Jurnal Ilmu Komputer dan Informasi (Journal of Computer Science and Information)
14/2 (2021), 113-126. DOI: http://dx.doi.org/10.21609/jiki.v14i2.957

Increasing The Capacity of Headstega Based on Bitwise Operation

Hasmawati, Ari Moesriami Barmawi

[1,2] Department of Informatics, Faculty of Informatics, Telkom University, Jl. Telekomunikasi No. 1 Terusan Buah Batu, Bandung, 40257, Indonesia

[1] *hasmawati@telkomuniversity.ac.id,* [2]*mbarmawi@melsa.net.id*

Abstract

Headstega (Head steganography) is a noiseless steganography that used email headers as a cover for concealing messages. However, it has less embedding capacity and it raises suspicion. For overcoming the problem, bitwise operation is proposed. In the proposed method, the message was embedded into the cover by converting the message and the cover into binary representation based on a mapping table that was already known by the sender and the receiver. Furthermore, XOR bitwise operations were applied to the secret message and cover bits based on random numbers that were generated using a modular function. Moreover, the result was converted into characters that represent the secret message bits. After embedding the message into the cover, an email alias was generated to camouflage the secret message characters. Finally, the sender sends the embedded cover and the email alias to the recipient. Using the proposed method, the embedding capacity is 89% larger than using the original Headstega. For reducing the adversary's suspicion, the existing email address was used instead of creating a new email address.

Keywords: *Noiseless Steganography, Headstega, XOR bitwise operation*

1. Introduction

Recently, the use of communication and information exchange are increasing. Since important information should be secured during information exchange, security against cheaters or eavesdroppers is necessary [1]. One method for securing information is steganography [2]. There are several types of steganography based on the covers used for camouflaging secret messages such as audio steganography [3], text steganography [4, 5, 6], image steganography [7, 8], and video steganography [9, 10]. Steganography can be classified as noisy [7, 9] and noiseless [11, 12]. In this research, noiseless steganography is observed. One of noiseless steganography is Headstega proposed by Desoky. Headstega is a steganography that camouflages data in an email header [12].

The drawback of Headstega is that the less embedding capacity and it raised suspicion since for embedding a message, a new email address is necessary. The capacity is low because the secret message was embedded into the first alphabetical character of the email address and also into numerical character in the email address. In other words, 4 bits of the secret message are embedded into the first alphabetical character of the email address while the next 7 bits are embedded in the numerical character, such that this method conceals 4 - 11 bits in each email address.

For overcoming the drawback, bitwise operation for embedding the message into the cover was proposed. In this case, the cover was an existing email address. The basic idea of the method was embedding a message into the cover based on mapping table. The embedding process was done by converting the message and the cover into binary representation based on a mapping table that was already known by the sender and the recipient. Furthermore, bitwise operations were performed on these bits based on random number that was generated using a modular function. After embedding the message into the cover, the binary numbers that represent the secret message were converted into characters and then email alias was generated based on these characters to camouflage the secret message characters. Finally, the sender sent the embedded cover and the email alias to the recipient.

The experiments were performed using 30 different covers and 30 messages of various lengths. The results of the experiment showed that the proposed method has a higher embedding capacity than the previous one. The proposed method can embed messages 89% larger than the previous method. Moreover, the previous method required the cover 95% higher than the proposed

method when observed in terms of the number of covers used. For reducing the adversary's suspicion, the existing email address is used. Thus, a new email address generation process is not necessary.

2. Related Work

Headstega is a method in Noiseless Steganography area or Nostega that camouflages messages in a part of email header such as email address, subject, cc, etc [11, 12]. Headstega consists of two main methods, encoding, and decoding. The encoding method consists of message encoding and message camouflaging. The architecture of the Headstega is shown in Fig. 1.

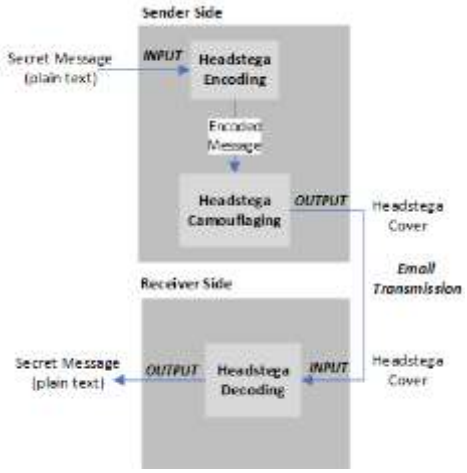


Fig. 1. The architecture of Headstega [12]

Message encoding is a method for encoding a message into an appropriate form based on the encoding parameters and steganographic coding map that was predefined. The secret message is converted and grouped into a certain length of a binary number. Furthermore, the binary representation is converted into a letter based on the coding map for steganography. The encoded message is camouflaged to generate the stego cover (email header) which conceals that encoded message. For example, according to the coding map in Table 1 [12], four bits slice "0111" represented the letter "H or X". Thus, the first characters of the email address that will be generated for embedding 0111 are letter H or X.

Message decoding is used for extracting the encoded message from the email address. For example, the receiver receives an email which consists of a group of receivers whose email addresses are "Hasmawati@test.xyz", "Erisdian@test.xyz". The secret message is extracted from these email addresses based on the same coding map used in message encoding method. Since the first character of the first email address consists of letter "H", then, "0111" is extracted. For the second email address "Erisdian@test.xyz", "E" is found as the first letter, then the binary number extracted from E based on the code mapping table is "0100". If those binary codes are concatenated, then "01110100" is obtained. Since this binary representation of the encoded message is equal to 116 in ASCII code, then it will be translated into the original message 't'.

