Soft Comput (2018) 22:4445–4455
https://doi.org/10.1007/s00500-017-2651-2

METHODOLOGIES AND APPLICATION

An optimized data hiding scheme for Deflate codes

Yuan Xue[1] · Yu-an Tan[1] · Chen Liang[1] · ChangYou Zhang[2] · Jun Zheng[1,3]

Published online: 1 June 2017

Abstract Compression file is a common form of carriers in network data transmission; therefore, it is essential to investigate the data hiding schemes for compression files. The existing data hiding schemes embed secret bits by shrinking the length of symbols, while they are not secure enough since the shrinking of symbol length is easily detected. First, we propose a longest match detecting algorithm that can detect the data hiding behavior of shrinking the length of symbols, by checking whether items of the generated dictionary are longest matches or not. Then, we propose a secret data hiding scheme based on Deflate codes, which reversibly embeds secret data by altering the matching process, to choose the proper matching result that the least significant bit of length field in [distance, length] pair is equal to the current embedded secret bit. The proposed data hiding scheme can resist on the longest match detection, and the embedding rate is higher than DH-LZW algorithm. The experiment shows that the proposed scheme achieves 5.12% of embedding rate and 10.18% size increase in the compressed file. Moreover, an optimization is made in providing practical suggestion for DH-Deflate data hiding. One can choose which format and size of files are to be selected based upon the optimization, and thus, data hiding work can be achieved in a convenient and targeted way.

Keywords Steganography · Information hiding · Deflate coding algorithm · Optimization

Communicated by V. Loia.

✉ Jun Zheng
zhengjun_bit@163.com

Yuan Xue
xueyuan_1007@163.com

Yu-an Tan
tan2008@bit.edu.cn

Chen Liang
1342313537@qq.com

ChangYou Zhang
changyou@iscas.ac.cn

[1] School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China

[2] Institute of Software, Chinese Academy of Sciences, Beijing 100190, China

[3] Beijing Engineering Research Center of Massive Language Information Processing and Cloud Computing Application, Beijing 100081, China

1 Introduction

The rapid development of the Internet requires the lightweight of data and its storage security much more than before (Parah et al. 2015; Zhang et al. 2017; Utku Celik et al. 2005; Ruijin et al. 2016). Accordingly, users need effective tools to decrease the size of large data blocks before data transmission (Xuan et al. 2005; Tseng and Chang 2004; Guo and Liu 2012; Mali et al. 2012; Zhu et al. 2017).

The wide application of compressing tools inspires a new way for data security, which is to embed secret data in compressed files (Tseng and Chang 2004; Guo and Liu 2012; Guo and Tsai 2012; Jian et al. 2015; Xuan et al. 2004; Nikolaidis 2015). Such kind of methods toughen the process of discovering hidden data for adversaries, since a proper decompressing method is essential to obtain hidden data (Parah et al. 2015; Lee et al. 2012; Zhang et al. 2013; Yan et al. 2016; Kumar and Pooja 2010). Additionally, these methods have advantages over reducing transmission costs and simultaneously make the transmission more secure (Nikolaidis 2015; Wu et al. 2006; Chang et al. 2006, 2016; Moulin

and Koetter 2005; Zhu et al. 2016; Najafi 2007; Liu and Tsai 2007).

Past 10 years has seen the development of data hiding algorithms based on compressed tools (Tseng and Chang 2004; Guo and Tsai 2012; Xuan et al. 2004; Kumar and Pooja 2010; Chang et al. 2006; Moon and Kawitkar 2007; Yadav et al. 2012). Among large number of compressing tools, Lempel–Ziv–Welch (LZW) coding is a widely applied lossless compression algorithm and Shim et al. (2004) proposed the DH-LZW scheme by one character of one symbol to hide the data. Chen and Chang (2010) proposed the HCDH-LZW scheme in which received a higher embedding capacity than Shim et al.'s scheme by shrinking the characters according to the length of the symbol used to hide the data. Wang et al. (2013) proposed the HPDH-LZW scheme by modifying the value of LZW codes to hide secret data.

Schemes based on LZW hold the advantage getting a higher capacity for data hiding as well as the convenience of secret data extracting (Shim et al. 2004; Chen and Chang 2010; Bender et al. 1996). However, on the one hand, since many optimized schemes have been proposed, which hold higher performance and prevalence, new data hiding algorithm is needed to adapt to these optimized schemes for both the less time consumption and better compression rate. On the other hand, since most of these schemes hide secret data by changing the length of symbols, we discover a detecting method which can detect the existence of hiding data in such kind of schemes. Thus, those schemes may not be considered as secure ones.

To make up for this defect, our proposed scheme, named DH-Deflate, is based on a kind of optimized and prevalence compressing algorithm, Deflate, which is widely applied in compressing tools and considered to outperform the original algorithm, LZ'77. Besides the application of Deflate, our scheme also changes the algorithm of data hiding to embed secret data in the process of altering the symbol producing process, which disables the proposed detecting method to discover hidden data.

Besides the proposed data hiding scheme, we also work on providing experiential advice for compression data hiding achievers. That is, once the size of carriers is constrained and the embedded rate is specifically required, an advice of choosing proper file formats can be generalized. Thus, an optimization is done to shed light on the relationship of data embedding rate between carrier files format and redundancy.

Our contribution:

1. DH-Deflate a data hiding scheme based upon Deflate algorithm is proposed to achieve data hiding tasks which owns compatibility and could be transplant to LZ'77 series algorithms. To the best of our knowledge, this is the first data hiding scheme realized in Deflate.
2. A longest match detecting algorithm is proposed to judge if compression files contain embedded data. This algorithm can be applied to detect most existed data hiding algorithms which achieve data hiding based on shrinking the length of the longest match. Of course, DH-Deflate is an exception for our detecting algorithm.
3. An optimization is done to provide advisable inspiration to data hiding work achievers. They can achieve data hiding in a more purposive and convenient way by utilizing the optimization in this paper.

The following section is devoted to an overview of data hiding algorithms based on compressed file and our proposed scheme. In Sect. 2, we will review several related compressing algorithms and data hiding schemes. Section 3 proposes a longest match detecting algorithm for detecting the existence of LZW-based hidden data. In Sect. 4, we propose our data hiding scheme. The experimental results and optimization are illustrated in Sect. 5. Limitation and future work are described in Sect. 6, and Sect. 7 shows our conclusion.

