symmetry

MDPI

Article

Symmetry Analysis in Construction Two Dynamic Lightweight S-Boxes Based on the 2D Tinkerbell Map and the 2D Duffing Map

Ala'a Talib Khudhair *, Abeer Tariq Maolood and Ekhlas Khalaf Gbashi

Computer Science Department, University of Technology, Baghdad 10066, Iraq;
abeer.t.maolood@uotechnology.edu.iq (A.T.M.); ekhlas.k.gbashi@uotechnology.edu.iq (E.K.G.)
* Correspondence: cs.22.20@grad.uotechnology.edu.iq

Abstract: The lack of an S-Box in some lightweight cryptography algorithms, like Speck and Tiny Encryption Algorithm, or the presence of a fixed S-Box in others, like Advanced Encryption Standard, makes them more vulnerable to attacks. This proposal presents a novel approach to creating two dynamic 8-bit S-Boxes (16 × 16). The generation process for each S-Box consists of two phases. Initially, the number initialization phase involves generating sequence numbers 1, sequence numbers 2, and shift values for S-Box1 using the 2D Tinkerbell map. Additionally, sequence numbers 3, sequence numbers 4, and shift values for S-Box2 are generated using the 2D Duffing map. Subsequently, the S-Box construction phase involves the construction of S-Box1 and S-Box2. The effectiveness of the newly proposed S-Boxes was evaluated based on various criteria, including the bijective property, balance, fixed points, and strict avalanche criteria. It was observed that S-Box1 achieved a remarkable linear and differential branch number of 4, surpassing any previous studies. Furthermore, it exhibited a non-linearity of 105.50, a differential uniformity of 12, and an algebraic degree of 7. Similarly, S-Box2 also achieved a linear and differential branch number of 4, a non-linearity of 105.25, a differential uniformity of 14, and an algebraic degree of 7. Moreover, the reduction in the number of linear and nonlinear operations for both S-Boxes makes them suitable for lightweight algorithms. The architecture of the proposed S-Boxes demonstrates robustness, with a total of 3.35×10^{504} possible S-Boxes, providing protection against algebraic attacks.

Keywords: lightweight algorithms; two dynamic S-Boxes; 2D Tinkerbell map; 2D Duffing map

Citation: Khudhair, A.T.; Maolood, A.T.; Gbashi, E.K. Symmetry Analysis in Construction Two Dynamic Lightweight S-Boxes Based on the 2D Tinkerbell Map and the 2D Duffing Map. *Symmetry* **2024**, *16*, 872. https://doi.org/10.3390/sym16070872

Academic Editor: Jie Yang

Received: 13 June 2024
Revised: 2 July 2024
Accepted: 3 July 2024
Published: 9 July 2024

1. Introduction

Lightweight cryptographic algorithms, like lightweight block ciphers designed for use in limited resource environments such as radio-frequency identification (RFID), sensor networks, and healthcare systems, rely heavily on the substitution box (S-Box) [1]. The S-Box, as the only nonlinear component in modern block ciphers, plays a key role in creating a complex relationship between plaintext and ciphertext. This role is also known as confusion [2]. This relationship has a significant impact on both hardware consumption and the critical path delay of a block cipher, making an efficiently structured S-Box crucial for optimal implementation performance [3].

The security of a block cipher depends on the confusion in the ciphertext caused by an S-Box. Therefore, researchers are developing new S-Box designs and assessing their strength using common benchmarks [4]. The advanced encryption standard (AES) utilizes a highly secure 8-bit S-Box based on perfect nonlinear transformation, but it is static and requires a minimum of 35 nonlinear operations for implementation [5]. In 2021, Kim, H. et al. proposed various methods for constructing S-Boxes from smaller S-Boxes, establishing criteria for ensuring a minimum linear and differential branch number of 3. However, some of these S-Boxes exhibit high fixed points and high differential uniformity and still demand a high number of linear operations for implementation, all while remaining static [6]. Block ciphers such as Fantomas [7], Robin [7], Scream v3 [8], FLY [9], and PIPO [10] employ 8-bit

S-Boxes composed of three smaller S-Boxes but also demonstrate a static nature requiring numerous linear operations for implementation, with a linear and differential branch number of 2. M. Sajjad et al. [11] proposed designing a pair of nonlinear components of a block cipher over quaternion integers. Jassim, S. A. and Farhan, A. K. [12] proposed designing a novel efficient substitution box by using a flower pollination algorithm and chaos system. Sajjad, M. et al. [13] proposed designing a pair of nonlinear components of a block cipher over gaussian integers. Zahid, A. H. and Arshad, M. J. [14] proposed an innovative design of substitution boxes using cubic polynomial mapping. Sajjad, M. et al. [15] proposed a novel approach for constructing substitution boxes (S-boxes) over Gaussian integers, which are complex numbers with integer coefficients.

This paper introduces a new method for constructing two dynamic, lightweight 8-bit S-Boxes (16 × 16) in HEX format, based on the 2D Tinkerbell map and the 2D Duffing map. The process of generating each S-Box has two phases. First is the initialization phase. This involves generating sequence numbers 1, sequence numbers 2, and shift values (buffer 1 and buffer 2) for S-Box1 using the 2D Tinkerbell map and generating sequence numbers 3, sequence numbers 4, and shift values (buffer 3 and buffer 4) for S-Box2 using the 2D Duffing map. Second is the S-Box construction phase. This involves constructing S-Boxes' values depending on the results from the number initialization phase. This methodology enables both linear and differential branch numbers of at least four, and this enhances security. In addition, high non-linearity, low fixed points, a high algebraic degree, high strict avalanche criteria, low differential uniformity, and reducing the number of linear and nonlinear operations required to implement the S-Boxes.

The paper is organized as follows: Section 2 introduces an overview; Section 3 presents the proposed method of constructing a new S-Box; Section 4 assesses the proposed S-Boxes and provides a comparison of our proposed S-Box and existing S-Boxes; and Section 5 presents the conclusions and directions for future work. Appendix A shows a complete example.

2. Overview of Chaotic Systems (the 2D Tinkerbell Map and the 2D Duffing Map) and Deoxyribose Nucleic Acid (DNA)

2.1. Chaotic Systems

A mathematical behavior that is both nonlinear and deterministic is the foundation of the chaos theory [16]. It has a higher sensitivity to any change in initial conditions, including the control parameters and initial values. Consequently, a slight alteration to the beginning values or control settings causes a large alteration in the chaotic outputs [17]. The properties of a good cipher in cryptography, such as confusion and diffusion, are linked to those of chaotic systems, aiding researchers in improving the security of cryptographic systems [18]. In this study, two chaotic maps are used: the 2D Tinkerbell map and the 2D Duffing map.

✓ The 2D Tinkerbell map. Equation (1) shows the formal definition of the 2D Tinkerbell map iterator [19]:

$$\begin{cases} x_{n+1} = x_n^2 - y_n^2 + a \times x_n + b \times y_n \\ y_{n+1} = 2 \times x_n \times y_n + c \times x_n + d \times y_n \end{cases} \quad (1)$$

There are four parameters: a, b, c, and d. The range of parameters is unbounded. The typical values of a, b, c, and d are: a = 0.9, b = 0.6013, c = 2, and d = 0.5, or a = 0.3, b = 0.6, c = 2, and d = 0.27. The range of the initial values is between x_0 and y_0 $[-1, 1]$.

✓ The 2D Duffing map. Equation (2) shows the formal definition of the 2D Duffing map iterator [20]:

$$\begin{cases} x_{n+1} = y_n \\ y_{n+1} = -b \times x_n + a \times y_n - y_n^3 \end{cases} \quad (2)$$

There are two parameters: a and b. The range of parameters is unbounded. The range of the initial values is between x_0 and y_0 $[-1, 1]$.

2.2. Deoxyribose Nucleic Acid (DNA)

The DNA sequence consists of four main nucleic acid cores: A (adenine), C (cytosine), G (guanine), and T (thymine). A and T correspond to each other, as do G and C [21]. When applying encoded function on the combinations 00, 01, 10, and 11 using the four cores A, C, G, and T, there are 24 different coding manners [22]. However, only eight coding manners adhere to the Watson–Crick complement law, which can be seen in Table 1.