Table 1. Headstega code mapping table[12]

No	Binary	Letters	No	Binary	Letters
1	0000	A or Q	9	1000	I or Y
2	0001	B or R	10	1001	J or Z
3	0010	C or S	11	1010	K
4	0011	D or T	12	1011	L
5	0100	E or U	13	1100	M
6	0101	F or V	14	1101	N
7	0110	G or W	15	1110	O
8	0111	H or X	16	1111	P

3. Proposed Method

For increasing the capacity of Desoky's method [12] as has been discussed in section 1, the bitwise operation was introduced in the proposed method. The architecture of the proposed method is shown in Fig. 2.

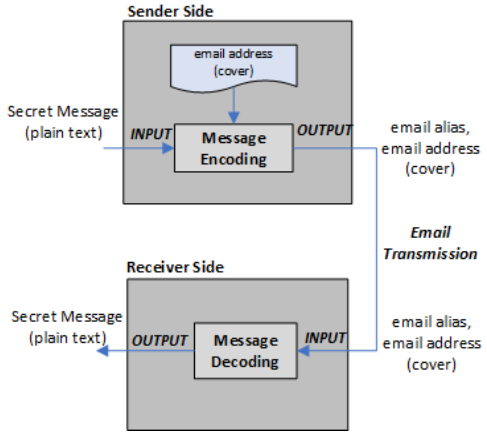


Fig. 2. The architecture of the proposed method

The main difference between Desoky's and the proposed method is that the cover generation was not necessary in the proposed method because the existing email address can be used as the cover.

Suppose Alice sent a secret message to Bob by email, then Alice embedded the secret message into the email address. For embedding the secret message, Alice needed a code mapping table such that Bob could extract the message using the same code mapping table. Therefore, the code mapping

table should be agreed upon in advance by Bob and Alice. The embedded message was represented as a sequence of characters. For extracting the message, Bob used the code mapping table and the characters to obtain the secret message.

The proposed method consisted of two main processes, encoding, and decoding. The sender conducted message encoding by embedding the secret message into a cover (existing email address). The receiver conducted message decoding by extracting the secret message from the cover.

3. 1. Message Encoding Process

The basic concept of message encoding was converting the secret message from letters into five bits binary codes based on the mapping table (table 2) that was already known by both parties. All five bits binary codes were concatenated to obtain the message bit stream. After converting the secret message into the message bits stream, the email address was also converted into five bits binary codes and concatenated all email addresses to obtain the cover bits stream.

The next process was XOR-ing the secret message and the cover bits stream. For increasing the security, the bit location where the XOR process starting point should be randomized, such that it was not easy for the attacker to find the location of the embedded secret message. For indicating the location of the embedded secret message, the random number generation was necessary. The random number was calculated based on the modulo function of the number of characters used in the mapping table, the time when the email was sent, the length of the secret message, and the length of the cover.

Finally, the result of the XOR process were encoded into a character based on Table 2 and then generated email alias to camouflage the secret message characters. The overview of message encoding is depicted in Fig. 3.

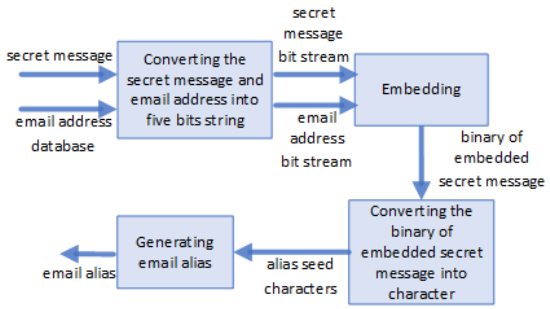


Fig. 3. Message encoding process

3. 2. Converting the Secret Message and the Cover into Five Bits String

The objective of this algorithm was to convert the secret message and the email address into a binary numbers. The conversion process consists of the following steps.

1. Each character of the email address and secret message were converted into five bits binary code based on Table 2.
2. Concatenating all five bits binary codes of the secret message to represent the secret message in binary form. A similar process was applied on the cover.

The algorithm for converting the secret message and the cover into five bits binary codes is depicted in Fig. 4.

Table 2. Character to five bits string mapping table

Char	Binary	Char	Binary	Char	Binary
a	00001	l	01100	w	10111
b	00010	m	01101	x	11000
c	00011	n	01110	y	11001
d	00100	o	01111	z	11010
e	00101	p	10000	comma	11011
f	00110	q	10001	period	11100
g	00111	r	10010	space	11101
h	01000	s	10011	sign @	11110
i	01001	t	10100	underscore	11111
j	01010	u	10101	question mark (?)	00000
k	01011	v	10110		

The character to five bits string mapping table as shown in Table 2 consisted of 26 alphabetic characters and some special characters. These special characters were selected based on the occurrence frequency of the specific characters in email addresses and words in Indonesian language. Generally, email addresses consist of a sequence of alphabetic characters and some special characters such as commas, periods, spaces, @ signs, underscores, and question marks (?), as well as words in Indonesian language. The output of this algorithm was the binary form of secret messages and covers.

```
Algorithm 1: Converting the secret message and cover into five bits string
Input : email address (cover), secret message
Output : cover bit stream and secret message bit stream

cbs ← empty string
for i 0 to len(cover) do
 cbs←cbs||conv_char2bin(cover[i])
endfor

smb ← empty string
for i  0 to len(sm) do
 smb←smb||conv_char2bin(sm[i])
endfor
return cbs, smb

note:
1. cbs : cover bit stream
2. smb : secret message bit stream
3. sm : secret message
```

Fig. 4. Algorithm for converting the secret message and the cover into five bits string

3. 3. Embedding Secret Message

The objective of this algorithm was to embed the secret message using the XOR operation. This process consists of the following steps :

1. Counting the length of secret message and cover bits stream
2. Determining the beginning location of the XOR process using equation (1)

$$i = l^u \ mod \ (x - y) \quad (1)$$

where i is the starting location of the XOR process, l is the number of characters used in the mapping table, u is the time when the email was sent, x is the length of the cover bit stream, and y is the length of the secret message bits stream. If $i + y > x$ then the XOR process for the remainder was started from the first character of the cover.