2 Preliminaries

In this section, LZW compression algorithm and its several data hiding schemes including DH-LZW scheme, HCDH-LZW scheme, HPDH-LZW scheme are introduced. Deflate compression algorithm is also briefly described.

2.1 LZW algorithm

LZW is a lossless data compression algorithm proposed by Abraham Lempel, Jacob Ziv, and Terry Welch, which codes variable-length symbols with fixed-length codes in the dictionary (Welch 1984). During encoding phase, sequences which do not exist in dictionary will be regarded as a new symbol and inserted into the next unused location of dictionary; then, the sequences are partly coded and outputted. In LZW algorithm, the dictionary initializes with ASCII values 0–255; thus, the shortest code length is 9 bits (Wu et al. 2006). During decoding phase, encoded data are decoded according to its dictionary. Since this phase can build a same dictionary with encoding phase automatically, receivers do not need to synchronize its dictionary with senders.

2.2 DH-LZW data hiding scheme

The DH-LZW scheme is a data hiding algorithm based on LZW algorithm which is proposed by Shim et al. (2004). Via dropping the last character of the symbols, secret data can be embedded into the compressed files by bytes.

The basic idea of DH-LZW is to modify symbol's length during the encoding phase until the least significant bit of

Table 1 Data hiding and data extracting phases of DH-LZW

Data hiding					Data extracting				
Input	Original code	Output	New item	Hidden bit	Input	Old code	Output	New item	Extracted bit
aa	97	97	256 = aa		97		a		
b	97	97	257 = ab		97	97	a	256 = aa	
b	98	98	258 = bb		98	97	b	257 = ab	
a	98	98	259 = ba		98	98	b	258 = bb	
aa	256	256	260 = aaa		256	98	aa	259 = ba	
bb	257	257	261 = abb		257	256	ab	260 = aaa	
ba	258	258	262 = bba		258	257	bb	261 = abb	
bbb	261	257	263 = abb	0	257	258	ab	262 = bba	
ab	262	258	264 = bba	0	258	257	bb	263 = abb	0
ba	261	261	265 = abba	1	261	258	abb	264 = bba	0
aa	260	260			260	261	aaa	265 = abba	1

symbol's length is equal to embedding secret bit, which means if the least significant bit of symbol's length equals to the hidden bit, the symbol will remain unchanged, else drop the last character of symbol to let them consistent. Thus, during the decoding phase, the embedded data can be extracted by figuring out the least significant bit of each symbol's length.

Accounting for algorithm's efficiency (Wang et al. 2013), the DH-LZW scheme sets a threshold called THD to judge if a symbol is available to hide secret bit, a symbol cannot be used to hide secret bit if its length is not larger than THD. When extracting embedded bits, if a symbol's length is not larger than THD, this symbol's least significant bit would not be outputted to the secret data files.

Example 1 Set the source file "aabbaaabbbabbbabbaaa," THD is 3, and secret file is "001."

An example of DH-LZW schemes is given in Example 1, and its specific results are shown in Table 1. In encoding phase, before the 263rd item is generated, the length of symbol "abbb" is 4, which can embed a secret bit. The first secret bit is "0" which is not equal to (4–1); thus, the last character of symbol "abbb" is dropped. Similarly, when getting the 264th item, symbol "bbab" drops the character "b" to hide the second secret bit "0." And before the 265th item is added to the dictionary, symbol "abba" whose length is 4 can be used to hide a secret bit. The third secret bit "1" equals to (4–1), and the symbol remains unchanged.

In decoding phase, the 263rd and 264th items have already existed in the dictionary, and we can get the secret bit "0" from the least significant bit of (3–1). The length of 265th item is 4, from which the secret bit "1" can be extracted since the least significant bit of (4–1) is 1.

2.3 Optimized schemes of DH-LZW

Some optimized schemes of DH-LZW have been proposed in the last several years. Chen and Chang (2010) proposed the HCDH-LZW scheme, which enlarged the data hiding capacity of the original scheme. The main idea of HCDH-LZW scheme is to shrink the symbol according to the symbol's length, $m = \log_2(n-1)$, where m is how many bit can be embedded in one symbol and n is symbol's length. For example, if symbol's length is 2 or 3, it can be used to hide 1 bit, or if the length is between 4 and 7, 2 bits can be hided in the symbol. Wang et al. (2013) proposed the HPDH-LZW scheme, which realized data hiding by modifying LZW code according to the hidden bit. For example, if the hidden bit is "0," the output LZW code will remain equal to the original LZW code; however, if the hidden bit is "1," the output LZW code will be the sum of original LZW code and currently dictionary size.

2.4 Deflate algorithm

Deflate algorithm is an optimize compressing algorithm based on LZ'77 algorithm (Lonardi et al. 2007; Lonardi and Szpankowski 2003). Different from LZW algorithm, Deflate algorithm compresses files by representing sequential symbols with its longest match's distance and length pairs in sliding window. A sliding window is a buffer which stores the compressed data blocks.

During the encoding phase, at very beginning, sliding window fills itself with data blocks and encodes them directly. Then, sliding window scans source files and encodes new data block with longest matches buffered in the window.

Accounting for algorithm's efficiency, if a symbol's longest match length is less than a threshold (usually set to 3), encoder will skip this symbol, else the encoder calculates

Table 2 Compression phase and decompression phase of Deflate

Compression			Decompression		
Input	Window	Output	Input	Window	Output
a		a	a		a
a	a	a	a	a	a
b	aa	b	b	aa	b
b	aab	b	b	aab	b
a	aabb	a	a	aabb	a
aabb	aabba	[5,4]	[5,4]	aabba	aabb
b	aaabb	b	b	aaabb	b
abbb	aabbb	[4,4]	[4,4]	aabbb	abbb
abb	babbb	[4,3]	[4,3]	babbb	abb
a	bbabb	a	a	bbabb	a
a	babba	a	a	babba	a
a	abbaa	a	a	abbaa	a

the distance between the longest match and symbol and conducts a [distance, length] pair. Then, the [distance, length] pair will be encoded to Huffman codes as output. Finally, the sliding window slides "distance" length and encodes the following data blocks.

In decoding phase, [distance, length] pairs are decoded according to the slide window. A simple example of the Deflate algorithm is presented in Table 2 for providing a better understanding of Deflate.

Example 2 The size of sliding window is assumed to be 5. The source file is assumed to be "aabbaaabbbabbbabbaaa." [5, 4] means distance is 5 and length is 4.

