applied sciences

MDPI

Article

An Improved Product Code-Based Data Hiding Scheme

Wen-Rong Zhang and Yuh-Ming Huang *

Department of Computer Science and Information Engineering, National Chi Nan University, Puli, Nantou 545, Taiwan; s101321508@ncnu.edu.tw

* Correspondence: ymhuang@csie.ncnu.edu.tw

Received: 31 July 2018; Accepted: 25 October 2018; Published: 1 November 2018

Abstract: This paper explores the data hiding schemes which are based on the principle of matrix embedding. Under the same embedding rate, the efficiency of each data hiding scheme is evaluated by the metric of average embedding efficiency. In the literature, both the row-column embedding and the weight approximation embedding algorithms are sub-optimal solutions for the product code-based data hiding problem. For the former, it is still based on the concept of one-dimensional (1-D) toggle syndrome, and the concept of two-dimensional (2-D) toggle syndrome is directly adopted for the latter one. Data hiding with multiple embedding channels is the practice of hiding messages into hidden media many times. Here, two multi-channel embedding-based data hiding techniques—one is the 1-D toggle syndrome-based embedding scheme (1DTS-1), and the other is the improved weight approximation-based embedding scheme (2DTS-1), are presented. In the former, the proposed one-off decision technique is used to determine the locations of the required modification bits, and the amount of modification will be reduced through utilizing the characteristics of the linear code. With the technique of the former, in the latter, the amount of modification bits can be further reduced because that a toggle array with better structure is generated, which is more suitable for being assigned as the initial toggle array while applying the weight approximation approach. The experimental results show our proposed hybrid 1-D/2-D toggle syndrome-based embedding scheme (2DTS-1) has increased the embedding efficiency by 0.1149 when compared to the weight approximation embedding algorithm. Further, the embedding efficiency of the latter one can be further and significantly enhanced through the Hamming+1 technique.

Keywords: matrix embedding; multiple-channel embedding; product code; Hamming+1; embedding efficiency

1. Introduction

Data (or information) hiding is one kind of steganographic technique to embed the secret information into a cover host, such as an image. Usually, the naked eye cannot perceive any change when the image is modified slightly.

In 1998, Crandall [1] first introduced the idea of matrix embedding. The technique of matrix embedding was deeply studied in [2–4] by utilizing the characteristic of the strong algebraic structure of one-dimensional (1-D) linear code. Matrix embedding is also termed as syndrome coding and 1-D linear code is categorized as one kind of data hiding code [3,4].

The idea of product code-based data hiding was introduced in [5–8]. Basically, it can be considered as an extension of the previous works [1–4] by taking a cover matrix block, not a cover sequence, for embedding on each time. They divided the cover image into disjoint matrix blocks of $(2^y - 1) \times (2^x - 1)$ pixels and applying the 1-D linear code-based embedding algorithm in each row and the first $2^{x-1} - 1$ columns. During each column embedding, it is very likely that the pixel modification to some row must

be adjusted with more pixels changed to ensure the correct extraction of the secret information. Although, it will induce higher distortion in image quality, the amount of embedded information can be increased.

The product code is a two-dimensional (2-D) linear code that is composed of two 1-D linear codes. Although, the embedding concepts for both the 1-D and 2-D linear code-based data hiding schemes are similar in respectively adopting the technique of 1-D syndrome coding and 2-D syndrome coding, the embedding efficiency of the latter is much higher. However, due to the tremendous computational complexity in finding a coset leader [9,10] of a 2-D linear code, only sub-optimal approaches with moderate computational complexity were presented in the literature [5–8,11,12]. In [5–8,12], the authors still utilized the 1-D toggle syndrome coding technique to solve the product code-based data hiding problem, whereas the authors of [11] directly utilized the concept of the 2-D toggle syndrome and proposed a sub-optimal approach with better embedding efficiency and low computational complexity. Therefore, in essence, each of the schemes that were proposed in [5–8,12] is not considered to be one kind of product code-based data hiding scheme.

Data hiding with multiple embedding channels is the practice of hiding messages into hidden media many times. Usually, divide and conquer is a good technique to solve a hard problem. In this paper, instead of directly utilizing the concept of the 2-D toggle syndrome, eventually a hybrid 1-D/2-D toggle syndrome-based embedding scheme is proposed with good embedding efficiency and low computational complexity. We first propose a 1-D toggle syndrome-based scheme with two embedding channels. In this scheme, a one-off decision technique is presented to determine the locations of the required modification bits and the amount of modification will be reduced by utilizing the characteristics of the linear code. Next, to integrate the technique of [11] into our proposed scheme, we continue to propose another 1-D toggle syndrome-based scheme with three embedding channels. Since the resultant toggle array is more suitable for being assigned as the initial toggle array in the weighted approximation embedding scheme, thus a better solution is found. Further, there almost exists at least one bit 1 in each row of the toggle array of the proposed 1-D/2-D toggle syndrome-based embedding scheme. Hence, the embedding efficiency can be further and significantly enhanced when combined with the strategy of Hamming+1.

This paper is organized as follows. Section 2 briefly introduces the concept of matrix embedding. The product code-based data hiding schemes are reviewed in Section 3. The proposed scheme is presented in Section 4. Section 5 presents the proposed Scheme combined with the technique of Hamming+1. Experimental results are shown in Section 6. Some concluding remarks are given in Section 7. Finally, the proposed 1-D toggle syndrome-based embedding scheme is also listed in the Appendix A.

2. Matrix Embedding

Given a 1-D linear code (n, k), the secret message $\mathbf{m} = (m_1, m_2, \ldots, m_{n-k})$ of n-k bits is hidden into the least significant bits (LSBs), denoted as $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ of the original n image pixels. Due to the characteristic of the strong algebraic structure, the resultant fact is at most only one LSB needs to be modified among those n LSBs. Taking a (7, 4) binary systematic Hamming code with the generator polynomial $1 + x + x_3$ as the example, as shown in Figure 1. Table 1 is the standard array of the (7, 4) Hamming code.

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(a)} \qquad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \quad \text{(b)}$$

Figure 1. (**a**) Generator matrix **G**; and, (**b**) Parity check matrix **H** of a (7, 4) Hamming code.

Table 1. The standard array of size $2^3 \times 2^4$ for a (7, 4) Hamming code.

Syndrome	
Coset Leader	
000	0000000 1101000 0110100 1011100 1110010 0011010 1000110 0101110 1010001 0111001 1100101 0001101 0100011 1001011 0010111 1111111
100	1000000 0101000 1110100 0011100 0110010 1011010 0000110 1101110 0010001 1111001 0100101 1001101 1100011 0001011 1010111 0111111
010	0100000 1001000 0010100 1111100 1010010 0111010 1100110 0001110 1110001 0011001 1000101 0101101 0000011 1101011 0110111 1011111
001	0010000 1111000 0100100 1001100 1100010 0001010 1010110 0111110 1000001 0101001 1110101 0011101 0110011 1011011 0000111 1101111
110	0001000 1100000 0111100 1010100 1111010 0010010 1001110 0100110 1011001 0110001 1101101 0000101 0101011 1000011 0011111 1110111
011	0000100 1101100 0110000 1011000 1110110 0011110 1000010 0101010 1010101 0111101 1100001 0001001 0100111 1001111 0010011 1111011
111	0000010 1101010 0110110 1011110 1110000 0011000 1000100 0101100 1010011 0111011 1100111 0001111 0100001 1001001 0010101 1111101
101	0000001 1101001 0110101 1011101 1110011 0011011 1000111 0101111 1010000 0111000 1100100 0001100 0100010 1001010 0010110 1111110

First, calculate the 1-D syndrome **s** (= $\mathbf{x} \times \mathbf{H}^{\mathrm{T}}$) and the 1-D toggle syndrome **ts** (= $\mathbf{s} \oplus \mathbf{m}$), where $\oplus$ denotes the component-wise EXCLUSIVE OR (XOR) operation between **s** and **m**. Then, according to Table 1, find the coset leader **e** with the minimum Hamming weight of the coset to which **ts** belongs, that is $\mathbf{e} \times \mathbf{H}^{\mathrm{T}} = \mathbf{ts}$, and change **x** to **y** (= $\mathbf{x} \oplus \mathbf{e}$) to complete the data embedding. In the receiving end, the receiver can easily extract the message **m** by $\mathbf{y} \times \mathbf{H}^{\mathrm{T}}$. Usually, the coset leader **e** for a 1-D linear code is called the optimal toggle sequence **t**, which is the optimal vector matrix that we are looking for to make the image pixels change as little as possible.

