A differential cryptanalysis of Yen–Chen–Wu multimedia cryptography system

Chengqing Li a,*, Shujun Li b,**, Kwok-Tung Lo a, Kyandoghere Kyamakya c

a Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong, China
b Fachbereich Informatik und Informationswissenschaft, Universität Konstanz, Fach M697, Universitätsstraße 10, 78457 Konstanz, Germany
c Universität Klagenfurt, Institut für Intelligente Systemtechnologien, Universitätsstraße 65-67, 9020 Klagenfurt, Austria

ARTICLE INFO

Article info

Article history:
Received 5 November 2009
Received in revised form 4 February 2010
Accepted 28 February 2010
Available online xxxx

Keywords:
Chaos
Cryptanalysis
Differential attack
Encryption
Multimedia
Security

1. Introduction

The prevalence of multimedia data makes its security become more and more important. However, traditional cryptosystems cannot protect multimedia data efficiently due to the big differences between texts and multimedia data, such as the bulky sizes and strong correlation between neighboring elements of uncompressed multimedia data. In addition, multimedia encryption schemes have some special requirements like high bitrate and easy concatenation of different components of the whole multimedia processing system. So, designing special encryption schemes protecting multimedia data becomes necessary. To meet this challenge, a great number of multimedia encryption schemes have been proposed in the past two decades (Bourbakis and Alexopoulos, 1992; Chung and Chang, 1998; Scharinger, 1998; Fridrich, 1998; Chen et al., 2004; Wu and Kuo, 2005; Flores-Carmona and Carpio-Valadez, 2006; Pareek et al., 2006; Xiao et al., 2006; Kim et al., 2007; Wong and Yuen, XXX). Due to the subtle similarity between chaos and cryptography, some of multimedia encryption schemes were designed based on one or more chaotic systems (Scharinger, 1998; Fridrich, 1998; Chen et al., 2004; Pareek et al., 2006; Xiao et al., 2006; Wong and Yuen, XXX). Meanwhile, a lot of cryptanalytic work has also been reported, showing that many encryption schemes were not designed carefully and are prone to various kinds of attacks (Jan and Tseng, 1996; Chang and Yu, 2002; Lian et al., 2005; Solak, 2005; Álvarez and Li, 2005; Wang et al., 2005; Arroyo et al., 2008; Zhou et al., 2007; Rhouma and Belghith, 2008; Jakimoski and Subbalakshmi, 2008; Zhou et al., 2009; Li et al., 2009).

In the past decade, a series of encryption schemes were proposed by Yen and Guo’s research group (Yen and Guo, 2000; Guo et al., 2002; Chen and Yen, 2003; Chen et al., 2003; Yen et al., 2005). The main idea of these schemes is to combine some basic encryption operations, under the control of a pseudorandom bit sequence (PRBS) generated by iterating a chaotic system. Unfortunately, most of Yen–Guo multimedia encryption schemes have been successfully cryptanalyzed (Li et al., 2005, 2006a,b, 2008a,b).

This paper reports a security analysis of MCS (multimedia cryptography system) – the latest multimedia encryption scheme proposed by Yen et al. (2005). Another hardware implementation of MCS was proposed in Chen et al. (2007). Compared with other earlier designs, such as RCES (Chen and Yen, 2003) and TDCEA (Chen et al., 2003), which have been cryptanalyzed in (Li et al., 2008b, 2005), MCS combines more encryption operations of different kinds in a more complicated manner, in the hope that the security can be effectively enhanced. This paper shows that MCS is still vulnerable to a differential chosen plaintext attack. Only seven chosen plaintexts (or six specific plaintext differentials) are enough to break MCS, with a divide-and-conquer (DAC) strategy.

The rest of this paper is organized as follows. Section 2 briefly introduces how MCS works. The proposed differential attack is

detailed in Section 3 with experimental results. Finally the last section concludes the paper.

2. Multimedia cryptography system (MCS)

MCS encrypts the plaintext block by block, and each block contains 15 bytes. As the first step of the encryption process, each 15-byte plain-block is expanded to a 16-byte one by adding a secretly selected byte. Then, the expanded block is encrypted with the following four different operations: byte-swapping (permutation), value masking, horizontal and vertical bit-rotations, which are all controlled by a secret PRBS.

Denote the plaintext by $F = (f(i))_{i=0}^{15}$, where $f(i)$ denotes the ith plaintext byte. Without loss of generality, assume that $N$ can be exactly divided by 15. Then, the plaintext has $N/15$ blocks: $f = f^{(15)}(k)_{k=0}^{N-1}$, where $f^{(15)}(k) = (f^{(15)}(k), 15)$. Similarly, denote the ciphertext by $F' = (f'(i))_{i=0}^{15}$, where $f'(k) = (f'(k), 15)_{i=0}^{N-1}$, which denotes the expanded ciphertext. With the above notations, MCS can be described as follows.

The secret key includes five integers $x_1, x_2, b_1, b_2$. Secret, and a binary fraction $x(0)$, where $1 < x_1 < x_2 + b_1, b_2 < 7, 1 < x_0 < x_2 + b_2 < 7$. $x(0) \in \{0, \ldots, 255\}$ and $x(0) = \sum_{i=0}^{64} x(i) 2^i$. $x(0)$ is selected from $\{1, 0\}$.

A PRBG (pseudorandom bit generator)

A pseudorandom number sequence $2^{N/15}$ is generated by iterating the following equation from $x(0)$:

