Multiple Dimensional Fault Tolerant Schemes for Crypto Stream Ciphers

Chang N. Zhang, Qian Yu and Xiao Wei Liu

Dept of Computer Science, University of Regina, Regina, SK, S4S 0A2, Canada

{zhang, yu209,liu273}@cs.uregina.ca

ABSTRACT

To enhance the security and reliability of the widely-used stream ciphers, a 2-D and a 3-D mesh-knight Algorithm Based Fault Tolerant (ABFT) schemes for stream ciphers are developed which can be universally applied to RC4 and other stream ciphers. Based on the ready-made arithmetic unit in stream ciphers, the proposed 2-D ABFT scheme is able to detect and correct any simple error, and the 3-D mesh-knight ABFT scheme is capable of detecting and correcting up to three errors in an n^2-data matrix with liner computation and bandwidth overhead. The proposed schemes provide one-to-one mapping between data index and check sum group so that error can be located and recovered by easier logic and simple operations.

KEYWORDS

Crypto Stream Ciphers, RC4, Fault Tolerant, Error Detection

1. INTRODUCTION

Stream cipher is one kind of widely used symmetric ciphers. In encryption, the plaintext is combined with the primarily generated keystream in a simple manner, typically exclusive-or (XOR) operation, to get a ciphertext and vice versa [1]. When compared to block cipher, stream cipher has the advantage of simpler arithmetic unit and easier implementation due to its briefness of protocol. Therefore, it is better to implement it either on smaller footprint devices or power constrained low-end hardware platforms [2-4].

Continuing increases in ubiquitous computing along with consequently rising demand for data security and privacy, it has become a central issue to address the fault tolerance and information reliability.

In previous research, few efforts were contributed to fault tolerance designed for stream ciphers [5]. People mostly focus on fault tolerance for block ciphers and public-key encryptions [6-10] which employ complex computations that cannot be easily embedded into resource constrained applications. Y. Q. Zhong et al. proposed low-cost and high efficiency architecture of AES crypto-engine [11]. Other groups implemented a low-complexity high-speed fault tolerant parity scheme for AES SBOX [12-13]. C. N. Zhang proposed a multiple-error fault tolerant technique for RSA

DOI : 10.5121/ijnsa.2010.2303

based digital signature [14].

There are several error detection/correction schemes applied to disk array, cluster-based storage, or grid storage [15]. Reed-Solomon [16] provides an optimal Maximum Distance Separable (MDS) and arbitrarily high fault tolerance which requires operations on a finite field. Other XOR-based erasure codes such as EVENODD [17] consist of simple XOR computations but can only tolerant two errors. WEAVER Codes [18] and its upgraded version HoVer [19] has a wide range of practical options that can cover a large portion of the performance/efficiency trade-off but require proportionally more parity elements placed outside of the data array.

The algorithm based tolerant (ABFT) technique is a general method for designing efficient fault tolerant schemes based on structures of the algorithms [20-22]. Compare with other fault tolerant schemes, the ABFT makes use of the same arithmetic operations of the targeted algorithm and poses a conceptual way to better create a fault tolerant version by altering the algorithm computation so that its output contains extra information for error detection and correction. It has relatively low overhead and no additional arithmetical logic unit is required. ABFT is an appropriate candidate to be integrated in stream ciphers.

In this paper, two multiple error fault tolerant schemes based on ABFT for stream ciphers are presented. They consist of only XOR computations which are typical arithmetic units in all stream ciphers and employ the row/column as well as "knight" check sums to achieve fault tolerance within the acceptable overhead. The 2-D mesh check sum scheme is able to detect and correct any single error during computation or communication. The 3-D mesh-knight check sum scheme is flexible in handling various positions of up to three errors. By looking up the indexes of inconsistent check sums, single and multiple errors can be detected and corrected concurrently. For simplicity, we take RC4 as an example to show how our schemes function.

The rest of the paper is organized as follows. Section 2 sets RC4 stream cipher as an example to outline how our proposed schemes work. We introduce the 2-D mesh check sum scheme for single error detection and correction in section 3. Section 4 presents 3-D mesh-knight check sum scheme tolerating up to three errors. Section 5 concludes and summaries the proposed schemes.

2. CONCURRENT ABFT SCHEME FOR RC4

RC4 is the most remarkable stream cipher used in popular protocols such as Secure Socket Layer (SSL) and Wired Equivalent Privacy (WEP).

