WHITE SPACE STEGANOGRAPHY ON TEXT BY USING LZW-HUFFMAN DOUBLE COMPRESSION

Gelar Budiman and Ledya Novamizanti

Electrical Engineering Faculty, Telkom University, Bandung, Indonesia.

ABSTRACT

Privacy, especially in a cellphone, is an important thing and should be protected. Steganography is a method used to protect a sensitive information. The issue tried to be discussed in this study is the issue on inserting technique in a text through a simple method of White Space Steganography on android. The inserted message has been compressed through a double compression method by using LZW and Huffman so that the size of message to be inserted can be minimized while the capacity of the inserted message can be minimized. The compression shows that the compression ratio much depends on the type of text input to the text to be sent; the more the repetition or duplication found on the message, the smaller the compression ratio will be. The compression process using Android based smartphone is relatively fast with the average duration of 0.045 seconds, either for the insertion or extraction.

KEYWORDS

White Space Steganography, teks, android, compression, LZW, Huffman Coding

1.INTRODUCTION

Difficulties in protecting one's privacy is getting challenging along with the development in digital communication technology. One of the popular communication technologies that consistently develops is cellphone, particularly android. Cellphone is often used as a medium to save and send sensitive information. As cell phone is an accessible device, the protection towards sensitive information within the cellphone becomes essential.

One of the methods that can be taken as to protect the information is by using steganography, i.e. a technique to hide information by inserting a message into another message[4], so that it is only the intended person who will aware about the message. One of the recent studies on steganography is the one conducted by Ratna E. and V.K. Govindan, i.e. the unlimited payload steganography[4]. However, the study on LZW and Huffman compression has also been conducted, as that one of Linawati and Henry P. Panggabean[6]. Compression is usually conducted on the watermarking technique towards cover image after the inserting process is taken, as conducted by Dr. Ajit Singh and Meenakshi Gahlawat[9]. Meanwhile, all of the studies have not been implemented on android.

In this study, double compression will be conducted by using LZW-Huffman technique on textual data which is then hidden through white space steganography. The system will then be implemented on android. By using LZW and Huffman techniques on the compression process, it is hoped that the capacity of the hidden textual data can be bigger than the previous study.

DOI : 10.5121/ijcnc.2015.7210

2. THEORETICAL BACKGROUND

2.1 White Space Steganography

Steganography is derived from Greek–*Steganós* which means hide and *Graptos* which means writing - that in general it can be defined as hidden writing[7]. In general, steganography is an art and knowledge in hiding a message into a medium in a way that it is only the sender and the receiver who know or realize that there is actually a secret message[10].

Steganography in digital era has been much developed. The media that can be used for steganography are text, picture, audio, and video. One of the simple media that can be used is steganography on texts through White Space Steganography method. White Space Steganography is a simple steganography method by using "space" and "tab" characters as to show hidden message bits. "Space" and "tab" can be used since it is difficult to recognize and is not reflected in text viewer.

2.2 LZW Compression Method

The storage of big size data or files takes up a big storage capacity. Pertaining to this, compression technique can be used to minimize the data size. Compression is a process in encrypting a group of data into a code as to optimize the storage space, as well as the transmission time[3].

Lossless compression method is data compression method which can generate data identical to the original one, i.e. by reconstructing the compressed data[1]. One of the examples of lossless compression is by using LZW algorithm.

LZW (Lempel Zev Welch) algorithm is developed by using compression method developped by Ziv and Lempel in 1977. This algorithm carries out the lossless compression by using dictionary, where the text fragments are replaced by the index derived from a "dictionary". Character String is replaced by table code created for each string coming. Table is made for the input reference for the upcoming string[4].

The whole LZW compression algorithm is as follow:

1) Dictionary is initialized by all basic existing characters : {'A'..'Z','a'..'z','0'..'9'}.
2) *P* ←the first character in character stream.
3) *C* ←the upcoming character in characterstream.
4) Isstring(*P* + *C*) found in *dictionary* ?
   - If yes, so*P* ←*P* + *C* (combine *P* and *C* into a new string).
   - If not, so :
     i. Output a code to replace string P.
     ii. Add string (P + C) into dictionary and give the next number/code that has not been used indictionary to the string.
     iii. *P* ← *C*.
5) Is there still any upcoming character remained in character stream ?
   - If yes, return to step 2.
   - If not,*output* the code that will replace *string P*, and terminate the process (*stop*).