$\begin{aligned}
\text{x(i + 1) = T(419(2^{i} x(i) \oplus H(x(i))) \bmod 2^{64}),}\n\text{where}
\text{x(0) = \sum_{i=0}^{64} x(i) 2^i, x(i) \in \{0, 1\}, H(x(i)) = \sum_{i=0}^{64} 2^{i} x(i) 2^i \oplus x(0) 2^i \oplus 0.1.}
\text{The controlling PRBG (b(i))_{i=0}^{2N/15-1-1} is derived from x(i)_{i=0}^{N/15-1} by extracting the 129 bits from each x(i + 10).}
\text{The above PRBG is a special case of the second class of chaos-based PRBG proposed in Kocarev and Jakimoski (2003), with the parameters p = 419, m = 8, M = k = 64.}
\text{The initialization process (1 run the above PRBG to generate the controlling PRBS (b(i))_{i=0}^{2N/15-1-1}; (2) set temp = Secret.}
\text{The encryption procedure (for each plain-block f^{(15)}(k), do the following operations consecutively:}
\text{- Step (a) Data expansion Add temp to the 15-byte plain-block to get an expanded 16-byte block f^{(15)}(k) = (f^{(15)}(k), 0), \ldots, f^{(15)}(k, 14), \text{temp}) and then set temp = f^{(15)}(k, k), where l(k) = \sum_{i=0}^{14} b(129 + i) + 2^i.}
\text{- Step (b) Byte-swapping Define a pseudorandom byte-swapping operation, Swaps, (129k + l + f^{(15)}(k), f^{(15)}(k)), which swaps f^{(15)}(k, i) and f^{(15)}(k, j) when b(129 + i) = 1. Then, perform the byte-swapping operation for the following 32 values of (i, l) one after another: (0, 8, 4), (1, 9, 5), (2, 10, 6), (3, 11, 7), (4, 12, 8), (5, 13, 9), (6, 14, 10), (7, 15, 11), (0, 4, 12), (1, 5, 13), (2, 6, 14), (3, 7, 15), (8, 12, 16), (9, 13, 17), (10, 14, 18), (11, 15, 19), (0, 2, 20), (1, 3, 21), (4, 6, 22), (5, 7, 23), (8, 10, 24), (9, 11, 25), (12, 14, 26), (13, 15, 27), (0, 1, 28), (2, 3, 29), (4, 5, 30), (6, 7, 31), (8, 9, 32), (10, 11, 33), (12, 13, 34), (14, 15, 35). Denote the permuted 16-byte block by f^{(15)}(k).
\text{- Step (c) Value masking Determine two pseudorandom variables, Seed1(k) = \sum_{i=0}^{15} (\pm b(129k + i + 4l) - 2^i) \text{ and Seed2(k) = \sum_{i=0}^{16} (\pm b(129k + 4l + 4i) - 2^i) - 2^i, and then do the following masking operation for j = 0 \ldots 7:}
\text{f'}^{(15)}(k) = f^{(15)}(k) \oplus \text{Seed1}(k, j),}
\text{where f^{(15)}(k) and f'}^{(15)}(k) \text{are composed of the jth bits of the 16 elements of f^{(15)}(k) and f^{(15)}(k), respectively,}
\text{and Seed(k, j) =}
\begin{align*}
\text{Seed1(k), & B(k, j) = 3,} \\
\text{Seed2(k), & B(k, j) = 2,}
\end{align*}
\text{B(k, j) = 1,}
\text{B(k, j) = 0.}
\text{Step (d) Horizontal bit rotation Construct an 8 \times 8 matrix M1 by assigning M1(i, j) as the j-th bit of f^{(15)}(k). Then, perform the following horizontal bit-rotation operations for i = 0, \ldots, 7 to get a new matrix M2:}
\text{M1(i, j) = Rotate^{2^j}(M1(i, j)), which shifts M1(i, j) (the j-th row of M1) by r_{1, j} elements (bit) to the left when p_{1, j} is 1 and to the right when p_{1, j} is 0. The values of the two parameters are as follows: p_{1, j} = b(129k + 65 + 2i) \text{, } r_{1, j} = x_1 + b_1 \cdot b(129k + 66 + 2i).}
\text{Similarly, the above process can be rewritten in the following way:}
\text{M1(i, j) = Rotate^{2i}(M1(i, j)),}
\text{where}
\text{r_{1, j} = x_1 + b_1 \cdot b(129k + 66 + 2i), p_{1, j} = b(129k + 65 + 2i) = 0,}
\text{p_{1, j} = b(129k + 65 + 2i) = 1.}
\text{In the following, we will use the latter form to simplify our further discussion.}
\text{In a similar way, construct another 8 \times 8 matrix M2 by assigning M2(i, j) as the j-th bit of f^{(15)}(k, 8 + i). Then, perform similar horizontal bit-rotation operations on M2 to get a new matrix M3:}
\text{M2(i, j) = Rotate^{2i}(M2(i, j)), where}
\text{r_{2, j} = x_1 + b_1 \cdot b(129k + 98 + 2i), p_{2, j} = b(129k + 97 + 2i) = 0,}
\text{p_{2, j} = b(129k + 97 + 2i) = 1. After the above horizontal bit-rotation operations, represent the ith byte in the 16-byte block as follows:}
\text{f^{(15)}(k, i) = \sum_{j=0}^{7} M3(i, j) \cdot 2^j, 0 \leq i \leq 7,}
\text{Step (e) Vertical bit rotation For i = 0, \ldots, 7, do the following vertical bit-rotation operations on M3 to get M4:}
\text{M4(i, j) = Rotate^{2^j}(M3(i, j)), which shifts M3(i, j) (the j-th column of M3) by s_{1, j} elements (bits) downwards. The value of the parameter is as follows:}
\text{several other notes...}
\[ s_{1,k} = \begin{cases} x_1 + \beta_1 \cdot b(129k + 82 + 2j), & q_{1,k} = b(129k + 81 + 2j) = 0, \\ 8 - (x_1 + \beta_1 \cdot b(129k + 82 + 2j)), & q_{1,k} = b(129k + 81 + 2j) = 1. \end{cases} \]

Similar vertical bit rotations are performed on \( \vec{M}_2 \) to get \( \vec{M}_2 \) as follows:

\[ s_{2,k} = \begin{cases} x_2 + \beta_1 \cdot b(129k + 114 + 2j), & q_{2,k} = b(129k + 113 + 2j) = 0, \\ 8 - (x_2 + \beta_1 \cdot b(129k + 114 + 2j)), & q_{2,k} = b(129k + 113 + 2j) = 1. \end{cases} \]

Finally, the cipher-block \( f^{(16)}(k) = (f^{(16)}(k, i))_{i=0}^{15} \), is derived from \( \vec{M}_1 \) and \( \vec{M}_2 \) as follows:

\[ f^{(16)}(k, i) = \begin{cases} \sum_{j=0}^{7} \vec{M}_1(i, j) \cdot 2^j, & 0 \leq i \leq 7, \\ \sum_{j=0}^{7} \vec{M}_2(i, j) \cdot 2^j, & 8 \leq i \leq 15. \end{cases} \]

Table 1

<table>
<thead>
<tr>
<th>Encryption process of the second 16-byte block of the image shown in Fig. 1a.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
<tr>
<td>( f^{(16)}(1) )</td>
</tr>
</tbody>
</table>

To show real performance of the above encryption scheme, a 512 × 512 plain-image “Peppers” and the corresponding cipher-image are shown in Fig. 1, where the randomly selected secret key is as follows: \( x_1 = 2, \beta_1 = 5, x_2 = 3, j_0 = 4. \) \( \text{Secr} \text{et} = 20, \) and \( x(0) = 0.251. \) Note that the cipher-image is 1/16 higher than the plain-image due to the data expansion. To show how MCS works more clearly, encryption process of the second 15-byte block of the image shown in Fig. 1a is shown in Table 1.

