ISSN: 0067-2904

Generating Streams of Random Key Based on Image Chaos and Genetic Algorithm

Fatima Faiz Saleh*, Nada Hussein M. Ali
Department of Computer Science, College of Science, University of Baghdad, Baghdad, Iraq

Received: 9/9/2021 Accepted: 13/2/2022 Published: 30/8/2022

Abstract
Today the Genetic Algorithm (GA) tops all the standard algorithms in solving complex nonlinear equations based on the laws of nature. However, permute convergence is considered one of the most significant drawbacks of GA, which is known as increasing the number of iterations needed to achieve a global optimum. To address this shortcoming, this paper proposes a new GA based on chaotic systems. In GA processes, we use the logistic map and the Linear Feedback Shift Register (LFSR) to generate chaotic values to use instead of each step requiring random values. The Chaos Genetic Algorithm (CGA) avoids local convergence more frequently than the traditional GA due to its diversity. The concept is using chaotic sequences with LFSR to generate seed values for genetic algorithms, which can generate keys with a high degree of randomness. The quality of key (generated sequence) was tested using known standard tests, then a comparison table is presented to show the increase in ratios in the test before and after applying GA, demonstrating that the proposed system generates sequence (key) with high randomness degree, The proposed system achieved an increase in the randomness rate by four degrees on average and thus it solves the problem of repetition and linearity, Finally, The system is built in the Java environment.
Keywords: Chaotic, Logistic Map, Linear Feedback Shift Register (LFSR), Genetic Algorithm, National Institute of Standards and Technology (NIST).

توليد تدفقات مفاتيح عشوائية بالاعتماد على الصورة الفوضوية والخوارزمية الجينية

فاطمة فائز صالح، ندى حسين محمد علي
قسم علوم الحاسبات ، كلية العلوم، جامعة بغداد، بغداد، العراق

الخلاصة
تتصدر الخوارزمية الجينية (GA) اليوم جميع الخوارزميات القياسية في حل المعادلات الغير خطية المعقدة بالاعتماد على قوانين الطبيعة. ومع ذلك ، يعتبر التقارب المتغير permute convergence أحد أهم عيوبها (GA)، والذي يُعرف بزيادة عدد التكرارات اللازمة لتحقيق المستوى الأمثل للحل. لمعالجة هذا القصور ، تقترح هذه الورقة طريقة جديدة للخوارزمية الجينية (GA) قائمة على الأنظمة الفوضوية. في هذا النظام الخوارزمية الجينية (GA)، تستخدم الخريطة اللوجيستية logistic map وسجل تحول التغذية المرتدة الخطي(LFSR) Linear Feedback Shift Register، لإنشاء قيم فوضوية لاستخدامها بدلاً من كل خطوة تتطلب قيمة عشوائية في الخوارزمية الجينية (GA). تتجنب خوارزمية الفوضى الجينية The(CGA) Chaos Genetic Algorithm التقارب المحلي بشكل متكرر أكثر من الخوارزمية الجينية التقليدي بسبب

*Email: fatima.saleh1201@sc.uobaghdad.edu.iq

تنوعها. يتم اختبار استخدام التسلسلات الفوضوية مع LFSR لإنشاء قيم أولية للخوارزميات الجينية (GA) ، والتي يمكن أن تولد مفاتيح بدرجة عالية من العشوائية ، ويتم اختبار جودة المفتاح التسلسل المتولد) باستخدام اختبارات قياسية معروفة وأضيف جدول مقارنة يظهر زيادة في النسب الاختبار قبل وبعد تطبيق الخوارزمية الجينية (GA)، حقق النظام المقترح زيادة في معدل العشوائية بمقدار أربع درجات كمتوسط مما يدل على أن النظام المقترح يولد تسلسلاً (مفتاحًا) بدرجة عالية من العشوائية وبهذا يحل مشكلة التكرار والخطية ، وأخيراً تم بناء النظام باستخدام لغة جافا JAVA .

