An Image Hiding Scheme by Linking Pixels in the Circular Way

Chi-Shiang Chan, Yuan-Yu Tsai* and Chao-Liang Liu
Department of Applied Informatics and Multimedia,
Asia University,
Wufeng, Taiwan 41354, R. O. C.
[e-mail: {CSChan, yytsai,jliu}@asia.edu.tw]
*Corresponding author: Yuan-Yu Tsai

Received April 2, 2011; revised May 24, 2012; accepted June 13, 2012; published June 25, 2012

Abstract

The proposed method in this paper is derived from Mielikainen's hiding method. However, there exist some significant differences between two methods. In Mielikainen's method, pixels are partitioned into pairs and a LSB matching function is applied to two pixels for hiding. On the contrary, the proposed method partitions pixels into groups with three pixels in each group. The bits of pixels in each group are linked by using an exclusive OR (XOR) operator in a circular way. If the number of different values between the calculated XOR values and the secret bits is smaller than or equal to 2 in a group, the proposed method can guarantee that at most one pixel is needed to be modified by adding/subtracting its value to/from one, and three secret bits can be embedded to three pixels. Through theoretical analysis, the amount of the embedded secret data in the proposed method is larger than those in other methods under the same amount of pixel modifications. Taking real images in our experiments, the quality of stego-images in the proposed method is higher than those in other methods.

Keywords: Information hiding, LSB matching, LSB matching revisited

This work was partially supported by the research project of Asia University under grant number 100-ASIA-38.

http://dx.doi.org/10.3837/tiis.2012.06.013

1. Introduction

Digital Steganography is a technique that embeds secret data into meaningful multimedia data, such as videos and images. The hiding techniques can be classified three domains roughly according to the types of images that are used to carrier secret data. They are frequency-based image hiding, VQ-based image hiding and pixel-based image hiding. Frequency-based image hiding and VQ-based image hiding embed secret data to AC coefficients and VQ indices, respectively. On the contrary, pixel-based image hiding directly embeds secret data into pixels.

Generally speaking, pixel-based image hiding has the largest hiding capacity among three domains. When talking about methods in this domain, the least-significant-bit (LSB) substitution method is the most straightforward. The method replaces the least-significant bits of cover pixels with secret data directly. Although LSB substitution method is the simplest way to hide secret data into digital images, it also degrades the quality of digital images most.

In order to alleviate the degradation, Wang et al. [9] proposed their method by transforming the values of secret bits into other values. Wang et al. also proposed the way to find the mapping relations that could help us transform the values of secret bits to other values. After transforming, the transformed values become closer to the bits of the host pixel that are used to embed secret data. Therefore, host pixels will not be degraded too much after embedding. Wang et al. proposed their method to find the approximately optimal mapping relations by using a genetic algorithm. In 2003, Chang et al. [2] used the dynamic programming strategy to find the optimal mapping relations. The results shown in Chang et al.'s method reveal that both the computation time and the quality of stego-images are better than those of Wang et al.'s method. Then, Thien and Lin [7] proposed their image-hiding method by using a modulus function to further reduce the degradation of the image quality after embedding.

Although all above-mentioned methods have high embedding capacity with good image quality, the secret data may be detected [3][4]. In 2006, Mielikainen proposed his method that used LSB Matching Function to do image hiding [6]. ielikainen's method partitions cover pixels into pixel pairs. Two secret bits are embedded to a pixel pair. The first secret bit is embedded in the LSB of the first pixel. And, the second secret bit is the result of LSB Matching Function which takes two pixels as the parameters. To make the extracted data equal to secret data, Mielikainen's method only needs to modify at most one pixel for each pixel pair by adding/subtracting the pixel value to/from one. Owing to this property, the Mielikainen's method can obtain the best results among above-mentioned methods under the condition that the embedding capacity is 1 bit per pixel.

Although there are some methods that can modify fewer pixels than Mielikainen's method can do, their embedding capacity is not as large as Mielikainen's method. For example, Westfeld [8] implemented the Matrix Coding to reduce the amount of modified pixels. In spite of fewer modified pixels of Westfeld's method, the embedding capacity is only three-sevenths of the total amount of the cover pixels. Furthermore, Zhang et al. [11] proposed their "Hamming+1" Scheme to reduce the amount of modified pixels. However, the embedding capacity is still lower than Mielikainen's method. Having high embedding capacity and satisfied amount of modified pixels makes Mielikainen's method superior to other methods.

Owing to the advantage described above, many papers do their further researches based on Mielikainen's method. For example, Chan's method [1] replaces LSB Matching Function with

Exclusive OR (XOR) Function to embed secret data. The main property of Chan's method is that his method dose not partition pixels into pixel pairs but uses XOR Function to link all pixel bits. If the different values between the calculated values of LSB Matching Function and the secret bits occur across pairs, Mielikainen's method must modify two pixels when embedding. On the contrary, Chan's method only needs to modify one pixel. After that, in 2011, Lin was inspired from Mielikainen's method to propose his method [5] which partitions pixels into groups with three pixels in each group. Differences between three pixels are used to hide secret data.

In order to further reduce the amount of modified pixels, we propose an image hiding method which is also derived from Mielikainen's method. In our method, pixels are partitioned into groups with three pixels in each group. The bits of pixels in a group are linked by using XOR Function in a circular way. If the amount of different values between the calculated XOR values and the secret bits is smaller than or equal to 2 in a group, our method can guarantee that at most one pixel is needed to be modified, and three secret bits can be embedded to three pixels. According to the experimental results, the amount of the embedded secret data in the proposed method is larger than those in other methods under the same amount of pixel modifications. Moreover, the image quality of stego-images produced by the proposed method is better than those produced by other methods.

The rest of this paper is organized as follows. Mielikainen's method is described in Section 2. Then, the concept of the proposed method is introduced in Section 3.1. And, the details of the proposed method are presented in Section 3.2. After that, Section 4 demonstrates the experimental results. Finally, some conclusions are made in Section 5.

2. Related Work

In this section, Mielikainen's method [6] is introduced first. The method partitions cover pixels into pixel pairs. Two secret bits are embedded to a pixel pair to form stego-pixels. The way to embed two secret bits into a pixel pair by using Mielikainen's method is described as follows. First of all, assume that a cover pixel pair, a stego-pixel pair and two secret bits are (y_1, y_2), $(\hat{y}_1, \hat{y}_2)$ and (s_1, s_2), respectively. The final purpose of Mielikainen's method is to embed the first secret bit s_1 to the least-significant bit (LSB) of the first cover pixel y_1. As for the second bit s_2, it is related to the function called LSB Matching Function. The result of LSB Matching Function, which takes two stego-pixels $\hat{y}_1$ and $\hat{y}_2$ as parameters, should be the value of the second bit s_2. LSB Matching Function is defined as follows:

$$F(x_1, x_2) = LSB\left(\left\lfloor \frac{x_1}{2} \right\rfloor + x_2\right). \tag{1}$$

The function $LSB(z)$ represents the least-significant bit of pixel z. Two parameters x_1 and x_2 are two input pixels.