Table 1. Eight map rules.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	A	A	C	C	G	G	T	T
1	C	G	A	T	A	T	C	G
2	G	C	T	A	T	A	G	C
3	T	T	G	G	C	C	A	A

The binary string is transformed into a DNA sequence by taking the intersection of the row and column for every 2 bits from Table 1 as shown in Algorithm 1.

Algorithm 1 Present DNA codes

Input: String.
Output: DNA codes.
begin

1. Read the string.
2. Convert each character of the string into ASCII code.
3. Convert the ASCII code of each character of the string into an 8-bit binary.
4. Compare every 2 bits with Table 1 by taking the intersection of the bits value with its location.
5. Represent the DNA codes.

End

For example, if the user inputs the string "book", here are the detailed steps of how to convert the string into DNA codes.

String	book															
Binary string	01100010011011110110111101101011															
Split into 2 bits	01	10	00	10	01	10	11	11	01	10	11	11	01	10	10	11
Value of each 2 bits	1	2	0	2	1	2	3	3	1	2	3	3	1	2	2	3
Location	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
DNA codes	C	C	C	A	A	A	A	A	C	C	G	G	A	A	G	A

3. The Proposed Method

This paper presents a novel approach to constructing two dynamic S-Boxes that are responsible for creating confusion in block ciphers. The generation process of the two S-Boxes goes through two stages.

A. **The initialization phase** includes generating sequence numbers 1, sequence numbers 2, and shift values (buffer 1, buffer 2) based on the 2D Tinkerbell map for S-Box1 by using Equation (1), as shown in Figure 1 and Algorithm 2.

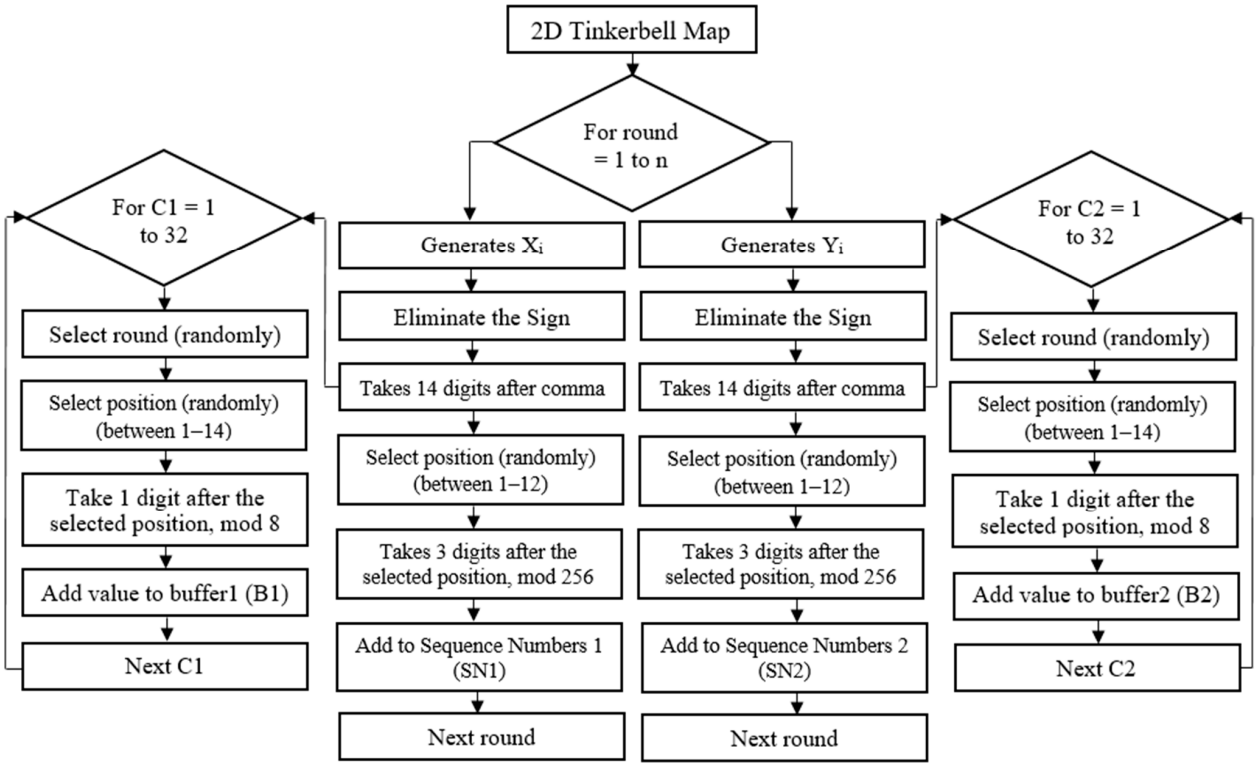

Figure 1. Generating sequence numbers 1, 2, and shift values using the 2D Tinkerbell map.

Algorithm 2 Generating sequence numbers 1, 2, and buffers 1 and 2 based on the 2D Tinkerbell map

Input: Initial conditions for the 2D Tinkerbell map (a, b, c, d, X_0, and Y_0)
Output: Generates sequence numbers 1, sequence numbers 2 and shift values (buffer 1 and buffer 2)
Begin

1. Read the initial conditions.
2. For round = 1 to n // n is a dynamic value, defined by the user randomly
   - 2.1: Generates X_i and Y_i using Equation (1) // The 2D Tinkerbell map
   - 2.2: Next round
3. Generates sequence numbers 1 (768 bits) depending on X_i values
   - 3.1: For j = 1 to 96
   - 3.2: Read X_i, remove the Sign and takes 14 digits after comma
   - 3.3: Select position randomly (not exceed 12), takes 3 digits after it, takes mod 256, convert to 8-bit binary format and store the result in sequence numbers 1.
   - 3.4: Next j
4. Generates sequence numbers 2 (768 bits) depending on Y_i values
   - 4.1: For k = 1 to 96
   - 4.2: Read Y_i, remove the Sign and takes 14 digits after comma
   - 4.3: Select position randomly (not exceed 12), takes 3 digits after it, takes mod 256, convert to 8 bits binary and store the result in sequence numbers 2.
   - 4.4: Next k
5. Generates shift values (buffer 1) depending on X_i
   - 5.1: For C1 = 1 to 32
   - 5.2: Select X_i randomly, read its value, remove the Sign, and takes 14 digits after comma
   - 5.3: Select position randomly (not exceed 14), read its value and store result in buffer 1
   - 5.4: Next C1
6. Generates shift values (buffer 2) depending on Y_i
   - 6.1: For C2 = 1 to 32
   - 6.2: Select Y_i randomly, read its value, remove the Sign, and takes 14 digits after comma
   - 6.3: Select position randomly (not exceed 14), read its value and store result in buffer 2
   - 6.4: Next C2

End

The same steps of Algorithm 2 can be followed to generate sequence numbers 3, sequence numbers 4, and shift values (buffer 3, buffer 4) based on the 2D Duffing map for S-Box2 by using Equation (2), as shown in Figure 2.

