International Journal of Automation and Computing 10(6), December 2013, 578-586
DOI: 10.1007/s11633-013-0756-3

A New Compression Scheme for Secure Transmission

M. Baritha Begum Y. Venkataramani
Saranathan College of Engineering, India

Abstract: Encryption techniques ensure security of data during transmission. However, in most cases, this increases the length of the data, thus it increases the cost. When it is desired to transmit data over an insecure and bandwidth-constrained channel, it is customary to compress the data first and then encrypt it. In this paper, a novel algorithm, the new compression with encryption and compression (CEC), is proposed to secure and compress the data. This algorithm compresses the data to reduce its length. The compressed data is encrypted and then further compressed using a new encryption algorithm without compromising the compression efficiency and the information security. This CEC algorithm provides a higher compression ratio and enhanced data security. The CEC provides more confidentiality and authentication between two communication systems.

Keywords: Data compression, encryption algorithm, compression with encryption and compression (CEC) algorithm, Huffman coding, run length coding, bits per character (BPC).

1 Introduction

Compression is the process in which the information is encoded using fewer bits than an uncoded representation. The redundancy of the data is decreased by the compression algorithm. This further reduces the storage space required for data. Data compression offers an important approach to reduce transmission costs by using pre-existing bandwidth efficiently[1]. In the past decade, there has been an increase in the amount of data transmitted via the internet, representing text, images, speech, video, sound and computer data. Hence, there is a need for efficient compression algorithms that can be effectively used in the existing network bandwidth.

Security is an important factor in our digital life. Encryption is the art of achieving security by encoding messages to make them non-readable by an intruder. To prove the theoretical feasibility of operations, the algorithm implements the compression of encrypted data[2−4].

There is a need for compression combined with encryption. In most cases, encryption followed by compression is used. WinZip and WinRAR[5] are two packages which encrypts the data using advanced encryption standard (AES) algorithm after compression. Due to improper injection of cryptography into compression, several attacks are possible[6, 7].

Another drawback is that text processing has to be performed on compressed data[8,9]. Finding keywords in the compressed data using compressed pattern matching methods[10, 11] is an example of text processing.

An alternate approach is unifying compression and encryption to secure the data. Some of the compression algorithms like interval splitting in arithmetic coding[8, 12], using multiple code trees in Huffman coding[13], encrypted dictionaries in dictionary based compression[1, 14], adaptive character word length (ACW) algorithm[15], differential method[16], dynamic Markov compression (DMC), prediction by partial matching (PPM), and burrows wheeler transform (BWT) algorithms are used for compression[17−19]. These algorithms do not achieve better compression ratio. Hence, there is a need for developing a better algorithm.

The preprocessing step is done on a source file prior to applying an existing compression algorithm. This transformation is designed to make the compression of the source data easier[20]. Dictionary based encoding is generally used for this type of preprocessing transformation of the source text[21]. The dictionary is created by commonly used words expected in the input files[10, 22, 23]. The dictionary must be prepared in advance, and must be known to the source and destination. In this technique, each word in the dictionary has to be replaced by ASCII character[24−26]. The dictionary based encoding is very much weak and vulnerable to attacks. But the dictionary based encryption provides the required security.

The idea of text compression is to preprocess the input text and transform it into an intermediate form. Then, this data can be compressed and secured with better efficiency. The purpose of the proposed technique is to reduce the consumption of expensive resources and to reduce data length. In this paper, a new hybrid technique for securing data is introduced. To attain better compression, a new compression and encryption cum compression algorithm is developed. In this approach, new dictionary based compression algorithm called multi-dictionary compression algorithm is applied to a source text leading to a better improvement in the existing algorithm, and further encryption cum compression (reduced array based encryption) offers a sufficient level of security of the transmitted information. So this method is called new compression and encryption cum compression (CEC). The multi-dictionary based encryption provides the required security.

The four main steps of this technique are multi-dictionary based compression, BWT with run length encoding (RLE) based compression, reduced array based encryption algorithm and Huffman coding based compression process. The words are extracted from the input files and formed as dictionary entries. In this technique, each word in the multi-dictionary has to be replaced by ASCII character. This multi-dictionary method increases the speed of encoding and decoding, because word retrieval is very easy and fast.

Manuscript received June 29, 2012; revised February 6, 2013

The multi-dictionary based compressed result is further compressed using a BWT with RLE. After this step, the compressed data is encrypted using reduced array based encryption algorithm. This reduced array based encryption algorithm is used for both encryption and compression.

In general encryption algorithm, plain text is converted into the cipher text with the same size or increased size. But this algorithm has not only done the encryption but also reduced the input array size by 3 times. Two keys are generated here. The primary key is generated by three values which are designed by user. The secondary key is generated by TA-key which represents the probability of data. Further, encrypted output is compressed with Huffman coding. This CEC technique combines the compression and encryption processes, and thus develops a better transformation yielding greater compression and added security. The reverse operation is performed on receiver side as shown in Fig. 1. The rest of the paper is organized as follows. Section 2 presents a multi-dictionary making algorithm. Section 3 presents encoding algorithm. Section 4 presents decoding algorithm. Section 5 provides performance analysis. Section 6 concludes the paper.

