International Journal of Computer Applications (0975 – 8887)
Volume 57– No.19, November 2012

A Novel Approach of Speedy-Highly Secured Data Transmission using Cascading of PDLZW and Arithmetic Coding with Cryptography

Sankalp Prakash
Research Scholar
State Govt. Technical Education,
Jaipur (Rajasthan)

Mridula Purohit
Reader, Dept. of Mathematics
Vivekanand Institute of Tech. (East),
Jaipur (Rajasthan)

Abhishek Raizada
Research Scholar
Jaipur (Rajasthan)

ABSTRACT

The spread of computing has led to an outburst in the volume of data in the communication world. The paper proposes the cascading of two algorithms PDLZW and Arithmetic Coding with cryptography. With the hierarchical parallel dictionary set, the search time can be reduced significantly and all these dictionaries are operated independently. While the results generated by Arithmetic Coding are close to the optimal value. Since cascaded compression may achieve the higher compression ratio for the file but does not provide required security aspects. Therefore, the advantage of cryptography has been taken by XORing the compressed data with the one-time key, providing compression and security simultaneously. This paper proposes a new system JDCE, where compression is provided to the data twice before encrypted it for getting it ready for transmission.

General Terms

Arithmetic Coding, Cascaded Compression, Cryptography, Parallel Dictionary LZW (PDLZW), One-Time Pad

Keywords

Arithmetic Coding, Cascaded Compression, Cryptography, Parallel Dictionary LZW (PDLZW), One-Time Pad

1. INTRODUCTION

The field of Information Technology has grown up abruptly in last decade, which pivots around data/message transmission. There are two important factors to be considered firstly transmission speed i.e. time taken from source to destination and secondly data security or integrity of the data which means to ensure that the receiver is receiving the original message send by the sender. The length of the data/ message may vary from a few bytes to gigabytes. Therefore it becomes crucial to compress and cipher the data/message before transmission. Data compression is the process of encoding the data, so that fewer bits will be needed to represent the original data whereby the size of the data is reduced. Cryptography is an art of protecting information by transforming it (encrypting it) into an unreadable format, called cipher text. Only those who possess a secret *key* can decipher (or decrypt) the message into plain text. So, a new system JDCE (Joint Double Compression and Encryption) has been proposed in which compression is attained by cascading of PDLZW and Arithmetic Coding and then result of the compression is encrypted to provide rapid transmission and triple layer security.

Data compression has important applications in the area of data transmission as well as data storage despite of large capacity storage devices are available these days. Therefore, there is need for an efficient way to store and transmit different type of data such as text, image, audio and video to reduce execution time and memory size [11]. The general principle of data compression algorithm on text files is to conjures up an assortment of ad hoc techniques such as compression of spaces in text to tabs, creation of special codes for common words or run length coding of picture data to produce new text file which contains the same information but with new length as small as possible [7]. The effective data compression algorithm is chosen according to some scales like: Compression Size, Compression Ratio, Compression Time and Entropy [12]. Compression size means size of new compressed file. Compression ratio refers to the percentage that results from dividing the compression size by the original uncompressed file size and then multiply it by 100. Entropy is the number that results from dividing the compression size in bits by the number of symbols in the original file and scales as bits/symbol [11]. Shannon's fundamental theorem of coding states that, given messages randomly generated from a model, it is impossible to encode them into less bits (on average) than the entropy of that model [1].

LZW and the Arithmetic Coding (AC) are two of the most appropriate compression algorithms for communication because firstly, both are adaptive i.e. they do not require a prior knowledge on the text to be compressed and secondly, during the compression process they do not require transfer of extra information from the sender to the receiver in addition to the compressed text [5]. Many researchers published comparative studies on the data compression techniques such LZW, Huffman, FLC, AC, LZ-77, etc. and found LZW and AC are the best in their own territory. LZW is appropriate when aim is raw speed rather than compression performance. The conventional LZW is a dictionary based compression so it requires large amount of processing time for adjusting and searching through the dictionary [3]. The parallel dictionary LZW (PDLZW) has designed to overcome this problem. AC has its own limitations. Though PDLZW and AC work on different principles, they can be cascaded yielding higher compression ratio while being appropriate for communication purpose, but does not provide the full security of transmitting data. The proposed mechanism will overcome this issue by incorporating cryptography with the cascading of PDLZW and AC.

