Modified Email Header Steganography Using LZW Compression Algorithm

Varian Caesar[1], Dr. Ir. Rinaldi Munir, M.T.[2], Dra. Harlili, M.Sc.[3]
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Indonesia
Email: variancaesar@gmail.com[1], rinaldi@informatika.org[2], harlili@informatika.org[3]

Abstract— **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. However, HeadStega has a low hiding capacity. An email is only capable of hiding 4 to 11 bits of secret message. This paper propose a new technique to improve the hiding capacity of HeadStega. 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- Email Header Steganography, Hiding Capacity, LZW Compression Algorithm, Noiseless Steganography.

I. 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 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 rest of this paper is organized as follows: Section II describes some theories for the proposed approach. Section III presents the proposed approach for improving HeadStega. Section IV presents about experimental results and discussion. Finally, Section V concludes this work with some future works.

II. 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.

A. HeadStega

In HeadStega, secret messages are hidden as email headers, such as recipient's email address. The message will be converted into binaries and grouped by a certain length. That binaries will then be converted to a letter represented them. For Example, binaries will be grouped by length of four and then converted into a letter based on the **Table 1** [3].

Table 1. Steganographic Code for four bits

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

Generating recipient's email address is done by taking the alphabet one by one and then searching the database for email that contains the same alphabet as the prefix. In the receiver side, the receiver will take the first letter of each email and convert it to binary form using the same table. That binaries then will be regrouped by the length of 8 to get the original message. For the rest of this paper, terms for converting binary into a letter will be called **Coding.**

B. LZW Compression Algorithm

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

Algorithm 1. LZW Compression Pseudocode

```
TABLE[0 to 255] <- ASCII CODE
STRING <- read input

while input not empty:
     SYMBOL <- read input
     if STRING + SYMBOL is in TABLE:
        STRING <- STRING + SYMBOL
     else:
        CODE <- CODE + TABLE[STRING]
        TABLE <- INSERT(STRING + SYMBOL)
        STRING <- SYMBOL
Output <- CODE
```

Algorithm 2. LZW Decompression Pseudocode

```
TABLE[0 to 255] <- ASCII CODE
TEXT = " "
CODE <- read symbol
STRING <- TABLE[CODE]
TEXT <- TEXT + STRING

while input is not empty:
    CODE = read next symbol
    if  TABLE[CODE] is not exist:
        ENTRY <- STRING + STRING[0]
    else:
        ENTRY <- TABLE[CODE]
    TEXT <- TEXT + ENTRY
    TABLE <- STRING + ENTRY[0]
    STRING <- ENTRY
Output <- TEXT
```

III. PROPOSED METHOD

This Section presents the proposed method to improve the hiding capacity of HeadStega. The proposed method consists of two major components:

- Cover Generator
- Message Extractor

A. Cover Generator

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

First, string table will be used to compress secret messages. The secret message is a list of integers now. Next is the email-generating process, the concept of domain mapping table will be introduced here. Domain mapping table is simply 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

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

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 domain of gmail contains the secret message.

Cover generating process starts with compressing the message using LZW and converting it to list of binary. Then 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 into a letter. 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. The cover-generating process is shown in **Algorithm 3**.

Algorithm 3. Cover Generating Process

```
1. Compress the message m into compressed
   form, m2
2. Convert m2 to binary form, m3, ex:
   ['00001101', '00001111']
3. set i = 0 and half = False,

while i < length(m3):
   if length(m3[i]) > 8:
     name <- query database for random name
     email <- name + 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
4. Return result
```

B. 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 0, then we know that the secret message only contained in the numeric part of the username. If *k* is not 0, 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. Message Extracting is shown in **Algorithm 4**.

Algorithm 4. Message Extraction Process

```
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 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)
```

C. Proposed Method Example

This subsection provides an end-to-end example on how the proposed method works. Let us define the secret message is "ICAICTA". First, this secret message will be compressed by LZW into a series of integer:

73, 67, 65, 256, 84, 65

Next, this list of integer will be converted into list of binary:

'01001001', '01000011', '01000001', '100000000', '01010100', '01000001'

Next, **Table 3** will shows the cover generating process. This process gives us three recipient's email addresses: ujuanda@msn.com, debaji256@msn.com and feurzieg@gmail.com.

Table 3. Cover Generating Process

Binary	Prefix	Email generated	Note
0100	u	-	DB match
1001	uj	-	DB match
0100	uju	-	DB match
0011	ujud or ujut	ujuanda@msn.com	DB failed
0011	d	-	DB match
0100	de	-	DB match
0001	deb	-	DB match
100000000	deb	debaji256@msn.com	Value larger than 255
0101	f	-	DB match
0100	fe	-	DBmatch
0100	feu	-	DB match
0001	feur	feurzeig@gmail.com	Stop

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.

Table 4. Message Extracting Process

Email	K	Message
ujuanda@msn.com	3	uju
debaji256@msn.com	3	deb, 256
feurzeig@gmail.com	4	feur

For every letter in the message, the algorithm will convert it into list of binary based on **Table 1** in Section II. For numeric value, that value will be converted into binary. So the resulting binaries from the message extracted are:

'01001001', '01000011', '01000001', '100000000', '01010100', '01000001'

If we convert it to integer, it will gives the following result:

73, 67, 65, 256, 84, 65

Decompressing it, will gives the original message, 'ICAICTA'.

IV. Results And Discussions

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. File 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_iamge_64.ico	16958
Code-icns_32.icns	3294
Perc13.wav	10504

Experiment conducted in Windows 10 Laptop, 8 GB RAM, AMD A10-5750M and I 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. Experimental Result

File	Emails generated		Efficiency (%)
	Head-Stega	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_iamge_64.ico	12334	1326	89.24
Code-icns_32.icns	2396	547	77.17
Perc13.wav	7640	4863	36.34
Mean			58.80

A. *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.

B. *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.

V. Conclusion and Future Works

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 contains 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] Welch, T. A. A Technique for High-Performance Data Compression. IEEE, 1984.

[2] Cox, I. J., Miller, M. L., Bloom J. A., Fridrich, J., & Kalker, T. Digital Watermarking and Steganography. United States, USA : McGraw-Hill, 2008.

[3] Abdelrahman Desoky. Headstega: e-mail-headers-based steganography methodology. International Journal Electronic Security and Digital Forensics,, 3(4):289_310, 2010.

[4] Desoky, A. Noiseless Steganography : The Key to Covert Communications. North West, NW:CRC Press, 2012.

[5] Verghese, G. Compression Algorithm: Huffman and Lempel-Ziv-Welch (LZW). lecture notes. Digital Communication System EECS II Massachusetts Institute of Technology, 2012.

[6] Hasmawati, & Barmawi, A. M. HeadStega Modification Based On Character Insertion [Modifikasi HeadStega Berdasarkan Penyisipan Karakter]. Ind. Journal on Computing Volume 2, Issue 1, 2017.