It can be seen from Table 2, in the compression phase, before the sliding window is fully filled, the input character is coded directly. The window then begins to slid and generates [distance, length] code pairs. The encoder tries to find the longest match and express it with [distance, length]. For those whose length of longest match is shorter than 3, the encoder codes it directly. In the data extracting phase, a single character is directly decoded and the [distance, length] pairs are transformed to strings according to the window.

3 Detect data hiding behavior of altering symbol length

3.1 Longest match detecting algorithm

The proposed longest match detecting algorithm can be applied to detect whether LZW compressed files contain secret data. In LZW algorithm encoding phase, all items in the dictionary are unique, which means each new symbol must be a longest match. However, in DH-LZW algorithm encoding phase, not all items are unique, since symbol may be modified for data hiding. Thus, new symbol may not always be a longest match. The possibilities that a symbol is a longest match or not are approximately equal, which means the more the secret bits, the higher the possibility we can discover a non-longest match. For practical reason, we always try to embed as much secret data as possible; in such situation, our detecting scheme obtains a very high possibility of detecting if a file contains hidden data. The algorithm is summarized as shown in Algorithm 1. With this algorithm, rivals can easily figure out if there are hidden data.

Algorithm 1 The data hiding detection algorithm

```
1  let T, buffer, i, f ← φ, φ, 1, 0
2  while have not finished decoding yet
3     phrase i ←decode the next phrase as according to the standard
LZW decoder
4     T ←The first character of phrase i
5     buffer ← phrase i
6     if i > 1
7        check the existence of phrase(i − 1) + T in buffer
8     if (exist)
9        f ← 1
10       i ← i + 1
11 return (f)
```

3.2 Examples of longest match detecting algorithm

To show how the longest match detecting algorithm works, an example is given in Table 3. The source files and secret files are all completely same with the corresponding examples given in Sect. 2. Example 3 is the longest match detection for DH-LZW.

For HCDH-LZW, since this scheme also embeds secret bit via changing symbols' length, the longest match detection can also detect whether there are hidden bits or not, just as DH-LZW does.

Table 3 Longest match detection for DH-LZW

Data extracting					
Input	Old code	Output	New item	Longest match	Detect flag
97		a			
97	97	a	256 = aa	aa	0
98	97	b	257 = ab	ab	0
98	98	b	258 = bb	bb	0
256	98	aa	259 = ba	ba	0
257	256	ab	260 = aaa	aaa	0
258	257	bb	261 = abb	abb	0
257	258	ab	262 = bba	bba	0
258	257	bb	263 = abb	abbb	1
261	258	abb	264 = bba	bbab	1
260	261	aaa	265 = abba	abba	0

For HPDH-LZW, this scheme does not change symbols' length, and the long longest match detection cannot find the hidden bits. However, as it has been introduced before, rivals can easily use other methods to find the hidden data.

It can be seen that longest match detection for DH-LZW and HCDH-LZW can detect if there is a hidden bit if only there is a change of symbol length.

4 The secure data hiding scheme for Deflate codes

Instead of embedding secret bits via shrinking symbols, our proposed scheme embeds secret bits into the longest match generating process. Thus, this scheme could confirm that all symbols generated will not be shrunken and remained as longest matches.

4.1 Data embedding

The encoding phase of the proposed scheme is shown in Fig. 1. n denotes the longest match's length, the generating phase of which is controlled to hide a secret bit. The encoder scans the whole sliding window from beginning side to end and finds out all of the longest matches in the sliding window (noting $T_1, T_2, \ldots$), their lengths are calculated (marked in $n_1, n_2, \ldots$), and those whose length is too short are dropped. Different from the original Deflate algorithm where $l = \{n_1, n_2, \ldots\}_{\max}$, the longest match is select from $(T_1, T_2, \ldots)$ in the principle of $l = \{n_k \mid n_k(\mathrm{mod}2) = b\}_{\max}$, $T = \{T_k \,|n_k = l, d_k = \{d_i \,|n_i = l\}_{\min}\}$. Thus, each successful match phase can embed a hidden bit. If $T = \phi$, the current position's character will be coded singly. The data hiding algorithm (DH-Deflate) is summarized as shown in Algorithm 2.

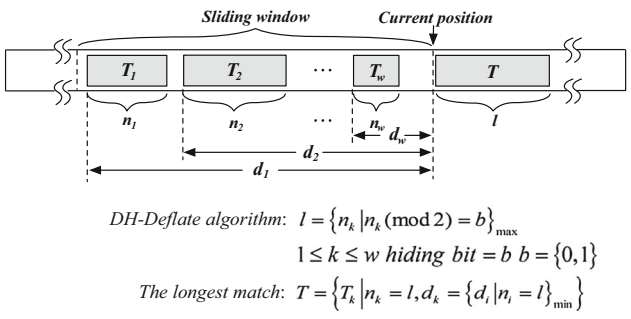

Fig. 1 Embedding strategy of the proposed scheme

Algorithm 2 DH-Deflate encoding algorithm

Input Source file and Secret file
Output Deflate code
1 **let** $b, THD, i, m, P \leftarrow \{0, 1\}, \phi, 0, |M|, ||$
2 **let** $THD \geq 3$
3 **while** $i < m$ do
4 **while** $n \geq THD$ do
5 **let** $R \leftarrow \{(d_1, n_1), (d_2, n_2), ..., (d_w, n_w)\}$be the character string set of longest match in sliding window
6 **if** $\exists k\ (1 \leq k \leq w)\ n_k(\mathrm{mod}2) = b$
7 **append** $T = \{T_k \,|n_k = l, d_k = \{d_i \,|n_i = l\}_{\min}\}$
8 **append** $T_k \leftarrow (d_k, n_k)$
9 **else** Code the single character separately.
10 **append** $p \leftarrow p + 1$
11 **while** $n < THD$ do
12 Code the single character separately
13 **return** P

4.2 Data extraction

In the data extracting procedure, assuming the current length of longest match is C', if C' is larger than 2, the secret bit is equal to the least significant bit of C'. Each [distance, length] pair contains a secret bit. For example, if C' is 6, then the secret bit is "0"; if C' is 5, then the secret bit is "1." The data extracting algorithm is summarized as shown in Algorithm 3.

Algorithm 3 The data extracting algorithm of DH-Deflate

Input Deflate code
Output Source file and Secret file
1 **let** $P, M \leftarrow$empty string, empty string
2 **for each** $(d, l) \in P$ do
3 **while** $n < THD$
4 Code the single character separately
5 **while** $n \geq THD$
6 $b' = n_k(mod2)$
7 **append** $P \leftarrow b'$
8 **append** $M \leftarrow (d_k, n_k)$
9 **return** (P, M)

To show our proposed scheme more intuitively, an example is given in Table 4. The compressing phase's 6th–11th input is given in Table 4.

