International Journal of Mathematical, Engineering and Management Sciences
Vol. 8, No. 2, 230-244, 2023
https://doi.org/10.33889/IJMEMS.2023.8.2.014

A Chaotic System and Count Tracking Mechanism-based Dynamic S-Box and Secret Key Generation

Sagnik Pal
School of Computer Science and Engineering,
Vellore Institute of Technology, Vellore, Tamil Nadu, India.
E-mail: palsagnik3@gmail.com

Ramani Selvanambi
School of Computer Science and Engineering,
Vellore Institute of Technology, Vellore, Tamil Nadu, India.
E-mail: ramani.s@vit.ac.in

Preeti Malik
Department of Computer Science and Engineering,
Graphic Era Deemed to be University, Dehradun 248002, Uttarakhand, India.
E-mail: preetishivach2009@gmail.com

Marimuthu Karuppiah
Department of Computer Science and Engineering and Information Science,
Presidency University, Bengaluru 560064, Karnataka, India.
Corresponding author: marimuthume@gmail.com

(Received on May 31, 2022; Accepted on December 24, 2022)

Abstract
In cryptography, Block ciphers use S-Boxes to perform substitution and permutation operations on a data block. S-Boxes provide non-linearity and confusion of bits to the cryptographic algorithms. In addition, secret keys are critical security aspects for encrypting and decrypting messages. The uncertainty and randomness of the secret key and S-boxes used in the algorithm determine the extent of security against any cryptanalysis attack. This paper proposes a new mechanism to dynamically generate a secret key and S-Box each time while sending and receiving the message. These dynamically generated S-Boxes and keys depend on mutually decided security parameters and message transfer history. Furthermore, a new counter-based mechanism is introduced in this paper. These enhancement techniques are applied to the serpent cipher algorithm, and a data transfer simulation is performed to validate the efficacy of the proposed method. We observe that the dynamically generated S-box follows the strict avalanche criteria. We further validate that the encrypted message shows higher sensitivity to the S-box and the secret key in enhanced serpent cipher than the original version. However, to validate our proposed method, we test and analyze the improvements in the Serpent Cipher Algorithm.

Keywords- Serpent block cipher, S-Box and permutation box, Chaotic equation, Cyclic generator, Count tracking mechanism.

1. Introduction

Symmetric key block cipher algorithms are used to encrypt the messages block-wise before sending them for transmission. These cryptographic block ciphers are popularly used to encrypt a piece of plaintext and send it as a cipher text to the receiver. The receiver then decrypts the cipher text using the decryption algorithm. For this whole process to work, both the sender and the receiver must be aware of the secret key used by the encryption and decryption algorithm. These symmetric block cipher algorithms have been consistently used along with asymmetric cipher algorithms to encrypt and decrypt messages for transmission. The symmetric block cipher algorithms are faster than asymmetric ones, but the asymmetric

key ciphers are more secure as the encrypted message is protected against unauthorized access. The cryptographic block cipher algorithm often involves too many rounds that increase execution time. Reducing the number of rounds would increase the execution speed and take less time for encryption and decryption, compromising security (Dawood and Hammadi, 2017). Hence a combination of these two techniques is used to encrypt a piece of information, especially when the message sent over a network is bigger in size. The proposed work will add an extra layer of uncertainty to symmetric key block ciphers and can be customized for different algorithms to fit security requirements.

The work discussed in this paper aims to enhance the security of symmetric key block ciphers so they can be used more efficiently. The idea is to provide the sender and receiver a mechanism to generate a new key and S-Box (Substitution box) each time while sending and receiving the message. A chaotic equation will be employed each time to generate the S-box. The sender and receiver will not only generate a new S-box and a new key for each communication but also make sure these generated S-boxes and keys are different for each communication. The most significant parameter for an intruder to intercept and decrypt the message used to be the key, but now, in addition to the key, there are added parameters and functional dependencies for the intruder to decipher the message within a specified time frame. The proposed ideology ensures that even if the interception is successfully done once and the dependencies are compromised, the same dependencies cannot be used to decrypt the message during the following interceptions. Furthermore, one of the generated parameters for the S-box also employs the usage of positional bits of the previously sent message. All these security parameters aim to decrease the probability for attackers to decipher messages and restrict the time frame they have for decryption. This methodology will allow most symmetric block cipher algorithms to perform faster than they do by reducing the number of rounds from their native version. Combining this work with the Serpent Cipher algorithm and reducing the number of rounds in the encryption and decryption steps, the overall execution time decreases.

In the following sections, we discuss the background work from recent research on dynamic S-box generation. First, the literature review (section 2) highlights algorithms and methodologies using pseudorandom number generators and chaotic equations. Then we explain the proposed algorithm for S-box and secret key generation in the methodology section (section 3). We start by explaining the random choice equation and the hyperparameters used in the chaotic map, followed by the dynamic S-box generation using an example. The count tracking mechanism and message history usage are explained in the later section. Further, a security analysis of the S-box and key is done in the results and discussions section (section 4). Finally, our approach's end outcome (conclusion) and limitations are presented in section 5.

2. Literature Review

The S-Boxes in this paper are designed to be generated dynamically. Hence, we have referred to the work by Manjula and Mohan (2016) to construct a key-dependent dynamic S-Box for AES. The authors have given experimental results and have used S-box rotation and subkey generation before each round. Further, a chaotic algorithm has been established by Gao et al. (2006), which uses 'Henon Chaotic' maps to meet the requirements of image transfer safety. They used the distribution of grey-level values as they are entirely random and show random behavior on encryption. Their proposal consists of 2 steps, Image fusion between the original and key image; then, they encrypt using the Henon Chaotic map. Yun-Peng et al. (2009) further take up the topic of digital image encryption and chaotic algorithms and applies them to DES. A new encryption scheme using the Logistic chaos sequencer is used to make the pseudorandom sequence. It shows high speed, sensitivity, and security and can be used in the actual image encryption. The paper gives a clear comparison between the AES and 3-DES with their algorithms. Elkamchouchi et al. (2018) describe the modified Serpent cipher algorithm. This paper proposes the serpent Cipher modification for image

encryption based on chaotic mapping and cyclic group substitution instead of byte substitution transformation (DES S-Box), decreasing the number of rounds and the time usage. The authors have done a statistical test on images of different sizes and compared the other algorithms.

