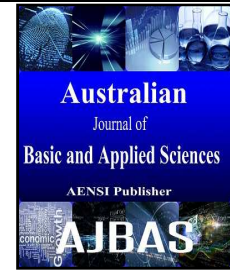




AUSTRALIAN JOURNAL OF BASIC AND APPLIED SCIENCES

ISSN:1991-8178 EISSN: 2309-8414
Journal home page: www.ajbasweb.com



Coalesce of Compression and Encoding in Short Messaging Service Using Lb64

Dr. K. Valarmathi, Ms. V. Sathiya, Mrs. M. Maheswari and Mr. G. Jegan

Department of CSE, Panimalar Engineering College Chennai, India

Address For Correspondence:

Dr. K. Valarmathi, Department of CSE, Panimalar Engineering College Chennai, India.
E-mail: m.mahe05@gmail.com

ARTICLE INFO

Article history:

Received 10 December 2015

Accepted 28 January 2016

Available online 10 February 2016

Keywords:

SMS, Compression, Encoding, LZW, Base64, LB64

ABSTRACT

Today in the world of mobile communication SMS is one of the fastest growing medium through which a sender can send message in a short period of time. During the transmission of data i.e. a threat on banking transaction, OTP, credit card processing, Provident Fund transaction, Mail Recovery SMS etc there is a need for securing text messages. An innovative LB64 algorithm is implemented specifically for SMS. This LB64 is a hybrid technique which integrates both compression and encryption using XOR operation which does not exist in previous techniques. The performance of this technique shows excellent results thereby improving the security mechanism and this also makes use of efficient bandwidth.

INTRODUCTION

Data compression is a technique of encoding system that let considerable reduction in the total number of bits to store or transmits a file. One of the lossless data compression widely used is LZW algorithm (Nishad, P.M., R. ManickaChezian, 2012), which is a dictionary based algorithm. LZW compression is coined based on its developers, A. Lempel and J. Ziv, which was then modified by Terry A. Welch (1984). Lempel-Ziv-Welch (LZW) (1984) this algorithm proposed by Welch in 1984. LZW compression works finest for files containing more of repetitive data. (Mohammad Hosseini, 2015) Shannon –Fano and Huffman works on historical evidence and research(probability coding) whereas LZW uses dictionary based compression and it is also been observed in paper (BoukariSouley, 2014; Haroon Altarawneh, Mohammad Altarawneh, 2011) as best algorithm for text compression.LZW compression can be used in a variety of file formats such as TIFF files, GIF files, PDF files. This can also be applied for text and monochrome images.

This algorithm reads a sequence of 8-bits from a text file and compresses them using LZW with 12-bit code words and writes the results to output file. It does not analyze the incoming text, but it adds each new string of characters.

Lossless compression technique is called as reversible compression as the original message is rephrased by the decompression process without any loss of data (SubhamastanRao, T., 2011; Vishwagupta, 2012). Henceforth the information does not modify during the compression and decompression process.

Base64 encoding design are generally used when there is a need to encode twofold data that needs be stored and transferred over media that are intended to pact with textual data (CongfuXu, 2010). This is to make sure that the data remains intact without alteration during transfer. Wikipedia says, Base64 encoding is a technique used for converting binary data into blocks of printable ASCII characters (i.e., A–Z, a–z, 0–9, +, /). The Base64 encoding algorithm uses a particular conversion process to determine ASCII characters to every 6 bits of a stream bytes .In other words, a binary stream of bytes is split into chunks of six bits and each chunk is

Open Access Journal

Published BY AENSI Publication

© 2016 AENSI Publisher All rights reserved

This work is licensed under the Creative Commons Attribution International License (CC BY).

<http://creativecommons.org/licenses/by/4.0/>



Open Access

To Cite This Article: Dr. K. Valarmathi, Ms. V. Sathiya, Mrs. M. Maheswari and Mr. G. Jegan., Coalesce of Compression and Encoding in Short Messaging Service Using Lb64. *Aust. J. Basic & Appl. Sci.*, 10(1): 181-184, 2016

associated to an ASCII character. The ASCII character (=) whose value is 64 indicates the end of the binary stream.

Base64 ensures that data won't contain any special or unpredictable characters like passing data in a URL or storing it in cookies.