2. PARALLEL DICTIONARY LZW (PDLZW)

The basic idea of dictionary based compression technique given by Lempel and Ziv as LZ-77. The main disadvantage of LZ-77 is the size of buffers is very small so in 1984, Terry Welch suggested LZW algorithm. LZW is general compression algorithm capable of working on almost any type of data [11]. LZW compression creates a table of string commonly occurring in the data being compressed, searches the table to identify the longest possible input data string that exists in the table, and replaces the actual data with references into the table. The table is formed during the compression at the same time at which the data is encoded and during decompression at the same time as the data is being decoded [8]. It can typically compress large English text to about half of the original sizes. However, the limit is imposed in the conventional LZW by the fact that once the 4K dictionary is complete, no more strings can be added; and requires large amount of processing time for adjusting and searching through the dictionary [7].

To improve the limitations of conventional LZW, the dynamic LZW (DLZW) and word-based DLZW (WDLZW) algorithms were proposed. In DLZW, the dictionary has been initialized with different combinations of characters. It is organized in hierarchical string tables. The baseline idea is to store the most frequently used strings in the shorter table, which requires fewer bits to identify the corresponding string. The tables are updated using the move-to-front and weighting system with associated frequency counter. During the compression time, after the longest matching string is recognized in the table, it is moved to the first position of its block. The table updating process is based on the least recently used (LRU) policy to ensure that frequently used strings are kept in the smaller tables. This is to minimize the average number of bits required to code a string when compare with a single table implementation [2, 3, 7].

The WDLZW algorithm is a modified version of DLZW that focuses on text compression by identifying each word in the text and make it a basic unit (symbol). The algorithm encodes the input word into literal codes and copy codes. If the search for a word has failed, it is sent out as a literal code, which is its original ASCII code preceded by other codes for identification. The copy code is the address of the matching string in the string table. However, both algorithms are too complicated. To improve this, parallel dictionary LZW (PDLZW) was proposed. Since not all entries of the DLZW dictionary contains the same word size, this leads to the need of the entire dictionary search for every character. Consequently, the PDLZW has designed to overcome this problem by partitioning the dictionary into several dictionaries of different address spaces and sizes. With the hierarchical parallel dictionary set, the search time can be reduced significantly since these dictionaries can operate independently and thus can carry out their search operation in parallel [2].

2.1 Compression Algorithm for PDLZW

The PDLZW compression algorithm is based on a parallel dictionary set that consists of *m* small variable-word-width dictionaries, numbered from *0* to *m-1*, each of which increases its word width by 1 byte (1B). More precisely, dictionary0 has 1B word width; dictionary1 has 2B, and so on. The actual size of the dictionary set used in a given application can be determined by the information correlation property of the application.

In the algorithm, two variables and one constant are used. The constant *max_dict_no* denotes the maximum number of dictionaries, excluding the first single-character dictionary (i.e., dictionary0), in the dictionary set. The variable max*_matched_dict_no* is the largest dictionary number of all matched dictionaries and the variable *matched_addr* identifies the matched address within the *max_matched_dict_no* dictionary. Each compressed codeword is a concatenation of max*_matched_dict_no* and *matched_addr*.

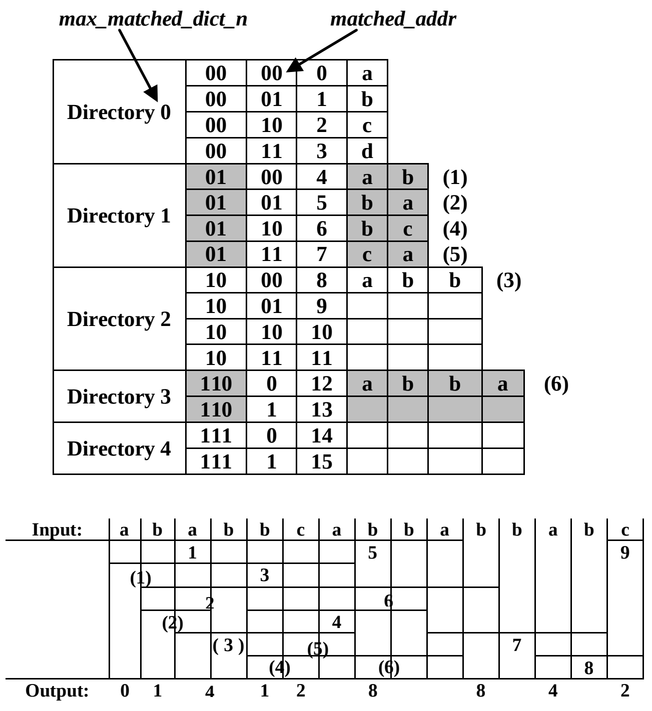
Input: The string to be compressed.