1. Introduction

Due to the continuous exchange of data over the Internet in different formats, such as text, audio, image, and video, the need to protect this data from unauthorized access, illegal copying and modification, has increased. The need increased to change the original data into the unreadable format based on a key, this is known as "Cryptography". A cryptographic algorithm and a cryptographic key are the two main components of cryptography. Where the algorithm Represent a mathematical function, while the key represents the parameter used by this algorithm [1]. By increasing complexity in the key generation process, it becomes more challenging for the cryptanalyst to discover the key. Thus, it becomes nearly impossible to decrypt the cipher file and get the original file.

Many types of studies have focused on chaotic cryptography in recent decades, pointing out the existence of a robust relationship between cryptography and chaos, special chaotic images. An image is a two-dimensional array of integer numbers, the height and width of which are represented by rows and columns. Each element in this array represents a point of colour known as a pixel, which is used to specify the size of an image. Each pixel in the image is indexed by x and y coordinates and is stored as a 24-bit colour image or an 8-bit grayscale image. The 24-bit images are spread across three bytes, each of which is an RGB (red, green, blue) colour. The colours are created by combining these three RGB colours in various properties [2].

Besides, the huge amount of data, which is included in the image, and the strong and complex correlation between image pixels, led to the development of the traditional image encryption algorithms, such as Advanced Encryption Standard (AES), Data Encryption Standard (DES), International Data Encryption Algorithm (IDEA), etc.. Those algorithms, which have strong computational security and employ plaintext as either a data stream or a block cipher, are sometimes considered unsuitable for video or image encryption in real time. In addition, the main advantage of using chaos stems from the observation that a chaotic image appears to unauthorized users as noise. Second, some exciting properties, such as mixing and sensitivity to initial conditions, can be linked to good cipher properties, such as confusion and diffusion [3]. Furthermore, generating chaotic images is commonly low-cost with simple iterations, making it suitable for the construction of stream ciphers. In addition to increasing the complexity of key generation, and to get the infinity space of the one-time pad key we can add GA and LFSR [4].

This paper introduces a detailed description of generating random key streams based on image chaos and a genetic algorithm that can be used to generate a strong cryptographic key(s) with a high degree of randomness for symmetric cryptographic applications. It also presents the study of the effectiveness of this approach by applying several randomness testes. Moreover, comparisons have been made on the generated key. The results showed that this final generated key provides more randomness than the proposed key without using chaotic maps with genetic algorithm.

2. Literature review

This section, presents the research of some prominent authors in the same field and explains a short description of various randomness keys generation.

Shakir M. Hussain and Hussein Al-Bahadili, 2016: The paper proposed a method of symmetric ciphering applications that require a strong cryptographic key(s). Symmetric ciphering applications require a strong cryptographic key(s). The Key-Based Random Permutation (KBRP) method is fed with an initial private/secret key. It produces a permutation key of size n, which is half the size of the cryptographic key required and creates four vectors of size n representing the DNA bases of the private key (A, C, G, and T). The DNA vectors are mathematically processed using a linear method to generate the cryptographic key. Using the same permutation vector, the produced bases are re-permuted by using the same permutation vector. This procedure can be performed indefinitely, with the generated bases being re-permuted [1].

Ramen Pal and Somnath Mukhopadhyay, 2019: The paper proposed a method to find the best seed value for a chaotic map actual programmed genetic algorithm evolutionary algorithm Real Coded Genetic Algorithm (RCGA). The proposed system used a technique divided into three-steps. First step, generate and encode an initial population of size n. where n refers to the total number of chromosomes that is generated. The second step, the genetic algorithm was adopted in one of these chaotic maps to optimize the system parameters. The last step, based on these optimized system parameters used the respective chaotic map equation to generate a random pseudorandom bit series. To ensure a high degree of randomness of the generated bit sequence, the randomness test was done, using the most common test NIST statistical test suit for randomness [5].

