Advances in Intelligent Systems Research, volume 172
Sriwijaya International Conference on Information Technology and Its Applications (SICONIAN 2019)

Modified Email Header Steganography Using LZW Compression Algorithm

Varian CAESAR[1], Rinaldi MUNIR[2]

[1]*rinaldi@informatika.org, STEI-ITB, Jalan Ganesha 10, Bandung, Indonesia*
[2]*variancaesar@gmail.com, STEI-ITB, Jalan Ganesha 10, Bandung, Indonesia*

ABSTRACT
Communication by using email has become frequent nowadays and creates a new medium for hiding secret data. Email Header Steganography or HeadStega is a technique in Noiseless Steganography that utilizes the header part of an email such as subject, to, cc, etc. to hide secret messages. HeadStega does not produce noise in data and the body of email remains legitimate. However, HeadStega has a low hiding capacity. A recipient's email address only able to hide 4 to 11 bits of secret message. This paper propose a new technique to improve the hiding capacity of HeadStega especially in the 'to' field of an email. By using LZW compression algorithm and modifying the message-hiding process, the capacity of HeadStega can be improved. The experiment results show that HeadStega hiding capacity increased by 39.72% for text message and increased by 58.80% for binary messages.

Keywords: *Noiseless Steganography, HeadStega, LZW compression*

1. INTRODUCTION

The needs to communicate safely and secretly brings up various technique to conceal and secure the message. Steganography is the art of message concealments in a medium such as picture, audio, video, etc. [2]. Noiseless Steganography, NoStega, is a new paradigm of steganography that converts the secret message into the medium itself [4]. One of the NoStega techniques is Email Header Steganography, HeadStega.

The widespread use of email nowadays and high traffic of email exchanges allows a party to establish a secret communication channel. This is the origin of HeadStega. HeadStega was first proposed by Desoky in 2010, HeadStega takes advantage of email headers to store the hidden message [3]. As the name says, HeadStega only hides secret message in the email header. The body of the email remains genuine and contains no secret message.

Unfortunately, HeadStega has a low capacity to store secret message. In the original paper, one email of HeadStega only able to hide combinations of 4 bits and 7 bits of secret message. Attempt to improve HeadStega hiding capacity have been made before by Hasmawati and Barmawi [6]. They used combination of vocal and consonant patterns to improve the hiding capacity of HeadStega. However, this method only works if the message is a text data. This method also fails if there is no emails in the database that match the pattern. This paper propose a method to improve the hiding capacity of HeadStega in the 'to' field by modifying the message-hiding process and utilizing LZW compression algorithm to reduce the size of the secret messages. This method works for both binary data and text data. The 'to' field of an email were selected because of the frequent use of that field in the HeadStega.

2. PRELIMINARY

This section describes some preliminary studies for HeadStega and LZW Compression. These two concepts are the building block for the proposed method in this paper.

2.1. HeadStega

Email Header Based Steganography (HeadStega) camouflages data or message as email header, such as recipient's email address in the 'to' field. At first, HeadStega will convert secret message into binary string and grouped by a certain length. For example, string 'steganography' converted into '01110011 01110100 01100101 01100111 01100001 01101110 01101111 01100111 01110010 01100001 01110000 01101000 01111001'. This binary string will grouped by certain length, for example, grouped by the length of four will give us: '0111 0011 0111 0100 0110 0101 0110 0111 0110 0001 0110 1110 0110 1111 0110 0111 0111 0010 0110 0001 0111 0000 0110 1000 0111 1001'. The next step is to convert this grouped binary string into a letter that represents them. For example, binary string with length of four can be transformed into a letter by using Table 1 [3].

Table 1. Steganographic Code for 4 bits

Binary	Letters		Binary	Letters
0000	a or q		1000	i or y
0001	b or r		1001	j or z
0010	c or s		1010	k
0011	d or t		1011	l
0100	e or u		1100	m
0101	f or v		1101	n
0110	g or w		1110	o
0111	h or x		1111	p

Using the binary string from the previous example, we convert it into 'h d h e w f g h g b g o w p w x h c g r h a g i h j'. For the rest of this paper, this process will be called **coding.** The final step of HeadStega is generating an email header to hide the message. Based on the collection of letters that we got, iterate each letter and generate a fake recipient's email address with that letter as the prefix. For example, the first two letters from previous example could give us: hancock23@gmail.com and davidson@yahoo.com.

