)\sum_{t=0}^{K-1} g^t &= Id + g + g + g^2 + \ldots + g^{K-1} + g^{K-1} + g^K \\ &= Id + g^K = Id + Id = 0.\end{aligned}$$

As $Id + g$ is non-degenerate, this implies that $\sum_{t=0}^{K-1} g^t = 0$. **QED**

Lemma 3.2.2 is a counterpart of a well-known fact from complex analysis that roots of unity add to zero.

Lemma 3.2.3. *Let g be a binary matrix such that $g^K = Id$ for an odd number K and $Id + g^m$ is non-degenerate for every proper divisor m of K. Then the matrix $g^l + g^t$ is non-degenerate for any $t, l:\ 0 \le t < l < K$.*

Proof. As $g^K = Id$, the matrix g is invertible. To prove the lemma, it is therefore sufficient to check the non-degeneracy of $Id + g^t$ for $0 < t < K$.

The group $Z_K = \{1, g, g^2, \ldots, g^{K-1}\}$ is cyclic. An element $g_t = g^t$ for $0 < t < K$ generates the cyclic subgroup $Z_{K/d}$ where d is the the greatest common divisor of K and

[3] a natural number $m < K$ that divides K

t and the element g^d generates the same subgroup. Since the matrix g^d satisfies all the conditions of Lemma 3.2.2, the sum of all elements of $Z_{K/d}$ is zero. Therefore,

$$0 = \sum_{m=0}^{\frac{K}{d}-1} g_t^m = (Id + g_t) + g_t^2(Id + g_t) + \ldots + g_t^{K-3}(Id + g_t) + g_t^{K-1} = 0. \quad (3.3)$$

The grouping of terms used in (3.3) is possible as K/d is odd. Assume that matrix $1 + g_t$ is degenerate. Then there exists a non-zero binary vector x such that $(1 + g_t)x = 0$. Applying both sides of (3.3) to x we get $g_t^{K-1}x = g^{t(K-1)}x = 0$. This contradicts non-degeneracy of g. Thus the non-degeneracy of $1 + g^t$ is proved for all $0 < t < K$. **QED**

The proof of Theorem 3.2.1. The matrix g described in the statement of the theorem satisfies all requirements of Lemma 3.2.3. The statement of the theorem follows from Definition 3.1.1 of the cone. **QED**

Theorem 3.2.1 allows one to determine whether elements of Z_K belong to the same cone by verifying a single non-degeneracy conditions imposed on the generator.

The following corollary of Theorem 3.2.1 makes an explicit link between the constructed cone and RAID-6:

Corollary 3.2.4. *Let g be an $n \times n$ binary matrix such that $g^K = Id$ for an odd number K and $Id + g^m$ is non-degenerate for every proper divisor m of K. The systematic linear block code defined by the parity check matrix*

$$P = \begin{pmatrix} I_{n\times n} & I_{n\times n} & I_{n\times n} & \ldots & I_{n\times n} & I_{n\times n} & 0 \\ I_{n\times n} & g & g^2 & \ldots & g^{K-1} & 0 & I_{n\times n}, \end{pmatrix}$$

can recover up to 2 n-bit lost blocks in known positions. Equivalently, the system of the parity check equations

$$\begin{aligned} d_1 + d_2 + \ldots + d_K + d_{K+1} &= 0 \\ d_1 + gd_2 + \ldots + g^{K-1}d_K + d_{K+2} &= 0 \end{aligned} \quad (3.4)$$

has a unique solution with respect to any pair of variables (d_i, d_j), $1 \le i < j \le K + 2$.

Proof. It follows from Theorem 3.2.1. that the first K powers of g belong to a cone. The statement of the corollary is an immediate consequence of Lemma 3.1.2 for $g_t = g^{t-1}$, $1 \le t \le K$. **QED.**

As a simple application of Theorem 3.2.1, let us show that the parity check matrix (2.5) does indeed satisfy all non-degeneracy requirements. The matrix $G = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ is non degenerate and has order 3. Also, the matrix $Id + G = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ is non-degenerate. Hence in virtue of Corollary 3.2.4, the parity check matrix (2.5) determines a RAID system consisting of five disks, that protects against the failure of any two disks.

