An Improved Data Hiding Method Based on Lempel-Ziv-Welch Compression Codes

Chin-Chen Chang[1], Ngoc-Tu Huynh[2], Yu-Kai Wang[3], and Yanjun Liu[1]
(Corresponding author: Yanjun Liu)

Department of Information Engineering and Computer Science, Feng Chia University[1]
Taichung 40724, Taiwan
(Email: yjliu104@gmail.com)
College of Information Technology, The University of Danang, Vietnam[2]
Department of Computer Science and Information Engineering, National Chung Cheng University[3]
(Received Dec. 27, 2016; revised and accepted Apr. 11, 2017)

Abstract

Data hiding techniques have been widely used for the last decades to protect secret data by hiding the secret data into a cover file. These techniques are categories based on working domain such as spatial domain, compressed domain and transform domain. In this paper, we propose an improved data hiding method, which employs Lempel-Ziv-Welch (LZW) compression codes to embed secret information since it requires very low computational cost. Our method not only reduces the size of files stored on the disk but also prevents them from being attacked. Experimental results show that the proposed method outperforms some previous schemes in terms of compression rate as well as embedding capacity.

Keywords: Compression Rate; Data Hiding; Embedding Capacity; Lempel-Ziv-Welch (LZW) Compression

1 Introduction

With the development of information technology, there are more and more data kept and transferred. Thus, there are two issues that users always concern about: (1) space of storage and (2) security and confidentiality of their information. For the first issue, in order to save storages, users would prefer some compression methods with faster rates of compression and decompression to compact their data before storing them. Unfortunately, most of effective compression algorithms are computationally expensive. For the second issue there are many traditional block ciphers, such as Data Encryption Standard (DES) and Advanced Encryption Standard (AES) [6,7,14], were proposed to encrypt information but they are not suitable for image encryption because of the redundancy and special storage format of an image. Among various protection methods, steganography technique is one of the most efficient and common methods for image information protection. Steganography techniques, also called data hiding techniques, embed secret information into another host container to avoid suspicion from attackers during transferring information through the Internet or a public channel.

In order to tackle the high complexity of compression methods, a universal lossless compression algorithm called Lempel-Ziv-Welch (LZW) algorithm, was proposed by Abraham Lempel and Jacob Ziv in 1977 and widely used soon due to its achievement of a good compromise between compression performance and speed of execution. Then, LZW was improved by Terry Welch in 1984 [25,28,29]. It is also known as an adaptive compression algorithm that does not assume any prior knowledge of the symbol probabilities.

Data hiding techniques are classified into two categories: reversible [1, 2, 4, 5, 9–13, 15–17, 19, 21–24, 26, 27] and irreversible data hiding [3, 8, 18, 20]. Reversible data hiding schemes are gained more attention since they are suitable for protecting sensitive information such as private information or medical images. In 2013, Wang *et al.* [23] proposed a data hiding method based on LZW code which modifies values of LZW codes to hide secret data. However, in their scheme, the compression rate is still high and data embedding capacity is quite low. In this paper, we propose a new improvement on Wang *et al.*'s method which employed the LZW compression algorithm to solve above two issues at the same time. The proposed scheme achieves a higher protection of secret information while enhancing the compression rate.

The rest of this paper is organized as follows. In Section 2, we briefly review LZW compression method. Our proposed scheme is described in Section 3. In Section 4, we demonstrate the performance of our proposed scheme. Our conclusions are given in Section 5.

2 Related Work

2.1 The LZW Algorithm

The LZ family, including LZ1 [28], LZ2 [29], LZW [25] and their variants are the most popular dictionary-based compression algorithms because they achieve low complexity and good compression rate. LZW algorithm achieves the compression by replacing a repeated sequence of characters with a reference back to its previous occurrence. Performance of the algorithm depends on how such references are represented and on how to select the sequences that are replaced. We now briefly review the LZW algorithm.

The algorithm compresses different lengths of sub-strings to a same length of compression code. That is to say, the user can acquire a same length of index. The characteristic of this algorithm is simple and effective to be implemented. Creating a dictionary including all of the characters in the input file is the first step of this algorithm. Secondly, we find the sub-string which can be expressed in the dictionary. Then, the sub-string will be combined with the next character. After that, we update it in the dictionary and output the index of the sub-string. The file is read repeatedly and the dictionary is updated until all of the characters in the file are read. Finally, we output is the index numbers. The details of the LZW algorithm are described as follows.

Algorithm 1 Compression procedure of the original LZW algorithm

Input. Source file which is needed to be compressed

Output. LZW codes

Step 1. Create a dictionary including all of the characters.

Step 2. Scan a sub-string w from the input file which can be found in the dictionary.

Step 3. Combine the sub-string w with the next character c, and give new index $I(w||c)$ and add it to the dictionary.

Step 4. Output the index of the sub-string $I(w)$ and remove the sub-string w in the input file.

Step 5. Repeat Step 2 through Step 4 until the source file is compressed.

After compressing, we have the LZW compression code with shorter size. Whenever the user wants to decompress to recover the original data, he/she will perform the decompression procedure which is described below.

Algorithm 2 Decompression procedure of the original LZW algorithm

Input. Compression file

Output. Source file which was decompressed

Step 1. Create a dictionary including all of the character.