The decompression process in LZW is conducted through the principles similar to those in compression process. In the beginning dictionary is initialized by all existing basic characters. In each step, the code is then read one by one from code stream and taken from string in dictionary which corresponds to the code. A new string is then added into dictionary. The following is the full decompression process:

1) Dictionary is initialized by all the existing characters : {'A'..'Z','a'..'z','0'..'9'}.
2) *CW* ←the first code stream (referring to one of the basic characters).
3) Consult dictionary and output string of the code (*string*.*CW*) into character stream.
4) *PW* ←*CW*; *CW* ←the upcoming code ofcode *stream*.
5) Is *string*.*CW* found in dictionary ?
   - If yes, then :
     i. *output string*.*CW* to character *stream*
     ii. *P* ←*string*.*PW*
     iii. *C* ←the first character of *string*. *CW*
     iv. Add *string* (*P*+*C*) into *dictionary*
   - If not, then :
     i. *P* ←*string*.*PW*
     ii. *C* ←the first character of *string*.*PW*
     iii. output string(*P*+*C*) into character *stream* and add the string into dictionary(it finally can correspond with *CW*);
6) Is there any other code in code *stream*?
   - If yes, back to the step 4.
   - If no, terminate the process (*stop*).

2.1 Huffman Compression Method[2]

Huffman coding is a popular method for data compression. It serves as the basis for several popular programs run on various platforms. Some programs use just the Huffman method, while others use it as one step in a multistep compression process. The Huffman method generally produces better codes when the probabilities of the symbols are negative powers of 2. Huffman constructs a code tree from the bottom up (builds the codes from right to left). Since its development, in 1952, by D. Huffman, this method has been the subject of intensive research into data compression.

The algorithm starts by building a list of all the alphabet symbols in descending order of their probabilities. It then constructs a tree, with a symbol at every leaf, from the bottom up. This is done in steps, where at each step the two symbols with smallest probabilities are selected, added to the top of the partial tree, deleted from the list, and replaced with an auxiliary symbol representing the two original symbols. When the list is reduced to just one auxiliary symbol (representing the entire alphabet), the tree is complete. The tree is then traversed to determine the codes of the symbols.

3.SYSTEM DESIGN

3.1 Analysis and Design

Problem to be studied is the insertion technique on text by using a simple method of White Space Steganography. The message inserted has been compressed by using a double compression method of LZW and Huffman so that the size of the message is smaller and the capacity of the inserted message is bigger.

Figure 2 General Model of the System

3.2 System Flow Chart

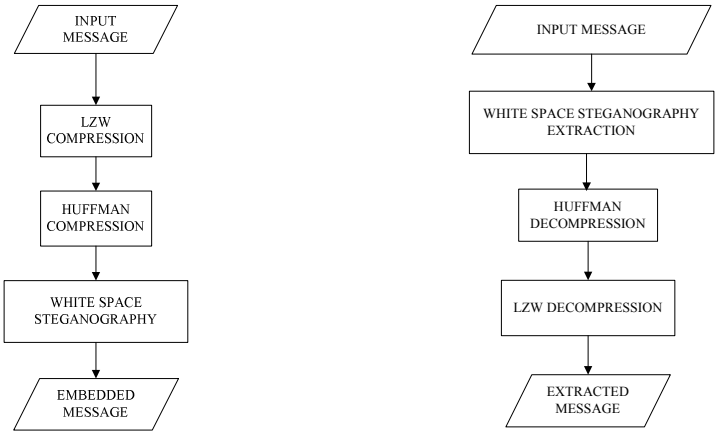

Figure 3 Flow Chart of the Compression-Embedding and Decompression-Extraction Process

The message input is a text used as the secret message that will be inserted into white space steganography. The text input should be in the list of LZW dictionary, if not, there will be a warning to reinput the text. The input text will then be compressed by using LZW (Lempel Zev Welch) technique. LZW technique uses dictionary as its reference. LZW priorinitial dictionary consists of 66 initial dictionaries comprising the characters usually used, which can be developed into 128 dictionaries. The 128 dictionaries are used to get a maximum result in the upcoming Huffman compresion. Table 1 in the appendix shows the prior initials of the LZW dictionary.

