STAR: An Efficient Coding Scheme for Correcting Triple Storage Node Failures

Cheng Huang, *Member*, *IEEE*, and Lihao Xu, *Senior Member*, *IEEE*

Abstract—Proper data placement schemes based on erasure correcting codes are one of the most important components for a highly available data storage system. For such schemes, low decoding complexity for correcting (or recovering) storage node failures is essential for practical systems. In this paper, we describe a new coding scheme, which we call the STAR code, for correcting triple storage node failures (erasures). The STAR code is an extension of the double-erasure-correcting EVENODD code and a modification of the generalized triple-erasure-correcting EVENODD code. The STAR code is an Maximum Distance Separable (MDS) code and thus is optimal in terms of node failure recovery capability for a given data redundancy. We provide detailed STAR code decoding algorithms for correcting various triple node failures. We show that the decoding complexity of the STAR code is much lower than those of existing comparable codes; thus, the STAR code is practically very meaningful for storage systems that need higher reliability.

Index Terms—Fault tolerance, high availability, error control codes, storage systems.

1 Introduction

In virtually all information systems, it is essential to have a reliable data storage system that supports data availability, persistence, and integrity. Here, we refer to a storage system in the general sense: It can be a disk array, a network of storage nodes in a clustered environment (SAN or NAS), or a wide area large scale P2P network. In fact, many research and development efforts have been made to address various issues of building reliable data storage systems to ensure data survivability, reliability, availability, and integrity, including disk arrays such as RAID [14], clustered systems such as NOW [2] and RAIN [12], distributed file systems such as the Network File System (NFS) [39], HA-NFS [4], xFS [3], AFS [36], Zebra [24], CODA [37], Sprite [29], Scotch [21], and BFS [13], storage systems such as NASD [20], Petal [26], and PASIS [41], and large-scale data distribution and archival networks such as Intermemory [22], OceanStore [25], and Logistical Network [32].

As already indicated by these efforts, proper data redundancy is the key to providing high reliability, availability, and survivability. Evolving from simple data replication or data striping in early clustered data storage systems such as the RAID system [14], people have realized that it is more economical and efficient to use the so-called *threshold schemes* to distribute data over multiple nodes in distributed storage systems [41], [40], [22], [25] than naive (multicopy) replications. The basic idea is to employ n storage nodes to hold k nodes worth of data. The data is broken up into k equal-sized pieces and then expanded to n pieces. The entire n pieces are stored on the n nodes in the system in such a way that the original data can be retrieved from at least m pieces (naturally, $m \geq k$). Such threshold schemes are called (n, k)-threshold schemes.

From the *error control code* point of view, an (n, k)-threshold scheme with equal-sized pieces is equivalent to an (n, k) block code and, especially, most (n, k)-threshold schemes are equivalent to (n, k) *Maximum Distance Separable* (MDS) codes [28], [27]. An (n, k) error control code uses mathematical means to transform a k-symbol data block to an n-symbol codeword block such that any m symbols of the codeword block can recover all of the k symbols of the original data block, where $k \leq m \leq n$. All of the data symbols are of the same size in bits. Obviously, by the simple *pigeonhole* principle, $k \leq m$. When $m = k$, such an (n, k) code is called an MDS code or meets the *Singleton Bound* [27]. Hereafter, we simply use (n, k) to refer to any data distribution scheme using an (n, k) MDS code. Using coding terminology, each share of (n, k) is called a data *symbol*. The process of creating n data symbols from the original data whose size is of k symbols is called *encoding* and the corresponding process of retrieving the original data from at least arbitrary k data symbols stored in the system is called *decoding*.

It is not hard to see that an (n, k) scheme can tolerate up to $(n - k)$ node failures at the same time and thus achieve data reliability or data *survivability* in case the system is under attack where some nodes cannot function normally. The (n, k) scheme can also ensure the *integrity* of the data distributed in the system since an (n, k) code can be used to detect data modifications on up to $(n - k)$ nodes. $r = n - k$ is a parameter that can describe the *reliability degree* of an (n, k) scheme.

While the concept of (n, k) codes has been well understood and suggested in various data storage projects, virtually all practical systems use the Reed-Solomon (RS) code [35] as an MDS code. (The so-called *information dispersal algorithm* [34] used in some schemes or systems

- *C. Huang is with Microsoft Research, One Microsoft Way, Redmond, WA 98052. E-mail: cheng.huang@microsoft.com.*
- *L. Xu is with the Department of Computer Science, Wayne State University, 5143 Cass Avenue, 431 State Hall, Detroit, MI 48202. E-mail: lihao@cs.wayne.edu.*

Manuscript received 5 Oct. 2006; revised 30 June 2007; accepted 31 Aug. 2007; published online 26 Sept. 2007.
Recommended for acceptance by B. Bose.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0394-1006.
Digital Object Identifier no. 10.1107/TC.2007.70830.

[1] is indeed just an RS code.) The computation overhead of using the RS code, however, is large, as demonstrated in several projects such as OceanStore [25]. Therefore, most commonly adopted schemes in practical systems are still full replication (which is $(n, 1)$), stripping without redundancy (corresponding to (n, n)), or single parity (which is $(n, n-1)$), rather than general (n, k) MDS codes, especially the RS code.

It is hence very important and useful to design general (n, k) codes with *both* MDS property and simple encoding and decoding operations. MDS *array codes* are such a class of codes with both properties.

Array codes have been studied extensively [17], [23], [8], [5], [7], [42], [43], [6], [15]. A common property of these codes is that their encoding and decoding procedures use only simple binary *XOR* (*exclusive OR*) operations, which can be easily and most efficiently implemented in hardware and/or software; thus, these codes are more efficient than the RS code in terms of computational complexity.

In an array code, each of the n (information or parity) symbols contain l *elements*. The code can be arranged in an array of size $l \times n$, that is, l rows and n columns. (When there is no ambiguity, we also refer to array elements as *symbols* for representation convenience.) Mapping to a storage system, all of the symbols in the same column are stored in the same storage node. If a storage node fails, then the corresponding column of the code is considered to be an *erasure*. (Here, we adopt a commonly used storage failure model, as discussed in [5], [15], where all of the symbols are lost if the host storage node fails.)

A few classes of MDS array codes have been successfully designed to recover double (simultaneous) storage node failures, that is, in coding terminology, codes of distance 3 that can correct two erasures [27]. The recent ones include the EVENODD code [5] and its variations, such as the RDP scheme [15], the X-Code [42], and the B-Code [43].

As storage systems expand, it becomes increasingly important to have MDS array codes of distance 4, which can correct three erasures, that is, codes that can recover from *triple* (simultaneous) node failures. (There have been parallel efforts to design near-optimal codes, that is, non-MDS codes, to tolerate triple failures, for example, recent results in [31].) Such codes will be very desirable in large storage systems, such as the Google File System [19]. To the best of our knowledge, there exist only a few classes of MDS array codes of distance 4: the generalized EVENODD code [7], [6], the Blaum-Roth code [9], and, more recently, Feng et al.'s code [18]. The Blaum-Roth code is nonsystematic, which requires decoding operations in any data retrieval even without node failures and thus probably is not desirable in storage systems. The generalized EVENODD code and Feng et al.'s code are already much more efficient than the RS code in both encoding and decoding operations. However, a natural question we ask is can its decoding complexity be further reduced? In this paper, we provide a positive answer with a new coding scheme, which we call the *STAR* code.

The STAR code is an alternative extension of the EVENODD code, a $(k+3, k)$ MDS code that can recover triple node failures (erasures). The structure of the code is very similar to the generalized EVENODD code and their encoding complexities are also the same. Our key contribution, however, is to exploit the geometric property of the EVENODD code and provide a new construction for an additional parity column. The difference in construction of the third parity column leads to a more efficient decoding algorithm than the generalized EVENODD code for triple-erasure recovery. Our analysis shows the decoding complexity of the STAR code is very close to three XORs per symbol, the theoretical *lower bound*, even when k is small, whereas the generalized EVENODD could need up to 10 XORs (Section 7) per symbol. Thus, the STAR code is perhaps the most efficient existing code in terms of decoding complexity when recovering from triple erasures.

It is worth noting that, for practical storage systems, just as the generalized EVENODD code does, the STAR code extends from the EVENODD code by simply adding one more parity column (or storage node). Thus, any functioning system using the EVENODD code for tolerating double node failures can later easily add one more node to tolerate triple node failures, with no need to modify any existing data. This property of the STAR code makes a practical storage system much easier to expand. (The EVENODD code itself has a similar property with respect to the single-parity code, the one used in RAID-5-type systems.)

It should be noted that the original generalized EVENODD papers [7], [6] only provide generic erasure decoding algorithms for multiple erasures. It might be possible to design a specific triple-erasure decoding algorithm to reduce the decoding complexity of the generalized EVENODD. It is, however, not clear whether such a decoding algorithm for the generalized EVENODD code can achieve the same complexity as the STAR code. Interested readers thus are welcome and encouraged to design an optimized triple-erasure decoding algorithm for the generalized EVENODD code and compare its performance with our decoding algorithm for the STAR code.

This paper is organized as follows: We first briefly describe the EVENODD code in Section 2, on which the STAR code is derived in Section 3. In Section 4, we constructively prove that the STAR code can correct any triple erasures by providing detailed decoding algorithms. We also provide an algebraic description of the STAR code and show that the STAR code's distance is 4 in Section 5. We then analyze and discuss the STAR decoding complexity in Section 6 and make comparisons with related codes in Section 7. We further share our implementation and performance tests of the STAR code in Section 8 and conclude in Section 9.