3. XOR the secret message bits with the cover bits starting from the location i that was obtained from step 2.

The output of this algorithm is the embedded secret message in binary form. The algorithm for embedding the secret message is shown in Fig. 5.

```
Algorithm 2: Embedding the secret message
Input : time, mapping_table, cover bit stream and secret message bit stream
Output : embedded secret message bit stream
i←len(mp)^t mod(len(cbs) – len(smb))
cstr←subStr(cbs, i, len(cbs))
esmb ← cstr XOR smb
return esmb
note :
 1. i : starting location of XOR process
 2. t : time the email was sent
 3. cstr : cover substring
 4. mp : mapping table
 5. cbs : cover bit stream
 6. smb : secret message bit stream
 7. esmb: embedded secret message bit stream
```

Fig. 5. Algorithm for embedding the secret message

3. 4. Converting the Binary Form of Embedded Secret Message into Character

The objective of this algorithm was for converting the binary of embedded secret message into the seed of alias by converting it into a character based on Table 2. The process for converting the binary of embedded secret message into character consists of the following steps:

1. Slicing the binary of embedded secret message into a group of five bits.
2. Each five bits string was converted into a character based on Table 2. This character is the alias seed character.

The output of this algorithm was the alias seed characters. The algorithm for converting the binary of embedded secret message into character is shown in Fig. 6.

3. 5. Generating Email Alias

The objective of this algorithm was to generate an email alias representing the character that was sent to the recipient. The process for generating alias consists of the following steps.

1. Determining the rules for generating the email alias using equation (2)

$$r = l^u \ mod \ x \quad (2)$$

where r is the result of the modular function (the number for choosing the rule used to construct the email alias), l is the number of characters used in the mapping table, u

is the time when the email was sent, x is the length of the cover bit stream.

2. If the result of equation 2 is even, then generate the email alias based on rule A, where rules A is as follows:
   1) If the first letter of the alias seed characters was a vowel, then added one consonant after a vowel
   2) If the first letter of the alias seed characters was a consonant, then added one vowel after a consonant,
   3) For the second until the last character of the alias seed characters, added one vowel before a consonant or one consonant before a vowel
   4) For special characters, an additional character is not necessary
3. If the result of equation 2 is odd then generate the email alias based on rule B, as follows:
   1) If the first letter of the alias seed characters was a vowel, then added one consonant before a vowel.
   2) If the first letter of the alias seed characters was a consonant, then added one vowel before a consonant.
   3) For the second until the last character, added one vowel after a consonant or one consonant after a vowel
   4) For special characters, an additional character was not necessary

The output of this algorithm was the email alias that will be sent to the recipient. The algorithm for generating email alias is shown in Fig. 7.

```
Algorithm 3: Converting the binary of embedded secret message into character
Input : embedded secret message bit stream
Output : alias seed character
sb ← slice esmb into five bit
as ← empty string
for i ← 0 to len(sb) do
  as ← as || conv_bin2char(sb[i])
endfor
return as

note:
   1. sb: sliced bit
   2. esmb: embedded secret
      message bit stream
   3. as : alias_seed_character
```

Fig. 6. Algorithm for converting the binary of embedded secret message into character

```
Algorithm 4: Generating Email Alias
Input : alias seed character, time,
cover bit, mapping_table
Output : email alias character

r  <- len(mp) ^ t mod len(cb)
 if r % 2 == 0 then
   if as[0] is v then
     ea ← as[0] + random c
   else if as[0] in c then
     ea ← as[0] + random v
   else
     ea←as[0]
 endif
 for i  1 to len(as) do
   if as[i] is v then
     ea← ea + random c + as[i]
   else if as[i] is c then
     ea←ea + random v + as[i]
   else
     ea ← ea + as[i]
   endif
 endfor
 else
   if as[0] is v then
     ea ← random c + as[0]
   else if as[0] is c then
     ea ← random v + as[0]
   else
     ea ← as[0]
   endif
   for i  1 to len(as) do
     if as[i] is v then
       ea ← ea + as[i] + random c
     else if as[i] is c then
       ea ← ea + as[i] + random v
     else
       ea ← ea + as[i]
     endif
   endfor
 endif
 return ea

note :
   1. mp : mapping table
   2. t : time the email was sent
   3. cb : cover bit stream
   4. as : alias_seed_character
   5. ea : email_alias_character
   6. v : vowel
   7. c : consonant
```

Fig. 7. Algorithm for generating the email alias

3. 6. Implementation of Message Encoding

For implementing the message encoding, the secret message length must be less than the cover length.

1. Suppose the secret message is 'waspada', the cover is'hasmawati@telkomuniversity.ac.id' and the time when the email was sent is 01:26.
2. Converting 'waspada' into five bits representation, such as w=10111, a=00001, s=10011, p=10000, a=00001, d=00100, a=00001 then concatenated all five bits of the character representation. The concatenation result is '10111000011001110000000010010000001'.
3. Converting'hasmawati@telkomuniversity.ac.id'into five bits binary and concatenated them. The results is '010000000110011011010000110111000011010001001111101010000010101100010110111110110110101011100100110110001011001010011010011010011000111000000100011111100010010010000'.
4. Counting the length of the secret message bits stream. The length of '10111000011001110000000010010000001' is 35.
5. Counting the bit length of the cover resulted from step 3 which is 160.
6. Determining the starting location of the XOR process by calculating $i = l^u \ mod \ (x - y) = 32^{126} \ mod \ (160 - 35)$. The result is 57.
7. XOR the secret message bits stream '10111000011001110000000010010000001' with the cover bits stream '0100000001100110110100001101110000110100010011111101010000101011000101101111011011010101110010011011000101100101001101001101001100111100000010001111100010010010' starting from bit 100. The result of the XOR process is '0001010000111100110110111000110000' which is called as embedded secret message.
8. Slicing '0001010000111100110110111000111000' into group of five bits where the result is '00010', '10000', '11110', '01101', '10111', '10001', '11000'.
9. Converting each five bits code into character based on Table 2 such as '00010 = b', '10000 = p', '11110 = @', '01101 = m', '10111 = w', '10001 = q', '11000 = x'. The result is bp@mwqx.
10. Generating email alias based on Algorithm 4 in Figure 7. Since the result of modular function using equation 2 is even then the generating process was performed using the algorithm in rules A. The result is boip@amiwaqux.