3.3 Extension of Z_K-based cones for certain primes.

We will now show that for certain primes, the cone constructed in the previous subsection can be extended. The existence of such extensions give some curious conditions on a prime number, one of which is new to the best of our knowledge. We start with the following

Lemma 3.3.1. *Let $K > 2$ be a prime number. Then the group ring $R = \mathbb{F}Z_K$ of the cyclic group of order K is isomorphic to $\mathbb{F} \oplus \widehat{\mathbb{F}}^t$ where $\widehat{\mathbb{F}} = GF(2^d)$, d is the smallest positive integer such that $2^d = 1 \mod K$, $t = (K-1)/d$.*

Proof. By the Chinese Remainder Theorem, $R \cong \oplus_{j=0}^{k} \mathbb{F}[X]/(f_j)$ where $X^K - 1 = f_0 \cdot f_1 \cdots f_t$ is the decomposition into irreducible over $\mathbb{F}$ polynomials and $f_0 = X - 1$. Let α be a root of f_j for some $j > 0$. Then $d = \deg f_j$ is the smallest number such that $\alpha \in GF(2^d) \cong \mathbb{F}[X]/(f_j)$. Hence, $\alpha^{2^d-1} = 1$ and d is the smallest with such property. As K is prime, α is a primitive K-th root of unity. Hence, K divides $2^d - 1$ and d is the smallest with such property.

It follows that for $j > 0$ all f_j have degree d and all $\mathbb{F}[X]/(f_j)$ are isomorphic to $GF(2^d)$. **QED**.

One case of particular interest is $t = 1$ which happens when 2 is a primitive $K-1$-th root of unity modulo K. This forces $t = 1$ and $d = K-1$. Such primes in the first hundred are 3, 5, 11, 13, 19, 29, 37, 53, 59, 61, 67, 83 [15]. If a generalised Riemann's hypothesis holds true then there are infinitely many such primes [5].

Lemma 3.3.2. *Let $K > 2$ be a prime number such that 2 is a primitive $K-1$-th root of unity modulo K. Let g be an $n \times n$ binary matrix such that $Id + g$ is non-degenerate and $g^K = Id$. Then the set of $2^{K-1} - 1$ matrices $S = \{g^{a_1} + g^{a_2} + \ldots + g^{a_s} \mid 0 \le a_1 < a_2 \ldots < a_s < K,\ 1 \le s \le \frac{K-1}{2}$ is a cone.$\}$*

Proof. Matrix g defines a ring homomorphism $\phi : R \to M_n(\mathbb{F})$, $\phi(\sum_k \alpha_k X^k) = \sum_k \alpha_k g^k$ from the group ring to a matrix ring. Since $1 + g$ is invertible,

$$1 + g + g^2 + \ldots + g^{K-1} = (1 + g^K)(1 + g)^{-1} = 0$$

and $1 + X + \ldots + X^{K-1}$ lies in the kernel of ϕ. Since $R/(1 + X + \ldots + X^{K-1}) \cong \mathbb{F}[X]/(f_1) \cong \widehat{\mathbb{F}}$, the image $\phi(R)$ is a field and S is a subset of $\phi(R)$. Finally, as $f_1 = 1 + X + \ldots + X^{K-1}$ is the minimal polynomial of g, all elements $g^{a_1} + g^{a_2} + \ldots + g^{a_s}$ listed above are distinct and nonzero and $S = \phi(R) \setminus \{0\}$. **QED**.

Notice that for $K = 3$, the set S consists only of Id and g.

The cone S in Lemma 3.3.2. may be difficult to use in a real system but it contains a very convenient subcone as soon as $K > 3$. This subcone consists of elements g^i and $Id + g^j$. The following theorem gives a condition on the prime p for these elements to form a cone. This condition is new to the best of our knowledge.

Theorem 3.3.3. *The following conditions are equivalent for a prime number* $K > 2$.

(1) For any $n \times n$ *binary matrix* g *such that* $Id + g$ *is non-degenerate and* $g^K = Id$ *the set of* $2K - 1$ *matrices* $S = \{Id, g, g^2, \ldots, g^{K-1}, Id + g, Id + g^2, \ldots, Id + g^{K-1}\}$ *is a cone.*

(2) For no primitive K*-th root of unity* α *in the algebraic closure of* $\mathbb{F}$*, the element* $\alpha + 1$ *is an* K*-th root of unity.*

(3) For any $0 < m < K$ *the polynomials* $X^K + 1$ *and* $X^m + X + 1$ *are relatively prime.*

(4) No primitive K*-th root of unity* α *in the algebraic closure of* $\mathbb{F}$ *satisfies* $\alpha^m + \alpha^l + 1 = 0$ *with* $K > m > l > 0$.

Proof. First, we observe that (1) is equivalent to (4). If (4) fails, there exists an K-th root of unity α such that $\alpha^m + \alpha^l + 1 = 0$. Let $f(X)$ be the minimal polynomial of α. The matrix g of multiplication by the coset of X in $\mathbb{F}[X]/(f)$ fails condition (1) with $g^m + g^l + 1 = 0$.