3. Product Code-Based Data Hiding

Two 1-D linear block codes $C1$ ($n1$, $k1$) and $C2$ ($n2$, $k2$) can be used to generate a 2-D ($n1 \times n2$, $k1 \times k2$) linear code (called product code), such that each codeword (called code array) is an array of size $n2 \times n1$, as shown in Figure 2. After sequentially encoding the $k2$ rows of the $k2 \times k1$ array of information bits (**IB**) by $C1$ to generate the array **CR** (Check on Rows), and the $k1$ columns of array **IB** are sequentially encoded by $C2$ to generate the array **CC** (Check on Columns). Then, the array **CCH** (Checks on Checks) can be obtained by performing either $C1$ encoding on the $n2$-$k2$ rows of array **CC**, or $C2$ encoding on the $n1$-$k1$ columns of array **CR** [13]. Note, the L-shaped parity check block that is the union of the arrays **CC**, **CCH**, and **CR** and is a function of array **IB**, denoted as P(**IB**). Take the (7, 4) Hamming code as the example. That is, $n1 = n2 = 7$ and $k1 = k2 = 4$, then a (49, 16) product code can be generated, and there are 33 parity check bits. For the (49, 16) product code, similar to Table 1, the size of the standard array is up to $2^{33} \times 2^{16}$. Each element of the standard array is an array of size 7×7. There are 2^{16} code arrays and 2^{33} coset leaders in the standard array.

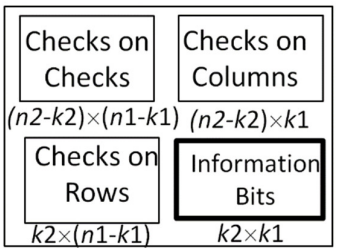

Figure 2. Code array of size $n2 \times n1$ for an ($n1 \times n2$, $k1 \times k2$) product code.

3.1. Trivial Solution

Using the (49, 16) product code for data hiding, the secret message of size 33 bits can be hidden into the cover array of 7×7 bits constructed by the LSB slicing of an array of 7×7 gray image pixels.

Let **X** be the cover array of 7×7 slicing bits, i.e., $[x_{i,j}]_{7\times7}$. Similar to the partition of the code array of a product code, **X** is partitioned into four parts, $\mathbf{X}_{\mathrm{IB}}$, $\mathbf{X}_{\mathrm{CC}}$, $\mathbf{X}_{\mathrm{CCH}}$, and $\mathbf{X}_{\mathrm{CR}}$. The L-shaped block that is the union of $\mathbf{X}_{\mathrm{CC}}$, $\mathbf{X}_{\mathrm{CCH}}$, and $\mathbf{X}_{\mathrm{CR}}$ is denoted as $\mathbf{X}_{\mathrm{P}}$, as shown in Figure 3a. The secret message with 33 bits is arranged as an L-shaped block $\mathbf{M} = (\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{11})$, where each $\mathbf{m}_i$ ($1 \leq i \leq 11$) is an array of size 1×3, as shown in Figure 3b.

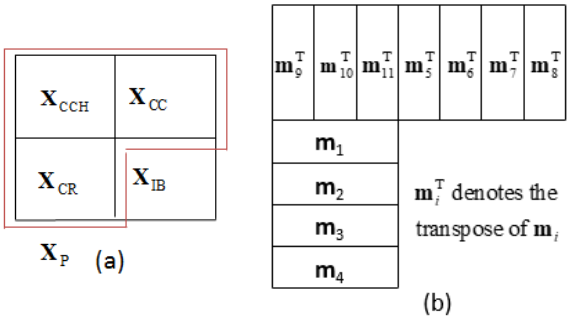

Figure 3. (**a**) $\mathbf{X}_P$ part of the cover array **X** (**b**) Secret message block **M**.

3.1.1. Embedding

Step 1: Calculate the 2-D syndrome of **X** denoted as S(**X**) which equals P($\mathbf{X}_{IB}$) $\oplus$ $\mathbf{X}_P$, where $\oplus$ denotes the element-wise EXCLUSIVE OR (XOR) operation between P($\mathbf{X}_{IB}$) and $\mathbf{X}_P$.
Step 2: Calculate the 2-D toggle syndrome **TS**, which equals S(**X**) $\oplus$ **M**, as shown Figure 4a.
Step 3: Attach the zero block of size 4 $\times$ 4 into the lower right corner of the toggle syndrome **TS**, as shown Figure 4b, to obtain a 7 $\times$ 7 toggle array **T**. That is, the syndrome of **T** denoted as S(**T**) equals **TS**.
Step 4: Perform the element-wise XOR operation between **X** and **T**, i.e., **X** $\oplus$ **T** = **Y**, to finish the embedding, as shown in Figure 4c.

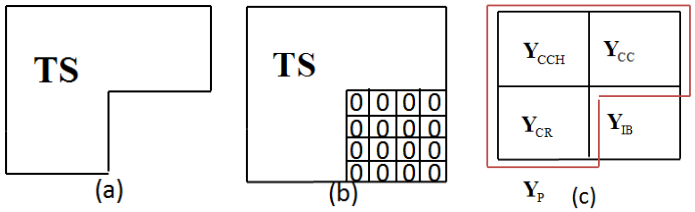

Figure 4. (**a**) Toggle syndrome **TS** (**b**) Toggle array **T** (**c**) Stego array **Y**.

3.1.2. Extracting

In the receiving end, once the 7 $\times$ 7 slicing bits (**Y**) of the stego image block is obtained, the secret message block **M** can be easily extracted by calculating the syndrome of **Y**, that is S(**Y**) = P($\mathbf{Y}_{IB}$) $\oplus$ $\mathbf{Y}_P$ = **M**.

Theorem 1. *The syndrome of Y is exactly equal to the secret message block M.*

Proof. Since S(**Y**) = S(**X**) $\oplus$ S(**T**) and S(**T**) = P($\mathbf{T}_{IB}$) $\oplus$ $\mathbf{T}_P$ = P($\mathbf{0}_{4\times4}$) $\oplus$ $\mathbf{T}_P$ = **TS**, hence S(**X**) $\oplus$ S(**T**) = S(**X**) $\oplus$ **TS** = **M**, where $\mathbf{T}_{IB}$ and $\mathbf{T}_P$ respectively denote the **IB** and **P** parts of **T**; and the code array of a zero information array is a zero code array, that is the parity check part is a zero block.□

3.2. Optimal Solution

In Section 2, suppose the toggle syndrome **ts** = (011) and the toggle sequence **t** = (0110000) is constructed by directly appending the four information bits into **ts**, then the found toggle sequence **t** is not the coset leader **e** = (0000100). Further, in Table 1, we know the coset leader **e** is equal to the toggle sequence **t** plus the codeword (0110100), which is closest to **t**. Hence, the toggle array **T** found in *Step 3* of Section 3.1.1 and illustrated in the form of Figure 4b is not a good solution. Usually, it is not a coset leader. Suppose we are allowed to sequentially add up to 2^{16} code arrays of the (49, 16) product code into the toggle array **T**. Then, among those 2^{16} resultant arrays, the one with the minimum Hamming weight is the coset leader, where the Hamming weight of a binary array is defined as the number of 1s in the array. Here, the found coset leader for a 2-D linear code is termed the optimal toggle array.

Since table lookup is impractical due to the issue that the size of the standard array for the (49, 16) product code is rather large, and finding the coset leader of the coset to which **T** belongs is a time-consuming process, so finding the sub-optimal approach is necessary.

3.3. Weight Approximation Embedding Scheme

Wang et al. [11] proposed a sub-optimal approach called the weighted approximation embedding scheme with low computational complexity to find a toggle array with smaller Hamming weight. The details of this scheme are outlined, as follows:

Step 0: Initially, those 16 code arrays, each of which owns only one bit 1 in the IB part, are chosen and included in the set *PO*, as shown in Figure 5.