2 Multi-dictionary generation

The words are extracted from the input files and formed as dictionary entries. If the preceding alphabet in a word is in upper case, it is changed into lower case and frequency of occurrence is calculated. Individual dictionaries are created for each starting letter of the words. Similarly, all ASCII characters are categorized into a separate dictionary. This multi-dictionary method increases the speed of encoding and decoding. This is because words retrieval is very easy and fast. ASCII character numbers (33–255) are used as codes. An ASCII character is assigned as code to every word. Table 1 shows codeword formation. In Table 1, 170 single ASCII characters are assigned as codes for first 170 words. For the next 170 words, the same 170 ASCII characters with a prefix of character "a" is employed. Thus, it becomes a two-character code. The words (341–4250) will have the combination of (b, $\cdots$, z) and the single 170 ASCII characters. For next 170 words (4251–4421), letter "A" combines with the ASCII and become codes. The words (4422–8840) will have the combination of (B, $\cdots$, Z) and the single 170 ASCII characters. $N\times$ 170 = Number of words assigned as codes for two characters combination. N = Number of alphabetic characters [(a, $\cdots$, z) + (A, $\cdots$, Z) = 52]. Further words (8841-13261) will have the combination of $170\times N$ codes with prefix of (a–z), thus, they became three character codes. Then, words with combinations are given as below: The first letter "A", second letter [A–Z] and third letter becomes 170 character codes. Similarly, other words in the dictionary form codes. $M\times N\times$ 170 = Number of words assigned as a code for three character combination. The combination of (a–z) (A–Z) or (A–Z) (a–z) or (A–Z) (A–Z) or (a–z) (a–z) and the single 170 ASCII characters create 4 59 680 codes.

The short codes are assigned to the most frequently used words. The longest codes are assigned to the less frequently used words[25].

3 Encoding algorithm

3.1 Burrows wheeler transform and Run length encoding

The input files are converted into code (ASCII character) by using multi-dictionary based compression. This code is given to the input of BWT. Most of the compression methods operate in the streaming mode, where the code inputs may be a byte or several bytes which are processed until the end of the file is sensed. The burrows wheeler method works in a block mode, where the input stream is read block by block and each block is encoded separately as one string. This method is also referred to as the block sorting. The BWT method is general purpose, it works well on images, sound and text. It can achieve very high compression ratios.

BWT output is given to the input of RLE. After reading the first character, the count is 1 and the character is saved. Subsequent characters are compared with the one already saved. If they are identical to it, the repeat-count is incremented. When different characters are read, the operation depends on the value of the repeat count. If it is small, the saved character is written on the compressed file and the newly read character is saved. Otherwise, a "@" is written followed by the repeat-count and the saved character. A run of three characters results in no compression. Only the runs longer than 3 characters get compressed[27].

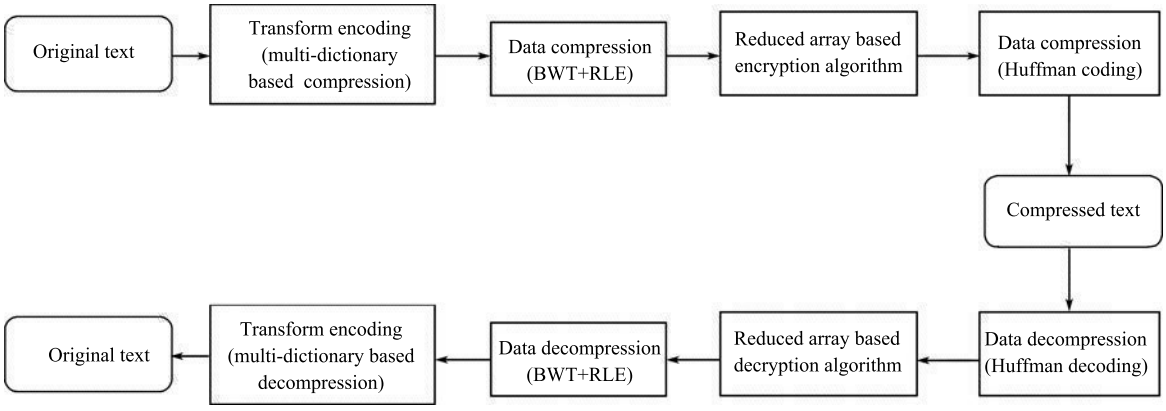

Fig. 1 Block diagram of CEC compression

Table 1 Codeword formation

No.	Code type	Symbol	Code word	Number of possible code words
1	Single character	!@#$%^*()_+ ⋯ 170	!@#$%^*	170
		170 single ASCII characters are assigned as a code for first 170 words		
2	Two Characters	a–z (prefix code) 26 in No	a!	8 840×
		A–Z (prefix code) 26 in No	a@	((26 + 26)×170)
		!@#$%^&*()_+ 170 in No	z!	
		1) The combination of (a–z) and the single 170 ASCII characters.	z@	
		2) The combination of (A–Z) and the single 170 ASCII characters.	A!	
			A@	
			Z!	
			Z@	
3	Three characters	a–z (prefix code) 26 in No	ba!	459 680×
		A–Z (prefix code) 26 in No	da@	(52 × 52 × 170)
		!@#$%^&*()_+ 170 in No	fD!	
		The combination of (a–z)(A–Z)or(A–Z)(a–z)or(A–Z)(A–Z)or	bz@	
		(a–z)(a–z) and the single 170 ASCII characters.	Ma!	
			PA@	
			Nj!	
			HZ@	
	Total			468 690

3.2 The proposed algorithm

Output from the run length coding is given as input to the encryption with compression algorithm. In general encryption algorithms, plain text is converted into the cipher text with the same size or increased size. This algorithm not only has done the encryption, but also has reduced input array size by 3 times. Two keys are generated here. One is primary key and the other is secondary key. Primary key is designed by three values such as starting value, maximum value and the factor. During encryption time, secondary key is generated by the number of characters occurring in the input string. Encryption output is reduced to 1/3 of an input array size. By using primary key and secondary key, we can retrieve the original input.