Consider a sender and a receiver, named A and B, respectively. Before communication, A and B possess the same key to run Key-Scheduling Algorithm (KSA) and get an initial 256-byte state, then recursively go through Pseudo-Random Generation Algorithm (PRGA) to yield byte keystream, $k_1, k_2, \cdots, k_n$. Suppose the plaintext is $m_1, m_2, \cdots, m_n$ (n bytes). The cipher text is obtained by $c_1 = m_1 \oplus k_1, c_2 = m_2 \oplus k_2, \cdots, c_n = m_n \oplus k_n$. The proposed ABFT scheme for RC4

works as follows, the n bytes plaintext is first XORed together to obtain the message check sum (M), $M = m_1 \oplus m_2 \oplus \cdots \oplus m_n$. Then A and B both calculate the keystream check sum denoted as (K) by $K = k_1 \oplus k_2 \oplus \cdots \oplus k_n$after the generation of keystream. In order to get the authentic check sum (P), Sender A computes $P = M \oplus K$when encryption is done, A attaches P to end of the ciphertext $c_1 \parallel c_2 \parallel \cdots \parallel c_n \parallel P$. Suppose B receives $c'_1 \parallel c'_2 \parallel \cdots \parallel c'_n \parallel P$ and decrypts the message by $m_1^* = c'_1 \oplus k_1, m_2^* = c'_2 \oplus k_2, \cdots, m_n^* = c'_n \oplus k_n$and calculate the authentic check sum (P^*), where $P^* = M^* \oplus K$. By comparing the value of P and P^* (e.g. $P \oplus P^*$), B is aware of whether error exists during transmission or computation, which is proved by Theorem 1.

Suppose that, at most, one error may occur during computation and transmission and that it is error-free when computing and transmitting message check sum (M and M^*), keystream check sum K, and authentic check sum (P and P^*).

***Theorem 1*:** There exists an i such that $m_i^* \neq m_i$, if and only if $P \neq P^*$.

***Proof*:** Firstly, suppose there exists an i, such that $m_i^* \neq m_i$ and $m_j^* = m_j$, $i \neq j$, $1 \leq j \leq n$.

Because of $M = m_1 \oplus m_2 \oplus \cdots \oplus m_n$, $M^* = m_1^* \oplus m_2^* \oplus \cdots \oplus m_n^*$, and $P = M \oplus K$, $P^* = M^* \oplus K$.

We have $P^* \neq P$.

Secondly, suppose $P \neq P^*$, because $P = M \oplus K$ and $P^* = M^* \oplus K$, we have $M = P \oplus K$ and $M^* = P^* \oplus K$. If $P \neq P^*$, then $M \neq M^*$.

Under the single error assumption, we can conclude that for some i, there exists $m_i^* \neq m_i$.

Figure 1 shows the logic diagram of the proposed check sum ABFT scheme for RC4.

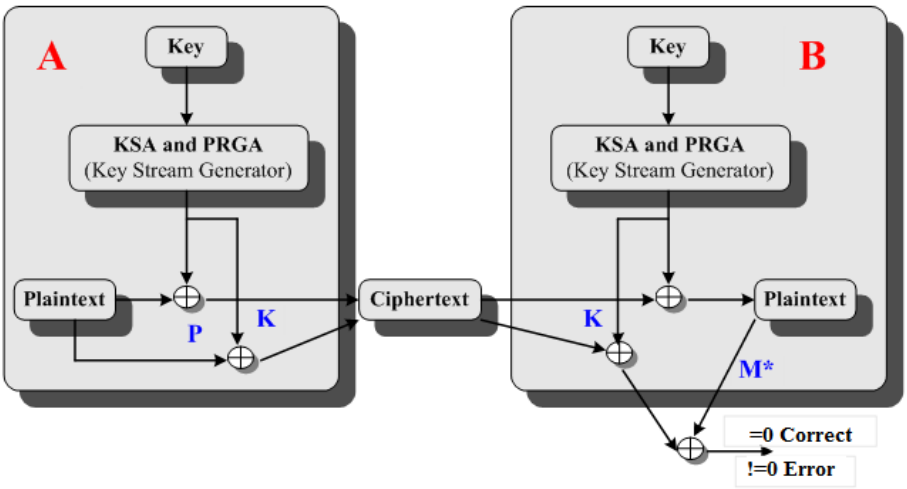


Figure 1: Logic diagram of check sum error detection scheme for RC4

3. 2-D ABFT SCHEME

To make it easy to digest, we begin with a single error case. Multiple errors will be discussed in Section 4.