If (4) holds and g is a matrix as in (1) then the elements of S are all invertible matrices by Theorem 3.2.1. Moreover, it only remains to establish that each matrix $g^m + g^l + Id$, $K > m > l > 0$ is invertible. Suppose that it is not invertible. It must have an eigenvector $v \in \mathbb{F}^n$ with the zero eigenvalue. It follows that $f_v(X)$, the minimal polynomial of g with respect to v, divides both $X^K + 1$ and $X^m + X^l + 1$. Since 1 is not a root of $X^m + X^l + 1$, any root α of $f_v(X)$ in the algebraic closure of $\mathbb{F}$ is a primitive K-th root of unity and satisfies $\alpha^m + \alpha^l + 1 = 0$.

Equivalence of (4) and (3) is clear: $\beta = \alpha^l$ is also a primitive root, hence condition (4) can be rewritten as no root β satisfies $\beta^s + \beta + 1 = 0$ with $K > s > 0$. Thus, $X^K + 1$ and $X^m + X + 1$ do not have common roots in the algebraic closure of $\mathbb{F}$ and must be relatively prime.

Equivalence of (3) and (2) comes from rewriting $\alpha^m + \alpha + 1 = 0$ as $\alpha^m = \alpha + 1$ and observing that α^m is necessarily a primitive K-th root of unity. **QED**.

This theorem allows us to sort out whether any particular prime K is suitable for extending the cone.

Corollary 3.3.4. *A Fermat prime* $K > 3$ *satisfies the conditions of Theorem 3.3.3. A Mersenne prime fails the conditions of Theorem 3.3.3.*

Proof. A Fermat prime is of the form $K = 2^t + 1$. Hence, for a primitive Kth root of unity α

$$(\alpha + 1)^K = (\alpha + 1)^{2^t}(\alpha + 1) = (\alpha^{2^t} + 1)(\alpha + 1) = \alpha^{-1} + \alpha.$$

If this is equal 1, then $\alpha^2 + \alpha + 1 = 0$, forcing $K = 3$. A Mersenne prime is of the form $K = 2^t - 1$. Hence,

$$(\alpha + 1)^K = (\alpha + 1)^{2^t}(\alpha + 1)^{-1} = (\alpha^{2^t} + 1)(\alpha + 1)^{-1} = (\alpha + 1)(\alpha + 1)^{-1} = 1.$$

QED.

In fact, most of the primes appear to satisfy the conditions of Theorem 3.3.3. In the first 500 primes, the only primes that fail are Mersenne and 73. Samir Siksek has found several more primes that fail but are not Mersenne. These are (in the bracket we state the order of 2 in the multiplicative group of $GF(p)$) 73 (9) 178481 (23), 262657 (27), 599479 (33), 616318177 (37), 121369 (39), 164511353 (41), 4432676798593 (49), 3203431780337 (59), 145295143558111 (65), 761838257287 (67), 10052678938039 (69), 9361973132609 (73), 581283643249112959 (77). It would be interesting to know whether there are infinitely many primes failing the conditions of Theorem 3.3.3.

Utilising the cone in Theorem 3.3.3., we start with a matrix generator of the cyclic group of an appropriate prime order K to build a RAID-6 system protecting up to $2K-1$ information disks. The explicit expression for Q-parity is

$$Q = \sum_{t=0}^{K-1} g^t d_t + \sum_{t=K}^{2K-2} (Id + g^{t-K+1}) d_t, \tag{3.5}$$

where $d_0, d_1, \ldots, d_{2K-2}$ are information blocks.

3.4 Specific examples of matrix generators of Z_K and the corresponding RAID-6 systems.

Now we are ready to construct explicit examples of RAID-6 based on the theory of cones developed in the above subsections. The non-extended code, based on the Sylvester matrix, is known as EVENODD code [2].

Lemma 3.4.1. *Let S_K be the $(K-1)\times(K-1)$ Sylvester matrix,*

$$S_K = \begin{pmatrix} 0 & 0 & 0 & \cdot & \cdot & \cdot & 1 \\ 1 & 0 & 0 & 0 & \cdot & \cdot & 1 \\ 0 & 1 & 0 & 0 & 0 & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & 1 & 0 & 1 \\ 0 & 0 & 0 & \cdot & \cdot & 1 & 1 \end{pmatrix}.$$

Then

(i) S_K has order K.

(ii) Matrix $Id + S_K$ is non-degenerate if K is odd and is degenerate if K is even.