Step 1: Sequentially add each code array in the set *PO* to **T**. Among those resultant arrays, let **T′** be the resultant array with the smallest Hamming weight and $\mathbf{T}' = \boldsymbol{c} \oplus \mathbf{T}$ for some code array $\boldsymbol{c}$. If the Hamming weight of **T′** is smaller than that of **T**, then replace **T** with **T′**, remove $\boldsymbol{c}$ from *PO*, and repeat *Step 1*. Otherwise, go to *Step 2*.

Step 2: **T** is the found toggle array with a smaller Hamming weight.

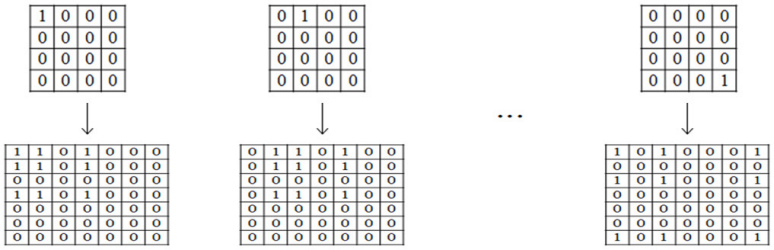

Figure 5. 16 chosen code arrays of the (49, 16) product code.

Note, all the above three methods are categorized as the 2-D toggle syndrome-based embedding scheme, and the 2-D syndrome S(**X**) is the unique embedding channel with 33 bits in total in this kind of scheme.

4. An Improved Product Code-Based Data Hiding Scheme

Inspired by the ideas of [5,6] and [11], in this section, we eventually design a sub-optimal scheme having three embedding channels to find a better solution with low computational complexity.

First, we propose a pure 1-D toggle syndrome-based scheme (1DTS-1) with two embedding channels, as shown in the Appendix A. Based on the one-off decision technique that was proposed in 1DTS-1, we then propose a better sub-optimal scheme (2DTS-1) with three embedding channels. In 2DTS-1, there are two phases in dealing with the construction of the toggle array. In the first phase (*Step 1~ Step 5*), based on the concept of 1-D toggle syndrome and the properties (Property 1 and the Property 2), a toggle array **T′** with better structure is generated, which is more suitable for being assigned as the initial toggle array in the weighted approximation embedding scheme. In the second phase (*Step 6*), based on the Property 2 and the concept of 2-D toggle syndrome, the number of 1s in **T′** can be further reduced through the technique of the weighted approximation embedding scheme to obtain the final toggle array **T″**.

4.1. Embedding

Step 1: Calculate the syndromes of the last four rows in the cover array **X**, i.e., $\mathbf{s}_i = [x_{i+3,1}\ x_{i+3,2} \ldots x_{i+3,7}] \times \mathbf{H}^{\mathrm{T}}$, $i = 1\sim 4$, where ($\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$, $\mathbf{s}_4$) is the first embedding channel with 12 bits in total. The toggle syndrome $\mathbf{ts}_i = \mathbf{s}_i \oplus \mathbf{m}_i$, $i = 1\sim 4$. From Table 1, respectively, find the corresponding coset leaders $\mathbf{e}_i$ (i.e., the toggle sequence $\mathbf{t}_i$), $i = 1\sim 4$. The four toggle sequences constitute a 4×7 array with each column denoted as $\boldsymbol{f}_i$, $i = 1\sim 7$. Denote $[\mathbf{f}_4\mathbf{f}_5\mathbf{f}_6\mathbf{f}_7]$ as **F**.

Step 2: Calculate the syndromes of the last four columns in **X**, i.e., $\mathbf{s}_i = [x_{1,i-1}\ x_{2,i-1} \ldots\ x_{7,i-1}] \times \mathbf{H}^{\mathrm{T}}$, i = 5~ 8. ($\mathbf{s}_5^{\mathrm{T}}$, $\mathbf{s}_6^{\mathrm{T}}$, $\mathbf{s}_7^{\mathrm{T}}$, $\mathbf{s}_8^{\mathrm{T}}$) is the second embedding channel with 12 bits in total. The toggle syndrome $\mathbf{ts}_i = \mathbf{s}_i \oplus \mathbf{m}_i$, i = 5~8.
Step 3: Make the syndromes of the last four rows and the last four columns of the resultant stego array after embedding respectively equal $\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{m}_3$, $\mathbf{m}_4$, and $\mathbf{m}_5$, $\mathbf{m}_6$, $\mathbf{m}_7$, $\mathbf{m}_8$. This can be done through Table 2 (instead of finding the coset leader) to find the toggle sequence $\mathbf{t}_i$ (under the constraint the last four bits of $\mathbf{t}_{\mathrm{i}}$ and the four bits of $\mathbf{f}_{\mathrm{i}-1}$ are equal) with its toggle syndrome equal to $\mathbf{ts}_i$, i = 5~8.
Step 4: First, calculate the CCH part of the parity $\mathrm{P}(\mathbf{X}_{\mathrm{IB}} \oplus \mathbf{F})$ denoted as $\mathrm{P}(\mathbf{X}_{\mathrm{IB}} \oplus \mathbf{F})_{\mathrm{CCH}}$, then we obtain the syndrome of size 3 × 3 which equals to $\mathrm{P}(\mathbf{X}_{\mathrm{IB}} \oplus \mathbf{F})_{\mathrm{CCH}} \oplus \mathbf{X}_{\mathrm{CCH}}$. Let the three columns of the syndrome be denoted, respectively, as $\mathbf{s}_i^{\mathrm{T}}$, $9 \le i \le 11$, where ($\mathbf{s}_9^{\mathrm{T}}$, $\mathbf{s}_{10}^{\mathrm{T}}$, $\mathbf{s}_{11}^{\mathrm{T}}$) is the third embedding channel with nine bits in total. Second, calculate the toggle syndrome $\mathbf{ts}_i^{\mathrm{T}}$, which equals to $\mathbf{s}_i^{\mathrm{T}} \oplus \mathbf{m}_i^{\mathrm{T}}$, $9 \le i \le 11$. Third, those column matrices $\mathbf{f}_{\mathrm{i}}$, i = 1~ 3,$\mathbf{t}_i^{\mathrm{T}}$, $5 \le i \le 8$, and $\mathbf{ts}_i^{\mathrm{T}}$, i = 9~ 11, constitute the toggle array **T**, as shown in Figure 6.
Step 5: To reduce the number of 1s in **T**, first check the last four columns of **T** and whether there is more than one column with a Hamming weight in each column of greater than 2. If it does not exist, just skip this step. Otherwise, take the element-wise OR operation among those columns to obtain a new vector **u**. Then, using the lookup of Table 1, find a codeword **c**, which is closest to **u** in terms of minimal Hamming distance. Add (XOR operation) the codeword **c** to each of those columns. Then, it is very likely some of the rows of T have been changed. Let **w** be the induced non-zero vector with the non-zero component denoting the changed position of any changed row. From Table 1, find the coset leader **e** of the coset to which the vector **w** belongs. Hence, to keep the syndrome of the toggle array unchanged, we need to add **e** respectively to those changed rows. Suppose that the bit 1 is located in the i-th component of **e**, once codeword **c** is also added into the i-th column of **T**, then the syndrome of the new toggle array remains the same as **T**. Denote the new toggle array as **T′**. If the Hamming weight of **T′** is not smaller than that of **T**, **T′** is changed back to **T**.
Step 6: Utilize the technique of weight approximation to further reduce the number of 1s in **T′** to get the toggle array **T″**.
Step 7: Perform $\mathbf{X} \oplus \mathbf{T''} = \mathbf{Y}$ to finish the embedding.

Theorem 2. *The technique described in Step 5 can guarantee the syndrome of* ***T′*** *will be the same as that of* ***T****.*

Proof. In *Step 5*, first the task of adding **e** respectively to those changed rows can be easily accomplished by adding the codeword **c** into the i-th column of **T**; second, $\mathbf{w} \oplus \mathbf{e}$ is a codeword. Then, the net vector added into any row or column of T is a codeword; hence, the syndrome remains unchanged.□

Suppose the probability of each secret bit with a value of 1 or 0 equals 1/2 (this can be done easily by encrypting the secret bit sequence with a pseudo-random bit sequence by the XOR operation). Whatever the probability of each cover bit 1 or 0, the probability of each toggle syndrome bit with the value of 1 or 0 will equal 1/2. That is, each coset leader will be evenly selected. Hence, in Figure 6, the last four rows of T (i.e., the four coset leaders) will most likely differ. That is, the last four bits of each column of the rightmost four columns in the toggle array **T** own at most one bit 1. Then, we have:

Property 1. *For each of the four toggle sequences* $\mathbf{t}_i^{\mathrm{T}}, 5 \le i \le 8$, *in upright form, most of the bits 1 are concentrated in the top three bits. Hence, the resultant sequence constructed by performing the component-wise OR operation on the toggle sequences with the Hamming weight greater than 2 is almost the same as each of the original toggle sequences. Therefore, the number of 1s in T can be reduced almost at each time through the technique described in Step 5 and most of the bits (1s) are located in the upper left corner of 3 × 3 sub-array of the new toggle array* ***T′****.*

Further, due to the property 2, while applying the technique of weight approximation to the toggle array **T′**, the number of 1s in T′ can be further reduced.