3.2.1 Key generation

Algorithm 1. Key generator
Input: Start value, MAX_VALUE, factor
Output: Stream of keys
Key size = 3
KEY (1)← Start value
for $i = 2, \cdots$, Key size do
KEY $(i) \leftarrow$ KEY $(i-1) \times$ MAX_VALUE+ factor
end for

Let the primary key be denoted by KEY (i), where $i = 1, 2$ and 3. KEY(1) is the start value of the key. MAX_VALUE represents that maximum value is used to generate key 2, 3, etc. Factor has integer value 2, 3 and 4. KEY (i) is generated by multiplication of maximum value (MAX_VALUE) with KEY $(i-1)$, where $i = 1$, 2 and 3, and then factor is added. Factor may be any value from 2 to 4.

3.2.2 Reduced array based encryption algorithm

Algorithm 2. Encryption coding for stream of data depends on the minimize array algorithm
Input: Key1 (primary key), RLE output
Output: TA-key, encrypted data
%Compute TA-key of Symbols
S_AC ← Number of data in X
for jAC=1, ⋯, S_AC do
S_2AC ← Number of symbols in the TA-key
flag =0
for kAC = 1, ⋯, S_2AC(2) do
if TA-key(kAC)=X(jAC)
flag = 1
end if
end for
end for
if Flag is zero
TA-key (S_2AC (2) +1) ← X (jAC)
end if
%Data Encrypted
S__ ← Number of data in X
Pad zeros after the last position in the array from
X(S_(2)+1 to S_(2)+3)
Initialize i and L is one
While i is less than or equal to S_(2)
Sum is zero
for j=0 to 2 do

Sum ← Sum+(X $(i+j)\times$ Key1 $(j+1)$)
end for
Encrypted data (L) ← Sum
$L4 \leftarrow L+1$
$i \leftarrow i+3$
end while

Run length coding output is given to the input of reduced array based encryption algorithm (X). Let X be the input character, S_AC be the number of characters in X. TA-key (T) be the secondary key, S_2AC be the number of characters in the TA-key. This algorithm is used to find the exact data depending on the TA-key, which represents the probability of data. The first character of (X) is the first character of TA-key. Instead of repeated characters in X, it occurs for single time in TA-key.

This algorithm is used to convert every 3 data in X into a real number. Each three input character set is multiplied by corresponding KEY (i) where i=1, 2, 3, and then three multiplication outcomes are added. It is considered as encrypted output. Encrypted output has a array size which is three times less array size of input X. Encrypted data length is unlimited. It depends upon the length of X.

3.3 Huffman coding

The encrypted output code undergoes Huffman coding for further compression. Huffman code was generated by using binary tree.

Consider the source symbols $\{b_1, b_2, \cdots, b_n\}$ with frequencies $\{y_1, y_2, \cdots, y_n\}$ for $y_1 \geqslant y_2 \geqslant \cdots \geqslant y_n$, where the symbol b_j has frequency y_j. Using the Huffman′s algorithm, the codeword z_j for $1 \leqslant j \leqslant n$, which is a binary string, for symbol b_j can be obtained. Let us denote C= $\{z_j, \cdots, z_n\}$ as the Huffman code. Let the level of the root of the Huffman tree be zero, and the level of the other node be equal to one more than that. Codeword length l_j for b_j can be known as the level of b_j.

Assume the right edge corresponds to "0" and the left edge corresponds to "1". The codeword of a node j, denoted z(j), is defined as the bit sequence corresponding to the path from the root to node j. The codeword of a sub tree T_j, denoted $I(T_j)$, is defined as the codeword of T_j's root. The level of a subtree T_j, denoted $l(T_j)$, is defined as the level of T_j's root. Given a string $x = x_1x_2, \cdots, x_m$, we define the jth prefix of x, for $j = 1, \cdots, m$, as prefix $j(x)$ $= x_1x_2, \cdots, x_j$ and prefix $0(x) =$ _ is an empty string. The Huffman procedure is based on two observations:

1) The symbols that occur more frequently will have shorter code words than the symbols which occur less frequently.

2) The two symbols that occur less frequently will have the same length.

4 Decoding algorithm

The output code obtained through Huffman coding is given to the Huffman decoder for decryption.

4.1 Reduced array based decryption algorithm

Encrypted as well as compressed data is decrypted by primary key, secondary key and encrypted data.

Algorithm 3. Decryption for minimized data

Input: Encrypted data, Key (primary key), TA-key (secondary key)

Output: New_arr

S_Enc ← Number of Encrypted Data
Set $L = 1$
for $i = 1, \cdots,$ S_Enc(2) do
  S_TA-key ← Number of symbol in the TA-key
  Set flag=1
  Set EST =0
  $T(1)$, $T(2)$, and $T(3)$ are set to one
  $S_1 = 1$, $S_2 = 1$, $S_3 = 1$
  while (check flag=1)
    EST=0
    $T(1) \leftarrow$ TA-key(S_1)
    $T(2) \leftarrow$ TA-key(S_2)
    $T(3) \leftarrow$ TA-key(S_3)
    for $K_2 = 1, 2$ and 3 do
      EST ← EST+ (T (K_2) × Key1 (K_2))
      EST ← mod(EST, 255)
      If check EST is equal to encrypted data (i)
        set flag =0
      else
        S1 ← $S_1 + 1$
        if (S_1 >S_TA-key(2))
            S2 ← $S_2 + 1$
            set $S_1 = 1$
        end if
        if (S_2 >S_TA-key(2))
            $S_3 \leftarrow S_3 + 1$ set $S_2 = 1$
        end if
        if (S_3 >S_TA-key(2))
            Set $S_3 = 1$
        end if
      end if
    end for
  end while
  New_arr(L:L+2) ← T of array one to three
  L ← L+3
end for