Proof. **(i)** An explicit computation shows, that for any $(K-1)$-dimensional binary vector x and for any $1 \leq t \leq K$,

$$S_K^t \begin{pmatrix} x_{K-1} \\ x_{K-2} \\ x_{K-3} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_1 \end{pmatrix} = \begin{pmatrix} x_t \\ x_t \\ x_t \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_t \end{pmatrix} + \begin{pmatrix} x_{t-1} \\ x_{t-2} \\ \cdot \\ x_1 \\ 0 \\ x_{K-1} \\ \cdot \\ x_{t+1} \end{pmatrix}. \tag{3.6}$$

In the above formula $x_j \equiv 0$, unless $1 \leq j \leq (K-1)$. Therefore, $S_K^t \neq Id$, for any $1 \leq t \leq K-1$. Setting $t = K$ in the above formula, we get $S_K^K x = x$ for any x, which implies that $S_K^K = Id$. Therefore, the order of the matrix S_K is K.

(ii) The characteristic polynomial of S_K is $f(x) = \sum_{t=0}^{K-1} x^t$. (In order to prove this it is sufficient to notice that the matrix S_K is the companion matrix of the polynomial $f(x)$ [6]. As such, $f(x)$ is both the characteristic and the minimal polynomial of the matrix S_K.) Therefore,

$$f(S_K) = \sum_{t=0}^{K-1} S_K^t = 0.$$

Notice that the matrix S_K is non-degenerate as it has a positive order. If K is odd, we can re-write the characteristic polynomial as

$$f(S_K) = (Id + S_K)(1 + S_K + S_K^3 + \ldots + S_K^{(K-3)}) + S_K^{K-1}$$

Therefore, the degeneracy of $Id + S_K$ will contradict the non-degeneracy of S_K. If K is even, the sum of all rows of $Id + S_K$ is zero, which implies degeneracy. **QED**

Lemma 3.4.1 states that the matrix S_K generates the cyclic group Z_K and that the matrix $Id + S_K$ is non-degenerate for any odd K. Given that K is an odd prime, Corollary 3.2.4 implies that using parity equations (3.4) with $g = S_K$, it is possible to protect K data disks against the failure of any two disks. Furthermore, if $K > 3$ is a Fermat prime or 2 is a primitive root modulo K, $2K-1$ data disks can be protected against double failure thanks to the results of section 3.1.

We will refer to the RAID-6 system based on Sylvester matrix S_K as Z_K-RAID. Let us give several examples of such systems.

(1) Z_3-RAID has been considered in subsections 2.1, 2.2. It can protect up to 3 information disks against double failure. As $K = 3$, protection of 5 information disks using extended Q-parity (3.5) is impossible.

(2) Using Z_{17}-RAID, one can protect up to $K = 17$ disks using Q-parity (3.4) and up to $2K - 1 = 33$ disks using extended Q-parity (3.5).

(3) Using Z_{257}-RAID, one can protect up to $K = 257$ disks using Q-parity (3.4) and up to $2K - 1 = 513$ disks using extended Q-parity (3.5).

It can be seen from (3.6), that the multiplication of data vectors with any power of the Sylvester matrix S_K requires one left and one right shift, one n-bit XOR and one AND only. Thus the operations of updating Q-parity and recovering data within Z_K-RAID does not require any special instructions, such as Galois field look-up tables for logarithms and products. As a result, the implementation of Z_K-RAID can in some cases be more efficient and quick than the implementation of the more conventional Reed-Solomon based RAID-6. In the next section we will demonstrate the advantage of Z_K-RAID using an example of Linux kernel implementation of Z_{17}-RAID system.

4 Linux Kernel Z_K-RAID Implementation

4.1 Syndrome Calculation for the Reed-Solomon RAID-6.

First, let us briefly recall the RAID-6 scheme based on Reed-Solomon code in the Galois field $\widetilde{\mathbb{F}}$, see [1] for more details. Let $D_0, \ldots, D_{K-1}$ be the bytes of data from K information disks. Then the parity bytes P and Q are computed as follows, using [4] $g = \{02\} \in \widetilde{\mathbb{F}}$:

$$P = D_0 + D_1 + \ldots + D_{K-1},\ Q \quad = D_0 + gD_1 + \ldots + g^{K-1}D_{K-1}. \tag{4.1}$$

The multiplication by $g = \{02\}$ can be viewed as the following matrix multiplication.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} x_7 \\ x_0 \\ x_1 \oplus x_7 \\ x_2 \oplus x_7 \\ x_3 \oplus x_7 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \tag{4.2}$$

Given (4.2), parity equations (4.1) become similar to (3.4). Indeed, the element g generates a cyclic group, so a 2-error correcting Reed-Solomon code is a partial case of a cone based RAID. However, Z_K-RAID has several advantages. For instance, using Sylvester matrices one can achieve a simpler implementation of matrix multiplication.

