Differential Fault Analysis of LEX

Jianyong Huang, Willy Susilo and Jennifer Seberry

Centre for Computer and Information Security Research,
School of Computer Science and Software Engineering,
University of Wollongong, Wollongong NSW 2522, Australia
{jyh33, wsusilo, jennie}@uow.edu.au

Abstract. LEX is a stream cipher based on the round transformation of the AES block cipher, and it was selected for the final phase evaluation of the eSTREAM project. LEX is 2.5 times faster than AES both in software and in hardware. In this paper, we present a differential fault attack on LEX. The fault model assumes that the attacker is able to flip a random bit of the internal state of the cipher but cannot control the exact location of the induced fault. Our attack requires 40 faults, and recovers the secret key with 2^{16} operations.

Keywords: LEX, stream cipher, AES, cryptanalysis, differential fault analysis.

1 Introduction

The aim of the eSTREAM project was to stimulate work in the area of stream ciphers. The call for primitives was released in 2004 and 34 proposals were submitted to the project. The competition was completed in 2008 and seven ciphers were selected in the eSTREAM portfolio.

LEX [4] was one of the candidates of the eSTREAM final phase evaluation. LEX is based on a design principle known as a leak extraction from a block cipher. In this construction, the output key stream is extracted from parts of the internal state of a block cipher at certain rounds (possibly after passing an additional filter function). The extracted parts of the internal state need to be selected carefully because leaking the wrong part of the state may endanger the security of the cipher. The underlying block cipher of LEX is AES [7], and the key stream is generated by extracting 32 bits from each round of AES in the Output Feedback (OFB) mode. LEX has a simple and elegant structure and is fast in software and hardware (2.5 times faster than AES).

There are two types of attacks against the security of cryptosystems: direct attacks and indirect attacks. In direct attacks, the cryptanalyst targets to exploit any theoretical weakness in the algorithm used in the cipher, and examples of direct attacks include differential cryptanalysis [1] and linear cryptanalysis [13]. In indirect attacks, the attacker tries to obtain information from the physical implementation of a cryptosystem, and aims to break the system by making use of the gained information. Instances of indirect attacks include timing attacks [11],

power attacks [12] and fault attacks [6]. The concept of fault analysis was first introduced by Boneh, DeMillo and Lipton [6] in 1996, and the attack was used to target certain implementations of RSA and Rabin signatures by taking advantage of hardware faults. Fault analysis was also used to attack block ciphers such as DES [2]. It was showed in [10] that a fault attack is a powerful cryptanalytic tool which can be employed to attack stream ciphers.

After LEX was submitted to the eSTREAM project, a few attacks against this cipher have been proposed. The resynchronization of LEX is vulnerable to a slide attack [15], and the attack needs $2^{60.8}$ random IVs and 20,000 keystream bytes generated from each IV. A generic attack, which requires $2^{65.7}$ resynchronizations, was published in [9]. A differential attack [8] can recover the secret key of LEX in time of 2^{112} operations by using $2^{36.3}$ bytes of key stream produced by the same key (possibly under many different IVs). A related key attack was shown in [14], and the attack requires $2^{54.3}$ keystream bytes and can recover the secret key with 2^{102} operations. These four proposed attacks on LEX belong to direct attacks.

In this paper, we describe a differential fault attack on LEX. The fault model assumes that the attacker can flip a random bit of the internal state of the cipher and she can carry out the operation many times for the same internal state. However, the attacker is not supposed to know the exact location of the flipped bit. The proposed attack requires 40 faults and recovers the secret key of LEX with 2^{16} operations.

This paper is organized as follows. Section 2 describes the AES block cipher and the LEX stream cipher. Section 3 provides the details of the differential fault analysis of LEX. The paper is concluded in Section 4.

2 Descriptions of AES and LEX

We briefly describe the AES block cipher in Section 2.1. The LEX stream cipher is described in Section 2.2. We provide the notations used throughout this paper in Section 2.3.

2.1 The AES Block Cipher

The Advanced Encryption Standard [7] is a block cipher with a 128-bit block length and supports key lengths of 128, 192 or 256 bits. For encryption, the input is a plaintext and a secret key, and the output is a ciphertext. The plaintext is first copied to a four-by-four array of bytes, which is called the state. After an initial round key addition, the state array is transformed by performing a round function 10, 12, or 14 times (for 128-bit, 192-bit or 256-bit keys respectively), and the final state is the ciphertext. Each round of AES consists of the following four transformations (the final round does not include the MixColumns operation).

– SubBytes (SB). It is a non-linear byte substitution that operates independently on each byte of the state using a substitution table.

- ShiftRows (SR). The bytes of the state are cyclically shifted over different numbers of bytes. Row i is shifted to the left i byte cyclicly, $0 \leq i \leq 3$.
- MixColumns (MC). It operates on the state column-by-column. The columns are treated as polynomials and multiplied by a constant 4×4 matrix over $GF(2^8)$.
- AddRoundKey (ARK). A round key is added to the state by a simple bitwise exclusive or (XOR) operation.

The AES round keys are derived from the cipher key by employing the key schedule. The cipher key is first expanded into an expanded key. The round keys are selected from this expanded key in the following way: the first round key consists of the first Nb (the number of columns comprising the state) words, the second one of the following Nb words, and so on. The expanded key is an array of 4-byte words and is denoted by $W[Nb*(Nr+1)]$, where Nr is the number of rounds. The first Nk (number of 32-bit words comprising the cipher key) words contain the cipher key. All other words are defined recursively in terms of words with smaller indices. The pseudocode for key expansion for 128-bit cipher keys is shown below, where Key is the cipher key, $SW(x)$ applies the substitution operation to each byte of the word, $RW(x)$ cyclically shifts the word to the left 8 bits, and $Rcon$ is an array of predefined constants.

```
for(i = 0; i < Nk; i++)
    W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);

for(i = Nk; i < Nb * (Nr + 1); i++)
    temp = W[i-1];
    if (i % Nk == 0)
    temp = SW(RW(temp)) ^ Rcon[i/Nk];
    W[i] = W[i-Nk] ^ temp;
```

2.2 The LEX Stream Cipher

Two versions of LEX, the original version [3] and the tweaked version [5], were submitted to the eSTREAM project. We only provide the description of the tweaked version in this paper. LEX uses the building blocks of the AES block cipher. First, a standard AES key schedule for a secret 128-bit key K is performed. Then, a given 128-bit IV is encrypted by a single AES encryption, $S = AES_K(IV)$. The 128-bit result S and the secret key K comprise a 256-bit secret state of the stream cipher. Under the key K, S is repeatedly encrypted in the OFB mode. In each round of the encryption, 32 bits are extracted from the intermediate state to form the key stream. The positions of the extracted 32 bits are shown in Fig. 1. The IV is replaced after 500 encryptions and the secret key is changed after 2^{32} different IVs are used.

2.3 Notations

An AES intermediate state, as well as an AES round key, is represented as a four-by-four array of bytes. A byte of an intermediate state is written as $b_{i,j}$,

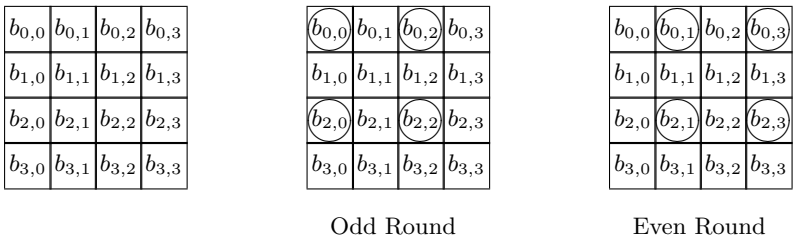

Fig. 1. The positions of the leak in the even and odd rounds

the corresponding faulty byte is denoted by $b'_{i,j}$, and the difference of $b_{i,j}$ and $b'_{i,j}$ is represented as $\Delta b_{i,j}$, $0 \leq i, j \leq 3$. A key byte of Round x is denoted by $K^x_{i,j}$, $0 \leq i, j \leq 3$. The symbol ? stands for an unknown byte. In all figures, a keystream byte of LEX is surrounded by a circle.

3 The Differential Fault Analysis

The fault model used in this paper assumes that the attacker can flip a random bit of the internal state of the cipher during the keystream generation and obtain the corresponding faulty key stream. Another assumption is that the attacker can reset the state back to its original status and repeat the fault injection process many times. However, the attacker is not supposed to know the exact location of the injected fault. Based on the fault model, we first describe a method to determine the fault position and then we show that the attacker can recover the secret key of LEX by analyzing the original and faulty key stream.