In the encryption section, sender A arrays the N-byte plaintext into $m \times n$plaintext matrix ($N = m \times n$),

$$\begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{m1} & m_{m2} & \cdots & m_{mn} \end{bmatrix} \quad (1)$$

Then A constructs three $(m+1) \times (n+1)$matrices denoted by plaintext matrix M, keystream matrix K, and transmitted matrix T during the encryption process.

$$M = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} & M_1 \\ m_{21} & m_{22} & \cdots & m_{2n} & M_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ m_{m1} & m_{m2} & \cdots & m_{mn} & M_m \\ M^1 & M^2 & \cdots & M^n & M_{cc} \end{bmatrix} \quad (2)$$

Where elements of the $(n+1)th$ column M_i and the $(m+1)th$ row M^j in the plaintext matrix M are obtained by

$$M_i = \bigoplus_{j=1}^{n} m_{ij} \ (1 \le i \le m) \quad M^j = \bigoplus_{i=1}^{m} m_{ij} \ (1 \le j \le n)$$

M_{cc} is the sum of the column and row check sums, of which the index cc stands for Corner Check Sum (CC), $M_{cc} = \bigoplus_{i=1}^{m} M_i = \bigoplus_{j=1}^{n} M^j$.

$$K = \begin{bmatrix} k_{11} & k_{12} & \cdots & k_{1n} & K_1 \\ k_{21} & k_{22} & \cdots & k_{2n} & K_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ k_{m1} & k_{m2} & \cdots & k_{mn} & K_m \\ K^1 & K^2 & \cdots & K^n & K_{cc} \end{bmatrix} \quad (3)$$

The keystream matrix K is obtained using the following operations:

$$K_i = \bigoplus_{j=1}^{n} k_{ij} \ (1 \le i \le m), \quad K^j = \bigoplus_{i=1}^{m} k_{ij} \ (1 \le j \le n)$$

$$K_{cc} = \bigoplus_{i=1}^{m} K_i = \bigoplus_{j=1}^{n} K^j$$

The transmitted matrix T is obtained by

$$T = M \oplus K = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} & P_1 \\ c_{21} & c_{22} & \cdots & c_{2n} & P_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} & P_m \\ P^1 & P^2 & \cdots & P^n & P_{cc} \end{bmatrix} \quad (4)$$

where $c_{ij} = m_{ij} \oplus k_{ij}$ $(1 \le i \le m, 1 \le j \le n)$.

P_i and P^j are row and column check sums given by

$P_i = M_i \oplus K_i$ $P^j = M^j \oplus K^j$ $P_{cc} = \overset{m}{\underset{i=1}{\oplus}} P_i = \overset{n}{\underset{j=1}{\oplus}} P^j$.

During the decryption process, suppose the transmitted matrix received by B is

$$T' = \begin{bmatrix} c'_{11} & c'_{12} & \cdots & c'_{1n} & P_1 \\ c'_{21} & c'_{22} & \cdots & c'_{2n} & P_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c'_{m1} & c'_{m2} & \cdots & c'_{mn} & P_m \\ P^1 & P^2 & \cdots & P^n & P_{cc} \end{bmatrix} \quad (5)$$

As mentioned before, B needs to build up Key matrix K synchronously with the decryption according to Equation 3.

$$K = \begin{bmatrix} k_{11} & k_{12} & \cdots & k_{1n} & K_1 \\ k_{21} & k_{22} & \cdots & k_{2n} & K_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ k_{m1} & k_{m2} & \cdots & k_{mn} & K_m \\ K^1 & K^2 & \cdots & K^n & K_{cc} \end{bmatrix} \quad (6)$$

where $K_i = \overset{n}{\underset{j=1}{\oplus}} k_{ij}$, $K^j = \overset{m}{\underset{i=1}{\oplus}} k_{ij}$, $K_{cc} = \overset{m}{\underset{i=1}{\oplus}} K_i = \overset{n}{\underset{j=1}{\oplus}} K^j$ $(1 \le i \le m, 1 \le j \le n)$.

Based on RC4, B decrypts the transmitted matrix T' which gives the following decrypted plaintext matrix M^*

$$M^* = \begin{bmatrix} m^*_{11} & m^*_{12} & \cdots & m^*_{1n} & M^*_1 \\ m^*_{21} & m^*_{22} & \cdots & m^*_{2n} & M^*_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ m^*_{m1} & m^*_{m2} & \cdots & m^*_{mn} & M^*_m \\ M^{*1} & M^{*2} & \cdots & M^{*n} & M^*_{cc} \end{bmatrix} \quad (7)$$

where $m^*_{ij} = k_{ij} \oplus c'_{ij}$ $M^*_i = P_i \oplus K_i (1 \le i \le m)$ $M^{*j} = P^j \oplus K^j (1 \le j \le n)$ $M^*_{cc} = K_{cc} \oplus P_{cc}$.