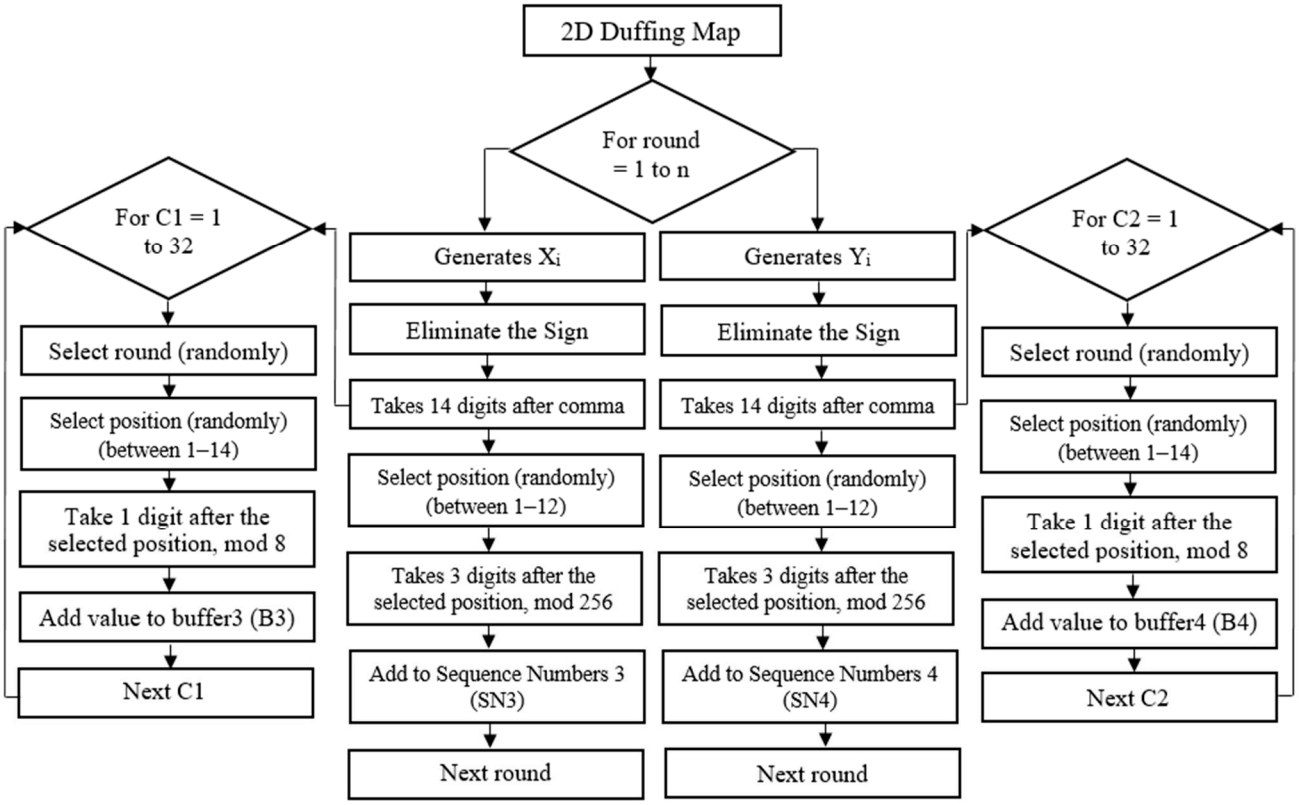

Figure 2. Generating sequence numbers 3, 4, and shift values using the 2D Duffing map.

B. The construction phase includes the construction of the two dynamic S-Boxes. To construct S-Box1 (16 × 16), the user randomly inputs secret key1 and secret key2, each consisting of 24 symbols. These secret keys are then transformed into a binary strings of 192 bits. The binary strings are then converted into a DNA coding of 768 bits. The DNA coding of secret key1 is XORed with sequence number 1 and shifted by buffer 1; each value is converted into HEX format, eliminating duplicate values, and adding the values to S-Box1 in the order of their appearance. On the other hand, the DNA coding of secret key2 is XORed with sequence numbers 2 and shifted by buffer 2; each value is converted into HEX format, eliminating duplicate values, and adding only values not already present in S-Box1. In cases where values ranging from 00 to FF are missing, they are identified and added accordingly, as illustrated in Figure 3 and Algorithm 3.

The construction process of S-Box2 (16 × 16) involves the user randomly inputting secret key3 and secret key4, each consisting of 24 symbols. These secret keys are then converted into a binary string of 192 bits. Subsequently, the binary strings are transformed into a DNA coding of 768 bits. The DNA coding of secret key3 is additive with sequence number 3, shifted by buffer 3, converting each value into HEX format, eliminating duplicate values, and adding the values to S-Box2 in the order of their appearance. Similarly, the DNA coding of secret key4 is additive with sequence numbers 4, shifted by buffer 4, converting each value into HEX format, eliminating duplicate values, and adding only values that are not already presented in S-Box2. In exceptional cases, any values ranging from 00 to FF that are missing are identified and supplemented accordingly. Depending on the inputs provided, the construction of S-Box2 can follow the same steps as algorithm 3, with the

only difference being that in step 2, the XOR process is replaced by the additive process, as depicted in Figure 3.

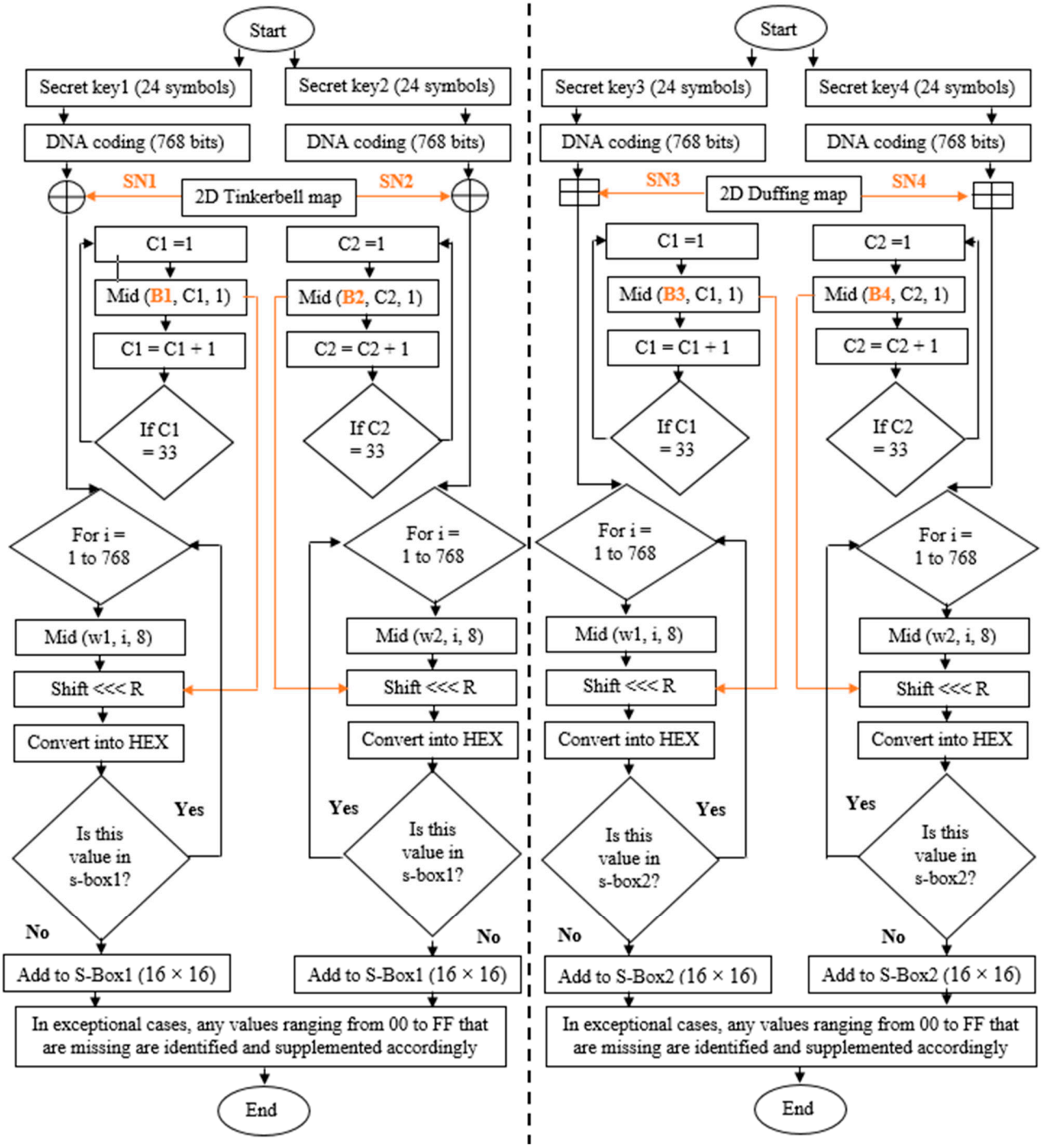

Figure 3. Construction phase of S-Box1 and S-Box2.