Example 3 Assuming that the size of slid window is 5, the source file is "aabbaaabbbabbbabbaaa." The secret bits "001," [5,3] means distance is 5 and length is 3.

In Table 4, when encoder is going to compress the 6th input character, the longest match is "aabb," the length of "aabb" is 4, and the least significant bit of which is equal to the first secret bit "0," which means it is available to hide the secret bit "0." The 8th input has the longest match "abbb," and the least significant bit of "abbb" is "0" which is equal to the second secret bit. The 9th input has the longest match "abb," least significant bit of which is "1" equaling to the third secret bit. In data extracting phase, the decoder finds all of [distance, length] pairs and their parities of length are equal to the secret file "001."

4.3 Performance discussion

This section deduces the performance of the proposed scheme. We analyze M_n for a binary source, $X = X_1X_2$. $X_3 \ldots$, where the x_i are random variables on the binary alphabet with $Pr(x_i = 0) = p_i$ and $Pr(x_i = 1) = q_i$. $Y = Y_1Y_2Y_3 \cdots$is the hiding data series where the Y_j are random variables on the binary alphabet with $Pr(Y_j = 0) = p_j{}'$ and $Pr(Y_j = 1) = q_j{}'$. Our goal is to figure out the probability of a successful data hiding for DH-Deflate. Firstly, assuming that the size of sliding window is w byte, we conclude (1), in which $Pr\left(\underset{m}{=} n\ byte\right)$ presents the longest match's length in position m byte(m is in the sliding window).

$$Pr\left(\underset{m}{=} n\ byte\right) = \prod_{v=0}^{8n-1} (p_{m+v}p_{i+w+v+1} + q_{m+v}q_{i+w+v+1}) \tag{1}$$

Nextly, we try to work out the probability of the longest match's length equals to n, which is marked as $Pr(\underset{\max}{=} n\ byte)$, and here we conclude (3). The $Pr(\underset{t}{<} n\ byte)$ in (3) is deduced from (1) which is represented in (2).

$$Pr\left(\underset{m}{<} n\ byte\right) = \sum_{r=1}^{n=1} Pr\left(\underset{m}{=} r\ byte\right) \tag{2}$$

$$\begin{aligned} Pr\left(\underset{\max}{=} n\ byte\right) &= \sum_{m=1}^{w} Pr\left(\underset{m}{=} n\ byte\right) \frac{\prod_{t=1}^{w} Pr(\underset{t}{<} n\ byte)}{Pr(\underset{m}{<} n\ byte)} \\ &+ \sum_{\substack{m_1=1 \\ (m_1 \neq m_2)}}^{w} \sum_{m_2=1}^{w} Pr\left(\underset{m_1}{=} n\ byte\right) Pr\left(\underset{m_2}{=} n\ byte\right) \\ &\frac{\prod_{t=1}^{w} Pr(\underset{t}{<} n\ byte)}{Pr(\underset{m_1}{<} n\ byte) Pr(\underset{m_2}{<} n\ byte)} + \cdots \\ &+ \sum_{\substack{m_1=1 \\ (m_a \neq m_b, if\ a \neq b)}}^{w} \sum_{m_2=1}^{w} \cdots \sum_{m_w=1}^{w} Pr\left(\underset{m_1}{=} n\ byte\right) \\ &\cdots Pr\left(\underset{m_w}{=} n\ byte\right) \frac{\prod_{t=1}^{w} Pr(\underset{t}{<} n\ byte)}{Pr(\underset{m_1}{<} n\ byte) \cdots Pr(\underset{m_w}{<} n\ byte)} \end{aligned} \tag{3}$$

Table 4 Data hiding and data extracting phases of DH-Deflate

Data hiding				Data extracting			
Input	Window	Output	Secret bit	Input	Window	Output	Extracted bit
a		a		a		a	
a	a	a		a	a	a	
b	aa	b		b	aa	b	
b	aab	b		b	aab	b	
a	aabb	a		a	aabb	a	
aabb	aabba	[5,4]	0	[5,4]	aabba	aabb	0
b	aaabb	b		b	aaabb	b	
abbb	aabbb	[4,4]	0	[4,4]	aabbb	abbb	0
abb	babbb	[4,3]	1	[4,3]	babbb	abb	1
a	bbabb	a		a	bbabb	a	
a	babba	a		a	babba	a	
a	abbaa	a		a	abbaa	a	

Fig. 2 Five test images

To figure out the probability of a successful data hiding for DH-Deflate in condition that the longest match's length is n, we calculated $Pr(n \ (\mathrm{mod}\ 2)) = Y_j{}'$ in (4), and then, we get the $Pr(success)$ in (5). Finally, our aim $Pr(success)$ is deduced in (6). As it has been shown above, the success rate of DH-Deflate is correlated with both the carrier source and the hiding data series.

$$\begin{aligned} Pr(n\ (\mathrm{mod}2) = Y_j{}') &= Pr((Y_j = 0)\&(n \underset{\max}{=} 4, 6, 8\cdots)) \\ &+ Pr((Y_j = 1)\&(n \underset{\max}{=} 3, 5, 7\cdots)) \\ &= p_j{}'(Pr(\underset{\max}{=} 4) + Pr(\underset{\max}{=} 6) + Pr(\underset{\max}{=} 8)\cdots) \\ &+ q_j{}'(Pr(\underset{\max}{=} 3) + Pr(\underset{\max}{=} 5) + Pr(\underset{\max}{=} 7)\cdots) \end{aligned} \tag{4}$$

$$\begin{aligned} Pr(\underset{n}{success}) &= Pr((\underset{\max}{=} n\ byte)\&n(\mathrm{mod}2) = Y_j{}') \\ &+ Pr((\underset{\max}{=} n\ byte)\&(n(\mathrm{mod}2) \neq Y_j{}')\&(\underset{\mathrm{m}}{=}(n-1)\ byte)) \\ &+ Pr((\underset{\max}{=} n\ byte)\&(n(\mathrm{mod}2) \neq Y_j{}') \\ &\quad \&(\underset{\mathrm{m}}{=}\overline{(n-1)\ byte})\&(=(n-3)\ byte)) \\ &+ \cdots + Pr((\underset{\max}{=} n\ byte)\&(n(\mathrm{mod}2) \neq Y_j{}') \\ &\quad \&(\underset{m}{=}\overline{(n-1)\ byte})\&(\underset{m}{=}\overline{(n-3)\ byte}))\&\cdots \\ &\quad \& \begin{cases} (\underset{m}{=}\overline{5\ byte})\&(\underset{m}{=}3\ byte)(n = 3, 5, 7, \cdots) \\ (\underset{m}{=}\overline{6\ byte})\&(\underset{m}{=}4\ byte)(n = 4, 6, 8, \cdots) \end{cases}) \end{aligned} \tag{5}$$

$$Pr(success) = \sum_{s=3}^{\infty} Pr(\underset{\max}{=} s\ byte)Pr(\underset{s}{success}) \tag{6}$$