The graphical programming tool for the encryption and decryption of the Serpent Algorithm for the Secured data Transmission is used by Kabilan et al. (2017). It is fast and easy. Nazlee et al. (2009) have shown parallelism using CUDA and how it can be used to implement the limitation of Serpent Cipher, which is speed. They proposed a technique that transforms the CPU-based implementation of the Serpent Cipher algorithm to CUDA implementation. The study of S-BOX properties in block cipher by Mohamed et al. (2014) is critical. This paper analyses the properties of S-BOX so that a new model in the future can be proposed, which will strengthen it further. The authors proposed a novel approach for designing S-BOX in AES using the chaotic map. Najafi et al. (2004) present a novel, generic, parameterizable Rijndael encryptor core capable of supporting varying key sizes. The 192-bit key and 256-bit key designs run at data rates of 5.8 Gbits/sec and 5.1 Gbits/sec, respectively. Another critical aspect of the serpent cipher is that it can be implemented on a 24-bit DSP processor and how efficient it can be with performance. Ivancic et al. (2001) dealt with less powerful Motorola's 56000 families. They have discussed various models of S-BOXES and compared the results and how they are better without the need for ROM data. The algorithm can be scaled down to 8-bit and 16-bit for high 32-bit microprocessors. Authors have provided various implementations of Serpent Cipher. Biham et al. (1998) propose the Serpent Cipher algorithm in this paper. They provided a highly conservative and efficient implementation that uses DES S-boxes and allows a more rapid avalanche, a more efficient bit-slice implementation, and a security analysis. They claim this to be as secure as triple DES. The authors, Dawood and Hammadi (2017), discuss the study's weaknesses, and the primary target of the paper is to highlight vulnerabilities and explain the gaps in design elements that can be exploited in the AES structure. One of the disadvantages they mentioned was that it could not work correctly with 64 bits systems. Also, they mentioned that decryption is slower than encryption, especially in embedded devices, and this feature refers to the unbalancing structure of AES. Biham et al. (2001) presented the attacks on 7-round, 8-round, and 10-round variants of Serpent. They have attacked with varying key lengths on 7, 8, and 10 rounds with 256 bits keys. The attack is being enhanced with the amplified boomerang attack and uses better differentials. They have described a rectangle attack, how it affects the various cipher rounds, and how to deal with that.

Ibrahim (2021) analyzed various cryptographic properties of constructed S-boxes by pseudorandom bijective algorithms. The Fisher-Yates algorithm for shuffling the bits is also analyzed. The authors study the randomness and cryptographic strength of dynamic S-boxes. AL-Zubaidy and Al-Bahrani (2018) proposed a new dynamic S-box generation algorithm using a Chebyshev chaotic map and an exponential map. A new key generation algorithm has been proposed. The authors show the usage of chaotic maps for S-box generation. The generated keys are shown to have high randomness. Manjula and Mohan (2018) explored a hash function to generate a dynamic S-Box. This paper also analyses the structure of AES algorithm. The experimental results of the modified AES algorithm with generated S-Boxes were shown. The time-based analysis compared encryption and decryption time for standard and dynamic S-box. A new S-Box generation technique for the AES algorithm was proposed in this paper. Zagi and Maolood (2020) presented a modified Serpent Cipher Algorithm. The original algorithm has 32 rounds, making it slower than Rijndael. This paper has improved the speed and uses a 4x4 S-Box constructed through the multiplicative group of finite commutative chain rings. The conclusion has shown a result that clearly shows the algorithm's improvement over the standard Serpent Algorithm. Fisher-Yates shuffle has been proposed as an S-box generation technique to enforce the non-linearity in a block cipher by Tayel et al. (2018). The S-boxes generated using this technique are entirely dynamic, and the generation is done under the complete control of the secret key. Rohiem et al. (2005) presented a couple of chaotic system-based pseudorandom

bit generators (CCS-PRBG), and the output was verified using the NIST tests for randomness. This algorithm has been used to encrypt and decrypt the image. It has been tested whether it satisfies the property of confusion and diffusion by analysis of histograms and correlation of pixels of a plain and ciphered image. Al-Khasawneh et al. (2018) proposed a multi-chaos algorithm for image encryption. It can solve common issues associated with the algorithm of a low-dimensional map. Pixel and bit-level permutation has been defined in this paper. As a novel solution to the earlier existing problems of image encryption, this paper proposes a chaos-based symmetric method of a key cryptosystem. This method employs an external secret key that Logistic, Henon, and Gauss iterated maps have previously expanded. Tsedura and Chibaya (2020) introduce a new approach to generate S-boxes during runtime and make computational changes to the DES algorithm. In this paper, the newly modified DES performs better than the original DES algorithm. Zahid et al. (2021) propose an enhanced method to generate secret key-dependent S-boxes using linear trigonometric transformations. This paper also introduces optimization for the characteristics of initial S-boxes generated using the proposed scheme.

3. Methodology

The S-box and the secret key are generated each time before encrypting the plaintext at the sender node; the plaintext is then encrypted utilizing the newly generated S-box and the secret key, then it is sent to the receiver node. After the receiver node receives the cipher text, it generates the same S-box and secret key that was generated by the sender before encrypting the plaintext and uses these (S-box and secret key) to decrypt the cipher text. The main principle behind the dynamic generation of S-boxes for each round is to use a random choice equation; the procedure is explained in detail in the later sections. The chaotic equation uses some hyperparameters (λ, x_0). This paper discusses how the hyperparameters are updated before and after each digital message transfer.