3. 7. Message Decoding Process

The basic concept of message decoding was extracting the secret message from the stego email header. The message decoding process started by extracting the received email alias to obtain the alias seed characters. Furthermore, all characters should be converted into five bits binary code to obtain the binary string of alias seed characters. Similarly, the email address was also converted into five bits binary code to obtain the cover bit stream. The next process was extracting the binary string of secret message by performing the XOR operation between the alias seed character bit stream and the cover bits stream. Before the XOR-ing process, random number generation should be conducted to indicate the beginning location of the XOR-ing process. The result of the XOR-ing process were sliced into five bits binary codes. Furthermore, the five bits binary codes were converted into character. The overview of the message decoding process is shown in Fig. 8.

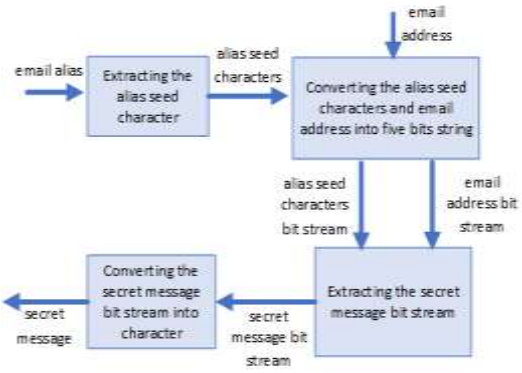


Fig. 8. Message decoding process

3. 8. Extracting the Alias Seed Characters

The objective of this algorithm was to extract the alias seed characters from the email alias received by the receiver. The extracting process consists of the following steps.

1. Determining the rule for extracting the alias seed characters using equation (2).
2. If the result of equation 2 is even, then extract the alias seed characters based on rules A, where rule A is as follows:

1) Set index of the email alias with $i = 0$, set index of the alias seed with $j = 0$. In this case, the i^{th} character of email alias is represented by $email\_alias\_character[i]$ and the j^{th} character of alias seed is represented by $alias\_seed\_character[j]$.
2) $Email\_alias\_character[i]$ is stored in $alias\_seed\_character[j]$, then increment *j* by 1.
3) If $i = 0$, then check whether $email\_alias\_character\ [i]$ is a special character or not.
   - If $email\_alias\_character\ [i]$ is a special character, then check the $email\_alias\_character\ [i + 1]$. If it is a special character, then increment i by 1, otherwise increment *i* by 2.
   - If $email\_alias\_character\ [i]$ is not a special character, then check the $email\_alias\_character\ [i + 2]$. If it's a special character, then increment i by 2, otherwise increment *i* by 3.

   If $i! = 0$ and $email\_alias\_character\ [i + 1]$ is a special character then increment *i* by 1, otherwise increment *i* by 2.
4) Repeat steps 2) and 3) until the last character of the email alias.

3. If the result of equation 2 is odd, then extract the alias seed characters based on rules B, where rule B is as follows:
   1) Set index of the email alias with $i = 0$, set index of the alias seed with $j = 0$
   2) If $email\_alias\_character\ [i]$ is a special character, then store $email\_alias\_character[i]$ into $alias\_seed\_character[j]$, increment *j* by 1 and increment *i* by 1. If it is not a special character, then check if $i = 0$, then store $email\_alias\_character[i + 1]$ into $alias\_seed\_character[j]$, increment *j* by 1 and increment *i* by 2. Otherwise if $i! = 0$, then store $email\_alias\_character[i]$ into $alias\_seed\_character[j]$, then increment *j* by 1 and increment *i* by 2.
   3) Repeating step 2) until the last character of the email alias

The algorithm for extracting the alias seed characters is shown in Fig. 9.

```
Algorithm 5 : Extracting the email
alias
Input : email alias, time, cover
bit stream, mapping table
Output : alias seed character
r  <- len(mp) ^ t mod len(cb)
if r % 2 == 0 then
 i← 0;   j← 0;   as← []
 while i < len(ea) do
   as[j]← ea[i];  j← j+1
   if i==0 then
     if ea[i] is sc then
       if ea[i+1] is sc then
         i← i+1
       else
         i← i+2
       endif
      else
         if ea[i+2] is sc then
           i← i+2
         else
           i← i+3
         endif
       endif
    else
      if ea[i+1] is sc then
         i← i+1
      else
         i← i+2
      endif
    endif
  endwhile
else
 i← 0;   j← 0;   as← []
 while i < len(ea) do
  if ea[i] is sc then
   as[j]←ea[i]; j←j + 1; i←i + 1
  else
   if i==0 then
    as[j]←ea[i+1];j←j + 1;i←i+ 2
   else
    as[j]←ea[i];j←j + 1; i←i + 2
       endif
    endif
  endwhile
endif
return as
note :
   1.  t : time the email was
   sent;
   2.  cb : cover bit stream;
   3.  as : alias_seed_character;
   4.  ea : email_alias_character;
   5.  v: vowel; c : consonant;
   6.  mp : mapping table;
```

Fig. 9. Algorithm for extracting the alias seed characters

3. 9. Converting the Alias Seed Characters and the Cover into Five Bits String

The objective of this algorithm was to convert the alias seed characters and the cover into five bits binary code. The conversion process consists of the following steps.