Let the encrypted data be denoted by encrypted data (S_Enc). S_Enc is the number of data in encrypted data. Number of symbols in the TA-key is denoted by S_TA-key. EST is the predicted encrypted data. First, three contents of S_TA-key (i) where i = 1, 2, 3 correspondingly are multiplied by primary key (j), where j = 1, 2 and 3, and added to form EST. Then EST is compared with first entry of encrypted data. If EST and encrypted data are the same, then S_TA-key (i) is the decrypted data. Otherwise, contents of S_TA-key are multiplied with primary key with different combinations [S_TA-key (i, i, i+1) or S_TA-key (i, i+1, i+1) or S_TA-key (i, i, i)], and compared with EST. When the match is found, decrypted data is estimated from the S_TA-key contents.

It has some zeros at the end of the estimated array (New_arr), because at the encryption or coding algorithm,

zeros are padded automatically.

4.2 Decoding from dictionary

Decrypted data is given to the RLE decoder and then BWT. The RLE decoded output converts short sequence symbol into a long sequence symbol. After this conversion, the output is given as an input to the BWT reverse transform which rearranges the data into the original order. The output of BWT becomes a code when the upper case letter (A–Z) or lower case letter (a–z) combines with ASCII character. Based on starting of the character, the word is searched and extracted from the respective dictionary. If two consecutive special characters accompany along with (a, ⋯, z) or (A, ⋯, Z), they are considered as different codes and are extracted from the dictionary.

As an example, a section of text from Calgary corpus version of paper 1 looks like this in the original text:

"Its performance is optimal without the need for blocking of input data. It encourages a clear separation between the model for representing data and the encoding of information with respect to that model. It accommodates adaptive models easily. It is computationally efficient." (1)

Number of characters required = 420

Memory space = 420 bytes

Running text (1) through the multi-dictionary based encoder yields the following text:

ÒNM{LBNå|HNMS{LBNå|HNMS{LBNå|He¥CP@B{Ù O⁻D{{E|ÔO°∼AEA}KCC4E&. (2)

Number of characters required = 61

Memory space = 61 bytes

Time requirement = 0.355457 s.

Running text (2) through the BWT encoder yields the following text:

KNNN∼EHÚÔCMMMH{SSSBEECP@C⁻A{4|||}{{{Ò BBBDåååA°LLL¥eOO&|{NNN7. (3)

Number of characters = 61

Memory space = 61 bytes

Time requirement = 0.052289 s.

Running text (3) through the run length encoder yields the following text:

KNNN∼EHHÙÔCMMMH{SSSBEECP@C⁻A{4|||}{{{Ò BBBDåååA°LLL¥eOO&|{NNN7. (4)

Number of characters required = 61

Memory space = 61 bytes

Time requirement = 0.179045 s.

Running text (4) through the encryption yields the following result:

Primary key =[38 2436 155908]

Secondarykey=KN∼E4@HÙÔCM{SBP⁻A|}Ò7Då°L¥e O&

Encrypted data = Ï*8Y©÷ ÉÛ+ *ü?³p⊙èB2 (5)

Number of character required = 21

Memory space = 21 bytes

Time requirement = 0.010227 s.

Running text (5) through the Huffman coding yields the following result: #àú÷l©9¸®<c

00101001100100000110001000001000001111010010111001000111101110010111010100100001100001110110

Memory space = 11 bytes

Time requirement = 0.179045 s.

In the above example, input data collected from test files paper 1 has 420 characters. The compression based on multi-dictionary based method reduces it to 61 characters. Then BWT changes the order of the input data. Then, compression is done using RLE. The obtained output data has 86 characters. This is encrypted and compressed to $\frac{1}{3}$ by array reduction algorithm. Then it reduces to 21 characters. Finally, 11 characters are achieved by using Huffman coding.

$$\text{Compression ratio} = \frac{11}{420} = 0.0261$$

$$\text{Bits per character} = \frac{11}{420} \times 8 = 0.2095.$$

5 Performance analysis

Experiments were performed on the CEC transformation algorithms described in Sections 2–4 using standard Calgary corpus test file collections[28].

In order to evaluate its performance, the CEC scheme is implemented using Matlab to compress test files from standard corpora such as Calgary. At this stage, it has been taken to optimize the runtime of the compression-decompression prototype codes. Therefore, results can be obtained for the compression ratio, bits per character and compression time are presented.