2 EVENODD Code: Double-Erasure Recovery

2.1 EVENODD Code and Encoding

We first briefly describe the EVENODD code [5], which was initially proposed to address disk failures in disk array systems. Data from multiple disks form a two-dimensional array, with one disk corresponding to one column of the array. A disk failure is equivalent to a column erasure. The EVENODD code uses two parity columns together with

TABLE 1
Notations

Symbol	Definition
p	number of information columns
$a_{i,j}$	symbol at the cell of row i, column j
S_j	adjustor of parity column j ($j = 1$, 2)
e_r, e_s, e_t	index of erasure columns
u, v	distance between erasure columns, $u =< e_s - e_r >_p$ and $u =< e_t - e_s >_p$
$\hat{S}_j$	adjustor recoverer for parity column j ($j = 0$, 1, 2)
$\tilde{s}_{i,j}$	syndrome at the cell of row i and $j + p$, which equals to the XOR sum of parity symbol $a_{i,j+p}$ and its corresponding non-erasure information symbols
$\hat{s}_{i,j}$	XOR sum of syndrome and adjustor, $\hat{s}_{i,j} = \tilde{s}_{i,j} \oplus S_j$
C_{i,e_r}	cross whose top left symbol is a_{i,e_r}
l_d	number of crosses in a ring
l_h	number of rows in a ring after cancellations

p information columns, where p is a prime number (see Table 1 for a list of notations used in this paper). As already observed [5], [15], p being prime in practice does not limit the k parameter in a real system configuration with a simple technique called codeword *shortening* [27]. The code ensures that all information columns are fully recoverable when *any* two disks fail. In this sense, it is an optimal two-erasure correcting code, that is, it is a $(p+2, p, 3)$ MDS code. Besides this MDS property, the EVENODD code is computationally efficient in both encoding and decoding, which needs only XOR operations.

The encoding process considers a $(p-1) \times (p+2)$ array, where the first p columns are information columns and the last two are parity columns. Symbol $a_{i,j}$ $(0 \leq i \leq p-2, 0 \leq j \leq p+1)$ represents symbol i in column j. A parity symbol in column p is computed as the XOR sum of all information symbols in the same row. The computation of column $(p+1)$ takes the following steps: First, the array is augmented with an imaginary row $p-1$, where all symbols are assigned *zero* values (note that all of the symbols are binary). The XOR sum of all information symbols along the same diagonal (indeed a diagonal of *slope* 1) is then computed and assigned to their corresponding parity symbol, as marked by different shapes in Fig. 1. Symbol $a_{p-1,p+1}$ now becomes nonzero and is called the EVENODD *adjuster*. To remove this symbol from the array, *adjuster complement* is performed, which adds (XOR addition) the adjuster to all symbols in column $p+1$.

The encoding operation can be algebraically described as follows $(0 \leq i \leq p-2)$:

$$a_{i,p} = \bigoplus_{j=0}^{p-1} a_{i,j},$$

$$a_{i,p+1} = S_1 \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i-j \rangle_p, j} \right),$$

$$\text{where} \quad S_1 = \bigoplus_{j=0}^{p-1} a_{\langle p-1-j \rangle_p, j}.$$

Here, S_1 is the EVENODD adjuster and $\langle x \rangle_p$ denotes $x \bmod p$. Refer to [5] for more details.

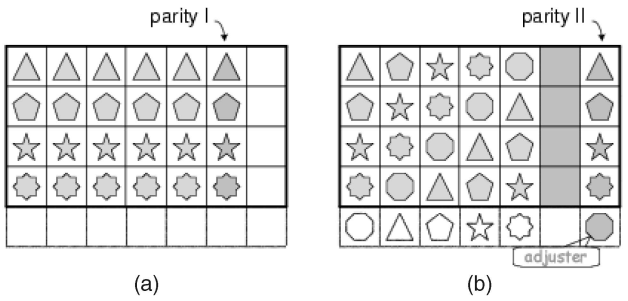

Fig. 1. EVENODD code encoding. (a) Horizontal redundancy. (b) Diagonal redundancy.

2.2 EVENODD Erasure Decoding

The EVENODD code is a double-erasure correcting MDS code and any two column erasures in a coded block can be fully recovered. With regard to the locations of the erasures, the work in [5] divides decoding into four cases. Here, we only summarize the most common and interesting one, where neither of the erasures is a parity column. Note that the other three cases are special ones and can be dealt with easily. A decoder first computes horizontal and diagonal *syndromes* as the XOR sum of all available symbols along those directions. Then, a *starting* point of decoding can be found which is guaranteed to be the only erasure symbol in its diagonal. The decoder recovers this symbol and then moves horizontally to recover the symbol in the other erasure column. It then moves diagonally to the next erasure symbol and horizontally again. Upon completing this *zigzag* process, all erasure symbols are fully recovered. In the example shown in Fig. 2 ($p = 5$), the starting point is symbol $a_{2,2}$ and the decoder moves from $a_{2,2}$ to $a_{2,0}, a_{0,2}, a_{0,0} \cdots$ and finally completes at $a_{1,0}$.

3 STAR Code Encoding: Geometric Description

Extending from the EVENODD code, the STAR code consists of $p+3$ columns, where the first p columns contain information data and the last three columns contain parity data. The STAR code uses the exact same encoding rules of the EVENODD code for the first two parity columns, that is, without the third parity column, the STAR code is just the EVENODD code. The extension lies in the last parity column, column $p+2$. This column is computed very similarly to column $p+1$ but along diagonals of slope -1 instead of slope 1 as in column $p+1$. (The original

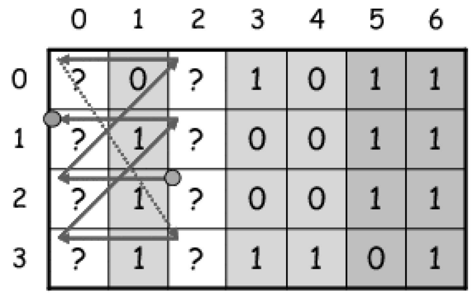

Fig. 2. EVENODD code decoding.

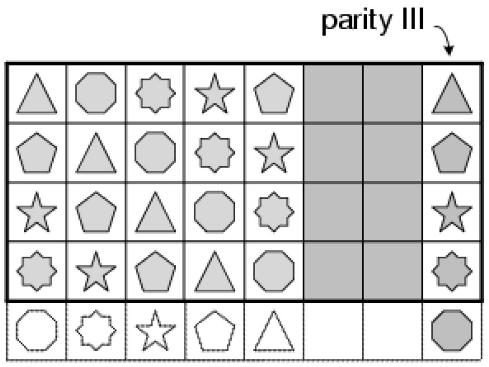

Fig. 3. STAR code encoding.

generalized EVENODD code [7], [6] uses slope 2 for the last parity column. That is the only difference between the STAR code and the generalized EVENODD code. However, as will be seen from the following section, it is this difference that makes it much easier to design a much more efficient decoding algorithm for correcting triple erasures.) For simplicity, we call this *antidiagonal* parity. The procedure is depicted in Fig. 3, where symbol $a_{p-1,p+2}$ in parity column $p+2$ is also an *adjuster*, similar to the EVENODD code. The adjuster is then removed from the final code block by the similar adjuster complement operation. Algebraically, the encoding of parity column $p+2$ can be represented as $(0 \le i \le p-2)$:

$$a_{i,p+2} = S_2 \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i+j \rangle_p, j} \right), \quad \text{where } S_2 = \bigoplus_{j=0}^{p-1} a_{\langle j-1 \rangle_p, j}.$$

4 STAR Code Erasure Decoding

The essential part of the STAR code is the erasure decoding algorithm. As will be presented in this section, the decoding algorithm involves pure XOR operations, which allows efficient implementation and thus is very desirable for virtually all related applications. The MDS property of the STAR code, which guarantees the recovery from arbitrary triple erasures, is explained along with the description of the decoding algorithm. A mathematical proof of this property will be given in a later section.

STAR code decoding can be divided into two cases based on different erasure patterns: 1) decoding without parity erasures, where all erasures are information columns, and 2) decoding with parity erasures, where at least one erasure is a parity column. The former case is harder to decode and is the focus of this section. This case in turn can be divided into two subcases, symmetric and asymmetric, based on whether the erasure columns are evenly spaced. The latter case, on the other hand, handles several special situations. It is much simpler and will be discussed in Section 4.4.

4.1 Decoding without Parity Erasures: Symmetric Case

We consider the recovery of triple information column erasures at position e_r, e_s, and e_t $(0 \le e_r, e_s, e_t \le p-1)$, among the total $p+3$ columns. Without loss of generality,

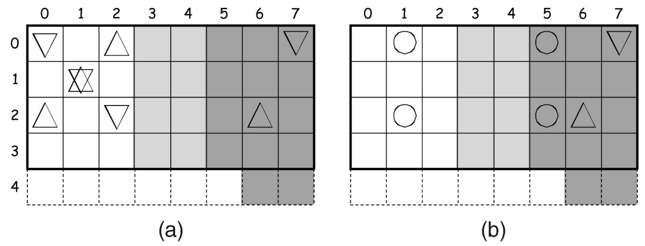

Fig. 4. STAR code decoding (symmetric case) (erasure columns $e_r = 0$, $e_s = 1$, and $e_t = 2$). (a) One cross. (b) Starting point.

assume that $e_r < e_s < e_t$. Let $u = e_s - e_r$ and $v = e_t - e_s$. The symmetric case deals with erasure patterns satisfying $u = v$.