LZW compression results in the numbers representing dictionary index used. LZW dictionary has a range of 0 and 127where each numbers requires the number of bits as to represent it. The numbers have been compressed by using Huffman algorithm with a static Huffman tree table which has been determined previously. Static tree table is used if it is difficult for the receivers to redecompress the steganography message by using adaptive tree table as they do not have

the adaptive tree table. Table2 in appendix shows the example of Huffman tree table used with different probability of occurrence of LZW dictionary index.

The Huffman compression results in representation bits. These bits will be inserted into the cover text by altering bit '1' into 'Tab' and '0' into 'space'. By using Tab and space characters, the message can be difficult to be recognized. The message output comprises the text cover along with the result fromwhite space steganography. The text output will be reproceeded as to obtain the secret message inserted previously. The text cover and the inserted message using 'tab' and 'space' included in the message generated from steganography will be separated first.

The white space steganography extraction process is quite simple, i.e. by changing the 'Tab' character into bit '1' and 'Space' character into bit '0'. Huffman decompression is same with the one used in the compression by using static Huffman tree table which has been determined previously. Since it uses static Huffman tree table, the receiver is not required to ask the huffman table to the sender since the static tree table is included in the program. LZW decompression process is conducted by adapting the adaptive dictionary so that the inserted secret message can be generated.

4. APPLICATION PERFORMANCE ANALYSIS

The test in the study is aimed at:

1. Finding out the sytem perfomance in the form of compression ratio
2. Finding out the system performance in accordance with the compression and decompression time.
3. Finding out and analyzing the system performance towards different inputs

4.1 Test Setting

The test is taken to be set on android smartphone Samsung S Advance device, with the following specification :

1) Processor Dual Core 800 MHz.
2) Internal Memory: 4 GB Storage.
3) RAM 768 MB
4) External Memory: 16 GB
5) OS Android 2.3 Gingerbread.
6) PLS TFT capacitive touchscreen, 16M colors (480 x 800 pixels, 3.8 inches (~246 ppi pixel density))

Message input to be inserted are comprised by different input samples as can be seen in table 3 in appendix. The scenario used in the test is as follow:

1. Testing the compression calculation by using the manual calculation and comparing it by using the program made.
2. Measuring the compression ratio of the message input with different sum of characters and symbols.
3. Measuring the time used in the compression and decompression

Parameters used in the application test are compression time and compression ratio. The process time is an important factor in compresion and decompression process. Ratio in percentage is calculated using the following equation[5]:

$$\text{Ratio} = (\text{original file size} - \text{compressed file size}) / \text{initial size} * 100\ \% \quad (1)$$

4.2 Test Result

4.2.1 Test on the Compression Ratio Manual Calculation and the Calculation using Program

The following is the explanation on the compression result obtained from the manual calculation compared with the calculation using program. Meanwhile, the LZW compression process with that input can be seen in table 4 in appendix. The dictionary index generated from LZW compression will be compressed by using Huffman compression with static tree table.

Manual calculation :
Input message = "makan makan"
The number of characters in the input message = 11 characters
The number of bits in the input message = 11*8 = 88 bits
LZW Compression Result = 9 dictionary indexes.
LZW Compression Ratio = (11-9)/11*100% = 18.18%
Huffman Compression Result = 1111100 1111000 1111010 1111000 1111101 001011 1000011 1000101 1111101
The number of bit obtained from lzw+huffman compression = 62 bits
LZW+huffman compression ratio = (88-62)/88*100% = 29.55%

The test result on the program showing the application interface can be seen in figure 4.