The performance metrics, such as compression ratio and bits per character (BPC) for this algorithm are compared with standard algorithm (arithmetic coding, Huffman coding, Lempel-Ziv-Storer-Szymanski (LZSS)), dictionary based encoding (DBE), multi-dictionary based compression, multi-dictionary BWT with RLE (MBR), MBR with Huffman coding (MBRH), MBR with new reduced array based encryption (CE) and CE with Huffman coding (CEC). The results are shown graphically. They prove that CEC outperforms the other techniques in compression ratio, bits per character, compression time and security.

$$\text{Compression ratio} = \frac{\text{Output file size}}{\text{Input file size}}$$

$$\text{Bits per character (BPC)} = \frac{\text{Output file size}}{\text{Input file size}} \times 8$$

Table 2 List of files used in experiments

File name	Size (byte)	Description
Bib	111 261	Bibliography
Geo	102 400	Geological seismic data
Obj1	21 504	VAX object program
Paper 1	53 161	Technical paper
Paper 2	82 199	Technical paper
Paper 3	46 526	Technical paper
Paper 4	13 286	Technical paper
Paper 5	11 954	Technical paper
Paper 6	38 105	Technical paper
Progc	39 611	Source code in "C"
Progl	71 646	Source code in "Pascal"
Progp	49 379	Text: English text

Details on test files are as shown in Table 2. This data is collected from Calgary corpus[28]. The experiments are

done by using selected existing compression algorithms as well as with a new encryption cum compression algorithm to compress the input files. CEC was implemented in Matlab. The experiment is to determine the percentage of decrease in text size using the CEC transformation algorithms. This CEC algorithm compares the eight coding formats, namely, arithmetic coding, Huffman coding, LZSS, (DBE), multi-dictionary based compression, MBR, MBRH and CE.

The compression ratios, bits per character and compression time determined for 12 test files of various sizes from the Calgary corpus are given.

First, the input text from test file is compressed by multi-dictionary based compression. Second, the resultant code is compressed by BWT and RLE. Further, it is encrypted as well as compressed using the CE scheme, then it is compressed by Huffman coding. So, the resultant compression ratio is the combination of CEC compression ratios.

Results are shown in Tables 3–5. Tables 3 and 4 compare the proposed algorithm with an existing standard compression algorithm. The results are shown graphically in Figs. 2–5.

Table 3 Comparison of compression ratios

File name	Arithmetic coding	Huffman BWT	LZSS BWT	Dictionary based compression	Multi-dictionary based compression	Multi-dictionary+ BWT+RLE[MBR]	MBRH	CE	CEC
Bib	0.654	0.457	0.627	0.278	0.277	0.244	0.192	0.08	0.074
Geo	0.707	0.725	0.788	0.57	0.607	0.521	0.423	0.17	0.159
Obj1	0.746	0.596	0.661	0.232	0.221	0.187	0.148	0.06	0.056
Paper 1	0.623	0.452	0.622	0.282	0.273	0.254	0.202	0.08	0.077
Paper 2	0.578	0.46	0.642	0.282	0.27	0.26	0.204	0.09	0.078
Paper 3	0.589	0.482	0.667	0.275	0.262	0.251	0.199	0.08	0.074
Paper 4	0.603	0.508	0.672	0.2765	0.267	0.252	0.195	0.08	0.076
Paper 5	0.633	0.507	0.657	0.31	0.294	0.274	0.213	0.09	0.082
Paper 6	0.626	0.454	0.619	0.301	0.294	0.273	0.211	0.09	0.084
Progc	0.655	0.438	0.591	0.286	0.276	0.248	0.2	0.08	0.077
Progl	0.595	0.335	0.456	0.237	0.235	0.195	0.155	0.06	0.057
Progp	0.612	0.345	0.461	0.174	0.165	0.144	0.119	0.05	0.044

Table 4 Comparison of bits per character

File Name	Arithmetic coding	Huffman BWT	LZSS BWT	Dictionary based compression	Multi-dictionary based compression	Multi-dictionary+ BWT+RLE[MBR]	MBRH	CE	CEC
Bib	5.232	3.656	5.016	2.224	2.219	1.955	1.538	0.652	0.593
Geo	5.656	5.8	6.304	4.56	4.857	4.168	3.386	1.392	1.27
Obj1	5.968	4.768	5.288	1.856	1.765	1.495	1.187	0.495	0.448
Paper 1	4.984	3.616	4.976	2.256	2.183	2.028	1.615	0.676	0.617
Paper 2	4.624	3.68	5.136	2.256	2.161	2.077	1.63	0.694	0.623
Paper 3	4.712	3.856	5.336	2.2	2.097	2.009	1.596	0.673	0.595
Paper 4	4.824	4.064	5.376	2.212	2.136	2.018	1.562	0.672	0.606
Paper 5	5.064	4.056	5.256	2.48	2.35	2.194	1.7	0.726	0.659
Paper 6	5.008	3.632	4.952	2.408	2.349	2.186	1.686	0.729	0.671
Progc	5.24	3.504	4.728	2.288	2.206	1.987	1.596	0.662	0.615
Progl	4.76	2.68	3.648	1.896	1.882	1.56	1.243	0.519	0.453
Progp	4.896	2.76	3.688	1.392	1.317	1.153	0.95	0.385	0.355

Table 5 Comparison of compression times

File name	Dictionary based compression (s)	Multi-dictionary based compression (s)	BWT(s)	Run length coding (s)	Encryption algorithm (s)	Huffman coding (s)
Bib	638	85.77	9	1.4	0.17	11.1
Geo	1620	409.5	10	2	0.5	44.8
Obj1	180	26.33	4.7	0.3	0.1	0.98
Paper 1	207	41.34	3.2	0.7	0.15	44.8
Paper 2	264	48.49	6.5	0.9	0.2	7.19
Paper 3	182	30.32	1.8	0.6	0.24	2.9
Paper 4	42	3.896	2.5	0.4	0.1	0.88
Paper 5	47	4.719	2.4	0.3	0.1	0.89
Paper 6	118	29.79	0.7	0.6	0.14	2.42
Progc	142	33.06	0.2	0.7	0.14	2.38
Progl	167	53.95	3.8	0.8	0.1	3.59
Progp	125	21.16	0.3	0.6	0.12	1.59

In Table 3 and Fig. 2, the compression ratios are compared with standard algorithm, such as arithmetic coding, Huffman coding, LZSS and CEC. In Table 3 and Fig. 3, the performance of our scheme with reference to compression ratio is compared with various schemes.