3.1 The Fault Position Determination Method

Since the attacker does not know the exact fault position in our fault model, we need to determine the fault location first. The idea is that we can find out the fault position by observing the changes of the key stream after the fault is injected. We show that we can identify into which byte the random bit fault is injected. We divide all possible cases into two categories. In the first category, the fault is injected into the state after the MC or ARK transformation, and in the second category, the fault is injected into the state after the SB or SR transformation.

We use Fig. 2 to describe the position determination method. Suppose the three-round diagram starts with an odd round, i.e., i is an odd number (we can also do the analysis by using the same idea if i is an even number). The keystream bytes are $g_{0,0}$, $g_{0,2}$, $g_{2,0}$ and $g_{2,2}$ in Round i, $l_{0,1}$, $l_{0,3}$, $l_{2,1}$ and $l_{2,3}$ in Round $i+1$ and $s_{0,0}$, $s_{0,2}$, $s_{2,0}$ and $s_{2,2}$ in Round $i+2$.

1. Category 1. The fault is injected into the state after the MC or ARK transformation. We only focus on cases where the fault is injected into the state after the MC transformation, and we can use the same idea to analyze cases

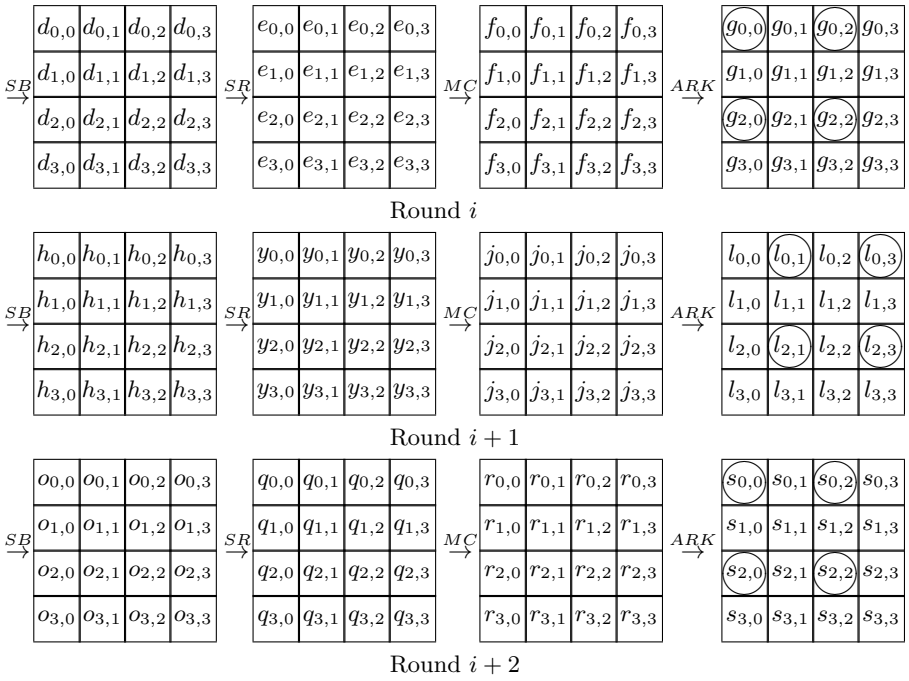

Fig. 2. The three-round diagram

where the fault is injected into the state after the ARK operation. Suppose the fault is induced to the state after the MC transformation in Round i. We further divide all possible cases of this category into four groups and each group contains four bytes.

- Group 1. This group includes four bytes: $f_{0,0}$, $f_{2,0}$, $f_{0,2}$ and $f_{2,2}$.
- Group 2. This group has four bytes: $f_{1,0}$, $f_{2,1}$, $f_{3,2}$ and $f_{0,3}$.
- Group 3. This group comprises four bytes: $f_{3,0}$, $f_{0,1}$, $f_{1,2}$ and $f_{2,3}$.
- Group 4. This group consists of four bytes: $f_{1,1}$, $f_{3,1}$, $f_{1,3}$ and $f_{3,3}$.

The relation between the fault position and the changes of the keystream bytes is summarized in Table 1. For example, if a fault is induced to $f_{0,0}$ (see the first entry of Group 1 in Table 1) in Fig. 2, we can see the change of $g_{0,0}$ from the key stream in Round i. Since $g_{0,0}$ is changed, $h_{0,0}$, $y_{0,0}$, $j_{0,0}$, $j_{1,0}$, $j_{2,0}$, $j_{3,0}$, $l_{0,0}$, $l_{1,0}$, $l_{2,0}$, $l_{3,0}$ are changed in Round $i+1$. However, we cannot see the changes from the key stream in Round $i+1$ because the keystream bytes of this round are $l_{0,1}$, $l_{0,3}$, $l_{2,1}$ and $l_{2,3}$. In Round $i+2$, all 16 bytes are changed after the MC transformation and we can see that all four keystream bytes $s_{0,0}$, $s_{0,2}$, $s_{2,0}$ and $s_{2,2}$ are changed. The difference among Group 1, Group 2, Group 3 and Group 4 is that the changes of the keystream bytes in Round i and Round $i+1$ take place at different positions.

2. Category 2. The fault is injected into the state after the SB or SR transformation. We only concentrate on cases where the fault is injected into the

Table 1. Four groups of fault positions

Group 1			
Fault	i	$i+1$	$i+2$
$f_{0,0}$	$g_{0,0}$	None	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{2,0}$	$g_{2,0}$	None	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{0,2}$	$g_{0,2}$	None	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{2,2}$	$g_{2,2}$	None	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$

Group 2			
Fault	i	$i+1$	$i+2$
$f_{1,0}$	None	$l_{0,3}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{2,1}$	None	$l_{0,3}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{3,2}$	None	$l_{0,3}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{0,3}$	None	$l_{0,3}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$

Group 3			
Fault	i	$i+1$	$i+2$
$f_{3,0}$	None	$l_{0,1}$, $l_{2,1}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{0,1}$	None	$l_{0,1}$, $l_{2,1}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{1,2}$	None	$l_{0,1}$, $l_{2,1}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{2,3}$	None	$l_{0,1}$, $l_{2,1}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$

Group 4			
Fault	i	$i+1$	$i+2$
$f_{1,1}$	None	None	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{3,1}$	None	None	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{1,3}$	None	None	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$f_{3,3}$	None	None	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$

state after the SR transformation, and we can use the same idea to analyze cases where the fault is injected into the state after the SB transformation. Assume that the fault is induced to the state after the SR transformation in Round i. We split all possible cases of this category into three groups.

Table 2. Three groups of fault positions

Group 5			
Fault	i	$i+1$	$i+2$
$e_{0,0}$	$g_{0,0}$, $g_{2,0}$	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{1,0}$	$g_{0,0}$, $g_{2,0}$	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{2,0}$	$g_{0,0}$, $g_{2,0}$	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{3,0}$	$g_{0,0}$, $g_{2,0}$	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$

Group 6			
Fault	i	$i+1$	$i+2$
$e_{0,2}$	$g_{0,2}$, $g_{2,2}$	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{1,2}$	$g_{0,2}$, $g_{2,2}$	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{2,2}$	$g_{0,2}$, $g_{2,2}$	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{3,2}$	$g_{0,2}$, $g_{2,2}$	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$

Group 7			
Fault	i	$i+1$	$i+2$
$e_{0,1}$	None	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{1,1}$	None	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{2,1}$	None	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{3,1}$	None	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{0,3}$	None	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{1,3}$	None	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{2,3}$	None	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$
$e_{3,3}$	None	$l_{0,1}$, $l_{0,3}$, $l_{2,1}$, $l_{2,3}$	$s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$

- Group 5. This group has four bytes: $e_{0,0}$, $e_{1,0}$, $e_{2,0}$ and $e_{3,0}$.
- Group 6. This group is made up of four bytes: $e_{0,2}$, $e_{1,2}$, $e_{2,2}$ and $e_{3,2}$.
- Group 7. This group contains eight bytes: $e_{0,1}$, $e_{1,1}$, $e_{2,1}$, $e_{3,1}$, $e_{0,3}$, $e_{1,3}$, $e_{2,3}$ and $e_{3,3}$.

