Error Spreading Control in Steganographic Embedding Schemes Using UEP Codes

Ching-Nung Yang and Guo-Jau Chen
Department of Computer Science and Information Engineering
National Dong Hwa University
#1, Da Hsueh Rd, Sec. 2, Shou-Feng, Hualien, Taiwan, R.O.C.
Tel: 886-3-8634025 Fax: 886-3-8634010
E-mail: cnyang@mail.ndhu.edu.tw

Abstract- A steganographic encoding scheme, *Coded LSB (Least Significant Bit)* method, was first proposed by Dijk and Willems. The scheme can alter less bits to hide the secret than the conventional *LSB* scheme, and meantime the distortion is reduced and the security against steganalysis is improved. Errors may be caused due to the channel noise or the tampering by the active warden; however, for the *Coded LSB* scheme one error bit may result in multi-bit error when extraction. This is called as *error spreading*. In this paper we propose unequal error protection (UEP) codes strategy to overcome this *error spreading* problem. Our method will save the parity bits when compared to the scheme using the conventional error correcting codes directly.

Keywords: Steganography, error correcting code, unequal error protection code.

1. Introduction

Steganography is a technique to conceal the secret under the communication media in which the secret cannot be disclosed. Hiding information in a gray image is commonly discussed in steganography. For a plain *LSB-Steganography* scheme, if the grayscale value of a pixel is x with binary form $(x_{n-1}, \cdots, x_1, x_0)$ where $x = \sum_{i=0}^{n-1} 2^i x_i$, one can simply modify the least significant bit x_0 to embed the secret bit s by $x_0 \oplus s$. A more efficient method, *coded LSB* method, based on encoding LSBs was proposed to reduce the distortion when noise occurs [1]. However, there will be an *error spreading* problem in the *coded LSB* method. One bit error in some LSBs will cause two or more errors when decoding and this will damage the clearness of the recovered secret image. A trivial method for solving this problem is to add the error correcting (EC) capability into the *Coded LSB* scheme. Here, we propose a more reasonable solution for solving the *error spreading* problem. According our observation, the *error spreading* only occurs in some bits but not all bits. Thus, we use the unequal error protection (UEP) codes strategy to give the different protection ability for these LSBs individually. Finally, we can save the parity bits when compared with the trivial method using EC codes.

The rest of this paper is organized as follows. In Section 2 the *Coded LSB* scheme is introduced. In Section 3, a trivial method based on EC codes is given. Our scheme based on UEP code is proposed in Section 4. The comparison and experimental results are shown in Section 5. Conclusion is drawn in Section 6.

2. Coded LSB Steganography Scheme

In the plain *LSB-Steganography* approach, the secret bits are embedded by simply replacing the LSBs of pixels. It means that when embedding n bits, we have 1/2 probability to alter LSBs of pixels, i.e., $n/2$ LSBs will be changed in average. For the *Cded LSB* scheme [1], the hidden secret is divided into many chips of l bits and each chip is embedded into n pixels to reduce the number of altered LSBs using the (n, k) Hamming codes where k and n are code lengths for information and codeword and $l=n-k$.

Coded LSB Scheme:
The construction method for the *Coded LSB* scheme is based on the cyclic coding. Let $G_1(x) = \sum_{i=0}^{k} g_{1,i} \cdot x^i$ and $G_2(x) = \sum_{i=0}^{l} g_{2,i} \cdot x^i$ where $g_{1,i}$ and $g_{2,i} \in \{0, 1\}$ be two binary polynomials with degree k and l, respectively. Their product is $G_1(x) \cdot G_2(x) = x^n + 1$. Using (n, k) Hamming codes with the generating function $G(x)=G_2(x)$, we can construct 2^l cosets including 2^k codewords in each coset and every coset does not contain the identical codewords. Since there are 2^l cosets, we may let each coset represent the l-bit secret chip. When encoding the LSBs to embed secret message, we use the nearest codeword in the coset to represent the embedded secret. The formal embedding process for the *Coded LSB* scheme is described in the following algorithm.

Algorithm 1: *[Process every n LSBs of the pixels in the cover image]*

Input: l embedded secret bits $(s_{l-1},\cdots, s_0)$, *n LSBs* $(c_{n-1},\cdots, c_0)$ *of the pixels in the cover image O.*

Output: n LSBs $(c'_{n-1},\cdots, c'_0)$ *of the pixels in the Stego-image* O'.