The CE coding achieves higher efficiency than MBR and MBRH coding formats. However, the ratio achieved using CE coding is less than that using CEC coding.

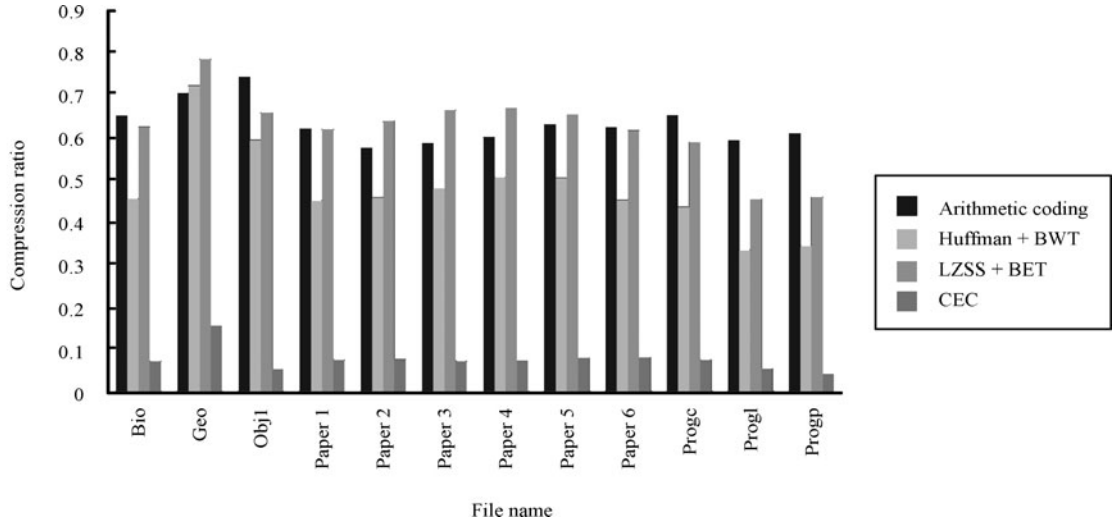

Fig. 2 Comparison of compression ratios

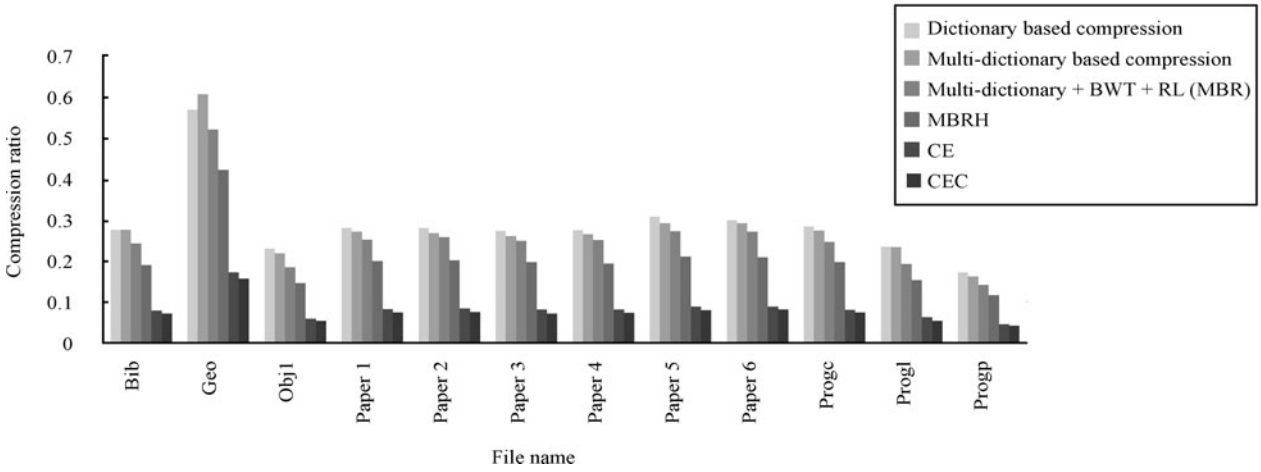

Fig. 3 Comparison of compression ratios

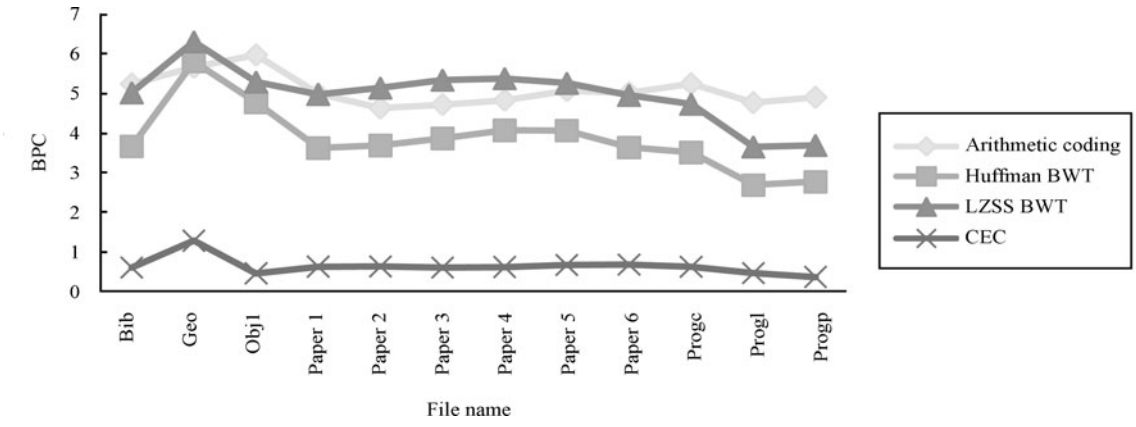

Fig. 4 Comparison of bits per character

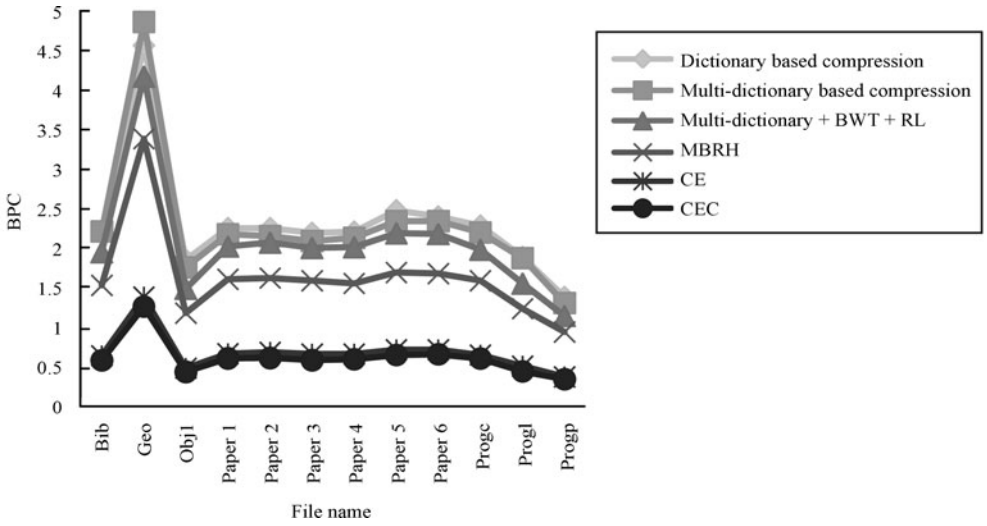

Fig. 5 Comparison of bits per character

For example, for the bib text file, the compression ratios achieved by the arithmetic coding, Huffman coding, LZSS, DBE, multi-dictionary based compression, MBR, CE and CEC coding formats are 0.654, 0.457, 0.627, 0.278, 0.277, 0.244, 0.192, 0.08 and 0.074, respectively. The biggest gain is achieved on progp (resulting mainly from the EOL-encoding) as well as on obj1 and progl.

The compression of geo is practically unaffected by CEC, because those files do not contain any textual data. There are two files in the corpus that do contain some textual data paper 3 and bib. CEC yields the same gain on paper 3 and bib. Paper 1 and progc files also have the same gain.

In Table 4 and Fig. 4, the performance with reference to bits per character is compared with standard algorithm such as arithmetic coding, Huffman coding, LZSS and CEC.

In Table 4 and Fig. 5, the performance of our scheme with reference to bits per character is compared with various schemes. The CE coding achieves higher efficiency in bits per character than statistical coding, DBE, MBR, MBRH and MBR coding formats. The results are listed in Table 4. The results reveal that the bits per character achieved using CEC is higher than that using CE coding by a very small margin. However, the BPC in CEC coding provides nearly the same performance as the CE algorithms. For example, for the paper 1 text file, the BPC achieved by the arithmetic coding, Huffman coding, LZSS, DBE, multi-dictionary based compression, MBR, CE and CEC coding formats are 4.984, 3.616, 4.976, 2.256, 2.183, 2.028, 1.615, 0.676 and 0.617, respectively.