Algorithm 3 Construction of S-Box1

Input: Secret key1 (24 symbols), secret key2 (24 symbols), buffer 1, buffer 2.
Output: S-Box1 (16 × 16).
Begin

1. Convert the secret key1 and secret key2 to DNA codes and store the results in D1 and D2 // D1 (result of secret key1) and D2 (result of secret key2).
2. Do the following:
   - 2.1: XOR operation between sequence numbers 1and bits value of D1 and store the results in W1 // W1 is binary string (768 bits), // in first, W1 = " ".
   - 2.2: XOR operation between sequence numbers 2 and bits value of D2 and store the results in W2 // W2 is binary string (768 bits), in first, W2 = " ".
3. For i = 1 to 768
   - 3.1: Mid (W1, i, 8)
   - 3.2: Mid (buffer 1, C1, 1) mod 8 and store the result in U1 // in first, C1 = 1
   - 3.3: Shift the result of step 3.1 by U1 value, convert to HEX format and store the result in S-Box1 if value not already presented in S-Box1
   - 3.4: C1 = C1 + 1
   - 3.5: if C1 = 33 then C1 = 1
   - 3.6: Next i
4. For j = 1 to 768
   - 4.1: Mid (W2, j, 8)
   - 4.2: Mid (buffer 2, C2, 1) mod 8 and store the result in U2 //in first, C2 = 1
   - 4.3: Shift the result of step 4.1 by U2 value, convert to HEX format and store the result in S-Box1 if value not already presented in S-Box1
   - 4.4: C2 = C2 + 1
   - 4.5: if C2 = 33 then C2 = 1
   - 4.6: Next j.
5. In cases where values ranging from 00 to FF are missing, they are identified and added accordingly.

End

4. Performance Evaluation

This section demonstrates the security strength of the proposed 8-bit S-Box, measured over balanced, bijective property, strict avalanche criteria, linear branch number, differential branch number, differential uniformity, non-linearity, fixed points, linear and nonlinear operations, algebraic degree, and algebraic attacks. It also compares the cryptanalysis of the proposed 8-bit S-Box with other 8-bit S-Box such as AES [5], Kim, H. et al. [6], Fantomas [7], Robin [7], Scream v3 [8], FLY [9], and Sajjad, M. et al [11,13,15].

A. Balanced

The obtained S-Box is balanced since it contains an equal number of 1s and 0s in the corresponding truth table of Galois field (GF) (2^8).

B. Bijective property

The obtained S-Box is bijective by balancing 0s and 1s. The Hamming weight of all Boolean functions is [128 128 128 128 128 128 128 128].

C. Strict Avalanche Criteria (SAC)

If any individual input bit is altered, it should affect approximately half of the output bits [23]. An SAC score of around 0.5 is considered sufficient. Tables 2 and 3 illustrate the dependency matrix of SAC scores for the proposed S-Boxes. Our S-Box1 has an average SAC score of 0.57645, while our S-Box2 has an average SAC score of 0.57227. Table 4 compares the average SAC score of our proposed S-Boxes with other S-Boxes in the literature.

Table 2. Dependency matrix of proposed S-Box1.

0.5	0.515625	0.515625	0.546875	0.515625	0.53125	0.484375	0.40625
0.5	0.5	0.59375	0.46875	0.546875	0.5625	0.46875	0.5625
0.484375	0.484375	0.546875	0.59375	0.515625	0.5625	0.515625	0.453125
0.46875	0.453125	0.46875	0.453125	0.546875	0.5	0.515625	0.46875
0.5	0.53125	0.5625	0.515625	0.53125	0.421875	0.53125	0.46875
0.484375	0.546875	0.453125	0.484375	0.515625	0.515625	0.484375	0.5
0.46875	0.5625	0.46875	0.53125	0.46875	0.5	0.515625	0.484375
0.5	0.484375	0.46875	0.546875	0.53125	0.5	0.453125	0.484375

Table 3. Dependency matrix of proposed S-Box2.

0.4375	0.515625	0.546875	0.484375	0.5	0.421875	0.53125	0.484375
0.484375	0.5	0.5	0.46875	0.453125	0.484375	0.5	0.484375
0.453125	0.515625	0.453125	0.515625	0.5	0.484375	0.53125	0.453125
0.546875	0.546875	0.46875	0.546875	0.515625	0.484375	0.453125	0.4375
0.46875	0.5625	0.46875	0.46875	0.4375	0.515625	0.515625	0.5
0.53125	0.53125	0.5	0.5625	0.5	0.484375	0.421875	0.5
0.59375	0.53125	0.53125	0.515625	0.625	0.5	0.515625	0.4375
0.515625	0.5	0.484375	0.546875	0.46875	0.578125	0.484375	0.546875

Table 4. Constituent Boolean functions and NL scores.

Boolean Function	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**	**B8**
NL (B), S-Box1	108	106	102	104	106	104	108	106
NL (B), S-Box2	104	106	106	106	104	106	106	104

D. Linear Branch Number

The linear branch number (LBN) of an S-Box [24] is defined as

$$\min_{a,b,\Phi(a,b)\neq 0}(wt(a)+wt(b)). \tag{3}$$

The upper bound of LBN $\leq n-1$, where n is input bits. A higher LBN enhances the system's resistance against linear attacks [25]. The proposed S-Box1 and S-Box2 have an LBN of 4, as shown in Table 4, a milestone not attained by any other research to date.

E. Differential Branch Number

The differential branch number (DBN) of an S-Box [26] is defined as

$$\min_{a,b\neq a}(wt(a\oplus b)+wt(S(a)\oplus S(b))). \tag{4}$$

The user calculates all possible XORs between the Galois field (GF) (2^n) entries and then all possible XORs between the S-Box entries. Finally, the user calculates their Hamming weights and compute the minimum value. The upper bound of DBN is $[\frac{2n}{3}]$, where n is input bits. A higher DBN enhances the resistance of the system against differential attacks [27]. The proposed S-Box1 and S-Box2 have a DBN of 4, as shown in Table 4, a milestone not attained by any other research to date.

F. Differential Uniformity

The differential uniformity (DU) of an S-Box [28] is defined as

$$\max_{\Delta\alpha\neq 0,\Delta\beta} \#\left\{x \in F_n^2 S(x) \oplus S(x \oplus \Delta\alpha) = \Delta\beta\right\}. \quad (5)$$

The lower the DU, the better the resistance to differential attack [29]. The lowest possible DU is two [30]. The proposed S-Box1 achieved a maximum DU score of 12, while S-Box2 achieved a score of 14. This low score affirms the potential of the proposed S-Boxes to resist differential cryptanalysis effectively. Furthermore, Table 4 compares the DU scores of the proposed S-Boxes with those found in the existing literature. This comparison underscores the efficacy of our S-Boxes in thwarting attempts at differential cryptanalysis.

G. Non-Linearity

The non-linearity (NL) of an S-Box [26] is defined as

$$2^{n-1} - 2^{-1} \times \max_{\lambda\alpha,\lambda\beta\neq 0} |\Phi(\lambda\alpha, \lambda\beta)|, \text{ where } \Phi\ (\lambda\alpha,\ \lambda\beta) = \sum_{x\in F_n^2} (-1)^{\lambda\beta \bullet S(x) \oplus \lambda\alpha \bullet x} \quad (6)$$

where "·" is the inner product in the respective Galois field. A higher NL implies greater complexity and unpredictability in the relationship between inputs and outputs, which enhances the resistance of the cryptographic algorithm against various attacks, including differential cryptanalysis and linear cryptanalysis [31]. The maximum NL in the GF (2^n) is (a) for n = even is $2^{n-1} - 2^{\frac{n}{2}-1}$ and (b) for n = odd is $2^{n-1} - 2^{\frac{n-1}{2}}$. The proposed S-Box1 attained an NL of 105.50, while the proposed S-Box2 attained an NL of 105.25, as demonstrated in Tables 4 and 5. However, it is worth noting that some S-Boxes have higher non-linearity but are static, which poses a significant weakness.

Table 5. Comparison of the cryptographic characteristics and operation counts of 8-bit S-Boxes.