The relation between the fault position and the changes of the keystream bytes is described in Table 2. For instance, if a fault is injected into byte

$e_{0,0}$ (see the first entry of Group 5 in Table 2) in Fig. 2, $g_{0,0}$, $g_{1,0}$, $g_{2,0}$ and $g_{3,0}$ are changed and we can observe the changes of $g_{0,0}$ and $g_{2,0}$ from the key stream in Round i. In Round $i+1$, all 16 bytes are changed after the MC transformation and we can see the changes of $l_{0,1}$, $l_{0,3}$, $l_{2,1}$ and $l_{2,3}$ from the key stream. Similarly, all 16 bytes are changed starting from the SB operation in Round $i+2$, and we can see the changes of $s_{0,0}$, $s_{0,2}$, $s_{2,0}$ and $s_{2,2}$ from the key stream.

By watching the changes of the keystream bytes listed in Table 1 and Table 2, we can identify the fault position. In this paper we are only interested in cases where a fault is injected into a byte which is listed in Table 1.

3.2 Recovering 4 Key Bytes of Round $i+2$

We first show that we are able to recover the actual values of 12 bytes after the SR transformation in Round $i+2$ by using 8 faults. Then, we describe the idea of recovering 4 key bytes of Round $i+2$ by using the 12 known values.

We provide an observation which is used to identify the position of a faulty byte in the MC transformation.

Observation 1 *In the MixColumns transformation, for each $0 \leq i \leq 3$, if we know three out of four input differences ($\Delta y_{0,i}$, $\Delta y_{1,i}$, $\Delta y_{2,i}$ and $\Delta y_{3,i}$) are zero and one input difference is non-zero and we also know two output differences ($\Delta j_{0,i}$ and $\Delta j_{2,i}$), the two unknown output differences ($\Delta j_{1,i}$ and $\Delta j_{3,i}$) and the position and the difference of the non-zero input can be uniquely determined.*

$$\begin{pmatrix} \Delta y_{0,i} \\ \Delta y_{1,i} \\ \Delta y_{2,i} \\ \Delta y_{3,i} \end{pmatrix} \xrightarrow{MC} \begin{pmatrix} \Delta j_{0,i} \\ \Delta j_{1,i} \\ \Delta j_{2,i} \\ \Delta j_{3,i} \end{pmatrix}.$$

Suppose a fault is injected into a byte which is $f_{0,3}$, $f_{1,0}$, $f_{2,1}$ or $f_{3,2}$ (Group 2 in Table 1). We use Fig. 3 to demonstrate the progress. We establish a formula, Formula (1), by using the input and out differences of the MC operation in Round $i+1$. We create another formula, Formula (2), with the input and out differences of the MC transformation in Round $i+2$.

$$\begin{pmatrix} 0\ 0\ 0\ \Delta y_{0,3} \\ 0\ 0\ 0\ \Delta y_{1,3} \\ 0\ 0\ 0\ \Delta y_{2,3} \\ 0\ 0\ 0\ \Delta y_{3,3} \end{pmatrix} \xrightarrow{MC} \begin{pmatrix} 0\ 0\ 0\ \Delta j_{0,3} \\ 0\ 0\ 0\ \Delta j_{1,3} \\ 0\ 0\ 0\ \Delta j_{2,3} \\ 0\ 0\ 0\ \Delta j_{3,3} \end{pmatrix} \tag{1}$$

$$\begin{pmatrix} 0 & 0 & 0 & \Delta q_{0,3} \\ 0 & 0 & \Delta q_{1,2} & 0 \\ 0 & \Delta q_{2,1} & 0 & 0 \\ \Delta q_{3,0} & 0 & 0 & 0 \end{pmatrix} \xrightarrow{MC} \begin{pmatrix} \Delta r_{0,0} & \Delta r_{0,1} & \Delta r_{0,2} & \Delta r_{0,3} \\ \Delta r_{1,0} & \Delta r_{1,1} & \Delta r_{1,2} & \Delta r_{1,3} \\ \Delta r_{2,0} & \Delta r_{2,1} & \Delta r_{2,2} & \Delta r_{2,3} \\ \Delta r_{3,0} & \Delta r_{3,1} & \Delta r_{3,2} & \Delta r_{3,3} \end{pmatrix} \tag{2}$$

1. We use Formula (1) to decide the values of $\Delta l_{1,3}$ and $\Delta l_{3,3}$ by performing the following steps. Since $\Delta j_{0,3}$ is equal to $\Delta l_{0,3}$ and $\Delta j_{2,3}$ is equal to $\Delta l_{2,3}$ ($\Delta l_{0,3}$ and $\Delta l_{2,3}$ can be calculated from the key stream), we know the values of $\Delta j_{0,3}$ and $\Delta j_{2,3}$. In the fourth columns of the input and output, there are 5 known bytes (3 zero bytes, $\Delta j_{0,3}$ and $\Delta j_{2,3}$) and 3 unknown bytes (the non-zero input byte, $\Delta j_{1,3}$ and $\Delta j_{3,3}$). Although we know there are three zero inputs and one non-zero input, we do not know the exact layout of the four input bytes. We can determine the 2 unknown output bytes ($\Delta j_{1,3}$ and $\Delta j_{3,3}$) and the position and the difference of the non-zero input byte by using Observation 1 (a similar method is described in [8], in which the authors used 4 known bytes to calculate the values of 4 unknown bytes). In Fig 3, we assume the faulty byte is $f_{1,0}$. As $\Delta l_{1,3}$ is equal to $\Delta j_{1,3}$ and $\Delta l_{3,3}$ is equal to $\Delta j_{3,3}$, we know the values of $\Delta l_{1,3}$ and $\Delta l_{3,3}$.
2. In Formula (2), $\Delta q_{3,0}$ and $\Delta q_{1,2}$ can be deduced as follows.
   (a) Because $\Delta r_{0,0}$ is equal to $\Delta s_{0,0}$ and $\Delta r_{2,0}$ is equal to $\Delta s_{2,0}$ ($\Delta s_{0,0}$ and $\Delta s_{2,0}$ can be computed from the key stream), we know the values of $\Delta r_{0,0}$ and $\Delta r_{2,0}$. In the first columns of the input and output, there are 5 known bytes (3 zero bytes, $\Delta r_{0,0}$ and $\Delta r_{2,0}$) and 3 unknown bytes ($\Delta q_{3,0}$, $\Delta r_{1,0}$ and $\Delta r_{3,0}$). Here we know the positions of the three zero inputs and the non-zero input (see the first column of the input in Formula (2)). The 3 unknown bytes can be deduced from the 5 known bytes.
   (b) Similarly, there are 5 known bytes (3 zero bytes, $\Delta r_{0,2}$ and $\Delta r_{2,2}$) and 3 unknown bytes ($\Delta q_{1,2}$, $\Delta r_{1,2}$ and $\Delta r_{3,2}$) in the third columns of the input and output. The values of 3 unknown bytes can be computed by making use of the 5 known bytes.

In Fig. 4, we know the values of $\Delta o_{1,3}$ and $\Delta o_{3,3}$ since we know $\Delta q_{1,2}$ and $\Delta q_{3,0}$ and the SR operation is just a permutation. Now we know the input differences ($\Delta l_{1,3}$, $\Delta l_{3,3}$) and the corresponding output differences ($\Delta o_{1,3}$, $\Delta o_{3,3}$) to the SB operation, and we can deduce 4 actual values for $o_{1,3}$, $o'_{1,3}$, $o_{3,3}$ and $o'_{3,3}$ by using a lookup table, which contains all possible input differences and their corresponding output differences of the SB operation. Here we encounter a 1-in-2 situation: although we already have 4 actual values for $o_{1,3}$, $o'_{1,3}$, $o_{3,3}$ and $o'_{3,3}$, we cannot distinguish the correct values ($o_{1,3}$ and $o_{3,3}$) from the faulty ones ($o'_{1,3}$ and $o'_{3,3}$). To address this problem, we need one more fault injected into $f_{1,0}$ and repeat the above steps since the correct values will appear twice in both keystream processing. After we get the actual values of $o_{1,3}$, $o'_{1,3}$, $o_{3,3}$ and $o'_{3,3}$, we know the actual values of $q_{1,2}$, $q'_{1,2}$, $q_{3,0}$ and $q'_{3,0}$ after the SR operation. As we know the actual values of $l_{0,1}$, $l_{0,3}$, $l_{2,1}$ and $l_{2,3}$ from the key stream, we know the actual values of $q_{0,1}$, $q_{0,3}$, $q_{2,1}$ and $q_{2,3}$ after the SB and SR operations. So far, we know the actual values of 6 bytes, $q_{1,2}$, $q_{3,0}$, $q_{0,1}$, $q_{0,3}$, $q_{2,1}$ and $q_{2,3}$, after the SR transformation in Round $i+2$:

$$\begin{pmatrix} ? & q_{0,1} & ? & q_{0,3} \\ ? & ? & q_{1,2} & ? \\ ? & q_{2,1} & ? & q_{2,3} \\ q_{3,0} & ? & ? & ? \end{pmatrix}.$$