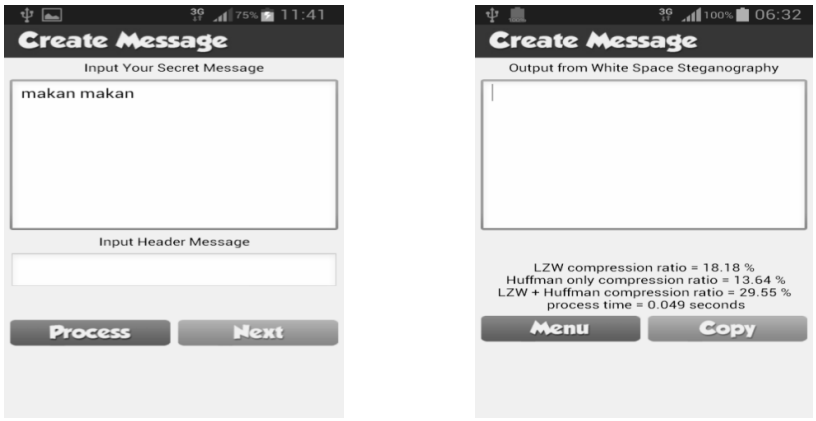

Figure 4 The application interface on the test of input compression and the decompression result

It can be seen that the LZW compression ratio is 18.18% and the Huffman+LZW ratio is 29.55%. It means that the result generated from the program is same withthat one resulted from the manual calculation.

4.2.2 Compression Ratio

The test is taken with ratio of the number character with input message symbol number 0. Table 5 in appendix shows compression ratio as the test result on several processes conducted in the implemented system. Table 5 can be presented in the analysis graphic as can be seen in figure 5.

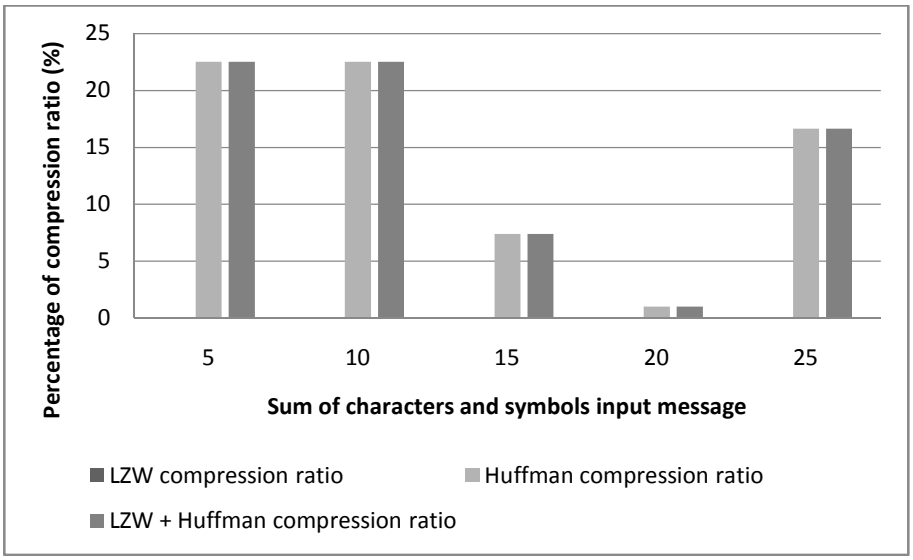

Figure 5 Graph of Compression Ratio of Input Message with Difference in Sum of Characters and Symbol 0

The test above shows that the number of characters and symbol is equal that causes no repeating character pair defined that causes LZW compression ratio of 1:1. The scenario above shows that it is only Huffman compression that works, where the compression ratio will increase if the input message character has a small bit representation in static Huffman tree table. A non letter symbol has a big bit representation so that the ratio is small.

The following test is the test on the differenceof the input message of 30 characters that has different total number of symbols. The test can be seen in table 6 in appendix. Table 6 can be summarized into the graphs in Figure 6.