Output: The compressed code words with each being a $\log_2 k$-bit codeword, which consists of *max_matched_dict_no* and *matched_ addr*, where *k* is the total number of entries of the dictionary set.

Begin:

```
1:  Initialization.
    1.1. string-1 ← null.
    1.2. max_matched_dict_no ← max_dict_no.
    1.3. update_dict_no ← max_matched_dict_no;. update_string ← ∅{empty}.
2:  while (the input buffer is not empty) do
    2.1. Prepare next max_dict_no + 1 character for searching.{max_matched_dict_no is reset to max_dict_no initially and the dictionary number of the dictionary set counts from 0 up to a constant max_dict_no}
         2.1.1. string-2 ← read next (max_matched_dict_no+1) characters from the input buffer.
         2.1.2. string ← string-1‖string-2. (Where ‖ is the concatenation operator.)
    2.2. Search string in all dictionaries in parallel and set the max_matched_dict_no and matched_addr.
    2.3. Output the compressed codeword containing max_matched_dict_no‖matched_addr.
    2.4. if (max_matched_dict_no < max_dict_no and update_string ≠ ∅) then add the update_string to the entry point by UP[update_dict_no] of dictionary[update_dict_no].(UP[update_dict_no] is the update pointer associated with the dictionary).
    2.5. Update the update pointer of the dictionary[max_matched_dict_no + 1].
         2.5.1. UP[max_matched_dict_no + 1] = UP[max_matched_dict_no + 1] + 1
         2.5.2. if UP[max_matched_dict_no + 1] reaches its upper bound then reset it to 0. {FIFO update rule.}
    2.6. update_string ← extract out the first (max_matched_dict_no + 2) bytes from string; update_dict_no ← max_matched_dict_no + 1.
    2.7. string-1 ← shift string out the first (max_matched_dict_no + 1) bytes.
End {End of PDLZW Compression Algorithm.}[2]
```

Here is an example to illustrate the operation of the PDLZW compression algorithm. It is assumed that the alphabet set ∑ is {a,b,c,d} and the input string is ababbcabbabbabc . The address space of the dictionary set is 16. The dictionary set initially contains only all single characters: a,b,c and d. The input string is grouped together by characters. These groups are denoted by a number with or without parenthesis. The number without parenthesis denotes the order to be searched of the group in the dictionary set and the number with parenthesis denotes the order to be updated of the group in the dictionary set. After the algorithm exhausts the input string, the contents of the dictionary set and the compressed output code words will be {a,b,c,d,ab,ba,bc,ca,abb,,,,,abba,,,} and {0,1,4,1,2,8,8,4,2} respectively [2, 3].



Directory 0	00	00	0	a				
	00	01	1	b				
	00	10	2	c				
	00	11	3	d				
Directory 1	01	00	4	a	b	(1)		
	01	01	5	b	a	(2)		
	01	10	6	b	c	(4)		
	01	11	7	c	a	(5)		
Directory 2	10	00	8	a	b	b	(3)	
	10	01	9					
	10	10	10					
	10	11	11					
Directory 3	110	0	12	a	b	b	a	(6)
	110	1	13					
Directory 4	111	0	14					
	111	1	15					

Figure 1: Example to illustrate the operation of PDLZW compression algorithm.