3.1 Random Choice Equation

Random choice chaotic equations have cryptographic properties that maintain high randomness and diffusion of bits using this Eqn. 1. The Eqn. 1 is a modified version of the logistic map polynomial.

$$x(n) = floor\left[\left[\lambda \cdot x(n-1).\left(1 - x(n-1)\right)\right] mod(p)\right] + 1. \quad (1)$$

λ is the security parameter. For values of λ between 0 and 4, $x(n)$ can take values between **1** and **p** (value of **p** is selected in later sections) inclusive. For the value of λ between 3.9 and 4, the output of the Eqn. 1 shows chaotic behavior (value of $x(n)$ starts to oscillate). For $\lambda > 4$, the value of $x(n)$ diverges and leaves the interval of [**1**, **p**]. Hence in our proposed methodology, we consider the value of λ between 3.9 and 4 (i.e., $3.9 \leq \lambda \leq 4$). The random choice equation can thus produce a random value between **1** to **p**.

Before we understand how the S-boxes are generated for each round, we must understand primitive roots and generator elements.

3.1.1. Primitive Roots

We define ***z*** as the primitive root modulo ***n*** *(where **n** is a positive integer), if for* every integer ***a*** (where ***a*** is coprime to ***n)***, there exists an integer ***x*** such that $z^x \equiv a \pmod{n}$. Here, ***x*** is the index or discrete logarithm of the integer ***a*** to the base ***z*** **modulo** ***n***. Here, ***z*** can be a primitive root modulo ***n*** only when ***z*** is a generator element of a set of integers coprime to **n**.

E.g., 2 is a primitive root modulo 5, as for every integer **a** (where **a** is coprime to 5), there exists an integer ***x*** such that $2^x \equiv a \pmod{5}$.

3.1.2 Generator Elements

If ***n*** is a prime number, the expression z^x ***mod (n)*** can generate all numbers between **1** and **n-1** (as all numbers from **1** to **n-1** are coprime to **n**), then **z** can be used to generate the complete set **S** = {1,2, 3, … n-1}. Here, **z** is called the generating element of set **S**.
E.g., Consider n = 7, then 3^x mod (7) can generate S = {1, 2, 3, 4, 5, 6}.

In Table 1, where n=7 and z=3, it is observed that the values start repeating after the exponent of 3 is more than 6. Hence, the period of repetition is 6. If the number ***n*** is a prime, the repetition period is always ***n-1***. From Figure 1, we see that the S-box for each round in the Serpent cipher must contain 16 entries, which must be in a random arrangement from 1 to 16. If we divide the plaintext of 128 bits into 8-bit divisions, then the plaintext block will have 16 segments. The S-box is used to shuffle these segments randomly. Figure 1 shows how an S-Box shuffles the given input segment by rearranging the blocks randomly to add randomness and confusion during encryption.

Table 1. Repetition period.

3^1	3^1 mod (7)	3
3^2	3^2 mod (7)	2
3^3	3^3 mod (7)	6
3^4	3^4 mod (7)	4
3^5	3^5 mod (7)	5
3^6	3^6 mod (7)	1
3^7	3^7 mod (7)	3

3.2 Dynamic S-Box Generation

Now, there is a need to generate a random arrangement of numbers from 1 to 16. Consider P = 17, every number in the set S = {3, 5, 6, 7, 10, 11, 12, 14} is a generator element of 17. For example, if z = 3. We have, e = 3^xmod (17). The next step is to choose the number of rounds used in the algorithm. Here 10 rounds are finalized. So, we have to generate 10 random numbers using the chaotic equation. The numbers generated will be scaled down between 0 to 7 (as there are 8 generators of 17). The generated integer will be used as an index to choose a generator from the set S as shown in Table 2.

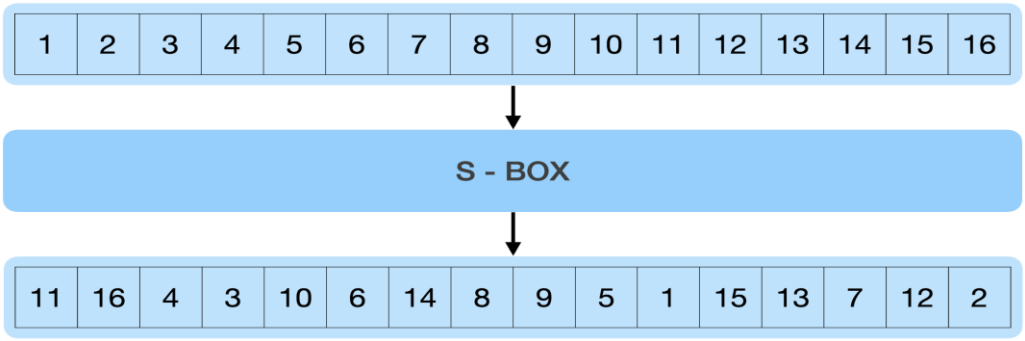

Figure 1. S-Box transformation.

Table 2. The generator set.

Index	0	1	2	3	4	5	6	7
S	3	5	6	7	10	11	12	14

The **modified chaotic equation** used for generating 10 numbers is given below and fixing the value of x(0) = x_0 and a value of λ, we could compute x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9), x(10) using the equation 2.

$$x(n) = \left(floor\left[[\lambda \cdot x(n-1).x(n-1).x(0)]\ mod(128)\right] + 7\right) mod(8) \quad (2)$$

3.2.1 S-Box Generation Example

Initialize $\lambda = 3.9863$, $x(0) = 5$. The 10 numbers generated using the modified chaotic equation are {1, 2, 6, 4, 5, 1, 2, 6, 4, 5}. Considering these generated numbers as indices we can form a new set of generators {S[1], S[2], S[6], S[4], S[5], S[1], S[2], S[6], S[4], S[5]} which is G = {5, 6, 12, 10, 11, 5, 6, 12, 10, 11}. S-box for ith round is generated using the Pseudocode.

for i from 1 to 10:
for x from 1 to 16:
S[i][x] = G[i]^x mod (17).