More precisely, Mielikainen's method modifies y_1 to $\hat{y}_1$ and y_2 to $\hat{y}_2$ such that the $LSB(\hat{y}_1)$ is s_1 and the result of $F(\hat{y}_1, \hat{y}_2)$ is s_2. LSB Matching Function has the property that $F(y_1\text{-}1, y_2)$ is not equal to $F(y_1\text{+}1, y_2)$ for all possible y_1 and y_2. This means we can make $F(\hat{y}_1, y_2)$ equal to s_2 through adjusting the value y_1 to $\hat{y}_1$ by adding/subtracting its value to/from one. Moreover, the value of the least-significant bit of $(y_1\text{-}1)$ or $(y_1\text{+}1)$ is also changed. Therefore, if $LSB(y_1)$ is not equal to s_1 and $F(y_1, y_2)$ is not equal to s_2, adjusting the value y_1 to $\hat{y}_1$ by adding/subtracting its value to/from one can make $LSB(\hat{y}_1)$ equal to s_1 and $F(\hat{y}_1, y_2)$ equal to s_2, simultaneously. On the

other hand, if $LSB(y_1)$ is not equal to s_1 but $F(y_1, y_2)$ is equal to s_2, adjusting the value y_1 to $\hat{y}_1$ by replacing the least-significant bit of y_1 with its complement can make $LSB(\hat{y}_1)$ equal to s_1 but $F(\hat{y}_1, y_2)$ still retainable. Finally, if $LSB(y_1)$ is equal to s_1 but $F(y_1, y_2)$ is not equal to s_2, adjusting the value from y_2 to $\hat{y}_2$ by replacing the least-significant bit of y_2 with its complement can make $F(y_1, \hat{y}_2)$ equal to s_2 but $LSB(y_1)$ still retainable.

Mielikainen introduced his embedding procedures in the way of pseudo code in [6]. To make them clear, Chan [1] drew Mielikainen's embedding procedures by using a flow chart as shown in Fig. 1. According to the procedures in **Fig. 1**, it can be seen that at most one cover pixel is needed to be modified by adding/subtracting its value to/from one and two secret bits are embedded to two cover pixels.

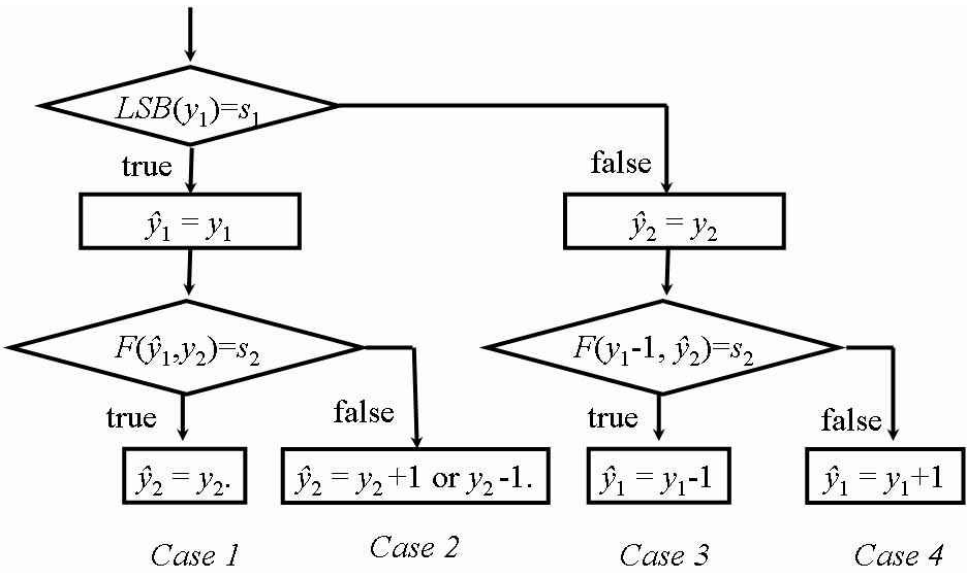


Fig. 1. Mielikainen's procedures

An example is given in **Fig. 2**. Assume that the pixel pair (y_1, y_2) is (4, 11), and then the calculated values of $LSB(y_1)$ and $F(y_1, y_2)$ are 0 and 1, respectively. It can be seen that $F(y_1\text{-}1, y_2)$ is not equal to $F(y_1+1, y_2)$, and $LSB(\hat{y}_1)$ becomes 1 where $\hat{y}_1$ comes from adding/subtracting the value of y_1 to/from one. Therefore, if the two secret bits are (1, 0) or (1, 1), adjusting y_1 by subtracting its value from one or adding its value to one can achieve our goal. On the other hand, if two secret bits (s_1, s_2) are (0, 0), it can be seen that $LSB(y_1)$ is equal to s_1 but $F(y_1, y_2)$ is not equal to s_2. Under this kind of situation, y_2 is adjusted to $\hat{y}_2$ by replacing the least-significant bit of y_2 with its complement, and therefore $\hat{y}_2$ becomes 10.

The above example shows the advantage of the Mielikainen's method comparing with the LSB substitution method. In all cases, at most only one cover pixel is needed to be modified by adding/subtracting its value to/from one, and two secret bits are embedded to two cover pixels. Although Mielikainen's method has reduced the amount of modified pixels when embedding, they can be further reduced by using the proposed method described in the next section.

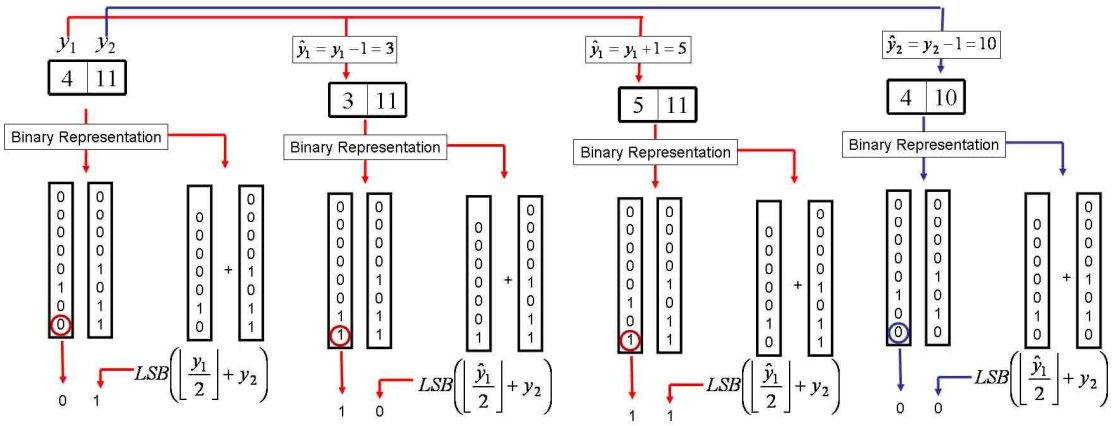


Fig. 2. An example of Mielikainen's method

3. The Proposed Method

In this section, the proposed method is introduced. In Subsection 3.1, the concept of the proposed method is introduced. After that, the proposed method is presented in Subsection 3.2. Finally, a short discussion is described in Subsection 3.3.