Property 2. *For each of the 16 chosen code arrays shown in Figure 5, the bits in the upper left corner of 3 × 3 sub-array of the code array have a higher density of 1.*

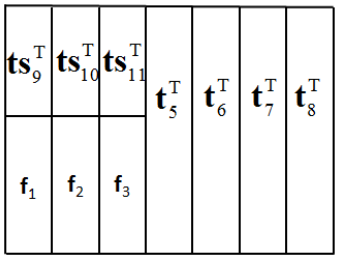

Figure 6. Toggle array **T**.

Table 2. Toggle sequence $\mathbf{t}_i$, i = 5~8.

$\mathbf{f}_{i-1}^T$	$\mathbf{ts}_i$							
	000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	0000000	0010000	0100000	0110000	1000000	1010000	1100000	1110000
0001	1010001	1000001	1110001	1100001	0010001	0000001	0110001	0100001
0010	1110010	1100010	1010010	1000010	0110010	0100010	0010010	0000010
0011	0100011	0110011	0000011	0010011	1100011	1110011	1000011	1010011
0100	0110100	0100100	0010100	0000100	1110100	1100100	1010100	1000100
0101	1100101	1110101	1000101	1010101	0100101	0110101	0000101	0010101
0110	1000110	1010110	1100110	1110110	0000110	0010110	0100110	0110110
0111	0010111	0000111	0110111	0100111	1010111	1000111	1110111	1100111
1000	1101000	1111000	1001000	1011000	0101000	0111000	0001000	0011000
1001	0111001	0101001	0011001	0001001	1111001	1101001	1011001	1001001
1010	0011010	0001010	0111010	0101010	1011010	1001010	1111010	1101010
1011	1001011	1011011	1101011	1111011	0001011	0011011	0101011	0111011
1100	1011100	1001100	1111100	1101100	0011100	0001100	0111100	0101100
1101	0001101	0011101	0101101	0111101	1001101	1011101	1101101	1111101
1110	0101110	0111110	0001110	0011110	1101110	1111110	1001110	1011110
1111	1111111	1101111	1011111	1001111	0111111	0101111	0011111	0001111

4.2. Extracting

The procedure of extraction is the same as that in Section 3.2.

4.3. Example

Given **X** and **M**, and they are respectively as follows:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & & & & \\ 1 & 1 & 1 & & & & \\ 1 & 0 & 1 & & & & \\ 0 & 1 & 1 & & & & \end{bmatrix}.$$

4.3.1. Embedding

Step 1: From ($\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$, $\mathbf{s}_4$) = (010, 011, 010, 000) and ($\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{m}_3$, $\mathbf{m}_4$) = (011, 111, 101, 011), we obtain ($\mathbf{ts}_1$, $\mathbf{ts}_2$, $\mathbf{ts}_3$, $\mathbf{ts}_4$) = ($\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$, $\mathbf{s}_4$) $\oplus$ ($\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{m}_3$, $\mathbf{m}_4$) = (001, 100, 111, 011). Hence, ($\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$, $\mathbf{e}_4$) = (0010000, 1000000, 0000010, 0000100). Then,

$$[\mathbf{f}_1\mathbf{f}_2\mathbf{f}_3\mathbf{f}_4\mathbf{f}_5\mathbf{f}_6\mathbf{f}_7] = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \text{ and } \mathbf{F} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Step 2: Since ($\mathbf{s}_5$, $\mathbf{s}_6$, $\mathbf{s}_7$, $\mathbf{s}_8$) = (001, 000, 010, 111) and ($\mathbf{m}_5$, $\mathbf{m}_6$, $\mathbf{m}_7$, $\mathbf{m}_8$) = (100, 010, 110, 101), therefore ($\mathbf{ts}_5$, $\mathbf{ts}_6$, $\mathbf{ts}_7$, $\mathbf{ts}_8$) = ($\mathbf{s}_5$, $\mathbf{s}_6$, $\mathbf{s}_7$, $\mathbf{s}_8$) $\oplus$ ($\mathbf{m}_5$, $\mathbf{m}_6$, $\mathbf{m}_7$, $\mathbf{m}_8$) = (101, 010, 100, 010).
Step 3: For each i, $5 \leq i \leq 8$, given $\mathbf{ts}_i$ and $\mathbf{f}_{i-1}$, $\mathbf{t}_i$ can be obtained respectively through the lookup in Table 2 as follows: $\mathbf{t}_5$ = [1010000], $\mathbf{t}_6$ = [1110001], $\mathbf{t}_7$ = [0110010], $\mathbf{t}_8$ = [0100000].
Step 4: X_{IB}, $\mathbf{X}_{IB} \oplus \mathbf{F}$, $P(\mathbf{X}_{IB} \oplus \mathbf{F})_{CCH}$, $\mathbf{X}_{CCH}$, and $P(\mathbf{X}_{IB} \oplus \mathbf{F})_{CCH} \oplus \mathbf{X}_{CCH}$ are respectively as follows:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Since ($\mathbf{s}_9$, $\mathbf{s}_{10}$, $\mathbf{s}_{11}$) = (101, 000, 110) and ($\mathbf{m}_9$, $\mathbf{m}_{10}$, $\mathbf{m}_{11}$) = (100, 011, 000), therefore ($\mathbf{ts}_9$, $\mathbf{ts}_{10}$, $\mathbf{ts}_{11}$) = ($\mathbf{s}_9$, $\mathbf{s}_{10}$, $\mathbf{s}_{11}$) $\oplus$ ($\mathbf{m}_9$, $\mathbf{m}_{10}$, $\mathbf{m}_{11}$) = (001, 011, 110).

Now,

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

Step 5: For the last four columns of **T**, the fifth and sixth columns, of each has a Hamming weight of greater than 2. Take the element-wise OR operation between both columns, we obtain **u** = (1110011). **c** = (1110010) is the closest codeword to **u**. Then, **c** (in the form of column vector) is added sequentially to the fifth and sixth columns. This is equivalent to vector **w** = (0000110) being added implicitly into the first, second, third, and sixth rows. The coset leader **e** = (1000000) of the coset to which **w** belongs is determined and the bit 1 is located in the first component. Therefore, the codeword **c** is also added to the 1st column of **T**. Now, the toggle array T is updated as T′ with a smaller Hamming weight.

$$\mathbf{T}' = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Note

$$\mathrm{S}(\mathbf{T}) = \mathrm{P}(\mathbf{T}_{\mathrm{IB}}) \oplus \mathbf{T}_{\mathrm{P}} = \begin{bmatrix} 1&0&0&0&1&1&0 \\ 1&1&1&0&0&1&0 \\ 1&0&0&0&1&1&0 \\ 0&0&0 & & & & \\ 0&0&0 & & & & \\ 1&1&1 & & & & \\ 0&1&1 & & & & \end{bmatrix} \oplus \begin{bmatrix} 0&0&1&1&1&0&0 \\ 0&1&1&0&1&1&1 \\ 1&1&0&1&1&1&0 \\ 0&0&1 & & & & \\ 1&0&0 & & & & \\ 0&0&0 & & & & \\ 0&0&0 & & & & \end{bmatrix} = \begin{bmatrix} 1&0&1&1&0&1&0 \\ 1&0&0&0&1&0&1 \\ 0&1&0&1&0&0&0 \\ 0&0&1 & & & & \\ 1&0&0 & & & & \\ 1&1&1 & & & & \\ 0&1&1 & & & & \end{bmatrix}.$$

$$\mathrm{S}(\mathbf{T}') = \mathrm{P}(\mathbf{T}'_{\mathrm{IB}}) \oplus \mathbf{T}'_{\mathrm{P}} = \begin{bmatrix} 0&0&0&0&0&0&0 \\ 0&1&1&0&1&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0 & & & & \\ 0&0&0 & & & & \\ 0&1&1 & & & & \\ 0&1&1 & & & & \end{bmatrix} \oplus \begin{bmatrix} 1&0&1&1&0&1&0 \\ 1&1&1&0&0&0&1 \\ 0&1&0&1&0&0&0 \\ 0&0&1 & & & & \\ 1&0&0 & & & & \\ 1&0&0 & & & & \\ 0&0&0 & & & & \end{bmatrix} = \begin{bmatrix} 1&0&1&1&0&1&0 \\ 1&0&0&0&1&0&1 \\ 0&1&0&1&0&0&0 \\ 0&0&1 & & & & \\ 1&0&0 & & & & \\ 1&1&1 & & & & \\ 0&1&1 & & & & \end{bmatrix}.$$

