Peer-to-Peer Networking and Applications (2021) 14:2651–2664
https://doi.org/10.1007/s12083-020-00981-8

An efficient secure data compression technique based on chaos and adaptive Huffman coding

Muhammad Usama[1] · Qutaibah M. Malluhi[1] · Nordin Zakaria[2] · Imran Razzak[3] · Waheed Iqbal[4]

Received: 7 April 2020 / Accepted: 4 August 2020 / Published online: 19 October 2020

Abstract
Data stored in physical storage or transferred over a communication channel includes substantial redundancy. Compression techniques cut down the data redundancy to reduce space and communication time. Nevertheless, compression techniques lack proper security measures, e.g., secret key control, leaving the data susceptible to attack. Data encryption is therefore needed to achieve data security in keeping the data unreadable and unaltered through a secret key. This work concentrates on the problems of data compression and encryption collectively without negatively affecting each other. Towards this end, an efficient, secure data compression technique is introduced, which provides cryptographic capabilities for use in combination with an adaptive Huffman coding, pseudorandom keystream generator, and S-Box to achieve confusion and diffusion properties of cryptography into the compression process and overcome the performance issues. Thus, compression is carried out according to a secret key such that the output will be both encrypted and compressed in a single step. The proposed work demonstrated a congruent fit for real-time implementation, providing robust encryption quality and acceptable compression capability. Experiment results are provided to show that the proposed technique is efficient and produces similar space-saving (%) to standard techniques. Security analysis discloses that the proposed technique is susceptible to the secret key and plaintext. Moreover, the ciphertexts produced by the proposed technique successfully passed all NIST tests, which confirm that the 99% confidence level on the randomness of the ciphertext.

Keywords Security · Secure compression · Encryption · Chaotic map

1 Introduction

The aim of securing data during storage or transmission is to increase the resistance level against various security attacks and protect the data from accidental modifications, illegal or unauthorized access [1]. Plenty of efforts were made in data security to overcome security challenges. The work in [2] highlights the importance of cryptography to protect data storage and transmission. However, cryptographic systems require high time and space complexity. Moreover, they have various performance and security limitations [3]. Since conventional cryptographic techniques may not fit well into the requirements of modern data storage and communication systems [4], many researchers are keen to investigate better security solutions. Subsequently, the underlying properties of the chaotic nonlinear systems such as randomness, sensitivity were found to be suitable for achieving cryptographic capabilities [5]. Many researchers have developed efficient cryptographic techniques based on chaos theory, and their security features were analyzed in detail in [6]. On the other hand, data sizes can be controlled by applying various data compression techniques during data storage and transmission, e.g., Huffman Coding (HC), Arithmetic Coding (AC), Lempel Zip (LZ) [7, 8]. Data compression transforms input data (source message or file) into another form of data, which is smaller in size (compressed or small output). Compression systems are based on different ideas and suitability for different types

This article is part of the Topical Collection: *Special Issue on Security of Mobile, Peer-to-peer and Pervasive Services in the Cloud*
Guest Editors: B. B. Gupta, Dharma P. Agrawal, Nadia Nedjah, Gregorio Martinez Perez, and Deepak Gupta

✉ Muhammad Usama
usama.khanzada@hotmail.com

1 Qatar University, Doha, Qatar
2 Universiti Teknologi PETRONAS, Perak, Malaysia
3 Deakin University, Melbourne, Australia
4 University of the Punjab, Lahore, Pakistan

of input data and produce separate outputs. However, all compression systems are grounded on the same principle, that is, compact input data by eliminating redundancy.

Hence, both data compression and encryption are essential, but their implementation is difficult and complicated, which requires extensive computation power and resources to process a large amount of data. Nevertheless, these two different processes can be employed sequentially for the same dataset. The sequential implementation of compression and encryption techniques can be classified into two approaches, i.e., encryption-first and compression-first approaches. In the encryption-first approach, compression is embedded into encryption for achieving a higher level of data security. However, these techniques have insufficient and relatively reduced compression ratio and performance as compare to standard compression techniques. Here, compression is by reducing redundancy present in input data, while encryption also works by reducing redundancy found in input data. However, encryption does not, in general, reduce the size of the input data but introduces the significant randomness in data, thus deeming the compression process in vain. In the compression-first approach, encryption is embedded in compression but suffers from severe processing and security limitations. It needs the output of the compression operation to be piped to the encryption operation, which makes this approach more complicated and time-consuming.

This work focuses on the problems of data compression and encryption collectively so that compression is carried out according to a secret key such that the output will be both encrypted and compressed in a single step. Towards this end, this work proposes an efficient, secure data compression technique by introducing cryptographic capabilities in adaptive Huffman coding using chaotic pseudorandom keystream generator and chaotic S-Box. The proposed technique intends to achieve confusion and diffusion properties of cryptography into the compression process of adaptive Huffman coding and overcome the performance issues of the prior techniques.

The paper is organized as follows. Section 2 discusses the related work. Section 3 introduces some preliminary components and work principles related to the proposed work. The proposed technique and its various parts are presented in Section 4. Section 5 analyses the compression and security of the proposed technique. Finally, we provide concluding remarks in Section 6.

2 Related work

Next-generation data storage and network systems are proliferating in unique manners due to the proliferation of diverse and ubiquitous connectivity across mobile systems, networks, and sensors. Moreover, data is growing into unprecedented ranges of Terabytes, Petabytes, or even Exabytes, where users require the continuing at full strength and intensity of data. The critical issue now is how to ensure efficient and reliable usage of resources while preserving the data storage and transmission security. In such a situation, data encryption and compression techniques are often applied together to perform secure data compression. In this direction, many chaotic systems were applied because of operational efficiency for achieving secure data compression [9, 10]. Compared with the conventional sequential implementation, simultaneous data compression and encryption is aimed to provide data security while also minimizing data redundancy. Notably, most of the existing research works were focused on integrating encryption into conventional data compression techniques [11–13]. Since both encryption and compression have a certain resemblance and common objective in the sense of secrecy, which removes redundant data. Furthermore, it is easy to turn the compression algorithm into an encryption algorithm by introducing a secret key control [12, 14].

The secure data compression techniques based on HC can be found in the literature [11–13]. In [15], a unified data compression and encryption technique is introduced that swaps Huffman tree (HT) branches, left and right, by a secret key. Furthermore, multiple mutated trees were generated for introducing confusion and diffusion in the compression process. However, all the mutated trees generated by this approach were fixed in size. Thus, a new optimized multiple HTs technique [13] was suggested to overcome the fixed size mutated trees issue by statistical-model-based compression for generating the mutated tables. Afterward, enhanced technique over [15], was proposed in [11] using chaotically mutated Huffman trees. It overwhelmed the multiple codeword issues in [15] by increasing the keyspace. The idea of this technique was like Huffman tree mutation, but it controls the tree updates by chaotic pseudorandom keystream. However, this technique suffers from a known-plaintext attack [12]. Further, an improved technique over [11] was presented in [12] by incorporating two chaotic functions to increase resistant level against known-plaintext attacks. Still, the length of the codewords attained by the statistical model continued unaffected, and the resulting system was found vulnerable against known-plaintext attacks [14].

