ISSN:0975-9646

An Indirect Encryption using Compression with Random bit stuffing

Dr. M. Ramesh[1] , Dr. B. Hemanth Kumar[2], Prof. M. Surendra Prasad Babu[9],

[1, 2.] Associate Professor, Dept., of Information Technology,
RVR & JC College of Engineering, Guntur, A.P. India
[3.] Professor , Dept., of CS & SE,
AU college of Engineering, Andhra University, Visakhapatnam, A.P, India.

Abstract: Traditional compression schemes in specific work on diminishing size of given data and do not concentrate on other aspects. Wide variety of compression methods are in use and differ in their compression ratios, speed and complexity involved in doing compression. Some techniques are suitable for huge data and others are suitable for small amount of data. But we present a compression technique that is straightforward to realize, implement and feasible in all conditions. Unlike traditional compression methods, besides reducing data size, it also acts as an encryption tool to encrypt data. The tool also provides authentication for the encrypted data.The proposed method is applicable to data of any size which consume no additional memory. Encryption and Decryption of this method are very simplified and require no complex mathematical operations. Our experiments disclose that, it achieves reasonable compression and performs good encryption in quick time bounds.

Keywords: Compression, Encoding, Decoding and Symmetric key

1. INTRODUCTION

Data compression is a mechanism used to bring down size of given data from size X to size Y where Y < X. Two basic categories of compression techniques [1] exist; lossless and lossy. In lossless compression original message can be retained completely from compressed one, but in lossy compression some pieces of data get lost in the process. In this paper we have bring in first category of compression which results in no loss of data. Prior to this work, various compression algorithms were designed to reduce data size. Huffman Coding [2] reduces size by giving assigning shorter prefix codes to highest occurring symbol and longest prefix codes to least occurring symbol. Dynamic Huffman Coding [3] attempts to resolve problems in [2] but in this, sender and receiver must construct tree dynamically which is a time consuming process.

Run Length Encoding [4] identifies repeating symbols and places (symbol,N) pairs where N is number of times symbol repeated. Arithmetic Coding [5] does compression by doing mathematical calculations. High Speed Search and Memory Efficient Huffman Coding [6] increases searching speed of symbols and mitigate memory size. Efficient Test Pattern Compression Techniques based on Complementary Huffman Coding [7] gives better compression than [2][3] by identifying complement values. An Authenticated Bit Shifting and Stuffing Methodology [8] reduces size of data by reading eight symbols and inserting bits of eighth symbol into preceding seven bytes. But these techniques do only compression and not else. Each of them has constraints, works on certain patterns of text and is time consuming in implementation. But our proposed method is suited to all sizes of data, feasible in implementation and majorly it provides security beyond compression.

Remaining sections of the paper proceed in this fashion. Section 2 gives overview of proposed approach. A simple example is demonstrated in section 3 and reverse approach of this technique is given in section 4. Section 5 shows authentication results are given in section 6 and finally conclusions are presented in section 7.

2. PROPOSED METHOD

The proposed method adopts the compression mechanism in [8] and refines it to act as encryption. It is a symmetric key encryption mechanism that starts with creating blocks of size eight bytes in given text in a sequential fashion. It tries to reduce the block size from eight to seven bytes by embedding eighth byte into first seven bytes. This is possible because, if we consider ASCII characters, each use only 7 bits instead of 8; hence MSB of each character gets wasted. We use this bit to store an additional bit from other characters.

```
while (not-end-of File)
{
  read eight bytes from File say B1 to B8
  for( x = 7 to 1)
  {
    c = B8[x] // read from MSB-1 to LSB
    if(c == 0)
     y = get 1's location randomly in B[8-x]
     else
     y = get 0's location randomly in B[8-x]
    if(y==MSB)
          B[8-x][y] = c
    else
     {
       B[8-x][MSB] = Bx[y];
       B[8-x][y] = c;
     }
     sub~key = get 3 bit value of y
     write sub~key to key file
  }
 Write B1 to B7 bytes to output file.
}
```

Algorithm 1: Compression and Encryption

In [8] additional characters are always stored in MSB bits which give regularity. But in our method, storing in MSB is not fixed and we select locations out of 8 in a random fashion. Randomly selected location becomes a sub key and is stored in key file. To avoid monopoly, we try to place 0 in the locations of 1 and place 1 in the locations of 0. Then previous value of selected location is moved to MSB of the corresponding character. This process is repeated for all the blocks resulted. It drastically changes values of blocks and reduces the blocks sizes from 8 to 7 by doing two things (compression and encryption) at same time. Since we store 7 bits of eighth character into 8 random locations of first seven characters in a block, we need 3 bits key for every character. Hence for a block we require 21 bits as a sub key and all the sub keys for all the blocks are recorded in a key file which is sent to other side. The proposed scheme algorithm is self explanatory and presented in Algorithm 1. At the end of algorithm, output file contains both compressed and encrypted data.