Reference	Differential Branch Number	Linear Branch Number	Differential Uniformity	Non-Linearity	Algebraic Degree	Fixed Points	Nonlinear Operations	Linear Operations	Strict Avalanche Criteria
Proposed S-Box1	4	4	12	105.50	7	1	0	2	0.57645
Proposed S-Box2	4	4	14	105.25	7	0	2	0	0.57227
[6], Listing1	3	3	16	96	6	16	12	30	0.5444
[6], Listing2	3	3	16	96	5	1	12	32	0.5234
[6], Listing3	3	3	16	96	5	0	11	24	0.5103
[10], PIPO	3	3	16	96	5	0	11	23	0.5469
[9], FLY	3	3	16	96	5	1	12	24	0.5400
[7], Fantomas	2	2	16	96	5	0	11	27	0.4858
[7], Robin	2	2	16	96	6	16	12	24	0.5188
[8], Scream v3	2	2	8	96	6	0	12	27	0.4985
[5], AES	2	2	4	112	7	0	35	93	0.5049
[11], S-Box A	2	2	10	107	7	0	30	45	0.504
[11], S-Box B	2	2	10	107	7	0	30	45	0.504
[13], S-Box A	3	3	12	107.50	7	1	12	24	0.5
[13], S-Box B	3	3	12	106.50	7	0	12	24	0.5
[13], S-Box C	3	3	12	106.75	7	0	12	24	0.5
[13], S-Box D	2	2	12	106.75	7	2	12	24	0.508
[15], S-Box1	3	3	12	107.50	7	1	20	32	0.57896
[15], S-Box2	3	3	12	106.50	7	2	20	32	0.56948

H. Fixed Points (FP)

If an S-Box returns the same value as its input (S-Box (m) = m), it is said to contain an FP [31]. An S-Box with fewer FPs is more resistant to attacks. The proposed S-Box1 possesses one FP, while the proposed S-Box2 is free of any FP. Table 4 compares the FP of the proposed S-Boxes with other S-Boxes in the literature.

I. Linear and Nonlinear Operations

The fewer linear (XOR, NOT) and nonlinear (AND, OR) operations, the better. In this method, the proposed S-Box1 does not utilize nonlinear operations but employs two linear operations, whereas the proposed S-Box2 utilizes two nonlinear operations and no linear operations, as shown in Table 4.

J. Algebraic Degree

The algebraic degree is the number of variables in the highest order term with non-zero coefficients. The maximum algebraic degree can be denoted as deg (f) = n − 1. A higher algebraic degree is preferable [30]. The proposed S-Box1 and S-Box2 achieved a favorable algebraic degree of 7, as evidenced in Table 4.

K. Algebraic Attacks

The structure of the proposed S-Box is robust. The total number of possible S-Boxes is 3.35×10^{504}, which is massive and noticeably greater than the 6-bit S-Box ($63! \approx 1.98 \times 10^{87}$), 5-bit S-Box ($31! \approx 8.22 \times 10^{33}$), and 4-bit S-Box ($15! \approx 1.3 \times 10^{12}$). Furthermore, the proposed 8-bit S-Box uses a dynamic chaotic system to introduce randomness into the S-Box's elements, making it difficult to break through.

5. Conclusions

This research introduces a novel approach to constructing dynamic S-Boxes by utilizing the 2D Tinkerbell map and the 2D Duffing map. The resulting S-Boxes, namely S-Box1 and S-Box2, achieved a remarkable linear and differential branch number of four, surpassing any previous studies. In the symmetry analysis, the reduction in the number of linear and nonlinear operations for both S-Boxes makes them suitable for lightweight algorithms. This increased robustness improves resistance against various attacks, especially linear and differential attacks. The generation of any S-Box1 value relies on the initial values of the 2D Tinkerbell map, secret keys 1 and 2, the rounds, and the selected positions to generate shift values. Similarly, the generation of any S-Box2 value depends on the initial values of the 2D Duffing map, secret keys 3 and 4, the rounds, and the selected positions to generate shift values. Consequently, any straightforward manipulation could potentially result in a collapse of the S-Box values. Subsequent studies might incorporate these S-Boxes into lightweight algorithms that currently lack S-Boxes, such as Tiny Encryption Algorithm, Lightweight Encryption Algorithm, etc., or into algorithms with static S-Boxes. Subsequently, they can compare the algorithmic results before and after the incorporation of these S-Boxes.

Author Contributions: Conceptualization, A.T.K.; software, A.T.K.; validation, A.T.K., A.T.M. and E.K.G.; formal analysis, A.T.K.; investigation, A.T.K., A.T.M. and E.K.G.; resources, A.T.K.; data curation, A.T.K.; writing—original draft, A.T.K.; writing—review and editing, A.T.K., A.T.M. and E.K.G.; visualization, A.T.K., A.T.M. and E.K.G.; supervision, A.T.M. and E.K.G.; project administration, A.T.K., A.T.M. and E.K.G.; funding acquisition, A.T.K. and E.K.G. All authors have read and agreed to the published version of the manuscript.

Funding: The authors received no specific funding for this study.

Data Availability Statement: Data sharing is not applicable to this article as no datasets were generated or analyzed during the current study.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

➢ A complete example of the generation of S-Boxe1 employing the 2D Tinkerbell map is shown below:

Initialization phase:

Run the 2D Tinkerbell map using a = 0.9, b = −0.6013, c = 2, d = 0.5, $X_0 = -0.721$, and $Y_0 = -0.64$

When i = 1, $X_1 = (-0.721)^2 - (-0.64)^2 + 0.9 \times (-0.721) + (-0.6013) \times (-0.64) = -0.153827$

$Y_1 = 2 \times (-0.153827) \times (-0.64) + 2 \times (-0.153827) + 0.5 \times (-0.64) = -0.43075544$

- ❖ The process of generating the first sequence number from X_i: the value of X_1 is −0.153827, eliminate from sign, comma and take 14 digits after comma; the result is 153827. Select position randomly and take 3 digits after the selected position mod 256, assume position 2 is selected so, 538 mod 256 = 26, convert to 8-bit binary; the result is 00011010.
- ❖ The process of generating the first sequence number from Y_i: the value of Y_1 is −0.43075544, eliminate from sign, comma and take 14 digits after comma; the result is 43075544. Select position randomly and take 3 digits after the selected position mod 256, assume position 1 is selected so, 430 mod 256 = 174, convert to 8-bit binary; the result is 10101110.
- ❖ The rest of the results are obtained in the same way, repeating the process for all X_i and Y_i values, as shown in Tables A1–A3.

Table A1. Generate X_i and Y_i using the 2D Tinkerbell map.

Number of i	X_i	Y_i
1	−0.153827	−0.43075544
2	−0.041318557088593721	−0.26241844769946288
3	0.053449292688840327	−0.052362799308130245
4	0.0797050787814888	0.12488159582076486
5	−0.012599246054863504	0.03409547789371576
6	−0.032844692918518312	−0.050881357892837195
...	...	...

Table A2. Eliminate from sign, comma and takes 14 digits after comma for X_i and Y_i.

Number of i	Eliminate from the Sign, Comma and Takes 14 Digits after Comma	Eliminate from the Sign, Comma and Takes 14 Digits after Comma
1	153827	43075544
2	04131855708859	26241844769946
3	05344929268884	05236279930813
4	07970507878148	12488159582076
5	01259924605486	03409547789371
6	03284469291851	05088135789283
...	...	...

Table A3. Generates sequence numbers based on X_i and Y_i values.

Number of i	Takes Digits in Positions (2, 3 and 4) Mod 256	Sequence Numbers 1	Takes Digits in Positions (1, 2 and 3) Mod 256	Sequence Numbers 2
1	538 mod 256 = 26	00011010	430 mod 256 = 174	10101110
2	413 mod 256 = 157	10011101	262 mod 256 = 6	00000110
3	534 mod 256 = 22	00010110	052 mod 256 = 52	00110100
4	797 mod 256 = 29	00011101	124 mod 256 = 124	01111100
5	125 mod 256 = 125	01111101	034 mod 256 = 34	00100010
6	328 mod 256 = 72	01001000	050 mod 256 = 50	00110010
…	…	…	…	…

❖ Assuming both the sender and recipient agree to randomly choose rounds and position values. Table A4 illustrates the process of generating the shift values (buffer 1 and buffer 2).

Table A4. Generates shift values (buffer 1 and buffer 2) using the 2D Tinkerbell map.