Nevertheless, secure data compression techniques based on HC still possess stern security vulnerabilities [16]. These techniques implement the concept of the tree mutation for achieving data compression and encryption at the same time. In contrast, data encryption is achieved by incorporating a secret key mechanism to control and randomize the Huffman tree process. Therefore, the integrating of encryption features in HC has been primarily limited to the control of Huffman tree branches using the secret key. Further discussion about the most recently used such secure data compression techniques based on HC and chaos and their strengths and weaknesses are presented in Table 1.

3 Preliminaries

Chaos theory is well-known due to its ability to effectively change to initial condition and parametric control values, which makes the entire system unpredictable and random [17]. The nonlinear dynamical systems of chaos theory can be employed for cryptographic purposes when they successfully implemented in infinite precision computing to achieve confusion and diffusion [18]. Thus, a straightforward and common application of chaos is to design a pseudorandom keystream generator [19]. The chaoticity of one-dimensional chaotic maps can be tested easily with different desirable properties in them, and it can be accessible to rigorous mathematical analysis and experiments such as Lyapunov exponent [20] and Bifurcation analysis [21]. This work implies the chaotic Logistic map to generate pseudorandom keystream due to its complex, chaotic behavior.

3.1 Chaotic logistic map

The Chaotic Logistic Map (CLM) is popular one-dimensional chaotic map that confirms effective change to initial condition and parametric control values [22]. CLM proves complex chaotic behavior and satisfies various cryptographic basics such as unpredictability and randomness [18, 22]. Equation 1 defines CLM:

$$x_{n+1} = F_\lambda(x_n) = \lambda x_n(1 - x_n) \tag{1}$$

where $0 < x_n \leq 1$; $n = 0, 1, 2 \ldots\ldots$; λ is a parametric control value in range $0 < \lambda \leq 1$; and x_0 is the initial condition. The rigorous Bifurcation and Lyapunov exponent results are presented in Figs.1 and 2, respectively that confirms chaotic behavior of the CLM beyond parametric control value λ at 3.57, where, orbits $\{x_n\}_{n=0}^{\infty}$ are distributed in a uniform manners between 0 and 1. The design proposed in this work employs the value of $\lambda = 3.57$ to generate a pseudorandom keystream for stealth key control to shuffle or randomize the symbols codes during Adaptive Huffman Tree (AHT) generation process.

3.2 Pseudorandom Keystream Generator based on CLM

The CLM was carefully implemented for generating pseudorandom keystream for key control in the proposed work. CLM requires two input parameters, an initial value x_0 and parametric control value λ to iterate and obtain the next value x_n. Further, these two inputs x_0 and λ are considered as a secret key. The x_n value is then used to generate c a threshold t value. The threshold t value is set as $t = 0.5$ (according to uniform probability model) as per (2):

$$B_n = \begin{cases} 0 & 0 \leq x_n < t \\ 1 & t \leq x_n \leq 1 \end{cases} \tag{2}$$

The pseudorandom keystream is obtained from real values x_n of CLM by comparing with threshold are presented in Fig. 3.

Table 1 A discussion about some recently used approaches based on HC

Technique	Strength	Weakness
Design of integrated multimedia compression and encryption systems [15].	1) It provides high compression ratio. 2) It swaps HT branches, left and right, using the control key.	1) Poor processing speed. 2) All the mutated trees generated by this approach were fixed in size.
Joint compression and encryption using chaotically mutated Huffman trees [11].	1) It overcomes multiple codeword issues and enlarge the key space of [15]. 2) It controls the tree updates using the PRKG.	1) Poor processing speed. 2) Vulnerable to known-plaintext attack.
Securing Multimedia Transmission Using Optimized Multiple Huffman Tables Technique [13].	1) Optimized multiple HTs technique. 2) It overcomes the fixed size mutated trees issue by statistical model for generating the mutated tables.	1) The length of the codewords obtained by the statistical model remained unchanged. 2) The resulting system was vulnerable to known-plaintext attack.
A Chaos-based Joint Compres-. sion and Encryption Scheme Using Mutated Adaptive Huffman Tree [13]	1) Two chaotic functions were adopted to avoid known-plaintext attacks.	1) It offers very poor compression and decompression efficiency. 2) Weak security issues.

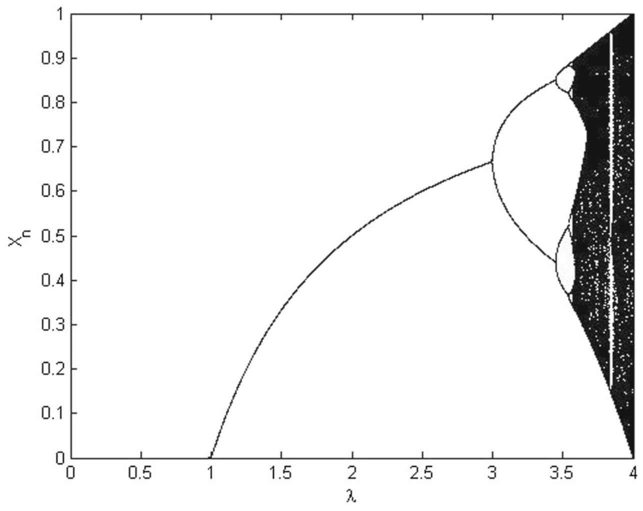

Fig. 1 Bifurcation diagram of the CLM [14]

The process of generating pseudorandom keystream by CLM is described as follows:

Inputs and outputs: The PRKG based on CLM requires two inputs, secret key K and length L of the pseudorandom keystream, where, inputs $x_0$and λ are considered as a secret key K. KS is an output pseudorandom keystream.

Step 1. Initialize the counter value $i = 1$ and output pseudorandom keystream KS to empty.
Step 2. Calculate the value by CLM as: $x_1 = \lambda x_0(1-x_0)$.
Step 3. If obtained x_1 value is greater or equal to 0.5, then go to Step 5 else go to Step 6.
Step 4. Concatenate 1 with KS as $KS = KS \parallel 1$, (where the symbol " $\parallel$ " represents concatenation operation) and go to Step 6.
Step 5. Concatenate 0 with KS as $KS = KS \parallel 0$.

