A Novel chaos-based Joint Compression and Encryption scheme using Normalized Conditional Bi-gram Probability

Yu-Chen Lin\textsuperscript{1,*}, Ja-Ling Wu\textsuperscript{1}

\textsuperscript{1} Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.

Received xx mm 201x; received in revised form xx mm 201x; accepted xx mm 201x

Abstract

A new approach to enhance the compression performance and execution time of chaos-based joint compression and encryption schemes is proposed. Instead of finding a new method to update the Look-up Table (LUT) in each iteration or using a different method to represent the ciphertext, we use multiple LUTs based on the conditional bi-gram probabilities of two consecutive source symbols occurring in the input. This high order probability model can represent the characteristics of whole input file more accurately. As a result, it will further decrease the search time as well as the length of the resultant ciphertext. Simulation results also verify that the proposed approach does lead to better compression performance and execution efficiency.

Keywords: Chaos, Encryption, Compression, Chaotic map, Cryptography, Simultaneous encryption and compression.

1. Introduction

In recent decades, networking as well as bandwidth allocation technologies are improving and the sharing of digital data through the network becomes much easier. This fact stimulates the demand for various multimedia applications and services. Multimedia files are usually large and should better be compressed before transmission. On the other hand, most users share their files using the open network that is susceptible to eavesdropping. The security and the efficiency requirements of information communication lead to a lot of research works focused on data encryption and compression. To advance the performance and the flexibility of multimedia applications, it is valuable to joint encryption and compression into a single process. In general, there are two different approaches in this special area. One is to embed key-controlled confusion and diffusion processes into source-coding schemes, whereas another is to incorporate compression into cryptographic algorithms. The first direction has attracted more attention since the entropy coder can be easily modified to become a cipher without using any complicated technique. At the encoder side, a private key is used to control the statistical model for encoding. To decode the encoded sequence correctly, the same secret key is needed to synchronize the decoding statistical model with the encoding one.

In Huffman coding, the Multiple-Huffman-table based approach [1] simultaneously performs encryption and compression based on a key-controlled swapping of the right and the left branches of a Huffman tree. Similarly, some cryptography features such as randomized coding [2] and the key-based interval splitting [3] have been considered in the area of arithmetic coding. Unfortunately, both Multiple-Huffman-table based approaches and randomized and interval-splitting based arithmetic coding schemes are vulnerable to some classic attacks, such as known-plaintext attack and chosen-plaintext attack [4, 5-10]. Secure Arithmetic Coding (SAC) [17] is a security enhancement version of key-based interval splitting, which includes a series of permutations at the input and output of the arithmetic code’s encoder. Nevertheless, its security was also cryptanalyzed in [8-10].

Owing to the security problems mentioned above, there is an increasing trend of designing joint compression and encryption schemes based on chaotic maps [11-17]. Most of the chaotic-based joint compression and encryption schemes and recently developed related works [18-21] were focused on bettering the system security [4-11, 19-21]. The compression capability of these algorithms is close to that of traditional entropy coding, but due to the requirement of extra high-precision processes, there is still a room for improving the corresponding execution speed [16, 18].

Under nearly the same security level, in this work, we try to develop novel chaotic-based joint compression and encryption schemes, in which both the compression performance and execution speed can be enhanced. In other words, the attempts done in [12, 14, 15] are the most closely related works to ours and will be explained briefly in the following section. Clearly, in practical applications, Compression Gain, Security Level and Computational Efficiency of the algorithms should be balanced. Therefore, a new chaotic-based joint compression and encryption scheme is developed in this work to achieve this challenging but demanding goal.

This paper is organized as follows. In Section 2, some related chaotic-based joint compression and encryption schemes are reviewed and analyzed briefly, first. In Section 3, the newly proposed scheme is addressed in detail. Performance and security analyses of the proposed approach are presented in Section 4. Finally, Section 5 concludes this write-up.

2. Related Work

2.1 Look-Up-Table and Chaotic-Map based Approach

An efficient chaos-based cipher searches the plaintext symbol in a codeword-based look-up table (LUT) using a key-dependent chaotic trajectory and treats the required number of iterations of the chaotic map as the ciphertext. Although the timing performance of the realization is good, it still suffers from the common issue of ciphertext expansion:
the length of the ciphertext is usually about 1.5 to 2 times that of the plaintext. In order to incorporate certain compression capability, [12] attempted to adaptively construct the LUT according to the probabilities of the plaintext symbols. With the aid of such a symbol-probability dependent codeword table construction, the resultant ciphertext length (obtained in [12]) is shorter than that of the plaintext, but the compressed result still has a distance from that of the source entropy. This is because the trajectory of a chaotic map frequently lands on the partitions corresponding to irrelevant source symbols, which leads to the fact that the number of iterations is larger than necessary.