Theorem 2: There is $m^*_{xy} \neq m_{xy}$ for some x and y ($1 \le x \le m, 1 \le y \le n$). If and only if the following conditions hold:

Row condition: $M^*_x \neq \overset{n}{\underset{j=1}{\oplus}} m^*_{xj}$

Column condition: $M^{*y} \neq \overset{m}{\underset{i=1}{\oplus}} m^*_{iy}$

Proof: If there is an error

$m^*_{xy} \neq m_{xy}$ ($1 \le x \le m$, $1 \le y \le n$), then we know that

$$M^*_x = P_x \oplus K_x = (\overset{n}{\underset{j=1}{\oplus}} p_{xj}) \oplus (\overset{n}{\underset{j=1}{\oplus}} k_{xj}) = \overset{n}{\underset{j=1}{\oplus}} m_{xj} \neq \overset{n}{\underset{j=1}{\oplus}} m^*_{xj} \quad M^{*y} = P^y \oplus K^y = (\overset{m}{\underset{i=1}{\oplus}} p_{iy}) \oplus (\overset{m}{\underset{i=1}{\oplus}} k_{iy}) = \overset{m}{\underset{i=1}{\oplus}} m_{iy} \neq \overset{m}{\underset{i=1}{\oplus}} m^*_{iy}$$

If there are x and y such that,

$M^*_x \neq \overset{n}{\underset{j=1}{\oplus}} m^*_{xj}$ and $M^{*y} \neq \overset{m}{\underset{i=1}{\oplus}} m^*_{iy}$

Then $\overset{n}{\underset{j=1}{\oplus}} m^*_{xy} \neq \overset{n}{\underset{j=1}{\oplus}} m_{xy}$ and $\overset{n}{\underset{i=1}{\oplus}} m^*_{xy} \neq \overset{n}{\underset{i=1}{\oplus}} m_{xy}$.

Under the single error assumption, it is $m^*_{xy} \neq m_{xy}$.

From Theorem 2, we come to a conclusion that under the single error assumption, not only the existence but also the exact position of the error can be precisely indicated by row and column conditions. Hence, once the single error is detected, it can be corrected according to the following fact:

Corollary 1: If $m^*_{xy} \neq m_{xy}$ ($1 \le x \le m, 1 \le y \le n$), the error can be corrected by computing either of the following two formulas:

Row direction: $m_{xy} = M^*_x \oplus (\overset{n}{\underset{j=1}{\oplus}} m^*_{xj}) \oplus m^*_{xy}$

Column direction: $m_{xy} = M^{*y} \oplus (\overset{m}{\underset{i=1}{\oplus}} m^*_{iy}) \oplus m^*_{xy}$

Note that according to Theorem 1, if $P_{cc} = P^*_{cc}$, then there is no error during the processes of encryption, decryption and data transmission. If $P_{cc} \neq P^*_{cc}$, then there is an error which can be located and corrected according to Theorem 2 and Corollary 1.

4. 3-D MESH-KNIGHT ABFT SCHEME

Under the single error assumption, an error can be found by both row and column indexes of the inconsistent check sums. In essence, the prerequisite for single error correction is to find the error's two indexes: i and j. In order to solve the indexes, we consider the row direction and column direction as a 2-D i-j coordinate. Under this geometric assumption two check sum index equations can be constructed the solution of this equation set is the indexes that indicate the exact error location.

For multiple errors, more information and more equations are needed to specify multiple errors' indexes. Take, for instance, a case with two errors where each error has two indexes (i, j), so there are four indexes in total, which means four different equations have to be constructed. However, in some cases, the row and column directions are not sufficient to locate multiple errors. For example, as shown in Figure 2, there are two errors located at (i_1, j) and (i_2, j), where $i_1 \neq i_2$. If both bit-errors are coincidently distributed in the same bit of a byte (e.g. "00110001" and "10101110" both have an error "1" in the 6th bit which should be "0"), as a result, two error bits counteract and erase the evidence of inconsistent bit in check sum according to the property of parity, making no change in check sums along the row direction. In other words, the check sums cannot find the exact locations of these two errors.

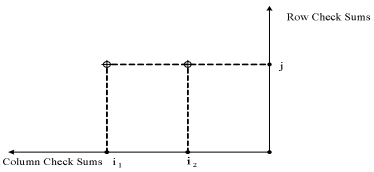


Figure 2: Two errors locate in the same row

4.1. The Knight Check Sum

The eight-queen puzzle greatly inspires us when finding the third suitable coordinate. This problem is originally proposed to find solutions for placing eight chess queens on an 8×8 chessboard such that no two queens share the same row, column or diagonal. Following the standard chess queen moves, none of them is able to capture others. Over the years, eight-queen has been extended to N-queen puzzle and several algorithms and solutions have been developed.

Based on the idea that no queens are in the same row, column and diagonal, we proposed a novel check sum coordinate which resembles the rules of chess knight.