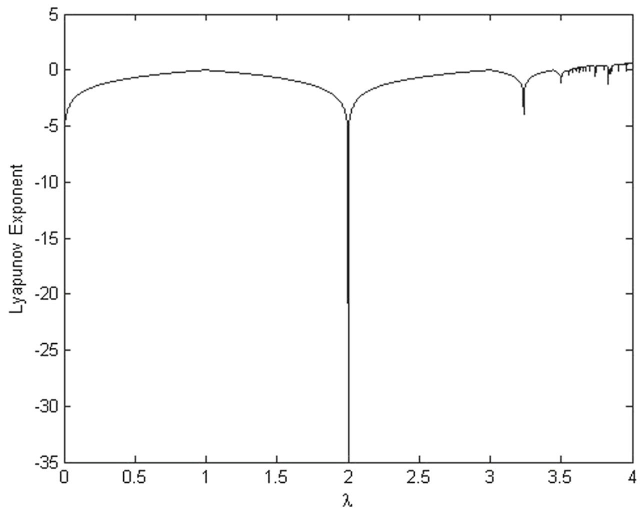

Fig. 2 Lyapunov exponent of the CLM [14]

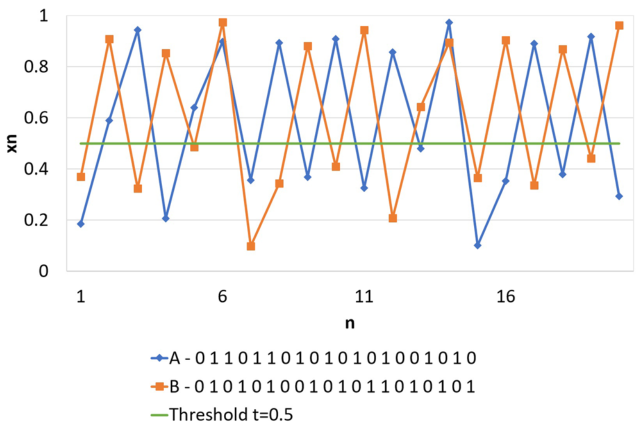

Fig. 3 Trajectory of two pseudorandom keystreams, where $x_0 = 0.19$ for blue, $x_0 = 0.37$ for orange and $\lambda = 3.99$

Step 6. Set $x_0 = x_1$ (to keep x_1 value for the next iteration of CLM in Step 2).
Step 7. Increment the counter value as $i = i + 1$.
Step 8. Check, if $i \geq L$, then go to Step 9 else go to Step 2.
Step 9. End of the pseudorandom keystream generation process.

3.3 Chaotic S-Box

A substitution box (S-Box) plays a central role in many conventional symmetric-key techniques in order to prevent differential and linear cryptanalyses [14]. Currently, these two are an extremely effective cryptanalyses attack [23]. Nevertheless, the major issue of the conventional S-Boxes is the static behavior and utilized as a fixed-size lookup table without any secret key control. Thus, chaotic systems were employed to produce dynamic S-Box with key control instead of being fixed [24, 25]. This work uses an efficient method proposed by Usama et al. [25] for constructing S-Boxes to perform data substitution and introduce confusion and diffusion in the proposed work without compromising compression capabilities. The presented method requires a secret key K as an input of size 106-bit to generate dynamic S-Box. Table 2 presents the output S-Box constructed using Usama et al. method [25], where, inputs $x_0$and λ are set to be $x_0 = 0.85$ and $\lambda = 0.99$ as a secret key K.

4 The proposed work

This work focuses on incorporating the recent results of chaos theory, which has proven to exhibit strong cryptographic properties into Adaptive Huffman Coding (AHC) to overcome the security and performance issues of the prior techniques. As per the literature review, the chaotic behavior of the CLM is favorable to achieve the

Table 2 The output S-Box of the proposed method

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	64	FB	21	94	EB	29	A8	CF	77	FE	03	45	DC	55	F0
1	19	83	FA	2B	AD	C5	91	EE	1E	8F	F1	17	7C	FD	0E	8E
2	16	78	05	4B	E5	3B	CD	7E	FC	7A	0A	59	F4	24	9E	DF
3	38	C8	8B	0B	5C	F6	4A	E3	3F	D3	6C	E4	3C	49	40	D5
4	67	12	70	3A	CB	2C	AE	C4	28	A6	D2	CA	85	F8	F9	2F
5	B5	B6	B4	B8	C3	96	E9	2E	B2	BB	04	48	E0	47	41	D6
6	66	18	7F	1A	84	B7	B0	BF	A0	DB	56	79	06	4D	E7	34
7	9F	DD	51	EC	25	50	A7	D1	72	FF	C6	CC	82	A9	7D	11
8	3D	74	39	8A	F5	0D	62	D0	2A	AB	36	95	EA	AF	C2	99
9	E6	C7	8D	F2	6E	02	20	93	23	9B	E2	42	D9	5D	F7	43
A	DA	5A	C	5E	01	3E	6F	00	0F	14	75	73	8C	F3	1D	15
B	76	4C	37	90	EF	1C	89	07	80	D4	6B	33	BD	A4	65	D8
C	5F	C9	86	2D	BC	26	A2	60	C0	9D	DE	4F	E1	44	57	13
D	9C	61	10	69	8	52	4E	71	54	27	A3	1B	87	D7	22	98
E	35	68	6A	30	C1	81	97	E8	32	BA	AA	46	ED	53	B9	AC
F	92	A1	5B	09	A5	6D	CE	1F	7B	58	31	88	BE	9A	B1	B3

confusion and diffusion properties of cryptography. Hence, an efficient, secure data compression technique (ESDC) is introduced, such that output is secured and compressed simultaneously (Fig. 4).

HC is a well-known data compression technique. It consists of two main parts, i.e., the statistical model known as the Huffman tree and the compression engine. It utilizes the Huffman tree to code the data symbols by assigning shorter codewords to more frequent symbols. However, to accomplish the compression process, it requires two times the scanning of input data. First, it needs to create a statistic model from input data symbols to construct the Huffman tree. Second, the compression engine assigns a shorter codeword to input data symbols using a constructed Huffman tree. An improved version of the HC was presented in [26] that does not require two times scanning for data compression, known as AHC.

AHC also consists of two main parts, i.e., the statistical model and the compression engine. However, it mutates the statistical model known as the adaptive Huffman tree and assigns the shorter codewords at the same time. Thus, it needs to scan input data only one time to complete the compression process. AH tree (AHT) plays an essential role in the AHC coding process to achieve efficient compression efficiency (when AHT matches the exact statistical characteristics of data). More importantly, there are three advantages of these two separate parts that are used as design principles to implement the proposed work:

1. AHC allows to shuffle or randomize the probabilities of the symbols during the AHT generation process, which leads to the achievement of Shannon's suggested fundamental properties of confusion and diffusion.
2. AHC allows symbol substitution or changes to the order of the symbols before updating AHT and data encoding, which provide significant flexibility to introduce complexities in the compression process without compromising compression capabilities.
3. AHC allows for integrating different cryptographic processes. For example, it will enable masking the coding output with some pseudorandom keystream that can enhance encryption quality.

Thus, this work carries out the following three operations in AHC to implement the above mentioned three operations, respectively, so that output is secured and compressed, simultaneously with key control:

1. Proposed work employed CLM to introduce key control in the compression and decompression processes of the AHC, named as secure adaptive Huffman coding (See Section 4.1).
2. The proposed work incorporates a chaotic S-Box construction method [25] to perform data substitution without compromising compression capabilities.
3. The proposed work implements a masking pseudorandom keystream using PRKG based on CLM (presented in Section 3.1) that enhances encryption quality.