3. Cryptanalysis

First of all, we point out that the sub-key Secret has no influence on the plaintext recovered from the decryption process. It is because Secret is only used to determine the expanded byte, and never used to change the value of any other byte in the plaintext. In fact, if we use a different value of Secret for the decryption process, the plaintext can still be correctly recovered. Furthermore, the probability that Secret becomes the expanded byte of \( f^{(16)}(k) \) is \( (15/16)^k \), which decreases exponentially with respect to \( k \). As a consequence, we can simply ignore the (statistically tiny) influence of Secret on the encryption process after \( k \) become sufficiently large. As a whole, Secret should be excluded from the key. In the rest of this paper, we will not consider Secret as a sub-key.

3.1. Some properties of MCS

Define the XOR-differential ("differential" in short hereinafter) of two plaintexts \( f_0 \) and \( f_1 \) as \( f_{0 \oplus f_1} = f_0 \oplus f_1 \). When \( f_0 \) and \( f_1 \) are encrypted with the same secret key, it is easy to prove the following three properties of MCS, which will be the basis of the proposed attack.
**Property 1.** The random masking in Step (c) cannot change the differential value, i.e., \( \forall k, j, f_{0,1}^{16}(k, j) = f_{0,1}^{16}(k, j) \).

**Proof.** It is a straightforward result of the following property of XOR: \( (a \oplus x) \oplus (b \oplus x) = a \oplus b \). □

**Property 2.** Each expanded plain-block \( f_{0,1}^{16}(k) \) is independent of the sub-key Secret.

**Proof.** This can be proved with mathematical induction on \( k \). When \( k = 0 \) and \( 0 \leq j \leq 15 \), i.e., for the \( j \)th byte of the first 16-byte block, \( f_{0,1}^{16}(0, j) = \begin{cases} f_{0,1}^{16}(0, j), & 0 \leq j \leq 14, \\ \text{Secret} \oplus \text{Secret} = 0, & j = 15. \end{cases} \)

which is obviously independent of the value of Secret. Now assume the property holds for the first \( k - 1 \) blocks. Then, for the \( k \)th 16-byte block, \( f_{0,1}^{16}(k, j) = \begin{cases} f_{0,1}^{16}(k, j), & 0 \leq j \leq 14, \\ f_{0,1}^{16}(k, j - 1), & j = 15. \end{cases} \)

which is also independent from Secret according to the assumption. Thus, this property is proved. □

**Property 3.** The byte-swapping in Step (b) cannot change each differential value, but its position in the 16-byte block.

**Property 4.** Both the horizontal bit rotation in Step (d) and the vertical bit rotation in Step (e) cannot change each differential bit itself, but its position in the binary representation of the 8-byte block.

The proofs of the above two properties are straightforward, so we omit them here.

3.2. The differential attack

Based on the above properties of MCS, the data expansion in Step (a), the first 8-byte-swapping operations in Step (b), the vertical bit rotation in Step (e), the horizontal bit rotation in Step (d), the other unknown byte-swapping operations in Step (b) and the value masking in Step (c) can be broken in order with a number of chosen plaintext differentials.

3.2.1. Breaking the secret data expansion in Step (a)

To facilitate the following discussion, let us denote the Hamming weight of a byte or a block \( x \), i.e., the number of 1-bits in \( x \), by \( |x| \). From Properties 3 and 4, and the proof of Property 2, one can see that there are 8 \(- 15 = 120 \) binary blocks of \( f_{0,1}^{16}(k) \) come from \( f_{0,1}^{15}(k) \) and other eight comes from \( f_{0,1}^{15}(k - 1, l(k - 1)) \) for \( k \geq 1 \) (the eight expanded blocks are all 0-bits when \( k = 0 \)). Since all the other steps do not change the Hamming weight of each 16-byte block, we can get \( f_{0,1}^{15}(k - 1, l(k - 1)) \) is unique in the last 15-byte block \( f_{0,1}^{15}(k - 1, l(k - 1)) \). We can uniquely determine the value of \( k - 1 \). Considering \( f_{0,1}^{15}(k - 1, l(k - 1)) \in \{0, \ldots, 8\} \), at least two plain-bytes in each 16-byte block have the same Hamming weight. So, the value of \( l(k - 1) \) may not be uniquely determined sometimes. To make the unique determination of \( k - 1 \) possible, we can choose two plain-text differentials \( f_{0,1}^{15} \) and \( f_{0,2}^{15} \) (i.e., differentials of three chosen plaintexts \( f_0, f_1, \) and \( f_2) \) to fulfill the following two requirements: (1) \( \forall k, j_1 \neq j_2, f_{0,1}^{15}(k, j_1) = f_{0,2}^{15}(k, j_2) \), (2) \( \forall k, j \neq f_{0,1}^{15}(k, j) \), \( f_{0,2}^{15}(k, j) \neq (0, 0) \). For example, the two plaintext differentials can be chosen to have the following Hamming weights:

\[ f_{0,1}^{15}(k) = f_{0,1}^{15}(k - 1), \quad f_{0,2}^{15}(k) = f_{0,2}^{15}(k - 1) \cdot \]

With the above chosen plaintexts, it is obvious that the value of \( l(k - 1) \) can always be uniquely determined, except when

\[ f_{0,1}^{15}(k - 1) = f_{0,2}^{15}(k - 1) \cdot \]

The exception (4) occurs when \( l(k - n - 1) \neq 15 \) and \( l(k - n + 1) = l(k - n) + 1 = \cdots = l(k - 1) \), where \( n = |(0, 0, \ldots, 01)| \). Assuming that the secret bits controlling \( l(k - n + 1), l(k - n) \ldots l(k - 1) \) distribute uniformly over \( \{0, 1\} \), the occurrence probability of the exception is less than \( \frac{1}{2^{15}} = \frac{1}{1.4305 \times 10^{15}} \). For a 512 x 512 image, this means that we will not be able to uniquely determine the value of \( l(k - 1) \) for less than 1.4305 x 10^15 blocks in an average sense. In other words, the value of \( l(k - 1) \) can be uniquely determined for almost all blocks. Note that breaking \( l(k - 1) \) implies breaking 4 controlling bits \( (b(129(k - 1) + 1))_{i} \).