Step 1: Choose one cyclic (n, k) code with generating function $G(x) = g_l \cdot x^l + \cdots + g_1 \cdot x + g_0$ *and then select any k-tuples as the input to construct* 2^k *codewords. This code set including the* 2^k *codewords is called a coset. Choose one n-tuple codeword that does no appear in this coset, and then add an unused n-tuple to all the codewords in the coset to form another coset.*

Step 2: Repeat Step 1, until all 2^n *codewords are used. Finally we have* 2^l *different cosets* $L_0, \cdots, L_{2^l-1}$ *that include* 2^k *codewords in each coset.*

Step 3: Encrypt the secret $(s_{l-1},\cdots, s_0)$ *by choosing the coset* L_i *where* $i = \sum_{i=0}^{l-1} s_i \cdot 2^i$. *Then, find the codeword* $(c'_{n-1},\cdots, c'_0)$ *in coset* L_i *such that the Hamming distance between* $(c'_{n-1},\cdots, c'_0)$ *and* $(c_{n-1},\cdots, c_0)$ *is as minimal as possible.*

Step 4: Deliver the n LSBs $(c'_{n-1},\cdots, c'_0)$ *to the corresponding pixels in the embedded Stego-image* O'.

Here, we define two parametrs to show the efficiency of steganographic sschemes. One is the embedding rate, *ER*, which is the number of embedded bits per pixel, i.e. $ER = \frac{l}{n}$; the other is the alteration rate, *AR*, which is the number of embedded bits per altered bit, i.e., $AR = \frac{l_{alt}}{n}$ where l_{alt} is the average alteration LSBs when *l* secret bits are embedded into the *n* LSBs. The value l_{alt} can be calculated from all codewords in coset by a computer program. The first parameter is for discussing the embedded capacity and the second parameter is to show the distortion of the cover image. For the plain *LSB-Steganography* scheme, it is obvious that $ER = \frac{l}{n} = \frac{n}{n} = 100\%$ and $AR = \frac{l_{alt}}{n} = \frac{n/2}{n} = 50\%$.

The following example shows the embedding operation of *Algorithm 1*.

Example 1. Construct a *Coded LSB* scheme with *l*=3 and *n*=7.

When $n = 7$, we have $x^7 + 1 = (x^4 + x^3 + x^2 + 1)\cdot(x^3 + x^2 + 1)$. Let $G(x) = x^3 + x^2 + 1$, and we have $k = 4$, and $l = 3$ i.e., we can embed three secret bits into seven LSBs. After the first two steps in *Algorithm 1*, we construct eight cosets with sixteen codewords in each coset based on (7, 4) Hamming code. If the secret is (101) and the 7-tuple LSBs is (0001110), we use the coset L_5 (since the binary representation of 5 is (101)) to find the codeword that has the minimum Hamming distance 1 from (0001110). We only alter one LSB to embed three secret bits. The embedding rate and alteration are *ER*=3/7=42.9% and *AR*=0.875/7=12.5%. (Note that l_{alt} is 0.875 for this example.) Compared with the plain *LSB-Steganography* scheme (*ER*=100%, *AR*=50%), the plain *LSB-Steganography* scheme has more embedded capacity but the *Coded LSB* scheme modifies fewer bits, i.e. reduces the distortion of the cover images.

Error Spreading Problem in Coded LSB Scheme:
One drawback of the *Coded LSB* scheme is that when one bit error occurs in the coded *n* LSBs, it may cause more than one bit errors of secret bits when extraction. Considering *Example 1*, we can use the 7-tuple LSBs (0000000) to carry the secret (000). Suppose that there is one bit error, for example (0010000), we recover the secret as (111) and there are three error bits. However, If the error pattern is (0000001), the recovered secret is (001) and no errors spread. From our observation, if the error falls in the right three positions, i.e. (0000100), (0000010) and (0000001), there will be only one error in the recovered secret and the damage is not expanded. For the following error patterns (1000000), (0100000), (0010000) and (0001000), the decoded secret is (110), (011), (111) and (101), respectively, and there are more than one bit errors in the decoded secret. This is called *error spreading* problem and will make the revealed secret image worse.

There is one trivial method to solve the *error spreading* problem in *Coded LSB* scheme. We can add the error correcting capability into this scheme using EC codes. The following section shows how to add the error correcting capability.