The component-level block diagram of the proposed ESDC technique with their sequence to perform secure data compression and decompression is presented in Fig. 5. It consists of three main components chaotic S-Box, secure adaptive Huffman coding, and PRKG. The description of each component is provided in the following sections. The process of the ESDC technique to perform secure data compression by chaotic S-Box, secure adaptive Huffman coding, and PRKG is described as follows:

Inputs and outputs: The secure data compression is performed by introducing three secret keys $K1$, $K2$, and $K3$ (each key is a combination of the initial value x_0 and control parameter λ for their respected chaotic map). $K1$ is used for chaotic S-Box [25] to construct dynamic S-Box, and $K2$ and $K3$ are used to generate the pseudorandom keystreams using CLM. Here, I is the input data, L represents the length of the input data, and C represents the output (compressed and encrypted data).

Step 1. Generate the initial AHT.
Step 2. Generate the chaotic S-Box using secret key $K1$. (See: Usama et al. S-Box construction method [25])
Step 3. Initialize the counter value $i = 1$ and output data D to empty.
Step 4. Read the input symbol I_i from input data I.
Step 5. Substitute the input symbol I_i using chaotic S-Box, the output symbol is s_i.
Step 6. Encode the s_i by secure adaptive Huffman coding method using key $K2$ (See: Section 4.1.1). The

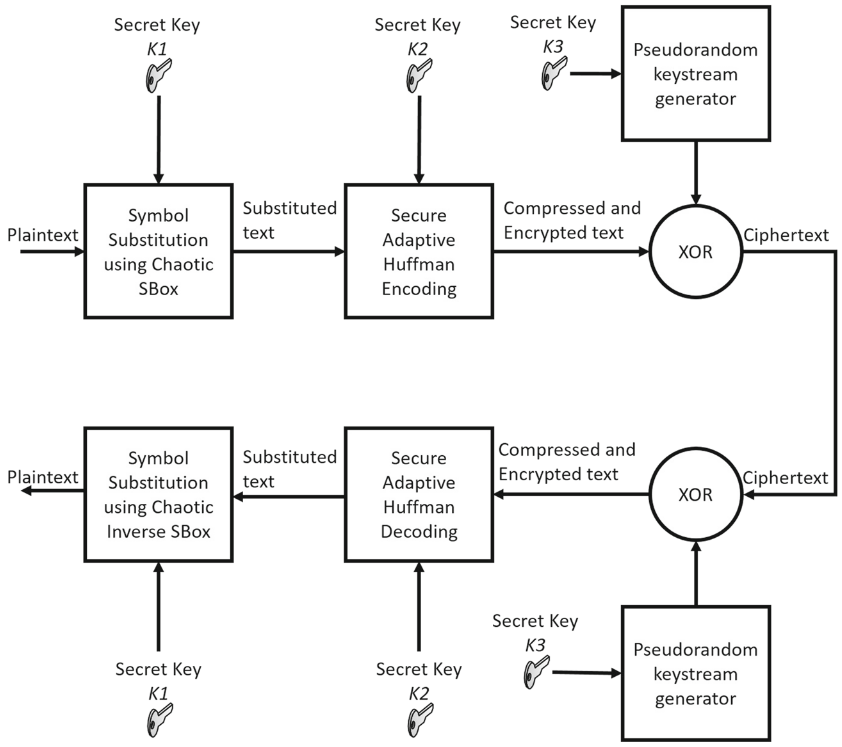
Secret Key K1
Secret Key K2
Secret Key K3
Pseudorandom keystream generator
Plaintext
Symbol Substitution using Chaotic SBox
Substituted text
Secure Adaptive Huffman Encoding
Compressed and Encrypted text
XOR
Ciphertext
Symbol Substitution using Chaotic Inverse SBox
Secure Adaptive Huffman Decoding

Fig. 4 Component level block diagram of the ESDC

output code c_i. Concatenate the c_i with the output data as $D = D \parallel c_i$ (where the symbol " $\parallel$ " represents concatenation operation).

Step 7. Increment the counter value as$i = i + 1$.

Step 9. Check, if $i > L$, then go to Step 9 else go to Step 4.

Step 10. Finally, key $K3$ generates a pseudorandom keystream KS using PRKG based on CLM (See: Section 3.2) and masks the completed output data D. Simple XOR operation is used to mask the output of D, which enhances overall randomness. This step produces the final ciphertext C as $C = D \otimes KS$.

Step 3. End of secure data compression.

The secure data decompression is like a secure data compression technique. It just reverses the order of secure

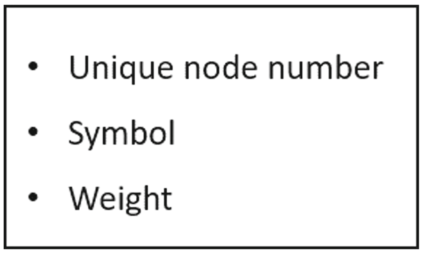

Fig. 5 AHT node structure

data compression. It requires the same three secret keys $K1$, $K2$, and $K3$ as that used in the encoding process. Firstly, it decrypts the ciphertext using PRKG based on CLM with the secret key $K3$, then decodes the data using secure adaptive Huffman Decoder with secret key $K2$, and then performs inverse substitution using chaotic S-Box by Usama et al. method [25] with secret key $K1$. Finally, it produces the original plaintext data. The process of the secure data decompression is described as follows:

Inputs and outputs: The secure data decompression is performed by the same three secret keys $K1$, $K2$, and $K3$ (each key is combination of the initial value x_0 and control parameter λ for their respected chaotic map) as that used in secure data compression. Where C is the input ciphertext data, and P represents the output plaintext data.

Step 1. Generate keystream using PRKG based on CLM with $K3$ (See: Section 3.2). It produces the keystream KS. This KS is used to unmask the ciphertext C to get the compressed data D as $D = C \otimes KS$.

Step 2. Generate the initial AHT.

Step 3. Generate the chaotic S-Box using secret key $K1$. (See: Usama et al. S-Box construction method [25])

Step 4. Initialize the plaintext data P to empty.

Step 5. Read the compressed code c from compressed data D.

Step 6. Decode the c by secure adaptive Huffman coding method using key $K2$, the output code s.
Step 7. Substitute the output code s, using chaotic S-Box, the outputs symbol is I. Concatenate I with P as $P = P \parallel I$, (where the symbol " $\parallel$ " represents concatenation operation).
Step 8. Check, check the end of the file; if the file ends, then go to Step 9 else go to Step 5.
Step 9. End of secure data decompression.

4.1 Secure adaptive Huffman coding

As mentioned earlier, AHC mutates the AHT and assigns the shorter codewords to produce compress output at the same time. AHT is a binary tree that has the shortest weighted path length. The node structure contains node number, symbol, and weight, as shown in Fig. 4. Thus, each node must have a unique number and symbol with its weight in AHT.