Step 2. Scan a LZW code $I(w)$ from the compression file and extract the corresponding symbol w.

Step 3. If the next index is equal to the dictionary plus one, combine the symbol w with the first character c of itself; otherwise, combine the symbol w with the first character c of the next LZW code, then give new index $I(w||c)$ and add it to the dictionary.

Step 4. Output the index of the symbol w and remove the LZW code $I(w)$ in the compression file.

Step 5. If all of LZW codes from the compression file are read, then the process is ended and output the decompressed data; otherwise, repeat Step 2 through Step 4.

The following example gives a clearer understanding of LZW algorithm. Assume that we have a dictionary which is represented by 8 bits in ASCII and the length of the compression code is 9 bits. Table 1 shows the LZW algorithm for the input string "aabaababbaabbabaaabb".

Table 1: An example to illustrate LZW algorithm

Compression			Decompression		
Input	Updated dictionary	Output	Input	Output	Updated dictionary
aa	aa=256	97	97	a	
ab	ab=257	97	97	a	aa=256
ba	ba=258	98	98	b	ab=257
aab	aab=259	256	256	aa	ba=258
bab	bab=260	258	258	ba	aab=259
bb	bb=261	98	98	b	bab=260
baa	baa=262	258	258	ba	bb=261
abb	abb=263	257	257	ab	baa=262
baba	baba=264	260	260	bab	abb=263
aaa	aaa=265	256	256	aa	baba=264
abb		263	263	abb	aaa=265

2.2 The HPDH-LZW Scheme

In 2013, Wang *et al.* [23] proposed a high performance reversible data hiding scheme based on LZW algorithm (HPDH-LZW). The main idea of the paper is that the compression code is divided into two ranges: in the dictionary or not. In the hiding phase, if the secret bit is 0, output the original compression index; if the secret bit is 1, calculate the original compression index, and then add the number of dictionary size and output it. In the extraction phase, if the value of current processing code is larger than current size of dictionary, we extract secret bit "1" and recover the original index by calculating the difference between this code and the dictionary size. Otherwise, we extract bit "0" and the original LZW code is equal to the current code. The details of the HPDH-LZW algorithm are described as follows.

The details of the HPDH-LZW decompression and secret extraction procedure are described below.

In the following example, we assume that a dictionary is represented by 8 bits in ASCII, the length of the com-

Algorithm 3 The HPDH-LZW algorithm

Input. Source file which is needed to be compressed and secret data

Output. LZW codes

Step 1. Create a dictionary which includes all characters.

Step 2. Scan a sub-string w from the input file, which can be found in the dictionary.

Step 3. Combine the sub-string w with the next character c from the source file, and add new index $I(w||c)$ to the dictionary.

Step 4.1. If secret bit is "0", then output the index of the sub-string $I(w)$. If the secret bit is "1", then output: $I(w) + ds$.

Step 4.2. Remove the sub-string w in the input file.

Step 5. Repeat Step 2 through Step 4 until the source file is compressed.

Algorithm 4 The HPDH-LZW decompression algorithm

Input. Compressed file

Output. The decompressed source file and secret file

Step 1. Create a dictionary size ds which includes all of the characters.

Step 2. Scan a LZW code $I(w)$ from the compression file. If $I(w)$ is more than the dictionary size ds, we extract the secret bit as "1" and recover the original LZW code $I(w) - ds$; otherwise, we extract the secret bit as "0" and the original LZW code is equal to $I(w)$. The corresponding symbol w is extracted according to the original LZW code.

Step 3. If the next LZW code is equal to $ds + 1$, we combine the symbol w with its first character c; otherwise, combine the symbol w with the first character c of the next LZW code, then give new index $I(w||c)$ and add it to the dictionary.

Step 4. Output the index of the symbol w and the secret bit, then remove the LZW code $I(w)$ in the compressed file.

Step 5. If all LZW codes from the compression file are read, output the decompression data and secret file; otherwise, repeat Step 2 to Step 4.

pression code is 10 bits and the largest size of the dictionary is 9 bits. It means that, the original dictionary indices are from 0 to 255, the new dictionary indices are from 256 to 511 and the highest index is 1023. Table 2 illustrates the HPDH-LZW algorithm for the input string "aabaababbaabbabaaabb" and the string of secret bits "1101101001".

It is noted that in the HPDH-LZW scheme, the size of the compression code and the capacity of secret bits are in inverse proportion, but the size of the compression code and the size of the compression file are in direct proportion.

3 Proposed Scheme

Our proposed scheme is called IDH-LZW, i.e., improved data hiding method based on Lempel-Ziv-Welch compression. In our proposed scheme, the values of LZW codes are modified to embed secret bits of different lengths while avoiding changing the content of the dictionary. That is to say, every new symbol inserted into the dictionary is used to embed secret bits of different lengths.

3.1 IDH-LZW Embedding Algorithm

The proposed embedding algorithm is described in Algorithm 5 as follows.

Algorithm 5 IDH-LZW embedding algorithm

Input. Source file and secret file

Output. Compression codes with secret

Firstly, the user must define how many bits to represent one compression code C_size and how many indices in the dictionary Ds.

Step 1.

- Read the character c_i from the source file.
- Set $s = sp||c_i$, where s is a string variable, sp is previous symbol and $||$ means the concatenation operation. If $i = 0$ then $s = c_0$.