[4] Algebraically, we use the standard representation in electronics: $\widetilde{\mathbb{F}} = GF(2)[x]/I$ where the ideal I is generated by $x^8 + x^4 + x^3 + x^2 + 1$ and $g = x + I$

4.2 Linux Kernel Implementation of Syndrome Calculation

To compute the Q-parity, we rewrite (4.1) as

$$Q = D_0 + g(D_1 + g(\ldots + g(D_{K-2} + gD_{K-1})\ldots))) \tag{4.3}$$

which requires $(K-1)$ multiplications by $g = \{02\}$.

The product y of a single byte x and $g = \{02\}$ can be implemented as follows.

```
uint8_t x, y;   y= (x<<1) ^ ((x & 0x80) ? 0x1d :  0x00);
```

Notice that `(x & 0x80)` picks out x_7 from x, so

```
((x & 0x80) ? 0x1d : 0x00)
```

selects between the two bit patterns 00011101 and 00000000 depending on x_7. Since the carry is discarded from `(x << 1)`,

$$\begin{aligned}\texttt{(x << 1)} = [x_6, x_5, x_4, x_3, x_2, x_1, x_0, 0]&\texttt{((x \& 0x80) ? 0x1d : 0x00)}\\ &= [0, 0, 0, x_7, x_7, x_7, 0, x_7].\end{aligned} \tag{4.4}$$

We can also implement the multiplication as follows.

```
int8_t x, y; y = (x +x)^ (((x < 0)? 0xff :  0x00) & 0x1d);
```

Here we treat the values as signed, rather than unsigned. Whilst this implementation appears more complex than the first (since it uses addition and comparison), it can efficiently be implemented using SIMD instructions on modern processors, such as MMX/SSE/SSE2/AltiVec.

In particular, we will use the following four SSE2 instructions, which store the result in place of the second operand:

```
pxor x, y   : y = x ^ y;   pand x, y   : y = x & y;
paddb x, y  : y = x + y;  pcmpgtb x, y   : y = (y > x) ?
0xff : 0x00;
```

We implement a single multiplication with the following pseudo SSE2 assembler code. We assume that the variables `y` and `c` are initialised as `y = 0` and `c = 0x1d`.

```
pcmpgtb x, y  : y = (x < 0) ? 0xff : 0x00; // (x < 0) ? 0xff
: 0x00
paddb x, x     : x = x + x;            // x + x
pand c, y : y = y & 0x1d;  // ((x < 0) ? 0xff : 0x00) & 0x1d
pxor x, y      : y = x ^ y;   // (x + x) ^ (((x < 0) ? 0xff :
0x00) & 0x1d)
```

The comparison operation overwrites the constant 0 stored in `y`. Therefore, when we implement the complete algorithm we must recreate the constant before each multiplication. We can do it as follows.

```
pxor y, y       : y = y ^ y;                    // y ^ y = 0
```

Besides the five instruction above we need three other instructions to complete the inner loop of the algorithm. They are multiply, fetch a new byte of data D and update the parity variables P and Q:

$$P = D + P, \qquad Q = D + gQ. \tag{4.5}$$

The complete algorithm requires the following eight instructions.

```
pxor y, y       : y = y ^ y;                    // y ^ y = 0
pcmpgtb q, y    : y = (q < 0) ? 0xff : 0x00; // (q < 0) ?
0xff : 0x00
paddb q, q      : q = q + q;                    // q + q
pand c, y       : y = y & 0x1d;       // ((q < 0) ? 0xff :
0x00) & 0x1d
pxor y, q    : q = q ^ y;        // g.q = (q + q) ^  (((q < 0) ?
0xff : 0x00) & 0x1d)
movdqa d[i], d : d = d[i]                       // d[i]
pxor d, q       : q = d ^ q;                    // d[i] ^ p
pxor d, p       : p = d ^ p;                    // d[i] ^ g.q
```

We can gain a further increase in speed by partially unrolling the 'for' loop around the inner loop.