Round	Value of Round in (X_i)	Selected Position	Value (Buffer 1)	Round	Value of Round in (Y_i)	Selected Position	Value (Buffer 2)
2	04131855708859	4	3	9	01568453777318	6	4
19	00018347337203	9	3	20	00019622033453	4	1
123	75223450882507	2	5	51	63146373964369	2	3
199	59499357900227	11	0	99	32777311612723	7	1
205	08597748105963	1	0	190	44559510179214	9	1
218	81183553962828	7	5	203	04531685949465	4	3
301	17178211627398	6	2	230	44786377288332	2	4
316	14931033209777	7	3	256	60070844814028	3	0
326	11189268962787	6	2	309	67067293115831	8	3
370	52787277277676	5	7	380	28479188740166	12	1
376	90536667489769	4	3	399	68019348527224	11	7
400	36643941799131	1	3	405	43036508653834	13	3
410	00739259397935	1	0	485	12763746694896	2	2
450	08732526297793	6	5	499	76687040131334	4	8
503	96344266362421	5	4	501	65037364263293	6	3
511	36736606881193	9	8	520	98220759711907	9	7
520	34170755924225	8	5	548	11941949554439	10	5
535	14015253078072	3	0	612	60421367553936	4	2
561	68493424111307	5	3	677	97731066803722	12	7
585	62020371657689	4	2	710	16644013610951	1	1
598	16238900523974	10	2	712	51771151090968	1	5
613	91604236453256	10	5	750	07844990204257	4	4
625	35244705821893	3	2	798	93234377567065	10	6
650	32228600805274	11	5	803	32948913215751	13	5
670	10410926177084	3	4	820	16515109042643	4	1
701	55864934042596	5	4	870	41410722474031	3	4
742	71939806374998	7	0	942	47286654332096	7	5

Table A4. *Cont.*

Round	Value of Round in (X_i)	Selected Position	Value (Buffer 1)	Round	Value of Round in (Y_i)	Selected Position	Value (Buffer 2)
799	51070276089034	8	6	950	44747327706102	3	7
804	52392921118775	12	7	965	29190505354133	10	5
865	43691118433292	6	1	970	63734618050187	8	8
916	87231473813097	13	9	994	15885221704716	6	2
981	73962678859339	8	8	1000	51922620430956	8	0

The values of buffer 1 are "33500523273305485032252544067198"; whereas, buffer 2 contains the values "41311340317328375271546514575820"

The construction phase:

- ✓ Assume user input secret key1 "ph.d in computer science". secret key2 "how_may i assist you now"
- ✓ DNA coding of secret key1:

CTCCAAGTACGAAACTACCCAAGGCCGAGATTCCCGAAAACCGTACTTCTATA
CCTCCATACTCACCCACTACCCGAAGGCCATAAACCCCGAACG

- ✓ DNA coding of secret key2:

CCTCAAAACTAGATAACCGTAATGCTTTGATTCCTTGATTCCCTACTACTCGAA
GGCTCGACCTACCCACGGCCGGACCGACCCAAACCCGGACCA

- ✓ XOR operation between binary of DNA coding of secret key1 and sequence numbers 1, the result is

"0101100111001001010101010101111000111100000010010100001111000111011001100 1
111110010111000101010001010000010001110100101101010101010001010011110101000010
1111001001110000011110010001101011011000010100101110001101101100100010100001011 0
0001110011001100100000011110110011011100100011111010110010001001010010010010001
110100101101001000000000101111000000000110011000001110111101010110101111001101100
0010001110011000000010010100000110011110001100101111101001110001001011110001101 1
1001010110000011011110001101000111100010110100111110101010010111111101000000101 0
1111100010111101111000010100011011001100110011110010101010101001011011101010000 1
0110111010001100100101100110110001011011000000010100010111001111001001111110101 1
0001011111111101111110111110111010000111010001100111010 1"

- ✓ XOR operation between binary of DNA coding of secret key2 and sequence numbers 2, the result is

"1110110101000101011000000011111101100011011100110101101101001110010011000 1
010000010001100100011101000010010101000100000000100000101000011010000110100011 1
0101010011111011101110011101011111101010001111110001000111100110000001110000000 1
1111001110111000001001111001101000000010111000111111001010011110001101010001101 0
1100101110010010110110010010011101010111010001101101011011110111111011011010110 11
1000110100110100101100111001001101100110001101111101000011000111001101011000000 1
1111011000101001010011000110011001010110010110000000011000110111111000010101110 10
0110100010010000010100101011101001011111010010011101010100100100000000101100010 11
1111011101001110001110111001000000000111010010000000000101010101100101001111001 111
1000000010001001001001101010100011011110010011010101 0110"

Eliminate duplicate values in Table A5, and add the values to S-Box1 in the order of their appearance. Proceed similarly with Table A6, ensuring the values are added to S-Box1 sequentially and only if they are not already present. In exceptional instances, identify any values ranging from 00 to FF that are absent and supplement them accordingly, as shown in Table A7.

Table A5. Shift above result from XOR operation depending on buffer 1.

8-Bit Binary	Value of Shift (Buffer 1)	8-Bit after Shifted	Convert to HEX Format
01011001	3	00101011	2B
10110011	3	01110110	76
01100111	5	00111011	3B
11001110	0	11001110	CE
10011100	0	10011100	9C
00111001	5	11001001	C9
01110010	2	10011100	9C
...	...	...	...

Table A6. Shift above result from XOR operation depending on buffer 2.

8-Bit Binary	Value of Shift (Buffer 2)	8-Bit after Shifted	Convert to HEX Format
11101101	4	11011110	DE
11011010	1	01101101	6D
10110101	3	10110110	B6
01101010	1	00110101	35
11010100	1	01101010	6A
10101000	3	00010101	15
01010001	4	00010101	15
...	...	...	...

Table A7. S-Box 1.

2B	76	3B	CE	9C	C9	72	25	A4	49	95	51	55	AA	5D	D5
7D	E5	CB	78	C7	E3	1E	87	0F	E0	C0	0C	80	42	24	94
41	05	A1	1A	E1	C3	7C	1F	E8	1D	EC	B3	CC	99	33	67
7E	E7	9F	F9	3F	5E	79	2E	E2	B8	71	54	45	8A	0A	28
14	82	04	40	44	64	D1	A3	47	74	4F	2D	A5	5A	96	6D
53	75	A8	A2	A0	29	3D	D3	F4	BE	D4	12	85	61	C5	F2
2F	4E	27	38	0E	1C	83	07	19	91	8D	D0	35	B5	6B	DA
AD	63	0B	4A	B4	97	F8	B1	36	BD	B6	D9	56	32	8C	88
2A	43	21	C2	B0	C1	6E	DC	66	46	84	08	02	18	30	3C
9E	F6	9D	CD	37	73	C8	7F	D7	5F	CA	59	22	31	90	92
52	48	23	4B	69	20	00	17	BC	F0	60	98	06	70	DD	DF
DB	B7	7A	AF	AE	5B	F5	CF	E6	1B	8E	39	9B	C4	A7	3E
65	FA	EB	E9	5C	26	89	D2	F1	6C	D8	93	B2	58	68	7B
8F	8B	BA	AB	A9	EF	F7	01	57	ED	FE	FB	2C	AC	D6	BB
EA	A6	50	0D	B9	FC	13	FF	FD	BF	86	DE	6A	15	03	4D
E4	4C	34	09	11	3A	9A	EE	F3	10	81	77	16	C6	6F	62

➢ A complete example of the generation of S-Box2 employing the 2D Duffing map is shown below:
Initialization phase:
Run the 2D Duffing map using a = 2.75, b = 0.15, X_0 = 0.7, and Y_0 = 0.93

When i = 1, $X_1 = 0.93$ and $Y_1 = (-0.15 \times 0.93) + (2.75 \times 0.93) - (0.93)^3 = 1.613643$

- The process of generating first sequence number from X_i: the value of X_1 is 0.93, eliminate from sign, comma and takes 14 digits after comma; the result is 93. Select position randomly and takes 3 digits after the selected position mod 256, assume position 2 is selected so, 3 mod 256 = 3, convert to 8-bit binary; the result is 00000011.
- The process of generates first sequence number from Y_i: the value of Y_1 is 1.613643, eliminate from sign, comma and take 14 digits after comma; the result is 613643. Select position randomly take 3 digits after the selected position mod 256, assume position 2 is selected so, 136 mod 256 = 136, convert to 8-bit binary; the result is 10001000.
- The rest of the results are obtained in the same way, repeats the process for all X_i and Y_i values, as shown in Tables A8–A10.

Table A8. Generates X_i and Y_i using the 2D duffing map.

Number of i	X_i	Y_i
1	0.93	1.613643
2	1.613643	0.0062024103465603275
3	0.0062024103465603275	0.016126028294987611
4	0.016126028294987611	0.041923480012845224
5	0.041923480012845224	0.10892736423984879
6	0.10892736423984879	0.28191870525515228
...	...	...

Table A9. Eliminate from sign, comma and takes 14 digits after comma for X_i and Y_i.