Let us take a $n \times n$ matrix D as an example to define the knight check sum, denoted as KCS.

$$D = \begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,n-1} & d_{1,n} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,n-1} & d_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ d_{n,1} & d_{n,2} & \cdots & d_{n,n-1} & d_{n,n} \end{bmatrix} \qquad (8)$$

d_{ij} $(1 \le i, j \le n)$ is a one-byte data—the message passing between sender and receiver. Depending on the following rules, we could collect data and compute KCS:

if n is odd number: $$K = \sum_{j=1}^{n} d_{(k+2\times(j-1)) \bmod (n+1),\, j}$$

if n is even number: $$K = \sum_{j=1}^{n} d_{(k+3\times(j-1)) \bmod (n+1),\, j}$$

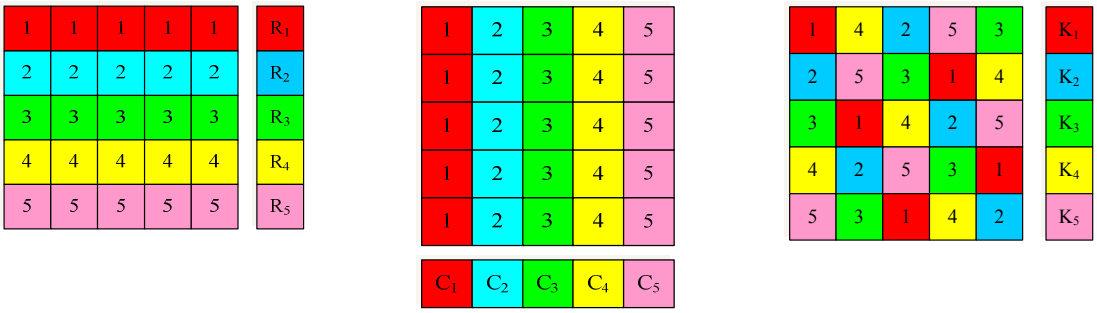
where $1 \le i, j, k \le n$ and symbol K stands for KCS. In each round searching the element in a certain knight group k, the index i of the element $\left(k + 2(j-1)\right)$ or $(k + 3(j-1))$must be iteratively decreased every iteration so that at the end of each round, the index i must be limited to less than or equal to n. We hereby note that the maximum value of the index i during the computation is less than 2n, so that we can implement MOD operation by only subtracting. Thus, KCS can be computed by the following pseudo-code:

```
for ( int i = 1; i <= n; i ++ ) {
        K[i] = 0;
        x = i;
        for ( int j = 1; j <= n; j ++ ) {
                y = j;
                if n is odd, then x += 2;
                if n is even, then x += 3;
                if( x > n )  x -= n;
                K[i] += D[x][y];
        }
}
return K;
```

Following is an example of a 5×5 matrix. The data with the same number specified in the following matrix represents data belonging to the same knight group:

$$\begin{bmatrix} 1 & 4 & 2 & 5 & 3 \\ 2 & 5 & 3 & 1 & 4 \\ 3 & 1 & 4 & 2 & 5 \\ 4 & 2 & 5 & 3 & 1 \\ 5 & 3 & 1 & 4 & 2 \end{bmatrix}$$

Figure 3 shows the differences between mesh and knight coordinates. The same color indicates the same check sum group.



a. Row check sum b. Column check sum c. Knight check sum

Figure 3: Mesh and knight check sums

The knight check sum is organized in a non-linear manner, resulting in a collection of data that are not in the same row, column or diagonal.

By applying every 3-D mesh-knight check sum to the $n \times n$ data matrix, more equations can be constructed to provide sufficient information about error locations. When inconsistent check sum P_x, P^y, K_i are detected, they can help to accurately detect up to 3 errors.

4.2. Multiple Errors Correction in the 3-D Mesh-Knight Coordinate

In this section, we assume that up to three errors may occur during computation and/or communication. We categorize all the possible relative positions of one, two and three errors in the Table 1. As indicated in Table 1, just by the information of numbers of inconsistent check sums, the number of errors as well as geometric relative positions among these errors can be classified into each specific case. After obtaining the indexes of errors, corresponding error correction equations introduced in Section 3 can be applied to recover any these error bytes.