3.1 The Concept of the Proposed Method

In this subsection, the concept of the proposed method is introduced. First of all, it is easy to induce that $LSB(\lfloor (y_1)/2 \rfloor)$ represents the value of the second least-significant bit of the pixel y_1. Moreover, $LSB(\lfloor (y_1-1)/2 \rfloor)$ and $LSB(\lfloor (y_1+1)/2 \rfloor)$ are always different for all possible pixel value y_1. An example is shown in **Fig. 3**. As we can see that the values of the second least-significant bits of (y_1-1) and (y_1+1) are different. Therefore, $LSB(\lfloor (y_1-1)/2 \rfloor)$ and $LSB(\lfloor (y_1+1)/2 \rfloor)$ are always different. Another example is shown in **Fig. 3(b)** .

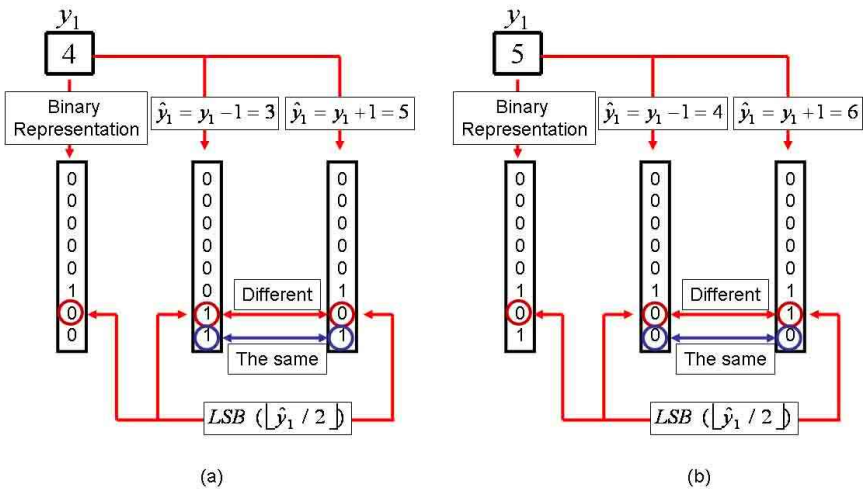


Fig. 3. The values of the second least-significant bits

According to the observation above, Mielikainen's method can be simulated by linking the second least-significant bit of y_1 to the least-significant bit of y_2 using an exclusive OR operator. The Exclusive OR Formula (XF) [1] can be written as follows:

$$XF(x_1, x_2) = LSB\left(\left\lfloor \frac{x_1}{2} \right\rfloor\right) \oplus LSB(x_2). \tag{2}$$

Again, the function $LSB(z)$ represents the value of the least-significant bit of pixel z. And, two parameters x_1 and x_2 are two input pixels. The operator $\oplus$ represents an exclusive OR operator.

According to the Formula (2), the value of $XF(x_1, x_2)$ can be changed through modifying the second least-significant bit of x_1 or the least-significant bit of x_2. In **Fig. 4**, the same example in **Fig. 2** is used to demonstrate the way to perform Formula (2). Assume the pixel pair (y_1, y_2) is (4, 11), and then the calculated values of $LSB(y_1)$ and $XF(y_1, y_2)$ are 0 and 1, respectively. It can be seen that $XF(y_1\text{-}1, y_2)$ is not equal to $XF(y_1\text{+}1, y_2)$, and the values of $LSB(y_1\text{-}1)$ and $LSB(y_1\text{+}1)$ are all changed. Therefore, if two secret bits are (1, 0), adjusting y_1 to y_1-1 can make $LSB(y_1\text{-}1)$ equal to 1 and $XF(y_1\text{-}1, y_2)$ equal to 0. If two secret bits are (1, 1), adjusting y_1 to y_1+1 can make $LSB(y_1\text{+}1)$ equal to 1 and $XF(y_1\text{+}1, y_2)$ equal to 1.

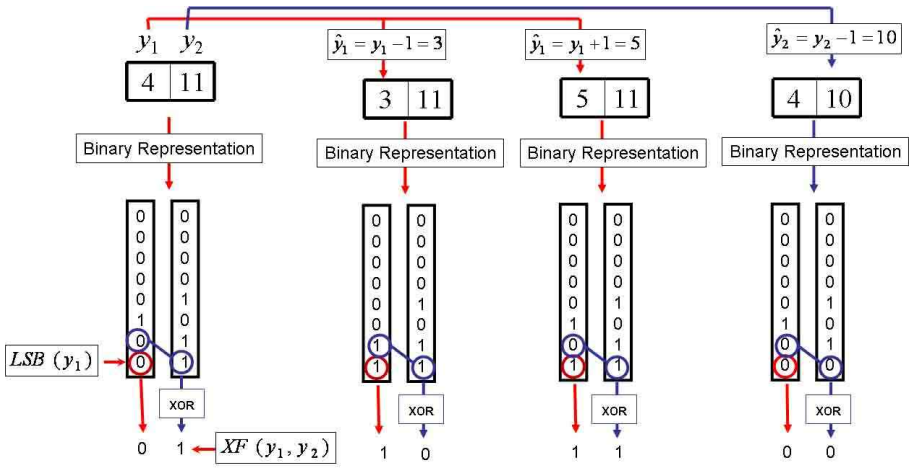


Fig. 4. The example of using the Exclusive OR Formula

The most important point of this mechanism is that once the produced bits $LSB(y_1)$ and $XF(y_1, y_2)$ are not the same as secret bits, at most one pixel is needed to be modified, and two secret bits can be embedded to two pixels. It is trivial that the advantage comes from the condition that both two calculated XF values are different to the secret bits. It means we can get a great benefit from the condition.

In order to amplify this property, in this paper, the exclusive OR operator is applied on three cover pixels in the circular way. Then, two continuous positions with different values between the produced bits and the secret bits can become the same by just adding/subtracting one pixel value to/from one. More precisely, if the calculated exclusive OR values and the secret bits are different at the first and second positions, only the pixel at the first position is needed to modify, and the calculated exclusive OR values will become the same as the secret bits. Similarly, if the calculated exclusive OR values and the secret bits are different at the second

and third positions, only the pixel at the second position is needed to modify, and the calculated exclusive OR values will become the same as the secret bits. Although the first and third positions are not continuous, once the calculated exclusive OR values and the secret bits are different at these two positions, it still only needs to modify the pixel at the third position, and the calculated exclusive OR values will become the same as the secret bits. The reason is that three pixels are already linked in the circular way so that these two positions are treated as two continuous positions.

