Performing Compression and Encryption Simultaneously in Data Transmission

Hind Kuhdair Abbass
Computer Science Department,
College of Education (Ibn - Haitham),
University of Baghdad

ABSTRACT

The main goals of this paper is to preserve data confidentiality, integrity and reduce the redundancy in data representation when transmitting data across a public computer network. The basic of this research is to develop a method for combining the operation of compression and encryption on the same set of data to perform these two operations simultaneously instead of separately. This is achieved through embedding encryption into compression algorithms since both cryptographic ciphers and entropy coders bear certain resemblance in the sense of secrecy.

The proposed system is implemented using Microsoft visual C#.NET programming language.

١- INTRODUCTION

Data compression and ciphering are essential features when digital data is stored or transmitted over insecure channels. Usually, we apply two sequential operations: first, we apply data compression to save disk space and to reduce transmission costs, and second, data ciphering to provide confidentiality. This solution works fine to most applications, but we have to execute two expensive operations, and if we want to access data, we must first decipher and then decompress the ciphertext to restore information [١].

The major problem existing with the current compression and encryption methods is the large amount of processing time required by the computer to perform the tasks. To lessen this problem, combine the two processes into one. To combine the two processes, new techniques will be introduced in this paper. The idea is confusing to Initial values of the LZW dictionary. Before one begins to submit the proposed methods, more security methods will be reviewed for generating random numbers. Blum Blum Shub Generator was chosen as a cryptographically secure pseudorandom bit generator (CSPRBG). Some additions are made for this method, but without prejudice to the substance of the method. The purpose of this addition is to be appropriate for the work of secure LZW at the same time.

٢- Cryptographically Secure Pseudo-Random Sequences

Cryptographic applications demand much more of a pseudo-random-sequence generator than do most other applications. Cryptographic randomness doesn't mean just statistical randomness, although that's part of it. For a sequence to be cryptographically secure pseudo-random, it must also have this property: It is unpredictable. It must be computationally infeasible to predict what the next random bit will be, given complete knowledge of the algorithm or hardware generating the sequence and all of the previous bits in the stream [٢].

٣- Blum Blum Shub Generator

A popular approach to generating secure pseudorandom number is known as the Blum Blum Shub (BBS) generator named for its developers. It has perhaps the strongest public proof of its cryptographic strength. The procedure is done as follows. First, choose two large prime numbers, p and q, that both have a remainder of ٣ when divided by ٤. That is, $p \equiv q \equiv ٣ \pmod{٤}$. This notation, simply means that (p mod ٤) = (q mod ٤) = ٣. For example, the prime numbers ٧ and ١١ satisfy $٧ \equiv ١١ \equiv ٣ \pmod{٤}$. Let $n = p * q$. Next, choose a random number s, such that s is relatively prime to n; this is equivalent to saying that neither p nor q is a factor of s. Then the BBS generator produces a sequence of bits Bi and Li according to the following algorithm: [٣]

Algorithm ١: Blum Blum Shub (BBS) generator

INPUT: Two large prime numbers, p and q and a random number s.

OUTPUT: Secure pseudorandom number.

```
$X_٠ = s^٢ \bmod n$
For i = ١ to ∞
   $X_i = (X_{i-١})^٢ \bmod n$
   $B_i = X_i \bmod ٢$
   $L_i = X_i \bmod ٢٥٦$
```

Thus, the least significant bit is taken at each iteration and stored in Bi. The Bi is significant for determine the number of null strings which is added to LZW initial dictionary to confuse it. The last step in the above algorithm is suggested for secure LZW work. Table (١), illustrates an example of BBS operation. Here, n = ١٩٢٢٦٤٩ = ٣٨٣ x ٥٠٣ and the seed s = ١٠١٣٥٥.

Table ١: Example Operation of BBS Generator

i	X_i	B_i	L_i
٠	٢٠٧٤٩	١	٣١
١	١٤٣١٣٥	١	٧
٢	١٧٧٦٧١	٠	٢٤
٣	٩٧٠٤٨	٠	١٣٦
٤	٨٩٩٩٢	١	٢٢٧
٥	١٧٤٠٥١	١	٩