5 Experimental result

5.1 Performance evaluation via scheme comparison

The proposed scheme is implemented based on zlib ver 1.2.8, an frequently used open-source library of providing compress/decompress features, and we modify the "longest_match()" function in Deflate.c which controls the generation of longest matches. Nearly, 300 lines of C code are added.

This experiment is done on Win7 64-bit operating system, Intel(R) Core(TM) i5-3210M CPU@2.50GHz,4 GB RAM.

In our experiment, four text files and five image files are used to evaluate the performance of the proposed scheme. For the text files, "Obama" is Obama's speech; "Lincoln" is Lincoln's Gettysburg address; "Hamlet" is one of the Shakespear's famous works; and "Pride" is a novel named "Pride and prejudice" written by Jane Austen, first published in 1813. Paper-b*.txt and paper-s*.txt in the same size and format of Paper-b.txt and paper-s.txt in Chen and Chang (2010) are chosen as testing samples. Such replacement is reasonable since DH-Deflate performs stable among txt carriers which will be deduced in the following Sect. 5.2. For the image files, five popular 256*256 images, as shown in Fig. 2, are included, such as Baboon, Goldhill, Boat, Lena, and Pepper.

We choose the DH-LZW scheme as a reference because both DH-LZW and our proposed scheme try to hide one secret bit in one symbol. The experiment results are shown in Table 5. The hidden secret bits are generated as a random sequence. The sizes of the files and compressed size (LZW and Deflate) shown in tables are measured in bytes, and the hidden data are measured in bit; the data hiding and data extracting time are measured in second. Tables 5 and 6

Table 5 Comparison to DH-LZW scheme for text and image files

File name	Original size	LZW size (A)	Deflate size (B)	DH-LZW		DH-Deflate		Embedded rate		Variant quantities	
				Size (C)	Hidden bit (D)	Size (E)	Hidden bit (F)	DH-LZW D/8*C	DH-Deflate F/8*E	DH-LZW 8*(C-A)/D	DH-Deflate 8*(E-B)/F
Obama	13, 504	9, 992	6, 158	10, 184	670	6,612	2,349	0.82%	4.44%	2.29	1.55
Lincoln	1, 450	1, 027	704	1, 059	94	737	193	1.11%	3.27%	2.72	1.37
Paper-b*	80, 316	57, 677	33, 361	59, 513	5,459	36,861	13,578	1.15%	4.60%	2.69	2.06
Paper-s*	20, 079	14, 477	9, 678	14, 869	1,358	10,368	3,229	1.14%	3.89%	2.31	1.71
Baboon	65, 536	71, 824	23, 040	71, 824	3	25,126	12,844	0.00%	6.39%	0.00	1.30
Goldhill	65, 536	67, 024	16, 506	67, 152	500	18,524	9,595	0.09%	6.47%	2.05	1.68
Boat	65, 536	59, 602	16, 251	60, 042	1,833	18,118	8,779	0.38%	6.06%	1.92	1.70
Lena	65, 536	69, 116	16, 900	69158	116	18,774	8,932	0.02%	5.95%	2.90	1.68
Pepper	65, 536	68, 443	15, 230	68, 513	144	17,296	8,355	0.03%	6.04%	3.89	1.98
Average					1,130.78		7539.33	0.53%	5.23%	2.31	1.67

Table 6 Comparison to the original Deflate scheme for data hiding and data extracting time

File name	Filesize	Original Deflate			DH-Deflate				Deflate size increase	Embedding rate
		Hiding time (ms)	Extracting time (ms)	Size(B)	Hiding time (ms)	Extracting time (ms)	Size(E)	Hidden bit (F)	(E-B)/B	F/8*E
Obama	13,504	2.72	0.93	6,158	3.12	1.06	6,612	2,349	7.37%	4.44%
Lincoln	1,450	1.19	0.61	704	1.31	0.7	737	193	4.69%	3.27%
Paper-b*	80,316	1.76	0.72	33,361	54.77	3.8	36,861	13, 578	10.49%	4.60%
Paper-s*	20,079	1.04	0.052	9678	12.56	2.4	10,368	3,229	7.13%	3.89%
Hamlet	180,728	22.96	0.372	77,061	34.46	4.29	85,712	33,203	11.23%	4.84%
Pride	689,323	88.1	12.08	266,907	117.69	13.91	303,436	121,306	13.69%	5.00%
Baboon	65,536	9.36	1.82	23,040	13.37	1.91	25,126	12,844	9.05%	6.39%
Goldhill	65,536	8.89	1.43	16,506	11.45	1.68	18,524	9, 595	12.23%	6.47%
Boat	65,536	9.18	1.54	16,251	12.08	1.88	18,118	8,779	11.49%	6.06%
Lena	65,536	9.94	1.47	16,900	10.67	1.64	18,774	8,932	11.09%	5.95%
Pepper	65,536	8.22	1.43	15,230	10.38	1.73	17,296	8,355	13.57%	6.04%
					Increase	Increase				
Average					42%	35.8%			10.18%	5.12%

compare the hiding capacities of the proposed scheme and the DH-LZW scheme. The variant quantities show on average the influence of embedded data to carrier files' size. The result shows that DH-LZW causes a larger size increment in hiding the same size of data. For testing data, the author of DH-LZW uses four text files and five image files.

According to the proposed scheme, the capacity of secret data that a file can hold is depending on symbols generated in the data hiding phase, which means the amount of [distance, length] pairs. For DH-LZW, the embedding capacity is depended on the number of embeddable symbols, the DH-LZW symbols which are shorter than 3 are not available for data hiding, and hence, the proposed scheme can get a larger capacity for hiding secret data.