Lempel-Ziv-Welch Compression Algorithm

Lempel-Ziv-Welch (LZW) is a data compression algorithm proposed by Terry A. Welch in 1984 [1]. The working principle of LZW Compression is to replace strings that have already appeared on files with a symbol. Same string table is required to do both compression and decompression. Fig. 1 and Fig. 2 are the pseudocode for compression and decompression using LZW Algorithm [5].

```
MAP<Char><Integer> <-
ASCII CODE
c <- read input

while input not empty:
    symbol <- read input
    if c + symbol is in
MAP:
        c <- c + symbol
    else:
        code <- code +
MAP[c]
        MAP <- INSERT(c +
symbol)
        c <- symbol
return code
```

Figure 1. Pseudocode for LZW Compression Algorithm

```
MAP<Integer><Char> <-
ASCII CODE
TEXT <- " "
code <- read input
c <- MAP[code]
TEXT <- TEXT + c

while input is not empty:
    code <- read input
    if MAP[code] is not
exist:
        ENTRY <- c + c[0]
    else:
        ENTRY <-
MAP[code]
    TEXT <- TEXT + ENTRY
    MAP <- INSERT(c +
ENTRY[0])
    c <- ENTRY
return TEXT
```

Figure 2. Pseudocode for LZW Decompression Algorithm

PROPOSED METHOD

In the original HeadStega, one email only able to hide 4 to 11 bits of secret message, in other words, an email only camouflage half a letter. The proposed method aims

to improve that step by utilizing LZW compression algorithm and modifying the current hiding technique so that an email capable of doing more. The proposed method consists of two major components:

1) Cover Generator
2) Message Extractor

Cover Generator

Cover Generator is part of the system that conceals the secret message into the form of recipient's email address. Cover Generator receives secret message, string table for LZW and domain mapping table as the input and produce list of recipient's email address as the output.

Table 2. Domain Mapping Table

Domain	k		Domain	k
Yahoo.com	0		Lavabit.com	10
Enron.com	1		Comcast.net	11
Hotmail.com	2		Naver.com	12
msn.com	3		qq.com	13
Gmail.com	4		Orange.net	14
Rocketmail.com	5		Mail.ru	15
Verizon.net	6		Skynet.be	16
Yahoo.co.uk	7		Me.com	17
Live.com	8		Mac.com	18
Outlook.com	9		Gmx.com	19

The first step is to compress the secret message using LZW Algorithm. String table will be used in this compression step. After the compression, secret message now become a list of integers. The next step is the email-generating process, the concept of domain mapping table will be introduced here. Domain mapping table is a table consist of various email domain and their respective value of *k*. Table 2 shows the example of domain mapping table that will be used for the rest of this paper.

This value of *k* will be used as additional information in the message extracting process. For example, gmail.com has value of four for its *k*, it shows that the first four letters of any email with 'gmail.com' as the domain, contains the secret message.

In detail, cover generating process starts with compressing the message using LZW algorithm and converting it to list of integers. Then, we convert each integer in the list into binary. After that, we start to parse the binary one by one by traversing the list. If the length of binary string is larger than eight (integer value larger than 255) then we query the database for a random name and append it with the integer value of that binary to form the recipient's email.

If the length of binary is eight or less, then we start reading its first 4 bit and convert it into a letter (coding). From this letter, we query database for a name that starts with that letter as prefix. If such name exists, then we continue to read the next 4 bit of binary string and append the letter with the previous letter to form a longer prefix. This process will continue until database failed to give us a name that matches the prefix. Next, we can generate a fake email by combining the last name given by database and email domain from domain mapping table that has value of k matches the length of prefix. Fig. 3 shows the pseudocode for cover generation process.

Message Extractor

Message extractor is a component that runs on the receiver side. It performs hidden message extraction from the list of emails and do the decompression using LZW Algorithm to get the original message. Message Extractor receives list of emails, LZW string table and domain mapping table as input and produces the original message as the output.

First, the algorithm will take the domain of the email and calculate the *k* value of that domain. If *k* is equal to zero, then we know that the secret message only contained in the numeric part of the username. If *k* is not zero, then we simply extract the first *k* letters from that email to get the hidden message. We also check the numeric part from the username. If an email contains numeric part in its username, we must also extract that numeric part. After done with the message extraction, we convert every element in the binary string into integer and pass it to LZW Decompressor. Fig. 4 shows the pseudocode for message extraction process.

```
m2 <- LZWCOMPRESS(input)
m3 <- TOBINARY(m2)
i <- 0; half <- false

while i < length(m3):
  if length(m3[i]) > 8:
    name <- query database for random
name
    email <- name + to_int(m3[i]) + @ +
domain map for k = 0
    insert email to the result
  else:
    if half is false:
      word <- take first 4 bit from m3[i]
    else:
      word <- take last 4 bit from m3[i]

    l <- take the CODING of word
    name <- query database for name
starts with l

    if name is not empty:
      repeat:
        temp <- name
        word <- take the next 4 bit
        l <- l + CODING of word
        name <- query database for name
starts with l
        if name is not empty:
          half <- not half
          if not half: i + 1
        else: l <- l[:-1]
      until name is empty or i >
length(m3)
      or length(l)>length(domain map)  or
length m3[i] > 8

      if half:
        email <- temp + @ + domain
map[length(l)]
      else if i < length(m3):
        email <- temp + int(m3[i]) + @ +
domain map[length(l)]
      insert email to result

    else:
      name <- query database for random
name
```

```
      email <- name + int(word) + @ +
domain map for k = 0
      half <- not half
      insert email to result
return result
```