For the ease of further discussion, the method of integrating compression with a chaos-based cryptosystem, presented in [12], is briefly in a bit more detail first. This scheme can be considered as a hybrid cipher and there are two operational modes in it. Source symbols with higher probability of occurrence are encrypted by searching through a dynamic LUT, and this kind of operation is said working in the search mode. Entropy coding is then applied to compress the outputs of this mode. Then, both the entropy-coded codewords and other less probable symbols are masked by a pseudorandom bitstream, and this is named as working in the mask mode. Since the mask mode does not contribute any reduction in plaintext length, the compression capability of this hybrid cipher is mainly provided by the search mode. That is, the more the symbols are encrypted in the search mode, the higher the compression ratio the hybrid cipher can achieve.

In search mode, the to be encrypted plaintext symbol is searched in an LUT on the basis of a pseudorandom sequence generated by iterating a chaotic map from the key-dependent parameters and the given initial condition. In [12], the logistic map is chosen as the underlying chaotic map and it can be represented mathematically as

\[ x_{n+1} = bx_n(1 - x_n) \] (1)

where \( x_n \) is the real number output between 0 and 1 at the discrete time index \( n \). The real valued parameter \( b \) should be a number between 3.6 and 4 for producing chaotic trajectory. The LUT is constructed by linking the partitioned phase space of the chaotic map and the corresponding map of plaintext symbols. More partitions are allocated to more probable symbols so that they will have a higher chance to be visited by the chaotic-based searching trajectory. The number of iterations of the chaotic map is defined to be the length of the searching trajectory, which is then taken as the corresponding ciphertext of the visited plaintext symbol.

The encryption process can be understood as follows. Encrypting the target plaintext symbol using the LUT is equivalent to randomly getting a symbol, from the set of symbols, until the target symbol is drawn. The time of drawing is defined to the length of the searching trajectory, i.e., the ciphertext.

Suppose that there are four source symbols, i.e., \( W, X, Y, \) and \( Z \), with probabilities of occurrences of \( 1/2, 1/6, 1/6, \) and \( 1/6 \), respectively. Hence, as shown in Fig. 1, half of the partitions are mapped to symbol \( W \). Symbols \( X, Y \) and \( Z \) each associates with \( 1/6 \) of the total number of partitions. In sampling with replacement scheme [12], the procedure of encrypting symbol \( W \) corresponds to find its geometric distribution in probability. For example, if the first success in getting \( W \) is achieved at the \( t \)-th draw after \( t-1 \) failures. The total number of draws, \( t \), is the ciphertext for symbol \( W \). The probability of drawing \( W \) from the table with replacement can be formulated as in (2), where \( (1 - P(W))^{t-1} \) is the probability of failing to obtain \( W \) in the first \( t-1 \) times. The cumulative probability of the first \( t \) draws, denoted as \( CP_t(W) \), is given by (3). These expressions indicate that \( P_t(W) \) is close to zero and \( CP_t(W) \) approaches 1 only if \( t \) tends to infinity. In other words, the ciphertext of \( W \) could be any value from one to infinity in theory, and therefore, it might occupy a lot of bits in practice and leads to the above-mentioned ciphertext expansion problem.

\[ P_t(W) = (1 - P(W))^{t-1} P(W) \] (2)

\[ CP_t(W) = P(W) + (1 - P(W))P(W) + \ldots + (1 - P(W))^{t-1} P(W) \]
\[ = 1 - (1 - P(W))^t \] (3)

2.2 Dynamic Updating Look-Up-Table based Approach

To improve the compression performance, in [14], a dynamically updated LUT is used to do encryption in the searching process. That is, if the plaintext symbol being encrypted is found, the number of iterations of the chaotic map is considered as the corresponding ciphertext; otherwise, the LUT will be updated. If the partition just landed maps to a non-target symbol, all the partitions associated with that symbol will be re-assigned to a new symbol. Considering about the compression performance and the simplicity of encryption process, those partitions are re-assigned to the not-yet-visited symbol with the highest probability. Ultimately, since the target symbol will be mapped to a larger phase space of the LUT, it will be reached with a higher chance. The required numbers of iterations for encryption, as pre-described, is reduced. The ciphertext is shortened and a better compression performance is accomplished. A brief example of the LUT updating is given in Fig. 2.
Fig. 2. The scheme proposed in [14]. Since the first iteration output $x_1$ (or symbol W) is not the being encoded symbol Y, the partitions corresponding to symbol W (in the first column) will be re-assigned to symbol X (the not-yet-visited symbol with the highest probability). And the second iteration output $x_2$ (or symbol X) is not the being encoded symbol Y as well. The partitions corresponding to symbol X (in the second column) will be re-assigned to symbol Y (the not-yet-visited symbol with the highest probability), and so on. Three iterations are needed to visit the target symbol Y, and therefore, the ciphertext of Y is three. When the current plaintext symbol has been encrypted, the lookup table is initialized again according to the symbols’ probabilities of occurrence, as shown in the 4-th column.