Hence, S(**T**) = S(**T′**) = **TS**, where

$$\mathrm{S}(\mathbf{X}) = \mathrm{P}(\mathbf{X}_{\mathrm{IB}}) \oplus \mathbf{X}_{\mathrm{P}} = \begin{bmatrix} 0&1&1&0&1&0&0 \\ 0&0&0&1&1&0&1 \\ 0&0&1&0&1&1&1 \\ 1&1&1 & & & & \\ 1&0&1 & & & & \\ 0&1&0 & & & & \\ 1&1&0 & & & & \end{bmatrix} \oplus \begin{bmatrix} 0&1&0&0&1&0&1 \\ 1&1&0&1&1&1&0 \\ 0&0&1&1&1&1&0 \\ 1&0&1 & & & & \\ 1&1&0 & & & & \\ 0&0&0 & & & & \\ 1&1&0 & & & & \end{bmatrix} = \begin{bmatrix} 0&0&1&0&0&0&1 \\ 1&1&0&0&0&1&1 \\ 0&0&0&1&0&0&1 \\ 0&1&0 & & & & \\ 0&1&1 & & & & \\ 0&1&0 & & & & \\ 0&0&0 & & & & \end{bmatrix}$$

and

$$\mathbf{TS} = \mathrm{S}(\mathbf{X}) \oplus \mathbf{M} = \begin{bmatrix} 1&0&1&1&0&1&0 \\ 1&0&0&0&1&0&1 \\ 0&1&0&1&0&0&0 \\ 0&0&1 & & & & \\ 1&0&0 & & & & \\ 1&1&1 & & & & \\ 0&1&1 & & & & \end{bmatrix}.$$

Step 6: **T′** is further refined as **T″** by adding the 4th code array in the first round to the toggle array **T′** while combining the technique of the weight approximation embedding scheme. In the second round, there are no code arrays remaining in set *PO*, which can be further used to refine the toggle array **T″**.

$$\mathbf{T}'' = \begin{bmatrix} 1&0&1&1&0&1&0 \\ 1&1&1&0&0&0&1 \\ 0&1&0&1&0&0&0 \\ 0&0&1&0&0&0&0 \\ 1&0&0&0&0&0&0 \\ 1&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0 \end{bmatrix} \oplus \begin{bmatrix} 1&0&1&0&0&0&1 \\ 1&0&1&0&0&0&1 \\ 0&0&0&0&0&0&0 \\ 1&0&1&0&0&0&1 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \end{bmatrix} = \begin{bmatrix} 0&0&0&1&0&1&1 \\ 0&1&0&0&0&0&0 \\ 0&1&0&1&0&0&0 \\ 1&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0 \\ 1&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0 \end{bmatrix}$$

Step 7: Finally, the stego array **Y** is obtained by adding **T″** to **X**.

4.3.2. Extracting

Y, P($\mathbf{Y}_{\mathrm{IB}}$), and S(**Y**), i.e., message **M** are respectively as follows:

$$\begin{bmatrix} 0&1&0&1&1&1&0\\ 1&0&0&1&1&1&0\\ 0&1&1&0&1&1&0\\ 0&0&1&0&0&1&1\\ 0&1&0&0&0&0&1\\ 1&0&0&1&0&1&0\\ 1&1&0&1&1&0&0 \end{bmatrix}, \begin{bmatrix} 1&1&0&0&1&0&1\\ 1&1&0&1&0&0&0\\ 0&0&1&0&1&1&1\\ 0&1&0\\ 1&0&1\\ 0&0&1\\ 1&0&1 \end{bmatrix}, \begin{bmatrix} 1&0&0&1&0&1&1\\ 0&1&0&0&1&1&0\\ 0&1&0&0&0&0&1\\ 0&1&1\\ 1&1&1\\ 1&0&1\\ 0&1&1 \end{bmatrix}.$$

5. Proposed Scheme Combined with Hamming+1

Zhang et al. [14] proposed the Hamming+1 concept in 2007 to increase the embedding rate and the embedding efficiency. For the Hamming code $(2^t - 1, 2^t - t - 1)$ with the strategy of Hamming+1, through checking (1a) and (1b), the secret message $\mathbf{m} = (m_1, m_2, \ldots, m_{t+1})$ of $t + 1$ bits can be hidden into 2^t pixels of that pixel values equal $(p_1, \ldots, p_{2^t})$, and at most, one pixel value will be changed by increasing or decreasing one:

$$[m_1, \ldots, m_t] = [\mathrm{LSB}(p_1),\ \mathrm{LSB}(p_2),\ \ldots,\ \mathrm{LSB}(p_{2^t-1})] \times H^T \tag{1a}$$

$$m_{t+1} = \mathrm{SLSB}(p_1) \oplus \mathrm{SLSB}(p_2) \oplus \ldots \oplus \mathrm{SLSB}(p_{2^t-1}) \oplus \mathrm{LSB}(p_{2^t}) \tag{1b}$$

where LSB(.) and SLSB(.) respectively represent the first and the second LSB slicing of the pixel value, and $\oplus$ represents the bit-wise exclusive-or operation. When only (1a) does not hold, the lowest bit of a certain pixel value p_i needs to be changed to make (1a) hold, where $1 \le i \le 2^t - 1$, and the status of (1-2) is not affected. When only (1b) does not hold, the lowest bit of the pixel value p_{2^t} needs to be changed. When neither of (1a) and (1b) hold, once a certain pixel value p_i, $1 \le i \le 2^t - 1$, is increased or decreased by one, then (1a) and (1b) hold simultaneously. Here, $t = 3$.

Suppose the proposed scheme introduced in the previous section allows for modification to the other bit of the pixel. We consider the block of 7×8 pixels formed by adding one more pixel for each row of the original 7×7 block. For the 7×8 block, the Hamming+1 technique can be used to let each row hide one more secret bit and the overall distortion is usually not increased, that is the value of the new pixel is unchanged.

Let the additional seven secret bits be denoted as $(m_1, m_2, \ldots, m_7)$. Let **PX** be the cover array of 7×8 pixels, i.e., $[px_{i,j}]_{7\times8}$, where $1 \le i \le 7$ and $1 \le j \le 8$. Let the second LSB of each pixel in the first seven columns of **PX** be denoted as $sx_{i,j}$, and the LSB of each pixel in the last column of **PX** be denoted as $x_{i,8}$, where $1 \le i \le 7$ and $1 \le j \le 7$. Let **PY** be the stego array of 7×8 pixels, i.e., $[py_{i,j}]_{7\times8}$, where $1 \le i \le 7$ and $1 \le j \le 8$. Let the second LSB of each pixel in the first seven columns of **PY** be denoted as $sy_{i,j}$, and the LSB of each pixel in the last column of **PY** be denoted as $y_{i,8}$, where $1 \le i \le 7$ and $1 \le j \le 7$.

According to the toggle array T″ that is described in *step* 6 of the embedding process in Section 4.1, the non-zero element of the toggle array **T″** implies the LSB of the corresponding pixel must be changed and its second LSB can also be changed simultaneously by increasing or decreasing the pixel value by one if necessary. Hence, we form Equation (2).

$$z_i = sx_{i,1} \oplus sx_{i,2} \oplus \ldots \oplus sx_{i,7} \oplus x_{i,8},\ 1 \le i \le 7. \tag{2}$$

$(z_1, z_2, \ldots, z_7)$ is the fourth embedding channel which can be used to embed another seven secret bits.