Step 2.

If s exists in the dictionary

- Set the previous symbol $sp = s$.

Else

- Compute $kp = Ds/ds$, where ds is the current size of the dictionary, kp is the hidden fragment and the value of kp is between 2^k and 2^{k+1}.
  - Get k secret bits b_k from the secret file and sb_k is b_k in decimal.
  - Get the code C_i, where C_i is the dictionary index of sp.
  - Set $C_i^{'} = C_i + ds * sb_k$, where ds is the current size of the dictionary.
  - Output $C_i^{'}$, where $C_i^{'}$ is the compression code with secret.

Step 3.

If $ds < Ds$

- Add s into the dictionary.
- Set $s = c_i$, where c_i is the last character of s.

If the source file is remained

- Repeat Step 1 to Step 3.

Step 4.

In case sp still has data without output after Step 3

- Set $kp = Ds/ds$.
- Take a secret digit sb_k from the secret file.
- Get code $C_i^{'}$ and set $C_i^{'} = C_i + ds * sb_k$.
- Output $C_i^{'}$.

Step 5.

- Transform $C_i^{'}$ to binary code Cb_i where binary code size is C_size.
- The final compression code is C, where C is the concatenation of Cb_i.

Table 2: An example to illustrate HPDH-LZW algorithm

Compression				Decompression			
Input	Updated dictionary	Hidden bit	Output	Input	Extracted bit	Output	Updated dictionary
aa	aa=256	1	353	353	1	a	
ab	ab=257	1	354	354	1	a	aa=256
ba	ba=258	0	98	98	0	b	ab=257
aab	aab=259	1	515	515	1	aa	ba=258
bab	bab=260	1	518	518	1	ba	aab=259
bb	bb=261	0	98	98	0	b	bab=260
baa	baa=262	1	520	520	1	ba	bb=261
abb	abb=263	0	257	257	0	ab	baa=262
baba	baba=264	0	260	260	0	bab	abb=263
aaa	aaa=265	0	256	256	0	aa	baba=264
abb		1	529	529	1	abb	aaa=265

3.2 IDH-LZW Decompression and Secret Extraction Algorithm

In the decompression phase, it is noted that the number of bits used to represent one compression code C_size and the number of indices in the dictionary Ds are shared between the dealer and the receiver. This information is used to extract the secret bits and decompress the compressed file. Algorithm 6 is used to demonstrate the decompression and extraction procedure of our method.

Algorithm 6 IDH-LZW decompression and secret extraction algorithm

Input. Compression codes

Output. Reconstructed source file and recovered secret file

Step 1.

- Cut C_size bits to present one LZW compression code $C_i^{'}$.

Step 2.

- Read a LZW code $C_0^{'}$.
- Compute $kp = Ds/ds$, where kp is the embedding range, Ds is the number of the dictionary and ds is the current size of the dictionary. The value of kp is from 2^k to 2^{k+1}.
- Compute $C_0 = C_0^{'}/ds$ and get the remainder $sb_k = C_0^{'} \bmod d$, where C_0 is the original compression code, sb_k are extracted secret bits in decimal and b_k are k secret bits of sb_k in binary.
- Get w where w is the symbol of C_0 in the dictionary.

Step 3.

- Read the next LZW code $C_i^{'}$.

If ds is equal to Ds

$kp = Ds/ds$.

Else

$kp = Ds/(ds + 1)$.

- Compute $C_i = C_i^{'}/ds$, get the remainder $sb_k = C_i \bmod d$ and extract secret bit b_k from sb_k.
- Output b_k.

Step 4.

- Set c_i to be the source file value.

If C_i exists in the dictionary

- Get c_i, where c_i is the symbol of C_i in the dictionary.

Else if C_i is equal to ds

- Set $c_i = w||$(the first symbol of w).

If $(ds < Ds)$

- Put $c_i = w||$(the first symbol of w) into the dictionary.
- Set $w = c_i$.
- Output c_i.

Step 5.

- Concatenate c_i to reconstruct the source file and b_k to recover the secret file.

We illustrate all the steps of our IDH-LZW algorithm in Table 3 as follows. Here, we give a dictionary which is represented by 8 bits in ASCII, the length of compression code is 10 bits and the maximum size of the dictionary is 9 bits as an example. It means that, original dictionary indices are from 0 to 255, new dictionary indices are from 256 to 511 and the maximum index is 1023. Table 3 shows the proposed IDH-LZW algorithm for the input string "aabaababbaabbabaaabb" and the string of secret bits "110110100011".

4 Experimental Results

To show that our proposed scheme is suitable for most data formats, we conduct experiments on different kinds of files such as text files, binary images, grayscale images and color images with different file sizes. Figure 1 shows the set of test images and Figure 2 shows the text files used to implement our method.