Afiqah Zahirah Zakaria, et.al, 2019: GA begins by generating a population of chromosomes from random keys generated by a computer. The length of the key used then equals the number of genes. Because of the DES and AES techniques, the key lengths used in the proposed method are 48-bits and 128-bits, respectively. As a result, after generating the population, it goes through genetic processes known as crossover then mutation, which cause an increase in the total number of chromosomes. Finally, individuals based on their fitness are chosen randomly to participate in genetic operations [5].

Yasser, Iet.al.2020: The paper presents a data encryption method for both confusion and diffusion rounds by using novel perturbation. The chaotic structure was hybrid, where multiple maps are combined for media encryption. Blended chaotic maps are used to generate the control parameters for the permutation (shuffling) and diffusion (substitution) structures. The method used guarantees maintaining great encryption quality reproduced by chaotic, and key sensitivity, and low residual clarity[6].

3. The Linear Feedback Shift Register

The linear feedback shift register (LFSR) is a shift register that Depends on the previous state of its linear function as an input. The exclusive-or (XOR) is used for single bits as a linear feedback function, as shown in Figure 1. The main advantages of this type of pseudorandom are its long period, ease of design and implementation, and good statistical properties of binary sources [7].

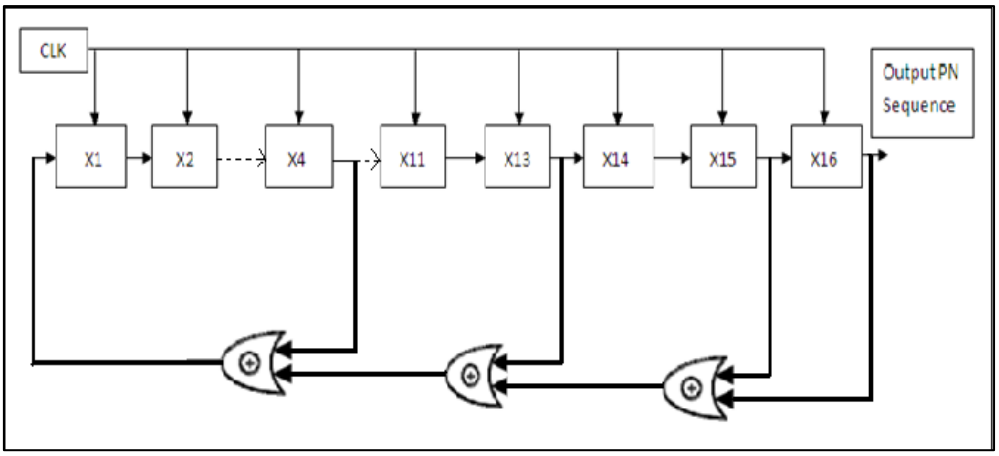

Figure 1-16-bit LFSR [8]

The LFSR is used in a variety of applications, including generating white noise, detection of errors and correcting the codes, manipulation algorithms, communication systems, compression algorithms, and cryptography systems. The LFSR's operation is completely deterministic; thus, its output stream is entirely dependent on the LFSR's seed, the seed is the initial state of LFSR. However, any register has a limited number of possible states, the LFSR should eventually repeat its output cycle. When the feedback function is a primitive polynomial, the maximum length of this repeated cycle is determined as in equation (2L– 1), where L is the length of the LFSR. In general, determining the primitive polynomials for L-bit LFSR is a difficult task [8].

4. Chaotic Map

Chaos is a type of restricted dynamic behaviour that occurs in nonlinear deterministic systems. In this paper, we generate the chaotic sequence using logistic and LFSR. Chaotic sequences have been shown to be simple and quick to generate and store, eliminating the need for long sequence storage. Furthermore, by changing the initial condition, an infinite number of different sequences can be generated. Moreover, these are deterministic and repeated sequences. It is extremely sensitive to changes especially in the initial condition, and even a minor change in the original initial condition can cause a significant change in system behaviour. Despite its appearance, it occurs under deterministic conditions in a deterministic nonlinear system.