2.3 Number of Distinct Plaintext Symbol based Approach

In [15], instead of treating the number of iterations of the chaotic map as the ciphertext, the number of distinct symbols visited is used. In this scheme, as shown in Fig. 3, the range of ciphertext is limited to the number of different symbols of the plaintext.

Fig. 3 The scheme proposed in [15]. Four iterations are needed to visit the target symbol Y; however, the symbol X has been met two times. That is, there are three distinct visited symbols, and therefore, the ciphertext of Y is three.

3. Proposed Method

As shown in section 2, the performances (including both compression ratio and execution speed) of chaotic-map based joint compression and encryption schemes have been improved by various methods. In this work, a higher-order source probability model, i.e., the bi-gram based conditional probability instead of the singleton based marginal probability, is utilized to further enhance the above-mentioned performances.

3.1 Source Symbol Models of the Proposed Scheme

Instead of finding a new method to update the LUT in each iteration or using a different method to represent the ciphertext, we use multiple LUTs based on the conditional bi-gram probability of two consecutive source symbols occurring in the input to build our scheme. Mathematically, we can prove that the normalized conditional bi-gram probability (NCBP) is always larger than or equal to the marginal bi-gram probability (MBP) and, with proper assumptions, we claim that if we use the number of distinct symbols visited as ciphertext and construct the LUT on the basis of NCBP model, a shorter ciphertext than that of [12], [14] or [15] could be expected. To the best of our knowledge, this work is the first one, in the literature, using higher-order statistics to increase the compression ratio of a joint compression and encryption scheme.

Let us denote the NCBP of two consecutive symbols start with $S_i, S_j$, as $P(S_{i} | S_{j})$, where $S_i, i \in \{1, 2, 3 \ldots N\}$, stands for the original data symbol and $x \in \{S_1, S_2, \ldots, S_N\}$, which appears in the parsed bi-gram set of the input. Clearly, N denotes the number of different plaintext symbols.

Take the input sequence “ABACCABAACAB?” as an example, the parsed bi-grams are AB, BA, AC, CC, CA, AB, BA, AA, AC, CA, AB, B? in succession, where “?” is a specific symbol used to denote End of File. Since N=3, in this example, A, B, and C are the three candidates for $S_i$ and the whole bi-gram set can be partitioned into the following three bi-gram subsets started with different leading symbols, that is

Bi-gram subset A = {AB, AC, AB, AA, AC, AB} 
Bi-gram subset B = {BA, BA, B?} 
Bi-gram subset C = {CC, CA, CA}.

Notice that “B?” indicates there is no successor to B; this bi-gram is not included in the table creation process.

By the definitions of bi-gram and conditional probability, we have

$$P(S_{i} | S_{j}) = \frac{P(S_{i} \cap S_{j})}{P(S_{j})} = \frac{P(S_{i} | S_{j})}{P(S_{j})}$$

In (4), the division by $P(S_{i})$ will normalize $P(S_{i} | S_{j})$ into the unit interval $[0, 1]$; moreover, $P(S_{i}) < 1$ proves that $P(S_{i} | S_{j}) > 1$ or equal to $P(S_{i})$. Let’s go back to the previous example, clearly, we have $P(A) = 1/2$, and $P(B) = P(C) = 1/4$. From the parsed bi-gram set, it is easy to verify that the corresponding MBPs are P(AA) = 1/12, P(AB) = 1/4, and P(AC) = 1/6. Since $P(A) = 1/2$, according to (3.1), the corresponding NCBPs become P(AA|A) = 1/6, P(AB|A) = 1/2, and P(AC|A) = 1/3, respectively. Notice that the above NCBPs conditional on the leading symbol A can also be derived from the Bi-gram subset associated with A. Similar relationships hold for NCBPs conditioned on symbols B and C, also.

Main Claim: The relative entropy $D(p||q_1)$ is larger than $D(p||q_2)$, where $p$, $q_1$ and $q_2$ indicate the probability density functions of the target source, approaches provided in [12, 14, 15] and our scheme, respectively.