٦	٨٠٦٤٩	١	٩٥
٧	٤٥٦٦٣	٠	٦٦
٨	٦٩٤٤٢	٠	١٤
٩	١٨٦٨٩٤	٠	١٥٠
١٠	١٧٧٠٤٦	٠	١٩٤
١١	١٣٧٩٢٢	١	٣٩
١٢	١٢٣١٧٥	٠	١٨٢
١٣	٨٦٣٠	٠	٢١٠
١٤	١١٤٣٨٦	١	١٥
١٥	١٤٨٦٣	١	١٥١
١٦	١٣٣٠١٥	١	٨١
١٧	١٠٦٠٦٥	٠	٤٦
١٨	٤٥٨٧٠	١	٢١١
١٩	١٣٧١٧١	٠	١٨٨
٢٠	٤٨٠٦٠	١	١٩
٢١	٩٤٧٣٩	٠	٤
٢٢	١٥٣٨٦٠	٠	١٦
٢٣	١٩٠٤٨٠	١	٨٩
٢٤	٨٠٩٨٥	١	٢١

The BBS is referred to as a cryptographically secure pseudorandom bit generator. A CSPRBG is defined as one that passes the next-bit test, which, in turn, is defined as follows: A pseudorandom bit generator is passed to the next-bit test if there is not a polynomial-time algorithm that, an input of the first k bits of an output sequence, can predict the (k + ١)st bit with probability significantly greater than ١/٢. In other words, giving the first k bits of the sequence, there is not a practical algorithm that can even allow you to state that the next bit will be ١ (or ٠) with probability greater than ١/٢. For all practical purposes, the sequence is unpredictable. The security of BBS is based on the difficulty of factoring n. That is, giving n, its two prime factors p and q are needed to determine [٣].

٤- Basic Lempel-Ziv-Welch

Many programs use a version of Lempel-Ziv created by Terry Welch in ١٩٨٤.This version is pretty simple and easy to code, so it is frequently included in many simple compression schemes. It is also relatively good, although it takes its time building up the dictionary [٤].

Its main feature is eliminating the second field of a token, and LZW token consists of just a pointer to the dictionary.

(A) LZW Encoding

To best understanding of LZW will temporarily forget that the dictionary is a tree, and will think of it as an array of variable-size strings. In theory, the dictionary is built from scratch and initially is empty. However, if the alphabet of a source is known , the alphabet and other commonly used symbols are stored as the first ٢٥٦ entries in the dictionary. In other words, the dictionary usually contains ٢٥٦ entries (e.g. ASCII codes) of single characters initially.

The principle of LZW is that the encoder inputs symbols are one by one and they accumulate them in a string I. After each symbol is input and is concatenated to I, the dictionary searches for string I. As long as I is found in the dictionary, the process continues. At a certain point, adding the next symbol x causes the search to fail; string I is in the dictionary but string Ix (symbol x concatenated to I) is not. At this point the encoder is:
(١) Outputs the dictionary pointer that points to string I.
(٢) Saves string Ix (which is now called a phrase) in the next available dictionary entry.
(٣) Initializes string I to symbol x.

Algorithm ٢: LZW Encoding

```
١- I ← " "
٢- While not EOF do
٣-     x ← read_next_character ( )
٤-     If   I + x   is in the dictionary then
٥-          I ← I + x
٦-     Else
٧-         Output the dictionary index for word
٨-         Add   I + x   to the dictionary
٩-         I ← x
١٠-    End if
١١- End while
١٢- Output the dictionary index for word
```

(B) LZW Decoding

In order to understand how the LZW decoder works, at first the three steps should be recall the encoder performs each time it writes something on the output stream. The decoder starts with the first entries of its dictionary initialized to all the symbols of the alphabet (normally ٢٥٦ symbols). It then reads its input stream (which consists of pointers to the dictionary) and uses each pointer to retrieve uncompressed symbols from its dictionary and write them on its output stream. It also builds its dictionary in the same way as the encoder.

In each decoding step after the first, the decoder inputs the next pointer, retrieves the next string J from the dictionary, writes it on the output stream, isolates its first symbol x, and saves string Ix in the next available dictionary entry (after checking to make sure string Ix is not already in the dictionary). The decoder then moves J to I and is ready for the next step.

٥- The Proposed Secure LZW Coding

People often find that LZW algorithms are easier to understand and they are the most popular ones. Therefore, try to make LZW coding secure by maintaining its clarity to be understood [٥].

In a codebook type of implementation such as the LZW compression, the dictionary consists of strings of characters that have been processed. Firstly, it contains all strings of length ١ in alphabetical order. In this case, at first confuse the initial dictionary by shuffling all strings of length ١ plus a small number of null strings which is determined by the first five bit generated by the cryptographically secure pseudo random bit generator. This means the maximum number of added nulls is ٣٢. The purpose of including null strings in the shuffling is to make chosen plaintext attacks more difficult. Moreover, it cripples any unauthorized decompression. Based on the previous discussion, encoding input consists of the following steps:

Algorithm ٣: Secure LZW Encoding Ideas

INPUT: Symbols one by one from source file.