In Table 1, two functions are involved in the error positioning process. Knowing the knight check sum index and one of the mesh check sum indexes, the function entitled “L” produces the other mesh check sum index. See the following pseudo-code:

Input: k, i or j, Output: j or i

L(k, i)

L(k, j)

x = k;

for (int count = 1; count <= n; count ++) {
y = count;

if n is odd, then x += 2;

if n is even, then x += 3;

if (x > n) x -= n;

if we know i ask for j & i == y, then return x;

if we know j ask for i & j == x, then return y;}

Table 1: Single error and multiple errors detection and correction cases (* Number, $^{+}$ Inconsistent)

Errors*	Same Row*	Same Column*	Same Knight*	Row SC^{+}	Column SC^{+}	Knight SC^{+}	Index of Errors
1	N/A	N/A	N/A	i1	j1	k1	(i1, j1)
2	2	0	0	N/A	j1, j2	k1, k2	$(i1_{=L(k1,j1)}, j1), (i2_{=L(k2,j2)}, j2)$
	0	2	0	i1, i2	N/A	k1, k2	$(i1, j1_{=L(k1,i1))},(i2, j2_{=L(k2,i2)})$
	0	0	2	i1, i2	j1, j2	N/A	(i1, j2),(i2, j2)
	0	0	0	i1, i2	j1, j2	k1, k2	(i1, j1),(i2, j2)
3	2	0	0	i3	j1, j2, j3	k1, k2, k3	$(i1_{=L(k1,j1)}, j1), (i2_{=L(k2,j2)}, j2),(i3, j3)$
	3	0	0	same as the above case			
	0	2	0	i1, i2, i3	j3	k1, k2, k3	$(i1, j1_{=L(k1,i1))},(i2, j2_{=L(k2,i2))},(i3, j3)$
	0	3	0	same as the above case			
	0	0	2	i1, i2, i3	j1, j2, j3	k1, k3	(i1, j1), (i2, j2), (i3, j3)
	0	0	0	i1, i2, i3	j1, j2, j3	k1, k2, k3	(i1, j1), (i2, j2), (i3, j3)
	2	2	0	i2	j3	k1, k2, k3	$(i2, j2_{=L(k2,i2)}), (i3_{=L(k3,j3)}, j3),$ $(i1_{=L(k1,\ i3L(k3,j3))}, j1_{=L(k3,\ j2L(k2,i2)))}$
	2	0	2	i2	j1, j2, j3	k3	$(i1_{=L(K(i1,\ j2),j1)}, j1), (i2, j2),(i3_{=L(k3,\ i3)}, j3)$
	0	2	2	i1, i2, i3	j2	k3	$(i1, j1_{=L(K(i2,j2),i1))}), (i2, j2),(i3, j3_{=L(k3,\ i3)})$
	2	2	2	i3	j2	k1	$(i1_{=L(K(i3,j2)}, j1_{=RT(i3,j2,k1))}), (i2_{=i1}, j2), (i3, j3_{=j3})$
	0	0	3	i1, i2, i3	j1, j2, j3	N/A	(i1, j2), (i2, j2),(i3, j3)

Figure 4 shows that the L function has to be applied to two known check sums and compute the other overlapping check sum where * denotes as the missing check sum. As shown in this figure, two of three errors line in the same row and other one are isolated to these two errors. The two errors overlap each other in the row dimension so that only one row check sum is obtained and their respective positions cannot be determined due to the lack of check sum information. Hence, to compute the lost check sum information based on the known integrated check sums, each overlapping errors has to be set through the L function: the inputs of L function is column and knight check sum indexes which are integrated with no overlapping check sums. By computing

the missing check sum for these two errors, their two column check sum indexes are obtained. As a result, we can distinguish two overlapping errors.

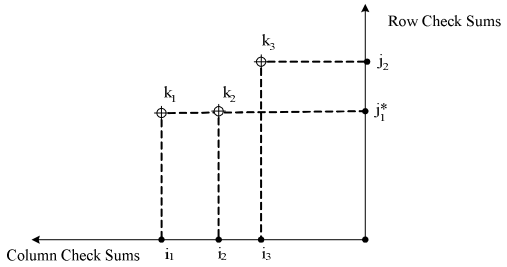


Figure 4: An example for L function

Function K is to compute the KCS index when two mesh check sum indexes are given. The next pseudo-code specifies its detail.

```
  Input: i, j, Output: k
K( i, j )
x = j;
for ( int index = i; index > 0; index -- ) {
        if n is odd, then x -= 2;
        if n is even, then x -= 3;
        if( x < 0 )          x += n; }
return x;
```

Figure 5 represents one of the errors positions where K function should be applied. There are three errors, two of them are located in the same row and two of them are in the same knight group. Under this situation, only one non overlapping check sum can be got. The other two dimensional check sums cannot be located due to the overlapping errors. K function treats the row check sum and column check sum of error (i2, j2) as the input, and comes out the KCS of this error. In this way, the KCS number of error (i2, j2) and (i3, j1) has been computed. Then another unknown check sum is the row check sum j1 for error (i1, j1) and (i3, j1). KSC of (i1, j1) and its column check sum is known, so that L function can be applied to compute the row check sum index, which is also the row check sum index of error (i3, j1). Moreover, this row index can also be computed by applying L function to (i3, j1) whose KCS and column check sums are known or computed.