3. Decryption and Decompression

The process of getting original message is given in algorithm 2. It reads seven bytes from the compressed file (also encrypted) to form a block. In each block, it picks a bit out of eight possible locations and does necessary exchanges and complementation's based on sub keys in key file. After this, every block yields an eighth byte that is hidden in first seven bytes. This process is repeated for the remaining blocks and finally actual message is revealed.

```
while (not-end-of File)
{
  read 7 bytes from File say B1 to B7
  for(x = 1 to 7)
  {
     sub~key = read 3 bit value from key file
     y = get decimal value of sub~key
     if ( y==MSB )
     {
       B8[8-x] = Bx[y];
       complement B8[8-x];
     }
     else
     {
       B8[8-x] = Bx[y];
       Bx[y] = Bx[MSB];
     }
  }
}
```

Algorithm 2: Decompression and Decryption

4. Illustrative Example

This section demonstrates the process with a simple example. Let us consider the message "MEDICINE". Treating them as an eight bytes belonging to a block, the procedure of compression and encryption process with results is shown in table 1.

First column of the table indicates bytes presented in the given text. Second and third columns present ASCII values and binary values of corresponding bytes. According to algorithm, in every block eighth byte is embedded into first seven bytes of the block in random locations. Therefore fourth column specifies which bit of E byte used for embedding process. The location where E bit is stored in a byte is selected randomly and this random value becomes sub key which is given in fifth column. Once bits are embedded in corresponding bytes, byte values gets changed whose resulting binary and ASCII values are presented in last two columns of the table. After completion of the process, E gets embedded into first seven bytes which reduces the block size from eight to seven. This mechanism is continued for all the remaining blocks.

5. Authentication

A 4 bit polynomial is generated dynamically and this is applied on encrypted data for authentication. This polynomial performs modulo 2 division operation on encrypted data. For every four characters of encrypted data the modulo division gets a three bit remainder. In decryption this remainder is act as an authenticated key for the four characters.

Byte	ASCII	Binary	E bit	subkey	Resulting Binary	ASCII
M	77	01001101	1	101	01101101	109
E	69	01000101	0	010	11000001	193
D	68	01000100	0	010	11000000	192
I	73	01001001	0	011	11000001	193
C	67	01000011	1	100	01010011	83
I	73	01001001	0	110	10001001	137
N	78	01001110	1	101	01101110	110
E	69	01000101				

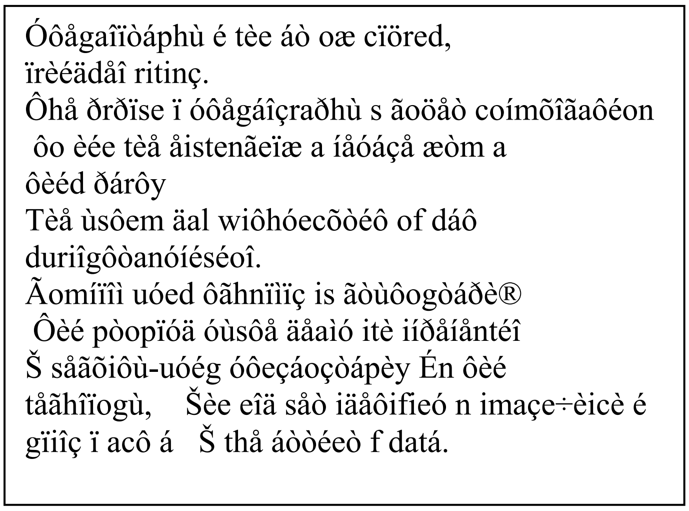
Table 1: Result of Compression and Encryption Process

6. Results

The plain text is compressed and encrypted. Authentication is provided for this encrypted data by using four bit dynamic polynomial. Figure:1 shows the actual plain text message. The compressed and encrypted message is shown in the figure :2. Figure 3 shows the authenticated encrypted data.