In Table 3, S-Box for each round [1, 10] is generated (e.g., S1 = S[1][x]) using the algorithm mentioned in the above pseudocode. In the later sections, we discuss how the S-Boxes generated for every communication will be different by updating the value of $\lambda\ and\ x(0)$ before each new digital communication.

Table 3. S Box values of different rounds.

Round																
S1	5	8	6	13	14	2	10	16	12	9	11	4	3	15	7	1
S2	6	2	12	4	7	8	14	16	11	15	5	13	10	9	3	1
S3	12	8	11	13	3	2	7	16	5	9	6	4	14	15	10	1
S4	10	15	14	4	6	9	5	16	7	2	3	13	11	8	12	1
S5	11	2	5	4	10	8	3	16	6	15	12	13	7	9	14	1
S6	5	8	6	13	14	2	10	16	12	9	11	4	3	15	7	1
S7	6	2	12	4	7	8	14	16	11	15	5	13	10	9	3	1
S8	12	8	11	13	3	2	7	16	5	9	6	4	14	15	10	1
S9	10	15	14	4	6	9	5	16	7	2	3	13	11	8	12	1
S10	11	2	5	4	10	8	3	16	6	15	12	13	7	9	14	1

3.3 Count Tracking Mechanism

This paper proposes a novel mechanism of count tracking to ensure that before each message communication between sender and receiver, the security parameters $(\lambda, x(0), key)$ are updated using the current count value. The 'count' is defined as the total number of successful digital communications that have already been completed between a sender-receiver pair. The value of 'count' is updated by incrementing it by '1' after every digital communication.

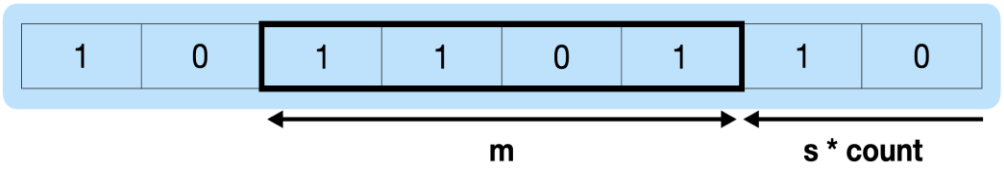

Figure 2. Updating the count value.

Figure 2 shows how the count variable increments after each successful message communication. A hash table data structure implementation with count entries (as values) of different sender-receiver pairs (as hash table keys) can be maintained for tracking different communications.

3.4 Message History Usage

The latest message transferred between a sender-receiver pair is also considered for our proposed approach. In this section, we see how the value of security parameter **m** is calculated using the previously exchanged message between two nodes in the network. The value of **m** depends again on three values (s, latest plaintext, count, and window size). The **s** is defined as the shift rate, which can take any value between 1 and the plaintext block size. The block size of plaintext is 128 bits in the case of the serpent cipher, but for understanding, we consider this example in Figure 3.

Figure 3. Message history value assignment.

In the Figure 3, block size = 8 bits, window size = 4 bits, s = 1 bits, count = 2, plaintext =”10110110”. **m** value is calculated as 13 (base 10 value of “1101”). The value of m is also calculated after the receiver node receives the cipher text from the sender. While testing our mechanism on the serpent cipher, we use (block size = 128 bits, s = 4 bits, window size = 16 bits, and the count value is calculated from the count tracking mechanism). The default value of **m** is set as 0. Figure 4 shows the calculation of the **m** value at the sender and receiver sides. The above figure window size is taken as 4, the plaintext size is taken as 8 bits, and the shift rate is 1. The window's position is determined by the instantaneous value of shift rate (**s**) and **count**. The value of **m** is stored in the lookup table every time before the message is sent. The lookup table can be implemented as a hash table with the sender-receiver pair acting as the key and **m** as their corresponding

value. Please note that the same plaintext is used in Figure 4 for every message transmission, but the same method is applicable even if the plaintext sent in each message transmission is different.