OUTPUT: The code for input string (only the numbers are output, not the strings in parentheses).

Step ١: Initialize dictionary to contain one entry for each byte.

Step ٢: The encryption key is used to initialize a cryptographically secure pseudo random bit and number generator.

- The first five bits represent the number of added nulls
- The pseudo random number generated (٠-٢٥٥) is representing the location where the nulls will insert in the LZW initial dictionary.

Step ٣: Initialize the encoded string with the first byte of the input stream.

Step ٤: Read the next byte from the input stream.

Step ٥: If the byte is an EOF, go to step ٨.

Step ٦: If concatenating the byte to the encoded string produces a string that is in the dictionary:

- concatenate the byte to the encoded string
- go to step ٤

Step ٧: If concatenating the byte to the encoded string produces a string that is not in the dictionary:

- add the new sting to the dictionary
- write the code for the encoded string to the output stream
- set the encoded string equal to the new byte
- Go to step ٤

Step ٨: Write out code for encoded string and exit.

٦- Encoding Explanation with Example

١- Add all ASCII characters in the table from ٠-٢٥٥. Let the output of CSPRBG in (table ٤,٤) be dependable in this example.

٢- Determine number of added nulls: Take the first five bits of Bi from CSPRBG and convert it to decimal number: $(١١oo١)_٢ \equiv (٢٥)_{١٠} \equiv$ number of added nulls.

٣- Determine the location of added nulls in LZW initial dictionary: After knowing the number of added nulls, take pseudorandom numbers from CSPRBG output (Li). These pseudorandom numbers represent the position of added nulls in the LZW initial dictionary. In this example, take the first twenty five pseudorandom numbers from (Li).

Cryptographically secure pseudorandom numbers in this example are: [٣١, ٧, ٢٤, ١٣٦, ٢٢٧, ٩, ٩٥, ٦٦, ١٤, ١٥٠, ١٩٤, ٣٩, ١٨٢, ٢١٠, ١٥, ١٥١, ٨١, ٤٦, ٢١١, ١٨٨, ١٩, ٤, ١٦, ٨٩, and ٢١]

٤- Confuse the LZW initial dictionary based on the introduced information from above steps. The secure LZW will depend on the confused dictionary to coding text data.

٥- Perform the reminder LZW processing steps in conventionality way.

Sample string used to demonstrate the algorithm is illustrated in table (٢). You can see that the first pass through the loop, a check is performed to see if the string "/A" is in the table. Since it is not, the code for '/' is output, and the string "/A" is added to the table. Since we have ٢٥٦ characters plus specific number of nulls (for security purpose) already defined for codes ٠-٢٨٠ (٢٥٦ char+٢٥ nulls =٢٨١ code), the first string definition can be assigned to code ٢٨١. After the third letter, 'B', has been read in the second string code, "AB" is added to the table, and the code for letter 'A' is output. This continues until in the second word, the characters '/' and 'A' are read in, matching string number ٢٨١. In this case, the code ٢٨١ is output, and a three character string is added to the string table. The process continues until the string is exhausted and all of the codes are output.

Input String = /ABC/AB/ABB/ABD/ABE							
Character Input	**In Dic?**	**New Entry**	**Code Output**	**Character Input**	**In Dic?**	**New Entry**	**Code Output**
/	Y			/ABB	N	٢٨٧(/ABB)	/AB(٢٨٥)
/A	N	٢٨١(/A)	/(٥٩)	B	Y		
A	Y			B/	Y		
AB	N	٢٨٢(AB)	A(٧٨)	B/A	N	٢٨٨(B/A)	B/(٢٨٦)
B	Y			A	Y		
BC	N	٢٨٣(BC)	B(٧٩)	AB	Y		
C	Y			ABD	N	٢٨٩(ABD)	AB(٢٨٢)
C/	N	٢٨٤(C/)	C(٨٠)	D	Y		
/	Y			D/	N	٢٩٠(D/)	D(٨١)
/A	Y			/	Y		

/AB	N	٢٨٥(/AB)	/A(٢٨١)	/A	Y		
B	Y			/AB	Y		
B/	N	٢٨٦(B/)	B(٧٩)	/ABE	N	٢٩١(/ABE)	/AB(٢٨٥)
/	Y			E	Y		
/A	Y			EOF			E(٨٢)
/AB	Y						

Table ٢: Trace the Operations of Secure LZW Encoding Algorithm

٧- Secure LZW Decoding Algorithm

It shouldn't be a big surprise that secure LZW data is decoded better than how it is encoded. The dictionary is initialized and confused in the same encoding manner so that it contains an entry for each byte plus nulls entry. Only the owner of correct key can confuse the LZW initial dictionary in proper way. New code words are read from the input stream, one at a time and string encoded by the new code is output. Based on the above previous discussion, decoding input consists of the following steps:

Algorithm ٤: Secure LZW Decoding Ideas

INPUT: Symbols one by one from compressed file (which consists of pointers to the dictionary).