Number of i	Eliminate from the Sign, Comma and Takes 14 Digits after Comma	Eliminate from the Sign, Comma and Takes 14 Digits after Comma
1	93	613643
2	613643	00620241034656
3	00620241034656	01612602829498
4	01612602829498	04192348001284
5	04192348001284	10892736423984
6	10892736423984	28191870525515
...	...	...

Table A10. Generates sequence numbers based on Xi and Yi values.

Number of i	takes Digits in Positions (2, 3 and 4) Mod 256	Sequence Numbers 3	Takes Digits in Positions (2, 3 and 4) Mod 256	Sequence Numbers 4
1	3 mod 256 = 3	00000011	136 mod 256 = 136	10001000
2	136 mod 256 = 136	10001000	062 mod 256 = 62	00111110
3	062 mod 256 = 62	00111110	161 mod 256 = 161	10100001
4	161 mod 256 = 161	10100001	419 mod 256 = 163	10100011
5	419 mod 256 = 163	10100011	089 mod 256 = 89	01011001
6	089 mod 256 = 89	01011001	819 mod 256 = 51	00110011
...	...	...	...	...

- Assuming both the sender and recipient agree to randomly choose rounds and position values. Table A11 illustrates the process of generating shift values (buffer 3 and buffer 4).

Table A11. Generates shift values (buffer 3 and buffer 4) using the 2D Duffing map.

Round	Value of Round in (X_i)	Selected Position	Value (Buffer 3)	Round	Value of Round in (Y_i)	Selected Position	Value (Buffer 4)
12	18593214845466	4	9	3	01612602829498	4	1
20	87506258198913	3	5	15	09793636228533	7	3
55	09271183018619	10	1	46	44124453946982	9	9
77	50990559007907	9	0	90	42973302791717	10	9
109	31943524005919	2	1	100	26723979306017	12	0
121	52316639095945	8	9	146	08900254185067	2	8
152	50782431167485	11	7	186	60088395125526	11	5
160	0929663449005	9	4	200	30318210376056	9	3
200	57393877668195	3	3	254	19127253136421	7	5
225	59362969399894	2	9	265	36508183684438	1	3
280	23088588434095	1	2	300	11837154607203	2	1
296	01044509908869	7	0	358	15884583637967	6	5
311	47513618864565	6	6	362	63540556201264	12	2
320	28191118809258	7	1	398	4257080980998	9	8
370	00444277740266	9	7	400	57385140618546	1	5
385	24263067398517	12	5	450	26548713745428	8	3
429	73911240108350	1	7	475	94224920230115	1	9
508	61215313640466	9	6	529	6105155976966	5	1
563	18884661563008	5	4	546	11132497372232	10	7
570	02926268812163	9	8	566	06371251418742	9	4
577	13054739937274	10	3	629	55551098903416	13	1
602	85593287759666	13	6	659	38522228867594	3	5
619	15499671076079	3	4	700	32320716990016	9	9
630	55551098903416	8	8	748	59543463057901	7	6
688	60283888109877	11	9	845	94505758387023	8	8
700	1250627843833	9	4	866	02976949427003	12	0
722	19564417742130	7	1	900	99953422006350	1	9
800	15017038758020	9	7	932	41741358037506	9	0
809	21073689470264	8	9	965	96475683417404	6	6
900	36705585508974	12	9	979	77274787232493	9	2
944	13491602891716	6	6	988	61324593262502	8	3
956	60348754062882	8	4	999	99884485431275	3	8

The values of buffer 3 are "95101974392061757648364894179964". Whereas buffer 4 contains the values "13990853531528539174159680906238".

The construction phase:

✓ Assume user input secret key3 "Life's a journey, enjoy!" and secret key4 "Dream Big, achieve more".

✓ DNA coding of secret key3:

CAGCAAGGCCAAAACGACAGACTAACCCAATGACCCAAGCCCGGACCGCTCAAAACCCATACGGACGCGATTCCATAAACCCTAAAAACTTTGATG

✓ DNA coding of secret key4:

CAACACTCCCATAATGCCGTGATTCACAAAGGCCAGGAATACCCAATGCCCGA
AGTCCTTAACGCTAAAACGACCCAAAGCCGGACTCCCATGAAC

✓ Additive operation between binary of DNA coding of secret key3 and sequence numbers3, the result is

"0100011011001001100001011110010011100100100110100111101010110000101110100
0001011000111011001110011011011111110101110011100011010010110011111001110000000
0011010101110011101111100010111001101101001000001100000101011011000100010100001 0
0010010111000100000000110010001100010100101011011010001000111010110010011111 0001
1000101011111100001111110111001011010010100110000000100011011111101011001100001 0
1110110110111111010110010110000101111000011100000110101011010111010011100000010 00
1110000010101000100110010000010000010111100111101011110000100010010011110010010 0
0110010101101010111100010010100101011000110010101000100110100100100000111111100 0
0100011111101110111110101110110101011110001010010100111001011010111100100101110 0
111100100111110111001111011010111010100011110000 10100000"

✓ Additive operation between binary of DNA coding of secret key4 and sequence numbers 4, the result is

"1100101101111111111000101110011010011010011101101011110110111010000010110 0
0111111001110011101110111101011100101001001110101111111110011100000010001101011
0000110110000100001101110000000000110011110000001010110010001000101000010001001 0
1101100010000000110010100100010100101011011010000000100000011001111111010111000 10
0000001101001111010110111011001110100110100000011011011101101110011100011011011 1
0010111110101101001100100011110000111000000110110011011111110011100000001010111000
0010111011100101111111111000010011101001001011110000101111010011010001111001100 1
0101010111111000000001011010101011011001100011110001010011010000011111110000100 01
0111110010111110101101110001100100000101111010011100101110000000010101011100111000
0101101100101111000111100010101110110111011001101000000100"

Eliminate duplicate values in Table A12, and add the values to S-Box2 in the order of their appearance. Proceed similarly with Table A13, ensuring the values are added to S-Box2 sequentially and only if they are not already present. In exceptional instances, identify any values ranging from 00 to FF that are missing and supplement them accordingly, as shown in Table A14.

Table A12. Shift above result from additive operation depending on buffer 3.

8-Bit Binary	Value of Shift (Buffer 3)	8-Bits after Shifted	Convert To HEX Format
01000110	9 mod 8 = 1	00100011	23
10001101	5	01101100	6C
00011011	1	10001101	8D
00110110	0	00110110	36
01101100	1	00110110	36
11011001	9 mod 8 = 1	11101100	EC
10110010	7	01100101	65
. . .	. . .	. . .	. . .

Table A13. Shift above result from additive operation depending on buffer 4.

8-Bit Binary	Value of Shift (Buffer 4)	8-Bits after Shifted	Convert to HEX Format
11001011	1	11100101	E5
10010110	3	11010010	D2
00101101	9 mod 8 = 1	10010110	96
01011011	9 mod 8 = 1	10101101	AD
10110111	0	10110111	B7
01101111	8 mod 8 = 0	01101111	6F
11011111	5	11111110	FE
...	...	...	...

Table A14. S-Box 2.

23	6C	8D	36	EC	65	46	39	C9	89	4C	62	18	C2	16	0B
2C	71	2F	CB	F2	97	72	9C	4E	27	E4	49	24	A4	D4	53
1A	5A	D3	9E	A7	3D	E9	F4	D7	AE	D5	75	59	CA	58	61
85	8B	B8	5D	A3	74	D0	50	A0	05	C6	B1	8F	3A	67	3B
CE	76	93	B9	CD	E6	D8	D6	ED	BD	B7	DF	BF	FD	5F	FA
BA	2E	73	9B	CF	38	0D	43	B4	A5	D2	AC	9D	B3	1F	F8
9F	7E	F3	E0	C1	06	01	00	60	9A	A6	EA	57	AB	3F	BB
3E	F1	E2	17	5C	DC	37	B6	AD	69	52	21	09	10	0A	0C
41	54	95	6D	DA	88	22	A2	15	14	04	11	13	E5	20	08
02	81	32	19	91	98	31	A8	82	94	29	56	B2	5B	6B	86
D1	51	44	1D	F5	4F	C7	E3	8C	03	C4	26	C5	45	AF	FE
E7	78	E1	C3	1E	7F	FB	77	BC	96	4B	25	A9	30	C0	40
80	C8	6E	7B	BE	D9	CC	66	6F	DB	DE	7D	2D	79	87	F0
0E	70	83	07	A1	68	6A	2B	1C	47	8A	33	42	28	F9	FC
EB	3C	12	84	92	4A	64	2A	4D	48	90	EF	EE	DD	F7	7A
5E	F6	B5	E8	0F	FF	34	63	B0	1B	35	7C	55	AA	8E	99