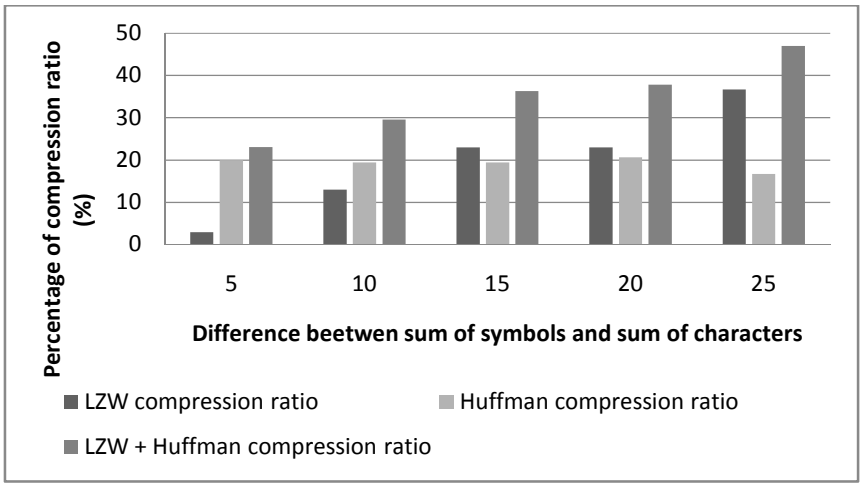

Figure6Graphs of compression ratio with the input message of 30 characters and variousdifference of the sum of characters and symbols

From the test, it can be seen that with the same total number of characters, the bigger difference in the sum of character and the sum of symbol, the bigger LZW+ Huffman will be. It is because that big difference means that there are many repeating symbols and as a consequence, LZW compression is getting better. Unfortunately,Huffman compression ratio is getting smaller if there are many repeating high bit symbols. However, LZW compression value will exceed the Huffman compression if the repeating symbols are found more, which also means that even if the difference is getting bigger, the Huffman+LZWdouble compresion ratio is still better.

The following test is the test using input message with repeating character pair. The test process can be seen in table 7 in the appendix. From the table, it can be concluded in the graphs in figure 7.

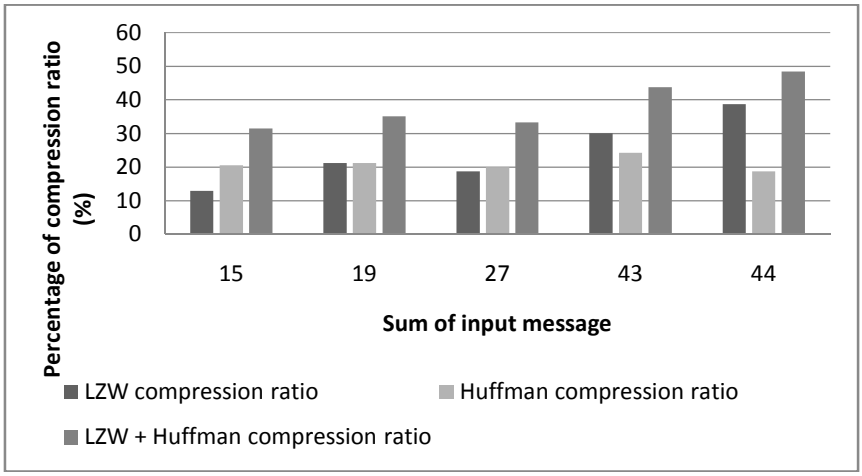

Figure7Graphs of compression ratio with the input message that has repeating character pairs and different sum of input message

The test shows that the Huffman+LZW double compression ratio is getting bigger if the number of input characters as well as the number of repeating character pairsare also getting bigger. It is influenced by the LZW compression which is getting better if the number of repeating characters is increasing. Meanwhile, for Huffman, the size of compression depends on the number of bit given for each character.

4.2.3 Process Time

Table 8 in the appendix is the result of process time used as to create and read messages involving 20 samples. However, figure 8 shows the visualization in accordance with table 8.