3.2 The Embedding and Extracting Procedures

In this subsection, the proposed method is described. The main difference between the proposed method and Mielikainen's method is that the proposed method partitions pixels into pixel groups with three pixels in each group. Three secret bits will be embedded to each pixel group. Assume a cover pixel group, a stego-pixel group and a three-bit secret are (y_1, y_2, y_3), $(\hat{y}_1, \hat{y}_2, \hat{y}_3)$ and (s_1, s_2, s_3), respectively. The Circular Exclusive OR Formula (C*XF*) is written as follow:

$$CXF(x_i) = LSB\left(\left\lfloor \frac{x_{(i-1) \bmod 3}}{2} \right\rfloor\right) \oplus LSB(x_i). \tag{3}$$

The symbol x_i means the pixel value at the i-th pixel of three pixels in a pixel group, and the variable i belongs to 0, 1 and 2. Using Formula (3), the *CXF* values of cover pixels can be calculated. Comparing the calculated *CXF* values with secret bits, the positions with different values can be located. The following procedures try to modify (y_1, y_2, y_3) to $(\hat{y}_1, \hat{y}_2, \hat{y}_3)$ such that three secret bits (s_1, s_2, s_3) can be embedded to $(\hat{y}_1, \hat{y}_2, \hat{y}_3)$.

To make it clear, all possible cases are classified to three cases according to the amount of different values between the calculated *CXF* values and the secret bits. These cases are Case A, Case B and Case C. If the amount of different values in a pixel group is only one, it belongs to Case A. Otherwise; it belongs to Case B or Case C when the amount of different values is two or three, respectively.

For the first case, Case A, there is only one position with different value between the calculated *CXF* values and the secret bits. Assume that the result of $CXF(y_i)$ is not equal to s_i, where i is 0, 1 or 2. Then, the least-significant bit of y_i is replaced with its complement. The way to modify the pixel can also be written as below:

$$\hat{y}_i = y_i - LSB(y_i) + \overline{LSB(y_i)}. \tag{4}$$

The symbol $\overline{LSB(y_i)}$ means the complement of the least-significant bit of y_i. Note that replacing the least-significant bit of pixel y_i with its complement does not affect the results of other pixels' *CXF* values.

Examples of Case A are demonstrated in **Fig. 5**. Assume the values of cover pixels (y_1, y_2, y_3) are (4, 11, 5). Three secret bits are (1, 1, 0), (0, 0, 0) and (0, 1, 1), respectively. The positions of different values between $(CXF(y_1), CXF(y_2), CXF(y_3))$ and three secret bits are at the first, the second and the third positions, respectively. According to Formula (3), the results of $(CXF(y_1), CXF(y_2), CXF(y_3))$ are (0, 1, 0), respectively. For the first case in **Fig. 5(a)**, its three secret bits (s_1, s_2, s_3) are (1, 1, 0). Because only $CXF(y_1)$ is not equal to s_1, the pixel value y_1 should be modified according to Formula (4). It can be seen that we only need to replace the

least-significant bit of y_1 with its complement, that is 1, and three secret bits are embedded to three cover pixels. The same procedure is performed on **Fig. 5(b)** and **Fig. 5(c)** and three secret bits (0, 0, 0) and (0, 1, 1) can be embedded to three cover pixels, respectively.

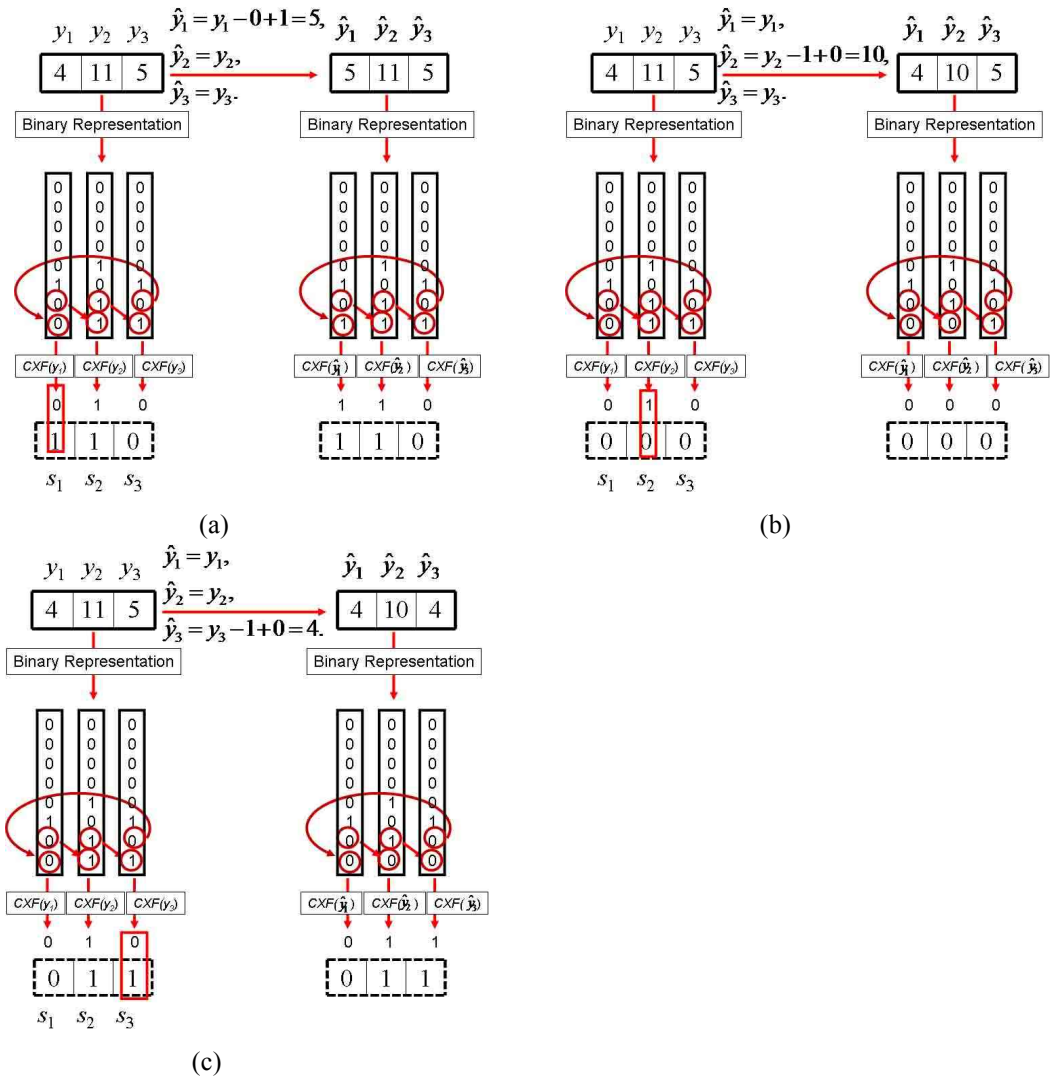


Fig. 5. Examples of the Case A

For the second case, it is more complex so that Case B is further divided into three cases, called Case B-I, Case B-II and Case B-III. The procedures of modifying (y_1, y_2, y_3) to $(\hat{y}_1, \hat{y}_2, \hat{y}_3)$ are drawn in **Fig. 6**. By observing these procedures, each will check the condition whether $LSB(\lfloor (y_i - 1)/2 \rfloor)$ is equal to $LSB(\lfloor y_i / 2 \rfloor)$. If $LSB(\lfloor (y_i - 1)/2 \rfloor)$ is equal to $LSB(\lfloor y_i / 2 \rfloor)$, it means y_i-1 will not modify the second least-significant bit of y_i. Note that there are only two different values between the calculated *CXF* values and the secret bits in Case B. If we want to make $CXF(y_i)$ equal to s_i and $CXF(y_{(i+1) \bmod 3})$ equal to $s_{(i+1) \bmod 3}$, either adding y_i to one or subtracting y_i from one will achieve our goal. The reason is that one of two operators can change the bit values of two least-significant bits of y_i at the same time. Since the least-significant bit of y_i has