4.3 Reconstruction

We consider a situation that two data disks D_x and D_y have failed. We must reconstruct D_x and D_y from the remaining data disks D_i $(i \neq x, y)$ and the parity disks P and Q, see (4.1). Let us define P_{xy} and Q_{xy} as the syndromes under an assumption that the failed disks were zero:

$$P_{xy} = \sum_{i \neq x,y} D_i, \qquad Q_{xy} = \sum_{i \neq x,y} g^i D_i. \tag{4.6}$$

Rewriting (4.1) in the light of (4.6),

$$D_x + D_y = P + P_{xy}, \qquad g^x D_x + g^y D_y = Q + Q_{xy}. \tag{4.7}$$

Let us define

$$A = (1 + g^{y-x})^{-1}, \qquad B = g^{-x}(1 + g^{y-x})^{-1}. \tag{4.8}$$

Now we eliminate D_x from equations (4.7):

$$\begin{aligned} D_y &= (1 + g^{y-x})^{-1}(P + P_{xy}) + g^{-x}(1 + g^{y-x})^{-1}(Q + Q_{xy}) \\ &= A(P + P_{xy}) + B(Q + Q_{xy}). \end{aligned} \tag{4.9}$$

Finally, D_x is computed from D_y by the back substitution into (4.7):

$$D_x = D_y + (P + P_{xy}). \tag{4.10}$$

4.4 Linux Kernel Implementation of Reconstruction

We compute the following values in $\widetilde{\mathbb{F}}$:

$$\begin{aligned} A &= (1 + g^{y-x})^{-1}, & B &= g^{-x}(1 + g^{y-x})^{-1} = (g^x + g^y)^{-1}, \\ D_y &= A(P + P_{xy}) + B(Q + Q_{xy}), & D_x &= D_y + (P + P_{xy}). \end{aligned} \tag{4.11}$$

It is worth pointing out that for specific x and y, we only need to compute A and B once. The Linux kernel provides the following look-up tables:

`raid6_gfmul[256][256]` : xy `raid6_gfexp[256]` : g^x
`raid6_gfinv[256]` : x^{-1} `raid6_gfexi[256]` : $(1 + g^x)^{-1}$

Using this, we compute A and B as follows:

`A = raid6_gfexi[y-x]` and `B = raid6_gfinv[raid6_gfexp[x] ^ raid6_gfexp[y]]`

To reconstruct D_x and D_y we start by constructing P_{xy} and Q_{xy} using the standard syndrome code. Then we execute the following code.

```
dP = P ^ Pxy;          // P + Pxy
dQ = Q ^ Qxy;          // Q + Qxy
Dy = raid6_gfmul[A][dP] ^ raid6_gfmul[B][dQ];  //
A(P + Pxy) + B(Q + Qxy)
Dx = Dy ^ dP;          // Dy + (P + Pxy)
```

4.5 Z_{17}-RAID Implementation

The multiplication by the Sylvester matrix g looks like

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{15} \end{bmatrix} = B \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{bmatrix} = \begin{bmatrix} x_{15} \\ x_0 \oplus x_{15} \\ x_1 \oplus x_{15} \\ x_2 \oplus x_{15} \\ x_3 \oplus x_{15} \\ x_4 \oplus x_{15} \\ x_5 \oplus x_{15} \\ x_6 \oplus x_{15} \\ x_7 \oplus x_{15} \\ x_8 \oplus x_{15} \\ x_9 \oplus x_{15} \\ x_{10} \oplus x_{15} \\ x_{11} \oplus x_{15} \\ x_{12} \oplus x_{15} \\ x_{13} \oplus x_{15} \\ x_{14} \oplus x_{15} \end{bmatrix}. \tag{4.12}$$

where

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{4.13}$$

We implement the multiplication of a double byte $y = gx$ as follows:

```
int16_t x, y;  y = (x + x) ^ ((x < 0) ? 0xffff : 0x0000);
```

We can implement this in assembler using the following seven instructions.

```
pxor y, y      : y = y ^ y;                      // y ^ y = 0
pcmpgtw q, y   : y = (q < 0) ? 0xffff : 0x0000; // (q < 0) ?
0xffff : 0x0000
paddw q, q     : q = q + q;                      // q + q
pxor y, q      : q = q ^ y;           // g.q = (q + q) ^
                      //  ((q < 0) ? 0xffff : 0x0000)
movdqa d[i], d : d = d[i]                       // d[i]
pxor d, q      : q = d ^ q;                    // d[i] ^ p
pxor d, p      : p = d ^ p;                    // d[i] ^ g.q
```

Below are the results of the Linux kernel RAID-6 algorithm selection programs, aimed to select the fastest implementation of the algorithm. Algorithms using CPU/MMX/SSE/SSE2 instructions with various levels of unrolling are compared. The results were obtained from a 2.8 GHz Intel Pentium 4 (x86).