<Embedding>

For i = 1 to 7
If (there exists at least one bit 1 in the row *i* of **T″**)
If (z_i and m_i are not equal)
randomly choose a pixel, said $p_{i,k}$ that must be changed //suppose we choose the first one
If (LSB($p_{i,k}$)==1) $p_{i,k} = p_{i,k} + 1$
Else $p_{i,k} = p_{i,k} - 1$
mark the pixel
Else
If (z_i and m_i are not equal)
change the LSB of $p_{i,8}$
For i = 1 to 7
For j = 1 to 7
If (**T″**$_{ij}$==1) and ($p_{i,j}$ is not marked) // **T″**$_{ij}$denotes the element of *i*-th row and *j*-th column of T″
change the LSB of $p_{i,j}$

<Extracting>

Step 1: ($\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{11}$) is extracted by the same extracting process of Section 4.2.
Step 2:

$$m_i = sy_{i,1} \oplus sy_{i,2} \oplus \ldots \oplus sy_{i,7} \oplus y_{i,8},\ 1 \le i \le 7. \tag{3}$$

Example 1. *The additional seven secret bits ($m_1, m_2, \ldots, m_7$) = (0, 0, 1, 0, 0, 1, 0). The cover array of 7 × 8 pixels, the array of LSB slicing bits for the last column of **PX**, and the array of SLSB slicing bits for the first seven columns of **PX** are respectively as follows:*

$$\mathbf{PX} = \begin{bmatrix} 80 & 81 & 82 & 82 & 83 & 80 & 81 & 81 \\ 81 & 81 & 80 & 81 & 81 & 81 & 82 & 82 \\ 80 & 80 & 81 & 81 & 81 & 81 & 82 & 82 \\ 81 & 80 & 81 & 84 & 84 & 83 & 84 & 84 \\ 81 & 81 & 82 & 84 & 84 & 84 & 85 & 85 \\ 80 & 80 & 82 & 83 & 83 & 83 & 84 & 83 \\ 81 & 81 & 82 & 83 & 82 & 82 & 84 & 83 \end{bmatrix}, \mathbf{X}_8 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \mathbf{SX} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

<Embedding>

From Equation (2), we have ($z_1, z_2, \ldots, z_7$) = (0, 1, 1, 1, 0, 1, 1). Since $m_2 \neq z_2$, $m_4 \neq z_4$, and $m_7 \neq z_7$, so the three pixels $p_{2,2}$, $p_{4,1}$, and $p_{7,5}$ are marked; and $p_{2,2} = p_{2,2} + 1$, $p_{4,1} = p_{4,1} + 1$, and $p_{7,5} = p_{7,5} - 1$. Then, according to the toggle array T″, the LSBs of the other pixels are modified accordingly. Further, there exists at least one bit 1 in each row of **T″**, so all of the pixel values in the last column of **PX** do not need to be changed. Now, we have

$$\mathbf{PY} = \begin{bmatrix} 80 & 81 & 82 & \overline{83} & 83 & \overline{81} & \overline{80} & 81 \\ 81 & \overline{\overline{82}} & 80 & 81 & 81 & 81 & 82 & 82 \\ 80 & \overline{81} & 81 & \overline{80} & 81 & 81 & 82 & 82 \\ \overline{\overline{82}} & 80 & 81 & 84 & 84 & 83 & \overline{85} & 84 \\ \overline{80} & 81 & 82 & 84 & 84 & 84 & 85 & 85 \\ \overline{81} & 80 & 82 & 83 & \overline{82} & 83 & 84 & 83 \\ 81 & 81 & 82 & 83 & \overline{\overline{81}} & 82 & 84 & 83 \end{bmatrix}.$$

<Extracting>

Step 1: From message block **M** of Section 4.3.2, ($\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{m}_3$, $\mathbf{m}_4$, $\mathbf{m}_5$, $\mathbf{m}_6$, $\mathbf{m}_7$, $\mathbf{m}_8$, $\mathbf{m}_9$) = (011, 111, 101, 011, 100, 010, 110, 101, 100, 011, 000).
Step 2: From Equation (3), the additional secret bit vector (m_1, m_2, ... , m_7) = (0, 0, 1, 0, 0, 1, 0) is obtained.

Property 3. *Usually, there is at least one bit 1 in each row of toggle array* $\mathbf{T}''$ *obtained in the proposed 2DTS-1 scheme, hence the additional cover pixels don't need to be modified.*

6. Experimental Results

In the experiment, each embedding scheme was performed for a total of 1 million times. At each time, 49 (or 56) LSB slicing bits, 49 SLSB slicing bits, and 33 (or 40) secret message bits were pseudo-randomly generated.

Let R be the embedding rate ($R = 33/49 = 0.673$) defined by the number of embedding bits per pixel (bpp) and M_a be the expected number of changed bits per embedding. Hence, the average modification for each bit denoted as D equals $M_a/49$. Then, the embedding efficiency α defined as the expected hidden bits per embedding modification equals R/D, and the PSNR (Peak Signal to Noise Ratio) equals $10 \times \log_{10}(255^2/D)$.

6.1. Computational Complexity

For the proposed 1-D Toggle Syndrome-based embedding scheme (1DTS-1) shown in the Appendix A.1, Table 3 shows the average number of different row combinations, that is the average number of times that the loop in *Step 4* will run. If all of the row combinations are considered in *Step 4*, then there will be $C_2^N + C_3^N + \ldots + C_N^N$ different row combinations and a more complicated scheme denoted as 1DTS-2 will have about one more loop than 1DTS-1 to get a gain of about 4 percent in embedding efficiency.

Table 3. Comparisons of computational complexity between 1DTS-1 and 1DTS-2.

1-D Toggle Syndrome-Based Embedding Scheme	The Average Number of Different Row Combinations	α
1DTS-1 (with C_2^N different combinations)	2.1454	2.5744
1DTS-2 (with $C_2^N + C_3^N + \ldots + C_N^N$ different combinations)	3.1529	2.6767

For the proposed hybrid 1-D/2-D Toggle Syndrome-based embedding scheme (2DTS-1) that is presented in Section 4, Table 4 shows the average number of different row combinations that is the average number of times the *Step 5* of Section 4.1 will perform. Similar to the 1DTS-2 scheme (here, $N \in [2 \ldots 4]$), if all kinds of combination are considered, then a more complicated scheme denoted as 2DTS-2 will have about two times the computational complexity of 2DTS-1. However, the gain in the embedding efficiency is only 0.0227. Thus, 2DTS-1 is preferred and the primary scheme with good embedding efficiency and low computational complexity.

Table 4. Comparison of computational complexity between 2DTS-1 and 2DTS-2.

1-D/2-D Toggle Syndrome-Based Embedding Scheme	The Average Number of Different Row Combinations	α
2DTS-1 (with only one combination)	0.4675	2.8261
2DTS-2 (with $C_2^N + C_3^N + \ldots + C_N^N$ different combinations)	0.8495	2.8488

6.2. Performance

In Table 5, except the last solution, the embedding rate for each solution is equal to 33/49 (≈0.6735) for fair comparison. In the original row-column scheme [5], after embedding in each row, only the first three columns are further used to embed data, whereas we adopt the first four columns in our implementation of [5] to enhance the embedding rate, but with larger distortion. The resultant algorithm is termed as Row-Column. Further, the one-off decision technique proposed in *Step 4* of Appendix A.1.1 is used to reduce the distortion and the resultant algorithms are respectively termed as 1DTS-1 and 1DTS-2. In the original row-column scheme, R and D respectively equal 30/49 (≈0.6122) and 0.212 for $x = y = 3$. When $x = 2$ and $y = 8$, R and D respectively equal about 0.6771 and 0.252. The performance in terms of embedding efficiency for this scheme and 1DTS-2 are close to each other. For the former, they divided the pixels of cover image into disjoint matrix blocks of skew and large size 255×3, whereas matrix block of small size 7×7 is adopted in the latter one.

Table 5. Comparison of embedding efficiency among all schemes.