been changed, the result of $CXF(y_i)$ is also changed. As for the result of $CXF(y_{(i+1) \bmod 3})$, it relates to the second least-significant bits of y_i according to Formula (3). Once the second least-significant bits of y_i is changed, the result of $CXF(y_{(i+1) \bmod 3})$ will also be changed. Therefore, checking the condition is used to see whether the second least-significant bit of y_i is changed or not after subtracting its value from 1. If it is not changed, $\hat{y}_i$ is set as y_i+1. Otherwise, $\hat{y}_i$ is set as y_i-1. It can be seen that in our proposed method, only one cover pixel is needed to be modified for Case B by adding/subtracting its value to/from one, and three secret bits can be embedded to three cover pixels. This property is the most important part of the proposed method.

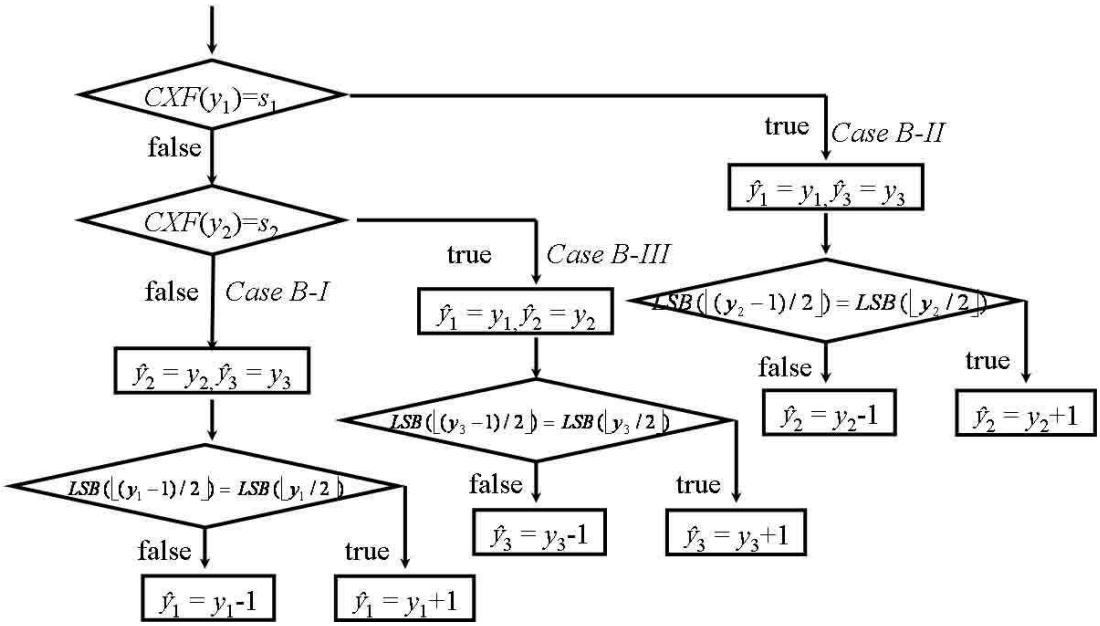


Fig. 6. The procedures of the proposed method for Case B

Examples of Case B are demonstrated in **Fig. 7**. Assume the values of cover pixels (y_1, y_2, y_3) are (4, 11, 5). Three secret bits are (1, 0, 0), (0, 0, 1) and (1, 1, 1), respectively. Those three secret bits are related to Case B-I, Case B-II and Case B-III, respectively. According to Formula (3), the results of $(CXF(y_1), CXF(y_2), CXF(y_3))$ are (0, 1, 0), respectively. For the first case in Fig. 6(a), its three secret bits (s_1, s_2, s_3) are (1, 0, 0). Because $CXF(y_1)$ is not equal to s_1 and $CXF(y_2)$ is not equal to s_2, the pixel values y_2 and y_3 are retained without being changed while the pixel value y_1 should be modified. In the first step, we check whether $LSB(\lfloor (y_1 - 1)/2 \rfloor)$ is equal to $LSB(\lfloor y_1/2 \rfloor)$. It is trivial that $LSB(\lfloor (4-1)/2 \rfloor)$ is not equal to $LSB(\lfloor 4/2 \rfloor)$. That means the second least-significant bit of y_1 has been changed after subtracting y_1 from 1. Moreover, the $CXF(y_2)$ value has also been changed, because $CXF(y_2)$ value is related to the second least-significant bit of y_1. It can be seen that we only need to subtract y_1 from 1, and three secret bits are embedded to three cover pixels. The ways to process Case B-II and Case B-III are the same except that the modified pixels are y_2 and y_3, respectively. The examples of those two cases are shown in **Fig. 6(b)** and **Fig. 6(c)**, respectively.