1. Each character of the alias seed is converted into five bits binary code based on Table 2.
2. Concatenating all five bits binary code of the alias seed characters to represent the alias seed characters bit stream.
3. Step 1 and 2 are also applied to the cover.

The output of this algorithm was the cover bits stream and the alias seed character bit stream. The algorithm for converting the character and the cover into five bits binary code is shown in Fig. 10.

```
Algorithm 6: Converting the alias seed characters and cover into five bits string
Input : alias seed character, email address(cover)
Output :  alias seed bit stream, cover bit stream

bas ← empty string
for i ← 0 to len(asc) do
 bas←bas||conv_char2bin(asc[i])
endfor

bc ← empty string
for i ← 0 to len(cover) do
 bc← bc || conv_char2bin(cover[i])
endfor
return bas, bc

note :
   1. bas : alias seed bit stream
   2. asc : alias_seed_character
   3. bc: cover bit stream
```

Fig. 10. Algorithm for converting the alias seed characters and the cover into five bits string

3. 10. Extracting the Secret Message

The objective of the algorithm was to extract the secret message using the XOR operation. The process for extracting the secret message consists of the following steps.

1. Counting the length of the embedded secret message bit and cover bits stream.
2. Determining the beginning location of the XOR process using equation (3)

$$i = l^u \ mod \ (x - z) \quad (3)$$

where i is the starting location of the XOR process, l is the number of character on the mapping table, u is the time when the email was sent, x is the length of the cover bit stream, and z is the length of the alias seed character bit stream. If $i + z > x$ then the XOR process for the remainder is started from the first character of the cover.

3. XOR the embedded secret message bits with the cover bits starting from the location i that is obtained from step 2.

The output of this algorithm was the secret message bits stream. The algorithm for extracting the secret message is shown in Fig. 11.

```
Algorithm 7: Extracting the secret message
Input : time, mapping_table, cover bit stream and alias seed bit stream
Output : secret message bit stream

i←len(mp)^t mod(len(bc) – len(bas))
cstr ← subString(bc, i, len(bc))
smb ← cstr XOR bas
return smb

note :
   1. i : starting location of XOR process
   2. t : time the email was sent
   3. bc : cover bit stream
   4. bas : alias seed bit stream
   5. cstr : cover substring
   6. mp : mapping table
   7. smb : secret message bit stream
```

Fig. 11. Algorithm for extracting the secret message

3. 11. Converting the Secret Message Bits Stream into Characters

The objective of this algorithm was to convert the secret message bits stream into characters. The process for converting the secret message bits stream into characters consists of the following steps:

1. Slicing the secret message bits stream into a group of five bits.
2. Each group is converted into a character based on Table 2.

The algorithm for converting the secret message bits stream into character is shown in Fig. 12.

```
Algorithm 8: Converting the secret
message bit stream into character
Input: secret message bit stream
Output: secret message

sb ← slice smb into five bit
sm ← empty string

for i ← 0 to len(sb) do
  sm ← sm || conv_bin2char(sb[i])
endfor
return sm

note :
   1. sb: sliced bit
   2. smb : secret message bit
   3. sm : secret message
```

Fig. 12. Algorithm for converting the secret message bits stream into character

3. 12. Implementation of Message Decoding

Suppose the email alias was 'boip@amiwaqux', the time when the email was sent is 01:26, the cover was 'hasmawati@telkomuniversity.ac.id' then, for extracting the secret message from alias and email address, the following process should be conducted.

1. Extracting the alias seed characters from the email alias using Algorithm 5. Since the result of modular function using equation 2 was even, then the extracting process is performed based on rules A. The alias seed characters obtained was bp@mwqx.
2. Converting 'bp@mwqx' into five bits binary code based on Table 2, the five binary code of each character are b = 00010, p = 10000, @ = 11110, m = 01101, w = 10111, q = 10001, x = 11000. Furthermore, all five bits code should be concatenated to obtain the alias seed character bit stream which was 000101000011110011011011110001111000.
3. Step 2 was also applied on the cover to obtain the cover bits stream which is '0100000001100110110100001101110000 1101000100111110101000010101100010 1011110110110101011100100110110001 0 1100101001101001101001'001111000000 100011111000100100100' with a length of 160.
4. Determining the starting location of the XOR process by calculating $i = l^u \ mod \ (x - z)$ where i is the starting location of the XOR-ing process, l is the number of character on the mapping table, u is the time when the email was sent, x is the length of the cover bit stream, and z is the length of the alias seed character bit stream. The result is 57.
5. Extracting the embedded secret message by performing the XOR operation between the alias seed character bit stream '000101000011110011011011110001110 0' with the cover bits stream '0100000001100110110100001101110000 1101000100111110101000010101100010 1011110110110101011100100110110001 0 1100101001101001101001'001111000000 100011111000100100100' starting from bit 100. The result of the XOR process is '1011100001100111000000001001000000 1' which is called secret message bit stream.
6. Slicing '1011100001100111000000001001000000 1' into groups of five bits where the result is '10111', '00001', '10011', '10000', '00001', '00100', '00001'.
7. Converting each five bits code into character based on Table 2. The character obtained are 10111='w', 00001='a', 10011='s', 10000='p', 00001='a', 00100='d', 00001='a'. Finally, 'waspada' is obtained as the result.

4. Experiment and Analysis

To evaluate the embedding capacity of the proposed and the Desoky's methods, two experiments were conducted. The first experiment was to calculate the number of covers used for embedding the message, and the second was to calculate the embedding capacity of the proposed and Desoky's method. Both of experiments were conducted using 30 covers and 30 messages of various lengths.

For conducting the experiment using Desoky's method the secret message were divided into two sentences. The first sentence was embedded in the first letter of the email address. The second sentence was embedded in the numeric characters of the email address. The cover used was an email address taken from a valid email address databases which is about 5,307 email addresses.