	DP-RAID	Z_{17}-RAID
int32x1	694 MB/s	766 MB/s
int32x2	939 MB/s	854 MB/s
int32x4	635 MB/s	838 MB/s
int32x8	505 MB/s	604 MB/s
mmxx1	1893 MB/s	2117 MB/s
mmxx2	2025 MB/s	2301 MB/s
sse1x1	1200 MB/s	1284 MB/s
sse1x2	2000 MB/s	2263 MB/s
sse2x1	1850 MB/s	2357 MB/s
sse2x2	2702 MB/s	3160 MB/s

Comparing the above results against the standard Linux kernel results shows an average of 14.5% speed increase and an increase of 16.9% for the fastest sse2x2 implementation. This is consistent with the theoretical increase of 14.3% for seven instructions instead of eight instructions. It is worth mentioning that no look-up tables have been used to implement Z_{17}-RAID.

4.6 Z_K RAID Reconstruction

We need to compute the following matrices and vectors:

$$\begin{aligned} A &= (1+g^{y-x})^{-1}, & B &= g^{-x}(1+g^{y-x})^{-1} = (g^x+g^y)^{-1} \\ D_y &= A(P+P_{xy}) + B(Q+Q_{xy}), & D_x &= D_y + (P+P_{xy}). \end{aligned} \tag{4.14}$$

We rewrite them as follows:

$$\begin{aligned} z &= y-x, & \Delta P &= P+P_{xy}, & \Delta Q &= Q+Q_{xy} \\ D_y &= (1+g^z)^{-1}\Delta P + g^{-x}(1+g^z)^{-1}\Delta Q, & D_x &= D_y + \Delta P. \end{aligned} \tag{4.15}$$

Using the standard identities $g^{-k} = g^{17-k}$ and $(1+g)^{-1} = 1+g^2+\ldots+g^{16}$, we derive new identities:

$$\begin{aligned} (1+g^z)^{-1} &= 1+g^{2z}+g^{4z}+\ldots+g^{16z}, \\ g^{-x}(1+g^z)^{-1} &= g^{17-x}(1+g^z)^{-1} = g^{17-x}(1+g^{2z}+g^{4z}+\ldots+g^{16z}). \end{aligned} \tag{4.16}$$

Consequently, we need to compute

$$\begin{aligned} &(1+g^z)^{-1}\Delta P = (1+g^{2z}+g^{4z}+\ldots+g^{16z})\Delta P = \\ &= 1+g^{2z}(1+g^{2z}(1+g^{2z}(1+g^{2z}(1+g^{2z}(1+g^{2z}(1+g^{2z}(1+g^{2z}\Delta P))))))) \end{aligned} \tag{4.17}$$

and

$$\begin{aligned} g^{-x}(1+g^z)^{-1}\Delta Q = & \quad g^{17-x}(1+g^{2z}+g^{4z}+\ldots+g^{16z})\Delta Q \\ = & \quad g^{17-x}(1+g^{2z}(1+g^{2z}(1+g^{2z}(1+g^{2z} \\ \times & \quad (1+g^{2z}(1+g^{2z}(1+g^{2z}(1+g^{2z}\Delta Q)))))))) \end{aligned} \tag{4.18}$$

Both (4.17) and (4.18) require only one principle operation, multiplication by g^t.

The multiplication of a single block $y = g^t x$ for $1 \le t \le 16$ can be implemented as follows.

```
int16_t x, y;
y = (x << t) ^ (x >> (17 - t)) ^ (((x << (t - 1)) < 0) ?
0xffff : 0x0000);
```

We precompute $m = t-1$ and $n = 17-t$ as they remain constant during reconstruction. This leads to the following assembler implementation.

```
pxor y, y    : y = 0      movdqa x, z  : z = x
psllw m, z   : z = x << (t-1)
pcmpgtw z, y : y = (((x << (t-1)) < 0) ? 0xffff : 0x0000
```

```
paddw z, z   : z = x << t
pxor z, y : y = (((x << (t-1)) < 0) ? 0xffff : 0x0000) ^ (x
<< t)
psrlw n, x   : x = x >> (17-t)  pxor x, y    : y = g ^ t x
```

Below is a table showing benchmark results of complete reconstruction algorithm implemented using SSE2 assembler and the standard Linux kernel look-up table reconstruction implementation, for the cases of double data disk failure, double disk failure of one data disk and the P-parity disk, and double parity disk failure. Note the data represents time taken to complete benchmark, so lower is better.

Failure	DD	DP	PQ
DP-RAID	2917	2771	905
Z_{17}-RAID	2711	1274	809

Comparing the complete reconstruction algorithm implemented using SSE2 assembler against the standard Linux kernel look-up table implementation, shows approximately 7% speed increase for DD failure, 54% speed increase for DP failure and 11% speed increase for PQ failure.