One of the most important characteristics of chaotic systems is their sensitivity to the initial state even if the two best-fit options were discovered via a series of iterations techniques are relatively near [9]. In this study, the chaotic sequence was generated using logistic and LFSR. Chaotic sequences have been shown to be simple and quick to generate and store, eliminating the need for long sequence storage. Furthermore, by changing the initial condition, an infinite number of different sequences can be generated.

Logistic Map: is one of the most fundamental types of chaotic mappings. This map is simply a polynomial mapping, using the equation:

$$\mathbf{X_{n+1} = r\, X_n\, (1 - X_n)} \qquad X_n \in (0, 1) \qquad \mathbf{(1)}$$

Where, r refers to a control parameter, its range from (0 to 4). Clearly, $X \in (0,1)$ on conditions that the initial $X_0 \in (0,1)$ and $X0 \in \{0.0, 0.25, 0.75, 0.5, 1.0\}$.When $3.57 < r \leq 4$, the system has demonstrated a chaotic state. The value of r =3.99 was used for this study [9].

5. Genetic Algorithm

In Artificial Intelligence (AI), GA is a function that allows to stimulate the process of natural selection. It could be used in the protein kinase C (PKC) field in a variety of ways, such as to generate key(s), to improve the security of a standard encryption algorithm, or to create new symmetric/asymmetric algorithms. The use of GA is investigated to find the best and most

randomized key for a cryptographic algorithm. The cryptanalyst would have a difficult time deciphering the ciphertext due to the high level of difficulty involved in the key generation process. In essence, a genetic algorithm entails three operators: selection, crossover, and mutation, which are all applied to the generated population[10].

6. The Proposed Method Structure

The design of the proposed system is illustrated in Figure 2. It comprises of several stages including a **Logistic map**, **LFSR**, and **Genetic Algorithm**. The following subsections describe in detail the main phases of the proposed system construction, where n number of keys will be generated and stored in directory.