4.1. Evaluating Cover's Size For Embedding Messages

The objective of the experiment for evaluating the cover's size for embedding messages was to evaluate the number of covers required for embedding a message using the previous and the

proposed method. The experiment result shows that to embed a message with a length of n, using the previous method required 5% of email addresses larger than the proposed method.

This condition was occurred because using Desoky's method, one email address could be embedded with a maximum of 11 bits of the secret message (by assuming that the character represented by 4 bits on the message were matched with the first character of the email address, such that these 4 bits could be embedded into the first character of the email address, and the numeric character represented by 7 bits on the message were matched with the three numeric characters of the email address such that these 7 bits could be embedded into the numeric character of the email address). Meanwhile, using the proposed method one email address could be embedded with a minimum of 16 bits (by assuming that one email address could be embedded with a minimum of two characters or 16 bits). The result of the experiment for evaluating the cover's size for embedding messages is shown in Table 3.

4.2. Evaluating of the Embedding Capacity

The objective of the experiment for evaluating the embedding capacity was to evaluate the number of secret message characters that can be embedded into the cover using Desoky's method and the proposed method. In this case, the cover used for both methods was similar. The experiment was conducted using 30 covers and 30 messages of various lengths. Since some covers could be embedded with a similar length of the secret message, then these covers were classified into 10 classes.

The result of the experiment shows that the number of characters that can be embedded using the proposed method was larger than using Desoky's method. In general, the proposed method could embed messages 89% larger than the Desoky's method.

Since the embedding capacity depended on the maximum number of bits that could be embedded into a cover divided by the number of bits of the cover, then the maximum capacity was calculated using equation 4.

$$capacity = \frac{(n)}{k} x\ 100\% \quad (4)$$

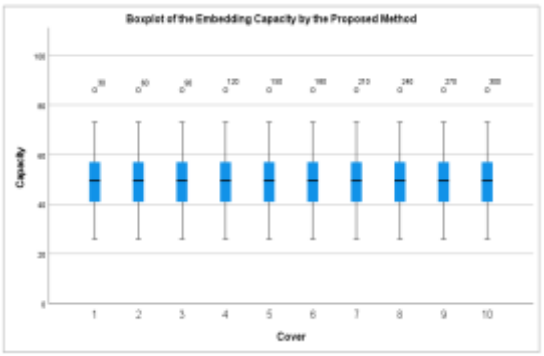
where n is the maximum number of bits that can be embedded into the cover, and k is the number of bits of the cover which can be calculated using equation (5).

$$k = \sum_{i=1}^{n} c_i \quad (5)$$

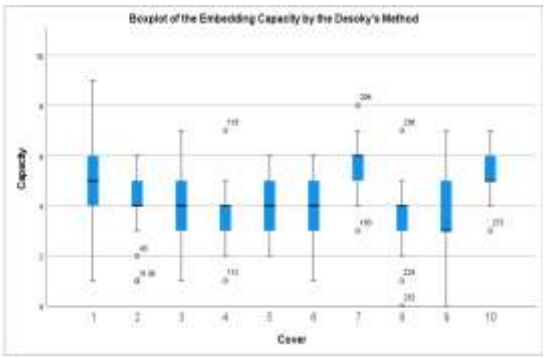
Using Desoky's method, one email address could be maximum embedded by 11 bits of message, then the maximum embedding capacity was 11 bits, while using the proposed method, one message could be maximum embedded by k bits. Thus, the embedding capacity using Desoky's method was $11/k$ % while using the proposed method is 100%.

Since the number of characters in one email address was larger than or equal to two characters, then using the proposed method, the minimum capacity was $(10/k) * 100\%$. Meanwhile, using Desoky's method, the minimum capacity was $\left(\frac{4}{k}\right) * 100\%$. Thus, it was shown that the maximum embedding capacity when using Desoky's method was less than the minimum embedding capacity when using the proposed method. Therefore, it can be concluded that the embedding capacity using the proposed method was larger than Desoky's method.

The results of the experiment for evaluating the embedding capacity is shown in Fig. 13.



(a)



(b)

Fig. 13. Comparison of the embedding capacity using (a) the proposed and (b) desoky's method