	Embedding Schemes	R	M_a	D	α	Average PSNR (dB)
1-D Toggle Syndrome-based	Row-Column	33/49	13.3084	0.2716	2.4796	53.7915
	Ours (1DTS-1)	33/49	12.8209	0.2616	2.5744	53.9544
	Ours (1DTS-2)	33/49	12.3297	0.2516	2.6767	54.1237
2-D Toggle Syndrome-based	Weight Approximation [11]	33/49	12.1725	0.2484	2.7112	54.1793
1-D/2-D Toggle Syndrome-based	Ours (2DTS-1)	33/49	11.6786	0.2383	2.8261	54.3596
	Ours (2DTS-2)	33/49	11.5854	0.2364	2.8488	54.3943
	Optimal solution	33/49	10.2812	0.2098	3.2100	54.9128
	Ours(2DTS-1) + (Hamming+1)	40/56	11.9699	0.2137	3.3424	54.8328

From the results of Table 5, the embedding efficiency of the 2-D or hybrid 1-D/2-D Toggle Syndrome-based embedding scheme is much higher than that of the 1-D Toggle Syndrome-based embedding scheme. Hence, the former one is preferred. Our proposed scheme (2DTS-1) reduces the average number of modification bits per embedding by 0.4938 when compared to [11]. That is, the average number of hidden bits per embedding modification is increased by 0.1149. Further, due to Property 3, the embedding efficiency can be significantly enhanced when combined with the strategy of Hamming+1.

6.3. Practical Examples

The secret image shown in Figure 7 is independently embedded into six cover images that are respectively shown in Figure 8 by 2DTS-1, and the stego images are shown in Figure 9. The embedding rate equals 0.6685 bpp and the PSNR values of all stego images are around 54.39 dB, as shown in Table 6.

Figure 7. Secret image Finger (148 × 148).

Figure 8. Cover images (512 × 512): Baboon, Boat, Cameraman, Goldhill, Lena, and Peppers.

Figure 9. Stego images (512 × 512): Baboon, Boat, Cameraman, Goldhill, Lena, and Peppers.

Table 6. Stego Image Quality.

Tested Image	Baboon	Boat	Cameraman	Goldhill	Lena	Peppers
PSNR	54.4018	54.3923	54.3850	54.3955	54.4018	54.3716

7. Conclusions

For the product code-based data hiding scheme, in constructing a toggle array, although the optimal solution can be obtained by directly finding the corresponding coset leader for a 2-D toggle

syndrome in 2-D linear codes, the computational complexity is rather tremendous. Usually, it is infeasible. Instead of only utilizing the concept of 2-D toggle syndrome to find the globally optimal solution, we propose a sub-optimal solution with reasonable computational complexity by utilizing the concepts of 1-D toggle syndrome and 2-D toggle syndrome.

First, we utilize the concepts of 1-D toggle syndrome and the algebraic structure of 1-D linear code to build a rough toggle array **T**. Second, the proposed one-off decision technique is used to reduce the number of 1s in **T** in order to obtain a new toggle array **T′**. Finally, due to the property of the structure of **T′**, it is more suitable for applying the strategy of the weighted approximation embedding scheme (2-D toggle syndrome-based) to further refine the toggle array.

Author Contributions: W.-R.Z. conceived and performed the experiments; Y.-M.H. conceived, validated, and wrote the paper.

Funding: This research is funded by the National Science Council, Taiwan, under Grant MOST 104-2221-E-260-007.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A.

The secret message **M** with 33 bits is arranged as 11 groups, $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{11}$, where each $\mathbf{m}_i$ ($1 \leq i \leq 11$) is an array of size 1×3.

Appendix A.1. 1-D Toggle Syndrome-based Embedding Scheme (1DTS-1)

Appendix A.1.1. Embedding

Step 1: Calculate the syndromes of the seven rows in **X**, i.e., $\mathbf{s}_i = [x_{i,1}\ x_{i,2}\ \ldots\ x_{i,7}] \times \mathbf{H}^{\mathrm{T}}$, $i = 1\sim 7$, where $(\mathbf{s}_1,\mathbf{s}_2,\mathbf{s}_3,\mathbf{s}_4,\mathbf{s}_5,\mathbf{s}_6,\mathbf{s}_7)$ is the first embedding channel with 21 bits in total. The toggle syndrome $\mathbf{ts}_i = \mathbf{s}_i \oplus \mathbf{m}_i$, $i = 1\sim 7$.
Step 2: Calculate the syndromes of the first four columns in **X**, i.e., $\mathbf{s}_i = [x_{1,i-7}\ x_{2,i-7}\ \ldots\ x_{7,i-7}] \times \mathbf{H}^{\mathrm{T}}$, $i = 8\sim 11$. $(\mathbf{s}_8^{\mathrm{T}},\mathbf{s}_9^{\mathrm{T}},\mathbf{s}_{10}^{\mathrm{T}},\mathbf{s}_{11}^{\mathrm{T}})$ is the second embedding channel with 12 bits in total. The toggle syndrome $\mathbf{ts}_i = \mathbf{s}_i \oplus \mathbf{m}_i$, $i = 8\sim 11$. From Table 1, respectively find the corresponding coset leaders $\mathbf{e}_i$ (i.e., the toggle sequence $\mathbf{t}_i$), $i = 8\sim 11$. The four toggle sequences (placed in an upright form) constitute a 7×4 array with each row denoted as $\mathbf{f}_i$, $i = 1\sim 7$.
Step 3: Make the syndromes of the seven rows and the first four columns of the resultant stego array after embedding respectively equal $\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, \mathbf{m}_4, \mathbf{m}_5, \mathbf{m}_6, \mathbf{m}_7$, and $\mathbf{m}_8, \mathbf{m}_9, \mathbf{m}_{10}, \mathbf{m}_{11}$. For the toggle sequence $\mathbf{t}_i$, $i = 1\sim 7$, instead of finding the coset leaders of the cosets to which $\mathbf{ts}_i$, $i = 1\sim 7$, respectively belongs, they are found by the lookup of Table A1 with the rule that the first four bits of $\mathbf{t}_i$ and the four bits of $\mathbf{f}_i$ are equal. Those seven toggle sequence $\mathbf{t}_i$, $i = 1\sim 7$, constitute a toggle array **T** of size 7×7.
// The following proposed one-off decision technique is used to reduce the number of 1s in **T**.
Step 4: To reduce the number of 1s in **T**, first check the seven rows of **T** and whether there is more than one row with a Hamming weight in each row of greater than 2. If it does not exist, jump out to the next step. Otherwise, suppose there are N rows ($N \in [2\ \ldots\ 7]$), each of which has a Hamming weight greater than 2, and two rows are chosen randomly at a time. Hence, there are C_2^N different row combinations. For each of those row combinations, do:

Take the component-wise OR operation among those rows to obtain a new vector **u**. Then, utilizing the lookup of Table 1, find a codeword **c** which is closest to **u** in terms of minimal Hamming distance. Add sequentially the codeword **c** to each of those rows. Then, it is very likely some of the first four columns of T have been changed. Let **w** be the induced non-zero vector with the non-zero component denoting the changed position of any changed column. From Table 1, find the coset leader **e** of the coset to which the vector **w** belongs. Hence, to keep the syndromes unchanged for the first four columns of the toggle array, we need to add **e** respectively to those changed columns. Suppose the bit 1 is located in the i-th component of **e**, hence once codeword **c** is also added into the i-th row

of **T**, then all of the syndromes of the seven rows and the first four columns of the new toggle array remain the same as **T**. Denote the new toggle array as **T′**.

Among those new toggle arrays **T′**s found in the above loop, choose the new toggle array denoted as $\mathbf{T}'_{min}$ with the minimum Hamming weight. If the Hamming weight of $\mathbf{T}'_{min}$ is smaller than that of **T**, replace **T** with $\mathbf{T}'_{min}$.

Step 5: Perform the element-wise XOR operation between **X** and **T** to get the stego array **Y**, i.e., **Y**= **X** $\oplus$ **T**.

Appendix A.1.2. Extracting

In the receiving end, once the 7×7 slicing bits (**Y**) of the stego image block is got, the secret messages $\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{m}_3$, $\mathbf{m}_4$, $\mathbf{m}_5$, $\mathbf{m}_6$, $\mathbf{m}_7$, and $\mathbf{m}_8$, $\mathbf{m}_9$, $\mathbf{m}_{10}$, $\mathbf{m}_{11}$ can be extracted easily by respectively calculating the syndromes of the seven rows of **Y** and the first four columns of **Y**.