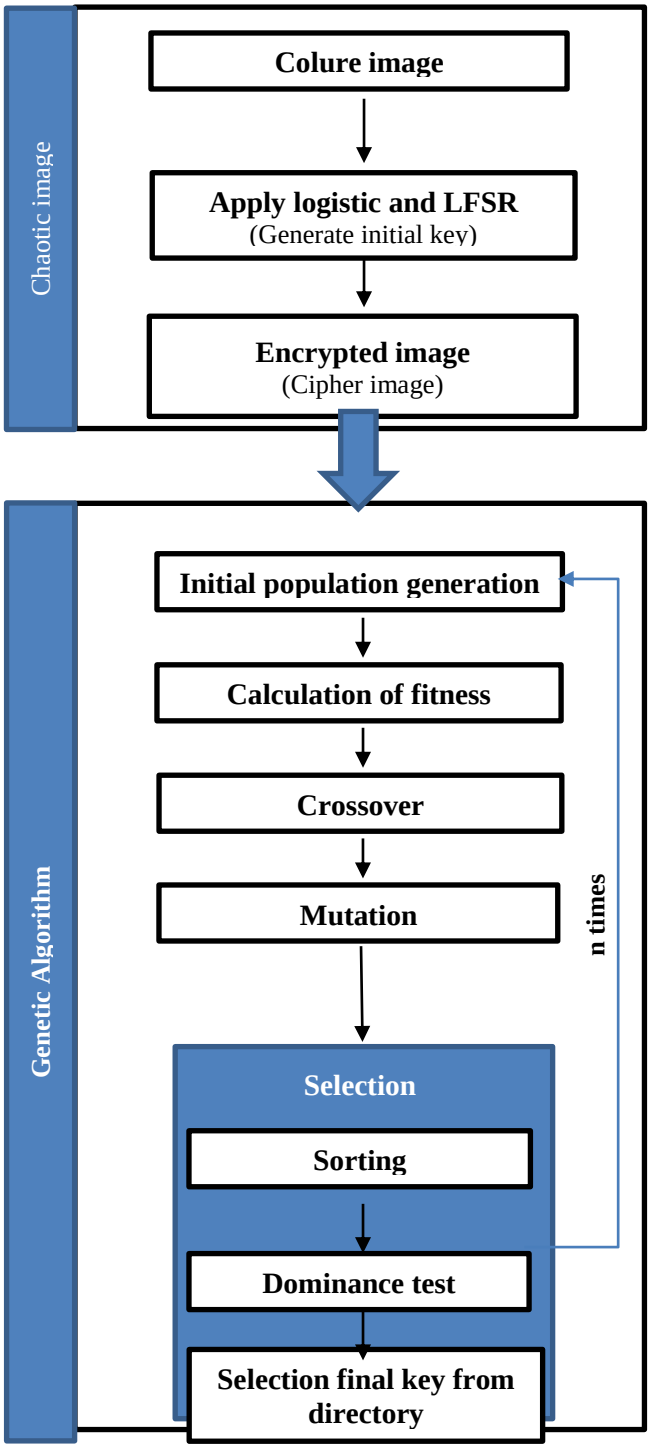

Figure 2- diagram of the proposed system

As demonstrated in Figure 3, the algorithm depends on two stages:

6.1 Initial key value generation stage:

The first stage is composed of three steps for generating the final stream (value of initial key), the last step represents the final stream obtained from this stage that is used in the second stage later. Figure 3 and Algorithm 1, Algorithm 2, and Algorithm 3 respectively demonstrate the necessary operation for key generation.

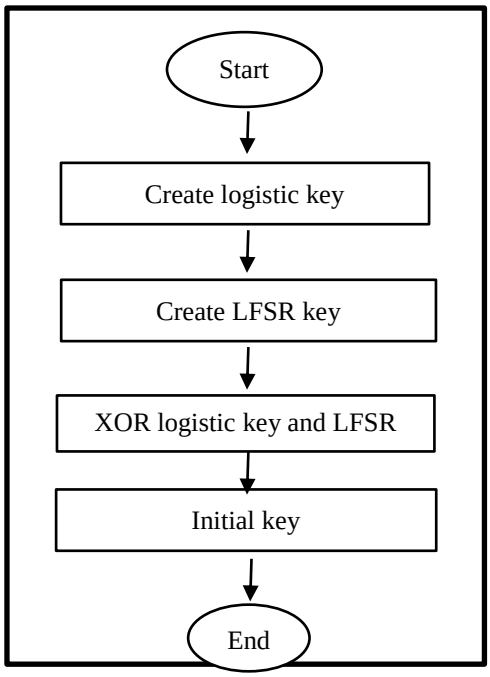

Figure 3-A flowchart for creating initial key in the proposed system

As shown in Figure 3, the generation of initial key depends on many Algorithms (1, 2, and 3):

- **Create logistic series as in Algorithm 1**

Algorithm (1): generate logistic sequence Input: r=3.99, x_0=0.1, n= 256*256 Output: logistic Key
Begin **Step 1:** using logistic equation eq. (1) **Step 2:** repeat step1 n times until became length of logistic KEY equal n **end**

- **Create LFSR Key as in Algorithm 2**

Algorithm (2): LFSR Key generation **Input:** seed 16-bits,taps bits[16,14,13,11],n length of required LFSR-KEY **Output:** LFSR-key
Start **Step 1:** take last bit from right side to be first bit output as LFSR-KEY. **Step 2:** shifting all bits in stream one step to RIGTH **Step 3:** doing XOR operation between taps bits as next-bit **Step 4:** INSERT next-bit in position 0 from stream **Step 5:** repeat steps 2,3,4,5 until became length of LFSR-KEY equal n **end**

Algorithm (3): initial key generation
Input: 16-bit LFSR-key, logistic Key
Output: initial key
Begin
Step 1: apply shift and exclusive or (XOR) between LFSR-key and Logistic Key key to result **Initial Key**
end

6.2 Encrypting image by logistic and LFSR (Initial Key):

The key obtained from previous stage will be used to encrypt the RGB (red, green, blue) image of size (256, 256) in the current stage. The resulting chaos image is used later in different methods, such as encryption algorithm, as in algorithm (4).