4.1 Compression Performance

Our IDH-LZW scheme can embed a large amount of secret bits while compressing the source file. As a result, we can save storage as well as protect our data. We conduct our experiments on different kinds of file formats such as text files, binary images, grayscale images sized

Table 3: Example to illustrate IDH-LZW algorithm

Compression				Decompression			
Input	Updated dictionary	Hidden bit	Output	Input	Extracted bit	Output	Updated dictionary
aa	aa=256	11	865	865	11	a	
ab	ab=257	0	97	97	0	a	aa=256
ba	ba=258	1	356	356	1	b	ab=257
aab	aab=259	1	515	515	1	aa	ba=258
bab	bab=260	0	258	258	0	ba	aab=259
bb	bb=261	1	359	359	1	b	bab=260
baa	baa=262	0	258	258	0	ba	bb=261
abb	abb=263	0	257	257	0	ab	baa=262
baba	baba=264	0	260	260	0	bab	abb=263
aaa	aaa=265	1	521	521	1	aa	baba=264
abb		1	529	529	1	abb	aaa=265

(a) Baboon (grayscale) (b) Lena (grayscale) (c) Boat (grayscale) (d) Peppers (grayscale)

(a) Baboon (color) (b) Lena (color) (c) Boat (color) (d) Peppers (color)

Figure 1: Test images

Four score and seven years ago our fathers brought forth, on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great

(a) The Gettysburg address

I am happy to join with you today in what will go down in history as the greatest demonstration for freedom in the history of our nation.Five score years ago, a great American, in whose symbolic shadow we stand today, signed the Emancipation Proclamation. This momentous decree came as a great beacon light of hope to

(b) I have a dream

My fellow citizens: I stand here today humbled by the task before us, grateful for the trust you've bestowed, mindful of the sacrifices borne by our ancestors. I thank President Bush for his service to our nation as well as the generosity and cooperation he has shown throughout this transition. Forty-four Americans have now taken the

(c) Obama

THE LITTLE PRINCE Antoine De Saint-Exupery Antoine de Saint-Exupery, who was a French author, journalist and pilot wrote The Little Prince in 1943, one year before his death. The Little Prince appears to be a simple children's tale, some would say that it is actually a profound and deeply moving tale, written in riddles and laced

(d) The little prince

BRAVE NEW WORLD by Aldous Huxley (1894-1963) Chapter One A SQUAT grey building of only thirty-four stories. Over the main entrance the words, CENTRAL LONDON HATCHERY AND CONDITIONING CENTRE, and, in a shield, the World State's motto, COMMUNITY, IDENTITY, STABILITY. The enormous room

(e) Brave new world

Figure 2: Text files

256×256, grayscale images sized 512×512 and color images, as shown in Tables 4-8, respectively. In these tables, the "LZW size" parameter is the size of LZW codes, measured by the number of indices. The "capacity" values are taken under different dictionary sizes (ds) on various images.

4.2 Embedding Performance

The proposed IDH-LZW scheme aims to embed secret information into compressed files. Therefore, the embedding rate is concerned to evaluate the performance of an algorithm. To further demonstrate that our scheme achieves high embedding rate, Tables 9-11 show the number of hidden bits per LZW code measured by byte on different file formats. It can be seen from the tables that, a larger size of dictionary is increased and a larger amount of secret bits is embedded. However, the compression rate is higher. Moreover, among these tables, we can see that the embedding rate in Table 10 is the highest while the compression rate is low. The reason is that in grayscale images, many pixels are the same as their neighbors.

4.3 Comparisons

In this subsection, we implement Wang *et al.*'s scheme (HPDH-LZW) and compare the results with our proposed scheme in terms of low compression rate and high embedding capacity. The "decreased size" parameter is used to evaluate the size of compression code that is reduced while comparing to Wang *et al.*'s scheme. The "increased bits" parameter is conducted to show that our scheme can embed more data into host compression codes. Tables 12-16 obviously show that our proposed scheme is better than HPDH-LZW scheme in terms of compression rate and embedding capacity. Furthermore, graphs in Figure 3 visibly show our comparisons.

5 Conclusions

In this paper, we proposed a novel compression-based data hiding scheme called IDH-LZW which not only solves

Table 4: The capacity performance of IDH-LZW scheme for text files

C_size	10		11		12		13	
Ds	9		10		11		12	
File name	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity
The Gettysburg address	803	804	717	975	717	1692	717	2409
I have a dream	5090	5091	4040	4298	3538	4565	3325	6145
Obama	7566	7567	5819	6077	5058	6085	4680	7500
Brave new world	237230	237231	179303	179561	146171	147198	126063	128883
The little prince	54821	54822	40392	40650	33474	34501	28055	30875

Table 5: The capacity performance of IDH-LZW scheme for binary images

C_size	10		11		12		13		14		15	
Ds	9		10		11		12		13		14	
File name	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity
Airplane	7610	8112	6717	7730	5885	7921	5501	9584	5467	13645	5467	19112
Boat	9200	9702	8028	9041	7156	9192	6671	10754	6543	14721	6543	21264
Gold	12408	12910	10757	11770	9633	11669	8952	13035	8665	16843	8640	25009
Lena	8025	8527	6807	7820	6220	8256	5911	9994	5811	13989	5811	19800
Peppers	7190	7692	6147	7160	5775	7811	5533	9616	5491	13669	5491	19160

Table 6: The capacity performance of IDH-LZW scheme for grayscale images (sized 256×256)