AHT generation process starts with a "RAW" node and keeps updating the AHT after performing encoding or decoding input symbol, according to the data statistics [26]. The details of data encoding and decoding processes are provided in the following sections. The process of the updating AHT is described as follows:

Inputs and outputs: The AHT updating process requires a symbol as an input and outputs updated AHT.

Step 1. Check, if the input symbol is included in AHT, then go to Step 10 else go to Step 2.
Step 2. Generate a new node for input symbol with weight 1.
Step 3. Generate a new "RAW" node.
Step 4. Add new symbol node and new "RAW" node as children nodes of the old "RAW" node. Continue with this node.
Step 5. Increase the node weight and go to Step 10.
Step 6. Continue with the parent node.
Step 7. Check for the most weighted node; if it is the highest weighted node, then go to Step 9 else go to Step 8.
Step 8. Swap the current node with the most weighted node.
Step 9. Increase the node weight.
Step 10. Check for the root node; if it is the root node, then go to Step 11 else go to Step 6.
Step 11. End of the AHT update process.

The proposed work incorporates a stealth key control to shuffle or randomize the symbols codes during the AHT generation process. As per the literature review, the chaotic behavior of the CLM is in favor of achieving the confusion and diffusion properties of cryptography. To randomize the compression process, two rules are incorporated in the update process of the AHT by pseudorandom keystream generated using CLM. CLM is an iterative map that requires two input values x_0 and λ to obtain x_n values from Eq. 1. The obtained x_n value is used to apply the rule as given below:

1. If obtained x_n value is greater or equal to 0.5 then add new symbol node on the right and RAW node on the left.
2. If obtained x_n value is less than 0.5 then add a new symbol node on the left and RAW node on the right.

Here, the uniform probability model is adapted to assign an equal probability to each rule. Moreover, the input parameters $x_0$and λ of the CLM are used as a secret key. The process of updating AHT using CLM is described as follows:

Inputs and outputs: The process of updating AHT using CLM is performed by introducing input secret key K (which is a combination of the initial value x_0 and control parameter λ for CLM). It requires a symbol as an input and outputs updated AHT.

Step 1. Check, if the input symbol is included in AHT, then go to Step 2 else go to Step 2.
Step 2. Generate a new node for input symbol with weight 1.
Step 3. Generate a new "RAW" node.
Step 4. Calculate the new value by CLM as: $x_1 = \lambda x_0(1 - x_0)$. As mentioned earlier, the input parameters $x_0$and λ of the CLM are used as a secret key to new obtain x_1.
Step 5. If obtained x_1 value is greater or equal to 0.5, then go to Step 6 else go to Step 7.
Step 6. According to rule 1, add a new node on the right and new "RAW" node on left as children nodes of the old "RAW" node. Continue with this node and go to Step 8.
Step 7. According to rule 2, add a new node on the left and new "RAW" node on the right as children nodes of the old "RAW" node. Continue with this node.
Step 8. Set $x_0 = x_1$ (to keep x_1 value for the next iteration of CLM in Step 4).
Step 9. Increase the node weight and go to Step 14.
Step 10. Continue with the parent node.
Step 11. Check for the most weighted node; if it is the highest weighted node, then go to Step 13 else go to Step 12.

Step 12. Swap the current node with the most weighted node.
Step 13. Increase the node weight.
Step 14. Check for the root node; if it is root node, then go to Step 15 else go to Step 10.
Step 15. End of the AHT update process.

Note: In above mentioned AHT updating steps, AHC and AHT implementation details are not included. This work only focused on providing the difference and modification in the AHC technique for secure data compression. Implementation details of the AHC and AHT can be found in [26].

4.1.1 Encoder

The data encoding process requires a symbol as an input to produce output compressed code using AHT [26]. In the end, it updates the AHT for input symbol [26]. The process of the data encoding is described as follows:

Inputs and outputs: Encoder require symbol as an input and outputs compressed code.

Step 1. Check, if the input symbol is new using AHT, then go to step 2 else go to step 3.
Step 2. Output the code for RAW, followed by the fixed code of the input symbol.
Step 3. Output the code of the existing symbol from AHT.
Step 4. Update the AHT for input symbol using the method described in the secure adaptive Huffman coding section.

4.1.2 Decoder

Like the data encoding process, the decoder takes the compressed code as input to decode the original symbol using AHT [26]. The decoding process also requires updating AHT after decoding the original symbol according to the data statistics [26]. The process of data decoding is described as follows:

Inputs and outputs: Decoder require compressed code as an input and outputs decoded symbol.

Step 4. Decode the code to get a symbol using AHT.
Step 2. Check, if the input code is RAW, then go to step 3 else go to step 4.
Step 3. Use the fixed code to decode the code to get the symbol.
Step 4. Update the AHT for the decoded symbol using a method described in the secure adaptive Huffman coding section.

5 Experiment analysis

The experimental analysis begins with defining the experiment setup for benchmarking the proposed technique. The required input files, software, and description of input data generation are also described. The performance of the proposed technique was evaluated to justify their superiorities over those similar existing techniques. Also, the strength of the proposed technique, as well as its various parts, are analyzed concerning the related parameters per the definitions in this paper.

5.1 Experiment design

The proposed technique and its different related parts were implemented using Java. A Personal Computer running Windows 7 with 3GB DDR3 RAM and Pentium-IV 2.4 MHz processor is used. Calgary Corpuses are commonly used to measure the performance of any compression technique as the standard benchmark input data. Hence, standard Calgary Corpuses [27] were used in the experiments to evaluate the performance of the proposed technique and their various aspects. The data compression techniques such as AC and AHC are well-known for producing comparatively good compression efficiencies. Hence these compression techniques are taken as the base reference against the performances of the proposed technique. Furthermore, prior techniques for joint operation of the data compression and encryption are also taken as a reference against the performance efficiencies of the proposed technique. These techniques are Chaotic Mutated Adaptive Huffman Tree (CMAHT) [12], Chaotic Huffman Tree (CHT) [11], and Simultaneous Arithmetic Coding and Encryption (SACE) [10]. All techniques are implemented using Java language and were benchmarked using Calgary Corpus files [27].

5.2 Compression efficiency analysis

The primary concern of any data compression technique is to reduce storage space. The ability to reduce the data size by a compression technique can be measured by determining its compression efficiency in terms of space-saving (%) capability [28, 29]. Literature studies indicate that any compression technique that produces higher space-saving (%) is supposed to be robust and efficient for saving disk space and reducing transmission overheads [28, 29]. In this section, the discussion of space-saving (%) is included to evaluate and show the compression efficiency of the proposed technique in comparison to well-known compression techniques and prior techniques for performing the joint operation of compression and encryption. Space-saving (%), produced by the proposed technique and other existing techniques, was calculated using Eq. 3 [29]:

$$Space\ Saving\ (\%) = \left(1 - \frac{Input\ data\ size}{Output\ data\ size}\right) \times 100 \tag{3}$$

The comparison results are presented in Table 3. Results demonstrate that the proposed technique produced similar space-saving (%) to AC and AHC techniques with slight variations. Thus, the space-saving (%) results identified that the proposed technique has acceptable compression efficiency while also providing adequate security. The detailed security analysis is presented in the security analysis section.