5 Conclusions.

In this paper we have demonstrated that *cones* provide a natural framework for the design of RAID. They provide a flexible approach that can be used to design a system. It is worth further theoretical investigation what other examples of cones can be constructed or what the maximal possible size of a cone is.

We have also demonstrated that cyclic groups give rise to natural and convenient to operate examples of cones. One particular advantage is that Z_K-RAID does not require support of the Galois field operations.

On the practical side, Z_{17}-RAID and Z_{257}-RAID are breakthrough techniques that show at least 10% improvement during simulations compared to DP-RAID.

Acknowledgements

The authors would like to thank Arithmatica, Ltd. for the opportunity to use its research facilities. The authors would also like to thank Robert Maddock and Igor Shparlinski for valuable information on the subject of the paper. Finally, the authors are indebted to Samir Siksek for the interest in the prime number condition that appears in this paper and computation of several primes satisfying it.

References

[1] H. P. Anvin, The mathematics of RAID-6, *online paper*, http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf, Accessed 6 June 2008.

[2] M. Blaum, J. Brady, J. Bruck, J. Menon, EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures, *IEEE Trans. Computers* **44** (1995), 192-202.

[3] M. Blaum, R. Roth, On lowest density MDS codes, *IEEE Trans. Inform. Theory* **45** (1999), 46-59.

[4] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, D. A. Patterson, Coding techniques for handling failures in large disk arrays, *Algorithmica* **12** (1994), 182-208.

[5] C. Hooley, On Artin's Conjecture, *J. Reine Angew. Math.* **226** (1967), 209-220.

[6] R. Lidl and H. Niederreiter, *Finite Fields*, Second Edition, Cambridge University Press, 1997.

[7] C. Lueth, RAID-DP^{TM}: NetApp implementation of RAID double parity for data protection, *online paper*, http://www.netapp.com/library/tr/3298.pdf, Accessed 6 June 2008 .

[8] R. Maddock, N. Hart, T. Kean, Surviving two disk failures. Introducing various 'RAID-6' implementations, *white paper*, Xyratex Ltd., May 2005, http://www.xyratex.com/technology/white-papers.aspx, Accessed 6 June 2008.

[9] D. Reine, IBM Introduces the DS3000 Series for the SMB Lowering Cost, Increasing Storage Capacity, *The Clipper Group Navigator*, TCG2007005, January 2007, http://www.clipper.com/research/TCG2007005.pdf, Accessed 17 March 2010.

[10] K. Srinivasan, C. J. Colbourn, Failed disk recovery in double erasure RAID arrays, *J. Discrete Algorithms* **5** (2007), 115-128.

[11] L. Xu, J. Bruck, X-code: MDS array codes with optimal encoding, *IEEE Trans. Inform. Theory* **45** (1999), 272-276.

[12] G. V. Zaitsev, V. A. Zinovev, N. V. Semakov, Minimum-check-density codes for correcting bytes of errors, erasures, or defects, *Problems Inform. Transmission* **19** (1983), 197-204

[13] NETGEAR Extends Leadership in SMB Storage with Two High-Performance ReadyNAS Solutions for Virtualized Environments, *Netgear press release*, March 2010, http://www.readynas.com/?p=3610, Accessed 17 March 2010.

[14] Next Generation Mobile Hard Disk Drives, *white paper*, Fujitsu Inc., http://www.fujitsu.com/downloads/COMP/fcpa/hdd/sata-mobile-ext-duty_wp.pdf, Accessed 6 June 2008.

[15] Primes with primitive root 2, *The On-Line Encyclopedia of Integer Sequences*, http://www.research.att.com/ njas/sequences/A001122, Accessed 6 June 2008.

Robert Jackson is a Senior Engineer at Imagination Technologies, Ensigma Communication group. His research interests include information theory, cryptography and computer arithmetic, targeting practical and efficient VLSI implementations. In 2008 he earned a Ph.D. in Mathematics from the University of Warwick.

Dmitriy Rumynin earned his MSc in Mathematics from Novosibirsk State University in 1994 and PhD in Mathematical from University of Massachusetts at Amherst in 1998. He is a reader at the Department of Mathematics, University of Warwick, Coventry, UK. His research interests are Algebra, Geometric Representation Theory and Computer Arithmetic.

Oleg Zaboronski was born in 1968 in Moscow. He earned his MSc in Theoretical Physics from Moscow Engineering Physics Institute in 1993 and PhD in Mathematical Physics from University of California at Davis in 1997. He is a reader at the Department of Mathematics, University of Warwick, Coventry, UK. Currently he is research leave with Siglead Inc. (Yokohama, Japan). His research interests are Statistical Physics, Turbulence, Data Detection and Decoding Algorithms.