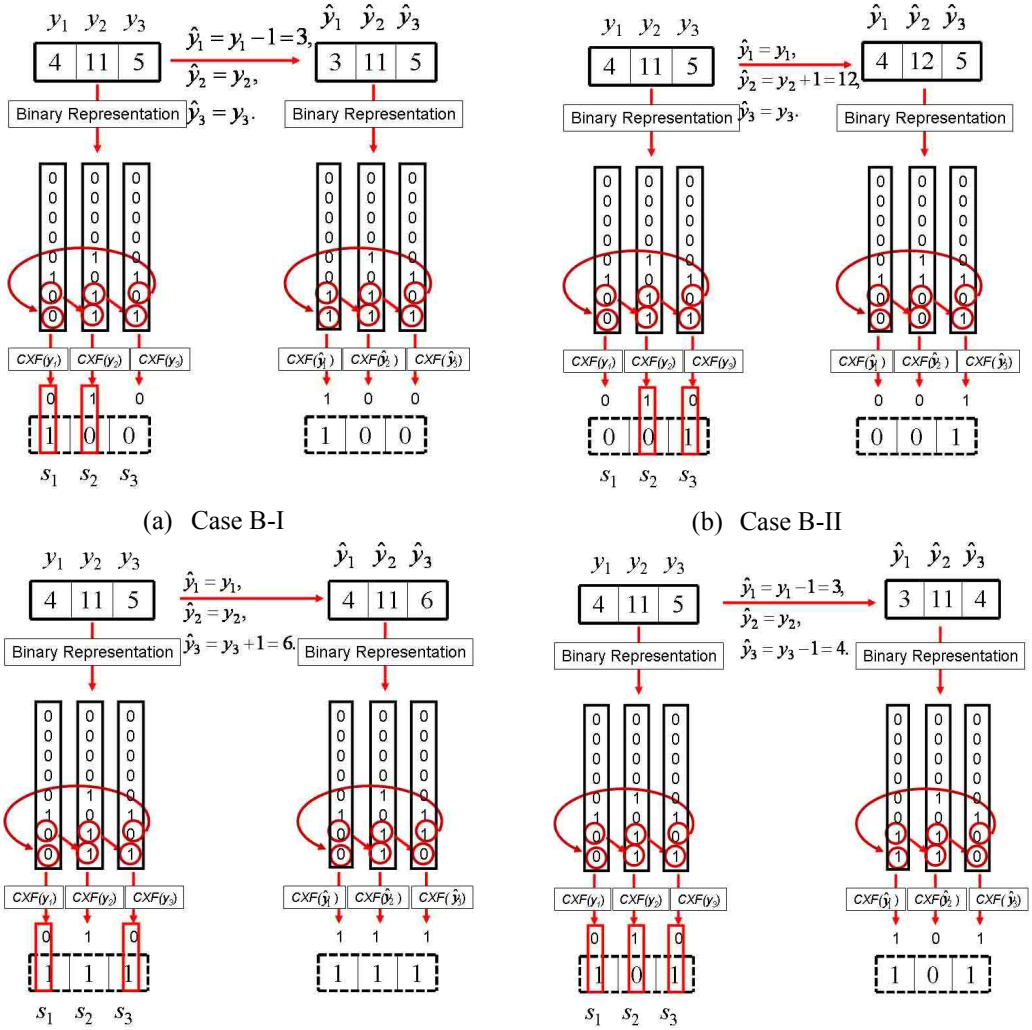


Fig. 7. Examples of the Case B and Case C

For the third case, Case C, there are three positions with different values between the calculated *CXF* values and the secret bits. In this case, the least-significant bit of y_3 is modified by using Formula (4). Meanwhile, y_1 should also be modified such that the values of $CXF(\hat{y}_1)$ and $CXF(\hat{y}_2)$ are all changed. This situation is the same as Case B-I and we use the same procedures to check and modify y_1. The whole procedures for Case C are drawn in **Fig. 8**. An example for Case C is shown in **Fig. 7(d)** and the secret bits in this case are (1, 0, 1).

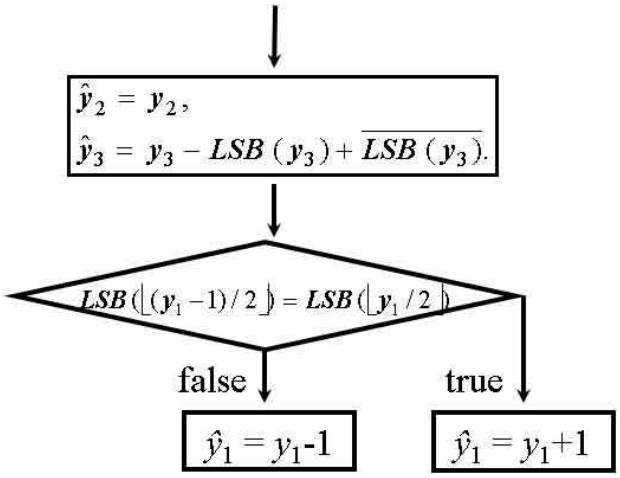
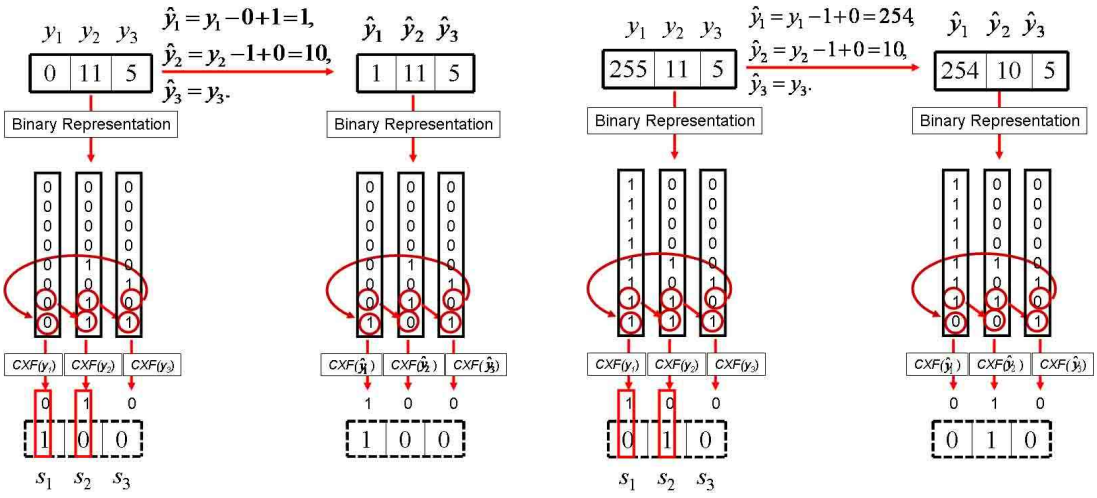
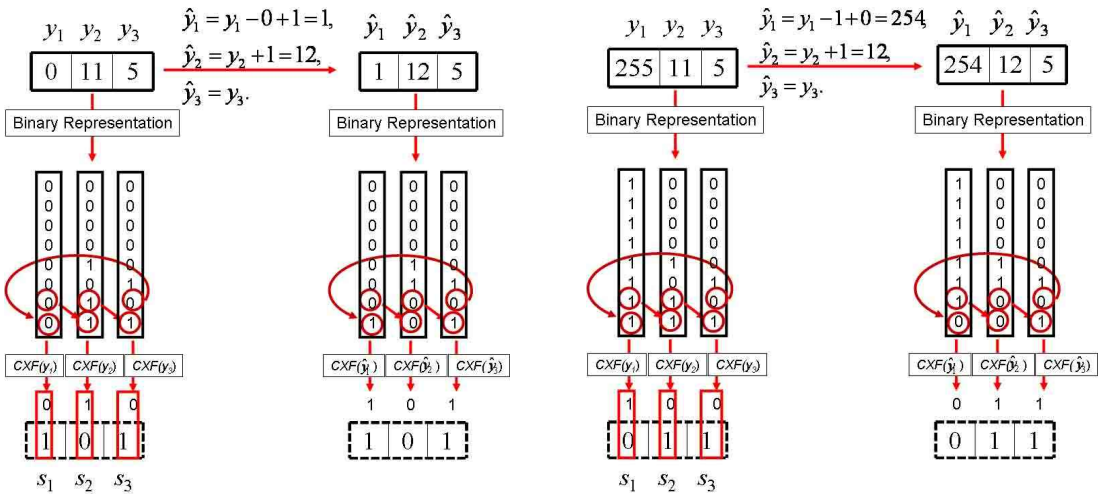


Fig. 8. The procedures of the proposed method for Case C

For a pixel with value 255 or 0, the pixel can not be added to one or subtracted from one. This situation may also occur in Mielikainen's method. If this condition occurs in Case B, the proposed method uses Formula (4) to embed secret bits. For example, we assume the value pixel at the first position in **Fig 6(a)** is 0 or 255 instead of 4. It is trivial that the value 0 or 255 can not be subtracted from one or added to one. Therefore, both the least-significant bits of the first and second pixels are modified by using Formula (4). Then, three secret bits can be embedded to three cover pixels as shown in **Fig. 9(a)** and **Fig. 9(b)**. On the other hand, if this condition occurs in Case C, the pixel with 255 or 0 will be modified by using Formula (4). Then, rest part will be the same as Case B. It will go through the same procedure of Case B to embed three secret bits as shown in **Fig. 9(c)** and **Fig. 9(d)**.