5.3 Processing time analysis

This section presents the processing efficiency offered by the proposed and various existing techniques. In any data storage and communication system, the processing time is an important factor as the entire data need to be stored or transmitted for processing within a specified time frame [1]. Thus, it is essential to measure the processing time of the proposed technique to justify performance efficiency. Equation 4 is used to calculate processing time:

$$Processing\ time = Process\ end\ time - Process\ start\ time \tag{4}$$

The performances of the proposed technique were compared with their corresponding current simultaneous data compression and security techniques. Tables 4 and 5 demonstrate a processing time comparison between the proposed and CHT, CMAHT, and SACE techniques running on standard Calgary Corpus input files. Results showed that the proposed technique was the fastest while performing data compression and encryption simultaneously, compared to the corresponding existing techniques.

Calgary Corpus test files were compressed and encrypted by standard AC and AHC with AES as well. Tables 6 and 7 give a comparison for processing time between the proposed, AC and AHC with AES techniques. These results showed that both well-known compression techniques AC and AHC were the fastest while performing only data compression without any encryption capabilities. Results show that the proposed technique requires less processing time compared to two separate operations. In a nutshell, experimental results show that AC and AHC had the fastest compression speeds, while proposed technique is in third place. Apart from well-known coding techniques, the proposed technique offered better time efficiency than the other existing simultaneous compression and encryption techniques. Hence, it can be claimed that the proposed technique is efficient compared to the corresponding related techniques to perform simultaneous compression and encryption by reducing the space and time overheads.

Table 3 Comparison of proposed and existing techniques with respect to space-saving (%)

File	Compression techniques		Existing simultaneous compression and encryption techniques			Proposed technique
	AC	HC	CHT	CMAHT	SACE	
bib	31.64	31.96	31.96	31.86	31.99	31.85
book1	40.76	40.46	40.46	40.41	40.03	40.41
book2	37.68	37.42	37.42	37.23	36.70	37.23
geo	26.21	26.67	26.67	26.48	26.15	26.48
news	32.91	32.68	32.68	32.47	31.81	32.47
obj1	18.80	22.07	22.07	18.45	19.41	18.45
obj2	19.56	19.48	19.48	16.83	18.46	16.84
paper1	33.62	34.65	34.65	33.79	33.57	33.82
paper2	38.91	39.41	39.41	39.10	38.79	39.10
paper3	37.20	38.51	38.51	38.19	37.57	38.19
paper4	31.06	36.47	36.47	36.01	36.19	36.02
paper5	27.46	33.46	33.46	32.42	32.86	32.42
paper6	32.50	34.10	34.10	32.60	33.31	32.69
pic	81.20	77.34	77.34	76.90	77.65	76.91
progc	30.56	32.12	32.12	31.30	31.41	31.35
progl	36.30	37.01	37.01	36.01	36.01	36.03
progp	34.51	35.70	35.70	34.98	36.85	34.98
trans	27.33	27.71	27.71	26.69	27.59	26.73

Table 4 Processing time (seconds) offered by proposed and existing techniques to perform secure data compression

File	Existing simultaneous compression and encryption techniques			Proposed technique
	CHT	CMAHT	SACE	
bib	393.73	988.26	218.43	137.32
book1	998.53	1593.58	375.88	192.33
book2	303.96	744.48	272.00	125.12
geo	150.24	29.08	47.14	28.69
news	155.90	342.90	158.66	115.19
obj1	42.02	87.46	11.41	9.37
obj2	201.63	160.61	96.17	63.81
paper1	25.89	33.82	18.56	14.38
paper2	31.01	46.69	18.37	18.59
paper3	20.95	23.50	33.64	12.81
paper4	9.27	12.53	13.29	7.71
paper5	8.17	26.89	5.16	7.48
paper6	22.94	68.14	4.25	12.73
pic	101.95	231.03	130.09	53.58
progc	19.01	29.80	13.71	12.83
progl	29.75	86.85	26.51	17.57
progp	22.26	26.54	20.56	14.57
trans	87.45	281.30	147.87	26.83

5.4 Security analysis

Along with an adequately sized key, the cryptographic technique should produce an output that is indistinguishable from random for any input data to prevent an attacker to discover the input data statistics and to minimize any relationship within cipher output. Furthermore, the cipher output should be highly sensitive and dependent upon the

Table 5 Processing time (seconds) offered by proposed and existing techniques to perform secure data decompression

File	Existing simultaneous compression and encryption techniques			Proposed technique
	CHT	CMAHT	SACE	
bib	194.51	488.22	266.75	172.29
book1	288.15	609.86	669.18	135.94
book2	211.77	518.67	472.08	99.39
geo	43.87	8.49	92.44	27.57
news	143.74	316.17	307.04	65.88
obj1	15.97	33.25	23.61	8.94
obj2	115.87	92.30	220.95	47.90
paper1	18.93	24.73	46.86	13.47
paper2	27.47	41.35	39.44	16.70
paper3	17.71	19.87	66.05	14.80
paper4	7.70	10.41	27.55	7.54
paper5	5.20	17.12	17.01	7.00
paper6	16.52	49.09	9.00	10.81
pic	68.37	154.94	316.36	40.28
progc	14.64	22.96	37.32	11.19
progl	27.41	80.02	61.15	16.86
progp	16.67	19.87	44.53	13.34
trans	41.43	133.26	126.39	21.03

Table 6 Processing time (seconds) offered by compression, encryption, proposed and sequential compression and encryption techniques to perform compression and encryption

File	Compression techniques		Encryption technique	Sequential compression and encryption		Proposed technique
	AC	HC	AES	AC + AES	HC + AES	
bib	135.39	126.76	332.54	281.71	319.84	137.32
book1	220.51	143.80	1127.45	1007.68	823.21	192.33
book2	165.81	92.53	894.76	747.89	703.13	125.12
geo	26.07	43.30	174.85	170.45	160.64	28.69
news	93.83	65.73	537.14	467.30	458.55	115.19
obj1	8.05	9.66	38.39	59.87	34.92	9.37
obj2	58.47	44.34	341.66	375.91	313.83	63.81
paper1	11.62	9.84	70.72	92.02	53.46	14.38
paper2	19.93	15.94	128.47	121.14	85.22	18.59
paper3	13.64	8.46	58.85	64.05	46.89	12.81
paper4	5.08	4.28	17.68	23.04	16.55	7.71
paper5	3.21	3.15	15.50	26.99	20.39	7.48
paper6	12.62	8.64	67.23	91.38	84.36	12.73
pic	77.69	43.04	791.92	248.63	212.00	53.58
progc	9.14	7.38	48.79	50.89	40.33	12.83
progl	16.75	13.07	100.10	110.94	77.74	17.57
progp	11.90	8.94	87.32	75.79	47.95	14.57
trans	92.16	14.19	132.46	179.57	102.63	26.83