Algorithm (4): Encrypting image by **Initial Key**
Input: RGB image (height, width), initial key
Output: Cipher image
begin
Step 1: split colour image to three RGB channel
Step 2: binarized each RGB channels
Step 3: xor each channel from RGB channels with initial key
Step 4: decimalized each channel from RGB channel
Step 5: merge RGB channels to single colure encrypted image
end

6.3 Generating Final Randomness Key(s) by Applying Genetic Algorithm

At this stage, numerous random keys are generated that can be used to encrypt any type of plaintext file to preserve privacy.
Which are all used during population generation as clarify in the following:

❖ ***Population Generation*:** The proposed algorithm generates a random initial set of 100 chromosomes, each containing 256 genes, by pulling decimal values from the set of encoded images. Then convert each value to binary ignoring the zeros from the left, repeat this process until we have 256 bits per 100 chromosomes.

❖ ***Crossover:*** The second step is to create a new child out of two randomly selected parents, by pulling from encrypted image, Crossover Rate range is selected at 0.6 as optimal value from crossover range [0,1][11] according to the following equation:

$$\mathbf{numberofcrossover} = \frac{\mathbf{Crossover\ Rate} \times \mathbf{Key\ length} \times \mathbf{Number\ of\ population}}{\mathbf{100}} \quad \mathbf{(2)}$$

Which gives the number of crossovers as 153. For each iteration, a different crossover point is selected by pulling random value from encrypted image to perform a single point crossover. Now the total population size becomes 253.

❖ ***Mutation:*** The next step is to perform mutation where a random chromosome is chosen from the existing population and a mutation point is selected also based on the encoded image by pulling random value to determine chromosome and mutation point and invert that bit. The selected Mutation Rate is 0.001 as optimal value from mutation range [0,1][11] , according to the following equation

$$\mathbf{Number\ of\ Mutation} = \frac{\mathbf{Mutation\ Rate} \times \mathbf{Key\ length} \times \mathbf{Number\ of\ population}}{\mathbf{100}} \quad \mathbf{(3)}$$

Which gives the number of mutations to be performed as 2 chromosomes.

❖ ***Fitness Function Calculation***

Many operations are executed in this function

- ***Conversion:*** The binary valued chromosomes from the population of size 253 were converted into the decimal number format.
- ***Fitness Value:*** The Fitness values calculate number of ones in key.
- ***Ordering:*** The chromosomes are then ordered in decreasing order based on their fitness values.
- ***Dominance Testing:*** The topmost key from the sorted population is chosen, i.e. the fittest value, and paired with the remaining chromosomes one at a time. The XOR function is applied to all these pairs, and the hamming distance of each pair is calculated. The pair with the greatest hamming distance is then chosen, and one of the chromosomes is chosen at random from that final pair. This key is then saved in the repository, and the entire process is repeated 15 times (Depending on the number of keys we want to generate), beginning with the first step.
- ***Final Key Selection from Repository:*** The Dominance Testing is again applied to all the 100 keys generated and then the final key is selected for further data encryption and decryption.

Algorithm (3): Generate Final Random Key
Input: encrypted image
Output: Random Key

Begin
Step 1: pull values from encrypted image and convert each value to binary continue until get 256-bit
Step 2: repeat step 1 n times, n number of chromosomes in population
Step 3: calculate crossover number as equation(2)
Step 4: pull random value from encrypted image to determined index of twice chromosomes
Step 5: pull random value from encrypted image to determined position of crossover operation
Step 6: create new population first part from first chromosome and other from second chromosome
Step 7: calculate mutation number as equation(3)
Step 8: pull random value from encrypted image to determined index of one chromosomes
Step 9: pull random value from encrypted image to determined position of mutation operation
Step 10: edit same chromosome by revers single bit in mutation position between 0 or 1.
Step 11: for each chromosome calculate fitness value as equation (3).
Step 12: search about chromosome with maximum fitness.
Step 13: XOR the top chromosome with remainder chromosomes.
Step 14: calculate number of 1's in each result from each xoring operation.
Step 15: select the pairs with maximum no. of 1's as candidates.
Step 16: random selection between candidate's pair to determined final randomness key.
Step 17: The same steps from [1-17] is iterated **n** times, where **n**: number of keys that we want to create.
End