3.2.2. Breaking the first 8-byte-swapping operations in Step (b)

From Properties 3 and 4, one can see that all the \( 8 - 16 = 128 \) bits of each 16-byte expanded plain-block \( f_{0,1}^{16}(k) \) are the same as the ones of the corresponding 16-byte cipher-block \( f_{0,1}^{15}(k) \), except that their locations may change. Observing how the bit locations are changed in the whole encryption process, we can see the following 8-byte-swapping operations are the only encryption operations moving bits from one 8-byte half-block to another: Swap3d_2(129k+i+4)(F(16k,i); f_{0,1}^{16}(k,i)), when \( i = 0, 1, 2, 3, 4, 5, 6, 7 \). Apparently, when the controlling bit is 1, each byte-swapping operation swaps the locations of one byte in the first half-block and the other byte in another half-block. This fact means that, by choosing the differences between the Hamming weights of the 8-bytes in the two half-blocks properly, we will be able to derive the values of the controlling bits \( (b(129k + i + 4))_{i, j} \). The simplest tactic is to choose \( f_{0,1}^{16}(k) \) such that each half-block has only one byte with a different Hamming weight from the corresponding byte in the other half-block. If we assume all the values of \( (l(k))_{i} \) have been recovered, which happens with high probability as shown in the previous subsection, the first 15 bytes in \( f_{0,1}^{16}(k) \) can be freely chosen by \( f_{0,1}^{15}(k) \). The last byte in
each 16-byte block \( f^{(16)}_{0:1}(k, 15) \) may not be chosen, if it is equal to \( \text{Secret} \). Fortunately, this has no influence on the process of breaking the first 8-byte-swapping operations, because what is chosen for the last byte is \( f^{(16)}_{0:1}(k, 15) \). Although we may not be able to choose \( f^{(16)}_{0:1}(k, 15) \), we can always choose \( f^{(7)}_{0:1}(k, 7) \) to have a different Hamming weight from that of \( f^{(7)}_{0:1}(k, 7) \). One chosen-block \( f^{(7)}_{0:1}(k) \) will be able to derive the value of one controlling bit, which controls the possible swapping of the two bytes (in two half-blocks, respectively) with different Hamming weights. We need eight chosen plain-blocks (thus eight chosen plaintext differentials) to determine the values of all the eight controlling bits.

While eight chosen plaintext differentials are enough to recover all the bits controlling the first 8-byte-swapping operations, we actually need only two chosen plaintext differentials to achieve this goal. To see how it is possible, denote the difference between the Hamming weights of the two half-blocks of the 4th cipher-block by \( |f^{(16)}_{0:1}(k) - a| \). Then, we have the following equation:

\[
A \left( f^{(16)}_{0:1}(k) \right)^7 \bigg|_{i=0} = \left( f^{(16)}_{0:1}(k, i) \right)^7 \bigg|_{i=0} - \left( f^{(16)}_{0:1}(k, i + 8) \right)^7 \bigg|_{i=0} = \left( f^{(16)}_{0:1}(k, i) \right)^7 \bigg|_{i=0} - \left( f^{(16)}_{0:1}(k, i + 8) \right)^7 \bigg|_{i=0} = \sum_{i=0}^{7} \left( f^{(16)}_{0:1}(k, i) - f^{(16)}_{0:1}(k, i + 8) \right) \bigg|_{i=0} = \sum_{i=0}^{7} b^+(k, i) \left( f^{(16)}_{0:1}(k, i) - f^{(16)}_{0:1}(k, i + 8) \right) \bigg|_{i=0},
\]

where

\[
b^+(k, i) = 1 - 2b(129k + i + 4) = \begin{cases} 1, & b(129k + i + 4) = 0; \\ -1, & b(129k + i + 4) = 1. \end{cases}
\]

By choosing the values of \( \left( f^{(16)}_{0:1}(k, i) - f^{(16)}_{0:1}(k, i + 8) \right)^7 \bigg|_{i=0} \) to be a set of numbers such that every nonzero number can be represented as a linear combination of other numbers in the set, the controlling bits corresponding to the nonzero numbers can be determined uniquely. For instance, to determine the values of \( b^+(k, 0), \ldots, b^+(k, 3) \), we can choose a plaintext differential such that

- \( f^{(16)}_{0:1}(k, i) - f^{(16)}_{0:1}(k, i + 8) = \pm 4, \pm 5, \pm 6, \pm 8 \) for \( i = 0, 1, 2, 3 \), respectively;
- \( f^{(16)}_{0:1}(k, i) - f^{(16)}_{0:1}(k, i + 8) = 0 \) for \( i = 4, 5, 6, 7 \).

The above chosen plaintext differential leads to the following result:

\[
A \left( f^{(16)}_{0:1}(k) \right)^7 \bigg|_{i=0} \in \{ \pm 23, \pm 15, \pm 13, \pm 11, \pm 7, \pm 5, \pm 3, \pm 1 \}.
\]

The 16 possible values of \( A \left( f^{(16)}_{0:1}(k) \right)^7 \bigg|_{i=0} \) correspond to the 16 possible values of \( b(129k + 4 + i) \). Choosing another plaintext differential such that

- \( f^{(16)}_{0:1}(k, i) - f^{(16)}_{0:1}(k, i + 8) = 0 \) for \( i = 0, 1, 2, 3 \);
- \( f^{(16)}_{0:1}(k, i) - f^{(16)}_{0:1}(k, i + 8) = \pm 4, \pm 5, \pm 6, \pm 8 \) for \( i = 4, 5, 6, 7 \), respectively,

we will be able to uniquely determine the other four controlling bits \( \{b(129k + 4 + i)\}_{i=4}^{7} \). As a whole, with only two chosen plaintext differentials, we can uniquely determine all the eight controlling bits \( \{b(129k + 4 + i)\}_{i=0}^{7} \).
3.2.4. Performance of the differential attack

We choose different values of the ESS are related to the unknown parameters observing the above four results, we can see all the four parts, we can get the ESS will move it to \( f_{0/0}^{(16)}(k, i) \). Similarly, for the second half-block, the real byte-swapping operation moves \( f_{0/0}^{(16)}(k, i) \) to \( f_{0/0}^{(16)}(k, 8 + 3_2) \), the one we obtained for the ESS will move it to \( f_{0/0}^{(16)}(k, 8 + 3_2) \).