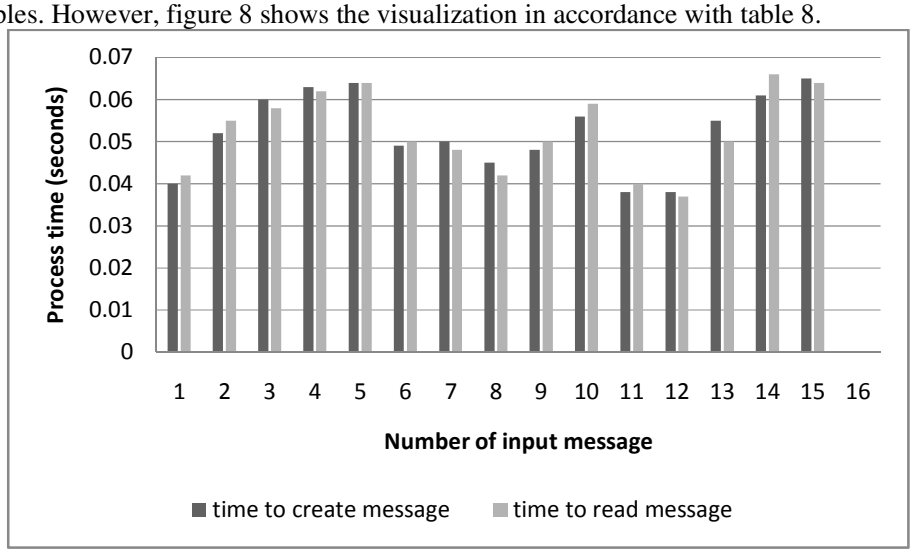

Figure8 Process time result

The process time used is almost same for all the tests conducted since all of the tests have a similar process. However, the input message with more characters requires a longer time. The process time cannot be separated from thehardware used in the experiment.

5. CONCLUSION AND RECOMMENDATION

5.1 Conclusion

Based on the implementation of steganography of text using double compression with LZW-Huffman, it can be concluded that:

1. From the compression ratio obtained, it can be seen that the system is able to produce different ratio depending on the input used. If the message input consisting of several repeating character pairs, the compression ratio will be big. If the input message involves few repeating words, the compression ratio will be smaller due to the LZW compression algorithm. If the input message is only comprised by symbol, instead of letter, the compression ratio will be small since in the static huffman tree table, the bit for non letter character is represented to be big and the compression ratio will be big if the input message comprises many letter characters.

2. The speed in the process time as to generate text stego is relatively fast and the influence of the input that has many repeating words as well as the one that has few repeating words is

not much different. The process time tocreate messageandto read messageis not much different.

5.2 Recommendation

1. Instead of text, picture or sound can also be usedas the media in steganography, i.e. as to ensure a higher security level.
2. Other compression algorithm can also be used as to get a proper compression ratio.

REFFERENCE

[1] Deorowicz, Sebastian.20013. *Universal Lossless Data Compression Algorithms*. Gliwice:Silesian University of Technology

[2] D. Salomon. Data Compression: The Complete Reference. Springer, 1998.

[3] Howe, D. *Free Online Dictionary of Computing*, http://www.foldoc.org/

[4] Kanikar, Prashasti, Ratnesh N. Chaturvedi dan Vibhishek Kashyap. 2013. *Image Steganography using DCT, DST, Haar and Walsh Transform*. International Journal of Computer Applications, Vol. 65, No. 17, Hal. 34-37.

[5] Khalid Sayood, Introduction to Data Compression, Academic Press, 2000.

[6] Linawati dan Henry P. Panggabean. 2004, *Perbandingan Kinerja Algoritma Kompresi Huffman, Lzw, dan Dmc pada Berbagai Tipe File.* Integral, Vol. 9, No. 1, 2004, hal 14-15

[7] Nosrati, Masoud., RonakKarimidan Mehdi Hariri.2011. *An Introduction to Steganography Methods.* World Applied Programming, Vol 1, No 3, 191-195, Agustus 2011

[8] Safaat, Nazruddin.2012. *Pemrograman Aplikasi Mobile Smartphone dan Tablet PC Berbasis Android*. Bandung : Informatika

[9] Singh, Dr. Ajit dan Meenakshi Gahlawat. 2013. *Secure Data Transmission using Watermarking and Image Compression*. International Journal of Advanced Research in Computer Engineering & Technology, Vol. 2, No.5, Hal. 1709-1715.

[10] Saleh Saraireh. 2013. *A Secure Data Communication System Using Cryptography and Steganography*. International Journal of Computer Networks & Communications (IJCNC) Vol. 5 No. 3, May 2013.