2.2 PDLZW Decompression Algorithm:

To recover the original string from the compressed one, reverse the operation of the PDLZW compression algorithm. This operation is called the PDLZW decompression algorithm. By decompressing the original substrings from the input compressed code words, each input compressed codeword is used to read out the original substring from the dictionary set. To do this without loss of any information, it is necessary to keep the dictionary sets used in both algorithms, the same contents. Hence, the substring concatenated of the last output substring with its first character is used as the current output substring and is the next entry to be inserted into the dictionary set. The PDLZW decompression algorithm has three variables and one constant. As in the PDLZW compression algorithm, the constant *max_dict_no* denotes the maximum number of dictionaries in the dictionary set. The variable *last_dict_no* memorizes the dictionary address part of the previous codeword. The variable *last_output* keeps the decompressed substring of the previous codeword, while the variable *current_output* records the current decompressed substring. The output substring always takes from the *last_output* that is updated by *current_output* in turn.

Input: The compressed codewords with each containing $\log_2 k$-bits, where *k* is the total number of entries of the dictionary set.
Output: The original string.
Begin:

```
1:  Initialization.
    1.1. if (input buffer is not empty) then
         current_output ← empty; last_output ← empty;
         addr ← read next $\log_2 k$-bit codeword from input buffer. {Where codeword = dict_no ‖ dict_addr and ‖ is the concatenation operator.}
    1.2. if (dictionary[addr] is defined) then
         current_output ← dictionary[addr];
         last_output ← current_output;
         output ← last_output;
         update_dict_no ← dict_no[addr] + 1.
2:  while (the input buffer is not empty) do
    2.1. addr ← read next log2k-bit codeword from input buffer.
    2.2. {output decompressed string and update the associated dictionary.}
         2.2.1. current_output ← dictionary[addr].
         2.2.2. if (max_dict_no ≥ update_dict_no) then
                add (last_output ‖ the first character of current_output) to the entry pointed by UP[update_dict_no] of dictionary[update_dict_no].
         2.2.3. UP[update_dict_no]←UP[update_dict_no]+ 1.
         2.2.4. if UP[update_dict_no] reaches its upper bound then reset it to 0.
         2.2.5. last_output ← current_output;
                Output ← last_output;
                update_dict_no dict_no(addr) + 1.
End {End of PDLZW Decompression Algorithm.}
```

The operation of the PDLZW decompression algorithm can be illustrated by the following example. Assume that the alphabet set ∑ is and input compressed codewords are {0, 1, 4, 1, 2, 8, 8, 4, 2}. Initially, the dictionaries numbered from 1 to 3 shown in Figure 1 are empty. By applying the entire input compressed codewords to the algorithm, it will generate the same content as is shown in Figure 1 and output the decompressed {a, b, ab, b, c, abb, abb, ab, c}substring [2, 3].

3. Arithmetic Coding Algorithm (AC)

In Arithmetic Coding, method for lossless data compression, a message is represented by an interval of real numbers between 0 and 1. As the message becomes longer, the interval needed to represent it becomes smaller, and the number of bits needed to specify that interval grows. Successive symbols of the message reduce the size of the interval in accordance with the symbol probabilities generated by the model. The more likely symbols reduce the range by less than the unlikely symbols and hence add fewer bits to the message [1]. This method is adaptive and does not need the probabilities of the symbols in the input in advance. These probabilities could be dynamically updated as input is read, and mapped into the interval [5].

The AC has advantages over Huffman Coding method (HC). HC indeed achieves "minimum redundancy." In other words, it performs optimally if all symbol probabilities are integral powers of ½. But this is not normally the case in practice; indeed, Huffman coding can take up to one extra bit per symbol. The worst case is realized by a source in which one symbol has probability approaching unity. Symbols emanating from such a source convey negligible information on average, but require at least one bit to transmit [1]. Arithmetic coding dispenses with the restriction that each symbol must translate into an integral number of bits, thereby coding more efficiently. It actually achieves the theoretical entropy bound to compression efficiency for any source

[13].In this article, the coding algorithm is adapted from an algorithm originally presented in C by Mark Nelson [16].