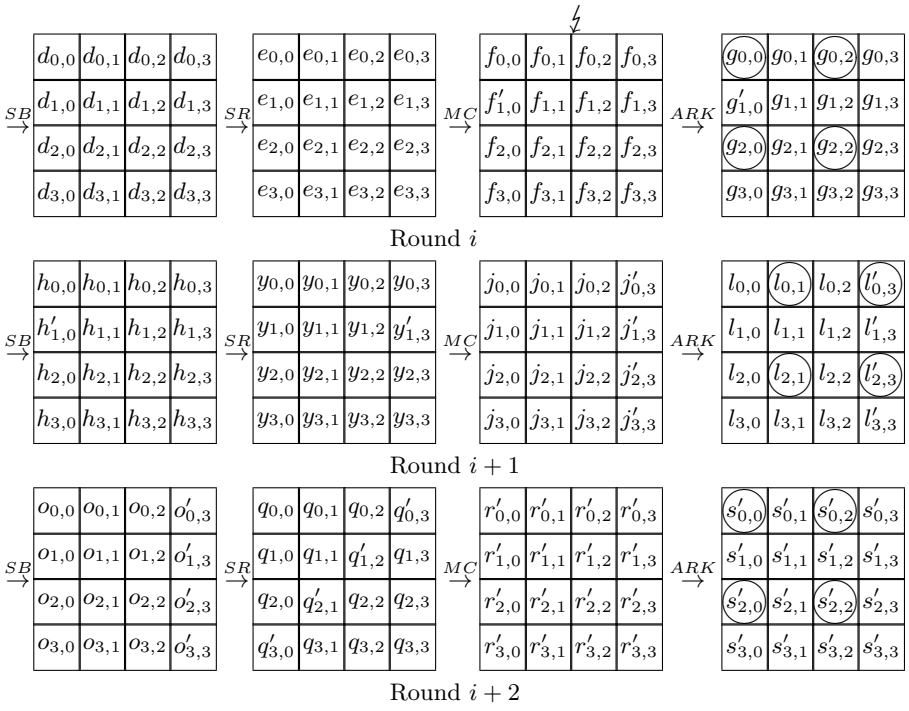

Fig. 3. Computing the values of $\Delta l_{1,3}$, $\Delta l_{3,3}$, $\Delta q_{1,2}$ and $\Delta q_{3,0}$

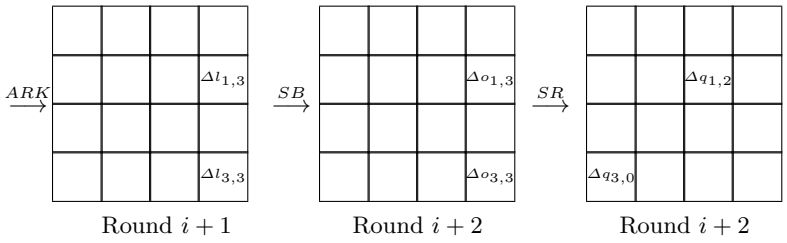

Fig. 4. Deducing the actual values of $q_{1,2}$ and $q_{3,0}$

We use the same idea to recover the actual values of $q_{0,0}$, $q_{2,2}$, $q_{1,0}$, $q_{3,2}$, $q_{2,0}$ and $q_{0,2}$. The details are listed as follows.

1. We are able to obtain the actual values of $q_{0,0}$ and $q_{2,2}$ by using 2 faults which are injected into $f_{0,0}$, or $f_{2,2}$. The procedure is shown in Appendix A.
2. We can recover the actual values of $q_{1,0}$ and $q_{3,2}$ by using 2 faults which are induced on $f_{0,1}$, $f_{1,2}$, $f_{2,3}$ or $f_{3,0}$. The steps are described in Appendix B.
3. We can get the actual values of $q_{2,0}$ and $q_{0,2}$ by using 2 faults which are injected into $f_{0,2}$ or $f_{2,0}$. The details are presented in Appendix C.

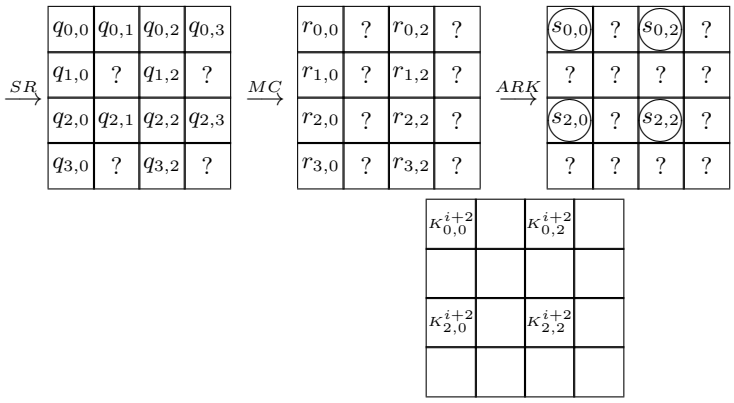

Fig. 5. Recovering 4 key bytes

Since we now know the actual values of the 12 bytes after the SR transformation in Round $i+2$ in Fig. 5, we can compute the actual values of the first column ($r_{0,0}$, $r_{1,0}$, $r_{2,0}$ and $r_{3,0}$) and third column ($r_{0,2}$, $r_{1,2}$, $r_{2,2}$ and $r_{3,2}$) of the MC transformation. By XORing ($r_{0,0}$, $r_{0,2}$, $r_{2,0}$, $r_{2,2}$) with ($s_{0,0}$, $s_{0,2}$, $s_{2,0}$, $s_{2,2}$), we finally recover 4 round key bytes: $K_{0,0}^{i+2}$, $K_{0,2}^{i+2}$, $K_{2,0}^{i+2}$ and $K_{2,2}^{i+2}$.

3.3 Retrieving 16 Key Bytes in Round $i-1$, i, $i+1$ and $i+3$

By using the same techniques as described in Section 3.2, we can recover 16 ($K_{0,1}^{i-1}$, $K_{0,3}^{i-1}$, $K_{2,1}^{i-1}$, $K_{2,3}^{i-1}$, $K_{0,0}^{i}$, $K_{0,2}^{i}$, $K_{2,0}^{i}$, $K_{2,2}^{i}$, $K_{0,1}^{i+1}$, $K_{0,3}^{i+1}$, $K_{2,1}^{i+1}$, $K_{2,3}^{i+1}$, $K_{0,1}^{i+3}$, $K_{0,3}^{i+3}$, $K_{2,1}^{i+3}$ and $K_{2,3}^{i+3}$) more key bytes in Round $i-1$, i, $i+1$ and $i+3$ (see Fig. 7) with 32 faults. The details of recovering these 16 key bytes are provided as follows. We use Fig. 6 to describe the complete details.

3.3.1 Computing $K_{0,1}^{i-1}$, $K_{0,3}^{i-1}$, $K_{2,1}^{i-1}$ and $K_{2,3}^{i-1}$

1. Inject 2 faults into $\theta_{0,1}$ or $\theta_{2,3}$, and use these 2 faulty bytes to determine the actual values of $z_{0,1}$ and $z_{2,3}$.
2. Induce 2 faults on $\theta_{0,2}$ or $\theta_{2,1}$, and employ these 2 faulty values to calculate the actual values of $z_{0,3}$ and $z_{2,1}$.
3. Inject 2 faults into $\theta_{0,0}$, $\theta_{1,1}$, $\theta_{2,2}$ or $\theta_{3,3}$, and use these 2 faulty bytes to decide the actual values of $z_{1,3}$ and $z_{3,1}$.
4. Induce 2 faults on $\theta_{0,2}$, $\theta_{1,3}$, $\theta_{2,0}$ or $\theta_{3,1}$, and employ these 2 faulty values to find out the actual values of $z_{1,1}$ and $z_{3,3}$.
5. Apply the MC operation to ($z_{0,1}$, $z_{1,1}$, $z_{2,1}$, $z_{3,1}$) and ($z_{0,3}$, $z_{1,3}$, $z_{2,3}$, $z_{3,3}$) to get the actual values of $\beta_{0,1}$, $\beta_{1,1}$, $\beta_{2,1}$, $\beta_{3,1}$, $\beta_{0,3}$, $\beta_{1,3}$, $\beta_{2,3}$ and $\beta_{3,3}$. XOR $\beta_{0,1}$ with $\lambda_{0,1}$, $\beta_{0,3}$ with $\lambda_{0,3}$, $\beta_{2,1}$ with $\lambda_{2,1}$, and $\beta_{2,3}$ with $\lambda_{2,3}$ to recover $K_{0,1}^{i-1}$, $K_{0,3}^{i-1}$, $K_{2,1}^{i-1}$ and $K_{2,3}^{i-1}$.