II. Proposed Method LB64:

The compression algorithm accepts single character at a time. It checks whether the code(ASCII Code) is there in the dictionary, then it adds the character to the source string and waits for the next one and this happens for the first character. If the source string is not in the dictionary it appends the source string and writes the newly generated code. Then the source string is set to the new character.

This focuses on multiple sizes of text data and the ratio decreases as the text size increases. This is because larger text data will create shorter compressed code. The compressed data is XORed with the cipher key. The resultant XORed data is given as input to the Encryption algorithm which produces encrypted data.

The Encrypted data is XORed with the cipher key and reverse process of Encryption algorithm is applied to the XORed data for decryption.

The decompressor constructs its own table therefore it matches exactly the compressor dictionary's data and so only the compressed codes need to be sent (Refer Fig1).



Fig. 1: Architectural framework of the proposed system.

PHASE I (Compression):

Step 1. Generate a code table with 256 entries (indexed with ASCII codes 0 through 255) representing the ASCII table.

Step 2. Initialize the encoded string(S) with the first byte of the input stream and read the next byte from the input stream(C).

Step 3. If the byte(C) is an EOF stop the compression process.

Step 4. Else generate a string that concatenate the bytes.

Step 5. Check the string is present in the table, read the next byte

Step 6. If the string is not present, add the new string to the table with the new code and write encoded string(S) to the output stream. Read the next byte.

Step 7. Repeat from 3 to 6 until EOF is encountered

Step 8. xor the compressed data and apply base64 to the xored data

III. Implementation:

The proposed LB64 compression and encryption mechanism is implemented with NetBeans IDE8.0.2. The performance estimation factors, compression ratios and time requirement is obtained from different size of text as shown in Table III.

Compression Algorithm makes use of two variables S and C. The Variable C holds a character (i.e. a value between 0 to 255). The variable S is a variable length string which contains one or more characters. The algorithm progress by taking first byte from the text and stores it in S refer Table I. Each time a byte is read from an input file and assign it to a variable C. The concatenation of two variables are searched in the dictionary. If a match is not found in the dictionary three actions are taken output the code for S. The concatenation of S + C is stored in the dictionary with the new code. C value is assigned to S. If a matching string is found in the dictionary no action should be taken.

Compression algorithm is illustrated in Table 1 for the input text : jingle bells jingle bells

Table I: Phase-I Compression.

S	C	Compressed output	Code	SC
j	i	106	256	[internal=[106,105]] = ji
i	n	105	257	[internal=[105,110]] = in
n	g	110	258	[internal=[110,103]] = ng
g	l	103	259	[internal=[103,108]] = gl
l	e	108	260	[internal=[108,101]] = le
e	"space"	101	261	[internal=[101,32]] = e"space"
"space"	b	32	262	[internal=[32,98]] = "space"b
b	e	98	263	[internal=[98,101]] = be
e	l	101	264	[internal=[101,108]] = el
l	l	108	265	[internal=[108,108]] = ll
l	s	108	266	[internal=[108,115]] = ls
s	"space"	115	267	[internal=[115,32]] = s"space"
"space"	j	32	268	[internal=[32,106]] = "space"j
ji	n	256	269	[internal=[106,105,110]] = jin
ng	l	258	270	[internal=[110,103,108]] = ngl
le	"space"	260	271	[internal=[108,101,32]] = le"space"
"space" b	e	262	272	[internal=[32,98,101]] = "space"be
el	l	264	273	[internal=[101,108,108]] = ell
ls	eof	108		

The input stream is converted into binary values. The octet stream is read from left to right to group into six bits chunks. Each chunk is encrypted using base64 index table to get encrypted string shown in Table II.

Table II : Phase II Encryption.

Input stream	Binary Values	Split binary values by 6 bits	Base64 Values	Encrypted String
106	1101010	011010	26	a
105	1101001	100110	38	m
110	1101110	100101	37	l
103	1100111	101110	46	u
108	1101100	011001	25	Z
101	1100101	110110	54	2
32	100000	110001	49	x
98	1100010	100101	37	l
101	1100101	001000	8	I
108	1101100	000110	6	G
108	1101100	001001	9	J
115	1110011	100101	37	l
		011011	27	b
		000110	6	G
		110001	49	x
		110011	51	z