(c) (d)

Fig. 9. Special cases for pixels with value 255 or 0

3.3 Discussion

The advantage of the proposed method is that if the number of different values between the calculated XOR values and the secret bits is smaller than or equal to 2, our method can guarantee that at most one pixel is needed to be modified by adding/subtracting their value to/from one and three secret bits can be embedded to three cover pixels. To achieve this purpose, the current pixel should be related to other pixels. That is why the proposed method uses Formula (3) to link current pixel to other pixels. Under the above-mentioned linking way, any two different values between the calculated *CXF* values and the secret bits become continuous. Then, it can be guaranteed that at most one pixel is needed to be modified under Case B.

If we partition cover pixels into pixel groups with "four" pixels instead of "three" pixels, the above advantage will be eliminated. That is, it may modify more than one pixel to embed secret data when the number of the different values between the calculated *CXF* values and the secret bits is two. For example, assume that the *CXF* values of four cover pixels and their corresponding secret bits are different at first and third positions. Because different values between the calculated *CXF* values and the secret bits are not continuous, at least two cover pixels are needed to be modified so as to achieve hiding.

4. Experimental Results

The experimental results are demonstrated in this section. In the first experiment, we do some theoretical analysis by using embedding efficiency [10]. Embedding efficiency is used to calculate the amount of embedded secret bits under the fixed quantity of pixel modifications in average. The definition of embedding efficiency is the average amount of secret bits carried by one embedding change in the cover data. The higher the value of embedding efficiency is, the lower the embedding change of cover pixels is. The way to calculate embedding efficiency can be written as below:

$$Embedding\ Efficiency(f_{emb}) = \frac{(n) * \log_2(n)}{\sum_{i=0}^{n} \# f_{emb}(i)}. \quad (5)$$

The symbol f_{emb} denotes the embedding algorithm and the variable n represents the total number of possible secret data that can be embedded by using embedding algorithm f_{emb}. Moreover, the symbol $\#f_{emb}(i)$ means the amount of the embedding change when embedding *i-th* secret data by using embedding algorithm f_{emb}. If the embedding algorithm f_{emb} modifies cover pixels by only adding/subtracting their values to/from one, the quantity of the embedding change for those modified pixel is one. Then, the total amount of embedding change is also equal to the number of modified pixels. That is the reason why the total amounts of modified pixels are listed in **Table 1**.

Taking the proposed method as an example, if the number of cover pixels is three, three secret bits can be embedded to them. Then, all possible secret data are from $(000)_2$ to $(111)_2$. The total amount of possible secret data n is 8. When embedding all possible secret data to three cover pixels one by one, the total amount of embedded secret bits is $8\times\log_2 8$. Moreover, the total amount of modified pixels is 8 which can be found in **Table 1**. In fact, the total number of modified pixels dose not relate to the values of cover pixels. It means expect for the pixel values 0 and 255, the total amounts of modified pixels are still 8 for all possible three cover pixels. According to the definition of embedding efficiency in [10], the embedding efficiency of the proposed method is $(8\times\log_2 8)/8 = 3$. It means that expect for the pixel values 0 and 255, in average the proposed method can embed 3 secret bits by modifying one pixel through adding/subtracting its value to/from one. On the contrary, the embedding efficiency of Mielikainen's method is 2.667.

Table 1. The total amounts of the modified pixels and the values of embedding efficiency

Number of cover pixels	Mielikainen's method		Lin's Method [5]		The Proposed Method	
	Number of modified pixels	Embedding Efficiency	Number of modified pixels	Embedding Efficiency	Number of modified pixels	Embedding Efficiency
3	—	—	10	2.853	8	3
4	24	2.667	—	—	—	—
6	**144**	**2.667**	**180**	**2.853**	**128**	**3**
8	768	2.667	—	—	—	—
9	—	—	2430	2.853	1536	3
10	3840	2.667	—	—	—	—
12	**18432**	**2.667**	**29160**	**2.853**	**16384**	**3**
14	86016	2.667	—	—	—	—
15	—	—	328050	2.853	163840	3

In **Table 1**, it also shows the results of Lin's method [5]. Lin's method is proposed in 2011 to do secret hiding. The reason why we take Lin's method to compare with our proposed method is that Lin's method is also inspired from Mielikainen's method. His method also takes 3 cover pixels as a group to hide secret data. However, the embedding and extracting procedures of Lin's method and our method are totally different. According to the results in **Table 1**, it can be

seen that the amounts of the modified pixels in the proposed method are always lower than those in Mielikainen's and Lin's method. Therefore, the values of embedding efficiency of proposed method are always higher than those of Mielikainen's and Lin's method.

In the second experiment, real images were used to perform the Mielikainen's method, Lin's method and the proposed method. The cover images were Lena and Plane with size 512×512 pixels as shown in **Fig. 10**. Three secret images were Tiffany, Boat and Toys with size 256×128 pixels as shown in **Fig. 11**.

(a) Lena (b) Plane

Fig. 10. Two 512×512-pixel cover images

(a) Tiffany (b) Boat (c) Toys

Fig. 11. Three 256×128-pixel secret images

The experimental results are shown in **Table 2** and **Table 3**. The amount of total modified pixels under the embedding capacity of 1 bit per pixel (bpp) is shown in **Table 2**. According to **Table 2**, Lin's method [5] can reduce about 6000 pixel modifications comparing with Mielikainen's method [6]. In the aspect of the proposed method, it can be seen that our method can further reduce about 5000 pixel modifications comparing with Lin's method [5].

The values in Table 3 are the peak signal to noise ratio (*PSNR*) values and hiding cost by using Mielikainen's method, Lin's method and the proposed method, respectively. The *PSNR* values are used to estimate the quality of the stego-images. The *PSNR* values can be calculated according to the following formula:

$$PSNR = 10 \times \log \frac{(255)^2}{MSE} dB \tag{6}$$

Here *MSE* means the mean square error, and is derived from the square errors of all pixels.

$$MSE = \frac{1}{(w \times h)} \sum_{I=1}^{w} \sum_{J=1}^{h} (\alpha(I,J) - \beta(I,J))^2 \tag{7}$$

The symbols $\alpha(I, J)$ and $\beta(I, J)$ represent the pixel values at the position (I, J) in the stego-image and the original image, respectively. The symbols w and h represent the pixel numbers for the width and the height of the image, respectively. The hiding cost represents the quantity of pixel modifications in average to embed a secret bit. The hiding cost can be calculated by dividing amount of the modified pixels by amount of secret bits.