Example 1. The decoding algorithm can be visualized with a concrete example, where $e_r = 0$, $e_s = 1$, $e_t = 2$, and $p = 5$, as shown in Fig. 4a, where empty columns are erasures.

The decoding procedure consists of the following four steps.

4.1.1 *Recover Adjusters and Calculate Syndromes*

Given the definitions of the adjusters S_1 and S_2, it is easy to see that they can be computed as the XOR sums of all of the symbols in parity columns 5 and 6 and parity columns 5 and 7, respectively. Assign the adjusters to symbols $a_{4,6}$ and $a_{4,7}$ and then apply them through XOR additions to all of the rest of the parity symbols in columns 6 and 7. This process reverses the adjuster complement. Still use $a_{i,j+p}$ $(0 \le j \le 2)$ to denote symbols in the parity columns.

Now, the parity check property of the code states that the XOR sum of all symbols along any parity check direction (horizontal, diagonal, and antidiagonal) should equal *zero*. Due to erasure columns, however, the XOR sum of the remaining symbols is nonzero, which we denote as the *syndrome* for the particular parity direction. To be specific, syndrome $\tilde{s}_{i,j}$ denotes the XOR sum of parity symbol $a_{i,j+p}$ and its corresponding nonerasure information symbols. For example, $\tilde{s}_{0,0} = a_{0,5} \oplus (a_{0,3} \oplus a_{0,4})$, $\tilde{s}_{0,1} = a_{0,6} \oplus (a_{2,3} \oplus a_{1,4})$, etc. To maintain the parity property, the XOR sum of all erasure information symbols along any parity check (redundancy) direction needs to match the corresponding syndrome. For example, $\tilde{s}_{0,0} = a_{0,0} \oplus a_{0,1} \oplus a_{0,2}$, $\tilde{s}_{0,1} = a_{0,0} \oplus a_{4,1} \oplus a_{3,2}$, etc.

In general, this step can be summarized as

1. adjusters recovery $(j = 0, 1, 2)$,

$$\hat{S}_j = \bigoplus_{i=0}^{p-2} a_{i,p+j},$$

$S_1 = \hat{S}_0 \oplus \hat{S}_1$ and $S_2 = \hat{S}_0 \oplus \hat{S}_2$,

2. reversion of adjuster complement $(0 \le i \le p-2)$,

$$a_{i,p+1} = a_{i,p+1} \oplus S_1,$$
$$a_{i,p+2} = a_{i,p+2} \oplus S_2,$$

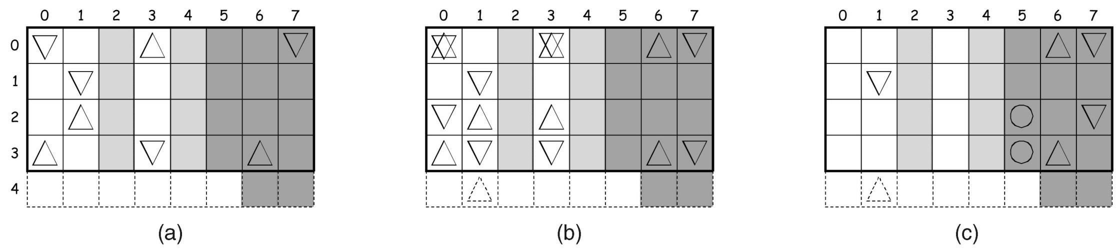

Fig. 5. STAR code decoding (asymmetric case) (erasure columns $e_r = 0$, $e_s = 1$, and $e_t = 3$). (a) One cross. (b) Multiple crosses. (c) Starting point.

3. syndrome calculation,

$$\tilde{s}_{i,0} = a_{i,0} \oplus \left(\bigoplus_{j=0}^{p-1} a_{i,j} \right),$$

$$\tilde{s}_{i,1} = a_{i,1} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle p+i-j \rangle_p, j} \right),$$

$$\tilde{s}_{i,2} = a_{i,2} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i+j \rangle_p, j} \right),$$

where $0 \le i \le p-1$ and $j \neq e_r$, e_s or e_t.

4.1.2 Finding a Starting Point

Recall that finding a starting point is the key step of EVENODD decoding, which seeks one particular diagonal with only one *unknown* symbol. This symbol is then recovered from its corresponding syndrome, which triggers the zigzag decoding process until all unknown symbols are recovered. In STAR decoding, however, it is *impossible* to find any parity direction (horizontal, diagonal, or antidiagonal) with only one unknown symbol. Hence, the approach adopted in the EVENODD decoding does *not* directly apply here and additional steps are needed to find a starting point.

For illustration purposes, we now assume that all syndromes are represented by the shadowed symbols in the three parity columns, as shown in Fig. 4a. Based on the diagonal parity property, it is clear that $\tilde{s}_{2,1}$ equals the XOR sum of three unknown symbols $a_{2,0}$, $a_{1,1}$, and $a_{0,2}$, as marked by "$\triangle$" signs in Fig. 4a. Similarly, $\tilde{s}_{0,2} = a_{0,0} \oplus a_{1,1} \oplus a_{2,2}$, which are all marked by "$\bigtriangledown$" signs along an antidiagonal.

Now, we introduce a new concept of *cross*, which represents the XOR sum of two syndromes (or corresponding unknown erasure symbols). Precisely, cross $C_{i,e_r} = \tilde{s}_{<i-e_r>_p,2} \oplus \tilde{s}_{<i+e_t>_p,1}$. For example, Fig. 4a shows cross $C_{0,0}$, which represents the XOR sum of $\tilde{s}_{0,2}$ and $\tilde{s}_{2,1}$. It also represents the XOR sum of unknown symbols $a_{0,0}$, $a_{1,1}$, $a_{2,2}$, $a_{2,0}$, $a_{1,1}$, and $a_{0,2}$, which indeed shows as a cross in Fig. 4a. Note that $a_{1,1}$ is XORed twice in the cross, so its value is canceled out. Hence, we have $C_{0,0} = a_{0,0} \oplus a_{2,2} \oplus a_{2,0} \oplus a_{0,2}$ (as shown in Fig. 4a). Moreover, from two horizontal parities, we know that $\tilde{s}_{0,0} = a_{0,0} \oplus a_{0,1} \oplus a_{0,2}$ and $\tilde{s}_{2,0} = a_{2,0} \oplus a_{2,1} \oplus a_{2,2}$. XORing them with cross $C_{0,0}$, we have $a_{0,1} \oplus a_{2,1} = C_{0,0} \oplus \tilde{s}_{0,0} \oplus \tilde{s}_{2,0} = \tilde{s}_{0,2} \oplus \tilde{s}_{2,1} \oplus \tilde{s}_{0,0} \oplus \tilde{s}_{2,0}$, which is shown in Fig. 4b.

To this end, we have computed the XOR sum of a pair of unknown symbols in column 1, that is, $a_{0,1} \oplus a_{2,1}$. Repeating this process and starting crosses from different rows (for example, $C_{1,0}$, $C_{2,0}$, etc.), we can obtain the XOR sum of any unknown symbol pair with a fixed distance 2 in column 1, that is, $a_{1,1} \oplus a_{3,1}$, $a_{2,1} \oplus a_{4,1}$, etc.

4.1.3 Recover Middle Erasure Column

In the previous step, we computed the XOR sum of arbitrary unknown symbol pair in column e_s with the fixed distance 2. Since symbol $a_{4,1}$ is an imaginary symbol with zero value, it is straightforward to recover symbol $a_{2,1}$. Next, symbol $a_{0,1}$ can be recovered as the XOR sum of the pair $a_{0,1}$ and $a_{2,1}$ is available. Consequently, symbols $a_{3,1}$ and $a_{1,1}$ are recovered. Indeed, in the next section, we will show that the recovery of all the symbols in column e_s is guaranteed.

4.1.4 Recover Side Erasure Columns

Now that column e_s is known, the first $p+2$ columns compose an EVENODD coded block with two erasures. Thus, this reduces to an EVENODD decoding of two erasures.

4.2 Decoding without Parity Erasures: Asymmetric Case

In this section, we describe the decoding of asymmetric erasure patterns, where $u \neq v$.

Example 2. Using the same STAR code as in the symmetric case, we now focus on a different erasure pattern. Let $e_r = 0$, $e_s = 1$, and $e_t = 3$ (as shown in Fig. 5). Note that, in this particular example, an asymmetric erasure pattern can be treated as a symmetric case by rotating e_r, e_s, and e_t. This point will be revisited in a later section. In this section, the goal is to design a general solution that does *not* rely on symmetry.

It turns out that the decoding process of asymmetric erasure patterns is very similar to the symmetric case. The only difference is in *finding a starting point*. As illustrated in Fig. 5a, we examine a particular cross (cross $C_{0,0}$). It now contains six unknown symbols: $a_{3,0}$, $a_{2,1}$, and $a_{0,3}$ (marked by "$\triangle$") and $a_{0,0}$, $a_{1,1}$, and $a_{3,3}$ (marked by "$\bigtriangledown$"). This is different from the four unknown symbols in the symmetric case because the two unknown symbols in column e_s are no longer canceled (due to $u \neq v$). Hence, one cross is not sufficient and the natural mitigation is to select multiple crosses. Indeed, the *key* of this step is to choose multiple crosses such that the following two conditions are satisfied:

Condition 1.