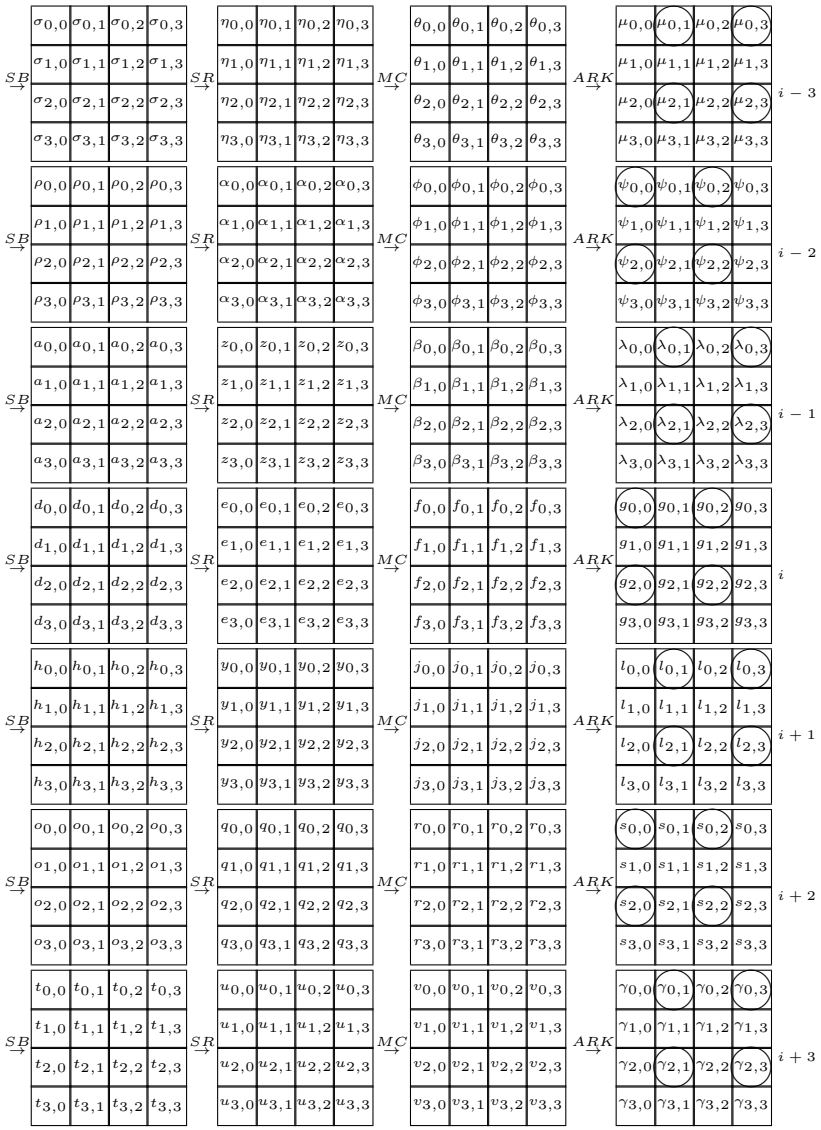

Fig. 6. The seven-round diagram

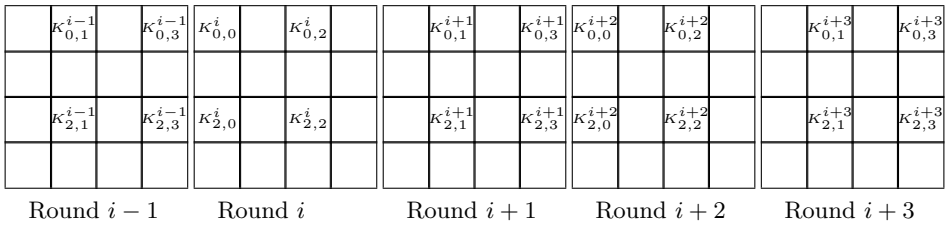

Fig. 7. The recovered key bytes

3.3.2 Determining $K^i_{0,0}$, $K^i_{0,2}$, $K^i_{2,0}$ and $K^i_{2,2}$

1. Induce 2 faults on $\phi_{0,0}$ or $\phi_{2,2}$, and employ these 2 faulty values to compute the actual values of $e_{0,0}$ and $e_{2,2}$.
2. Inject 2 faults into $\phi_{0,2}$ or $\phi_{2,0}$, and use these 2 faulty values to calculate the actual values of $e_{0,2}$ and $e_{2,0}$.
3. Induce 2 faults on $\phi_{0,3}$, $\phi_{1,0}$, $\phi_{2,1}$ or $\phi_{3,2}$, and use these 2 faulty values to decide the actual values of $e_{1,2}$ and $e_{3,0}$.
4. Inject 2 faults into $\phi_{0,1}$, $\phi_{1,2}$, $\phi_{2,3}$ or $\phi_{3,0}$, and employ these 2 faulty bytes to determine the actual values of $e_{1,0}$ and $e_{3,2}$.
5. Apply the MC operation to ($e_{0,0}$, $e_{1,0}$, $e_{2,0}$, $e_{3,0}$) and ($e_{0,2}$, $e_{1,2}$, $e_{2,2}$, $e_{3,2}$) to get the actual values of $f_{0,0}$, $f_{1,0}$, $f_{2,0}$, $f_{3,0}$, $f_{0,2}$, $f_{1,2}$, $f_{2,2}$ and $f_{3,2}$. XOR $f_{0,0}$ with $g_{0,0}$, $f_{0,2}$ with $g_{0,2}$, $f_{2,0}$ with $g_{2,0}$, and $f_{2,2}$ with $g_{2,2}$ to retrieve the actual values of $K^i_{0,0}$, $K^i_{0,2}$, $K^i_{2,0}$ and $K^i_{2,2}$.

3.3.3 Calculating $K^{i+1}_{0,1}$, $K^{i+1}_{0,3}$, $K^{i+1}_{2,1}$ and $K^{i+1}_{2,3}$

1. Inject 2 faults into $\beta_{0,1}$ or $\beta_{2,3}$, and use these 2 faulty bytes to decide the actual values of $y_{0,1}$ and $y_{2,3}$.
2. Induce 2 faults on $\beta_{0,3}$ or $\beta_{2,1}$, and employ these 2 faulty bytes to compute the actual values of $y_{0,3}$ and $y_{2,1}$.
3. Inject 2 faults into $\beta_{0,0}$, $\beta_{1,1}$, $\beta_{2,2}$ or $\beta_{3,3}$ and use these 2 faulty bytes to determine the actual values of $y_{1,3}$ and $y_{3,1}$.
4. Induce 2 faults on $\beta_{0,2}$, $\beta_{1,3}$, $\beta_{2,0}$ or $\beta_{3,1}$, and employ these 2 faulty bytes to calculate the actual values of $y_{1,1}$ and $y_{3,3}$.
5. Apply the MC transformation to ($y_{0,1}$, $y_{1,1}$, $y_{2,1}$, $y_{3,1}$) and ($y_{0,3}$, $y_{1,3}$, $y_{2,3}$, $y_{3,3}$) to obtain the actual values of $j_{0,1}$, $j_{1,1}$, $j_{2,1}$, $j_{3,1}$, $j_{0,3}$, $j_{1,3}$, $j_{2,3}$ and $j_{3,3}$. XOR $j_{0,1}$ with $l_{0,1}$, $j_{0,3}$ with $l_{0,3}$, $j_{2,1}$ with $l_{2,1}$, and $j_{2,3}$ with $l_{2,3}$ to recover $K^{i+1}_{0,1}$, $K^{i+1}_{0,3}$, $K^{i+1}_{2,1}$ and $K^{i+1}_{2,3}$.