According to experiment results in **Table 3**, the *PSNR* values in the proposed method are always higher than those in Mielikainen's and Lin's method. This means the stego images produced by the proposed method have better quality than those produced by Mielikainen's and Lin's method. Moreover, the hiding costs in the proposed method are always lower than those in Mielikainen's and Lin's method. This implies that the proposed method can modify fewer pixels to embed the same quality of secret bits. All in all, the values in **Table 3** show that the proposed method is superior to Mielikainen's method [6] and Lin's method [5].

Table 2. The amount of the modified pixels (the embedding capacity: 1 bit per pixel(bpp))

		Cover Images					
		Lena			**Plane**		
		Mielikainen's Method [6]	**Lin's Method [5]**	**Our Method**	**Mielikainen's Method [6]**	**Lin's Method [5]**	**Our Method**
Secret Images	**Tiffany**	98341	92304	87234	98287	92017	87450
	Boat	98162	92040	87387	98338	92199	87325
	Toys	97962	92322	87225	98383	92069	87240

Table 3. Comparisons of different methods

	Cover Images											
	Lena						Plane					
	Mielikainen's Method [6]		Lin's Method [5]		Our Method		Mielikainen's Method [6]		Lin's Method [5]		Our Method	
	PSNR	Hiding Cost	*PSNR*	Hiding Cost	*PSNR*	Hiding Cost (HC)	*PSNR*	HC	*PSNR*	HC	*PSNR*	HC
Tiffany	52.39	0.375	52.66	0.352	52.91	0.333	52.39	0.375	52.69	0.35	52.90	0.33

	Boat	52.40	0.374	52.68	0.351	52.90	0.333	52.39	0.375	52.67	0.35	52.91	0.33
	Toys	52.41	0.374	52.66	0.352	52.91	0.333	52.39	0.375	52.68	0.35	52.91	0.33

5. Conclusion

In 2006, Mielikainen proposed his method to do image hiding. Mielikainen's method partitions cover pixels into pixel pairs. Two secret bits are embedded to a pixel pair. In Mielikainen's method, at most one pixel is needed to be modified for each pixel pair by adding/subtracting the pixel value to/from one. In this paper, the proposed method which was derived from Mielikainen's method partitions pixels into pixel groups with three pixels in each group. The bits of pixels in a group are linked by using XOR Function in a circular way. This way, any two positions with different values between the calculated XOR values and the secret bits become continuous, and at most one pixel is needed to be modified by adding/subtracting it value to/from one. According to theoretical analysis, the amount of the embedded secret data in the proposed method is larger than those in other methods under the same amount of pixel modifications. Our experimental results also demonstrate that the quality of stego-images in the proposed method is higher than those in other methods.

References

[1] C. S. Chan, "On using lsb matching function for data hiding in pixels," *Fundamenta Informaticae*, vol.96, pp.49-59, 2009. Article (CrossRef Link)

[2] C. C. Chang, J. Y. Hsiao, and C. S. Chan, " finding optimal least-significant-bit substitution in image hiding by dynamic programming strategy," *Pattern Recognition*, vol.36, no.7, pp.1583-1595, Jul.2003.Article (CrossRef Link)

[3] J. J. Harmsen, and W. A. Pearlman, "steganalysis of additive noise modelable information hiding," in *Proc. of SPIE Security Watermarking Multimedia Contents V*, vol.5020, pp.131-142, 2003.Article (CrossRef Link)

[4] A. Ker, "Improved detection of LSB steganography in grayscale image," in *Proc. of Information Hiding Workshop*, vol.3200, pp.97-115, 2004. Article (CrossRef Link)

[5] C. C. Lin, "An information hiding scheme with minimal image distortion," *Computer Standards & Interfaces*, vol.33, pp.477-484, 2011. Article (CrossRef Link)

[6] J. Mielikainen, "LSB Matching Revisited," *IEEE Signal Processing Letters*, vol.13, no.5, pp.285-287, 2006. Article (CrossRef Link)

[7] C. C. Thien, and J. C. Lin, "A simple and high-hiding capacity method for hiding digit-by-digit data in images based on modulus function," *Pattern Recognition*, vol.36, pp.2875-2881, 2003.Article (CrossRef Link)

[8] A. Westfeld, "F5-A Steganographic Algorithm: High Capacity Despite Better Steganalysis," *ILNCS 2137, Springer-Verlag*, vol. 2137, no.11, pp. 289-302, 2001. Article (CrossRef Link)

[9] R. Z. Wang, C. F. Lin, and J. C. Lin, "Image hiding by optimal LSB substitution and genetic algorithm," *Pattern Recognition*, vol.34, no.3, pp.671-683, 2001. Article (CrossRef Link)

[10] X. Zhang, and S. Wang, "Efficient steganographic embedding by exploiting modification direction," *IEEE Communications Letters*, vol.10, no.11, pp.781-783, 2006. Article (CrossRef Link)

[11] W. Zhang, S. Wang, and X. Zhang, "Improving embedding efficiency of covering codes for applications in steganography," *IEEE Communications Letters*, vol.11, no.8, pp.680-682, 2007. Article (CrossRef Link)

Chi-Shiang Chan was born in Taiwan in 1975. He received the B.S. degree in computer science in 1999 from the National Cheng-Chi University and the M.S. degree in computer science and information engineering in 2001 from the National Chung Cheng University. He received his Ph.D. degree in computer engineering in 2005 also from the National Chung Cheng University. From 2007 to 2010, he has worked as an assistant professor with the Department of Information Science and Applications, Asia University. He is currently an associate professor with the Department of Applied Informatics and Multimedia, Asia University. His research interests include image and signal processing, image compression, information hiding, and data engineering.

Yuan-Yu Tsai was born in Taichung, Taiwan, in 1978. He received the B.S. degree in Department of Computer Science and Information Engineering from National Central University, Taiwan, in 2000, and the Ph.D. degree in Institute of Computer Science from National Chung Hsing University, Taiwan, in 2006. He is currently an assistant professor at the Department of Applied Informatics and Multimedia, Asia University, Taiwan. His research interests include computer graphics and information hiding algorithms for three-dimensional models and images. He is a member of the ACM and the IEEE Computer Society.

Chao-Liang Liu was born in Hsinchu, Taiwan, in 1966. He completed his B.Sc. degree in mathematics from National Cheng-Kung University, Taiwan, in 1989, and the Ph.D. degree in Institute of Computer Science from National Chung Hsing University, Taiwan, in 2007. He is currently an assistant professor at the Department of Applied Informatics and Multimedia, Asia University, Taiwan. His research interests include information security, cryptography, elliptic curve cryptosystem and pairing computation.