1. *Each cross is shifted vertically downward from a previous one by v symbols* (offset) *and*

2. *the bottom row of the last cross (after wrapping around)* steps over (coincides with) *the top row of the first cross.*

In the example, two crosses are chosen (cross $C_{0,0}$ and cross $C_{2,0}$), where $C_{2,0}$ is $v = 2$ symbols offset from $C_{0,0}$ and contains six erasure symbols: $a_{0,0}$, $a_{4,1}$, and $a_{2,3}$ (marked by "$\triangle$") and $a_{2,0}$, $a_{3,1}$, and $a_{0,3}$ (marked by "$\bigtriangledown$"), as shown in Fig. 5b. As a reminder, the XOR sum of the two crosses equals the XOR sum of the corresponding syndromes, that is, $C_{0,0} \oplus C_{2,0} = \tilde{s}_{3,1} \oplus \tilde{s}_{0,2} \oplus \tilde{s}_{0,1} \oplus \tilde{s}_{2,2}$.

On the other hand, the XOR sum of $C_{0,0}$ and $C_{2,0}$ includes symbols $a_{0,0}$ and $a_{0,3}$ twice, which are thus canceled out (the result of the bottom row of the last cross $C_{2,0}$ steps over the first cross $C0,0$). Moreover, the XOR sum includes all unknown symbols in rows 2 and 3, which can simply be replaced by their corresponding horizontal syndromes (marked by "$\bigcirc$") since $\tilde{s}_{2,0} = a_{2,0} \oplus a_{2,1} \oplus a_{2,3}$ and $\tilde{s}_{3,0} = a_{3,0} \oplus a_{3,1} \oplus a_{3,3}$. To this end, we get

$$a_{1,1} \oplus a_{4,1} = \tilde{s}_{3,1} \oplus \tilde{s}_{0,2} \oplus \tilde{s}_{0,1} \oplus \tilde{s}_{2,2} \oplus \tilde{s}_{2,0} \oplus \tilde{s}_{3,0},$$

as are marked in Fig. 5c.

The rest of the decoding is the same as in the symmetric case, as we can obtain the XOR sum of any unknown symbol pair with a fixed distance 3 in column 1 by choosing different sets of multiple crosses. In other words, we can get $a_{0,1} \oplus a_{3,1}$, $a_{2,1} \oplus a_{0,1}$, etc. Then, the middle column can be decoded and all other erasure symbols can be recovered.

4.3 Correctness of Decoding without Parity Erasures

In this section, we show that the selection of multiple crosses yields deterministic results and also guarantees the recovery of the middle column. Note that the method works with both symmetric and asymmetric cases, which are hence treated the same here.

Conceptually, the first condition of choosing crosses ensures the alignment of unknown symbols in the middle erasure column with those in the side erasure columns. Essentially, it groups unknown symbols together and replaces them with known syndromes. This is one way to cancel unknown symbols and results in a chain of crosses. The other way to cancel unknown symbols comes from the second condition, where unknown symbols in the *head row* (the first row of the first cross) of the chain of crosses are canceled with those in the *tail row* (the bottom row of the last cross). This is analogous to "gluing" the head of the first cross with the tail of the last one and turns the chain of crosses into a *ring*. The number of crosses in the ring is completely determined by the erasure pattern (e_r, e_s, and e_t) and the STAR code parameter p. Lemma 1 ensures the existence of such a ring for any given $u = e_s - e_r$, $v = e_t - e_s$, and p.

Lemma 1. *A ring satisfying Condition 1 always exists and consists of l_d $(0 \leq l_d < p)$ crosses, where l_d is determined by the following equation:*

$$\langle u + l_d v \rangle_p = 0, \tag{1}$$

where $0 \leq u, v < p$.

Proof. Since p is a prime number, integers modulo p define a finite field $GF(p)$. Let v^{-1} be the unique inverse of v in this field. Then, $l_d = (p - u)v^{-1}$ exists and is unique. □

Given a ring, rows with three unknown symbols are substituted with horizontal syndromes (*substitution*) and symbols with even numbers are simply removed (*simple cancellation*). For simplicity, we refer to both cases as *cancellations*. Eventually, there are exactly two rows left with unknown symbols, which is confirmed by Lemma 2.

Lemma 2. *After cancellations, there are exactly two rows with unknown symbols in a ring. The row numbers are u and $p - u$, as offsets from the top row of the first cross.*

Proof. To simplify the proof, we only examine the ring whose first cross starts at row 0. Now, the first cross contains two unknown symbols in column e_r and they are in rows 0 and $u + v$. We can represent them with a polynomial $(1 + x^{u+v})$, where power values (modulo p) of x correspond to row entices. Similarly, the unknown symbols in column e_s can be represented as $(x^u + x^v)$. Therefore, the first cross can be completely represented by $(1 + x^{u+v} + x^u + x^v)$ and the l_1th cross by

$$(1 + x^{u+v} + x^u + x^v)x^{l_1 v},$$

where $0 \leq l_1 < l_d$ and the coefficients of x are binary. Note that we do not explicitly consider unknown symbols in column e_t, which are reflected by polynomials representing column e_r. Using this representation, the cancellation of a polynomial term includes both cases of substitution and simple cancellation. The XOR sum of all crosses is

$$\begin{aligned} &\sum_{l_1=0}^{l_d-1} (1 + x^{u+v} + x^u + x^v)x^{l_1 v} \\ &= (1 + x^u) \sum_{l_1=0}^{l_d-1} (1 + x^v)x^{l_1 v} \\ &= (1 + x^u)(1 + x^{p-u}) \\ &= x^u + x^{p-u}, \end{aligned} \tag{2}$$

where l_d is substituted using the result from Lemma 1. Thus, only two rows with unknown symbols are left after cancellations and the distance between them is $d = \langle p - 2u \rangle_p$. □

It is important to point out that unknown symbols in the remaining two rows are *not* necessarily in column e_s. For example, if $e_r = 0$, $e_s = 2$, and $e_t = 3$, the remaining unknown symbols would be $a_{2,0}$, $a_{2,3}$, $a_{3,0}$, and $a_{3,3}$, which are indeed in columns e_r and e_t. This is similar to the symmetric case, where we can use two additional horizontal syndromes and obtain the XOR sum of the corresponding unknown symbol pair in column e_s.

To summarize this step, we denote l_h to be the number of rows in a ring which are canceled through substitutions and define the set of corresponding row indices as $F_h = \{h_{l_2} | 0 \leq l_2 < l_h\}$. The set F_h is simply obtained by enumerating all crosses of the ring and then counting rows with three unknown symbols. Let $\tilde{a}_u$ denote the XOR sum of the unknown symbol pair a_{0,e_s} and $a_{\langle p-2u\rangle_p,e_s}$. Then, the ith pair has

$$\tilde{a}_{u+i} = \bigoplus_{l_1=0}^{l_d-1} \tilde{s}_{\langle -e_r+i\rangle_p,2} \bigoplus_{l_2=0}^{l_h-1} \tilde{s}_{\langle h_{l_2}+i\rangle_p,0}, \bigoplus_{l_1=0}^{l_d-1} \tilde{s}_{\langle e_t+i\rangle_p,1}, \tag{3}$$

where $0 \le i \le p-1$.

Now, we show that the middle column can be recovered entirely by Lemma 3.

Lemma 3. *Given the XOR sum of an arbitrary symbol pair with a fixed distance d, all symbols in the column are recoverable if there is at least one symbol available.*

Proof. Since p is prime, $F = \{\langle di\rangle_p | 0 \le i \le p-1\}$ covers all integers in $[0, p)$. Therefore, a "tour" starting from row $p-1$ with the stride size d will visit all other rows exactly once before returning to it. As the symbol in row $p-1$ is always available (zero indeed) and the XOR sum of any pair with distance d is also known, all symbols can then be recovered along the tour. □

In summary, this step computes

$$\tilde{a}_{\langle (p-1)-di\rangle_p} = \tilde{a}_{\langle (p-1)-di\rangle_p} \oplus a_{\langle (p-1)-d(i-1)\rangle_p}, \tag{4}$$

where $0 \le i \le p-1$. Then, $a_{i,e_s} = \tilde{a}_i$ (where there are two unknown symbols left in the ring after cancellations) or $a_{i,e_s} = \tilde{a}_i \oplus \tilde{s}_{i,0}$ (where four unknown symbols are left) for all is. Thus far, column e_s is completely recovered.

4.4 Decoding with Parity Erasures

In this section, we consider the situation when there are erasures in parity columns. The decoding is divided into the following three subcases:

4.4.1 Column $p+2$ Is an Erasure

This then reduces to EVENODD decoding of two erasures. Note that this case also takes care of all patterns with fewer than three erasures.

4.4.2 Column $p+1$ Is an Erasure, while $p+2$ Is Not

This is almost the same as the previous case except that, now, the "EVENODD" coded block consists of the first $p+1$ columns and column $p+2$. In fact, this coded block is no longer a normal EVENODD code but rather a mirror reflection of one over the horizontal axis. Nevertheless, it can be decoded with a slight modification of the EVENODD decoding, which we simply leave to interested readers.