Figure 3. Pseudocode for Cover Generator

```
binstring = []
for email in input:
  domain <- get domain from
email
  k <- get k from domain using
domain mapping

  if k = 0:
    num <- get number from email
    binstring <- insert the
binary of num
  else:
    secret <- take first k
letters from email
    for c in secret:
      binary <- convert c to
binary
      if length(binstring[-1]) <
8:
        binstring[-1] <- append
binary
      else:
        binstring <- insert
binary
    num <- get number from email
    if num is not empty:
      binstring <- insert num

result <- convert every element
of binstring into integer
message <- LZWDECOMPRESS(result)
return message
```

Figure 4. Pseudocode for Message Extractor

Proposed Method Example

This subsection provides an end-to-end example on how the proposed method works. Let us define the secret message as "mathematics". First, this secret message will be compressed by LZW into a series of integer:
'109, 97, 116, 104, 101, 256, 116, 105, 99, 115'
.Next, this list of integer will be converted into list of binary:
'01101101', '01100001', '01110100', '01101000', '01100101',
'100000000', '01110100', '01101001', '01100011', '01110011'.

Next, Table 3 will shows the cover generating process. This process gives us six recipient's emails:
gngora@msn.com, raavi116@enron.com, giwargiz@msn.com, vaasen256@enron.com, xeni105@hotmail.com, gtoldstein115@hotmail.com.

Table 3. Cover Generation Process

Binary	Prefix	Email Generated	Note on Prefix
0110	g	-	DB match
1101	gn	-	DB match
0110	gng	-	DB match
0001	gngr	gngora@msn.com	DB failed
0001	r	-	DB match
0111	rx	raavi116@enron.com	DB failed, it creates email with prefix 'r' and add 116 (the next integer) as the part of email
0110	g	-	DB match
1000	gi	-	DB match
0110	giw	-	DB match
0101	giwv	giwargiz@msn.com	DB failed
0101	v	-	DB match
100000000	v	vaasen256@enron.com	Value greater than 255
0111	x	-	DB match
0100	xe	-	DB match
0110	xeg	xenil105@hotmail.com	DB failed, it creates email with prefix 'xe' and add 105 (the next integer) as the part of email
0110	g	-	DB match
0011	gt	-	DB match
0111	gtx	gtoldstein115@hotmail.com	DB failed, it creates email with prefix 'gt' and add 115 (the next integer) as the part of email

After getting list of email addresses, we copy them into header of email. Fig. 5 shows an example of HeadStega above. List of email addresses in the header are as follows: gngora@msn.com, raavi116@enron.com, giwargiz@msn.com, vaasen256@enron.com, xeni105@hotmail.com, gtoldstein115@hotmail.com. Subject and body of email is an arbitrary message, for example, say "Hello". The third party who reads this email will not be suspicious. Only e-mail recipient can decode (and decompress) all e-mail addresses into the original secret message "*mathematics*".

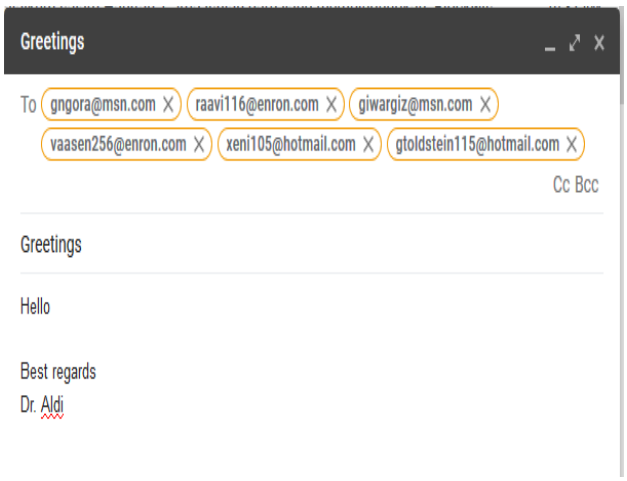

Figure 5. Example of LZW HeadStega

Next, in the receiver side, the receiver will get all those emails and perform an extraction to get the message. Table 4 shows the process of extracting messages from emails. For every letter in the message, the algorithm will convert it into list of binary based on Table 1 in Section 2. For numeric value, that value will be converted into binary. So the resulting binaries from the message extracted are: '01101101', '01100001', '01110100', '01101000', '01100101', '100000000', '01110100', '01101001', '01100011', '01110011'. If we convert it to integer, it will gives the following result: 109, 97, 116, 104, 101, 256, 116, 105, 99, 115. Decompressing it, will gives us the original message, 'mathematics'.