Table 7 Processing time (seconds) offered by compression, encryption, proposed and sequential compression and encryption techniques to perform decompression and decryption

File	Compression techniques		Encryption technique	Sequential compression and encryption		Proposed technique
	AC	HC	AES	AC + AES	HC + AES	
bib	115.38	74.98	405.93	290.44	286.42	172.29
book1	298.13	90.40	1806.12	1424.10	1169.79	135.94
book2	197.64	47.19	1480.98	1101.37	947.99	99.39
geo	37.78	9.01	246.14	244.78	185.26	27.57
news	127.66	63.98	852.26	729.25	669.72	65.88
obj1	11.00	4.60	49.73	49.34	46.85	8.94
obj2	90.71	48.40	588.10	551.81	531.43	47.90
paper1	17.65	4.32	121.88	100.96	81.18	13.47
paper2	26.03	6.79	224.20	140.99	135.36	16.70
paper3	15.24	3.80	99.81	90.41	71.07	14.80
paper4	7.53	3.25	30.73	27.33	25.58	7.54
paper5	4.14	1.39	27.42	32.04	25.82	7.00
paper6	21.11	4.54	131.73	90.22	110.73	10.81
pic	155.76	13.92	1229.81	375.93	291.22	40.28
progc	17.99	4.24	90.51	87.13	73.97	11.19
progl	26.52	7.11	171.33	157.86	130.67	16.86
progp	18.02	4.40	129.87	109.88	80.33	13.34
trans	50.98	7.55	216.06	208.55	179.55	21.03

key and plaintext such that a single bit is altered within the key or plaintext, the cryptographic technique should produce entirely different output every time. The proposed technique was designed to achieve high security while reducing data storage and transmission overheads. The security analysis of the proposed technique and its various parts are evaluated in the following sections through several assessment parameters.

5.4.1 Key space analysis

Cryptographic techniques should have sufficiently large enough keyspace to prevent the attacker from decoding the ciphertext or reveal the key in any reasonable time [17]. Henceforth, this section discusses the keyspace of the proposed technique. The proposed technique utilizes CLM for two purposes. First, CLM was employed to generate pseudorandom keystreams and second to randomize the compression process in the update process of the AHT. The CLM is iterative chaotic map (start from two input parameters, an initial value x_0 and a control parameter λ) and obtain x_n values from Eq. 1. The obtained x_n values from CLM are then converted to generate pseudorandom keystreams. These input parameters $x_0$and λ are used as a secret key for generating pseudorandom keystreams from CLM. As mentioned earlier, this work employed an efficient method proposed by Usama et al. in [25] for constructing S-Boxes to perform data substitution. This method is based on the mixing property of the chaotic Sine map, where the initial value x_0 and control parameter λ were used as a secret key K. Thus, the proposed technique requires six input parameters. If they are comprehended in a finite precision system, then the keyspace is $\sim$318 bits. According to the [17], the keyspace of the proposed technique is acceptable to resist brute-force attacks and satisfy cryptographic requirements.

5.4.2 Key and plaintext sensitivity

Along with a sufficient keyspace, the cryptographic technique should produce a random output from any input data to prevent an attacker from discovering the input data statistics and minimizing the relationship within cipher output [17, 30]. Moreover, the cipher output should be highly sensitive and dependent upon the key and plaintext such that if the even single bit is altered within the key or plaintext, the cryptographic technique should produce entirely different output every time [18]. The key sensitivity is the bit change percentage of the ciphertexts obtained after performing the data encryption process using slightly different keys. Experiments were performed by changing $x_0 = 0.30896$ to $x_0' = 0.30897$ to assess the key sensitivity, where ciphertext incurred from the corresponding Calgary Corpus files [27] were compared. Similarly, plaintext sensitivity of the proposed technique was assessed by randomly toggling a single bit in the plaintext while performing secure data compression with the same key, where ciphertext incurred from the corresponding Calgary Corpus files [27] were analyzed.

The bit change percentage for key sensitivity analysis of any cryptographic algorithm must possess a 50% change in bits to resist against cryptanalysis attacks [31]. Table 8 provides an experimental result for key and plaintext sensitivity trials of the proposed technique. Results show that the bit-change-percentage for both analyses was very close to the ideal value 50%. This confirms that the proposed technique is highly sensitive to input key and plaintext.

5.4.3 Randomness analysis of the proposed technique

The NIST SP800-22 [32] test suite is the most common tool for randomness analysis. This study applied the NIST suite to perform the randomness analysis of the proposed technique. This suite includes fifteen statistical tests, where each test gives p-values between 0 and 1. This p-value is further applied to evaluate the randomness of ciphertext by defining the significance level α, e.g., when$p < \alpha$, the ciphertext is non-random otherwise random. In this work, the significance levelα is set to 0.01 to confirm 99% confidence level for the randomness analysis of the proposed technique. Table 9 lists all computed p values for all tests with default input parameter settings defined in

Table 8 Key sensitivity analysis of ESDC

File	Key sensitivity	Plaintext sensitivity
bib	50.1441	49.9049
book1	50.0791	49.5351
book2	50.0867	49.9958
geo	50.1548	49.8277
news	50.0735	50.0418
obj1	50.0528	49.9784
obj2	50.0699	49.4375
paper1	50.0761	49.5703
paper2	50.1109	49.9181
paper3	50.0765	50.4847
paper4	50.2837	49.9058
paper5	50.1851	50.609
paper6	50.0557	49.9222
pic	50.1083	50.6542
progc	50.0753	49.7119
progl	50.0847	50.1187
progp	50.0971	49.894
trans	50.1198	50.0457

Table 9 NIST randomness test results

Statistical test	Proposed technique	
	p-value	Result
Frequency	0.2474	Success
Block frequency	0.6714	Success
Runs	0.0167	Success
Long runs of one's	0.7697	Success
Binary Matrix Rank	0.0270	Success
Spectral DFT	0.1587	Success
No overlapping templates	0.5340	Success
Overlapping templates	0.5335	Success
Universal	0.4365	Success
Linear complexity	0.2796	Success
Serial	0.9393	Success
Approximate entropy	0.7847	Success
Cumulative sums	0.2575	Success
Random excursions	0.7	Success
Random excursions variant	0.4425	Success

the NIST statistic test suite. Results clearly show that the proposed technique successfully passed all NIST tests, thus, proved secure with a 99% confidence level.