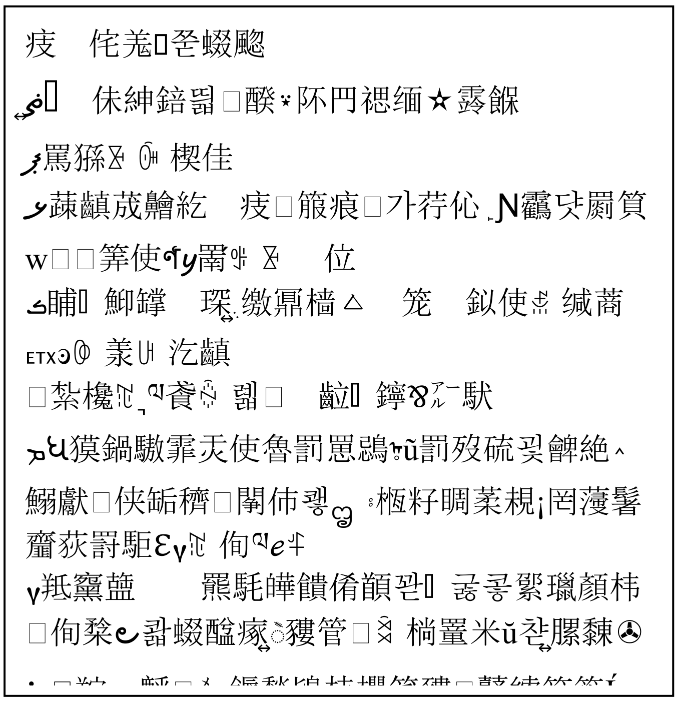
Steganography is the art of covered, or hidden writing. The purpose of steganography is covert communication - to hide the existence of a message from a third party. The system deal with security of data during transmission. Commonly used technology is cryptography. This proposed system deals with implementing security- using steganography. In this technology, the end user identifies an image which is going to act as the carrier of data.

Figure 1: Original message



Óôågaîïòáphù é tèe áò oæ cïöred,
ïrèéädåî ritinç.
Ôhå ðrðïse ï óôågáîçraðhù s ãoöåò coímõîãaôéon
ôo èée tèå åistenãeïæ a íåóáçå æòm a
ôèéd ðárôy
Tèå ùsôem äal wiôhóecõòéô of dáô
duriîgôòanóíéséoî.
Ãomíïîì uóed ôãhnïïïç is ãòùôogòáðè®
Ôèé pòopïóä óùsôå äåaió itè iíðåíåntéî
Š såãõiôù-uóég óôeçáoçòápèy Én ôèé
tåãhïïogù,   Šèe eîä såò iäåôifieó n imaçe÷èicè é
gïiîç ï acô á   Š thå áòòéeò f datá.

Figure 2: Encrypted message



疺　侘羗□咢蝃颾
ݦ□　俅紳錇뢺□醆⅍阫円禵缅☆霿餜
ݧ罵猻ⶼ ⵀ 楔佳
ݷ蕀黰荿鰫紌　疺□箛痕□가荐伀 ݐ靍닷劂貿
w□□箄使ꝗy罱ꞁ ⶼ　位
ݭ睸□ 鲫鏄　琛.缴鼏槒△　笼　釵使ꝱ 缄蔐
ETX϶ⓟ 羡ꀱ 汔黰
□紮檇ꞁ ꟼ飡ꞑ 뢺□　齸□ 鐞ꝸ⺀馱
ꞔꝡ貘鍋鰍霏兲使鲁罰罳鵖ꝵũ罰歿硫긪鞞絶^
鰯獻□佚鲘穦□鬫佈캥 ꝰ椢籽睭茉粯¡罔蔆髫
齏莸罰駏ɛγꞁ 侚ꟼe丰
γ羝爨鹽　　羆馲曅饋俙頏꽌□ 긇긓綤瓕顏栉
□侚桊ꝺ갊蝃醓瘶Ꝺ鞻管□ꝸ 梢置米ŭ찬膘繛✇
[illegible]

Figure 3: Encrypted message with key

7. Conclusion

We have presented a new compression method that is unambiguous, easy to implement, understand and require no generation of prefix codes or coding tables like other technique. It is applicable to data of all the sizes and for any text it gives 12.5% compression. It also secures data by encrypting data beside to compressing data. Since every block of eight bytes generate 21-bit key, total size of the key file would be (FileSize/8)*21. If the original file is not in multiples of 8 bytes we do some padding in last byte. With this scheme, key file occupies 32.8125% of original file size. To break this indirect encryption, a cryptanalyst need to strive for $2^{21}\times B$ trials where B is the number of blocks resulted from original message of size eight bytes.

References

[1] "Introduction to Data Compression", Khalid Sayhood, Morgan Kaufmann, 1996.

[2] "A Method for the Construction of Minimum – Redundancy codes", Proceedings of the IRE, sept 1952, 1098 – 1102.

[3] http://en.wikipedia.org/wiki/DynamicHuffmanCoding

[4] http://en.wikipedia.org/wiki/Information_Theory

[5] http://en.wikipedia.org/wiki/Arithmetic_Coding

[6] "Memory Efficient and High Speed Search Huffman Coding", Reza Hashemian, IEEE Transactions on Communications, vol 43, no 10, oct 1995, 2576-2581.

[7] "Efficient Test Pattern Compression Techniques based on Complementary Huffman Coding", Shyue-Kung Lu et al, IEEE 2009.

[8] "An Authenticated Bit Shifting and Stuffing (BSS) Methodology for Data Security", B. Ravi Kumar et al, Computer Engineering and Intelligent Systems, vol 2, no 3, 94 – 104.