3.2.3.3. Obtaining the byte-swapping part of the EES. After obtaining the byte-horizontal/vertical bit-rotation parts of the ESS, we can apply the inverse horizontal/vertical bit rotations to \( f_{0/0}^{(16)}(k, j) \) to get \( f_{0/0}^{(16)}(k, 3_3 + j) \) and \( f_{0/0}^{(16)}(k, 8 + (3_2 k + i)) \). If we choose \( f_{0/0}^{(16)}(k, i) \) such that all the 8-bytes of each half-block are different from each other, we will be able to obtain the following byte-swapping part of the ESS. For the first half-block, the real byte-swapping operation moves \( f_{0/0}^{(16)}(k, i) \) to \( f_{0/0}^{(16)}(k, 3_3) \), the one we obtained for the ESS will move it to \( f_{0/0}^{(16)}(k, 8 + 3_2) \).

3.2.3.4. Obtaining the value masking part of the ESS. After obtaining the byte-swapping part of the ESS, we can get \( f^{(16)}(k, i + 3_1) \) and \( f^{(16)}(k, 8 + (i + 3_1)) \) from any known plaintext. In addition, after obtaining both the horizontal and vertical bit-rotation parts, we can get \( f^{(16)}(k, i + 3_1) \) and \( f^{(16)}(k, 8 + (i + 3_1)) \) from any known ciphertext. We do not need to choose more plaintexts, but can simply reuse any chosen plaintext used in previous steps. Note that the value masking performed in Step (c) can be rewritten as the equivalent form: for \( i = 0, \ldots, 15 \),

\[
f^{(16)}(k, i) = f^{(16)}(k, i) \oplus \text{Seed}'(k, i),
\]

where \( \text{Seed}'(k, i) = \sum_{a=0}^{15} \text{Seed}(k, j), j = 2^i \) and \( \text{Seed}(k, j) \) is the \( i \)-th bit of \( \text{Seed}(k, j) \). Then, by XORing \( f^{(16)}(k, i + 3_1) \) and \( f^{(16)}(k, 8 + (i + 3_1)) \), we get \( \text{Seed}(k, 8 + (i + 3_1)) \). Similarly, by XORing \( f^{(16)}(k, 8 + (i + 3_1)) \) and \( f^{(16)}(k, 0 + (i + 3_1)) \), we get \( \text{Seed}(k, 8 + (i + 3_1)) \).

Observing the above four results, we can see all the four parts of the ESS are related to the unknown parameters 3_1 and 3_2. If we choose different value of \( a \) in Section 3.2.3.1, we may have a different ESS. All the possible ESS are equivalent to each other (and to the real encryption scheme), so we can use any of them to decrypt any ciphertext encrypted with the same key, as long as the size of the ciphertext is not larger than \( N \). In the next subsection, we will show the values of 3_1 and 3_2 can be uniquely determined if the sub-keys 3_1, 3_2, \( \beta_1 \) and \( \beta_2 \) satisfy some requirements.

3.2.4. Performance of the differential attack

To sum up, the differential attack outputs the following items as an equivalent key:

- for data expansion: \( (k \oplus 1)_{c(k/N)} \) which is equivalent to \( (b \oplus 129k)_{c(k/N)} \) and \( (b \oplus 129k)_{c(k/N)} \);
- for the first 8-byte-swapping operations \( (b \oplus (129k + i))_{c(k/N)} \);
- for the vertical bit rotations:
  \[
  (\text{Rotate}_{\mathbb{G}_{11k}})(k, i) \quad 0 \leq k < N/15 - 1
  \]
- for the horizontal bit rotations:
  \[
  (\text{Rotate}_{\mathbb{G}_{11k}})(k, i) \quad 0 \leq k < N/15 - 1
  \]
  \[
  0 \leq j < 7
  \]

All the above items form an encryption system equivalent to MCS and can be used to decrypt any ciphertexts encrypted with the same secret key. The (equivalent) encryption operations performed on some expanded bytes \( f^{(16)}(k, 15) \) may not be recovered, but which does not influence the effectiveness of the differential attack, since those expanded bytes will finally be discarded.

The total number of chosen plaintexts is the sum of the following:

- (a) two differentials for breaking the data expansion; (b) two differentials for breaking the first 8-byte-swapping operations; (c) four differentials for obtaining the ESS. Note that the plaintext differential needed in Section 3.2.3.3 can be replaced by the two differentials in Section 3.2.1. So, we only need two more differentials for obtaining the ESS. As a whole, the differential attack requires \( 2 + 2 + 2 + 6 = 12 \) plaintexts or seven plaintexts, to break MCS.

The complexity of the differential attack is also very small. In each step, the equivalent sub-key can be directly derived from the plaintext and the ciphertext, so the complexity is proportional to the size of the plaintext. With 6 chosen plaintext differentials, the computational complexity of the attack is just \( O(6N) = O(N) \), which is the same as that of the normal encryption/decryption process of MCS.

3.3. Breaking some sub-keys and more controlling bits

The differential attack described in the previous subsection outputs an equivalent key, which include some controlling bits \((b \oplus 129k + i)_{c(k/N)} \), but does not include any part of the secret key. In this subsection, we show we may further derive more controlling bits and the following four sub-keys: 3_1, \( \beta_1 \), \( \beta_2 \) and \( \beta_3 \). Although we have not found a way to break the underlying pseudorandom bit generator (PRBG) and then break the sub-key \( \chi(0) \), breaking more controlling bits makes it easier to analyze more potential weaknesses of the PRBG and opens the door to a successful cryptanalysis in future.