APPENDIX

Table 1Prior Initials of LZW Dictionary

Index	Dictionary	Index	Dictionary	Index	Dictionary	Index	Dictionary
0	a	17	R	34	7	51	:
1	b	18	S	35	8	52	;
2	c	19	T	36	9	53	,
3	d	20	U	37	@	54	.
4	e	21	V	38	#	55	=
5	f	22	W	39	%	56	_
6	g	23	X	40	&	57	<
7	h	24	y	41	*	58	>
8	i	25	z	42	/	59	{
9	j	26	(spasi)	43	-	60	}
10	k	27	0	44	+	61	[
11	l	28	1	45	(	62	]
12	m	29	2	46	)	63	|
13	n	30	3	47	?	64	^
14	o	31	4	48	!	65	$
15	p	32	5	49	“		
16	q	33	6	50	‘		

Table 2 Huffman Tree

Index of LZW Dictionary	Probability	Huffman Bit	Index of LZW Dictionary	Probability	Huffman Bit
0	0.0234375	1111000	64	0.0078125	11101111
1	0.0234375	1111001	65	0.0078125	0100110
2	0.0234375	000100	66	0.015625	1000011
3	0.0234375	001100	67	0.015625	1000100
4	0.0234375	001101	68	0.015625	1000101
5	0.0234375	001110	69	0.015625	1000110
6	0.0234375	001111	70	0.015625	1000111
7	0.0234375	010000	71	0.015625	1001000
8	0.0234375	010001	72	0.015625	1001001
9	0.0234375	010010	73	0.015625	1001010
10	0.0234375	1111010	74	0.015625	1001011
11	0.0234375	1111011	75	0.015625	1001100
12	0.0234375	1111100	76	0.015625	1001101
13	0.0234375	1111101	77	0.015625	1001110
14	0.0234375	1111110	78	0.015625	1001111
15	0.0234375	1111111	79	0.015625	1010000
16	0.0234375	000000	80	0.015625	1010001
17	0.0234375	000001	81	0.015625	1010010
18	0.0234375	000010	82	0.015625	1010011
19	0.0234375	000011	83	0.015625	1010100
20	0.0234375	000101	84	0.015625	1010101
21	0.0234375	000110	85	0.015625	1010110
22	0.0234375	000111	86	0.015625	1010111
23	0.0234375	001000	87	0.015625	1011000
24	0.0234375	001001	88	0.015625	1011001
25	0.0234375	001010	89	0.015625	1011010
26	0.0234375	001011	90	0.015625	1011011
27	0.0078125	11001010	91	0.015625	1011100
28	0.0078125	11001011	92	0.015625	1011101
29	0.0078125	11001100	93	0.015625	1011110
30	0.0078125	11001101	94	0.015625	1011111
31	0.0078125	11001110	95	0.015625	1100000
32	0.0078125	11001111	96	0.015625	1100001
33	0.0078125	11010000	97	0.015625	1100010
34	0.0078125	11010001	98	0.015625	1100011
35	0.0078125	11010010	99	0.015625	1100100
36	0.0078125	11010011	100	0.015625	0100111
37	0.0078125	11010100	101	0.015625	0101000
38	0.0078125	11010101	102	0.015625	0101001
39	0.0078125	11010110	103	0.015625	0101010
40	0.0078125	11010111	104	0.015625	0101011
41	0.0078125	11011000	105	0.015625	0101100
42	0.0078125	11011001	106	0.015625	0101101
43	0.0078125	11011010	107	0.015625	0101110
44	0.0078125	11011011	108	0.015625	0101111
45	0.0078125	11011100	109	0.015625	0110000
46	0.0078125	11011101	110	0.015625	0110001
47	0.0078125	11011110	111	0.015625	0110010
48	0.0078125	11011111	112	0.015625	0110011
49	0.0078125	11100000	113	0.015625	0110100

50	0.0078125	11100001	114	0.015625	0110101
51	0.0078125	11100010	115	0.015625	0110110
52	0.0078125	11100011	116	0.015625	0110111
53	0.0078125	11100100	117	0.015625	0111000
54	0.0078125	11100101	118	0.015625	0111001
55	0.0078125	11100110	119	0.015625	0111010
56	0.0078125	11100111	120	0.015625	0111011
57	0.0078125	11101000	121	0.015625	0111100
58	0.0078125	11101001	122	0.015625	0111101
59	0.0078125	11101010	123	0.015625	0111110
60	0.0078125	11101011	124	0.015625	0111111
61	0.0078125	11101100	125	0.015625	1000000
62	0.0078125	11101101	126	0.015625	1000001
63	0.0078125	11101110	127	0.015625	1000010

Table 3 Samples of input message

No	Input message
1	Detik
2	Abcdefghij
3	detik0123456789
4	!@#$%^&*()_-+=:;,.<>
5	abcdefghijklmnopqrst!@#$%
6	aabbbbcdefghijklmnopqrstuvwxy
7	aaaabcbcbcddddefghijklmnopqrst
8	aaaabcccdefffefgefghijklmnoooo
9	aaabbabccbcbbbcddeefghijjjjjjj
10	abababababcdceabcdedeaaaaaaabc
11	ular lari lurus
12	saya suka susu sapi
13	eleven benevolent elephants
14	freezy breeze made these three trees freeze
15	can you can a can as a canner can can a can?