6 Conclusion

This work concentrates on the problems of data compression and encryption collectively without negatively affecting each other. An efficient, secure data compression technique was introduced, which provides cryptographic capabilities for use in combination with an adaptive Huffman coding, pseudorandom keystream generator, and S-Box. This work carries out three operations in adaptive Huffman coding, so that output is secured and compressed, simultaneously with key control. 1) The proposed work employed a chaotic Logistic map to introduce key control in the compression and decompression processes of the adaptive Huffman coding, named as secure adaptive Huffman coding. 2) The proposed work incorporates a chaotic S-Box to perform data substitution without compromising compression capabilities. 3) Furthermore, implementing a masking pseudorandom keystream based on chaotic Logistic map enhanced encryption quality. Thus, data compression is carried out according to a secret key such that the output will be both encrypted and compressed in a single step. Experimental results proved that the proposed technique achieved faster processing time compared to performing the encryption and compression techniques as separate steps. The proposed algorithm decisively achieved secure data compression; thus, proving useful for real-time implementation by reducing data space and transmission consumption. Security analysis also revealed that the proposed work is highly sensitive to both key and plaintext and that the generated ciphertexts successfully passed all NIST tests showing that the randomness of the ciphertext has 99% confidence level. Compression efficiency analysis demonstrates that the proposed technique produced similar space-saving (%) to standard techniques with slight variations while also providing adequate security.

Funding This publication was made possible by the NPRP award NPRP8-2158-1-423 from the Qatar National Research Fund (a member of The Qatar Foundation). The statements made herein are solely the responsibility of the authors. Open access funding provided by the Qatar National Library.

References

1. Bajaj S, Sion R (2014) IEEE Trans Knowl Data Eng 26(3):752. https://doi.org/10.1109/TKDE.2013.38. http://ieeexplore.ieee.org/document/6468039/
2. Gentry C (2010) Commun ACM 53(3):97. https://doi.org/10.1145/1666420.1666444. http://portal.acm.org/citation.cfm?doid=1666420.1666444
3. Xiaolin Y, Nanzhong C, Zhigang J, Xiaobo. C. (2010) In: 2010 Second International Workshop on Education Technology and Computer Science. IEEE, pp 329–332. https://doi.org/10.1109/ETCS.2010.460. http://ieeexplore.ieee.org/document/5458952/
4. Puangpronpitag S, Kasabai P, Pansa. D. (2012) In: 2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology. IEEE, pp 1–4. https://doi.org/10.1109/ECTICon.2012.6254320. http://ieeexplore.ieee.org/document/6254320/
5. Baptista MS (1998) Phys Lett A 240(1-2):50. https://doi.org/10.1016/S0375-9601(98)00086-3
6. Fridrich J (1998) Int J Bifurcat Chaos 8(06):1259. https://doi.org/10.1142/S021812749800098X
7. Starosolski R (2014) J Vis Commun Image Represent 25(5):1056. https://doi.org/10.1016/j.jvcir.2014.03.003. http://linkinghub.elsevier.com/retrieve/pii/S1047320314000595
8. Chen S. k. K. (2011) Comput Stand Interfaces 33(4):367. https://doi.org/10.1016/j.csi.2010.11.002. http://linkinghub.elsevier.com/retrieve/pii/S092054891100002X
9. Nagaraj N, Vaidya PG, Bhat KG (2009) Commun Nonlinear Sci Numer Simul 14(4):1013. https://doi.org/10.1145/1666420.cnsns.2007.12.001

10. Wong KW, Lin Q, Chen J (2010) IEEE Trans Circ Syst II: Express Briefs 57(2):146. https://doi.org/10.1109/TCSII.2010.2040315
11. Hermassi H, Rhouma R, Belghith S (2010) Commun Nonlinear Sci Numer Simul 15(10):2987. https://doi.org/10.1016/j.cnsns.2009.11.022
12. Zhu ZL, Tang Y, Liu Q, Zhang W, Yu H (2012) In: 2012 Fifth International Workshop on Chaos-fractals Theories and Applications. IEEE, pp 212–216.https://doi.org/10.1109/IWCFTA.2012.52. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6383208
13. El-said SA, Hussein KFA, Fouad MM (2011) Int J Signal Process Image Process Pattern Recogn 4(1):49
14. Usama M, Zakaria N (2017)
15. Wu CP, Kuo CCJ (2005) IEEE Trans Multimed 7(5):828. https://doi.org/10.1109/TMM.2005.854469
16. Zhou J, Au OC, Wong PHW (2009) IEEE Trans Signal Process 57(5):1825. https://doi.org/10.1109/TSP.2009.2013901
17. ALVAREZ G, LI S (2006) Int J Bifurcat Chaos 16(08):2129. https://doi.org/10.1142/S0218127406015970
18. Usama M, Khan MK, Alghathbar K, Lee C (2010) Comput Math Appl 60(2):326. https://doi.org/10.1016/j.camwa.2009.12.033. http://linkinghub.elsevier.com/retrieve/pii/S0898122110000064
19. Luca A, Ilyas A, Vlad A (2011). In: ISSCS 2011 - International Symposium on Signals, Circuits and Systems. IEEE, pp 1–4. https://doi.org/10.1109/ISSCS.2011.5978664. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5978664
20. Schuster HG, Just W (2006) Deterministic chaos: an introduction, 4th edn. Wiley, New York
21. Zhu H, Zhao C, Zhang X (2013) Signal Process Image Commun 28(6):670. https://doi.org/10.1016/j.image.2013.02.004
22. Kanso A, Smaoui N (2009) Chaos Solitons Fract 40(5):2557. https://doi.org/10.1016/j.chaos.2007.10.049. http://linkinghub.elsevier.com/retrieve/pii/S0960077907009320
23. Biham E, Shamir A (1991) J Cryptol 4(1):3. https://doi.org/10.1.1.31.2000
24. Belazi A, Rhouma R, Belghith S (2015) In: 2015 International Wireless Communications and Mobile Computing Conference (IWCMC). IEEE, pp 611–615. https://doi.org/10.1109/IWCMC.2015.7289153. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7289153
25. Usama M, Rehman O, Memon I, Rizvi S (2019) Int J Distrib Sens Netw 15(12):155014771989595. https://doi.org/10.1177/1550147719895957
26. Vitter JS (1987) J ACM 34(4):825. https://doi.org/10.1145/31846.42227. http://portal.acm.org/citation.cfm?doid=31846.42227
27. Witten J, Bell I, Cleary T Calgary Corpus (1990). http://www.data-compression.info/Corpora/CalgaryCorpus/
28. Zhan W, El-Maleh A (2012) Integr VLSI J 45(1):91. https://doi.org/10.1016/j.vlsi.2011.05.001. http://linkinghub.elsevier.com/retrieve/pii/S0167926011000514
29. Klein ST, Shapira D (2014) Discret Appl Math 163:326. https://doi.org/10.1016/j.dam.2013.08.022. http://linkinghub.elsevier.com/retrieve/pii/S0166218X13003636
30. Peng JPJ, Jin SJS, Chen GCG, Yang ZYZ, Liao XLX (2008) Fourth Int Conf Natur Comput 4:601. https://doi.org/10.1109/ICNC.2008.227
31. Mishra M, Mankar VH (2012) pp 169–179. https://doi.org/10.1007/978-3-642-30111-7_17
32. Rukhin A, Soto J, Nechvatal J, Miles S, Barker E, Leigh S, Levenson M, Vangel M, Banks D, Heckert A, Dray J, Vo S (2010) Natl Inst Stand Technol 800:131