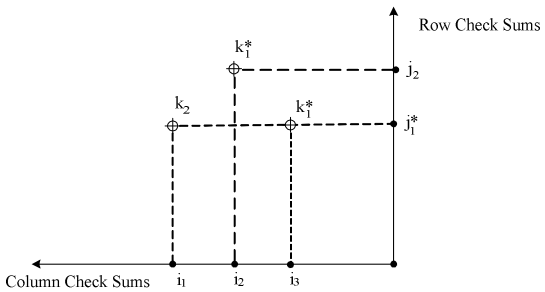


Figure 5: An example for K function

There is one particular function called Right Triangle function which may apply when three errors form a right triangle. In this case, only one knight, one row and one column check sum are known, so that we have to take advantage of the only KCS to search for the errors located on the right angle which is shown in the pseudo-code below:

```
Input:    i, j, and k
Output: i1, j1, i2, j2, i3 and j3
x = k;
for ( int count = 1; count <= n; count ++ ) {
        y = count;
 if n is odd, then x += 2;
 if n is even, then x += 3;
        if ( x > n )          x -= n;
        if ( K( x, j ) == K( i, y ) ) {
                i1 = x;
                j1 = y;
                break; }
        i2 = i1;
        j2 = j;
        i3 = i
        j3 = j1;
        return i1, j1, i2, j2, i3, j3; }
```

The particular right triangle case is shown in Figure 6. It seems that only one non overlapping check sum is appeared in each direction. The above function is used to tackle this problem. Input the known three check sums i2, j1, k2, and output i1, j2, k1.

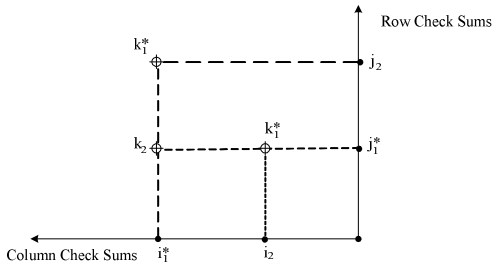


Figure 6: Example for right triangle function

Because the mapping between error and check sum are no longer one-to-one when errors are up to 4. In practical applications, errors can possibly be mixed and integrated cases, so that combination solutions could be analyzed and then applied to tolerant more errors. Generally, three arbitrary errors are guaranteed to be detected and corrected in a $n \times n$ data matrix.

For a $n \times n$ matrix, this scheme involves 3n more bytes as the check sums. Therefore, we can get to the conclusion that larger size gains more efficiency and the size of matrix should be more than three otherwise it is not worthy to apply.

In conclusion, suppose N = n^2-byte messages are arrayed into a $n \times n$ matrix, the proposed 3-D mesh-knight check sum ABFT scheme brings in overhead of $O(N^{\frac{1}{2}})$. The proposed scheme can efficiently detect and correct up to three errors with arbitrary positions. The redundancy is mainly introduced by the storage of previous computation results and the delay of data processing. The proposed scheme has general applicability to tackle most error cases which might happen during computation or transmission. In addition, the scheme is consistent with the performance evaluation of check sum based ABFT [23].

5. CONCLUSIONS

In this paper, algorithm based 2-D and 3-D fault tolerant schemes for stream ciphers are presented. By utilizing the ready-made arithmetic unit in RC4, they are compatible in existing system. The two schemes can conform to different interests for practical. The row and column check sum denoted as 2-D mesh check sum, as well as 2-D plus the knight check sum achieve accurate detection and correction of single error and up to three arbitrary errors. The innovations of the scheme are concluded below:

1. Single or up to three arbitrary errors can be detected and corrected.

2. Errors can occur either in computation process or communication channel.

3. The 3-D coordinate constructs one-to-one mapping between index and check sum, so that error can be located and recovered by easier logic and simpler operation.

4. The scheme has general applicability, is flexible in tackling most possible cases, and is convenient to be implemented both on software and hardware.

5. The overhead is within an acceptable range. For a given $n \times n$ matrix, the bandwidth overhead of two schemes are $2n$ and $3n$ respectively.

6. The proposed schemes are applicable to other stream ciphers and massive memories with fault tolerant capability.

REFERENCES

[1] William Stallings. Cryptography and Network Security 4th Edition. Pearson Education Inc, Upper Saddle River, New Jersey, 2006.

[2] Matt J.B. Robshaw, “Stream Ciphers Technical Report”, TR-701, ver. 2.0, RSA Labs, 1995.

[3] L. Batina, J. Lano, N. Mentens, S.B. Ors, B. Preneel, I. Verbauwhede. Energy, Performance, Area Versus Security Trade-offs for Stream Ciphers. The State of the Art of Stream Ciphers: Workshop Record, Belgium, October 2004, 2004, pp 302–310.