From previous definition of $P(S_{i} | S_{j})$ and basic properties of probability we have

$$\sum P(S_{i} | S_{j}) = P(x)$$

(5)

By (4) and (5), the difference between P(B) and P(AB|A) can be understood as follows. Instead of taking all possible bi-grams into account, in computing P(AB|A) all bi-gram pairs
lead by symbols B and C, i.e. the bi-gram subset B and bi-gram subset C, are removed from the universal set. For instance, in the above example input ("ABACCABAACAB?"), the symbol B always comes after A directly. Therefore, rather than considering P(B) like [12, 14, 15], we directly pondering P(AB|A) (=P(B)/P(A) in this case) instead, which will be larger than P(B). It means NCBP more closely approaching to the true source probability than that of 1-gram probability. Therefore, according to the definition of relative entropy, we obtain our claim that the relative entropy \( D(p||q) \) would be smaller than \( D(p||q_1) \). This counting sample removal depicts the physical meaning of NCBP and also explains why the proposed NCBP will model the source input better than its 1-gram counter-part. Moreover, with the proposed NCBP model, the range (or size) of the ciphertext will be limited by the number of different symbols in the LUT associated with each \( P(S_i|X S_i) \). For instance, as shown in Fig. 5, is only one plaintext symbol A in LUT B, which means the corresponding ciphertext range equals to one. Clearly, the number of symbols in NCBP based LUTs will always be less than or equal to that of the conventional 1-gram LUT; therefore, in the pre-described chaos-based joint compression and encryption schemes, better compression performance than that of [12, 14, 15] could be expected if NCBP-model is adopted.

3.2 The Proposed Joint Compression and Encryption Scheme

As illustrated in Fig. 4, in the proposed scheme, we first generate multiple LUTs (one for each plaintext symbol) based on the plaintext data, where the pre-described NCBP model is involved. Then, a similar method to [15] is applied to do chaotic-based encryption. Finally, the modified Huffman encoding presented in [17] is used to compress the encrypted data. The details of each step will be addressed in the following sub-sections, respectively.

![Fig. 4 The block diagram of the proposed chaotic-map based joint compression and encryption scheme.](image)

a) Table Creation Step

In LUT creation, N+1 tables will be generated. One of them is the same as that of [12], [14] or [15], while the others are generated according to the proposed NCBP model, given in (2). To easily implement the encryption and decryption processes, we order the N Bi-gram subsets (or equivalently the N newly generated LUTs) according to the alphanumerical order of plaintext symbols. An example of table creation, for the example given in sub-chapter 3.1, is illustrated in Fig. 5.

In order to use a higher-order statistical model to enhance system performance, the proposed approach belongs to two-pass joint compression and encryption schemes. That is, a parsing process must be executed in advance, to obtain the bi-grams of an input. Notice that the cost for doing bi-gram parsing is nearly the same as that for 1-gram parsing, used in all pre-mentioned conventional approaches.

![Table Creation Step](image)

Convention

Fig. 5 The four LUTs, generated based on the proposed NCBP model, for the plaintext input sequence ABACCABAACAB?, where "?" is a special symbol used to indicate the end of the to be compressed sequence.

First, we generate a table that is the same with those generated in [12], [14] or [15], as shown in the first column of Fig 5 (we called it the convention-LUT for short). Second, the parsed bi-grams, started with symbol A, are AB, AC, AB, AA, AC, and AB in succession. Since N=3, the LUT for symbol A (as shown in the second column of Fig. 5), are divided into 3 partitions represented by 3 different colors. In which, symbol A (i.e., bi-gram AA) holds 1/6 range of the table, symbol B (i.e., bi-gram AB) occupies 1/2 range of the table, and symbol C (i.e., bi-gram AC) holds 1/3 range of the table. Third, the LUTs for symbols B and C can, then, be generated in the same way (as shown in the third and the forth columns of Fig. 5).

As shown in Fig. 5 and also explained in the previous section, for example, in compressing the bi-gram BA, the required number of chaotic map iterations (or the number of distinct symbols visited) will be largely reduced if LUT-B is used instead of the convention-LUT. This implies better compression performance can be expected.

![Fig. 6 The example of a 2-map table splitting. The original table A can be further split into two sub-tables to reduce the length of ciphertext and time spend for searching.](image)
their probability values of occurrence and distributed into \( P \) sub-tables in turn. That is, the first sub-table will contain \( \frac{L}{P} \) symbols with the highest probabilities, and the second sub-table will contain the \((\frac{L}{P}+1)\)-th, \((\frac{L}{P}+2)\)-th, ... , \((2\frac{L}{P})\)-th probable symbols, and so on. This kind of table-splitting is called P-map approach in [12, 14, 15].