3. Add Error Correcting Capability into Coded LSB Scheme Using EC Codes

We can directly add error correcting capability to ensure the correctness of the *n*-tuple LSBs to prevent the *error spreading* in the secret. Let $x^n + 1 = G_1(x)\cdot G_2(x)$ where the degree of $G_1(x)$

and $G_2(x)$ are k, and l. It is obvious that we can construct a *Coded LSB* scheme with $ER = \frac{l}{n}$ using the generating function $G(x)=G_2(x)$. After that we encode 2^k k-tuple vectors in each coset into (N_E, n) EC codes. The embedding rate for this new *Coded LSB* scheme is reduced to $ER = \frac{l}{N_E}$, but now we have the error correcting capability of (N_E, n) EC codes. The following example shows the efficiencies *ER* and *AR* for this EC-based scheme.

Example 2. Construct a EC-based *Coded LSB* scheme with l=3, n=7 and N_E=11 and one-error correcting capability.
Considering *Example 1*, first use (7, 4) Hamming code to embed three secret bits and then use (11, 7) shorten Hamming code to achieve one error correcting capability. To show the error correcting capability, assume that we embed the secret (000) into 7-tuple (0001101) in coset L_0 using (7, 4) hamming code. Then append the parity (1001) to form a shorten (11, 7) Hamming code (10010001101). Now, for example, if one error occurs in the sixth position, i.e. (10010101101), we first decode codeword to (0001101) using (11, 7) code and then find that the codeword (0001101) is in coset L_0. Finally, we get secret (0000). The error is always corrected no matter where the one bit error occurs due to the capability of (11, 7) shorten Hamming code. So, we can solve the *error spreading* problem. However, the cost is the reduction of the embedding rate and alteration rate. In this example, the *ER* and *AR* are reduced to 3/11=27.3% and 2.625/11=23.9%, where l_{alt} is 2.625.

As we mention in *error spreading* problem of Section 2. It is observed that only the error in the first k bits of n-tuple will result in more errors. So, to prevent the *error spreading*, we can just make sure the validity of the first k bits instead of all the n bits.

4. Add Error Correcting Capability into Coded LSB Scheme Using UEP Codes

A category of EC code called UEP code is to present different protecting capability for different bit locations [2], [3]. Some information bits are protected against a greater number of errors than other information bits. UEP codes are denoted as $[n, k, (d_1, d_2, \cdots, d_k)]$, where k means the length of the information vector and n is the code length. When decoding, if no more than $\lfloor (d_i - 1)/2 \rfloor$ errors occurr in the transmitted codeword, the correctness of the ith bit can be guaranteed.

Since the first k bits of vectors in the coset need more error protection, we can apply UEP codes to assure the correctness of these k bits and meantime the redundant parity bits are less than EC codes.

First, Let $x^n + 1 = G_1(x) \cdot G_2(x)$ where the degree of $G_1(x)$ and $G_2(x)$ are k, and l and take $G_2(x)$ as the generating polynomial to construct the 2^l cosets. Then encode the n-tuple vector by using the $[N_U, n, (d_1, d_2, \cdots, d_k)]$ UEP code. Since the value of N_U is less than N_E, so our proposed UEP-based *Coded LSB* scheme will have higher embedding rate and also prevent the *error spreading*.

Example 3. When N_U =10, n = 7, k = 4 and l = 3, we use [10, 7, (3333222)] UEP code to add one correcting capability into the *Coded LSB* scheme.
Eight cosets for this *Coded LSB* are shown in Table 1. (Note that the parity bits are added from right for easy description.) Suppose that the original codeword is “0000000000”, and consider the following cases that one error occurs:
Case 1: If the first bit (from left) is wrong, we receive the codeword “1000000000”. By Table 1, there is only one codeword “0000000000” in coset 0, which is closet to “1000000000” with Hamming distance 1.
Case 2: If the error falls in the 5th bit (from left) as “0000100000”, we can find that two codewords, “0000000000” in coset 0 and “0000100100” in coset 4, are close to “0000100000” with Hamming distance 1. It can be observed that the first 4 bits of “0000000000” and “0000100100” are the same, so after decoding, the first 4 bits “0000” will be still correct, while the last 3 bits may be “000” or “010”. Thus, even though we choose 010 in the other three bits, the recovered secret has only one error, i.e., no errors spread.
Case 3: If the error falls in the 10th bit as “0000000001”, it will be decoded as “0000000000” in coset 0 or “0000001001” in coset 1, the first 4 bits “0000” is also guaranteed.