OUTPUT: Uncompressed symbols.

Step ١: Initialize dictionary to contain one entry for each byte.

Step ٢: The encryption key is used to initialize a cryptographically secure pseudo random bit and number generator.

- The first five bits represent the number of added nulls
- The Li column value in CSPRBG represents the location where the nulls will insert in the LZW initial dictionary.

Step ٣: Read the first code word from the input stream and write out the byte it encodes.

Step ٤: Read the next code word from the input stream.

Step ٥: If the code word is an EOF exit.

Step ٦: Write out the string encoded by the code word.

Step ٧: Concatenate the first character in the new codeword to the string produced by the previous codeword and add the resulting string to the dictionary.

Step ٨: Go to step ٤.

٨- Decoding Explanation with Example

The algorithm above is just like the compression algorithm. It generates the same cryptographically secure pseudorandom bit, this is done only when having the encryption key. Use the output of CSPRBG to confuse the initial dictionary. Add a new string to the string table each time it reads in a new code. It needs in addition to that to translate each incoming code into a string and send it to the output.

Table (٣) illustrates the output of the algorithm given and the input created by the compression earlier in the section. The output string is identical to the input string from the compression algorithm. Note that the first ٢٥٦ codes plus specific number of nulls are already defined to translate single character strings, just like the compression code.

Table ٣: Trace the Operations of Secure LZW Decoding Algorithm

Input Codes: / A B C ٢٨١ B ٢٨٥ ٢٨٦ ٢٨٢ D ٢٨٥ E							
Character Input	**In Dic?**	**New Entry**	**Code Output**	**Character Input**	**In Dic?**	**New Entry**	**Code Output**
/	Y			/AB	Y		
/A	N	٢٨١(/A)	/(٥٩)	/ABB/	N	٢٨٧(/ABB/)	/AB(٢٨٥)
A	Y			B/AB	N	٢٨٨(B/AB)	B(٧٩)
AB	N	٢٨٢(AB)	A(٧٨)	/AB	Y		
B	Y			/ABD	N	٢٨٩(/ABD)	/AB(٢٨٥)
BC	N	٢٨٣(BC)	B(٧٩)	D	Y		
C	Y			D/AB	N	٢٩٠(D/)	D(٨١)
C/A	N	٢٨٤(C/)	C (٨٠)	/AB	Y		
/A	Y			/ABE	N	٢٩١(/ABE)	/AB(٢٨٥)
/A B	N	٢٨٥(/AB)	/A (٢٨١)	E	Y		
B	Y			EOF			E(٨٢)
B/AB	N	٢٨٦(B/)	B(٧٩)				

Output string is [/ABC/AB/ABB/ABD/ABE] and this is equal to encoding input string. It is possible that it is in the mind of the reader which leads to an important question. What are the results that will be obtained by unauthorized persons in the case of trying to decompress the data compressed by secure LZW coding?

This question will be answered in practice through the obtained results in table (٤), which explains what the results gets by the people who are not authorized in the case of them trying to spy on the data compressed by the secure LZW coding in previous example.

Table ٤: Trace of Decoding Operations from Unauthorized View of Point

Input Codes: ٥٩ ٧٨ ٧٩ ٨٠ ٢٨١ ٧٩ ٢٨٥ ٢٨٦ ٢٨٢ ٨١ ٢٨٥ ٨٢			
Character Input	**In Dic?**	**New Entry**	**Code Output**
;	Y		
;N	N	٢٥٦(;N)	; (٥٩)
N	Y		
NO	N	٢٥٧(NO)	N(٧٨)
O	Y		
OP	N	٢٥٨(OP)	O (٧٩)
P	Y		
P ٢٨١	STOP		

The unauthorized persons will stop and cannot proceed because of lack of knowledge the value of (٢٨١) and all the unauthorized person got was [; N O]. This shows the strength of security enjoyed by the proposed method against those trying to get the data compressed by the secure LZW coding.

٩- Evaluating Compression Performance

The Secure Lempel Ziv and Welch methods maintain the effectiveness of their respective antecedents. The dictionary built in the secure LZW method is almost identical to the dictionary built in the original compression algorithm. These imply that the efficacy of the compressions is not compromised by the proposed method.