3.3.4 Recovering $K^{i+3}_{0,1}$, $K^{i+3}_{0,3}$, $K^{i+3}_{2,1}$ and $K^{i+3}_{2,3}$

1. Use 2 faulty bytes which are $j_{0,1}$ or $j_{2,3}$ to determine the actual values of $u_{0,1}$ and $u_{2,3}$.
2. Employ 2 faulty bytes which are $j_{0,3}$ or $j_{2,1}$ to retrieve the actual values of $u_{0,3}$ and $u_{2,1}$.
3. Make use of 2 faulty bytes which are $j_{0,0}$, $j_{1,1}$, $j_{2,2}$ or $j_{3,3}$ to calculate the actual values of $u_{1,3}$ and $u_{3,1}$.
4. Employ 2 faulty bytes which are $j_{0,2}$, $j_{1,3}$, $j_{2,0}$ or $j_{3,1}$ to compute the actual values of $u_{1,1}$ and $u_{3,3}$.
5. Apply the MC operation to $(u_{0,1}, u_{1,1}, u_{2,1}, u_{3,1})$ and $(u_{0,3}, u_{1,3}, u_{2,3}, u_{3,3})$ to get the actual values of $v_{0,1}$, $v_{1,1}$, $v_{2,1}$, $v_{3,1}$, $v_{0,3}$, $v_{1,3}$, $v_{2,3}$ and $v_{3,3}$. We can retrieve $K^{i+3}_{0,1}$, $K^{i+3}_{0,3}$, $K^{i+3}_{2,1}$ and $K^{i+3}_{2,3}$ (see Fig. 7) by XORing $v_{0,1}$ with $\gamma_{0,1}$, $v_{0,3}$ with $\gamma_{0,3}$, $v_{2,1}$ with $\gamma_{2,1}$, and $v_{2,3}$ with $\gamma_{2,3}$.

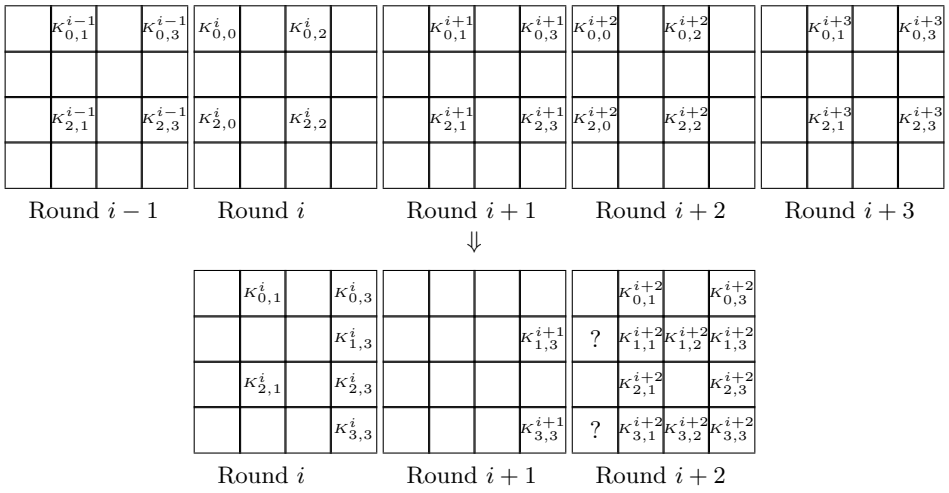

Fig. 8. The deduced key bytes

3.4 Deducing 10 more Key Bytes in Round $i+2$

By employing the definition and the properties of the AES key schedule, we use the 20 recovered key bytes to deduce 10 more key bytes in Round $i+2$ (see Fig. 8). The steps of deducing these 10 key bytes are listed as follows, where SB^{-1} is the inverse of the byte substitution transformation and $Rcon^i$ represents the round constant used to generate round key K^i.

1. Deduce $K_{0,1}^{i+2}$, $K_{2,1}^{i+2}$, $K_{0,3}^{i+2}$ and $K_{2,3}^{i+2}$.

$$\begin{aligned} K_{0,1}^{i+2} &= K_{0,1}^{i+1} \oplus K_{0,0}^{i+2}, \\ K_{2,1}^{i+2} &= K_{2,1}^{i+1} \oplus K_{2,0}^{i+2}, \\ K_{0,3}^{i+2} &= K_{0,3}^{i+1} \oplus K_{0,2}^{i+2}, \\ K_{2,3}^{i+2} &= K_{2,3}^{i+1} \oplus K_{2,2}^{i+2}. \end{aligned}$$

2. Calculate $K_{3,3}^{i+2}$ and $K_{1,3}^{i+2}$.

$$\begin{aligned} K_{2,1}^{i+3} &= K_{2,0}^{i+3} \oplus K_{2,1}^{i+2} = SB(K_{3,3}^{i+2}) \oplus RCON^{i+3}(2) \oplus K_{2,0}^{i+2} \oplus K_{2,1}^{i+2} \\ &= SB(K_{3,3}^{i+2}) \oplus Rcon^{i+3}(2) \oplus K_{2,1}^{i+1}, \\ K_{3,3}^{i+2} &= SB^{-1}(K_{2,1}^{i+3} \oplus Rcon^{i+3}(2) \oplus K_{2,1}^{i+1}), \\ K_{0,1}^{i+3} &= K_{0,0}^{i+3} \oplus K_{0,1}^{i+2} = SB(K_{1,3}^{i+2}) \oplus Rcon^{i+3}(0) \oplus K_{0,0}^{i+2} \oplus K_{0,1}^{i+2} \\ &= SB(K_{1,3}^{i+2}) \oplus Rcon^{i+3}(0) \oplus K_{0,1}^{i+1}, \\ K_{1,3}^{i+2} &= SB^{-1}(K_{0,1}^{i+3} \oplus Rcon^{i+3}(0) \oplus K_{0,1}^{i+1}). \end{aligned}$$

3. Determine $K_{3,1}^{i+2}$ and $K_{1,1}^{i+2}$.

$$\begin{aligned}
K_{0,3}^{i} &= K_{0,3}^{i-1} \oplus K_{0,2}^{i}, \\
K_{2,3}^{i} &= K_{2,3}^{i-1} \oplus K_{2,2}^{i}, \\
K_{2,1}^{i+1} &= K_{2,0}^{i+1} \oplus K_{2,1}^{i} = SB(K_{3,3}^{i}) \oplus Rcon^{i+1}(2) \oplus K_{2,0}^{i} \oplus K_{2,1}^{i} \\
&= SB(K_{3,3}^{i}) \oplus Rcon^{i+1}(2) \oplus K_{2,0}^{i} \oplus K_{2,0}^{i} \oplus K_{2,1}^{i-1} \\
&= SB(K_{3,3}^{i}) \oplus Rcon^{i+1}(2) \oplus K_{2,1}^{i-1}, \\
K_{3,3}^{i} &= SB^{-1}(K_{2,1}^{i+1} \oplus Rcon^{i+1}(2) \oplus K_{2,1}^{i-1}), \\
K_{3,1}^{i+2} &= K_{3,3}^{i} \oplus K_{3,3}^{i+2}, \\
K_{0,1}^{i+1} &= K_{0,0}^{i+1} \oplus K_{0,1}^{i} = SB(K_{1,3}^{i}) \oplus Rcon^{i+1}(0) \oplus K_{0,0}^{i} \oplus K_{0,1}^{i} \\
&= SB(K_{1,3}^{i}) \oplus Rcon^{i+1}(0) \oplus K_{0,1}^{i-1}, \\
K_{1,3}^{i} &= SB^{-1}(K_{0,1}^{i+1} \oplus Rcon^{i+1}(0) \oplus K_{0,1}^{i-1}), \\
K_{1,1}^{i+2} &= K_{1,3}^{i} \oplus K_{1,3}^{i+2}.
\end{aligned}$$

4. Decide $K_{3,2}^{i+2}$ and $K_{1,2}^{i+2}$.

$$\begin{aligned}
K_{0,1}^{i} &= K_{0,1}^{i-1} \oplus K_{0,0}^{i}, \\
K_{2,1}^{i} &= K_{2,1}^{i-1} \oplus K_{2,0}^{i}, \\
K_{2,1}^{i+2} &= K_{2,0}^{i+2} \oplus K_{2,1}^{i+1} = SB(K_{3,3}^{i+1}) \oplus Rcon^{i+2}(2) \oplus K_{2,0}^{i+1} \oplus K_{2,1}^{i+1} \\
&= SB(K_{3,3}^{i+1}) \oplus Rcon^{i+2}(2) \oplus K_{2,1}^{i}, \\
K_{3,3}^{i+1} &= SB^{-1}(K_{2,1}^{i+2} \oplus Rcon^{i+2}(2) \oplus K_{2,1}^{i}), \\
K_{3,2}^{i+2} &= K_{3,3}^{i+1} \oplus K_{3,3}^{i+2}, \\
K_{0,1}^{i+2} &= K_{0,0}^{i+2} \oplus K_{0,1}^{i+1} = SB(K_{1,3}^{i+1}) \oplus Rcon^{i+2}(0) \oplus K_{0,0}^{i+1} \oplus K_{0,1}^{i+1} \\
&= SB(K_{1,3}^{i+1}) \oplus Rcon^{i+2}(0) \oplus K_{0,1}^{i}, \\
K_{1,3}^{i+1} &= SB^{-1}(K_{0,1}^{i+2} \oplus Rcon^{i+2}(0) \oplus K_{0,1}^{i}), \\
K_{1,2}^{i+2} &= K_{1,3}^{i+1} \oplus K_{1,3}^{i+2}.
\end{aligned}$$

In summary, we recover 14 key bytes of Round $i+2$, and the 2 unknown key bytes ($K_{1,0}^{i+2}$ and $K_{3,0}^{i+2}$, represented by a question mark) can be determined by exhaustive search with 2^{16} operations.