Figure 4 represent visually the histogram analyses for cipher image. The most important thing to remember here is that the distribution of the histogram diagram for the cipher image must appear uniform and hide the redundancy of the plain image as possible as obviously the difference in Figure 4.

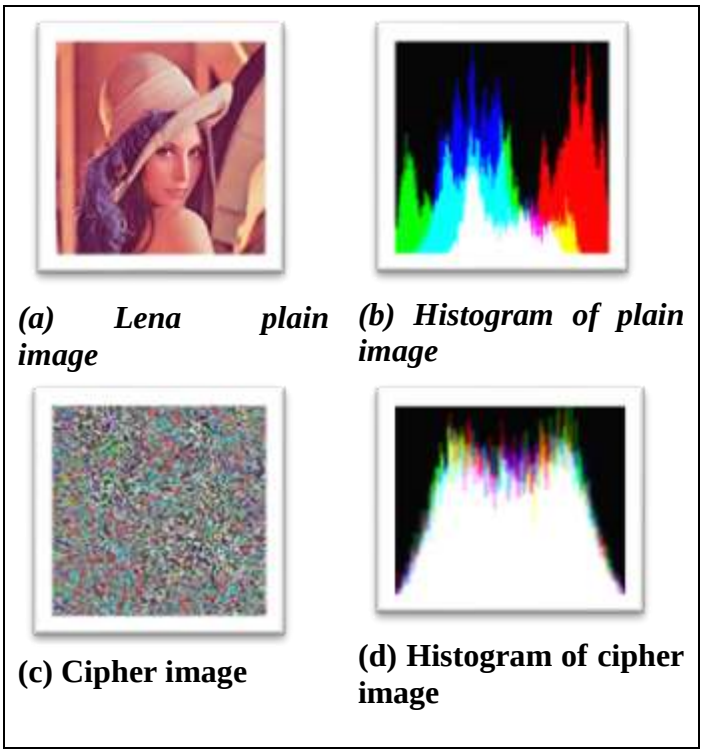

Figure 4- Histogram for both plain and cipher Images.

7. Results

The implementation of the proposed method was accomplished using the Java platform. Random samples were created by encryption of colure image by a logistic map with r of 3.99, x_0 of 0.1 and 16 bit LFSR. After obtaining a cipher image, and applying the GA, a random population of 100 chromosomes was generated to produce random samples. Various tests were performed on the samples, and the results were positive.

After generating 100 chromosomes, the crossover function was used to increase the population size to 253 chromosomes, with a crossover rate of 0.5. The mutation rate here was set to 0.009. The number of ones in the generated key is used to calculate and analyse the fitness values of the keys. Finally, the effectiveness of the proposed system can be seen in table 4 by comparing the results of the NIST test before and after applying the GA.to validate of randomness. The proposed work has been validated using the statistical test suite from the National Institute of Standards and Technology (NIST). Five tests out of 15 different tests were performed to determine the randomness for created keys of size 256 bits. The reason for choosing those five tests is that these tests work correctly for a minimum length of the bit sequence.

Table 1-Statistical test results for encryption image (Before apply GA)

Test	P values	States
Frequency	0.169131	Success
Block Frequency	0.385534	Success
Runs Test	0.093373	Success
Cumulative Sums Forward	0.338189	Success
Cumulative Sums Backward	0.267215	Success

Table 2- Statistical test results for encryption image (After apply GA**)**

Test	P values	States
Frequency	0.707660	Success
Block Frequency	0.495036	Success
Runs Test	0.102115	Success
Cumulative Sums Forward	0.629223	Success
Cumulative Sums Backward	0.945889	Success

By comparing the results of each test in Table 1 and Table 2, the success of the five tests in the second table is noticeable, and the results increased by an average of 4 degrees, which proves the effectiveness of the proposed system.