References

1. Bogdanov, A.; Knudsen, L.R.; Leander, G.; Paar, C.; Poschmann, A.; Robshaw, M.J.B.; Seurin, Y.; Vikkelsoe, C. *Present: An Ultra-Lightweight Block Cipher*; Springer: Cham, Switzerland, 2007; Volume 4727, pp. 450–466.
2. Basha, H.A.; Mohra, A.S.; Diab, T.O.; Sobky, W.I. Efficient Image Encryption Based on New Substitution Box Using DNA Coding and Bent Function. *IEEE Access.* **2022**, *10*, 66409–66429. [CrossRef]
3. Afify, E.W.; Sobky, W.I.; Khalil, A.T.; Alez, R.A.; Wahba, E.; El, W.I.; Twakol, A. Algebraic construction of powerful substitution box. *Int. J. Recent Technol. Eng.* **2020**, *8*, 405–409. [CrossRef]
4. Abd Zaid, M.; Hassan, S. Proposal Framework to Light Weight Cryptography Primitives. *Eng. Technol. J.* **2022**, *40*, 516–526. [CrossRef]
5. Daemen, J.; Rijmen, V. *The Design of Rijndael: AES—The Advanced Encryption Standard*, 2nd ed.; Information Security and Cryptography; Springer: Berlin/Heidelberg, Germany, 2020.
6. Kim, H.; Jeon, Y.; Kim, G.; Kim, J.; Sim, B.; Han, D.; Seo, H.; Kim, S.; Hong, S.; Sung, J.; et al. A New Method for Designing Lightweight S-Boxes with High Differential and Linear Branch Numbers, and its Application. *IEEE Access* **2021**, *9*, 150592–150607. [CrossRef]
7. Grosso, V.; Leurent, G.; Standaert, F.; Varici, K. *Ls-Designs: Bitslice Encryption for Efficient Masked Software Implementations*; FSE Springer: Berlin/Heidelberg, Germany, 2014; Volume 8540, pp. 18–37.
8. Adomnicai, A.; Berger, T.P.; Clavier, C.; Francq, J.; Paul, H.L.V.; Gouguec, K.L.; Minier, M.; Reynaud, L.; Thomas, G. *Lilliput-AE: A New Lightweight Tweakable Block Cipher for Authenticated Encryption with Associated Data Submitted to NIST Lightweight Cryptography Standardization Process*; Université de Lorraine: Lorraine, France, 2019.

9. Karpman, P.; Grégoire, B. The littlun s-box and the fly block cipher. In *Lightweight Cryptography Workshop*; hal.science: Gaithersburg, MD, USA, 2016.
10. Kim, H.; Jeon, Y.; Kim, G.; Kim, J.; Sim, B.; Han, D.; Seo, H.; Kim, S.; Hong, S.; Sung, J.; et al. PIPO: A Lightweight Block Cipher with Efficient Higher-Order Masking Software Implementations. *Int. Conf. Inf. Secur. Cryptol.* **2020**, *12593*, 99–122. [CrossRef]
11. Sajjad, M.; Shah, T.; Alsaud, H.; Alammari, M. Designing pair of nonlinear components of a block cipher over quaternion integers. *AIMS Math.* **2023**, *8*, 21089–21105. [CrossRef]
12. Jassim, S.; Farhan, A.K. Designing a Novel Efficient Substitution-Box by Using a Flower Pollination Algorithm and Chaos System. *Int. J. Intell. Eng. Systems.* **2022**, *15*, 176–187.
13. Sajjad, M.; Shah, T.; Serna, R.J. Designing pair of nonlinear components of a block cipher over gaussian integers. *Comput. Mater. Contin.* **2023**, *75*, 5287–5305. [CrossRef]
14. Zahid, A.H.; Arshad, M.J. An Innovative Design of Substitution-Boxes Using Cubic Polynomial Mapping. *Symmetry* **2019**, *11*, 437. [CrossRef]
15. Sajjad, M.; Shah, T.; ul Haq, T.; Almutairi, B.; Xin, Q. SPN based RGB image encryption over Gaussian integers. *Heliyon* **2024**, *10*, e30353. [CrossRef]
16. Kudhair, A.T.; Gbashi, E.K.; Maolood, A.T. Novel Dynamic S-Box Based on Password Key and Circle Map. *Iraqi J. Sci.* **2023**, *64*, 4767–4778. [CrossRef]
17. Gbashi, E.K.; Maolood, A.T.; Jurn, Y.N. Privacy Security System for Video Data Transmission in Edge-Fog-cloud Environment. *Int. J. Intell. Eng. Syst.* **2023**, *16*, 307–318.
18. Abeer, T.M.; Ekhlas, K.G.; Eman, S.M. Novel lightweight video encryption method based on ChaCha20 stream cipher and hybrid chaotic map. *Int. J. Electr. Comput. Eng.* **2022**, *12*, 4988–5000.
19. Pal, P.K.; Kumar, D.; Agarwal, V. Efficient image encryption using the Tinkerbell map in conjunction with linear feedback shift registers. *Multimed. Tools Appl.* **2023**, *83*, 44903–44932. [CrossRef]
20. Natiq, H.; Roy, A.; Banerjee, S.; Misra, A.P.; Fataf, N.A.A. Enhancing chaos in multistability regions of Duffing map for an asymmetric image encryption algorithm. *Soft Comput.* **2023**, *27*, 19025–19043. [CrossRef]
21. Farhan, A.K.; Subhi, R.; Yassein, H.R.; Mohammed, N.; Al-Saidi, G.; Hameed, G. A new approach to generate multi Sboxes based on RNA computing. *Int. J. Innov. Comput. Inf. Control.* **2020**, *16*, 331–348.
22. Alawi, A.R.; Hassan, N.F. A Proposal Video Encryption Using Light Stream Algorithm. *Eng. Technol. J.* **2021**, *39*, 184–196. [CrossRef]
23. Abdallah, A.A.; Farhan, A.K. New S-Box Design for Image Encryption Based on Multi-Chaotic System. *Eng. Technol. J.* **2023**, *41*, 1211–1219. [CrossRef]
24. Ishfaq, F. A MATLAB Tool for the Analysis of Cryptographic Properties of S-Boxes. Master's Thesis, Department of Mathematics, Capital University of Science & Technology, Islamabad, Pakistan, 2018.
25. Khan, M.A.M.; Azam, N.A.; Hayat, U.; Kamarulhaili, H. A novel deterministic substitution box generator over elliptic curves for real-time applications. *J. King Saud Univ. Comput. Inf. Sci.* **2023**, *35*, 219–236. [CrossRef]
26. Yang, Y.; Dong, H.; Li, Z.; Xiao, S. LWED: Lightweight white-box encryption communication system for drones over CARX algorithm. *J. King Saud Univ. Comput. Inf. Sci.* **2023**, *35*, 101727. [CrossRef]
27. Zhu, D.; Tong, X.; Zhang, M.; Wang, Z. A New S-Box Generation Method and Advanced Design Based on Combined Chaotic System. *Symmetry* **2020**, *12*, 2087. [CrossRef]
28. Özkaynak, F. On the effect of chaotic system in performance characteristics of chaos based s-box designs. *Phys. A Stat. Mech. Its Appl.* **2020**, *550*, 124072. [CrossRef]
29. Hussain, I.; Anees, A.; Al-Maadeed, T.A.; Mustafa, M.T. Construction of S-Box Based on Chaotic Map and Algebraic Structures. *Symmetry* **2019**, *11*, 351. [CrossRef]
30. Zahid, A.H.; Arshad, M.J.; Musheer Ahmad, M.; Soliman, N.F.; El-Shafai, W. Dynamic S-Box Generation Using Novel Chaotic Map with Nonlinearity Tweaking. *Comput. Mater. Contin.* **2023**, *75*, 3011–3026.
31. Steiner, M.J. A lower bound for differential uniformity by multiplicative complexity & bijective functions of multiplicative complexity 1 over finite fields. *Cryptogr. Commun.* **2024**, *16*, 285–308.