C_size	16		17		18		19		20	
Ds	15		16		17		18		19	
File name	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity
Baboon	40104	70831	39977	103217	39977	143194	39977	183171	39977	223148
Barbara	40497	71224	40257	103497	40257	143754	40257	184011	40257	224268
Boat	34849	65576	34818	98058	34818	132876	34818	167694	34818	202512
Family	38009	68736	37895	101135	37895	139030	37895	176925	37895	214820
Girl	33797	64524	33789	97029	33789	130818	33789	164607	33789	198396
Lena	34861	65588	34843	98083	34843	132926	34843	167769	34843	202612
Peppers	351997	65926	35149	98389	35149	133538	35149	168687	35149	203836
Toys	32193	62920	32193	95113	32193	127306	32193	159499	32193	191692

Table 7: The capacity performance of IDH-LZW scheme for grayscale images (sized 512×512)

C_size	16		17		18		19		20	
Ds	15		16		17		18		19	
File name	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity
Baboon	146755	177482	136511	199751	133149	261670	132994	392332	132994	525326
Boat	146376	177103	124444	187684	109856	238377	109856	348233	109856	458089
F16	114876	145603	96009	159249	94785	223306	94785	318091	94785	412876
Lena	127139	157866	117295	180535	114434	242955	114434	357389	114434	471823
Peppers	129000	159727	118875	182115	116070	244591	116070	360661	116070	476731
Gold	140641	171368	122040	185280	118513	247034	118513	365547	118513	484060

Table 8: The capacity performance of IDH-LZW scheme for color images (sized 512×512)

C_size	16		17		18		19		20	
Ds	15		16		17		18		19	
File name	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity	LZW size (index)	Capacity
Baboon	509088	539815	470992	534232	428694	557215	407172	666510	402207	923434
Boat	559908	590635	510732	573972	426127	554648	344134	603472	337656	858883
F16	430883	461610	399157	462397	320466	448987	288861	548199	288400	809627
Lena	190343	221070	153265	216505	135890	264411	135842	395180	135842	531022
Peppers	516584	547311	490857	524097	415514	544035	387969	647307	382513	903740
Gold	588303	619030	469542	532782	384609	513130	361901	621239	356391	877618

Table 9: The embedding rate of the proposed scheme for text files

C_size	10		11		12		13	
Ds	9		10		11		12	
File name	LZW size (byte)	Hidden bit per byte	LZW size (byte)	Hidden bit per byte	LZW size (byte)	Hidden bit per byte	LZW size (byte)	Hidden bit per byte
The Gettysburg address	1003.75	0.80	985.88	0.99	1075.50	1.57	1165.13	2.07
I have a dream	6362.50	0.80	5555.00	0.77	5307.00	0.86	5403.13	1.14
Obama	9457.50	0.80	8001.13	0.76	7587.00	0.80	7605.00	0.99
Brave new world	296537.50	0.80	246541.63	0.73	219256.50	0.67	204852.38	0.63
The little prince	68526.25	0.80	55539.00	0.73	50211.00	0.69	45589.38	0.68

Table 10: The embedding rate of the proposed scheme for grayscale images (sized 256×256)

C_size	16		17		18		19		20	
Ds	15		16		17		18		19	
File name	LZW size (byte)	Hidden bit per byte	LZW size (byte)	Hidden bit per byte	LZW size (byte)	Hidden bit per byte	LZW size (byte)	Hidden bit per byte	LZW size (byte)	Hidden bit per byte
Baboon	80208.00	0.88	84951.13	1.22	89948.25	1.59	94945.38	1.93	99942.50	2.23
Barbara	80994.00	0.88	85546.13	1.21	90578.25	1.59	95610.38	1.92	100642.50	2.23
Boat	69698.00	0.94	73988.25	1.33	78340.50	1.70	82692.75	2.03	87045.00	2.33
Family	76018.00	0.90	80526.88	1.26	85263.75	1.63	90000.63	1.97	94737.50	2.27
Girl	67594.00	0.95	71801.63	1.35	76025.25	1.72	80248.88	2.05	84472.50	2.35
Lena1	69722.00	0.94	74041.38	1.32	78396.75	1.70	82752.13	2.03	87107.50	2.33
Peppers	70398.00	0.94	74691.63	1.32	79085.25	1.69	83478.88	2.02	87872.50	2.32
Toys	64386.00	0.98	68410.13	1.39	72434.25	1.76	76458.38	2.09	80482.50	2.38

Table 11: The embedding rate of the proposed scheme for color images (sized 512×512)

C_size	16		17		18		19		20	
Ds	15		16		17		18		19	
File name	LZW size (byte)	Hidden bit per byte	LZW size (byte)	Hidden bit per byte	LZW size (byte)	Hidden bit per byte	LZW size (byte)	Hidden bit per byte	LZW size (byte)	Hidden bit per byte
F16	861766	0.54	848209	0.55	721049	0.62	686045	0.80	721000	1.12
Baboon	1018176	0.53	1000858	0.53	964562	0.58	967034	0.69	1005518	0.92
Boat	1119816	0.53	1085306	0.53	958786	0.58	817318	0.74	844140	1.02
Gold	1176606	0.53	997777	0.53	865370	0.59	859515	0.72	890978	0.99
Lena	380686	0.58	325688	0.66	305753	0.86	322625	1.22	339605	1.56
Peppers	1033168	0.53	1043071	0.50	934907	0.58	921426	0.70	956283	0.95

Table 12: Comparisons between IDH-LZW scheme and Wang *et al.*'s scheme conducted on binary images (sized 512×512)