Since $\lfloor (3-1)/2 \rfloor = 1$, when less than 1 error falls in the first 4 bits of the original 7-bit vector it can be corrected during decoding. If one error occurs in other places, we can guarantee that the first 4 bits are correct, but the last 3 bits may be not. When k = 4, and l = 3, the [10, 7, (3333222)] UEP code is suitable to provide more protection on the first 4 bit against the *error spreading*. For this UEP-based scheme $ER = \frac{l}{N_U}$ =30% and $AR = \frac{l_{alt}}{N_U} = 22.5\%$ where l_{alt} is 2.25 for this example.

From the above example, for [10, 7, (3333222)] UEP code, the first four bits have one-error correcting capability. When using EC code, we need to use (11, 7) shorten Hamming code to correct one error. Using the same representation like UEP codes, (11, 7) shorten Hamming code can be represented as [11, 7, (3333333)]. Therefore, if we just want to protect the first four bits, then we can save one redundant checking bits by using the UEP code. Our error-correcting capability of UEP codes is not better than EC codes, but it has better embedding and alteration rates and also overcomes the weakness of *error spreading*.

5. Comparison and Experimental Results

There are some powerful analyses for steganographic schemes, like the sample pair analysis [4], and image quality metrics [5]. To resist the different and new analysis, we have to construct the steganographic scheme that not only considering the higher embedding rate but also the relation between the cover-image and the stego-image, or the relations between pixels and pixels. In this section, we show the PSNR of the cover image and Stego-image for our proposed scheme. Table 2 shows the *Coded LSB* schemes constructed by EC codes and UEP codes to avoid the *error spreading* for n = 2 to 12. In Table 2, the (n, k, l) is original *Coded LSB* scheme, and N_E and N_U are the code lengths of EC and UEP codes. From Table 2, all the UEP-based schemes have shorter lengths than the EC codes and both of them have the ability to correct the first k bits when 1 error occurs, i.e. avoid the error-spreading problem.

To compare the distortion caused by these three *Coded LSB* schemes, conventional, EC-based and UEP-based schemes, we use “Baboon”, “Barb”, “Boat”, “Elain”, “Mena”, and “Peppers” as the test cover images and the test embedded image is a text picture (see Figure 1), the “NDHU” (National Dong Hwa University). All of the test cover images are 256×256 and due to the different embedding rate for different schemes, we use three sized “NDHU” text image 59×59, 47×47 and 49×49 for conventional, EC-based and UEP-based schemes, respectively. The PSNR between each cover image and its Stego-image is shown in Table 3. It is observed that the conventional scheme have higher PSNR, since the *AR*=12.5% is smaller than 23.9% (EC-based) and 22.5% (UEP-based) of EC-based and UEP-based schemes. Thus adding the error correcting capability does not distort the Stego-image seriously. It is reasonable and practical to use the error correcting codes into the steganographic scheme.

To further study the *error spreading* problem, we add the error patterns in two types: one is to add the errors randomly and the other is to add the errors in the worst places (the first k bits) that will cause *error spreading*. Figure 1 and Figure 2 show the extracting results for the random case and the worst case. Two embedded images are used: one is the “NDHU” text picture and the other is “NDHU” logo picture.

It is observed that the proposed scheme is almost the same to EC-based scheme and better than the conventional *Coded LSB* scheme in a noisy environment. Moreover, our UEP-based scheme has more embedding rate than the EC-based scheme.

6. Conclusion

In this paper we integrate UEP codes into the *Coded LSB* steganographic scheme such that the error-spreading problem can be solved. Though a trivial EC-based scheme can have the same effect, its embedding rate is lower than our UEP-based scheme and needs to use more pixels to embed the same secret.

REFERENCES

[1] M. Dijk and F. Willems, “Embedding information in grayscale images,” in *Proc. 22nd Symp. Inform. Theory in the Benelux*, The Netherlands, 2001, pp. 147-154.

[2] W. J. V. Gils, “Two topics on linear unequal error protection codes: bounds on their length and cyclic code classes,” *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 866-876, Nov. 1983.

[3] M. C. Lin, C. C. Lin and S. Lin, “Computer search for binary cyclic UEP codes of odd length up to 65,” *IEEE Trans. Inform. Theory*, vol. 36, No. 4, pp.924-935, July 1990.

[4] S. Dumitrescu, X. Wu and Z. Wang, “Detection of LSB steganography via sample pair analysis,” *IEEE Trans. Signal Processing*, vol. 51, NO. 7, July 2003.