4.4.3 Column p Is an Erasure, while $p+1$ and $p+2$ Are Not

In this case, $0 \le e_r < e_s \le p-1$ and $e_t = p$.

First, it is not possible to recover adjusters S_1 and S_2 as symbols in column p are unknown. However, $S_1 \oplus S_2$ is still computable, which simply equals the XOR sum of all symbols in columns $p+1$ and $p+2$. This is easy to see from the definitions of S_1 and S_2; S_0 is added twice and canceled out. It is thus possible to reverse the adjuster complement. The results from syndrome calculation are the XOR sums of syndromes and their corresponding adjusters, rather than syndromes themselves. We use $\hat{s}_{i,j}$ to denote the results, which thus satisfy

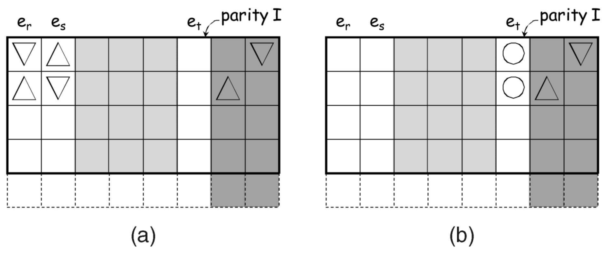

Fig. 6. STAR code decoding (parity erasure) (erasure columns $e_r = 0$, $e_s = 1$, and $e_t = 5$). (a) One cross. (b) Starting point.

$$\hat{s}_{i,j} = \tilde{s}_{i,j} \oplus S_j, \tag{5}$$

where $j = 1$ or 2 and $0 \le i \le p-1$. Note that $\hat{s}_{i,0} = \tilde{s}_{i,0}$ for all is.

The next step is similar to the decoding of the symmetric case without parity erasures as it is also the case that only one cross is sufficient to construct a ring.

Example 3. We again use an example to illustrate this. Let $e_r = 0$, $e_s = 1$, and $e_t = 5$, as shown in Fig. 6. Consider cross $C_{0,0}$, which consists of unknown symbols $a_{0,0}$, $a_{1,1}$, $a_{1,0}$, and $a_{0,1}$. From the definition of a cross, we know that $C_{0,0} = \hat{s}_{1,1} \oplus \hat{s}_{0,2}$. By substituting (5), we can get the following:

$$a_{0,0} \oplus a_{1,1} \oplus a_{1,0} \oplus a_{0,1} = \hat{s}_{1,1} \oplus \hat{s}_{0,2} \oplus S_1 \oplus S_2.$$

Hence, the XOR sum of the unknown symbols in $C_{0,0}$ can be computed. Combined with horizontal parities (following the same approach as in the symmetric case), the XOR sum of two unknowns ($\hat{s}_{0,0} \oplus \hat{s}_{1,0}$) in the erasure parity column can be computed (marked by "○" in Fig. 6b). It should now be clear that any pair of unknowns with distance 1 in erasure column $e_t = 5$ can be computed and, thus, the entire column can be recovered. After column 5 is recovered, the first seven columns can again be treated as an EVENODD-coded block with two erasures at columns $e_r = 0$ and $e_s = 1$. Therefore, the EVENODD decoding can be applied to complete the recovery of all of the remaining unknown symbols.

This procedure is summarized as follows:

$$S_1 \oplus S_2 = \left(\bigoplus_{i=0}^{p-2} a_{i,p+1}\right) \oplus \left(\bigoplus_{i=0}^{p-2} a_{i,p+2}\right)$$

and

$$\hat{s}_{i,0} = a_{i,0} \oplus \left(\bigoplus_{j=0}^{p-1} a_{i,j}\right),$$

$$\hat{s}_{i,1} = a_{i,1} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle p+i-j\rangle_p,j}\right),$$

$$\hat{s}_{i,2} = a_{i,2} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i+j\rangle_p,j}\right),$$

where $0 \le i \le p-1$ and $j \ne e_r$ or e_s. Then,

$$\tilde{a}_i = \hat{s}_{\langle e_s+i\rangle_p,1} \oplus \hat{s}_{\langle -e_r+i\rangle_p,2} \oplus S_1 \oplus S_2,$$

where $0 \leq i \leq p-1$, and

$$\tilde{a}_{\langle (p-1)-ui\rangle_p} = \tilde{a}_{\langle (p-1)-ui\rangle_p} \oplus a_{\langle (p-1)-u(i-1)\rangle_p},$$

where $1 \leq i \leq p-1$. Next, column p is recovered as

$$a_{i,p} = \tilde{a}_i \oplus \hat{s}_{i,0}$$

for all is. Finally, the EVENODD decoding is applied to recover the remaining two columns.

Putting all of the above cases together, we conclude this section with the following theorem:

Theorem 1. *The STAR code can correct any triple column erasures and, thus, it is a $(p+3,p)$ MDS code.*

5 Algebraic Representation of the STAR Code

As described in [5], each column in the EVENODD code can be regarded algebraically as an element of a polynomial ring, which is defined with multiplication taken modulo $M_p(x) = (x^p-1)/(x-1) = 1 + x + \cdots + x^{p-2} + x^{p-1}$. For the ring element x, it is shown that its multiplicative order is p. Using β to denote this element, column j $(0 \leq j \leq p+1)$ can be represented using the notation $a_j(\beta) = a_{p-2,j}\beta^{p-2} + \cdots + a_{1,j}\beta + a_{0,j}$, where $a_{i,j}$ $(0 \leq i \leq p-2)$ is the ith symbol in the column. Note that the multiplicative inverse of β exists and can be denoted as β^{-1}. Applying the same notations to the STAR code, we can then get its parity check matrix as

$$H = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & 0 & 0 \\ 1 & \beta & \cdots & \beta^{p-1} & 0 & 1 & 0 \\ 1 & \beta^{-1} & \cdots & \beta^{-(p-1)} & 0 & 0 & 1 \end{bmatrix}. \tag{6}$$

It is not hard to verify that, as in [7], any three columns in the parity check matrix are linearly independent. Therefore, the minimum distance of the STAR code is indeed 4 (each column is regarded as a single element in the ring) and, thus, arbitrary triple (column) erasures are recoverable. This is an alternative way of showing its MDS property.

6 Complexity Analysis

In this section, we analyze the complexity of the STAR code erasure decoding. The complexity is dominated by XOR operations; thus, we count the total number of XORs and use this as a measurement of the decoding complexity. Since decoding without parity erasures is the most complicated case, including both asymmetric and symmetric erasure patterns, our analysis is focused on this case.

6.1 Erasure Decoding Complexity

It is not difficult to see that the complexity can be analyzed individually for each of the four decoding steps. Note that a complete STAR code consists of p information columns and $r = n - k = 3$ parity columns. When there are only k $(k \leq p)$ information columns, we can still use the same code by resorting to the *shortening* technique, which simply assigns zero value to all symbols in the last $p-k$ information columns. Therefore, in the analysis here, we assume that the code block is a $(p-1) \times (k+3)$ array.

In Step 1, the calculation of S_0 takes $(p-2)$ XOR operations and those of S_1 and S_2 take $(p-1)$ XORs each. The reversion of adjuster complement takes $2(p-1)$ XORs in total. Directly counting the XORs of the syndrome calculations is fairly complicated and we can resort to the following alternative approach: First, it is easy to see that the syndrome calculations of any parity direction for a code block without erasures (a $(p-1) \times (p+3)$ array) take $(p-1)p$ XORs. Then, notice that any information column contributes $(p-1)$ XORs to the calculations. Therefore, for a code block with $(k-3)$ information columns (with triple erasures), the number of XORs becomes $(p-1)p - (p-k+3)(p-1) = (k-3)(p-1)$. In total, the XORs in this step are

$$\begin{aligned} &(p-2) + 2(p-1) + 2(p-1) + 3(k-3)(p-1) \\ &= (3k-4)(p-1) - 1. \end{aligned} \tag{7}$$

In Step 2, the computation of each ring takes $(2l_d + l_h - 1)$ XORs and there are $(p-1)$ rings to compute. Thus, the number of XORs is

$$(2l_d + l_h - 1)(p-1). \tag{8}$$

In Step 3, it is easy to see that the number of XORs is

$$(p-1) - 1 = p - 2. \tag{9}$$

In Step 4, the horizontal and the diagonal syndromes need to be updated with the recovered symbols of column e_s, which takes $2(p-1)$ XORs. Note that there is no need to update the antidiagonal syndromes because the decoding hereafter deals with only double erasures. The zigzag decoding then takes $2(p-1)-1$ XORs. Therefore, the number of XORs in this step is

$$2(p-1) + 2(p-1) - 1 = 4(p-1) - 1. \tag{10}$$

Note that, in Step 2, the number of XORs is computed assuming the case where only two unknown symbols are left in a ring after cancellations. If the other case happens, where four unknown symbols are left, additional $(p-1)$ XOR operations are needed to recover column e_s. However, this case does *not* need to update the horizontal syndromes in Step 4 and thus saves $(p-1)$ XORs there. Therefore, it is just a matter of moving XOR operations from Step 2 to Step 4 and the total number remains the same for both cases.