File name	Original file size	HPDH-LZW		Proposed scheme		Decreased size	Increased bits
		LZW size (C_size = 10 and $Ds = 9$)	Capacity	LZW size (C_size = 13 and $Ds = 12$)	Capacity		
F16	32768	9513	7610	8939	9584	6.03%	25.94%
Boat	32768	11500	9200	10840	10754	5.74%	16.89%
Goldhill	32768	15510	12408	14547	13035	6.21%	5.05%
Lena	32768	10031	8025	9605	9994	4.25%	24.54%
Peppers	32768	8988	7190	8991	9616	-0.04%	33.74%

Table 13: Comparisons between IDH-LZW scheme and Wang *et al.*'s scheme conducted on grayscale images (sized 256×256)

File name	Original file size	HPDH-LZW		Proposed scheme		Decreased size	Increased bits
		LZW size (C_size = 10 and $Ds = 9$)	Capacity	LZW size (C_size = 13 and $Ds = 12$)	Capacity		
Baboon	65536	80886	64709	80208	70831	0.84%	9.46%
Barbara	65536	79999	63999	80994	71224	-1.24%	11.29%
Boat	65536	73238	58590	69698	65576	4.83%	11.92%
Family	65536	79809	63847	76018	68736	4.75%	7.66%
Girl	65536	77253	61802	67594	64524	12.50%	4.40%
Lena	65536	76428	61142	69722	65588	8.77%	7.27%
Peppers	65536	77546	62037	70398	65926	9.22%	6.27%
Toys	65536	80878	64702	64386	62920	20.39%	-2.75%

Table 14: Comparisons between IDH-LZW scheme and Wang *et al.*'s scheme conducted on grayscale images (sized 512×512)

File name	Original file size	HPDH-LZW		Proposed scheme		Decreased size	Increased bits
		LZW size (C_size = 10 and $Ds = 9$)	Capacity	LZW size (C_size = 13 and $Ds = 12$)	Capacity		
Airplane	262144	315065	252052	225114	318091	28.55%	26.20%
Baboon	262144	321820	257456	302155	382967	6.11%	748.75%
Barbara	262144	320539	256431	300846	381865	6.14%	48.92%
Boat	262144	279783	223826	260908	348233	6.75%	55.58%
Elaine	262144	313738	250990	284174	367825	9.42%	46.55%
Family	262144	316699	253359	271211	356909	14.36%	40.87%
Girl	262144	299851	239881	257669	345505	14.07%	44.03%
Gold	262144	301103	240882	265326	351953	11.88%	46.11%
Lena	262144	300391	240313	254852	343133	15.16%	42.79%
Peppers	262144	305998.8	244799	252750	341363	17.40%	39.45%
Sailboat	262144	316698.8	253359	274785	359919	13.23%	42.06%
Bridge	262144	277127.5	221702	200623	297467	27.61%	34.17%
Tiffany	262144	286201.3	228961	232441	324261	18.78%	41.62%
Toys	262144	306772.5	245418	243457	333537	20.64%	35.91%
Zelda	262144	301776.3	241421	248214	337543	17.75%	39.82%