4 Conclusions

We described a differential fault attack on LEX in this paper. We presented a method to decide the fault position by observing the changes of the key stream after a fault is injected. The attack makes use of the differential properties of the AES round transformations, the AES key schedule properties and the structural features of LEX. The proposed attack needs 40 faults and recovers the secret key of LEX with 2^{16} time complexity.

References

1. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer, Heidelberg (1993)
2. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513-525. Springer, Heidelberg (1997)
3. Biryukov, A.: A New 128-bit Key Stream Cipher LEX. ECRYPT stream cipher project report 2005/013. http://www.ecrypt.eu.org/stream (2005)
4. Biryukov, A.: The Design of a Stream Cipher LEX. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 67-75. Springer, Heidelberg (2006)
5. Biryukov, A.: The Tweak for LEX-128, LEX-192, LEX-256. ECRYPT stream cipher project report 2006/037. http://www.ecrypt.eu.org/stream (2006)
6. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37-51. Springer, Heidelberg (1997)
7. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, Heidelberg (2002)
8. Dunkelman, O., Keller, N.: A New Attack on the LEX Stream Cipher. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 539-556. Springer, Heidelberg (2008)
9. Englund, H., Hell, M., Johansson, T.: A Note on Distinguishing Attacks. In: Preproceedings of State of the Art of Stream Ciphers workshop (SASC 2007). Bochum, Germany, pp. 7378 (2007)
10. Hoch, J.J., Shamir, A.: Fault Analysis of Stream Ciphers. In: Joye, M., Quisquater, J.J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 240-253. Springer, Heidelberg (2004)
11. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104-113. Springer, Heidelberg (1996)
12. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388-397. Springer, Heidelberg (1999)
13. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386-397. Springer, Heidelberg (1993)
14. Mondal, M., Mukhopadhyay, D.: Related Key Cryptanalysis of the LEX Stream Cipher. http://eprint.iacr.org/2010/011 (2010)
15. Wu, H., Preneel, B.: Resynchronization Attacks on WG and LEX. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 422-432. Springer, Heidelberg (2006)

A Calculating the actual values of $q_{0,0}$ and $q_{2,2}$

Assume a fault is induced on $f_{0,0}$ or $f_{2,2}$ (Group 1 in Table 1), and suppose the faulty byte is $f_{0,0}$. We create a formula, Formula (3), by employing the input and out differences of the MC operation in Round $i+1$. We build another formula, Formula (4), with the input and out differences of the MC transformation in Round $i+2$.

$$\begin{pmatrix} \Delta y_{0,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{MC} \begin{pmatrix} \Delta j_{0,0} & 0 & 0 & 0 \\ \Delta j_{1,0} & 0 & 0 & 0 \\ \Delta j_{2,0} & 0 & 0 & 0 \\ \Delta j_{3,0} & 0 & 0 & 0 \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} \Delta q_{0,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta q_{1,3} \\ 0 & 0 & \Delta q_{2,2} & 0 \\ 0 & \Delta q_{3,1} & 0 & 0 \end{pmatrix} \xrightarrow{MC} \begin{pmatrix} \Delta r_{0,0} & \Delta r_{0,1} & \Delta r_{0,2} & \Delta r_{0,3} \\ \Delta r_{1,0} & \Delta r_{1,1} & \Delta r_{1,2} & \Delta r_{1,3} \\ \Delta r_{2,0} & \Delta r_{2,1} & \Delta r_{2,2} & \Delta r_{2,3} \\ \Delta r_{3,0} & \Delta r_{3,1} & \Delta r_{3,2} & \Delta r_{3,3} \end{pmatrix} \tag{4}$$

1. In Formula (3), $\Delta y_{0,0}$ can be computed from the key stream of Round i by using the values of $g_{0,0}$ and $g'_{0,0}$. In the first columns of the input and output, there are 4 known bytes ($\Delta y_{0,0}$ and 3 zero bytes) and 4 unknown bytes ($\Delta j_{0,0}$, $\Delta j_{1,0}$, $\Delta j_{2,0}$ and $\Delta j_{3,0}$). The 4 unknown bytes can be decided by using the 4 known bytes. After $\Delta j_{0,0}$, $\Delta j_{1,0}$, $\Delta j_{2,0}$ and $\Delta j_{3,0}$ are decided, we know the values of $\Delta l_{0,0}$, $\Delta l_{1,0}$, $\Delta l_{2,0}$ and $\Delta l_{3,0}$.
2. In Formula (4), $\Delta r_{0,0}$ can be computed from the key stream of Round $i+2$ by using the values of $s_{0,0}$ and $s'_{0,0}$. Similarly, we get the value of $\Delta r_{2,0}$. In the first columns of the input and output, there are 5 known bytes ($\Delta r_{0,0}$, $\Delta r_{2,0}$ and 3 zero bytes) and 3 unknown bytes ($\Delta q_{0,0}$, $\Delta r_{1,0}$ and $\Delta r_{3,0}$). We get the values of $\Delta q_{0,0}$, $\Delta r_{1,0}$ and $\Delta r_{3,0}$ by using the 5 known bytes. By using the same method to analyze the third columns of the input and output, we can deduce the values of $\Delta q_{2,2}$, $\Delta r_{1,2}$ and $\Delta r_{3,2}$ by employing the 5 known bytes ($\Delta r_{0,2}$, $\Delta r_{2,2}$ and 3 zero bytes).
3. We know $\Delta o_{0,0}$ and $\Delta o_{2,0}$ because $\Delta q_{0,0}$ is equal to $\Delta o_{0,0}$ and $\Delta q_{2,2}$ is equal to $\Delta o_{2,0}$. Now we know the input differences ($\Delta l_{0,0}$, $\Delta l_{2,0}$) and the corresponding output differences ($\Delta o_{0,0}$, $\Delta o_{2,0}$) to the SB operation, and we can deduce 4 actual values for $o_{0,0}$, $o'_{0,0}$, $o_{2,0}$ and $o'_{2,0}$. Although we have 4 actual values for $o_{0,0}$, $o'_{0,0}$, $o_{2,0}$ and $o'_{2,0}$, we are not able to separate the correct values ($o_{0,0}$ and $o_{2,0}$) from the faulty ones ($o'_{0,0}$ and $o'_{2,0}$). To overcome this problem, we need one more fault induced on $f_{0,0}$ and repeat the steps mentioned above to make the correct values emerge twice. After knowing the actual values of $o_{0,0}$, $o'_{0,0}$, $o_{2,0}$ and $o'_{2,0}$, we obtain the actual values of $q_{0,0}$, $q'_{0,0}$, $q_{2,2}$ and $q'_{2,2}$ after the SR operation.