6 Conclusions

The CEC algorithm provides better results than other algorithms. This algorithm has an admirable and viable performance as it outperforms the other widely used data compression algorithms. The compression ratio depends on the number of words in the file, size of the test file, and the frequencies and distribution of words within the file.

Nine coding formats have been investigated. It reveals that the highest compression ratio achieved is for CEC as it performs coding and compression at the same time. The CEC scheme can be used as a complementary scheme to any statistical and dictionary based lossless compression algorithm, such as static or adaptive Huffman coding, arithmetic coding, the LZSS algorithms, or any modified form of them. Our approach is to secure the message using CEC technique, to compress it for the reduction in length, and to encrypt it using the new reduced array based encryption algorithm. The CEC results have achieved excellent improvement in data compression and security without increasing its size over the existing techniques. Our future work will focus on the performance of this scheme in compressing multimedia files.

References

[1] V. K. Govindan, B. S. Shajeemohan. Compression scheme for faster and secure data transmission over networks. In *Proceedings of the International Conference on Mobile Business*, IEEE Computer Society Washington, DC, USA, pp. 678–681, 2005.

[2] D. Klinc, C. Hazay. A. Jagmohan, H. Krawczyk, T. Rabin. On compression of data encrypted with block ciphers. In *Proceedings of Data Compression Conference*, IEEE, Snowbird, UT, USA, pp. 213–222, 2009.

[3] T. M. Mahmoud, B. A. Abdel-latef, A. A. Ahmed, A. M. Mahfouz. Hybrid compression encryption technique for securing SMS. *International Journal of Computer Science and Security*, vol. 3, no. 6, pp. 473–482, 2010.

[4] R. E. L. Metzler, S. S. Agaian. Cipherstream covering for secure data compression. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, Anchorage, AK, USA, pp. 3370–3377, 2011.