In order to construct the output number, the symbols being encoded have to have a set probabilities assigned to them. In general, using AC depends on creating a statistical model of the data. For example, to encode the random message “BILL GATES”, would have a probability distribution. Once the character probabilities are known, the individual symbols need to be assigned a range along a *probability line*, which is nominally 0 to 1. It doesn’t matter which characters are assigned which segment of the range, as long as it is done in the same manner by both the encoder and the decoder. The nine character symbol set use here would look like this:

Table 1. Initial Assignment

Character	Probability	Range
SPACE	1/10	0.00 – 0.10
A	1/10	0.10 – 0.20
B	1/10	0.20 – 0.30
E	1/10	0.30 – 0.40
G	1/10	0.40 – 0.50
I	1/10	0.50 – 0.60
L	2/10	0.60 – 0.80
S	1/10	0.80 – 0.90
T	1/10	0.90 – 1.00

Each character is assigned the portion of the 0-1 range that corresponds to its probability of appearance. Note also that the character “owns” everything up to, but not including the higher number. So the letter ‘T’ in fact has the range 0.90 – 0.9999….The most significant portion of an arithmetic coded message belongs to the first symbol to be encoded. When encoding the message “BILL GATES”, the first symbol is “B”. In order for the first character to be decoded properly, the final coded message has to be a number greater than or equal to 0.20 and less than 0.30. To encode this number is to keep track of the range that this number could fall in. So after the first character is encoded, the low end for this range is 0.20 and the high end of the range is 0.30.

After the first character is encoded, range for output number is now bounded by the low number and the high number. What happens during the rest of the encoding process is that each new symbol to be encoded will further restrict the possible range of the output number. The next character to be encoded, ‘I’, owns the range 0.50 through 0.60. If it was the first number in the message, set low and high range values directly to those values. But ‘I’ is the second character. So ‘I’ owns the range that corresponds to 0.50-0.60 in the new sub range of 0.2 – 0.3. This means that the new encoded number will have to fall somewhere in the 50th to 60th percentile of the currently established range. Applying this logic will further restrict number to the range 0.25 to 0.26.

The algorithm to accomplish this for a message of any length is shown below:

```
Set low to 0.0
Set high to 1.0
While there are still input symbols do
    get an input symbol
    code_range = high - low.
    high = low + range*high_range(symbol)
    low = low + range*low_range(symbol)
End of While
output low
```

So the final low value, 0.2572167752 will uniquely encode the message “BILL GATES” using present encoding scheme.

Given this encoding scheme, it is relatively easy to see how the decoding process will operate. Find the first symbol in the message by seeing which symbol owns the code space that the encoded message falls in. Since the number 0.2572167752 falls between 0.2 and 0.3, the first character must be “B”. So remove the “B” from the encoded number. Since the low and high ranges of B, their effects can be removed by reversing the process that put them in. First, subtract the low value of B from the number, giving 0.0572167752. Then divide by the range of B, which is 0.1. This gives a value of 0.572167752. Now it can be calculated where that lands, which is in the range of the next letter, “I” and so on.

The algorithm for decoding the incoming number looks like this:

```
get encoded number
Do
    find symbol whose range straddles the encoded number
    output the symbol
    range = symbol low value - symbol high value
    subtract symbol low value from encoded number
    divide encoded number by range
until no more symbols
```

In summary, the encoding process is simply one of narrowing the range of possible numbers with every new symbol. The new range is proportional to the predefined probability attached to that symbol. Decoding is the inverse procedure, where the range is expanded in proportion to the probability of each symbol as it is extracted.

3.1 Practical Matters

The process of encoding and decoding a stream of symbols using AC is not too complicated. But at first glance, it seems completely impractical. Most computers support floating point numbers of up to 80 bits or so. As it turns out, AC is the best accomplished using standard 16-bit and 32-bit integer math. No floating point math is required, nor would it help to use it. What is used instead is an incremental transmission scheme, where fixed size integer state variables receive new bits in at the low end and shift them out the high end, forming a single number that can be as many bits long as are available on the computer’s storage medium.

The previous section has shown how the algorithm works by keeping track of a high and low number that bracket the range of the possible output number. When the algorithm first starts up, the low number is set to 0.0, and the high number is set to 1.0. The first simplification made to work with integer math is to change the 1.0 to 0.999…., or .111… in binary.