We first try to break the two sets \( \mathbb{R}_1 = (x_1, 8 - x_1, x_1 + \beta_1, 8 - (x_1 + \beta_1)) \) and \( \mathbb{R}_2 = (x_2, 8 - x_2, x_2 + \beta_2, 8 - (x_2 + \beta_2)) \). Then, we may be able to further determine sub-keys 3_1, \( \beta_1 \), \( \beta_2 \), 3_2 and more controlling bits.
3.3.1. Breaking R₁ and R₂

In the differential attack, what we have obtained for the horizontal bit rotations are \((\text{Rotate}_{8j(1:k+1:ka)}) 0 \leq k < N/15 - 1\) and 0 \leq j \leq 7. \((\text{Rotate}_{8j(1:k+2:ka)}) 0 \leq k < N/15 - 1\) and according to how \(r_{1,k}\) and \(r_{2,k}\) are determined, it is obvious that \(R_{1,k} = (r_{1,k+i+1:ka})^7 \odot R_1\) and \(R_{2,k} = (r_{2,k+i+1:ka})^7 \odot R_2\).

3.3.2. Determining sub-keys \(x_1, \beta_1, x_2\) and \(\beta_2\)

After getting \(R_1\) and \(R_2\), the four sub-keys \(x_1, \beta_1, x_2\) and \(\beta_2\) may be uniquely determined. Following a similar process of the proof of Proposition 1, we consider the following three cases for \(m = 1, 2, 3\).

- \#(R_{m}) = 2: This case happens only when \(2x_m + \beta_m = 8\). There are three possible sets \(R_m = \{1, 7\}, \{2, 6\}, \{3, 5\}\), which corresponds to \(x_m, \beta_m = \{1, 6\}, \{2, 4\}, \{3, 2\}\), respectively. Apparently, knowing \(R_m\) allows us to uniquely determine the values of \(x_m, \beta_m\).

- \#(R_{m}) = 3: This case happens when \(x_m = 8 - x_m = 4\) or \(x_m + \beta_m = 8 - (x_m + \beta_m) = 4\). There are only possible sets \(R_m\), each of which corresponds to two possible values of \(x_m, \beta_m\): - \(R_m = \{4, 1, 7\}\) : \(x_m, \beta_m = \{4, 3\}\) or \(\{3, 1\}\); - \(R_m = \{4, 2, 6\}\) : \(x_m, \beta_m = \{4, 2\}\) or \(\{2, 2\}\); - \(R_m = \{4, 3, 5\}\) : \(x_m, \beta_m = \{4, 1\}\) or \(\{1, 3\}\). It can be seen that \(x_m, \beta_m\) cannot be uniquely determined in this case.

- \#(R_{m}) = 4: This case includes three possible sets \(R_m\), each of which corresponds to four different values of \(x_m, \beta_m\):

\[
R_m = \{1, 2, 5, 7\} : \{x_m, \beta_m\} = \{1, 1\}, \{1, 5\}, \{2, 5\}, \{6, 1\};
\]

\[
R_m = \{1, 3, 5, 7\} : \{x_m, \beta_m\} = \{1, 2\}, \{1, 4\}, \{3, 4\}, \{5, 2\};
\]

\[
R_m = \{2, 3, 5, 6\} : \{x_m, \beta_m\} = \{2, 1\}, \{2, 3\}, \{3, 3\}, \{5, 1\}.
\]

3.3.3. Determining \(\tilde{s}_{1,ka}\) and \(\tilde{s}_{2,ka}\)

In the differential attack, what we have obtained for the bit rotations are \((\text{Rotate}_{8j(1:k+1:ka)}) 0 \leq k < N/15 - 1\) and \((\text{Rotate}_{8j(1:k+2:ka)}) 0 \leq k < N/15 - 1\).

According to how \(s_{1,k}\) and \(s_{2,k}\) are determined in the encryption process, we can get \(s_{1,k} = (s_{1,k+1} + s_{1,k+2})^7 \odot s_{1,k}\) and \(s_{2,k} = 2s_{2,k+1} + (s_{2,k+2} + s_{2,k+3} - 2s_{2,k} + s_{2,k+2})\). Comparing \(s_{1,k}\) with \(s_{2,k}\), we may be able to determine the values of \(\tilde{s}_{1,ka}\) and \(\tilde{s}_{2,ka}\). There are four different cases:

- \(S_{n,k} \subseteq \tilde{s}_{1,ka}\): If \(S_{n,k}\) does not contain all elements in \(\tilde{s}_{1,ka}\), it is generally impossible to uniquely determine \(\tilde{s}_{1,ka}\). From Proposition 1, the occurrence probability of this case is \(2^2/2^8 = 1/2^6\).

- \(S_{n,k} \cap \tilde{s}_{1,ka} = \emptyset\): When \(S_{n,k} \in \{1, 2, 3, 5, 6, 7\}\), its value can be uniquely determined. When \(S_{n,k} = 0\) or 4, it is impossible to distinguish one value from the other.

- \(S_{n,k} \cap \tilde{s}_{1,ka} \neq \emptyset\) and \(R_n = \{1, 7\}, \{3, 5\}, \{4, 1, 7\}, \{4, 2, 6\}, \{4, 3, 5\}, \{1, 2, 6, 7\}\) or \{2, 3, 5, 6\}: The value of \(S_{n,k}\) can always be uniquely determined.

- \(S_{n,k} \subseteq \tilde{s}_{2,ka}\) and \(R_n = \{1, 3, 5, 7\}\): The value of \(S_{n,k}\) can never be uniquely determined. One can only determine which of the following two sets \(S_{n,k}\) belongs to: \{0, 2, 4, 6\} and \{1, 3, 5, 7\}.