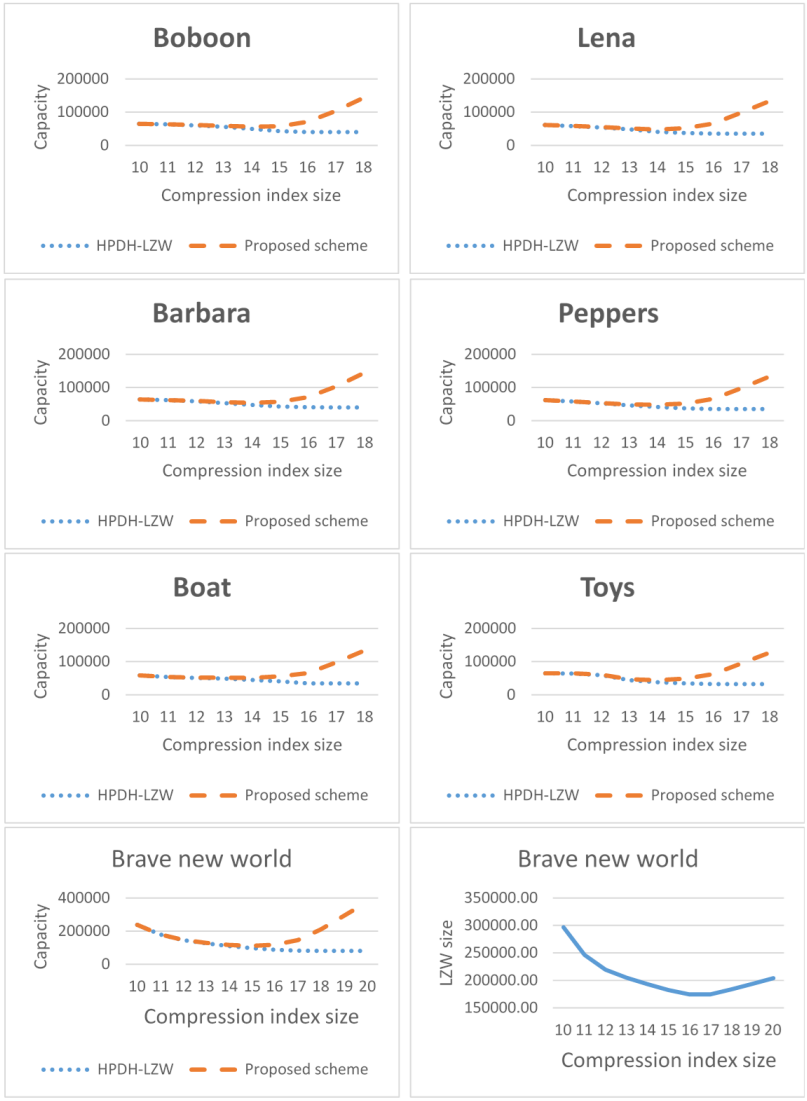

Figure 3: Comparisons between HPDH-LZW and our method

Table 15: Comparisons between IDH-LZW scheme and Wang *et al.*'s scheme conducted on color images (sized 512×512)

File name	Original file size	HPDH-LZW		Proposed scheme		Decreased size	Increased bits
		LZW size (R+G+B) ($C\_size = 10$ and $Ds = 9$)	Capacity	LZW size (R+G+B) ($C\_size = 13$ and $Ds = 12$)	Capacity		
Airplane	786432	894215	715372	711305	984557	20.45%	37.63%
Baboon	786432	971015	776812	970988	1186851	0.00%	52.78%
Boat	786432	912838	730270	775112	1038289	15.09%	42.18%
Gold	786432	948944	759155	835991	1089555	11.90%	43.52%
Lena	786432	693664	554931	464123	771629	33.09%	39.05%
Peppers	786432	934743	747794	829136	1083783	11.30%	44.93%

Table 16: Comparisons between IDH-LZW scheme and Wang *et al.*'s scheme conducted on text files

File name	Original file size	HPDH-LZW		Proposed scheme				Decreased size	Increased bits
		LZW size ($C\_size = 10$ and $Ds = 9$)	Capacity	C_size	Ds	LZW size	Capacity		
The Gettysburg Address	1461	1004	803	11	10	986	975	1.78%	21.42%
I have a dream	9167	6363	5090	14	13	5819	9470	8.55%	86.05%
Obama	13507	9458	7566	14	13	8160	11324	13.72%	49.67%
Brave new world	375467	296538	237230	19	18	193791	291713	34.65%	22.97%
The little prince	91141	68526	54821	17	16	46963	74927	31.47%	36.68%

the issue of data security but also reduces the size of storage. Experimental results show that our proposed scheme achieves good compression rate and high embedding capacity. Moreover, our proposed scheme has very low computation cost but guarantees the efficiency, that is suitable for real time applications.

References

[1] K. Bharanitharan, C. C. Chang, H. R. Yang, and Z. H. Wang, "Efficient pixel prediction algorithm for reversible data hiding," *International Journal of Network Security*, vol. 18, no. 4, pp. 750–757, 2016.

[2] D. Cavagnino, M. Lucenteforte, and M. Grangetto, "High capacity reversible data hiding and content protection for radiographic images," *Signal Processing*, vol. 117, pp. 258–269, 2015.

[3] H. Chen, X. Du, Z. Liu, and C. Yang, "Optical color image hiding scheme by using gerchberg-saxton algorithm in fractional fourier domain," *Optics and Lasers in Engineering*, vol. 66, pp. 144–151, 2015.

[4] D. Coltuc, "Low distortion transform for reversible watermarking," *IEEE Transactions on Image Processing*, vol. 21, no. 1, pp. 412–417, 2012.

[5] G. Galambos and J. Bekesi, "Data compression: theory and techniques. Department of informatics, teacher's training college," *Database and Data Communication Network Systems*, vol. 1, 2002.

[6] T. Gulom, "The encryption algorithms GOST28147-89-IDEA8-4 and GOST28147-89-RFWKIDEA8-4," *International Journal of Electronics and Information Engineering*, vol. 6, no. 2, pp. 59-71, 2017.

[7] T. Gulom, "The encryption algorithm GOST28147-89-PES16-2 and GOST28147-89-RFWKPES16-2," *International Journal of Electronics and Information Engineering*, vol. 6, no. 1, pp. 1–11, 2017.

[8] R. Jafari, D. Ziou, and M. M. Rashidi, "Increasing image compression rate using steganography," *Expert Systems with Applications*, vol. 40, no. 17, pp. 6918–6927, 2013.

[9] B. Jana, D. Giri and S. K. Mondal, "Dual-image based reversible data hiding scheme using pixel value difference expansion," *International Journal of Network Security*, vol. 18, no. 4, pp. 633–643, 2016.

[10] F. Li, Q. Mao, and C. C. Chang, "A reversible data hiding scheme based on IWT and the sudoku method," *International Journal of Network Security*, vol. 18, no. 3, pp. 410–419, 2016.

[11] J. J. Li, Y. H. Wu, C. F. Lee, C. C. Chang, "Generalized PVO-K embedding technique for reversible data hiding," *International Journal of Network Security*, vol. 20, no. 1, pp. 65-77, 2018.

[12] X. Li, B. Li, B. Yang, and T. Zeng, "General framework to histogram-shifting-based reversible data hiding," *IEEE Transactions on Image Processing*, vol. 22, no. 6, pp. 2181–2191, 2013.