In order to store these numbers in integer registers, first justify them so the implied decimal point is on the left hand side of the word. Then load as much of the initial high and low values as will fit into the word size. The implementation uses 16-bit unsigned math, so the initial value of high is 0xFFFF, and low is 0. The high value continues with FFs forever, and low continues with 0s forever, so those extra bits can be shifted in with impunity when they are needed. If imagine the “BILL

GATES" example in a 5 digit register, the decimal equivalent of setup would look like this:

HIGH: 99999
LOW: 00000

In order to find the new range numbers, it is needed to apply the encoding algorithm from the previous section. First calculate the range between the low value and the high value. The difference between the two registers will be 100000, not 99999. This is because it is assumed the high register has an infinite number of 9′s added on to it, so need to increment the calculated difference. Then compute the new high value using the formula from the previous section:

high = low + high_range(symbol)

In this case the high range was 0.30, which gives a new value for high of 30000. Before storing the new value of high, it is needed to decrement it, once again because of the implied digits appended to the integer value. So the new value of high is 29999. The calculation of low follows the same path, with a resulting new value of 20000. So now high and low look like this:

HIGH: 29999 (999…)
LOW: 20000 (000…)

At this point, the most significant digits of high and low match. Due to the nature of the algorithm high and low can continue to grow closer to one another without quite ever matching. This means that once they match in the most significant digit, that digit will never change. So the output can be obtained of that digit as the first digit of the encoded number. This is done by shifting both high and low left by one digit, and shifting in a 9 in the least significant digit of high.

As this process continues, high and low are continually growing closer together, and then shifting digits out into the coded word.

This scheme works well for incrementally encoding a message. There is enough accuracy retained during the double precision integer calculations to ensure that the message is accurately encoded. However, there is potential for a loss of precision under certain circumstances.

The cumulative frequency table is stored in frequency orders to minimize the number of updates to it after every symbol is processed. Translation tables of character to index and index to character are used to simplify the process of sorting the cumulative frequency table. These translation tables are also adjusted whenever the cumulative frequency table updated. To overcome the overflow and underflow problems of the integer arithmetic, frequencies are scaled down by a normalization factor at regular interval [5].

4. Cascading of PDLZW and AC

Each compression algorithm has its own limitations and this is true with PDLZW and AC too. So solution to overcome the weakness of one compression technique has been found by combining it to another compression technique. This process is known as cascaded compression. In this process, the raw data is given to the PDLZW encoding algorithm. The output of the PDLZW is given to the AC for further compression. The decompression process is totally reversing [9]. Figure 2 shows the block diagram of cascading of PDLZW and AC.

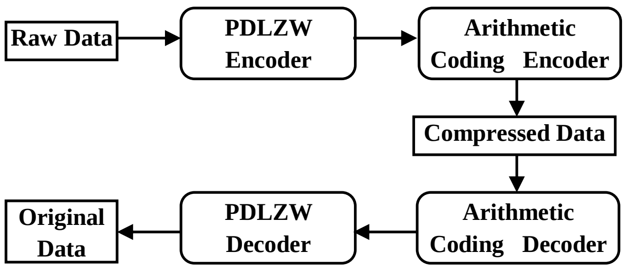


Figure 2: Block Diagram of Cascading of PDLZW and Arithmetic Coding.

Table 2 shows the results of implementation of Cascading of PDLZW and Arithmetic Coding on various text files.

Table 2. Implementation Results of Cascading of PDLZW and AC

File Name	Original Size (Bytes)	After Cascading Compression	
		File Size (Bytes)	Compression Ratio
new.txt	261	97	62.83
file1.txt	12288	948	92.88
file2.txt	68608	2560	96.26
file3.txt	88064	2867	96.74
main.txt	872448	4505	94.83

5. CRYPTOGRAPHY

In this fast-paced technological world the importance of information and communication systems is escalating with the increasing significance and quantity of data that is transmitted. Unfortunately systems and data are increasingly vulnerable to a variety of threats, such as unauthorized access and use, misappropriation, alteration, and destruction. Cryptography is the foundation of all data as well as information security aspects. Classical cryptosystems is very easy to understand, easily implemented and very easy to be broken. New forms of cryptography came after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium.

In the present scenario the cryptographic techniques have become the immediate solution to protect information against third parties. These techniques required that data and information should be encrypted with some sort of mathematical algorithm where only the party that shares the information could possible decrypt to use the information. Within the context of any communication, there are some specific security requirements includes (1) Authentication which means the process of providing one's identity; (2) Confidentiality that ensures no one can read the message except the intended receiver; (3) Integrity for assuring the receiver, the received message has not been altered in any way from the original and (4) Non-repudiation is a mechanism to prove that the sender really send this message [19].