In summary, the total number of XORs required to decode triple information column erasures can be obtained by putting (7), (8), (9), and (10) together as

$$\begin{aligned} &(3k-4)(p-1) - 1 + (2l_d + l_h - 1)(p-1) \\ &\quad + (p-2) + 4(p-1) - 1 \\ &= (3k + 2l_d + l_h)(p-1) - 3 \end{aligned} \tag{11}$$

$$\approx (3k + 2l_d + l_h)(p-1), \tag{12}$$

which becomes $3 + (2l_d + l_h)/k$ per symbol after normalization by the total number of information symbols $(p-1)k$.

6.2 A Decoding Optimization

From (12), we can see that, for fixed code parameters k and p, the decoding complexity depends on l_d and l_h, which are

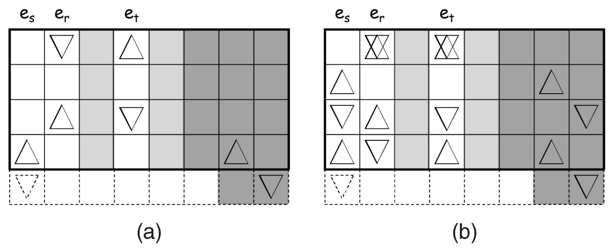

Fig. 7. Optimization of STAR decoding. (a) One cross. (b) Multiple crosses.

completely determined by actual erasure patterns (e_r, e_s, and e_t). In Section 4, we present an algorithm to construct a ring of crosses, which yields a starting point for successful decoding. Within the ring, each cross is $v = e_t - e_s$ symbols offset from the previous one. From (2), there are exactly two rows with unknown symbols left after cancellations. Based on the symmetric property of the ring construction, it is not difficult to show that using offset $u = e_s - e_r$ will also achieve the same goal. Moreover, if choosing u as the offset results in smaller l_d and l_h values (to be specific, smaller $2l_d + l_h$), then it is certainly better to do so.

Moreover, the aforementioned decoding algorithm assumes that $e_r < e_s < e_t$. Although it helps to visualize the key procedure of finding a starting point, such an assumption is unnecessary. Indeed, it is easy to verify that all proofs in Section 4 still hold without this assumption. By swapping values among e_r, e_s, and e_t, it might be possible to reduce the decoding complexity. For instance, in the previous example, $e_r = 0, e_s = 1$, and $e_t = 3$ result in $l_d = 2$ and $l_h = 2$. Instead, if letting $e_r = 1$, $e_s = 0$, and $e_t = 3$, then $u = -1$ and $v = 3$. The corresponding single cross is shown in Fig. 7a. It is clear that two crosses close a ring (shown in Fig. 7b) which contains exactly two rows (rows 1 and 4) with unknown symbols after cancellations. This choice yields the same $l_d = 2$ and $l_h = 2$. However, if we let $e_r = 0$, $e_s = 3$, and $e_t = 1$, we can get $u = e_s - e_r = 3$ and $v = e_t - e_s = -2$. It is easy to verify that the unknown symbols in column e_s are canceled in every single cross. Thus, the complexity is reduced by this choice, as $l_d = 1$ and $l_h = 0$ now. (As a matter of fact, this is an equivalence of the symmetric case.) Note that, for general u and v, the condition of symmetry now becomes $\langle u - v \rangle_p = 0$, instead of simply $u = v$.

Now let us revisit the ring construction algorithm described in Section 4. The key point there is to select multiple crosses such that the bottom row of the last cross "steps over" the top row of the first one and there are exactly two rows left with unknown symbols after cancellations. Further examination reveals that it is possible to construct rings using alternative approaches. For instance, the crosses can be selected in such a way that, *in the middle column*, the bottom symbol of the last cross "steps over" the top symbol of the first one. Naturally, one might wonder whether there are other cross selections which might lead to minimum decoding complexity.

Next, we describe a method of selecting the minimum number of crosses while ensuring that there are two rows left with unknown symbols after cancellations. In this way, successful decoding is guaranteed and the decoding complexity is minimized as well. Recall that a single cross is represented by $C(x) = 1 + x^u + x^v + x^{u+v}$ and a cross of f symbol offset by $C(x)x^f$. Therefore, the construction of a ring is to determine a polynomial term $R(x)$ such that $C(x)R(x)$ results in exactly two entries. For instance, the example in Section 4 has $R(x) = 1 + x^2$ and $C(x)R(x) = x + x^4$. Theorem 2 shows that the decoding complexity is minimized if an $R(x)$ with minimum entries is chosen.

Theorem 2. *The decoding complexity is nondecreasing with respect to the number of crosses (l_d) in a ring.*

Proof. Whenever a new cross is included into the ring, two new nonhorizontal syndromes (one diagonal and one antidiagonal) need to be added to the XOR sum. With this new cross, at most four rows can be canceled (simple cancellation due to even numbers of additions), among which two can be mapped with this cross and the other two with an earlier cross. Thus, each cross adds two nonhorizontal syndromes but subtracts at most two horizontal syndromes. The complexity is thus nondecreasing with respect to the number of crosses. □

Note that l_d is in fact the number of entries in $R(x)$. An optimal ring needs to find an $R(x)$ with minimum entries, which then ensures that $C(x)R(x)$ has only two terms. An efficient approach to achieve this is to test all polynomials with two terms. If a polynomial is divisible by $C(x)$, then the quotient yields a valid $R(x)$. An $R(x)$ with minimum entries is then chosen to construct the ring. It is important to point out that there is no need to worry about common factors (always powers of x) between two terms in the polynomial as it is not divisible by $C(x)$. Thus, the first entry of all polynomials can be fixed as 1, which means that only $p-1$ polynomials $(1 + x^i, 0 < i \leq p-1)$ need to be examined. As stated in an earlier section, polynomials are essentially elements in the ring constructed with $M_p(x) = 1 + x + \cdots + x^{p-2} + x^{p-1}$. Based on the argument in [8], $(1 + x^u)$ and $(1 + x^v)$ are invertible in the ring. Thus, $C(x) = (1 + x^u)(1 + x^v)$ is also invertible and it is straightforward to compute the inverse using Euclid's algorithm. For instance, $C(x) = 1 + x + x^2 + x^3$ as $u = 1$ and $v = 2$ in the previous example. The generator polynomial $M_p(x) = 1 + x + x^2 + x^3 + x^4$ as $p = 5$. Applying Euclid's algorithm [27], it is clear that

$$1(\mathbf{1} + \mathbf{x} + \mathbf{x^2} + \mathbf{x^3} + \mathbf{x^4}) + x(\mathbf{1} + \mathbf{x} + \mathbf{x^2} + \mathbf{x^3}) = 1. \quad (13)$$

Thus, the inverse of $C(x)$ is $inv(C(x)) = x$. When examining the polynomial $1 + x^3$, we get $R(x) = inv(C(x))(1 + x^3) = x + x^4$ or, equivalently,

$$(1 + x + x^2 + x^3)(x + x^4) = 1 + x^3 \bmod M_p(x). \quad (14)$$

It is desirable that $R(x)$ carry the entry of power 0 since the ring always contains the original cross. Therefore, we multiply x to both sides of (14), which now becomes

$$(1 + x + x^2 + x^3)(1 + x^2) = x + x^4 \bmod M_p(x).$$

Thus, we have $R(x) = 1 + x^2$ and the ring can be constructed using two crosses ($l_d = 2$) with an offset of two symbols. Once the ring is constructed, it is straightforward to get l_h.

Note that this optimal ring construction only needs to be computed once in advance (offline). Thus, we do not count the ring construction in the decoding procedure.

7 Comparison with Existing Schemes

In this section, we compare the erasure decoding complexity of the STAR code to two other XOR-based codes: one proposed by Blaum et al. [7] (Blaum code hereafter) and the other by Blomer et al. [10].

The Blaum code is a generalization of the EVENODD code whose horizontal (the first) and diagonal (the second) parities are now regarded as redundancies of slope 0 and 1, respectively. A redundancy of slope $q-1$ ($q \geq 3$) generates the $q\mathrm{th}$ parity column. This construction is shown to maintain the MDS property for triple parity columns when the code parameter p is a prime number. The MDS property continues to hold for selected p values when the number of parities exceeds three. To make the comparison meaningful, we focus on the triple-parity case of the Blaum code. We compare the complexity of triple-erasure decoding in terms of XOR operations between the Blaum code and the STAR code. As in the previous sections, we confine all three erasures to information columns.

The erasure decoding of the Blaum code adopts an algorithm described in [8], which provides a general technique to solve a set of linear equations in a polynomial ring. Due to special properties of the code, however, ring operations are *not* required during the decoding procedure, which can be performed with pure XOR and shift operations. The algorithm consists of four steps, whose complexities are summarized as follows:

1. syndrome calculation: $3(k-3)(p-1)-1$,
2. computation of $\hat{Q}(x;z)$: $\frac{1}{2}r(3r-3)p$,
3. computation of the right-hand value: $r((r-1)p+(p-1))$, and
4. extracting the erasure values: $r(r-1)(2(p-1))$.