[13] K. Ma, W. Zhang, X. Zhao, N. Yu, and F. Li, "Reversible data hiding in encrypted images by reserving room before encryption," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 3, pp. 553–562, 2013.

[14] S. D. Putra, S. Sutikno, Y. Kurniawan, A. S. Ahmad, "Design of an AES device as device under test in a DPA attack," *International Journal of Network Security*, vol. 20, no. 2, pp. 256-265, 2018.

[15] C. Qin, C. C. Chang, and T. J. Hsu, "Reversible data hiding scheme based on exploiting modification direction with two steganographic images," *Multimedia Tools and Applications*, vol. 74, no. 15, pp. 5861–5872, 2015.

[16] C. Qin and Y. C. Hu, "Reversible data hiding in vq index table with lossless coding and adaptive switching mechanism," *Signal Processing*, vol. 129, pp. 48–55, 2016.

[17] V. Sachnev, H. J. Kim, J. Nam, S. Suresh, and Y. Q. Shi, "Reversible watermarking algorithm using sorting and prediction," *IEEE Transactions Circuit Systems for Video Technology*, vol. 19, no. 7, pp. 989–999, 2009.

[18] E. Satir and H. Isik, "A compression-based text steganography method," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2385–2394, 2012.

[19] J. Wang, J. Ni, and Y. Hu, "An efficient reversible data hiding scheme using prediction and optimal side information selection," *Journal of Visual Communication and Image Representation*, vol. 25, no. 6, pp. 1425–1431, 2014.

[20] S. Wang, J. Sang, X. Song, and X. Niu, "Least significant qubit (lsqb) information hiding algorithm for

quantum image," *Measurement*, vol. 73, pp. 352–359, 2015.
[21] Y. L. Wang, J. J. Shen, M. S. Hwang, "A survey of reversible data hiding for VQ-compressed images," *International Journal of Network Security*, vol. 20, no. 1, pp. 1-8, 2018.
[22] Y. L. Wang, J. J. Shen, M. S. Hwang, "An improved dual image-based reversible hiding technique using LSB matching", *International Journal of Network Security*, vol. 19, no. 5, pp. 858–862, 2017.
[23] Z. H. Wang, H. R. Yang, T. F. Cheng, and C. C. Chang, "A high-performance reversible data-hiding scheme for lzw codes," *The Journal of Systems and Software*, vol. 86, no. 11, pp. 2771–2778, 2013.
[24] Z. H. Wang, X. Zhuang, C. C. Chang, C. Qin, Y. Zhu, "Reversible data hiding based on geometric structure of pixel groups", *International Journal of Network Security*, vol. 18, no. 1, pp. 52–59, 2016.
[25] T. A. Welch, "A technique for high-performance data compression," *IEEE Computer*, vol. 17, no. 6, pp. 8–19, 1984.
[26] S. Zhang, T. Gao, L. Yang, "A reversible data hiding scheme based on histogram modification in integer DWT domain for BTC compressed images," *International Journal of Network Security*, vol. 18, no. 4, pp. 718–727, 2016.
[27] X. Zhang, "Reversible data hiding with optimal value transfer," *IEEE Transactions on Multimedia*, vol. 15, no. 2, pp. 316–325, 2013.
[28] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. IT-23, no. 3, pp. 337–343, 1977.
[29] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. IT-24, no. 5, pp. 530–536, 1978.

Biography

Chin-Chen Chang received his Ph.D. degree in computer engineering from National Chiao Tung University. His current title is Chair Professor in Department of Information Engineering and Computer Science, Feng Chia University, from February 2005. He is currently a Fellow of IEEE and a Fellow of IEE, UK. And, since his early years of career development, he consecutively won Outstanding Talent in Information Sciences of the R. O. C., AceR Dragon Award of the Ten Most Outstanding Talents, Outstanding Scholar Award of the R. O. C., Outstanding Engineering Professor Award of the R. O. C., Distinguished Research Awards of National Science Council of the R. O. C., Top Fifteen Scholars in Systems and Software Engineering of the Journal of Systems and Software, and so on. On numerous occasions, he was invited to serve as Visiting Professor, Chair Professor, Honorary Professor, Honorary Director, Honorary Chairman, Distinguished Alumnus, Distinguished Researcher, Research Fellow by universities and research institutes. His current research interests include database design, computer cryptography, image compression, and data structures.

Ngoc-Tu Huynh received the BS degree in mathematics – informatics in 2006 from Danang University, Vietnam, and the MS degree in information engineering and computer science in 2010 from Feng Chia University. Since 2006, she has been a lecturer of Department of Computer Science, College of Information Technology, Danang University, Vietnam. She is currently pursuing her Ph.D in information engineering and computer science, Feng Chia University, Taichung, Taiwan. Her research interests include visual cryptography, watermarking, steganography and image processing.

Yu-Kai Wang received his MS degree from Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan. His current research interests include information hiding and image processing.

Yanjun Liu received her Ph.D. degree in 2010, in School of Computer Science and Technology from University of Science and Technology of China (USTC), Hefei, China. She has been an assistant professor serving in Anhui University in China since 2010. She currently serves as a senior research fellow in Feng Chia University in Taiwan. Her specialties include E-Business security and electronic imaging techniques.