After splitting into \( P \) sub-tables, we can search the target symbol only in the sub-table which it belongs to, rather than search in the original full LUT of plaintext symbols. Because we order the symbols by their probabilities, the longest ciphertext (i.e. the maximal number of the different symbols in the searching trajectory) of plaintext symbols with the highest probability will all be restricted less than or equal to \( \frac{L}{P} \). Due to the above reasons, ideally we can enhance the execution speed by \( P \) times, and in addition, the compression performance can also be improved at the same time. Fig. 6 illustrates an example of 2-map table-splitting, in which the four symbols in the original LUT A are re-arranged into two sub-tables, according to their probabilities (or frequencies) of occurrence.

Following this procedure, symbols with higher probabilities of occurrence will be mapped to more partitions in the same sub-table or be located to sub-tables in front than those with lower ones. With such an arrangement, a better compression ratio and faster execution speed can be obtained, even in sequential realization. Of course, this table splitting technique is adopted for speeding up our scheme.

b) Encryption Step

After accomplishing the table creation step, we start the process of encryption. Similar to the scheme described in [15], we sequentially encrypt each plaintext symbol of the input sequence by searching in the corresponding LUT, built in the previous step, using a secret chaotic trajectory. If the symbol to be encrypted is found, the number of different symbols the trajectory has met in the chaotic mapping is considered as its ciphertext.

Notice that in encrypting the first symbol of the input, the 1-gram probability based LUT, i.e., the convention-LUT, is used. After encrypted the first input element, the tables generated based on the above-mentioned NCBP model, are exploited to do subsequent encryption. As analyzed in sub-section 3.1, symbols appeared in the NCBP-based LUTs will mostly have higher probability to be met than those in 1-gram-based one. This fact embraces the superiority of our approach to related works [12, 14, 15], in compression ratio.

As pre-described, “Mask Mode” has been proposed in [12, 14, 15] to speed up the whole processing. In mask mode, only popular symbols, the symbols with probability of occurrence higher than a given value, are put into the LUT. That is, the above-mentioned LUT searching approach will only be applied to encrypt popular symbols. More specifically, the chaotic map continues to iterate until its trajectory falling in a partition that is mapped to that popular symbol. In addition, one can set a threshold \( T \), which indicates the maximum number of allowable chaotic map iteration (or the largest size of permissible ciphertext). If the symbol is not popular or its ciphertext is longer than \( T \), a special symbol is used to reflect that the being encrypted symbol should be treated with mask mode and the corresponding original plaintext symbol is linked directly to the special symbol, as the resultant ciphertext.

After the whole plaintext sequence has been encrypted, a modified Huffman tree [17] is built for compressing the obtained ciphertext.

c) Huffman Coding Step

A Huffman dictionary consisting of codewords, \( h_0, h_1, ... , h_M \), is constructed according to the probability distributions of distinct ciphertext symbols \( s_i \in S \). In which, \( h_0 \) denotes the Huffman code of the above-mentioned special symbol, \( h_i \) for \( i > 1 \) stand for the Huffman codes of the popular symbols, \( M \) denotes the number of popular symbols (clearly, \( M < N \)), and \( S \) is the set of ciphertexts obtained in the previous step. Assuming that the length of Huffman codes varies from \( l_{min} \) to \( l_{max} \), where \( l_{min} \) and \( l_{max} \) denote the shortest and the longest existing code-lengths, respectively. In [17], different disjoint sets are formed, from the set \( S \), as per different length of Huffman codes such that

\[
D_1 = \{ x \mid x \in S, \text{ code} - \text{length}(x) = l_{min} \} \\
D_2 = \{ x \mid x \in S, \text{ code} - \text{length}(x) = l_{min} + 1 \} \\
\vdots \\
D_{l_{max}-l_{min}+1} = \{ x \mid x \in S, \text{ code} - \text{length}(x) = l_{max} \} \\
S = D_1 \cup D_2 \cup ... \cup D_{l_{max}-l_{min}+1}
\]

Logistic map is then iterated to generate a pseudorandom sequence of the size of a disjoint set \( D_i \), where \( i \) varies from 1 to \( l_{max} - l_{min} + 1 \). The symbols within the formed disjoint subsets are then scrambled using the generated pseudorandom sequences. Different initial conditions of the map have been assumed for different subsets. This yields a different mapping function between symbols and the codewords for each subset.