[4] A. Biryukov, A. Shamir. Cryptanalytic Time/ Memory/ Data Tradeoffs for Stream Ciphers. Asiacrypt, Springer Verlag, 2000, pp 1–13.

[5] J.J. Hoch, A. Shamir. Fault Analysis of Stream Ciphers. CHES, Springer Verlag, 2004, pp 240–253.

[6] S. Murphy, Matt J.B. Robshaw. Key-Dependent S-Boxes and Differential Cryptanalysis Designs. Codes and Cryptography, Vol. 27, 2002, pp 229–255.

[7] W.A. Jackson, S. Murphy. Projective aspects of the AES inversion. Des Codes Crypt, Vol.43, 2007, pp 167–179.

[8] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri. Error Analysis and Detection Procedures for Hardware Implementation of the Advanced Encryption Standard. IEEE Transactions on Computers, Vol. 52, No. 4, April 2003, pp 492-505.

[9] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri. An Efficient Hardware-Based Fault Diagnosis Scheme for AES Performances and Cost. Proceedings DFT'04, Oct 2004, pp 130-138.

[10] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri. Detecting and locating faults in VLSI implementations of the Advanced Encryption Standard. Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03), Nov 2003, pp 105-113.

[11] Y.Q. Zhong, J.M. Wang, Z.F. Zhao, D.Y. Yu, L. Li. A low-cost and high efficiency architecture of AES crypto-engine. Second International Conference on Comm. and Networking, Shanghai, Aug 2007.

[12] Xinmiao Zhang, K.K. Parhi. High-speed VLSI architectures for the AES algorithm. Very Large Scale Integration (VLSI) Systems, IEEE Transactions, Vol. 12, No. 9, Sep 2004, pp 957 – 967.

[13] Aaron E. Cohen, Keshab K. Parhi. A low-complexity High-Speed Fault-Tolerant Parity Implementation for the AES SBOX. Very Large Scale Integration Systems, IEEE Transactions, 2002.

[14] C.N. Zhang. An Integrated Approach for Fault Tolerance and Digital Signature in RSA. IEE Proc. Comp. Digit. Tech., Vol. 146, No. 3, May 1999, pp 151-159.

[15] Jay J. Wylie, Ram Swaminathan. Determining fault tolerance of XOR-based erasure codes efficiently. Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007, pp 206-215.

[16] I.S.Reed, G.Solomon. Poynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics, Vol. 8, 1960, pp 300-304.

[17] Blaum, M. Brady, J. Bruck, J. Menon. EVENODD: an optimal scheme for tolerating double disk failures in RAID architectures. Computer Architecture, Proceedings the 21st Annual International Symposium, 1994, pp 245-254.

[18] James Lee Hafner. WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage System. 4th USENIX Conference on File and Storage Technologies, 2005, pp 211-221.

[19] James Lee Hafner. HoVer Erasure Codes for Disk Arrays. Proceedings of the International Conference on Dependable Systems and Networks, 2006, pp 217-226.

[20] PATEL, J.H., and FUNG, L.Y. Concurrent error detection in ALU's by recomputing with shifted operands, IEEE Trans. Comput., C-31 (1982) pp. 589-595

[21] GULATI, R.K., and REDDY, S.M. Concurrent error detection in VLSI array structures. Proc. IEEE Internet, Conf. on Computer Design, 1986, pp. 488-491

[22] KUHN, R.H. Yield enchancement by fault-tolerant systolic arrays in VLSI and modern signal processing (Prentice-Hall, 1985), pp.178-184

[23] Ahmad A. Al-Yamani, Nahmsuk Oh, Edward J. McCluckey. Performance Evaluation of Checksum-Based ABFT. Proceeding of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2001, pp 461.

AUTHORS

Chang N. Zhang has been at the University of Regina since 1990 where he is currently a Professor of the Department of Computer Science, and Adjunct Scientist with Telecommunication Research Labs (TRLabs). He received his B.S. in Applied Math from Shanghai University, and his Ph.D. in Computer Science and Engineering from Southern Methodist University.

Qian Yu is currently a Ph.D. candidate in the Department of Computer Science at the University of Regina. He received his Master of Science degree in Computer Science from University of Regina in 2007 and his Bachelor of Engineering degree in Computer Science and Technology from National University of Defense Technology, China in 2003.

Liu, Xiao Wei was graduated from University of Regina, Computer Science Department. During her post-graduate period, she was supervised by Dr. Chang N. Zhang, funded by Telecommunication Research Lab (TRLabs), and her major research interest was fault tolerance in Cryptography. She is currently working for IBM China Development Lab and mainly focuses on cloud computing related area.