Table 3. Comparison of cover used using the proposed and desoky's method

NO	(1)	(2)	(3)	(4)	(5)	(6)	(7)
1	rahasia	7	56	h3ndra_gun82@yahoo.com;ca94110@gmail.com;gadismode@yahoo.com;baektex@hanmail.net;gaojian@fortune-oil.com.cn;ibon_02@yahoo.com;garuda@billmelk.com;bahanaputra@yahoo.com;halliewong@hotmail.com;dada.sandramargaretha@gmail.com;gede_mahatma@yahoo.com;j.charleston@chello.nl;general@btcocoa.com;baligm@cti_usa.com;	bahanaputra@yahoo.com	14	19
2	telpon bu si	12	96	h3ndra_gun82@yahoo.com;e.dolman@quicknet.nl;gadismode@yahoo.com;fahrulsani@gmail.com;gaojian@fortune-oil.com.cn;maetrading@bdg.centrin.net.id;halliewong@hotmail.com;a183rt_santoso@yahoo.com;garuda@billmelk.com;pangestutis@yahoo.com;gede_mahatma@yahoo.com;ohayotravel@hotmail.com; aldhy16@yahoo.com;satomai24gianyar@hotmail.com;ssongsam78@hanmail.net;h3ndra_gun82@yahoo.com;dlucs_pg3@yahoo.com;asep.setiawan77@yahoo.com;ahalim82@gmail.com;alfebad53@yahoo.com;adi_s88@hotmail.com;be_tours1@yahoo.com;	e.dolman@quicknet.nl	30	20
3	ubah hak akses service	22	176	h3ndra_gun82@yahoo.com;fahrulsani@gmail.com;gadismode@yahoo.com;ca94110@gmail.com;gaojian@fortune-oil.com.cn;baektex@hanmail.net;garuda@billmelk.com;ibon_02@yahoo.com;camat169@gmail.com;a183rt_santoso@yahoo.com;gede_mahatma@yahoo.com;idabsa@yahoo.com;general@btcocoa.com;bahanaputra@yahoo.com;general@mincom.co.id;l.yamin@cgiar.org;,u-aldhy16@yahoo.com;satomai24gianyar@hotmail.com;azzy45@gmail.com;achmadi55@gmail.com;kenichi.megumi27@yahoo.com;ferimon21@gmail.com;afekn102@oct.zaq.ne.jp;dannyang32@yahoo.com;basith57@yahoo.com;rosyeni89@yahoo.com;andrayani46@gmail.com;ahmad.syafei39@yahoo.co.id;rukiman51@gmail.com;haruru37@nifty.com;plamet70@naver.com;happy_tour101@yahoo.com	baektex@hanmail.net;general@mincom.co.id	50	40
4	twonight@eightAMuse my secret key	34	272	h3ndra_gun82@yahoo.com;e.dolman@quicknet.nl;halliewong@hotmail.com;handhee@yahoo.com;gadismode@yahoo.com;pangestutis@yahoo.com;gaojian@fortune-oil.com.cn;ohayotravel@hotmail.com;garuda@billmelk.com;j.charleston@chello.nl;gede_mahatma@yahoo.com;handjasa@ymail.com;general@btcocoa.com;ibon_02@yahoo.com;hans.bouma@casema.nl;eddy-syam@exxonmobil.com;edy_noor@yahoo.com;a183rt_santoso@yahoo.com;general@mincom.co.id;fahrulsani@gmail.com;gleanos@dnet.net.id;j.elisabrth0204@yahoo.com;gloria@bumi.net.id;happyizzim@hanmail.net;gnp_b@hotmail.com;idabsa@yahoo.com;harpindojaya@yahoo.com;ehud_myer@yahoo.com;eis_234@yahoo.com;baektex@hanmail.net; ellysthamrin@gmail.com;nadirautama105@yahoo.com; aldhy16@yahoo.com;yul29@menlh.go.id;andrayani46@gmail.com;aanfarhan54@yahoo.com;aidanf41@hotmail.com;be_tours1@yahoo.com;bennywijaya90@gmail.com;chongwei121@gmail.com;dh4nie16@yahoo.com;jongjava_lk28@yahoo.com;halim108@yahoo.com;adisap86@yahoo.com;kenichi.megumi27@yahoo.com;wengwai73@yahoo.com.sg;deje74@gmail.com;bjr116@gmail.com;doddyputra16@gmail.com;amyhendo26@hotmail.com;pergas108@hotmail.com;liacute87@yahoo.com;afrini9@yahoo.com;	handjasa@ymail.com;baektex@hanmail.net	76	37

4.3. Imperceptibility

Imperceptibility is calculated/identified as the probability for an attacker to detect the existence of a hidden message in the cover message. The hidden message should be invisible and should not raise the suspicions of human vision systems. Practically, imperceptibility was evaluated by analyzing the degree of imperceptibility. The degree of imperceptibility was obtained by comparing the cover message and the cover message after a message was embedded into it. The difference between those two messages was calculated using Jarro distance [14,15,16] (see equation 6).

$$d_j = \begin{cases} 0 & if\ m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & else \end{cases} \quad (6)$$

where m is the number of matching characters between those two messages, s_1 is the length of the cover message, s_2 is the length of the cover message after the embedding process, and t is half the number of transpositions.

Since either Desoky's and the proposed method was a noiseless steganography, then there was no difference between the cover message and the cover after embedding process, or in other words, $m = s_1 = s_2, and\ t = 0$ (because of no transpositions). Thus, in this case, d was 0.

4.4. Robustness

During the transmission process, multiple attacks may occur on the stego-text, such as text manipulation and deletion parts of the stego-text, by attackers. A robust steganography method should have the capability to make it difficult for attackers to change or destroy a hidden message. This capability could be measured based on the probability of how much proportion of the embedded message that had been lost from the stego text, $P(L)$. Suppose the number of embedded locations in the cover message was NL and the length of the cover message was TC. Thus, the $P(L) = NL\ /\ TC$ and the $P(DR)$ can be calculated as follows [14,15,16]:

$$P(DR_{HM}) = [1 - P(L)]; 1 < NL < TC, NL \in N, TC \in N \quad (7)$$

Since using the proposed method all locations of the cover message could be embedded, then if all locations were used, then the loosing probability could be maximum 1. However, since the lower losing hidden message probability leads to a more robust steganography methods, then not all locations could be embedded. Suppose, 11 bits characters (11 bits is the maximum embedding capacity using Desoky's method) was embedded into the cover message using Desoky's method (7 bits were embedded into 3 number characters and 4 bits were embedded into one character), then the probability of losing the hidden message $P(L)_{Desoky} = 4/TC$, while using the proposed method $P(L)_{Proposed} = 3/TC$ (since one character could be embedded by 5 bits, then for embedding 11 bits, 3 characters were necessary). Thus, the loosing probability for embedding 11 bits using the proposed method was less than the loosing probability when using Desoky's method. Since $P(L)_{Desoky} > P(L)_{Proposed}$, then $P(DR)_{HM-Desoky} < P(DR)_{HM-Proposed\ method}$ or in other words the proposed method was more robust than Desoky's one.

4.5. Security Analysis

Since in Desoky's method the security only depends on the map used for transforming the character into the binary code, then the security evaluation was conducted based on the probability of obtaining the map. Suppose there are n characters that had to be map to n codes by assuming that each character had to be map to a unique code, then there were $n!$ possible maps.