B Computing the actual values of $q_{1,0}$ and $q_{3,2}$

Suppose a fault is induced on a byte which is $f_{0,1}$, $f_{1,2}$, $f_{2,3}$ or $f_{3,0}$ (Group 3 in Table 1). By using the input and out differences of the MC operation in Round $i+1$, we establish a formula, Formula (5). By employing the input and out differences of the MC transformation in Round $i+2$, we build another formula, Formula (6).

$$\begin{pmatrix} 0 & \Delta y_{0,1} & 0 & 0 \\ 0 & \Delta y_{1,1} & 0 & 0 \\ 0 & \Delta y_{2,1} & 0 & 0 \\ 0 & \Delta y_{3,1} & 0 & 0 \end{pmatrix} \xrightarrow{MC} \begin{pmatrix} 0 & \Delta j_{0,1} & 0 & 0 \\ 0 & \Delta j_{1,1} & 0 & 0 \\ 0 & \Delta j_{2,1} & 0 & 0 \\ 0 & \Delta j_{3,1} & 0 & 0 \end{pmatrix} \tag{5}$$

$$\begin{pmatrix} 0 & \Delta q_{0,1} & 0 & 0 \\ \Delta q_{1,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta q_{2,3} \\ 0 & 0 & \Delta q_{3,2} & 0 \end{pmatrix} \xrightarrow{MC} \begin{pmatrix} \Delta r_{0,0} & \Delta r_{0,1} & \Delta r_{0,2} & \Delta r_{0,3} \\ \Delta r_{1,0} & \Delta r_{1,1} & \Delta r_{1,2} & \Delta r_{1,3} \\ \Delta r_{2,0} & \Delta r_{2,1} & \Delta r_{2,2} & \Delta r_{2,3} \\ \Delta r_{3,0} & \Delta r_{3,1} & \Delta r_{3,2} & \Delta r_{3,3} \end{pmatrix} \tag{6}$$

1. In Formula (5), $\Delta j_{0,1}$ can be computed from the key stream of Round $i+1$ by using the values of $l_{0,1}$ and $l'_{0,1}$, and $\Delta j_{2,1}$ can be computed from the key stream of Round $i+1$ by using the values of $l_{2,1}$ and $l'_{2,1}$. In the second columns of the input and output, we employ Observation 1 and use the 5 known bytes ($\Delta j_{0,1}$, $\Delta j_{2,1}$ and 3 zero input bytes) to decide the 2 unknown output bytes ($\Delta j_{1,1}$ and $\Delta j_{3,1}$) and the position and the difference of the non-zero input byte. Assume the recovered non-zero input byte is $\Delta y_{2,1}$.
2. In Formula (6), $\Delta r_{0,0}$ can be computed from the key stream of Round $i+2$ by using the values of $s_{0,0}$ and $s'_{0,0}$. Similarly, we get the value of $\Delta r_{2,0}$. In the first columns of the input and output, we use the 5 known bytes ($\Delta r_{0,0}$, $\Delta r_{2,0}$ and 3 zero bytes) to decide the 3 unknown bytes ($\Delta q_{1,0}$, $\Delta r_{1,0}$ and $\Delta r_{3,0}$). By using the same method to analyze the third columns of the input and output, we can deduce the values of 3 unknown bytes ($\Delta q_{3,2}$, $\Delta r_{1,2}$ and $\Delta r_{3,2}$) by employing the 5 known bytes ($\Delta r_{0,2}$, $\Delta r_{2,2}$ and 3 zero bytes).
3. We know $\Delta o_{1,1}$ and $\Delta o_{3,1}$ because $\Delta q_{1,0}$ is equal to $\Delta o_{1,1}$ and $\Delta q_{3,2}$ is equal to $\Delta o_{3,1}$. We now know the input differences ($\Delta l_{1,1}$, $\Delta l_{3,1}$) and the corresponding output differences ($\Delta o_{1,1}$, $\Delta o_{3,1}$) to the SB operation, and we can deduce 4 actual values for $o_{1,1}$, $o'_{1,1}$, $o_{3,1}$ and $o'_{3,1}$. Although we have 4 actual values for $o_{1,1}$, $o'_{1,1}$, $o_{3,1}$ and $o'_{3,1}$, we cannot distinguish the correct values ($o_{1,1}$ and $o_{3,1}$) from the faulty ones ($o'_{1,1}$ and $o'_{3,1}$). To address this issue, we need one more fault injected into $f_{2,3}$ and repeat the above steps to make the correct values appear twice in both keystream processing. After knowing the actual values of $o_{1,1}$, $o'_{1,1}$, $o_{3,1}$ and $o'_{3,1}$, we know the actual values of $q_{1,0}$, $q'_{1,0}$, $q_{3,2}$ and $q'_{3,2}$ after the SR operation.

C Deducing the actual values of $q_{2,0}$ and $q_{0,2}$

Assume a fault is injected into a byte which is $f_{0,2}$ or $f_{2,0}$ (Group 1 in Table 1), and assume the faulty byte is $f_{2,0}$. We create a formula, Formula (7), with the input and out differences of the MC operation in Round $i+1$. We establish another formula, Formula (8), by employing the input and out differences of the MC transformation in Round $i+2$.

$$\begin{pmatrix} 0\,0 & 0 & 0 \\ 0\,0 & 0 & 0 \\ 0\,0 & \Delta y_{2,2} & 0 \\ 0\,0 & 0 & 0 \end{pmatrix} \xrightarrow{MC} \begin{pmatrix} 0\,0 & \Delta j_{0,2} & 0 \\ 0\,0 & \Delta j_{1,2} & 0 \\ 0\,0 & \Delta j_{2,2} & 0 \\ 0\,0 & \Delta j_{3,2} & 0 \end{pmatrix} \tag{7}$$

$$\begin{pmatrix} 0 & 0 & \Delta q_{0,2} & 0 \\ 0 & \Delta q_{1,1} & 0 & 0 \\ \Delta q_{2,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta q_{3,3} \end{pmatrix} \xrightarrow{MC} \begin{pmatrix} \Delta r_{0,0} & \Delta r_{0,1} & \Delta r_{0,2} & \Delta r_{0,3} \\ \Delta r_{1,0} & \Delta r_{1,1} & \Delta r_{1,2} & \Delta r_{1,3} \\ \Delta r_{2,0} & \Delta r_{2,1} & \Delta r_{2,2} & \Delta r_{2,3} \\ \Delta r_{3,0} & \Delta r_{3,1} & \Delta r_{3,2} & \Delta r_{3,3} \end{pmatrix} \tag{8}$$

1. In Formula (7), $\Delta y_{2,2}$ can be computed from the key stream of Round i by using the values of $g_{2,0}$ and $g'_{2,0}$. In the third columns of the input and

output, we use 4 known bytes ($\Delta y_{2,2}$ and 3 zero bytes) to decide 4 unknown bytes ($\Delta j_{0,2}$, $\Delta j_{1,2}$, $\Delta j_{2,2}$ and $\Delta j_{3,2}$). The values of $\Delta l_{0,2}$, $\Delta l_{1,2}$, $\Delta l_{2,2}$ and $\Delta l_{3,2}$ are also decided.

2. In Formula (8), $\Delta r_{0,0}$ can be computed from the key stream of Round $i+2$ by using the values of $s_{0,0}$ and $s'_{0,0}$. Similarly, we get the value of $\Delta r_{2,0}$ by using the values of $\Delta s_{2,0}$ and $\Delta s'_{2,0}$. In the first columns of the input and output, we can use the 5 known bytes ($\Delta r_{0,0}$, $\Delta r_{2,0}$ and 3 zero bytes) to decide the 3 unknown bytes ($\Delta q_{2,0}$, $\Delta r_{1,0}$ and $\Delta r_{3,0}$). Similarly, we can use the 5 known bytes ($\Delta r_{0,2}$, $\Delta r_{2,2}$ and 3 zero bytes) to decide the 3 unknown bytes ($\Delta q_{0,2}$, $\Delta r_{1,2}$ and $\Delta r_{3,2}$) in the third columns of the input and output.
3. We know $\Delta o_{0,2}$ and $\Delta o_{2,2}$ because $\Delta q_{0,2}$ is equal to $\Delta o_{0,2}$ and $\Delta q_{2,0}$ is equal to $\Delta o_{2,2}$. We now know the input differences ($\Delta l_{0,2}$, $\Delta l_{2,2}$) and the corresponding output differences ($\Delta o_{0,2}$, $\Delta o_{2,2}$) to the SB operation, and we can deduce 4 actual values for $o_{0,2}$, $o'_{0,2}$, $o_{2,2}$ and $o'_{2,2}$. Although we have 4 actual values for $o_{0,2}$, $o'_{0,2}$, $o_{2,2}$ and $o'_{2,2}$, we cannot separate the correct values ($o_{0,2}$ and $o_{2,2}$) from the faulty ones ($o'_{0,2}$ and $o'_{2,2}$). To overcome this obstacle, we need one more fault injected into $f_{2,0}$ and repeat the above steps because the correct values will emerge twice in both keystream processing. After knowing the actual values of $o_{0,2}$, $o'_{0,2}$, $o_{2,2}$ and $o'_{2,2}$, we know the actual values of $q_{0,2}$, $q'_{0,2}$, $q_{2,0}$ and $q'_{2,0}$ after the SR operation.