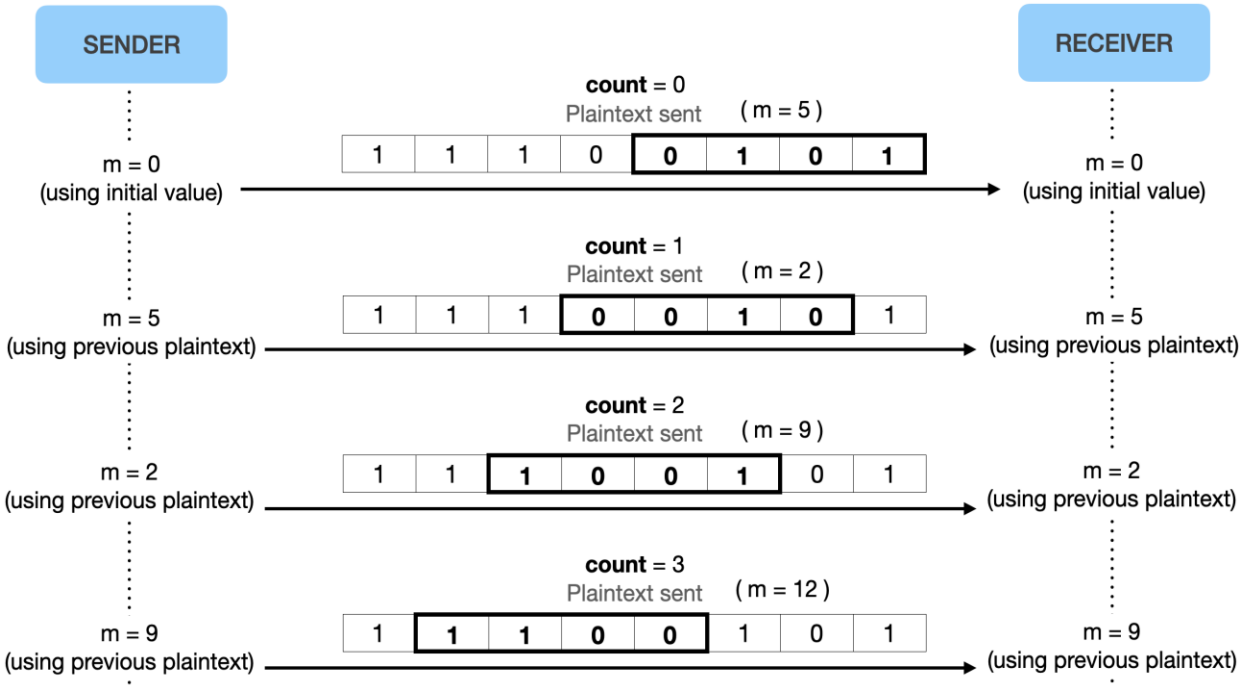

Figure 4. 'm' calculation demonstration.

3.5 Updating Security Parameters

The idea behind using the count tracking mechanism and latest message history is to ensure that each time before a digital message transfer, the security parameters $(\lambda, x(0))$ are updated, which in turn will ensure that S-box and key generated will be unique for every message transfer between a pair. This is mainly done to prevent an attacker from tracking down the security parameters and hence deciphering the ciphertext. For updating the values of $(\lambda, x(0))$, we use user-defined functions, which will be only known by the sender-receiver pair and will be stored as their corresponding lookup entries (the lookup entries can be stored in a hash table similar to **m** values with sender-receiver pair acting as key).

3.5.1 Example of Security Parameter Update

The function used to compute $\lambda \, and \, x(0)$ can vary from simple linear functions to higher order polynomials. Consider one such example below. The function to update $\lambda \, and \, x(0)$ is given by Eqn. 3 and Eqn. 4.

$$\lambda_{new} = \lambda_{old} + \big((0.00001 * count) * m\big) + \big((0.00001 * count) * m\big)^2 + \big((0.00001 * count) * m\big)^3 \quad (3)$$

$$x(0)_{new} = x(0)_{old} + (count + m) + (count + m)^2 + (count + m)^3 \quad (4)$$

Also note that since λ can only take values between 3.9 and 4, if the value exceeds 4, we reinitialize λ to 3.9. The authority to decide what functions to use entirely lies with the sender and receiver. They must mutually agree upon using these functions for performing updates. After $\lambda\ and\ x(0)$ values are updated, they can be used again by the chaotic equation to calculate the new S-boxes for each round.

Let us consider an example with random values for security parameters (m = 13, count = 2, λ_{old} = 3.91452 and $x(0)_{old}$ = 5). Now using Eqn. 3 and Eqn. 4, the new values generated for λ_{new} will be 3.91478 (rounded to 5 decimal places) and $x(0)_{new}$ will be 3620. Now using the modified chaotic equation (Eqn. 2), a new dynamic S-Box can be generated.

3.6 Updating the Key

For updating the value of key each time before a digital communication at the sender side and after digital communication at the receiver side, the **count** and **m** values are used, as shown in Figure 4. A function is again decided upon by both the communicating parties for updating the key before and after message transfer (see Eqn. 5).

$$Key_{new} = (Key_{old} + (count + m) + (count + m)^2) + (count + m)^3)\%keySize \quad (5)$$

While choosing a function it is recommended to pick a continuous function that has a large range and is non-oscillating. A good example is a cubic polynomial. As we increase the degree of the function or replace it with a higher-order function, it will be difficult for the attacker to guess the function. In addition to that, it must be noted that if the key size is small, the range of values generated will be confined because of the modulo operator (modulo key Size), and randomness will be restricted. Both the communicating parties must know the initial secret key value and the key updating function to transfer messages.

3.7 Sender-Receiver Message Transfer

It is time to summarize all the procedures discussed in the above sections and define a pipeline of steps to be followed at the sender and receiver side for transferring a message successfully using the proposed modifications. The complete process pipeline is described below in Figure 5. The sequence of steps must be followed by every sender-receiver pair in order to communicate with each other. All the security parameters ($\lambda, key, m, count, x_0$) must be maintained in a lookup table (a hash table implementation with sender-receiver entries as key), which will contain these parameters for every sender-receiver node.

To optimize the space requirements, we can specify a storage limit on the capacity of the lookup table and store only the parameters which communicate most frequently. After the security parameters of a pair are dropped from the lookup table, the parameters are reinitialized and stored in the lookup table whenever there is a need to re-communicate.