Here, $r = n - k = 3$. Therefore, the total number of XORs is

$$\begin{aligned} &3(k-3)(p-1)-1+9p+(9p-3)+12(p-1) \\ &= (3k+21)(p-1)+14 \end{aligned} \tag{15}$$

$$\approx (3k+21)(p-1), \tag{16}$$

which becomes $3 + 21/k$ per symbol after normalization by the total number of information symbols $(p-1)k$. Comparison results with the STAR code are shown in Fig. 8, where we can see that the complexity of the STAR decoding remains fairly constant and is just slightly above 3. Note that this complexity depends on actual erasure locations; thus, the results reported here are average values over all possible erasure patterns. The complexity of the Blaum code, however, is rather high for small k values, although it *does* approach 3 asymptotically. The STAR code is thus probably more desirable than the Blaum code. Fig. 8 also includes the complexity of the EVENODD decoding as a reference, which is roughly constant and slightly above two XORs per symbol. Note that, in Fig. 8, p is always taken for each k as the next smallest prime.

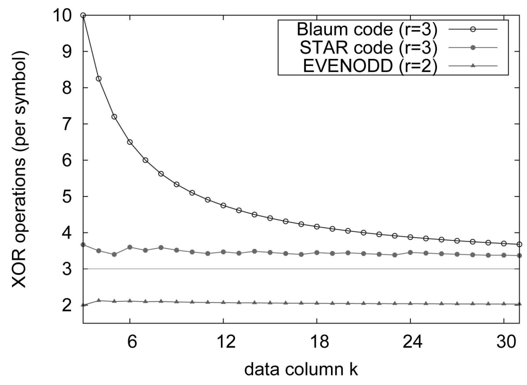

Fig. 8. The complexity comparisons $(r = n - k)$.

Further reflection on the Blaum code and the STAR code would reveal that the construction difference between them lies solely on the choice of the third redundancy slope, where the Blaum code uses slope 2 and the STAR code -1. One might wonder whether the decoding approach adopted here could be applied to the Blaum code as well. Based on STAR decoding's *heavy* reliance on the geometric property of individual crosses in the step to find a starting point, it seems difficult to achieve the same ring construction in the Blaum code when symmetry is no longer obvious. Moreover, the intuitiveness of the decoding process would be completely lost even if it is possible at all. Instead, we would be more interested in investigating whether the STAR code construction, in particular the decoding approach, could be extended to handle more than triple erasures, as the Blaum code already does.

The XOR-based code proposed in [10] uses Cauchy matrices to construct an RS code. It replaces generator matrix entries, information, and parity symbols with binary representations. Then, the encoding and decoding can be performed with primarily XOR operations. To achieve maximum efficiency, it requires the message length to be multiples of 32 bits. In that way, the basic XOR unit is 32 bits, or a single word, and can be performed by a single operation. To compare with this scheme fairly, we require the symbol size of the STAR code to be multiples of 32 bits too. It is shown that the XOR-based decoding algorithm in [10] involves krL^2 XOR operations and r^2 operations in a finite field $GF(2^L)$, where k and r are the numbers of information symbols and erasures, respectively. We ignore those r^2 finite-field operations (due to the inversion of a decoding coefficient matrix), which tend to be small as the number of erasures is limited. Then, the RS code's normalized decoding complexity (by the total information length of kL words) is rL. As the total number of symbols n $(= k + r)$ is limited by L $(n \leq 2^L)$, we have to increase L and, thus, in turn, the decoding complexity when n increases (see Table 2). Compared to Fig. 8, where the STAR code decoding complexity is slightly more than three XORs per symbol (multiples of 32 bits now), it is clear that the STAR code is much more efficient than the XOR-based RS code. However, as studied in Plank and Xu's recent work [33], carefully chosen RS codes can greatly improve encoding operations. If efficient decoding operations can also be developed adopting such methods, it will be very interesting

TABLE 2
Complexity of the RS Code (per 32 Bits)

# of total columns (n)	# of XORs ($= rL$)	
	$r = 2$	$r = 3$
$n \leq 8$	6	9
$9 < n \leq 16$	8	12
$17 < n \leq 32$	10	15
$33 < n \leq 64$	12	18

to compare them with the STAR decoding. Finally, the complexity of the *regular* (finite-field-based) RS code implementation (for example, [30]) turns out to be much higher than the XOR-based one, so we simply skip the comparison here.

8 Implementation and Performance

The implementation of the STAR code encoding is straightforward and simply follows the procedure described in Section 3. Thus, in this part, our main focus is on the erasure decoding procedure. As stated in Section 6, the decoding complexity is solely determined by l_d and l_h, given the number of information columns k and the code parameter p. As l_d and l_h vary according to actual erasure patterns, so does the decoding complexity. To achieve the maximum efficiency, we apply the optimization technique as described in the earlier section.

An erasure pattern is completely determined by the erasure columns e_r, e_s, and e_t (again assume that $e_r < e_s < e_t$) or, further, by the distances u and v between these columns, as the actual position of e_r does *not* affect l_d or l_h. Therefore, it is possible to set up a mapping from (u, v) to (l_d, l_h). To be specific, given u and v, the mapping returns the positions of horizontal, diagonal, and antidiagonal syndromes which would otherwise be obtained via ring constructions. The mapping can be implemented as a lookup table and the syndrome positions using bit vectors. Since the lookup table can be built in advance of the actual decoding procedure, it essentially shifts the complexity from online decoding to the offline preprocess. Note that the table lookup operation is only needed once for every erasure pattern; thus, there is no need to keep the table in memory (or cache). This is different from finite field-based coding procedures, where intensive table lookups are used to replace complicated finite field operations. For example, an RS code implementation might use an exponential table and a logarithm table for *each* multiplication/division. Furthermore, the number of entries in the lookup table is not large at all. For example, for code parameter $p = 31$, u and v are at most 30, which requires a table of at most $30 \times 30 = 900$ entries, where each entry contains three bit vectors (32 bits each) for the ring construction, 1 byte for the decoding pattern, and 1 byte for l_h. The cost of maintaining a few tables of this size is then negligible.

During the decoding procedure, u and v are calculated from the actual erasure pattern. Based on these values, the lookup table returns all syndrome positions, which essentially indicates the ring construction. The calculation of the ring is thus performed as the XOR sums of all of the

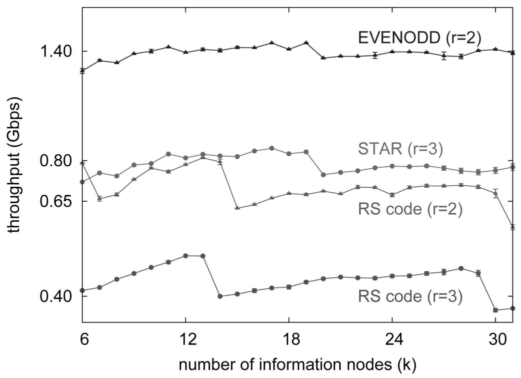

Fig. 9. Throughput performance ($r = n - k$ erasures are randomly generated among information nodes).

indicated syndromes. Then, the next ring is calculated by offsetting all syndromes with one symbol and the procedure continues until all rings are computed. The steps afterward are to recover the middle column and then the side columns, as detailed in Section 4.

We implement the STAR code erasure decoding procedure and apply it to a reliable storage platform [44]. The throughput performance is measured and compared to the publicly available implementation of the XOR-based RS code [11]. The results are shown in Fig. 9, where the size of a single data block from each node is 2,880 bytes and the number of information storage nodes (k) varies from 6 to 31. Note that our focus is on decoding erasures that all occur at information columns since, otherwise, the STAR code just reduces to the EVENODD code (when there is one parity column erasure) or a single parity code (when there are two parity column erasures), so we only simulate random information column erasures in Fig. 9. Recall that a single data block from each node corresponds to a single column in the STAR code and is divided into $p - 1$ symbols, so the block size needs to be a multiple of $p - 1$. For comparison purposes, we use 2,880 here since it is a common multiple of $p - 1$ for most p values in the range. In real applications, we are free to choose the block size to be any multiple of $p - 1$ once p, as a system parameter, is determined. These results are obtained from experiments on a Pentium 3 450 MHz Linux machine with 128 Mbytes of memory running Redhat 7.1. It is clear that the STAR code achieves throughput that is about twice that of the RS code. Note that there are jigsaw effects in the throughputs of both the EVENODD and the STAR code. This happens mainly due to the shortening technique. When the number of storage nodes is not prime, the codes are constructed using the closest larger prime number. A larger prime number means that each column (data block here) is divided into more pieces, which in turn incurs additional control overhead. As the number of information nodes increases, the overhead is then amortized, reflected by the performance ramping up after each dip. (Similarly, the performance of the RS code shows jigsaw effects too, which happens at the change of L due to the increment of total storage nodes n.) Moreover, note that the throughputs are not directly comparable between $r \ (= n - k) = 2$ and $r \ (= n - k) = 3$ (for example, the EVENODD and the STAR code) as they correspond to different reliability degrees. The results of codes with $r = 2$ are depicted only for reference purposes.

9 Conclusions

In this paper, we present the STAR code, a new coding scheme that can correct triple erasures. The STAR code extends from the EVENODD code and requires only XOR operations in its encoding and decoding operations. We prove that the STAR code is an MDS code of distance 4 and thus is optimal in terms of erasure correction capability versus data redundancy. An efficient erasure decoding algorithm for the STAR code is presented as the focus of this paper. Detailed analysis shows that the STAR code has the lowest decoding complexity among existing comparable codes. We hence believe that the STAR code is very suitable for achieving high availability in practical data storage systems.