Cryptography itself splits into here main branches:

(1) Symmetric (or Private-Key) Algorithm: two parties have an encryption and decryption method for which they share a secret key.

(2) Asymmetric (or Public-Key) Algorithm: a user possesses a secret key (private key) as in symmetric cryptography but also a public key.

(3) Hybrid Cryptography: symmetric and asymmetric algorithms (and often also hash functions) are all used together.

Further the symmetric cryptography can be divided in stream ciphers and block ciphers. Stream ciphers encrypt bits individually. This is achieved by adding a bit from a key stream to a plain bit. Block ciphers encrypt an entire block of plain text bits at a time with the same key. In stream ciphers each bit x_i is encrypted by adding a secret key stream bit s_i modulo 2. If arithmetic modulo 2 is done, the only possible values are 0 and 1 because if a number is divided by 2 the only possible remainders are 0 or 1. If the truth table of modulo2 addition is drawn then it is found that it is similar to the *exclusive-OR (XOR)* gate. So the XOR operation plays a major role in modern cryptography.

Though how good the compression techniques are, they do not provide security of data/message from intruders, hackers and code-breakers. So the compressed data must be encrypted to provide authenticity, confidentiality, integrity and non-repudiation.

6. NEW PROPOSED 3-TIER SYSTEM (JDCE)

This paper proposes a new 3-tier system that provides double compression with cryptography for speedy-highly secured data transmission. In this system, first compress the raw data using PDLZW encoder (Tier-1), and the output of this encoder is redirected to AC encoder (Tier-2). That is how the highly compressed data is achieved. Since the compression technique are not secured, so the cryptography has been integrated with the compression techniques. The output of Tier-2 is a number which is actually the compressed data. In this paper a truly unbreakable cipher: the One-Time Pad (OTP) is being used [21]. A stream cipher for which (1) the key stream s_0, s_1,s_2,… is generated by a true random number generator (2) the key stream is only known to the legitimate communicating parties, and (3) every key stream bit s_i is only used once, is called a One-Time Pad. The OTP is unconditionally secure [21]. Obtained number from Tier-2 is encrypted using private key encryption technique by XORing it with OTP which is a random key as long as message (Tier-3). This one time key is used to encrypt and decrypt a single message, and then discarded. Each new message requires a new key of same length as a new message. The output is highly compressed and secured. The Figure 2 has been modified here to achieve JDCE as Figure 3.

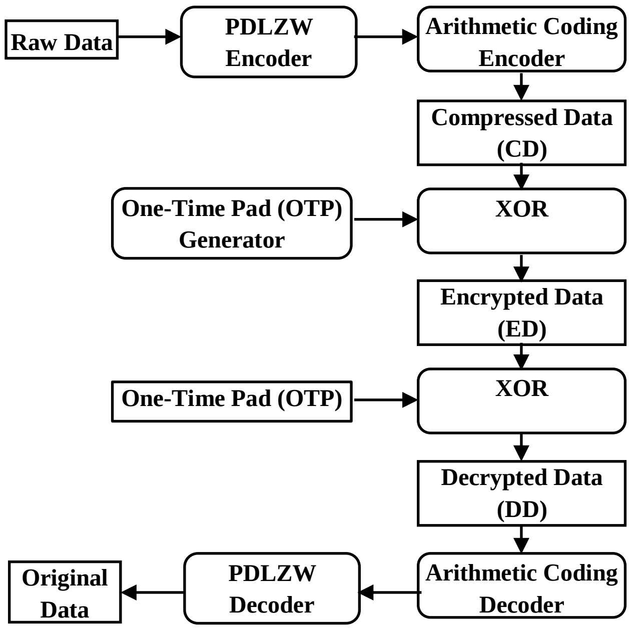


Figure 3: Block Diagram of Proposed 3-Tier System (JDCE).