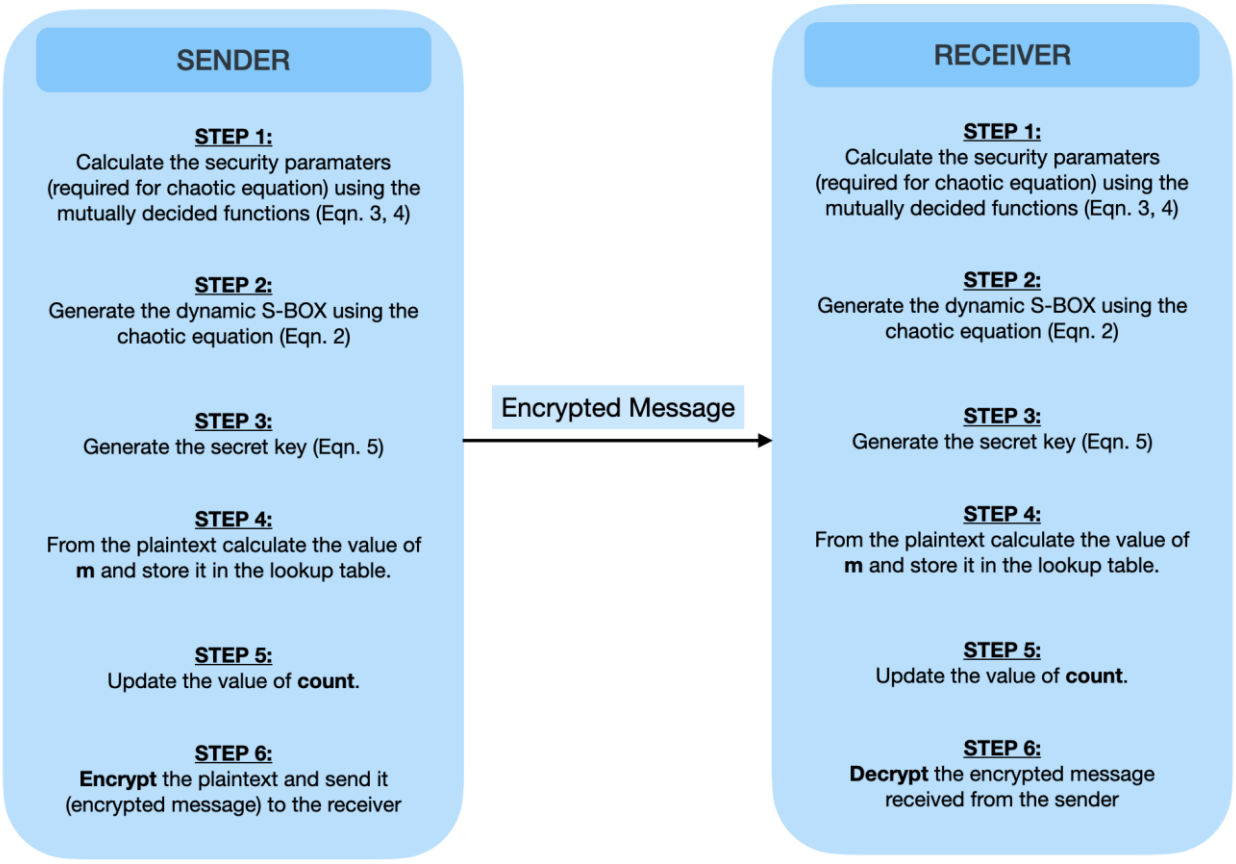

Figure 5. Sender-Receiver Message transfer diagram.

4. Results and Discussion

Based on the methodology proposed in the above section, the dynamic S-box and key generation mechanism are applied to the classical serpent cipher algorithm to observe the effects on its speed and robustness. The implementation includes count variable incrementation and message history usage to update **count** and **m** values. The effect of newly added security parameters is also tabulated and discussed. In this section, an execution speed followed by an S-box and secret key security analysis is done after applying the proposed algorithm to the serpent cipher. Further, the security of the S-Box is validated using Strict Avalanche Criteria.

4.1 Execution Speed Analysis

For quantitative analysis of execution speed, a system with Intel(R) Core (TM) i7-7700HQ CPU and 2.80GHz processing speed, 8.00 GB Installed memory (RAM), Windows 10 (64-bit operating system) is used. A mini–Local Area Network consisting of two systems (nodes) behaving as sender and receiver is created to simulate data transfer. The input data is an image in .jpeg format, and the image size is 180 x 166. The image is first converted into a binary stream encoding (base-64 encoding) and then encrypted to be sent at the sender node and decrypted after receiving it at the receiver node. For execution speed analysis, the first classical Serpent cipher algorithm is used to encrypt the data and decrypt the cipher text at the

receiver side Biham et al (1998), the work proposed in this paper is used to develop an enhanced version of the serpent cipher, the modified serpent cipher. Then the Modified Serpent Cipher algorithm is again used to encrypt and decrypt the data at the appropriate communicating nodes. The improvement in execution speed of the algorithm is compared in Table 4 and Table 5, given below.

Table 4. Execution speed of the encryption process with seconds.

No of Rounds	Classical Serpent	No of Rounds	Modified Serpent
32	55.68941	10	15.90857

Table 5. Execution speed of the decryption process with seconds.

No of Rounds	Classical Serpent	No of Rounds	Modified Serpent
32	55.43567	10	16.92635

4.2 Security Analysis

As discussed in the above sections, the security of a cipher algorithm is dependent on the number of trials needed to determine the key and the S-Box. The difficulty of breaching the cipher increases exponentially with key size (2^n, where n is the key size in bits). In order to ensure that the cryptanalysis attacker cannot determine the correct key used within the theoretical limits of a computer's processing speed, the key sizes were traditionally kept large. However, suppose a restriction is brought regarding the time available for performing any cryptanalysis attack, the key size can be reduced safely without compromising the security of the encryption-decryption algorithm. The security parameters depend on the sender-receiver pair's instantaneous count value. As a result of the message history and count tracking methodology, the security parameters are randomized before each message transaction. Hence, the count tracking mechanism reduces the time for the attackers to determine the security parameters, as mentioned in Table 6.

Table 6. Count tracking performance.