After the Huffman tree has been built, the ciphertexts and the special symbol are replaced by the corresponding variable-length Huffman codes, as shown in Fig 7.

![Huffman Coding Diagram](Image)

Fig. 7 The Huffman encoding step, where \( h_i \) (\( i > 0 \)), \( S \) and \( P \) represent the variable-length Huffman codes of the popular symbols, the ciphertext symbols and the non-popular plaintext symbols, respectively. The specific Huffman code \( h_0 \) is used to indicate that the plaintexts directly concatenated to it, such as \( P_{129} \) and \( P_{235} \) are encrypted by mask mode.

d) Decryption Step

The secret key, in Fig. 4, consists of the parameters and the initial value of the chaotic map. It must be secretly transmitted to the receiver. In addition, the information about plaintext's probability and NCBP must be available to the receiver for reconstructing the corresponding LUTs. They must be secretly
delivered to the receiver or be attached to the secret data. The
decryption process is similar to the encryption one. The
decoder generates the ciphertext based on the modified
Huffman table and the secret key of each disjoint set. Then,
decoder regenerates the chaotic trajectory with the secret key
and then retrieves the plaintext symbol from the LUTs. If the
number of different symbols has met is smaller than the
ciphertext, the chaotic search trajectory has not landed on the
target symbol yet. With the correct key, the original plaintext
sequence can be perfectly reconstructed.

3.3 Remarks

In our approach, the first plaintext symbol is encrypted
based on the traditional 1-gram probability model, and the
just-before-being-encrypted symbol is used to determine which
table should be used. As pre-described, with the aid of
NCBP modeling, better compression performance can be
expected; however, the overheads for generating the extra
Tables 1 and 2, respectively.

4.4 Analyses and Experimental Results

4.1 Analysis of Overheads

We first analyze the overheads of extra LUTs. Because, in
general, all the plaintext symbols need to be stored, we will
have N extra tables. The overheads for storing extra LUTs,
against [12], [14] or [15], should be proportional to N*N,
where N is the number of different plaintext symbols. Of
course, we can use Huffman codes to further reduce the sizes
of required LUTs. According to our experiments on
benchmarking data set (will be detailed later), the overhead per
each plaintext symbol for storing extra LUTs is 7.063 bits in
average. In other words, as compared with related works, our
overhead for storing extra LUTs is around (N*N+7.063) bits,
in average.

Second, let’s consider the sizes of the required Huffman
tables build on the basis of the above LUTs. Assuming that
there are K different numbers of values will be generated
according to bi-gram NCBP of the plaintext data, for perfect
decoding, we allocate a fixed K_bit to represent K. Suppose,
for each one of the K possible symbols, we use S_bit bits to
represent the symbol itself and L_bit bits to indicate its
codeword length. Then, the total size with respect to Huffman
tables will be

\[ K_{\text{bit}} + K \times (S_{\text{bit}} + L_{\text{bit}} + H L_{\text{bit}}) \text{ bits} \]  

(7)

where H L_{\text{bit}} is the average length of all Huffman codes. In
our implementation, integer-format is used to represent each
symbol in the extra tables. In order to facilitate the
implementation, we split each integer into four bytes. After that,
we encode the bytes by Modified Huffman Coding. Since the
different number of a byte is 256, the K in our setting will be
256. Moreover, in our approach, the size of ciphertext equals to
that of the plaintext, so we need not to record the ciphertext
symbol. Therefore, in our approach, the overheads for storing
Huffman tables of the whole file become

\[ N \times (L_{\text{bit}} + H L_{\text{bit}}) \text{ bits} \]  

(8)

where N is the number of plaintext (or ciphertext) symbols.

4.2 Experimental Results

The logistic map is chosen as the underlying chaotic map,
with the same settings given in [14] and [15]. The plaintext
symbols are read in bytes. Two simulation configurations are
test. The first one is that only the most probable 16-plaintext
symbols of the whole file are selected and each of them is
mapped to each one's LUT. In the second case, the most
probable 128-plaintext bi-gram symbols (according to the
magnitudes of their NCBPs) are selected and the
subsequent 256 LUTs are divided and distributed to 16
sub-tables (i.e. the 16-map approach, as described in
sub-section 3.2-1), eight symbols for each sub-table. That is,
we will have 16*N+1 LUTs, in the second case. Clearly, the
16-map approach is adopted mainly for reducing the table's
searching time.

To benchmark the capability of compression, the standard
testing files taken from Silesia Corpus [22] are adopted. They
contain files of sizes between 6 MB and 51 MB. The 12 files
include two medical images, the SAO star catalog, and a few
executable files.