Assuming the value of \(S_{n,k}\) distributions uniformly over \(\{0, \ldots , 7\}\), the probability that each \(S_{n,k}\) cannot be uniquely determined is \(1/2^2/[(1/2^2) + (1/2^2)] = 0.2086\). We may choose more different values of \(a\) in Section 3.2.3.1 to decrease this probability, but the probability has a lower bound \(1/2^2/[(1/2^2) + (1/2^2)] = 0.1968\). We can see this probability is always not sufficiently small, so we will not be able to uniquely determine the value of \(s_{1,ka}\) or that of \(s_{2,ka}\) for quite a lot of blocks.
3.3.4. Determining the secret bits controlling the 9th to 35th byte-swapping operations

In case $s_{1,k.a}$ and $s_{2,k.a}$ can be uniquely determined, we will be able to uniquely recover the 9th to 35th byte-swapping operations, i.e., we can determine the values of $(s_{1,k})_{i}^{63-0}$ and $(s_{2,k})_{i}^{63-0}$. Note that $(s_{1,k})_{i}^{63-0}$ and $(s_{2,k})_{i}^{63-0}$ actually define two permutation maps over $\{0, \ldots, 7\}$. Observing the 9th to 35th byte-swapping operations in Step (b), one can notice that the permutation map has a strong pattern: 12 byte-swapping operations for the first half-block and the other 12 ones for the second half-block, and each group of 12 byte-swapping operations can be divided into three phases. For the 12 byte-swapping operations performed on the first half-block, the three phases are as follows:

- **Phase 1**: $i, j, a = (0, 4, 12), (1, 5, 13), (2, 6, 14), (3, 5, 15);
- **Phase 2**: $i, j, a = (0, 2, 20), (1, 3, 21), (4, 6, 22), (5, 7, 23);
- **Phase 3**: $i, j, a = (0, 1, 28), (2, 3, 29), (4, 5, 30), (6, 7, 31).

Appropriately, Phase 1 swaps the bytes in the two 4-byte quarter-block of the first 8-byte half-block, and Phases 2 and 3 only permute the bytes with each 4-byte quarter-block. Then, for $i = 0, 1, 2, 3$, we can check in which quarter-block $f_{i,129}(k)$ belongs to after the byte-swapping operations. In other words, we check if $s_{1,k} \in \{0, 1, 2, 3\}$ or $\{4, 5, 6, 7\}$, which corresponds to $b_{129k + 12} = 0$ and 1, respectively. This allows us to completely determine $(b_{129k + 12 + i})_{i}^{63-0}$, i.e., to break Phase 1. Then, we can derive a new permutation map represented by $(s_{1,k})_{i}^{63-0}$, which consists of only Phases 2 and 3. Then, according to the byte-swapping operations involved in Phases 2 and 3, we can derive the following rule to break the 4 controlling bits involved in Phase 2:

- **when $i = 0$$\text{, } b_{129k + 20 + i} = 0$$\text{, } s_{1,k} \in \{0, 1\};$$\text{, } s_{2,k} \in \{2, 3\};$
- **when $i = 2$$\text{, } b_{129k + 20 + i} = 1$$\text{, } s_{1,k} \in \{4, 5\};$$\text{, } s_{2,k} \in \{6, 7\}.$

After breaking both Phases 1 and 2, we can immediately break the 4 controlling bits $(b_{129k + 28 + i})_{i}^{63-0}$ involved in Phase 3. Now, we completely break all the 12 controlling bits involved in the byte-swapping operations performed on the first half-block. The same process can be applied to the second half-block, and 12 controlling bits can be uniquely determined. As a whole, we will be able to break all the 24 controlling bits $(b_{129k + j})_{i}^{63-12}$.

3.3.5. Determining the secret bits controlling value masking

In case $s_{1,k.a}$ and $s_{2,k.a}$ can be uniquely determined as described in Section 3.3.3, we will be able to determine $(\text{Seed}(k, j))_{i}^{63-0}$, or equivalently, $(\text{Seed}(k, j))_{i}^{63-0}$. This allows us to obtain $(\text{Seed}(k, j))_{i}^{63-0} \subset (\text{Seed}(k, j))_{i}^{63-0} \subset (\text{Seed}(k, j))_{i}^{63-0}$. To break the controlling bits, we need to recover $\text{Seed}(k, j)$ and Seed(2), which are calculated from $(b_{129k + j})_{i}^{63-0}$ and $(b_{129k + 64 + j})_{i}^{63-0}$, respectively. Note that we can always break $(b_{129k + j})_{i}^{63-0}$ if $s_{1,k.a}$ and $s_{2,k.a}$ are uniquely determined. This means that we can break the 36/4 = 9 least significant bits (LSBs) of Seed(k), since each bit of Seed(1) is determined by four controlling bits. Then, if the nine LSbs of Seed(1) are not all equal to those of Seed(2) or those of Seed(1), we can uniquely determine Seed(1) and then Seed(2). Assuming Seed(2) and Seed(2) are independent of each other and each bit distributes uniformly over $\{0, 1\}$, the probability that Seed(1) cannot be uniquely determined is $2^{-2}$.

In case Seed(1) is uniquely determined, we have the following results:

- When Seed($k$, $j$) $\in$ Seed(1), Seed(2):

  $b_{129k + 36 + 2j} = 1$; $b_{129k + 37 + 2j} = \begin{cases} 0, & \text{Seed}(k, j) = \text{Seed}(k), \\ 1, & \text{Seed}(k, j) = \text{Seed}(2). \end{cases}$

- When Seed($k$, $j$) $\notin$ Seed(1), Seed(2):

  $b_{129k + 36 + 2j} = 1$; $b_{129k + 37 + 2j} = \begin{cases} 0, & \text{Seed}(k, j) = \text{Seed}(k), \\ 1, & \text{Seed}(k, j) = \text{Seed}(2). \end{cases}$

Note that in this case, Seed(2) has to be guessed from the set $(\text{Seed}(k, j))_{i}^{63-}$. If $s_{1,k.a}$ and $s_{2,k.a}$ can be uniquely determined as described in Section 3.3.3, we will be able to uniquely determine the horizontal and vertical bit rotations exerted on $M_{i}$, $M_{j}$, $M_{k}$, and $M_{l}$.

Depending on how well the values of $x_{1}$, $x_{2}$, $x_{3}$, and $x_{4}$ are determined in Section 3.3.2, some information about the controlling bits involved in the bit rotations may be obtained, although it is always impossible to uniquely determine the value of any controlling bit involved. Since the determination process of the controlling bits is similar for $M_{i}$, $M_{j}$, $M_{k}$, and $M_{l}$, we consider only the case of $M_{i}$ (i.e., horizontal bit rotations exerted on the first half-block) to simplify the discussion. For the case, we get $(r_{i,1,k})_{i}^{63-0}$ by substituting $k_{i}^{63-0}$ into $(r_{i,1,k})_{i}^{63-0}$. In Step (d), $r_{i,1}$ is determined by two controlling bits as follows:

- **$r_{i,1} = (1, 7), (2, 6)$ or (3, 5)**: In this case, $x_{2}$ and $x_{3}$ can be uniquely determined, but we cannot differentiate $x_{1}$ from $x_{3}$, and $x_{2}$ from $x_{4}$. Hence, we can determine neither $b_{129k + 65 + 2i}$ nor $b_{129k + 66 + 2i}$, just the following:

  - $(b_{129k + 65 + 2i}), b_{129k + 66 + 2i}) = \begin{cases} (0, 0) \text{ or } (1, 1), & r_{i,1,k} \in \{1, 2, 3\}, \\ (0, 1) \text{ or } (1, 0), & r_{i,1,k} \in \{5, 6, 7\}. \end{cases}$

- **$r_{i,1} = (4, 1), (4, 2, 6)$ or (3, 5)**: In this case, $x_{2}$ and $x_{3}$ have two possible values, so $b_{129k + 65 + 2i}$ and $b_{129k + 66 + 2i}$ cannot be uniquely determined. What we can get is the following:

  - $(b_{129k + 65 + 2i}), b_{129k + 66 + 2i}) = \begin{cases} (0, 0) \text{ or } (1, 1), & r_{i,1,k} \in \{1, 2, 3\}, \\ (0, 1) \text{ or } (1, 0), & r_{i,1,k} \in \{5, 6, 7\}, \\ (0, 1) \text{ or } (1, 0) \text{ or } (1, 1), & r_{i,1,k} = 4, \end{cases}$

- **$r_{i,1} = (1, 2, 6, 7)$**: In this case, $x_{2}$ and $x_{3}$ have four possible values, $(1, 1), (1, 5), (2, 5)$, or $(6, 1)$, so $b_{129k + 65 + 2i}$ and $b_{129k + 66 + 2i}$ cannot be uniquely determined. What we can get is the following:

  - $(b_{129k + 65 + 2i}), b_{129k + 66 + 2i}) = \begin{cases} (0, 0) \text{ or } (1, 1), & r_{i,1,k} = 1, \\ (0, 0) \text{ or } (1, 0), & r_{i,1,k} = 7, \\ (0, 0, 1) \text{ or } (1, 0) \text{ or } (1, 1), & r_{i,1,k} \in \{2, 6\}. \end{cases}$

- **$r_{i,1} = (1, 3, 5, 7)$**: In this case, $x_{2}$ and $x_{3}$ have four possible values, $(1, 2), (1, 4), (3, 4)$, or $(5, 2)$, so $b_{129k + 65 + 2i}$ cannot be uniquely determined, either. What we can get is the following:
The two plaintext differentials for breaking the first 8-byte-swapping operations.

Fig. 4. The two plaintext differentials for obtaining the vertical and horizontal bit-operation part of the EES: (a) vertical bit-operation; (b) horizontal bit-operation.

To verify the real performance of the differential attack proposed in this paper, some experiments were carried out with the secret key used in Section 2. The plain-image shown in Fig. 1a is used as one of the chosen plaintexts to generate the required chosen plaintext differentials. The two differentials used for breaking secret data expansion are shown in Fig. 2. The two differentials used for breaking the first 8-byte-swapping operations, i.e., the secret bits \( b(129k + i) \) \( 0 \leq k \leq N/15 - 1 \), are shown in Fig. 3. The two differentials shown in Fig. 4 and those two shown in Fig. 2 were used to obtain an EES. The recovered equivalent key (i.e., all the items shown in Section 2.4) was used to encrypt a cipher-image as shown in Fig. 5a. The result is given in Fig. 5b. It can be seen that the secret plain-image was successfully recovered by the differential attack. To show the breaking process more clearly, the

<table>
<thead>
<tr>
<th>Obtained items</th>
<th>The corresponding values</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b(129 + 4) - b(129 + 11) )</td>
<td>5</td>
</tr>
<tr>
<td>( f(1, i) ) ( i \in {1, 2, 3, 4, 6, 12, 24} )</td>
<td>2, 4, 6, 6, 6, 6, 6, 6, 6</td>
</tr>
<tr>
<td>( f(1, i) ) ( i \in {1, 2, 3, 4, 6, 12} )</td>
<td>6, 2, 2, 6, 6, 6, 6</td>
</tr>
<tr>
<td>( f(1, i) ) ( i \in {1, 2, 3, 4, 6, 12} )</td>
<td>7, 6, 1, 6, 7, 7, 2</td>
</tr>
<tr>
<td>( f(1, i) ) ( i \in {1, 2, 3, 4, 6, 12} )</td>
<td>4, 3, 0, 5, 1, 2, 6, 7</td>
</tr>
<tr>
<td>( f(1, 8, i) ) ( i \in {1, 2, 3, 4, 6, 12} )</td>
<td>4, 5, 3, 1, 0, 7, 6, 2</td>
</tr>
<tr>
<td>( f(1, i) ) ( i \in {1, 2, 3, 4, 6, 12} )</td>
<td>55, 228, 200, 200, 200, 200, 200, 27</td>
</tr>
<tr>
<td>( f(1, i) ) ( i \in {1, 2, 3, 4, 6, 12} )</td>
<td>27, 55, 228, 55, 27, 27, 27, 27, 27</td>
</tr>
</tbody>
</table>

Fig. 2. The two plaintext differentials for breaking data expansion.

items determining the equivalent secret key of the second 16-byte block of cipher-image are shown in Table 2 also.

4. Conclusion

In this paper, we evaluate the security of a recently-proposed multimedia encryption system called MCS (Yen et al., 2005), and propose a differential attack to break it with a divide-and-conquer (DAC) strategy. The differential attack is very efficient in the sense that only seven chosen plaintexts are needed to get an equivalent key and the computational complexity is only O(N), where N is the number of bytes in the plaintext. The real performance of the proposed attack was also verified with experiments. Similar to some other image encryption schemes proposed in the literature, the MCS was not designed by following some good principles of designing such systems. Some of these principles are discussed in (Alvarez and Li, 2006; Li et al., 2008b).

Acknowledgements

Chengqing Li was supported by The Hong Kong Polytechnic University’s Postdoctoral Fellowships Scheme under Grant No. G-XXL. Shujun Li was supported by a fellowship from the Zukunfts-kolleg of the University of Konstanz, Germany, which is part of the “Excellence Initiative” Program of the DFG (German Research Foundation). The work of Kwok-Tung Lo was supported by the Research Grant Council of the Hong Kong SAR Government under Project 523206 (PolyU 5232/06E).

References