Table 4 LZW compression process

Step	Position	Character	*New Dictionary*	Output
1	1	M	[66]ma	[12]
2	2	A	[67]ak	[0]
3	3	K	[68]ka	[10]
4	4	A	[69]an	[0]
5	5	n	[70]n(spasil)	[13]
6	6	(spasi)	[71](spasi)m	[26]
7	7	ma	[72]mak	[66]
8	9	Ka	[73]kan	[68]
9	11	N	-	[13]

Table 5 Compression Ratio of input message with different in the sum of character and symbol 0

Input Message	Sum of Characters	Sum of Symbols	Difference	LZW Compression Ratio(%)	Huffman Compression Ratio (%)	LZW + Huffman Compression Ratio (%)
detik	5	5	0	0	22,5	22,5
abcdefghij	10	10	0	0	22,5	22,5
detik0123456789	15	15	0	0	7,4	7,4
!@#$%^&*()_-+=:;,.<>	20	20	0	0	1	1
abcdefghijklmnopqrst!@#$%	25	25	0	0	16,67	16,67

Table6 Compression Ratio of input message with the sum of characters of three and different sum of symbols

Input Message	Sum of Characters	Sum of Symbols	Difference	LZW Compression Ratio(%)	Huffman Compression Ratio (%)	LZW + Huffman Compression Ratio (%)
aabbbbcdefghijklmnopqrstuvwxy	30	25	5	2.9	20	23.07
aaaabcbcbcddddefghijklmnopqrst	30	20	10	13	19.4	29.6
aaaabcccdefffefgefghijklmnoooo	30	15	15	23	19.4	36.31
aaabbabccbcbbbcddeefghijjjjjjj	30	10	20	23	20.6	37.89
abababababcdceabcdedeaaaaaaabc	30	5	25	36.7	16.67	47

Tabel 7 Input Message

Message that has many repeating character pairs				
Input Message	Sum of Characters	LZW Compression Ratio(%)	Huffman Compression Ratio (%)	LZW + Huffman Compression Ratio (%)
ular lari lurus	15	13.02	20.63	31.51
saya suka susu sapi	19	21.26	21.26	35.06
eleven benevolent elephants	27	18.69	20	33.33
freezy breeze made these three trees freeze	43	30.07	24.24	43.82
can you can a can as a canner can can a can?	44	38.65	18.69	48.45

Table8 Processing Time

Input message with difference in the sum of characters and symbol 0			
No	Input Message Input	Time to *create message*(second)	Time to *read message* (second)
1	detik	0.04	0.042
2	abcdefghij	0.052	0.055
3	detik0123456789	0.06	0.058
4	!@#$%^&*()_-+=:;,.<>	0.063	0.062
5	abcdefghijklmnopqrst!@#$%	0.064	0.064
Input message with different in the sum of characters and different symbols			
No	Input Message Input	Time to *create message*(second)	Time to *read message* (second)
6	aabbbbcdefghijklmnopqrstuvwxy	0.049	0.05
7	aaaabcbcbcddddefghijklmnopqrst	0.05	0.048
8	aaaabcccdefffefgefghijklmnoooo	0.045	0.042
9	aaabbabccbcbbbcddeefghijjjjjjj	0.048	0.05
10	abababababcdceabcdedeaaaaaaabc	0.056	0.059
Input message with repeating character pairs			
No	Input Message Input	Time to *create message*(second)	Time to *read message* (second)
1	ular lari lurus	0.038	0.04
2	saya suka susu sapi	0.038	0.037
3	eleven benevolent elephants	0.055	0.05
4	freezy breeze made these three trees freeze	0.061	0.066
5	can you can a can as a canner can can a can?	0.065	0.064