Performance comparisons of both 1-map and 16-map
approaches are shown in Table 1 and Table 2, respectively.
The files in the tables are sorted by their file sizes. The
compression ratio is calculated by

\[ R = \frac{\text{Ciphertext Length}}{\text{Plaintext Length}} \times 100\% \]  

(9)

All the compression ratios reported in Table 1 and Table 2
are less than 100%, which means all files can be compressed
by all compared methods. Notably, nearly all the compression
ratios of our approach are the best among the compared works.

4.3 Encryption and Decryption Speed

For providing fair speed comparison, besides our approach
we also implemented related state-of-the-art methods
presented in [14] and [15] using C language and run on
standard MacBook with retina-2015, which has 15 CPU and
8G RAM.

To show the improvement our approach achieved, only the
time spend in the chaotic based encryption is considered. From
Tables 1 and 2, 1-map approach is faster than 16-map in most
cases. This is because there are more symbols to be encrypted
by search mode in the latter case. Again, nearly all the
execution speeds of our approach are the best among the
compared works.

Table 1 Performance comparisons, in terms of
compression ratio and execution time, of the proposed
approach and related works. In which, 1-map (the single
whole LUT) approach is adopted.

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Proposed-1 map</th>
<th>[14]-1 map</th>
<th>[15]-1 map</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>Proposed-1 map</td>
<td>[14]-1 map</td>
<td>[15]-1 map</td>
</tr>
<tr>
<td>xml</td>
<td>74.88%</td>
<td>0.581987</td>
<td>82.90%</td>
</tr>
<tr>
<td>office</td>
<td>84.83%</td>
<td>0.511534</td>
<td>89.86%</td>
</tr>
<tr>
<td>reymont</td>
<td>59.26%</td>
<td>1.042513</td>
<td>73.00%</td>
</tr>
<tr>
<td>sao</td>
<td>94.89%</td>
<td>0.359626</td>
<td>98.91%</td>
</tr>
<tr>
<td>x-ray</td>
<td>84.00%</td>
<td>1.190557</td>
<td>84.36%</td>
</tr>
<tr>
<td>mr</td>
<td>52.55%</td>
<td>1.065045</td>
<td>53.30%</td>
</tr>
<tr>
<td>osdb</td>
<td>90.23%</td>
<td>1.105094</td>
<td>92.71%</td>
</tr>
<tr>
<td>dickens</td>
<td>61.84%</td>
<td>2.837324</td>
<td>67.57%</td>
</tr>
<tr>
<td>samba</td>
<td>80.75%</td>
<td>3.614417</td>
<td>87.03%</td>
</tr>
<tr>
<td>mci</td>
<td>35.09%</td>
<td>15.00370</td>
<td>57.82%</td>
</tr>
<tr>
<td>webster</td>
<td>67.22%</td>
<td>15.16381</td>
<td>74.76%</td>
</tr>
<tr>
<td>nectar</td>
<td>78.18%</td>
<td>8.769112</td>
<td>85.20%</td>
</tr>
</tbody>
</table>
Table 2 Performance comparisons, in terms of encryption ratio and execution time, of the proposed approach and related works. In which, 16-map approach is adopted.

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Proposed-16 map</th>
<th>[14]-16 map</th>
<th>[15]-16 map</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>Ratio</td>
<td>Time (Sec)</td>
<td>Ratio</td>
</tr>
<tr>
<td>xml</td>
<td>49.73%</td>
<td>0.847767</td>
<td>69.74%</td>
</tr>
<tr>
<td>office</td>
<td>71.85%</td>
<td>1.032073</td>
<td>83.80%</td>
</tr>
<tr>
<td>reynette</td>
<td>43.29%</td>
<td>1.357393</td>
<td>61.91%</td>
</tr>
<tr>
<td>sao</td>
<td>83.45%</td>
<td>1.060916</td>
<td>97.80%</td>
</tr>
<tr>
<td>x-ray</td>
<td>84.18%</td>
<td>1.477452</td>
<td>86.22%</td>
</tr>
<tr>
<td>mr</td>
<td>49.67%</td>
<td>1.834388</td>
<td>51.22%</td>
</tr>
<tr>
<td>osdb</td>
<td>71.56%</td>
<td>2.588659</td>
<td>83.82%</td>
</tr>
<tr>
<td>dickens</td>
<td>49.87%</td>
<td>2.80192</td>
<td>58.26%</td>
</tr>
<tr>
<td>samba</td>
<td>60.74%</td>
<td>5.489460</td>
<td>77.61%</td>
</tr>
<tr>
<td>nci</td>
<td>31.57%</td>
<td>10.70759</td>
<td>35.18%</td>
</tr>
<tr>
<td>webster</td>
<td>48.94%</td>
<td>17.41480</td>
<td>62.80%</td>
</tr>
<tr>
<td>mozilla</td>
<td>66.65%</td>
<td>14.38173</td>
<td>82.20%</td>
</tr>
</tbody>
</table>