Table 6 shows the data hiding and data extracting time of the original Deflate scheme and the proposed scheme. It can be easily seen that the average hiding time of the proposed scheme increases 42%, and the average extracting time of the proposed scheme increases 35.8%.The time of hiding and extracting phase increases because of the existence of hidden bit. In the phase of data embedding, if all lengths of the longest matches in slid window do not match with the embedded hidden bit, the hidden bit will be left until it finds a corresponding longest match, which causes the increase in the hiding time. For the extracting phase, it needs time to calculate the parity of length for each [length distance] pair, which generates the increase in extracting time.

Figure 3 is generated based on Table 5 and shows the variation tendency of embedding rate for both DH-LZW and DH-Deflate. The embedding rate means how many percent of capacity is used to embed secure data bits in the compressed files. It is obviously that DH-Deflate owns a higher

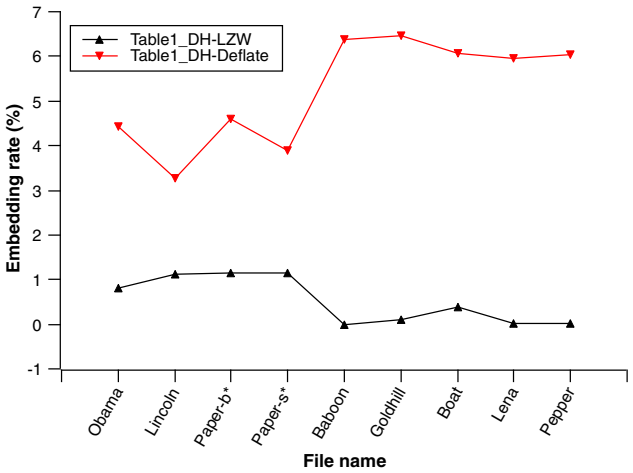

Fig. 3 Embedding rates for the texts and images

embedding rate in our nine testing samples and DH-Deflate performs much better than DH-LZW in image format carriers. The average embedding rate of the DH-LZW and the DH-Deflate, respectively, is 0.53% and 5.23%, and it means how many percent of capacity is used to embed secure data bits in the whole compressed files.

5.2 Hiding capacity inspiration for different file formats

This section discusses the relationship between data redundancy and embedding rate of DH-Deflate while it makes an optimization in providing practical suggestion for data hiding.

Table 7 Test results of 33 typical sample files

File name	File type	Original size (I)	Deflate size (B)	DH-Deflate size (E)	Hidden bit size (F)	Embedding rate (G)	Compression rate (H)	G/H	F/I
A10	.jpg	842,468	841,483	841,979	1,110	0.02%	99.88%	0.02%	0.02%
DSCN3974	.jpg	1,114,198	1,105,070	1,106,377	756	0.01%	99.18%	0.01%	0.01%
DSCN4465	.jpg	694,895	683,616	684,759	367	0.01%	98.38%	0.01%	0.01%
paper3	pdf	130,799	75,627	76,838	5,709	0.93%	57.82%	1.61%	0.55%
paper2	pdf	2,597,907	2,457,775	2,465,497	5,608	0.03%	94.61%	0.03%	0.03%
FlashMX	pdf	4,526,946	3,868,993	3,916,959	130,942	0.42%	85.47%	0.49%	0.36%
MicroSql	.DLL	36,032	18,193	19,176	2,256	1.47%	50.49%	2.91%	0.78%
Microres	.DLL	2,358,944	387,849	425,088	82,074	2.41%	16.44%	14.66%	0.43%
MSO97	.DLL	3,782,416	2,226,806	2,407,974	441,445	2.29%	58.87%	3.89%	1.46%
document2	.doc	872,488	331,607	370,173	136,341	4.60%	38.01%	12.10%	1.95%
document3	.doc	285,184	71,411	81,279	13,730	2.11%	25.04%	8.43%	0.60%
ohs	.doc	4,168,192	1,056,474	1,133,454	108,896	1.20%	25.35%	4.73%	0.33%
Goldhill	.bmp	65,536	16,506	18,524	9,595	6.47%	25.19%	25.68%	1.83%
Boat	.bmp	65,536	16,269	18,063	8,710	6.06%	24.80%	24.44%	1.67%
rafale	.bmp	4,149,414	1,399,893	1,861,304	530,019	3.56%	33.74%	10.55%	1.60%
BINARIES	.txt	17,847	7,102	7,716	2,453	3.97%	39.79%	9.98%	1.72%
GRAPHICS	.txt	14,106	5,341	6,324	1,866	3.69%	37.86%	9.75%	1.65%
world95	.txt	2,988,578	1,019,013	1,148,030	370,047	4.03%	34.10%	11.82%	1.55%
binaries	.dat	2,205,710	676,760	739,125	289,950	4.90%	30.68%	15.97%	1.64%
graphics	.dat	333,535	129,338	140,751	57,549	5.11%	38.78%	13.18%	2.16%
texts	.dat	774,167	268,996	293,419	120,766	5.14%	34.75%	14.79%	1.95%
pg53617	.mobi	645,142	548,916	577,494	16,819	0.36%	85.08%	0.42%	0.33%
pg53618	.mobi	471,573	408,803	417,277	11,821	0.35%	86.69%	0.40%	0.31%
pg53624	.mobi	717,935	622,450	632,486	20,542	0.41%	86.70%	0.47%	0.36%
pg53617	Epub	189,938	177,818	178,360	1318	0.09%	93.62%	0.10%	0.09%
pg53618	Epub	153,883	150,651	150,843	533	0.04%	97.90%	0.04%	0.04%
pg53624	Epub	213,551	211,348	211,509	351	0.02%	98.97%	0.02%	0.02%
every	wav	6,994,092	6,715,767	6,708,656	4,654	0.01%	96.02%	0.01%	0.01%
OnTheRadio	wav	31,907,570	30,574,399	30,616,317	130,555	0.05%	95.82%	0.05%	0.05%
mike	wav	1,708,300	1,208,638	1,264,838	163,565	1.62%	70.75%	2.29%	1.20%
serrano	tif	1,498,414	142,191	184,651	67,901	4.60%	9.49%	48.47%	0.57%
monarch	tif	1,179,784	842,534	905,554	106,066	1.46%	71.41%	2.04%	1.12%
peppers3	tif	786,568	665,630	701,155	46,329	0.83%	84.62%	0.98%	0.74%

In total, 33 typical sample files in 11 different formats (Table 7) are selected from maximum compression library and Internet. All of these file formats are frequently used in huge size data transmission. Their specific basic information (including file format and original size) and experiment data (for example, compression rate) are shown in table. Figure 4 shows the relationship between original file size and embedding file size of DH-Deflate. Figures 5 and 6 give a data hiding inspiration for some most common file types.