[5] I. Avcibas, N. Memon and B. Sankur, “Steganalysis using image quality metrics,” *IEEE Trans. Image Processing*, vol. 12, NO. 2, February 2003.

Table 1. Eight cosets for the *Coded LSB* scheme with *l*=3, *N*=11 using [10, 7, (3333222)] UEP code

L_0	(0000000000)*(0001101110)(0011010100)(0010111010)(0110100100)(0111001010) (0101110000)(0100011110)(1101000000)(1100101110)(1110010100)(1111111010) (1011100100)(1010001010)(1000110000)(1001011110)
L_1	(0000001001)*(0001100111)(0011011101)(0010110011)(0110101101)(0111000011) (0101111001)(0100010111)(1101001001)(1100100111)(1110011101)(1111110011) (1011101101)(1010000011)(1000111001)(1001010111)
L_2	(0000010010)(0001111100)(0011000110)(0010101000)(0110110110)(0111011000) (0101100010)(0100001100)(1101010010)(1100111100)(1110000110)(1111101000) (1011110110)(1010011000)(1000100010)(1001001100)
L_3	(0000011011)(0001110101)(0011001111)(0010100001)(0110111111)(0111010001) (0101101011)(0100000101)(1101011011)(1100110101)(1110001111)(1111100001) (1011111111)(1010010001)(1000101011)(1001000101)
L_4	(0000100100)*(0001001010)(0011110000)(0010011110)(0110000000)(0111101110) (0101010100)(0100111010)(1101100100)(1100001010)(1110110000)(1111011110) (1011000000)(1010101110)(1000010100)(1001111010)
L_5	(0000101101)(0001000011)(0011111001)(0010010111)(0110001001)(0111100111) (0101011101)(0100110011)(1101101101)(1100000011)(1110111001)(1111010111) (1011001001)(1010100111)(1000011101)(1001110011)
L_6	(0000110110)(0001011000)(0011100010)(0010001100)(0110010010)(0111111100) (0101000110)(0100101000)(1101110110)(1100011000)(1110100010)(1111001100) (1011010010)(1010111100)(1000000110)(1001101000)
L_7	(0000111111)(0001010001)(0011101011)(0010000101)(0110011011)(0111110101) (0101001111)(0100100001)(1101111111)(1100010001)(1110101011)(1111000101) (1011011011)(1010110101)(1000001111)(1001100001)

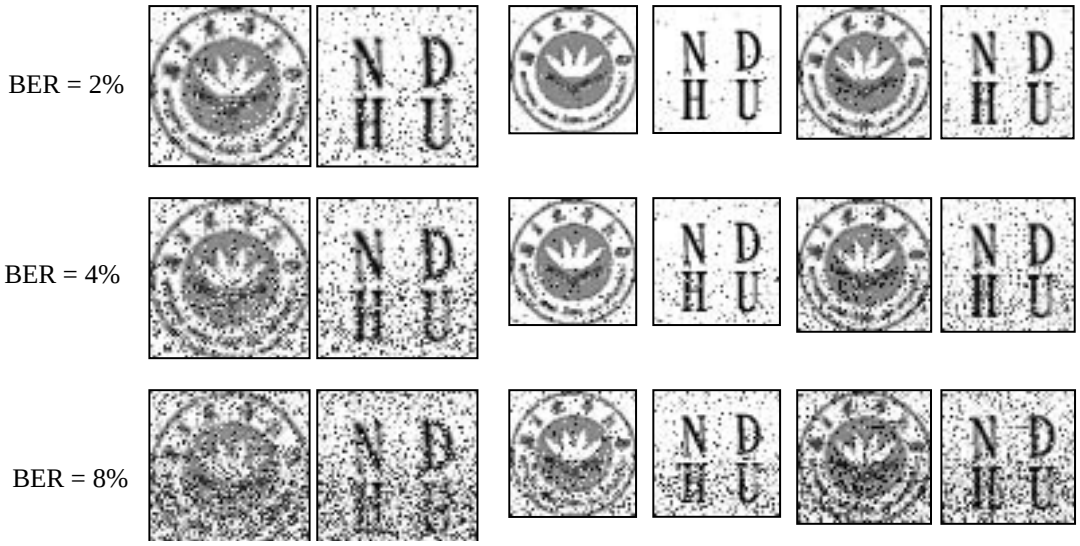
Table 2. Conventional, EC-based and UEP-based *Coded LSB* schemes for n = 2 to 12