4.4 Security Analysis

On the basis of 1-gram probability model, in [14] or [15], each plaintext symbol corresponds to a static probability during the encryption processing; therefore, the distance between ciphertexts of the same plaintext symbol, obtained from different times of encryption, will be rather short. On the other hand, NCBP is adopted in our approach, depending on its predecessor each plaintext symbol in each LUT occupies a different of number of partitions. Therefore, the above-mentioned distance between ciphertexts of the same symbols will be relatively large. Recall that, the ciphertext of higher probability plaintext symbol will be smaller, and vice versa. In all chaotic-map based joint encryption and compression scheme, an attacker may trace back to find the original plaintext according to the magnitude of the ciphertext. Clearly, the larger the distance between two ciphertexts is, the less probable the same corresponding plaintext symbols will be.

As in [14] and [15], double precision format of 52 bits is used to represent the chaotic parameters b and x₀ in our software implementation. For further improving the security, different b and x₀ are adopted for constructed LUTs based on NCBP. The mapping between (b, x₀) and LUTs will be reassigned during encryption. Therefore, the total key space reaches 2*52*N bits, where N is the number of plaintext symbols (i.e. the number of LUTs). That is, the key space of our approach is, at least, comparable to that of 128-bit AES, for a reasonable N.

The same as all chaos-based joint compression and encryption schemes, the key sensitivity of our approach can be evaluated by checking the changed portion of files which are encrypted using two sets of secret keys with only a tiny difference. That is, the two resultant ciphertext sequences are compared bit-by-bit and the percentage of bit difference is then computed. The result is listed in Table 3. It follows that all the bit change percentages are very close to 50%, which justify the high sensitivity of the ciphertext to the key, in our approach.

To numerically evaluate the plaintext sensitivity, a bit is changed at different positions of the plaintext file, which is then encrypted using the same key. The two resultant ciphertext sequences are compared bit-by-bit. The averaged bit-change values (in percentage) can be found in the third column of Table 3. The measured average bit changes are close to 50%, which imply that the ciphertext is very sensitive to the plaintext.

Table 3 Percentages of bit change with respect to single bit difference in encryption keys and plaintext, where b and x₀ are the parameter and initial value of chaotic-map used for LUT searching.

<table>
<thead>
<tr>
<th>Sensitive</th>
<th>Proposed-16 map</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>b</td>
</tr>
<tr>
<td>xml</td>
<td>49.6443%</td>
</tr>
<tr>
<td>office</td>
<td>49.5035%</td>
</tr>
<tr>
<td>reynette</td>
<td>49.4221%</td>
</tr>
<tr>
<td>sao</td>
<td>49.8768%</td>
</tr>
<tr>
<td>x-ray</td>
<td>49.6275%</td>
</tr>
<tr>
<td>mr</td>
<td>49.7791%</td>
</tr>
<tr>
<td>osdb</td>
<td>49.7625%</td>
</tr>
<tr>
<td>dickens</td>
<td>49.5366%</td>
</tr>
<tr>
<td>samba</td>
<td>49.3928%</td>
</tr>
<tr>
<td>nci</td>
<td>49.0457%</td>
</tr>
<tr>
<td>webster</td>
<td>49.6045%</td>
</tr>
<tr>
<td>mozilla</td>
<td>49.7030%</td>
</tr>
</tbody>
</table>

5. Conclusions

Various algorithms for providing simultaneous compression and encryption, based on chaotic maps, have been investigated. For example, in [14], they dynamically update the LUT to achieve better compression result. And in [15], they reduce ciphertext by removing the same symbol in the searching trajectory. In our approach, instead of finding a new method to update the LUT in each iteration or using a different method to represent the ciphertext, a bi-gram NCBP model based multi-LUTs approach is proposed and investigated. Through a serious of experiments, the superiorities (in terms of both Compression ratio and Execution time) of our method to related state-of-the-art works are justified.

Acknowledgement

This paper is a partial result of the project “A study of Privacy Preserving Multimedia Security related Techniques”, under the contract NO: MOST103-2221-E-002-153-MY3

References