8. Conclusion

In this paper, a new approach was proposed to generate the initial key using the concept of chaotic image and LFSR, then GA was subsequently used to generate the final keys. The originality of this method of this work has a flexible space due to: First, the addition of 16-bit LFSR. Second, generated chaotic values rather than random values from using a logistic map. Third the complexity of the work of the proposed system increased by using a genetic algorithm, where a 16-bit LFSR stream was combined with a logistic map stream to create a chaotic image, which was then utilized to pluck random values to run the GA and obtain the random keys. Finally, randomness was tested by using NIST test groups where it was observed that P-value was larger than or equal to 0.01 in all experiments.

In addition, the comparison between the NIST test results showed that this final generated key provides more randomness degree than the proposed key without using chaotic maps with genetic algorithm. Which determines that the new approach can generate random keys that are difficult to predict. Hence, it can be concluded that the new proposed system is strong against unauthorized attacks and is an improved part of a strong encryption system in the future.

References

[1] S. M. Hussain and H. Al-bahadili, "A DNA-Based Cryptographic Key Generation Algorithm," *Int'l Conf. Secur. Manag.* |, no. July, pp. 338–342, 2016.

[2] N. H. M. Ali and S. A. Abead, "Modified Blowfish Algorithm for Image Encryption using Multi Keys based on five Sboxes," *Iraqi J. Sci.*, vol. 57, no. 4, pp. 2968–2978, 2016.

[3] H. S. Kwok and W. K. S. Tang, "A fast image encryption system based on chaotic maps with finite precision representation," *Chaos, Solitons & Fractals*, vol. 32, no. 4, pp. 1518–1529, May 2007, doi: 10.1016/J.CHAOS.2005.11.090.

[4] K. Faraoun, "Chaos-based key stream generator based on multiple maps combinations and its application to images encryption," *Int. Arab J. Inf. Technol.*, vol. 7, no. 3, pp. 231–240, 2010.

[5] A. Z. Zakaria, S. N. Ramli, C. C. Wen, C. F. M. Foozy, P. Siva Shamala Palaniappan, and N. F. Othman, "Enhancing the randomness of symmetric key using genetic algorithm," *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 8, pp. 327–330, 2019.

[6] I. Yasser, M. A. Mohamed, A. S. Samra, and F. Khalifa, "A chaotic-based encryption/decryption framework for secure multimedia communications," *Entropy*, vol. 22, no. 11, pp. 1–23, 2020, doi: 10.3390/e22111253.

[7] S. Falih, "A Pseudorandom Binary Generator Based on Chaotic Linear Feedback Shift Register," *Iraqi J. Electr. Electron. Eng.*, vol. 12, no. 2, pp. 155–160, 2016, doi: 10.37917/ijeee.12.2.5.

[8] R. V Kshirsagar, "Design of 8 and 16 Bit LFSR with Maximum Length Feedback Polynomial & Its pipelined Structure Using Verilog HDL," pp. 3337–3339, 2014.

[9] M. Javidi and R. Hosseinpourfard, "Chaos genetic algorithm instead genetic algorithm," *Int. Arab J. Inf. Technol.*, vol. 12, no. 2, pp. 163–168, 2015.

[10] S. Jawaid and A. Jamal, "Generating the Best Fit Key in Cryptography using Genetic Algorithm," *Int. J. Comput. Appl.*, vol. 98, no. 20, pp. 33–39, 2014, doi: 10.5120/17301-7767.

[11] C. Johansson and G. Evertsson, "Optimizing genetic algorithms for time critical problems," *Engineering*, no. June, 2003, [Online]. Available: http://denver.bth.se/fou/cuppsats.nsf /all/ 7f65a64 6dddb44a 7c1256d44003e9326/$file/Optimizing Genetic Algorithms for time critical problems.pdf.