Conventional *scheme*		EC-based scheme		UEP-based scheme	
(n, k, l)	$ER = \frac{l}{n}$	$[N_U, n, (d_1, d_2, \cdots, d_k)]$	$ER = \frac{l}{N_E}$	$[N_E, n, d]$	$ER = \frac{l}{N_U}$
(2, 1, 1)	50.0%	[5, 2, 3]	20.0%	[4, 2, (32)]	25.0%
(4, 1, 3)	75.0%	[7, 4, 3]	42.9%	[6, 4, (3222)]	50.0%
(5, 4, 1)	20.0%	[9, 5, 4]	11.1%	[8, 5, (33332)]	12.5%
(6, 4, 2)	33.3%	[10, 6, 4]	20.0%	[9, 6, (333322)]	22.2%
(7, 4, 3)	42.9%	[11, 7, 4]	27.3%	[10, 7, (3333222)]	30.0%
(8, 4, 4)	50%	[12, 8, 4]	33.3%	[11, 8, (33332222)]	36.4%
(9, 4, 5)	55.6%	[13, 9, 4]	38.5%	[12, 9, (333322222)]	41.7%
(10, 4, 6)	60%	[14, 10, 4]	42.9%	[13, 10, (3333222222)]	46.2%
(11, 4, 7)	63.6%	[15, 11, 4]	46.7%	[14, 11, (33332222222)]	50.0%
(12, 4, 8)	66.7%	[17, 12, 5]	47.1%	[15, 12, (333322222222)]	53.3%

Table 3. PSNR(dB) between the cover images and its Stego-images

Coded LSB scheme / Cover image	Conventional	EC-based	UEP-based
Baboon	57.173	54.671	54.691
Barb	57.148	54.687	54.675
Boat	57.181	54.696	54.677
Elain	57.143	54.675	54.681
Mena	57.151	54.683	54.710
Peppers	57.181	54.693	54.671

(a) Two Embedded images with three size: 59×59, 47×47 and 49×49



(b) Recovered image with different BER

Figure 1. Recovered images for different *Coded LSB* schemes with BER = 2%, 4% and 8% in random positions

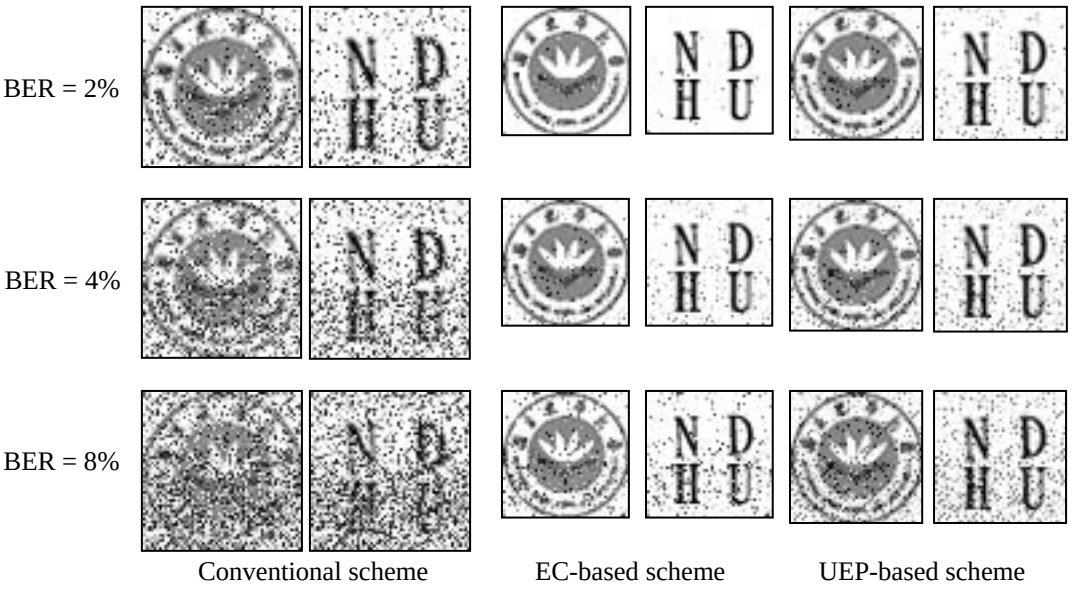


Figure 2. Recovered images for different *Coded LSB* schemes with BER = 2%, 4% and 8% in worst positions