Acknowledgments

A preliminary version of this work was presented at the Fourth Usenix Conference on File and Storage Technologies (FAST) in 2005. This work was supported in part by US National Science Foundation Grant IIS-0541527.

References

[1] G.A. Alvarez, W.A. Burkhard, and F. Christian, "Tolerating Multiple Failures in RAID Architectures with Optimal Storage and Uniform Declustering," *Proc. 24th Ann. Int'l Symp. Computer Architecture,* pp. 62-72, 1997.

[2] T.E. Anderson, D.E. Culler, and D.A. Patterson, "A Case for NOW (Networks of Workstations)," *IEEE Micro,* vol. 15, no. 1, pp. 54-64, 1995.

[3] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang, "Serverless Network File Systems," *ACM Trans. Computer Systems,* pp. 41-79, Feb. 1996.

[4] A. Bhide, E. Elnozahy, and S. Morgan, "A Highly Available Network File Server," *Proc. Usenix Winter Technical Conf.,* pp. 199-205, Jan. 1991.

[5] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures," *IEEE Trans. Computers,* vol. 44, no. 2, pp. 192-202, Feb. 1995.

[6] M. Blaum, J. Brady, J. Bruck, J. Menon, and A. Vardy, "The EVENODD Code and Its Generalization," *High Performance Mass Storage and Parallel I/O,* pp. 187-208, John Wiley & Sons, 2002.

[7] M. Blaum, J. Bruck, and A. Vardy, "MDS Array Codes with Independent Parity Symbols," *IEEE Trans. Information Theory,* vol. 42, no. 2, pp. 529-542, Mar. 1996.

[8] M. Blaum and R.M. Roth, "New Array Codes for Multiple Phased Burst Correction," *IEEE Trans. Information Theory,* vol. 39, no. 1, pp. 66-77, Jan. 1993.

[9] M. Blaum and R.M. Roth, "On Lowest-Density MDS Codes," *IEEE Trans. Information Theory,* vol. 45, no. 1, pp. 46-59, Jan. 1999.

[10] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-Based Erasure-Resilient Coding Scheme," Technical Report TR-95-048, ICSI, Berkeley, Calif., Aug. 1995.

[11] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, http://www.icsi.berkeley.edu/~luby/cauchy.tar.uu, 2007.

[12] V. Bohossian, C. Fan, P. LeMahieu, M. Riedel, L. Xu, and J. Bruck, "Computing in the RAIN: A Reliable Array of Independent Node," *IEEE Trans. Parallel and Distributed Systems,* special issue on dependable network computing, vol. 12, no. 2, pp. 99-114, Feb. 2001.

[13] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *Operating Systems Rev.,* pp. 173-186, 1999.

[14] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson, "RAID—High-Performance, Reliable Secondary Storage," *ACM Computing Surveys,* vol. 26, no. 2, pp. 145-185, 1994.

[15] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-Diagonal Parity for Double Disk Failure Correction," *Proc. Third Usenix Conf. File and Storage Technologies,* Mar.-Apr. 2004.

[16] C. Fan and J. Bruck, "The Raincore API for Clusters of Networking Elements," *IEEE Internet Computing,* vol. 5, no. 5, pp. 70-76, Sept./Oct. 2001.

[17] P.G. Farrell, "A Survey of Array Error Control Codes," *European Trans. Telecomm.,* vol. 3, no. 5, pp. 441-454, 1992.

[18] G.-L. Feng, R.H. Deng, F. Bao, and J.-C. Shen, "New Efficient MDS Array Codes for RAID Part I: Reed-Solomon-Like Codes for Tolerating Three Disk Failures," *IEEE Trans. Computers,* vol. 54, no. 9, pp. 1071-1080, Sept. 2005.

[19] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," *Proc. 19th ACM Symp. Operating Systems Principles,* pp. 29-43, Oct. 2003.

[20] G.A. Gibson and R. van Meter, "Network Attached Storage Architecture," *Comm. ACM,* vol. 43, no. 11, pp. 37-45, Nov. 2000.

[21] G.A. Gibson, D. Stodolsky, F.W. Chang, W.V. Courtright II, C.G. Demetriou, E. Ginting, M. Holland, Q. Ma, L. Neal, R.H. Patterson, J. Su, R. Youssef, and J. Zelenka, "The Scotch Parallel Storage Systems," *Proc. 40th IEEE CS Int'l Conf.,* 1995.

[22] A.V. Goldberg and P.N. Yianilos, "Towards an Archival Intermemory," *Proc. IEEE Int'l Forum Research and Technology Advances in Digital Libraries,* Apr. 1998.

[23] R.M. Goodman, R.J. McEliece, and M. Sayano, "Phased Burst Error Correcting Arrays Codes," *IEEE Trans. Information Theory,* vol. 39, pp. 684-693, 1993.

[24] J.H. Hartman and J.K. Ousterhout, "The Zebra Striped Network File System," *ACM Trans. Computer Systems,* vol. 13, no. 3, pp. 274-310, 1995.

[25] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, 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," *Proc. Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* Nov. 2000.

[26] E. Lee and C. Thekkath, "Petal: Distributed Virtual Disks," *Proc. Seventh Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* pp. 84-92, Oct. 1996.

[27] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error Correcting Codes.* North Holland, 1977.

[28] R.J. McEliece and D. Sarwate, "On Sharing Secrets and Reed-Solomon Codes," *Comm. ACM,* vol. 24, no. 9, pp. 583-584, 1981.

[29] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch, "The Sprite Network Operating System," *Computer,* vol. 21, no. 2, pp. 23-26, Feb. 1988.

[30] J.S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-Like Systems," *Software: Practice and Experience,* vol. 27, no. 9, pp. 995-1012, Jan. 1999.

[31] J.S. Plank, R.L. Collins, A.L. Buchsbaum, and M.G. Thomason, "Small Parity-Check Erasure Codes—Exploration and Observations," *Proc. Int'l Conf. Dependable Systems and Networks,* June 2005.

[32] J.S. Plank, M. Beck, and T. Moore, "Logistical Networking Research and the Network Storage Stack," *Proc. Usenix Conf. File and Storage Technologies,* work in progress report, Jan. 2002.

[33] J.S. Plank and L. Xu, "Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications," *Proc. Fifth IEEE Int'l Symp. Network Computing and Applications,* July 2006.

[34] M. Rabin, "Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance," *J. ACM,* vol. 32, no. 4, pp. 335-348, Apr. 1989.

[35] I.S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. SIAM,* vol. 8, no. 10, pp. 300-304, 1960.

[36] M. Satyanarayanan, "Scalable, Secure and Highly Available Distributed File Access," *Computer,* vol. 23, no. 5, pp. 9-21, May 1990.

[37] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, and D.C. Steere, "CODA—A Highly Available File System for a Distributed Workstation Environment," *IEEE Trans. Computers,* vol. 39, no. 4, pp. 447-459, Apr. 1990.

[38] A. Shamir, "How to Share a Secret," *Comm. ACM,* pp. 612-613, Nov. 1979.

[39] *NFS: Network File System Version 3 Protocol Specification,* Sun Microsystems, Feb. 1994.

[40] M. Waldman, A.D. Rubin, and L.F. Cranor, "Publius: A Robust, Tamper-Evident, Censorship-Resistant, Web Publishing System," *Proc. Ninth Usenix Security Symp.,* pp. 59-72, http://www.cs.nyu.edu/~waldman/publius/publius.pdf, Aug. 2000.

[41] J.J. Wylie, M.W. Bigrigg, J.D. Strunk, G.R. Ganger, H. Kiliccote, and P.K. Khosla, "Survivable Information Storage Systems," *Computer,* vol. 33, no. 8, pp. 61-68, Aug. 2000.

[42] L. Xu and J. Bruck, "X-Code: MDS Array Codes with Optimal Encoding," *IEEE Trans. Information Theory,* vol. 45, no. 1, pp. 272-276, Jan. 1999.

[43] L. Xu, V. Bohossian, J. Bruck, and D. Wagner, "Low Density MDS Codes and Factors of Complete Graphs," *IEEE Trans. Information Theory,* vol. 45, no. 1, pp. 1817-1826, Nov. 1999.

[44] L. Xu, "Hydra: A Platform for Survivable and Secure Data Storage Systems," *Proc. Int'l Workshop Storage Security and Survivability,* Nov. 2005.

Cheng Huang received the BS and MS degrees in electrical engineering from Shanghai Jiao Tong University in 1997 and 2000, respectively, and the PhD degree in computer science from Washington University, St. Louis, Missouri, in 2005. He is currently a member of the Communication and Collaboration Systems Group at Microsoft Research, Redmond, Washington. His research interests include peer-to-peer applications, distributed storage systems, erasure correction codes, multimedia communications, networking, and data security. He is a member of the IEEE.

Lihao Xu received the BSc and MSc degrees in electrical engineering from Shanghai Jiao Tong University, China, in 1988 and 1991, respectively, and the PhD degree in electrical engineering from the California Institute of Technology in 1999. He has been an associate professor of computer science at Wayne State University, Detroit, Michigan, since August 2005. He was an associate professor in July 2005 and an assistant professor from September 1999 to June 2005 with the Department of Computer Science at Washington University, St. Louis, Missouri. From 1991 to 1994, he was a lecturer in the Electrical Engineering Department at Shanghai Jiao Tong University. His current research interests include distributed computing and storage systems, error-correcting codes, information theory, and data security. He is a senior member of the IEEE.