7. CONCLUSION

The proposed technique 3-Tier System (JDCE) provides an excellent integration of data compression by cascading of PDLZW and Arithmetic Coding algorithms along with the cryptography to enhance the data security and transfer rate during data communication. In this technique the data size can be reduced by using cascaded compression technique and after that compressed data can be encrypted to provide the data security. The present network scenario demands exchange of information with reduction in both space requirement for data storage and time for data transmission along with security. The proposed technique fulfils all such requirements as this technique use the concept of data compression and encryption. This paper can be extended for the storage of files.

8. REFERENCES

[1] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. Commun. ACM, 30(6):520–540, June 1987.

[2] Ming-Bo Lin, Jang-Feng Lee, and Gene Eu Jan. A lossless data compression and decompression algorithm and its hardware architecture. IEEE Trans. Very Large Scale Integr. Syst., 14(9):925–936, September 2006.

[3] Ming-Bo Lin. A hardware architecture for the lzw compression and decompression algorithms based on parallel dictionaries. J. VLSI Signal Process. Syst., 26(3):369–381, November 2000.

[4] Mark Nelson and Jean-Loup Gailly. The Data Compression Book. M&T Books, 2nd edition, 1996.

[5] Yehoshua Perl, V. Maram, and N. Kadakuntla. The cascading of the LZW compression algorithm with arithmetic coding. In James A. Storer and John H. Reif, editors, Data Compression Conference, pages 277–286. IEEE Computer Society, 1991.

[6] Archana V. Nair. S, G. Kharmega Sundararaj and T. Sudarson Rama Perumal. Simultaneous compression and encryption using arithmetic coding with randomized bits. International Journal of Computer Technology and Electronics Engineering, 2:38–42, April 2012.

[7] P. Vichitkraivin and Orachat Chitsobhuk. An Improvement of PDLZW implementation with a Modified WSC Updating Technique on FPGA. World Academy of Science, Engineering and Technology, 2009.

[8] David Salomon. Data compression-The Complete Reference, 4th Edition. Springer, 2007.

[9] Nirali Thakkar and Malay Bhatt. Cascading of the PDLZW compression algorithm with arithmetic coding. International Journal of Computer Applications, 46(16):21–24, May 2012. Published by Foundation of Computer Science, New York, USA.

[10] Ajit Singh and Rimple Gilhotra. Data security using private key encryption system based on arithmetic coding, May 2011.

[11] Haroon Altarawneh and Mohammad Altarawneh. Data compression techniques on text files: A comparison study. International Journal of Computer Applications, 26(5):42–54, July 2011. Published by Foundation of Computer Science, New York, USA.

[12] Debra A. Lelewer and Daniel S. Hirschberg. Data compression. ACM Comput. Surv., 19(3):261–296, September 1987.

[13] Jiantao Zhou, Oscar C. Au, Xiaopeng Fan, and Peter H. W. Wong. Joint security and performance enhancement for secure arithmetic coding. In ICIP, pages 3120–3123, 2008.

[14] Raj S. Katti, Sudarshan K. Srinivasan, and Aida Vosoughi. On the security of randomized arithmetic codes against ciphertext-only attacks. IEEE Transactions on Information Forensics and Security, 6(1):19–27, 2011.

[15] Helen A. Bergen and James M. Hogan. A chosen plaintext attack on an adaptive arithmetic coding compression algorithm. Computers & Security, pages 157–167, 1993.

[16] Mark R. Nelson. Arithmetic coding and statistical modeling: achieving higher compression ratios. Dr. Dobb's J., 16(2):16–ff., December 1990.

[17] Whitfield Diffie & Martin E. Hellman. Privacy and authentication: An introduction to cryptography, 1979.

[18] Tarek M Mahmoud, Bahgat A. Abdel-latef, Awny A. Ahmed, Ahmed M Mahfouz, Tarek M. Mahmoud, Bahgat A. Abdel-latef, Awny A. Ahmed, and Ahmed M. Mahfouz. Hybrid compression encryption technique for securing sms. International Journal of Computer Science and Security, 2009.

[19] Gary C. Kessle. An overview of cryptography. [Online]. http://www.garykessler.net/library/crypto.html.

[20] William Stallings. Cryptography and Network Security: Principles and Practice. Prentice Hall Press, Upper Saddle River, NJ, USA, 5th edition, 2010.

[21] Christof Paar and Jan Pelzl. Understanding Cryptography - A Textbook for Students and Practitioners. Springer, 2010.