	Key Size (bits)	Number of trials to decipher key	Time available with attacker
Performance without count tracking	n	2^n	No time restriction
Performance after applying count tracking	n	2^n	c

where, **t** = It is the total time between 2 consecutive message transfers between sender-receiver. The robustness of the proposed mechanism can only be ensured if the security parameters used for the dynamic generation of the S-box and the key are practically infeasible. The number of trials needed to accurately predict the security parameters ($\lambda, key, m, count, x_0$) is mentioned in the table below. Before the inclusion of these security parameters, the number of trials needed to guess the key accurately was $\mathbf{2^n}$ (where **n** is the key size in bits). After the security parameters are added to modify the algorithms, the total number of trials needed is $\mathbf{2^{n1}x\ 2^{n2}x\ 2^{n3}x\ 2^{n4}x\ 2^{n5}}$ (refer to n1, n2, n3, n4, n5 from the table). Table 7 shows the maximum number of trials required to decipher each security parameter using brute force attack and the dependence of some parameters on other security parameters (**count**, **m**, and **s**).

Table 7. Security parameters.

Security Parameter	Parameter Size (in bits)	Trials needed to decipher parameter	Dependency on other parameters
λ	n1	2^{n1}	count, m
x_0	n2	2^{n2}	count, m
key	n3	2^{n3}	count, m
m	n4	2^{n4}	s, count
count	n5	2^{n5}	-
s	n6	2^{n6}	-

where, **count** = count value maintained by the sender and receiver
m = value of last b bits of the previous message transferred
d = any value less than the actual size of the message
s = it is the shift rate of the sliding window which is used to calculate the value of **m, d** is the size of the window.

It is crucial for the security of the block cipher that the cipher text generated by the algorithm is sensitive to the key and S-box used. Even if a single bit of the encryption key is changed, the cipher text generated must be completely different. The S-box must possess the same property. Therefore, even if two S-boxes differ by a single entry, they must produce completely different cipher texts. To quantitatively measure the extent of sensitivity of the encrypted message on the key and S-box, we find the correlation between the original message and the decrypted message with altered S-box and key (differing by a single bit position), respectively. Table 8 shows the correlation values when the key is altered by complementing a single bit only. All the keys and S-boxes listed in Tables 8 and 9 differ by a single bit from the original key and S-box, respectively. If the correlation values are close to zero, it shows high sensitivity to the key and S-box. In Table 8, the row corresponding to "Serpent" contains the correlation values between the original message (before encryption using the secret key) and the decrypted message (using the 1-bit complemented secret key). The row corresponding to "Enhanced Serpent" contains the correlation values between the original message (before encryption using the dynamically generated secret key) and the decrypted message (using the 1-bit complemented dynamic secret key). Clearly, the modified serpent cipher has more sensitivity to the secret key than the original version (Table 8).

Table 8. Correlation values for key sensitivity analysis.

Block Cipher	Key_1	Key_2	Key_3	Key_4
Serpent	0.0112	0.0126	0.0212	0.0231
Modified Serpent	0.0027	0.0052	0.0020	0.0024

Similarly in Table 9, the S-box's sensitivity is quantified and represented using correlation values. Again, the Modified Serpent cipher has better correlation values (closer to zero) than the original version. The serpent cipher's modified version, aided by our dynamic s-boxes, has successfully passed the key and S-box sensitivity test.

Table 9. Correlation values for S-Box sensitivity analysis.

Block Cipher	$S\text{-}Box_1$	$S\text{-}Box_2$	$S\text{-}Box_3$	$S\text{-}Box_4$
Serpent	0.0223	0.0233	0.0338	0.0213
Modified Serpent	0.0121	0.0048	0.0102	0.0045

4.3 Strict Avalanche Criteria

According to the strict avalanche criteria, with a change in a single input bit, every output bit must change by a probability of 0.5. Hence if one of the input bits is changed, more than half of the output bits must change in the encrypted text. In Table 10, a tabulation of the change in the number of output bits to input bits is done. The block size of plaintext is 128 bits. From Table 10, it is visible that our generated S-box follows the Strict Avalanche Criteria.

The modification proposed in this paper is not just limited to increasing the attacker's difficulty in interpreting the cipher text. The mechanism also ensures that even if the security parameters are compromised once and have been spotted correctly, they still cannot be used to decipher the cipher text in succeeding communications, as the value of the parameters updates after each transmission. Hence the complete process of deciphering has to be started again from scratch.

Table 10. Strict avalanche criteria validation.

Bit Position Complemented	Number of Output bits changed
0	87
1	84
4	90
16	89

The accurate values of the security parameters can only be determined if the functions (given in Eqn. 3 and Eqn. 4) used to update the values of these parameters are known to the attacker. The difficulty of determining these functions can be varied by increasing the degree of these functions or using strictly increasing functions.

5. Conclusions

With the growing need and importance of information security in the present setting, this paper has proposed some innovative modifications to existing techniques and algorithms. A novel approach to generating S-boxes and a secret key using message history and a counter-based mechanism was proposed in this paper. The dependence of S-box generation on the previous messages encrypted by the block cipher adds a new security element. Compared to traditional brute force attacks, the proposed work has significantly reduced the probability of deciphering a cipher text by an unauthorized intruder. Security is added for symmetric key block ciphers by adding more security hyperparameters and functional dependencies. The execution speed of symmetric key block cipher algorithms is increased by reducing the number of rounds without compromising security. Freedom of customizability of the cryptographic algorithms is provided for the organizations without compromising accepted standards. This modification can be masked with any symmetric key block cipher, which involves the use of an S-Box and a secret key; hence its scope is not limited to any specific algorithm. It must be noted that the given methodology can be partially applied to asymmetric block cipher algorithms. The counter-tracking technique and message history usage can be masked with asymmetric block ciphers. If we observe, the count tracking mechanism introduced in this paper is based on the logical synchronization of clocks at the sender and receiver sides. Making the shared security variables between sender and receiver dependent on the logical clock decreases the chances of message interception. However, it must be noted that if all the security parameters, functions used for key generation, and the encryption-decryption algorithm are compromised along with the logical count of message transfers, the encrypted message can be intercepted by an intruder. Also, if the functions defined for key generation in this methodology are not formulated wisely by the sender and receiver party

(e.g., if the function chosen is linear or of degree 1), an intruder may be able to guess the function being used. Hence, selecting a function that produces a unique key for a range of continuous values is essential.