In Fig. 4, the histogram shows the rate between embedded size and original file size of 11 common file formats. Displayed in ascending order of the average format rate (the average embedding rate in a same file type), it is obviously that the embedding rates in unit size are varied in different file format. Figure 5 shows the average embedding rate of 11 formats. For format, eg., JPEG, files in such formats have been encoded in a compression way for data transmission, compression tools usually cannot make them further smaller, and some situation may in turn become larger. On the other hand, files in text format such as .bpm or .dat which contains larger redundancy hold the highest average embedding rate in around 5.1%.

When trying to achieve data hiding, if a specific file is affordable in embedding, all secret information shall be taken

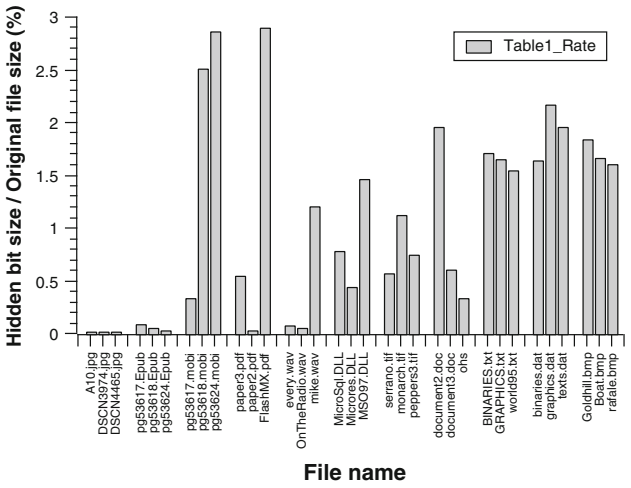

Fig. 4 Relationship of file redundancy and embedded capacity which is illustrated as the rate of embedded size and original file size of 33 typical sample files in 11 different formats

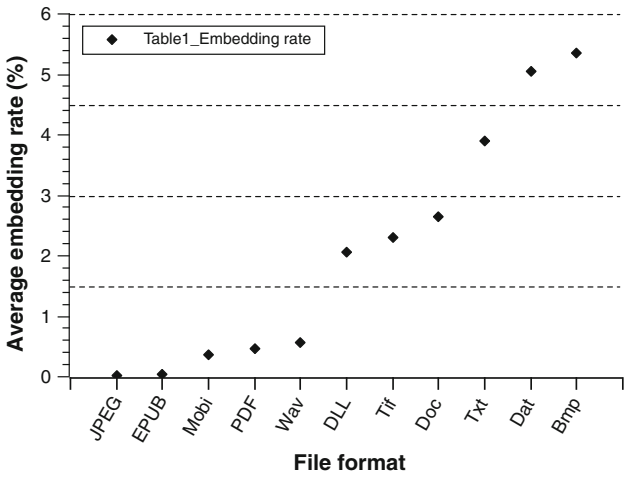

Fig. 5 Distribution of average embedding rate in 11 different formats

into consideration. In this paper, by analyzing the average embedding rate of 11 formats of file carriers, optimization in providing practical suggestion for DH-Deflate data hiding is done based upon experiment results. As Fig. 5 shows, four sections are divided considering the average embedding rates: 0–1.50, 1.51–3.00, 3.01–4.50, and 4.51–6.00%. This brings an inspiration on piratical DH-Deflate data hiding. For example, if 10,000 bytes are to be embedded, the average sizes of carrier files are varying: 76,923.1 KB for JPEG, 2173.9 KB for pdf, 378.8 KB for doc, 256.4 KB for txt and 198.1 KB for dat.

Figure 6 intuitively reveals which format of carrier files is available for achieving some certain DH-Deflate data hiding tasks. If the sizes of carrier files are constrained to no larger than 400 KB while no less than 4 KB data are to be embedded, according to Fig. 6, only the strings of tif, doc, txt, bmp, and dat pass through the section which represents the section met

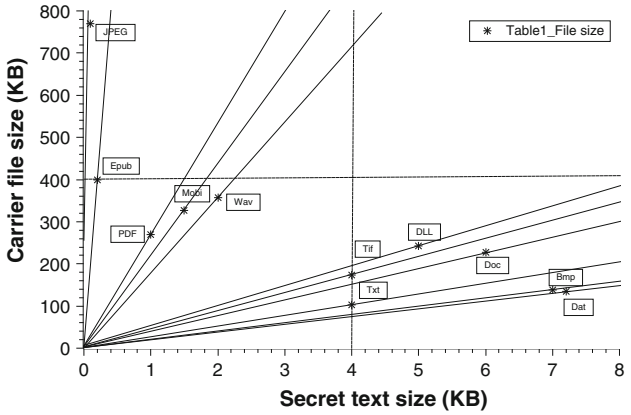

Fig. 6 Format of carrier files available for achieving a certain DH-Deflate data hiding tasks. (Constraint: secret text size = 4 KB, carrier file size $\leq$ 400 KB)

the requirement. Thus, .tif, .doc, .txt, .bmp, and .dat are the possible carriers. One can choose which format and size of files are to be selected and data hiding work can be achieved in a convenient and targeted way.

6 Limitation and future work

DH-Deflate will change the original size of compressed files by altering the compressing process. If rivals can get the relevant information of compressed files such as the version of compression software, compression parameters (window size, compression level, etc.), it is possible to discover whether a suspicious compressed file contains hidden secret data by uncompressing it to get the source data, compressing the source data again with the same version of compression tool and parameters, and then checking if the resulting compressed data are same as the suspicious one. However, since the compress file usually does not contain all of above information, and there are a plenty more of compression tools frequently used, it is hard for rivals to detect the existence of secret data embedded by our proposed scheme.

In the future work, we are committed to improving the efficiency of our scheme by referring to the improved scheme of DH-LZW.