١٠- Security Strength

The proposed method introduces more proper security strength than substitution or transposition ciphers. Without the identical crypto-compression key, the decompression process will be disabling and the cryptanalysis can’t even be completed.

A brute-force attack involves trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained. On average, half of all possible keys must be tried to achieve the success. To prove cryptanalysis difficult for the proposed approach, the sample of text is taken and it is encrypted in different encryption algorithms such as DES (Data Encryption Standard), triple DES, AES (Advanced Encryption Standard) and the proposed approach SLZW (Secure Lempel Ziv and Welch).

Table (٥) illustrates how much time is involved for various key spaces. Results are shown for four binary key sizes. The ٥٦-bit key size is used with the DES algorithm, and the ١٢٨-bit key size is used for triple DES. The minimum key size specified for AES is ١٦٨ bits. The ٢٥٦-bit key size is used with the proposed SLZW .For each key size, the results are shown assuming that it takes ١ ms to perform a single decryption, which is a reasonable order of magnitude for today's machines. With the use of massively parallel organizations of microprocessors, it may be possible to achieve processing rates many orders of magnitude greater.

Table ٥: Average Time Required for Exhaustive Key

Encryption Algorithm	**Key size (bits)**	**Number of alternative keys**		**Time required at ١ decryption/□s**	
DES	٥٦	$٢^{٥٦}$	= ٧,٢ x $١٠^{١٦}$	$٢^{٥٥}$ ms	= ١١٤٢ years
Triple DES	١٢٨	$٢^{١٢٨}$	= ٣,٤ x $١٠^{٣٨}$	$٢^{١٢٧}$ ms	= ٥,٤ x $١٠^{٢٤}$ years
AES	١٦٨	$٢^{١٦٨}$	= ٣,٧ x $١٠^{٥٠}$	$٢^{١٦٧}$□□ms	= ٥,٩ x $١٠^{٣٦}$ years
SLZW	٢٥٦	$٢^{٢٥٦}$	= ١,١ x $١٠^{٧٧}$	$٢^{٢٥٥}$ ms	= ١,٨ x $١٠^{٦٣}$ years

١١- Conclusions

The complete system design has been implemented and tested, the results are found as they were expected. The results proved the correctness of system design and its reasonable considerations and choices. This work arrives at the following conclusions:

١. The main theme behind the design of this system is implemented to get the best security performance and best-case compression ratio via the abilities of strongest for lossless compression and cryptography principles.
٢. The common encryption methods generally manipulate an entire data set, and most encryption algorithms tend to make the transfer of information more costly in terms of time and sometimes bandwidth. Thus, users pay a price for security proportional to their desired level of security. The proposed method of simultaneous encryption and compression may serve to remedy the security and speed issues that currently concern the multimedia world.
٣. In particular, the secure LZW method is based on the cryptographically secure pseudorandom bit generation function, which gives a level of security that cannot be ignored because they generate numbers very difficult to predict them.
٤. The compression and encryption processes are complicated processes. One of the advantages of the proposed method is that the processes are done transparently, their details are buried, and the user should not be aware that these processes are taking place, further than the requirement to enter his password.
٥. Experimental results added levels of security over the existing compression method (LZW).

٦. all test results are found as close as they were expected. The results proved the correctness of system design and its reasonable considerations and choices.

REFERENCES

[١] Milidiu, R.L., Mello, C.G, Fernandes J.R. Adding security to compressed information retrieval systems, SPIRE - String Processing and Information Retrieval, Chile, ٢٠٠١.

[٢] Bruce Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", ٢nd Edition, John Wiley & Sons, Inc., ١٩٩٦.

[٣] William Stallings, "Cryptography and Network Security: Principles and Practices", International ٤th Edition, Prentice Hall, November ١٦, ٢٠٠٥.

[٤] Peter Wayner, "Data Compression for Real Programmers", Morgan Kaufman, Inc., ١٩٩٩.

[٥] Ida Mengyi Pu, "Fundamental Data Compression", Elsevier Inc, ٢٠٠٦.

المستخلص

الهدف الرئيسي لهذا البحث هو تحقيق السرية وسلامة البيانات المرسلة وتقليص التكرار بتمثيل البيانات في اثناء نقلها عبر شبكات الحاسوب العامة. وذلك من خلال تطوير طريقة تجمع بين الضغط والتشفير في المجموعة نفسها من البيانات, وتنفيذ العمليتين في وقت واحد بدلا من تنفيذها منفردتين. وهذا يتحقق عن طريق تضمين التشفير في خوارزميات الضغط.

نفذ هذا النظام بواسطة لغة البرمجة سي شارب (Visual C#.NET) .