[5] J. Daemen, V. Rijmen. *The Design of Rijndael: AES-the Advanced Encryption Standard*, New York: Springer, 2002.

[6] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohn, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, H. X. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. *Journal of Cryptology*, vol. 21, no. 3, pp. 350–391, 2008.

[7] G. S. W. Yeo, R. C. W. Phan. On the security of the WinRAR encryption feature. *International Journal of Information Security*, vol. 5, no. 2, pp. 115–123, 2006.

[8] H. Kim, J. T. Wen, J. D. Villasenor. Secure arithmetic coding. *IEEE Transactions on Signal Processing*, vol. 55, no. 5, pp. 2263–2272, 2007.

[9] H. Kaplan, E. Verbin. Most Burrows-Wheeler based compressors are not optimal. *Combinatorial Pattern Matching, Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer-Verlag, vol. 4580, pp. 107–118, 2007.

[10] T. Gagie, G. Manzini. Move-to-front, distance coding, and inversion frequencies revisited. *Combinatorial Pattern Matching, Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer-Verlag, vol. 4580, pp. 71–82, 2007.

[11] W. K. Hon, R. Shah, J. S. Vitter. Compression, indexing, and retrieval for massive string data. *Combinatorial Pattern Matching, Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer-Verlag, vol. 6129, pp. 260–274, 2010.

[12] J. T. Zhou, O. C. Au, P. H. W. Wong. Adaptive chosen-ciphertext attack on secure arithmetic coding. *IEEE Transactions on Signal Processing*, vol. 57, no. 5, pp. 1825–1838, 2009.

[13] Y. K. Lin, S. C. Huang, C. H. Yang. A fast algorithm for Huffman decoding based on a recursion Huffman tree. *The Journal of Systems and Software*, vol. 85, no. 4, pp. 974–980, 2012.

[14] W. K. Hon, T. H. Ku, R. Shah, S. V. Thankachan, J. S. Vitter. Compressed text indexing with wildcards. *String Processing and Information Retrieval, Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer-Verlag, vol. 7024, pp. 267–277, 2011.

[15] H. Al-Bahadili, S. M. Hussain. A bit-level text compression scheme based on the ACW algorithm. *International Journal of Automation and Computing*, vol. 7, no. 1, pp. 123–131, 2010.

[16] S. Kumar, S. S. Bhadauria, R. Gupta. A temporal database compression with differential method. *International Journal of Computer Applications*, vol. 48, no. 6, pp. 65–68, 2012.

[17] D. Salomon. *Data Compression: The Complete Reference*, 2nd ed., New York: Springer, pp. 18–22, 2000.

[18] M. O. Külekci. On scrambling the Burrows-Wheeler transform to provide privacy in lossless compression. *Computers and Security*, vol. 31, no. 1, pp. 26–32, 2012.

[19] Y. F. Chien, W. K. Hon, R. Shah, J. S. Vitter. Geometric Burrows-Wheeler transform: Linking range searching and text indexing. In *Proceedings of the Data Compression Conference*, IEEE, Washington, DC, USA, pp. 252–261, 2008.

[20] M. A. Martínez-Prieto, J. Adiego, P. de la Fuente. Natural language compression on edge-guided text preprocessing. *Information Sciences*, vol. 181, no. 24, pp. 5387–5411, 2011.

[21] J. T. Zhou, O. C. Au, X. P. Fan, P. H. W. Wong. Secure Lempel-Ziv-Welch (LZW) algorithm with random dictionary insertion and permutation. In *Proceedings of IEEE International Conference on Multimedia and Expo*, IEEE, Hannover, pp. 245–248, 2008.

[22] A. Carus, A. Mesut. Fast text compression using multiple static dictionaries. *Information Technology Journal*, vol. 9, no. 5, pp. 1013–1021, 2010.

[23] U. S. Bhadade, A. I. Trivedi. Lossless text compression using dictionaries. *International Journal of Computer Applications*, vol. 13, no. 8, pp. 27–34, 2011.

[24] J. Tadrat, V. Boonjing. An experiment study on text transformation for compression using stoplists and frequent words. In *Proceedings of the 5th International Conference on Information Technology: New Generations, ITNG*, IEEE Computer Society, Las Vegas, NV, USA, pp. 709–713, 2008.

[25] M. B. Begum, Y. Venkataramani. LSB based audio steganography based on text compression. *Procedia Engineering*, vol. 30, pp. 703–710, 2012.

[26] G. Navarro, V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, vol. 39, no. 1, Article No. 2, 2007.

[27] K. Sayood. *Introduction to Data Compression*, 2nd ed., San Francisco: Morgan Kaufmann Publishers, pp. 39–61, 149–154, 2000.

[28] The canterbury corpus. [Online], Available: http://corpus.canterbury.ac.nz/descriptions/.

M. Baritha Begum graduated from National Institute of Technology (NIT), India in 2000. She received the M. Eng. degree from Saranathan College of Engineering, India in 2008, and is pursuing Ph. D. degree in engineering. She is currently an assistant professor at the Saranathan college of Engineering, India.

Her research interests include text processing, image processing and information security.

E-mail: baritha_m@yahoo.co.in (Corresponding author)

Y. Venkataramani graduated from Indian Institute of Technology (IIT), India in 1967. He received the M. Tech. degree from IIT Madras, India in 1972, and the Ph. D. degree from IIT Kanpur, India in 1982. He has 34 years of teaching experience in NIT. He is currently the dean (R&D) of Saranathan College of Engineering, India.

His research interests include speech processing, Image processing and networks.

E-mail: deanrd44@gmail.com