Table 4. Message Extraction Process

Email	k	Message
gngora@msn.com	3	gng
raavi116@enron.com	1	r, 116
giwargiz@msn.com	3	giw
vaasen256@enron.com	1	v, 256
xenil105@hotmail.com	2	xe, 105
gtoldstein115@hotmail.com	2	gt, 115

RESULTS AND DISCUSSION

This section describes performance of proposed method compared to the original HeadStega, in terms of hiding capacity. After that, this section also provide some discussion about the results. For the experiment, the original HeadStega is implemented based on [3, 4] and different type of files is used. Table 5 shows the data used for the experiment.

Table 5. Files used in experiment

File	Size(bytes)
sample1.txt	91
sample2.txt	527
sample4_en.txt	649
sample3.txt	2568
home-icon_16.png	601
landing-page_16.jpg	803
sample_image_32.gif	4286
sample_image_64.ico	16958
code-icns_32.icns	3294
perc13.wav	10504

Experiment conducted in Windows 10 Laptop, 8 GB RAM, AMD A10-5750M and 1 TB memory. The email generated from both method are measured and compared, as shown in Table 6. LZW HeadStega is the name of the proposed method. Efficiency is a comparison between the number of emails generated by the original HeadStega and the number of emails generated by the proposed method.

Table 6. Experiment Result

File	Emails Generated		Efficiency (%)
	HeadStega	LZW HeadStega	
sample1.txt	67	46	31.34
sample2.txt	384	231	39.84
sample4_en.txt	472	281	40.46
sample3.txt	1868	985	47.26
Mean			39.72
home-icon_16.png	438	328	25.11
landing-page_16.jpg	584	385	34.07
sample_image_32.gif	3118	285	90.85
sample_image_64.ico	12334	1326	89.24
code-icns_32.icns	2396	547	77.17
perc13.wav	7640	4863	36.34
Mean			58.80

Effectiveness of the solution

As the Table 6 shows, the proposed method can save the number of email generated by 39.72% for text and 58.80% for binary files. The original HeadStega, one recipient's email address only conceals maximum of 11 bits. In proposed method, one email can conceals more than that. In fact, the proposed method utilize all elements of an email to the max. The LZW Compression also works pretty well on binary files like gif and ico. The compression cuts down the size of the secret message before entering the cover generator.

Resistance against sending limit

Many email provider have a limitation for the number of recipients you can include in the header. For messages that have big size, the email generated will be huge too. But, it can be solved by sending the email in many batch. For example, for 1000 recipient's emails, we can divide it into 5 batch with 200 emails in each batch. To lower the suspicions from third party, sender can use multiple accounts to send the email. For the example, batch-1 will be sent using account1, batch-2 will be sent using account2 and so forth.

CONCLUSION AND FUTURE WORK

The proposed method success to improve the hiding capacity of HeadStega by utilizing LZW Compression Algorithm and modifying the cover generating process. Result shows that the proposed method can improve the hiding capacity by 39.27% for text data and 58.80% for binary data. The key for this improvement is by utilizing an email to contain much information. The writer knows that this work is far from perfect, there are some performance issues like slow embedding time and the needs of large data on the database. This can be improved greatly by using machine learning approach to generate email username rather than querying the database, which takes a lot of time.

REFERENCES

[1] T. A. Welch, *A Technique for High Performance Data Compression*, *Computer*., **17**, 8-19, (1984)

[2] I. J. Cox, M. L. Miller, J. A. Bloom, J. Fridrich, T. Kalker, "Digital Watermarking and Steganography, second ed., 54 (Morgan Kaufmann, Burlington, 2008)

[3] A. Desoky, *HeadStega: e-mail-headers-based steganography methodology*, *Int. J. Electronic Security and Digital Forensics*, **3**, 289-310, (2010)

[4] A. Desoky, Noiseless Steganography - The Key to Covert Communication, (CRC Press, Boca Raton, 2012).

[5] MIT, *Compression Algorithms: Huffman and Lempel-Ziv-Welch (LZW)*, http://web.mit.edu/6.02/www/s2012/handouts/3.pdf, (2012).

[6] Hasmawati, A. M. Barmawi, *Modifikasi HeadStega Berdasarkan Penyisipan Karakter HeadStega Modification Based On Character Insertion*]. *Ind. Journal on Computing*, **2**, 79-90, (2017)