Appendix A.2. Example

Given the same X and M show in Section 4.3, but here M is not arranged as a L-shaped block.

Appendix A.2.1. Embedding

Step 1: From ($\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$, $\mathbf{s}_4$, $\mathbf{s}_5$, $\mathbf{s}_6$, $\mathbf{s}_7$) = (100, 100, 011, 010, 011, 010, 000) and ($\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{m}_3$, $\mathbf{m}_4$, $\mathbf{m}_5$, $\mathbf{m}_6$, $\mathbf{m}_7$) = (011, 111, 101, 011, 100, 010, 110), we obtain ($\mathbf{ts}_1$, $\mathbf{ts}_2$, $\mathbf{ts}_3$, $\mathbf{ts}_4$, $\mathbf{ts}_5$, $\mathbf{ts}_6$, $\mathbf{ts}_7$) = (111, 011, 110, 001, 111, 000, 110).

Step 2: From ($\mathbf{s}_8$, $\mathbf{s}_9$, $\mathbf{s}_{10}$, $\mathbf{s}_{11}$) = (010, 000, 111, 001) and ($\mathbf{m}_8$, $\mathbf{m}_9$, $\mathbf{m}_{10}$, $\mathbf{m}_{11}$) = (101, 100, 011, 000), we have ($\mathbf{ts}_8$, $\mathbf{ts}_9$, $\mathbf{ts}_{10}$, $\mathbf{ts}_{11}$) = (111, 100, 100, 001). Hence, ($\mathbf{t}_8$, $\mathbf{t}_9$, $\mathbf{t}_{10}$, $\mathbf{t}_{11}$) = ($\mathbf{e}_8$, $\mathbf{e}_9$, $\mathbf{e}_{10}$, $\mathbf{e}_{11}$) = (0000010, 1000000, 1000000, 0010000). Then,

$$\left[\mathbf{f}_1^{\mathrm{T}}\mathbf{f}_2^{\mathrm{T}}\mathbf{f}_3^{\mathrm{T}}\mathbf{f}_4^{\mathrm{T}}\mathbf{f}_5^{\mathrm{T}}\mathbf{f}_6^{\mathrm{T}}\mathbf{f}_7^{\mathrm{T}}\right] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Step 3: For each i = 1~ 7, given $\mathbf{ts}_i$ and $\mathbf{f}_i$, $\mathbf{t}_i$ can be obtained respectively through the lookup in Table A1 as follows: $\mathbf{t}_1$ = [0110110], $\mathbf{t}_2$ = [0000100], $\mathbf{t}_3$ = [0001000], $\mathbf{t}_4$ = [0000111], $\mathbf{t}_5$ = [0000010], $\mathbf{t}_6$ = [1000110], $\mathbf{t}_7$ = [0000101]. Now,

$$\mathbf{T} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

Step 4: For the seven rows of **T**, the first, forth, and sixth rows, each of which has a Hamming weight greater than 2. Suppose we choose the first and forth rows, take the element-wise OR operation between both rows, we obtain **u** = (0110111). **c** = (0010111) is the closest codeword to **u**. Then, **c** is added sequentially to the first and forth rows. This is equivalent to vector **w** = (1001000) being added implicitly into the 3rd, 5th, 6th, and 7th columns. The coset leader **e** = (0100000) of the coset to which **w** belongs is determined and the bit 1 is located in the second component. Therefore, the

codeword **c** is also added to the 2nd row of **T**. Now, the toggle array T is updated as T′ with smaller Hamming weight.

$$\mathbf{T}' = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Step 5: Finally, the stego array **Y** is obtained by adding **T′** to **X**.

$$\mathrm{Y} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Appendix A.2.2. Extracting

The syndromes of the seven rows of **Y** and the first four columns of **Y** can be obtained respectively as (011, 111, 101, 011, 100, 010, 110) and (101, 100, 011, 000).

Table A1. Toggle sequence $\mathbf{t}_i$, i = 1~7.

$\mathbf{f}_i$	$\mathbf{ts}_i$							
	000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	0000000	0000111	0000011	0000100	0000110	0000001	0000101	0000010
0001	0001101	0001010	0001110	0001001	0001011	0001100	0001000	0001111
0010	0010111	0010000	0010100	0010011	0010001	0010110	0010010	0010101
0011	0011010	0011101	0011001	0011110	0011100	0011011	0011111	0011000
0100	0100011	0100100	0100000	0100111	0100101	0100010	0100110	0100001
0101	0101110	0101001	0101101	0101010	0101000	0101111	0101011	0101100
0110	0110100	0110011	0110111	0110000	0110010	0110101	0110001	0110110
0111	0111001	0111110	0111010	0111101	0111111	0111000	0111100	0111011
1000	1000110	1000001	1000101	1000010	1000000	1000111	1000011	1000100
1001	1001011	1001100	1001000	1001111	1001101	1001010	1001110	1001001
1010	1010001	1010110	1010010	1010101	1010111	1010000	1010100	1010011
1011	1011100	1011011	1011111	1011000	1011010	1011101	1011001	1011110
1100	1100101	1100010	1100110	1100001	1100011	1100100	1100000	1100111
1101	1101000	1101111	1101011	1101100	1101110	1101001	1101101	1101010
1110	1110010	1110101	1110001	1110110	1110100	1110011	1110111	1110000
1111	1111111	1111000	1111100	1111011	1111001	1111110	1111010	1111101

References

1. Crandall, R. Some notes on Steganography. Posted on Steganography Mailing List. 1998. Available online: http://dde.binghamton.edu/download/Crandall_matrix.pdf (accessed on 30 October 2018).
2. Westfeld, A. F5—A steganographic algorithm. In Proceedings of the 4th International Workshop on Information Hiding (IHW'01), Pittsburgh, PA, USA, 25–27 April 2001; Volume 2137, pp. 289–302.
3. Bierbrauer, J.; Fridrich, J. Constructing good covering codes for applications in steganography. In *Transactions on Data Hiding and Multimedia Security III*; Springer: Berlin/Heidelberg, Germany, 2008; Volume 4920, pp. 1–22.
4. Zhang, W.; Li, S. A coding problem in steganography. *Des. Codes Cryptogr.* **2008**, *46*, 68–81. [CrossRef]
5. Rifa-Pous, H.; Rifa, J. Product perfect codes and steganography. *Dig. Signal Process.* **2009**, *19*, 764–769. [CrossRef]
6. Zhao, Z.; Gao, F. An improved steganographic method of product perfect codes. In Proceedings of the IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), Xi'an, China, 14–16 September 2011.
7. Rifa-Pous, H.; Rifa, J.; Ronquillo, L. Perfect Z2Z4-linear Codes in Steganography. February 2010. Available online: https://arxiv.org/pdf/1002.0026.pdf (accessed on 30 October 2018).
8. Rifa, J.; Ronquillo, L. Product Perfect Z2Z4-linear codes in steganography. In Proceedings of the 2010 International Symposium on Information Theory and its Applications, Taiwan, 17–20 October 2010; pp. 696–701.
9. Borges, J.; Fernandez, C.; Pujol, J.; Rifa, J.; Villanueva, M. Z2Z4-linear codes: generator matrices and duality. *Des. Codes Cryptogr.* **2010**, *54*, 167–179. [CrossRef]
10. Rifa, J.; Zinoviev, V.A. New completely regular q-ary codes based on Kronecker products. *IEEE Trans. Inf. Theory* **2010**, *56*, 266–272. [CrossRef]
11. Wang, J.J.; Chen, H.; Chang, W.W. Binary data hiding using product code with sub-optimal algorithm. In Proceedings of the National Symposium on Telecommunications, Taiwan, 3–4 December 2010.
12. Zhang, L.; Li, H. A product code in steganography with improved embedding rate. In Proceedings of the 2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference, Chongqing, China, 20–22 May 2016; pp. 246–250.
13. Lin, S.; Costello, D.J. *Error Control Coding: Fundamental and Applications*; Pearson Prentice Hall: Upper Saddle River, NJ, USA, 2004.
14. Zhang, W.; Wang, S.; Zhang, X. Improving embedding efficiency of covering codes for applications in steganography. *IEEE Commun. Lett.* **2007**, *11*, 680–682. [CrossRef]