Conflict of Interest

The authors confirm that there is no conflict of interest to declare for this publication.

Acknowledgments

This research did not receive any specific grant from public, commercial, or not-for-profit funding agencies. The authors would like to thank the editor and anonymous reviewers for their comments that helped improve the quality of this work.

References

Al-Khasawneh, M.A., Shamsuddin, S.M., Hasan, S., & Bakar, A.A. (2018, July). An improved chaotic image encryption algorithm. In *2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE)* (pp. 1-8). IEEE. Shah Alam, Malaysia.

AL-Zubaidy, R.N., & Al-Bahrani, E. (2018). New key generation algorithm based on dynamical chaotic substitution box. *In 2018 Al-Mansour International Conference on New Trends in Computing, Communication, and Information Technology (NTCCIT)* (pp. 93-98). IEEE. Baghdad, Iraq.

Biham, E., Anderson, R., & Knudsen, L. (1998). Serpent: A new block cipher proposal. *In International Workshop on Fast Software Encryption* (pp. 222-238). Springer, Berlin, Heidelberg.

Biham, E., Dunkelman, O., & Keller, N. (2001). The rectangle attack—rectangling the serpent. *In International Conference on the Theory and Applications of Cryptographic Techniques* (pp. 340-357). Springer, Berlin, Heidelberg.

Dawood, O.A., & Hammadi, O.I. (2017). An analytical study for some drawbacks and weakness points of the AES cipher (rijndael algorithm). *Qalaai Zanist Journal*, *2*(2), 111-118.

Elkamchouchi, H.M., Takieldeen, A.E., & Shawky, M.A. (2018). A modified serpent based algorithm for image encryption. *In 2018 35th National Radio Science Conference (NRSC)* (pp. 239-248). IEEE. Cairo, Egypt.

Gao, H., Zhang, Y., Liang, S., & Li, D. (2006). A new chaotic algorithm for image encryption. *Chaos, Solitons & Fractals*, *29*(2), 393-399.

Ibrahim, S. (2021). Performance analysis of dynamic bijective S-BOX construction algorithms. *In 2021 National Computing Colleges Conference (NCCC)* (pp. 1-4). IEEE. Taif, Saudi Arabia.

Ivancic, D., Runje, D., & Kovac, M. (2001, June). Implementation of serpent encryption algorithm on 24-bit DSP processor. *In ISPA 2001. Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis. In conjunction with 23rd International Conference on Information Technology Interfaces* (pp. 411-416). IEEE. Pula, Croatia.

Kabilan, K., Saketh, M., & Nagarajan, K.K. (2017). Implementation of SERPENT cryptographic algorithm for secured data transmission. *In 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)* (pp. 1-6). IEEE. Coimbatore, India.

Manjula, G., & Mohan, H.S. (2016). Constructing key dependent dynamic S-Box for AES block cipher system. *In 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)* (pp. 613-617). IEEE. Bangalore, India.

Manjula, G., & Mohan, H.S. (2018). Improved dynamic S-box generation using hash function for AES and its performance analysis. *In 2018 Second International Conference on Green Computing and Internet of Things (ICGCIoT)* (pp. 109-115). IEEE. Bangalore, India.

Mohamed, K., Pauzi, M.N.M., Ali, F.H.H.M., Ariffin, S., & Zulkipli, N.H.N. (2014). Study of S-box properties in block cipher. *In 2014 International Conference on Computer, Communications, and Control Technology (I4CT)* (pp. 362-366). IEEE. Langkawi, Malaysia.

Najafi, B., Sadeghian, B., Zamani, M.S., & Valizadeh, A. (2004). High speed implementation of serpent algorithm. *In Proceedings. The 16th International Conference on Microelectronics* (pp. 718-721). IEEE. Tunis, Tunisia.

Nazlee, A.M., Hussin, F.A., & Ali, N.B.Z. (2009). Serpent encryption algorithm implementation on compute unified device architecture (cuda). *In 2009 IEEE Student Conference on Research and Development (SCOReD)* (pp. 164-167). IEEE. Serdang, Malaysia.

Rohiem, A.E., Elagooz, S., & Dahshan, H. (2005). A novel approach for designing the s-box of advanced encryption standard algorithm (AES) using chaotic map. *In Proceedings of the Twenty-Second National Radio Science Conference, 2005. NRSC 2005* (pp. 455-464). IEEE. Cairo, Egypt.

Tayel, M., Dawood, G., & Shawky, H. (2018). Block cipher S-box modification based on fisher-yates shuffle and ikeda map. *In 2018 IEEE 18th International Conference on Communication Technology (ICCT)* (pp. 59-64). IEEE. Chongqing, China.

Tsedura, N.A., & Chibaya, C. (2020, November). Effects of runtime generated S-boxes to the DES Model. In *2020 2nd International Multidisciplinary Information Technology and Engineering Conference (IMITEC)* (pp. 1-6). IEEE. Kimberley, South Africa.

Yun-Peng, Z., Wei, L., Shui-Ping, C., Zheng-Jun, Z., Xuan, N., & Wei-di, D. (2009). Digital image encryption algorithm based on chaos and improved DES. *In 2009 IEEE International Conference on Systems, Man and Cybernetics* (pp. 474-479). IEEE. San Antonio, TX, USA.

Zagi, H.R., & Maolood, A.T. (2020). A novel serpent algorithm improvement by the key schedule increase security. *Tikrit Journal of Pure Science*, *25*(6), 114-125.

Zahid, A.H., Ahmad, M., Alkhayyat, A., Hassan, M.T., Manzoor, A., & Farhan, A.K. (2021). Efficient dynamic S-box generation using linear trigonometric transformation for security applications. *IEEE Access*, *9*, 98460-98475.