7 Conclusion

In this paper, we proposed a longest match detecting algorithm which can detect the existence of secret embedded by the existing LZW data hiding scheme. We also proposed DH-Deflate, a secure data hiding scheme based on the Deflate compression code, that can fight against the above detecting algorithm. The experiment shows the proposed scheme

achieves 5.12% of embedding rates with the cost of 10.18% size increase in the compressed file. In addition, the data hiding performance of Deflate algorithm is much better than LZW algorithm because that Deflate codes use different ways to store the redundant information. Moreover, the Deflate algorithm is now widely used in many compression tools, and our proposed scheme shall be much easier to deploy than the existing LZW-based ones. Lastly, via optimization analysis, it can be found that formats with high redundancy such as .txt, .dat, .bmp own higher and stable embedding rate.

Acknowledgements This research was supported by the National Natural Science Foundation of China (Nos. U1636213, 61370063, 61379048, 61672508).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

Bender W, Gruhl D, Morimoto N, Lu A (1996) Techniques for data hiding. IBM Syst J 35((3.4)):313–336

Chang C-C, Tai W-L, Lin C-C (2006) A reversible data hiding scheme based on side match vector quantization. IEEE Trans Circuits Syst Video Technol 16(10):1301–1308

Chang C-C, Nguyen T-S, Lin M-C, Lin C-C (2016) A novel data-hiding and compression scheme based on block classification of SMVQ indices. Digit Signal Proc 51:142–155

Chen C-C, Chang C-C (2010) High-capacity reversible data-hiding for lzw codes. In: Computer modeling and simulation, 2010. ICCMS'10. Second international conference on, vol 1. IEEE, pp 3–8

Guo J-M, Liu Y-F (2012) High capacity data hiding for error-diffused block truncation coding. IEEE Trans Image Process 21(12):4808–4818

Guo J-M, Tsai J-J (2012) Reversible data hiding in low complexity and high quality compression scheme. Digit Signal Proc 22(5):776–785

Jian LI, Pan Z, Zheng J, Sun F, Xinxin YE, Yuan K (2015) The security analysis of quantum sagr04 protocol in collective-rotation noise channel. Chin J Electron 24(4):689–693

Kumar A, Pooja K (2010) Steganography-a data hiding technique. Int J Comput Appl 9(7):19–23

Lee Y-P, Lee J-C, Chen W-K, Chang K-C, Jiunn S, Chang C-P (2012) High-payload image hiding with quality recovery using tri-way pixel-value differencing. Inf Sci 191:214–225

Liu T-Y, Tsai W-H (2007) A new steganographic method for data hiding in microsoft word documents by a change tracking technique. IEEE Trans Inf Forensics Secur 2(1):24–30

Lonardi S, Szpankowski W (2003) Joint source-channel lz'77 coding. In: Data compression conference, 2003. Proceedings. DCC 2003. IEEE, pp 273–282

Lonardi S, Szpankowski W, Ward MD (2007) Error resilient lz'77 data compression: algorithms, analysis, and experiments. IEEE Trans Inf Theory 53(5):1799–1813

Mali SN, Patil PM, Jalnekar RM (2012) Robust and secured image-adaptive data hiding. Digit Signal Proc 22(2):314–323

Moon SK, Kawitkar RS (2007) Data security using data hiding. In: Conference on computational intelligence and multimedia applications, 2007. International conference on, vol 4. IEEE, pp 247–251

Moulin P, Koetter R (2005) Data-hiding codes. Proc IEEE 93(12):2083–2126

Najafi HL (2007) A neural network approach to audio data hiding based on perceptual masking model of the human auditory system. Appl Intell 27(3):269–275

Nikolaidis A (2015) Reversible data hiding in jpeg images utilising zero quantised coefficients. IET Image Proc 9(7):560–568

Parah SA, Sheikh JA, Hafiz AM, Bhat GM (2015) A secure and robust information hiding technique for covert communication. Int J Electron 102(8):1253–1266

Ruijin ZHU, Tan Y, Zhang Q, Fei WU, Zheng J, Yuan XUE (2016) Determining image base of firmware files for arm devices. IEICE Trans Inf Syst E99.D(2):351–359

Shim Hiuk Jae, Ahn Jinhaeng, Jeon Byeungwoo (2004) Dh-lzw: lossless data hiding in lzw compression. In: Image processing, 2004. ICIP'04. 2004 International conference on, vol 4. IEEE, pp 2195–2198

Tseng H-W, Chang C-C (2004) High capacity data hiding in JPEG-compressed images. Informatica 15(1):127–142

Utku Celik M, Sharma G, Murat Tekalp A, Saber E (2005) Lossless generalized-lsb data embedding. IEEE Trans Image Process 14(2):253–266

Wang Z-H, Yang H-R, Cheng T-F, Chang C-C (2013) A high-performance reversible data-hiding scheme for lzw codes. J Syst Softw 86(11):2771–2778

Welch TA (1984) A technique for high-performance data compression. Computer 17(17):8–19

Wu Y, Lonardi S, Szpankowski W (2006) Error-resilient lzw data compression. In: Data compression conference (DCC'06). IEEE, pp 193–202

Xuan G, Shi YQ, Ni ZC, Chen J, Yang C, Zhen Y, Zheng J (2004) High capacity lossless data hiding based on integer wavelet transform. In: Circuits and systems, 2004. ISCAS'04. Proceedings of the 2004 international symposium on, vol 2. IEEE, pp II–29

Xuan G, Shi YQ, Yang C, Zheng Y, Zou D, Chai P (2005) Lossless data hiding using integer wavelet transform and threshold embedding technique. In: 2005 IEEE international conference on multimedia and expo. IEEE, pp 1520–1523

Yadav D, Singhal V, Bandil DK (2012) Reversible data hiding techniques. Int J Electron Comput Sci Eng 1(2):380–383

Yan F, Tan Y, Zhang Q, Fei W, Cheng Z, Zheng J (2016) An effective raid data layout for object-based de-duplication backup system. Chin J Electron 25(5):832–840

Zhang X, Tan Y, Xue Y, Zhang Q, Li Y, Zhang C, Zheng J (2017) Cryptographic key protection against frost for mobile devices. Clust Comput 1–10. doi:10.1007/s10586-016-0721-3

Zhang W, Xiaocheng H, Li X, Nenghai Y (2013) Recursive histogram modification: establishing equivalency between reversible data hiding and lossless data compression. IEEE Trans Image Process 22(7):2775–2785

Zhu R, Tan Y, Zhang Q, Li Y, Zheng J (2016) Determining image base of firmware for arm devices by matching literal pools. Digit Investig 16:19–28

Zhu R, Zhang B, Mao J, Zhang Q, Tan YA (2017) A methodology for determining the image base of arm-based industrial control system firmware. Int J Crit Infrastruct Prot 16(3):26–35