Thus, if an attacker tried to attack Desoky's method, the probability for obtaining the correct map was $\frac{1}{n!}$. In the case of the proposed method, instead of only guessing the map, the alias guessing was also necessary. Since there were two possible alias types (rules A and B), then the probability for obtaining the correct map of the proposed method was $\frac{1}{n!*2}$.

Instead of evaluating the security based on statistics, we also evaluated the security based on the security model [17]. Since the hidden message (E) was not determined by the stego-text (S), then $H(E|S) = H(E)$ (where $H(E)$ is the uncertainty of E). Suppose the attacker knew S and Cs (source of cover), but the attacker could not obtain E, because the stego-text was noiseless which means there was no difference between the stego-text and the cover. Thus, the uncertainty about E if the knowledge of Cs and S was obtain, $H\big(E\big|(S, Cs)\big) = H(E)$ or it can be concluded that $H\big(E\big|(S, Cs)\big) = H(E|S) = H(E)$.

The proposed method used secret XOR starting location which was depended on the number of codes in the map, which was assigned as the secret K. Furthermore, based on [17], the uncertainty

about K if the knowledge of S and Cs is obtain, $H(K|(S,Cs) = H(E) + H(K|E)$.
The uncertainty about K if the knowledge of E was obtain, $H(K|E) = H(K)$. Since K was not determined by E, then $H(K|(S,Cs) = H(E) + H(K)$ which was greater than $H(E)$. Since, $H(E|(S,Cs)) = H(E|S) = H(E)$ and $H(K|(S,Cs) \geq H(E)$, then it was proven that the proposed method was information theoritically secure.

4.6. Time Complexity

Since our proposed method used a function to generate the secret XOR starting location based on modular exponentiation, then the time complexity of our proposed method is $O(2^n)$, while Desoky's method is $O(n)$.

5. Conclusion

The weakness of Headstega [12] is that it has less embedding capacity and it raises adversary's suspicion. Based on the experiment result, it is shown that the proposed method can improve the embedding capacity while preserving the security of Headstega by introducing the bitwise operation process. The result shows that the capacity of the proposed method is 89% larger than the previous method. Moreover, for embedding similar messages the previous method required 5% number of email address larger than using the proposed method. For reducing the adversary's suspicion, the existing email address is used instead of generating a new email address.

The imperceptibility of the proposed method is similar to the Desoky's one, while the robustness of the proposed method is better than Desoky's one. Furthermore, the security of the proposed method is better than Desoky's method.

References

[1] M. E. Whitman, and H. J. Mattord, (2011). "Introduction to Information Security". Principles of Information Security. p.1-38. 2011.

[2] W. Stallings, (2011). "Cryptography and Network Security". Pearson Education,. p.57. 2011.

[3] M. Anwar, M. Sarosa, R. Rohadi, "Audio Steganography Using Lifting Wavelet Transform and Dynamic Key". 2019 International Conference of Artificial Intelligence and Information Technology (ICAIIT). 2019

[4] F. H. Rabevohitra, and Y. Li, "Text Cover Steganography Using Font Color of the Invisible Characters and Optimized Primary Color-intensities". 2019 IEEE 19th International Conference on Communication Technology. 2019

[5] R. Lockwood and K. Curran, "Text based steganography," int J. Information Privacy, Security and Integrity, Vol. 3, No. 2, pp.134-153. 2017.

[6] G. Gustaman and A. M. Barmawi, "Increasing the Capacity of Listega Based on Syllable Pattern Using Multicolumn and Bigram Mapping", Conference: 2018 IEEE World Symposium on Communication Engineering (WSCE).2018.

[7] O. Elharrouss, N. Almaadeed, S. Al-Maadeed, "An image steganography approach based on k-least significant bits (k-LSB)". 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT). 2020.

[8] A. Darbani, M. M. AlyanNezhadi, and M. Forghani, "A New Steganography Method for Embedding Message in JPEG Images". 2019 IEEE 5th International Conference on Knowledge-Based Engineering and Innovation (KBEI). 2019.

[9] Y. Zhang, M. Zhang, K. Niu, J. Liu, "Video Steganography Algorithm Based on Trailing Coefficients". 2015 International Conference on Intelligent Networking and Collaborative Systems. 2015.

[10] S. Teotia, and P. Srivastava, "Enhancing Audio and Video Steganography Technique Using Hybrid Algorithm". International Conference on Communication and Signal Processing, April 3-5, 2018.

[11] A. Desoky, "Noiseless steganography: The key to covert communications". CRC Press, 2012.

[12] A. Desoky, "Headstega: e-mail-headers-based steganography methodology". International Journal Electronic Security and Digital Forensics, 3(4):289310, 2010.

[13] Schneier, Bruce. "Applied Cryptography. Second edition". John Wiley & Sons. p.535. 1996.

[14] M. T. Ahvanooey, Q. Li, J. Hou, H. D. Mazraeh, and J. Zhang, "AITSteg: An Innovative Text Steganography Technique for Hidden Transmission of Text Message via Social Media", IEEE Access, Vol.6, pp. 65981-65995, 2018.

[15] M. T. Ahvanooey, Q. Li , J. Hou, A. R. Rajput and C. Yini, "Modern Text Hiding, Text Steganalysis, and Applications: A

Comparative Analysis, Entropy", Vol.21, pp. 350-381, 2019.

[16] M. T. Ahvanooey, Q. Li , H. J. Shim, and Y. Huang, "A Comparative Analysis of Information Hiding Techniques for Copyright Protection of Text Document", Security and Communication Networks, Vol. 2018, pp.1-22, 2018.

[17] J. Zöllner, H. Federrath, H. Klimant, A. Pfitzmann, R. Piotraschke, A. Westfeld, G. Wicke, G. Wolf, "Modeling the security of steganographic systems", International Workshop on Information Hiding, pp 344-354, 1998.