IV. Measuring Compression Performances:

Depending on the nature of the application there are various criteria to measure the performance of a compression algorithm which includes time and space efficiency. The compression behavior depends on the redundancy of symbols, type and structure of input source, category of compression algorithm (loss and lossless), therefore it is difficult to measure the performance of algorithm in general. If a lossy compression algorithm is used to compress a particular source file, the space efficiency and time efficiency would be higher than that of the lossless compression algorithm. The procedure to evaluate the lossless compression algorithm is as follows:

Compression Ratio (CR):

$$CR = \frac{\text{size after compression}}{\text{size before compression}} \quad (1)$$

Compression Factor (CF):

$$CF = \frac{\text{size before compression}}{\text{size after compression}} \quad (2)$$

Saving Percentage (SP), calculates the reduction of the source file in percentage:

$$SP = \frac{(\text{size before compression} - \text{size after compression})}{\text{size before compression}} \quad (3)$$

The compression algorithm works well for the text data of size 322 bytes with a saving percentage of 39.44%. The saving % decreases as the data size increases due to the increase in the repetition of words (highly repetitive value) that match with the vocabulary inside the library.

Table III: LB64 Algorithm Results.

LB64 algorithm results (4)							
Original File		Phase I				Phase II	
S.No	File Size (bytes)	Condensed file size (bytes)	Condensed/Compression ratio	Condensed Time (sec)	Saving %	Encrypted file size (bytes)	Encryption time (sec)
1.	322	195	.60	1.232	39.44	340	0.0177
2.	258	171	.66	1.217	33.72	294	0.0172
3	162	132	.81	1.217	18.51	224	0.0172
4.	110	105	.95	1.232	4.54	172	0.0167

Conclusion:

Incorporating security in existing compression algorithm is on focus by present researchers. This approach is mainly developed to improve the speed and to provide more security LB64 provides lossless compression and compressed message is XORed with the given key and encryption algorithm is applied. The proposed algorithm reduces the file size and encrypts the file in any type of public and private application for sending confidential data.

This LB64 technique can be enhanced to send multimedia files. In this LB64 small file size decreases the ratio which can be improved in future. As LB64 uses only 64 characters it will not accept alphanumeric inputs and can be enhanced with all ASCII characters.

REFERENCES

- Nishad, P.M., R. ManickaChezian, 2012. "Optimization Of Lzw (Lempel-Ziv-Welch) Algorithm To Reduce Time Complexity For Dictionary Creation In Encoding And Decoding" *AJCSIT*, 2(5): 114–118.
- Asral BAHari Jambek and Alina Khairi, 2014. "Performance comparison of Huffman and Lempel-Ziv Welch Data Compression for Wireless sensor Node Application", *American Journal of Applied Science*, 11 (1): 119-126, ISSN: 1546-9239.
- Welch, T.A., 1984. "A technique for high-performance data compression". *IEEE Comput*, 17(6): 8–19.
- Kodituwakku, S.R., U.S. Amarasinghe, "Comparison Of Lossless Data Compression Algorithms For Text Data". *Indian Journal of Computer Science and Engineering*, 1(4): 416-425.
- SubhamastanRao, T., M. Soujanya, T. Hemalatha, T. Revathi, 2011. "Simultaneous Data Compression And Encryption" *International Journal of Computer Science and Information Technologies*, 2(5): 2369-2374.
- Vishwagupta, GajendraSingh, Ravindra Gupta, 2012. "Advance Cryptography Algorithm For Improving Data Security" *International Journal of Advanced Research in Computer Science and Software Engineering*, 2-1, ISSN: 2277 128X.
- Kodituwakku, S.R., U.S. Amarasinghe, "Comparison Of Lossless Data Compression Algorithms For Text Data", *Indian Journal Of Computer Science And Engineering*, 1(4): 416-425.
- CongfuXu, Yafang Chen, Kevin Chiew, 2010. "An Approach To Image Spam Filtering Based On Base64 Encoding And N-Gram Feature Extraction", *Tools With Artificial Intelligence(ICTAI)*, 22nd IEEE International Conference on (1) ISSN :1082-3409.
- BoukariSouley, Prodipto Das and Shirley Tanko, 2014. "A Comparative Analysis of Data Compression Techniques", *International Journal of Applied Sciences & Engineering (IJASE)*, 63-82.
- Haroon Altarawneh, Mohammad Altarawneh, 2011. "Data Compression on Text Files:AComparison Study", *International Journal of Computer Applications(0975-8887)*, 26-5.
- Mohammad Hosseini, 2015. "A Survey of Data Compression Algorithms and their Applications", conference on Research Gate.