

# A hybrid modified lightweight algorithm for achieving data integrity and confidentiality

Lamia A. Muhalha, Imad S. Alshawi

Department of Computer Science, College of Computer Science and Information Technology, University of Basrah, Basrah, Iraq

## Article Info

### Article history:

Received Apr 2, 2022

Revised Jul 18, 2022

Accepted Aug 19, 2022

### Keywords:

Hash function

Keystream generator Salsa20

NIST statistical test suite

Salsa20 algorithm

Speck algorithm

## ABSTRACT

Encryption algorithms aim to make data secure enough to be decrypted by an attacker. This paper combines the Speck and the Salsa20 to make it difficult for an attacker to exploit any weaknesses in these two algorithms and create a new lightweight hybrid algorithm called Speck-Salsa20 algorithm for data integrity and confidentiality (SSDIC). SSDIC uses less energy and has an efficient throughput. It works well in both hardware and software and can handle a variety of explicit plaintext and key sizes. SSDIC solves the difficulties of the Speck algorithm. The sequence generated by Speck is not random and fails to meet an acceptable success rate when tested in statistical tests. It is processed by generating a random key using the Salsa20 algorithm. Salsa20 is a high-speed secure algorithm that is faster than advanced encryption standard (AES) and can be used on devices with low resources. It uses a 256-bit key hash function. The recovery of the right half of the original key of the Speck algorithm is also handled by modifying the Speck round function and the key schedule. Simulation results show, according to a National Institute of Standards and Technology (NIST) test, the performance achieved by the SSDIC is increased by nearly 66% more than that achieved from the Speck in terms of data integrity and confidentiality.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



## Corresponding Author:

Imad S. Alshawi

Department of Computer Science, College of Computer Science and Information Technology

University of Basrah

Basrah, Iraq

Email: emad.alshawi@uobasrah.edu.iq

## 1. INTRODUCTION

The protection of data from unauthorized access, disclosure, alteration, or destruction while ensuring confidentiality, integrity, and availability is very important to information security [1]. As there are unknown risks, threats, and vulnerabilities, there is no 100% guaranteed security [1]–[4]. Cryptography is used to keep data secure while it is in transit (electronic or physical). The increasing demand for the confidentiality of information necessitates the creation of new encryption techniques and algorithms [1], [2], [5], [6]. According to William Stallings, the security of encrypted data is entirely dependent on two factors: the strength of the cryptographic technology and the secrecy of the key [7]. These algorithms must be fast and secure enough to prevent wasting resources in low constrain devices.

Modern encryption algorithms are essential in data transmission systems. Choosing an appropriate encryption algorithm will have an impact on device longevity and performance in terms of battery life, device memory, processing lag, and connection bandwidth [8], [9]. Conventional encryption techniques are slow, complex, and very energy-intensive when dealing with resource-constrained systems [9], [10]. Solutions for resource-limited hardware lightweight algorithms are becoming more common and used [9],

[10]. The face of lightweight cryptography has been a popular research topic for decades. The lightweight cipher design has better performance in resource-limited devices than conventional ciphers because of: (limited block sizes, fewer key sizes, simpler rounds, reduced key tables, and few implementations) [8], [9]. Here are some lightweight algorithms like (Present, Simon, Speck, Rabbit, and Salsa 20/12) [8], [9], [11], [12].

Cryptographic analysts have successfully exploited any flaws in algorithm designs and there have been security breaches in [7]–[9]. To prevent a cryptanalyst from exploiting algorithm weaknesses, two or more algorithms are combined to create a new one called a hybrid cipher. Decrypting a hybrid text is more difficult than decrypting a block or stream of ciphertext [8]. We propose a hybrid algorithm called Speck-Salsa20 algorithm for data integrity and confidentiality (SSDIC) that combines a lightweight stream with a lightweight block cipher to achieve data integrity and confidentiality. It is a secure and lightweight hybrid cipher with greater energy efficiency and effective throughput, as well as good software and hardware performance, and is a suitable choice for devices with low resources. SSDIC is a hybrid lightweight algorithm combined (Speck-Salsa20) that generates a fast key using a hash function that generates a wide range of plain text and key sizes that the SSDIC proposed to address Speck algorithm weaknesses.

When examining the Speck algorithm in statistical analysis, the Speck fails because the generated sequence is not random, and it also restores the right half of the original key. To increase the randomness of Speck, we generate the key using the Salsa20 method instead of using the Speck key. Due to Salsa20 being faster than advanced encryption standard (AES) and is a suitable choice for devices with low resources. It uses a hash function and has a key size of 128 or 256 bits updating the Speck round function and key tables solved the problem of restoring the right half of the original key. SSDIC is more random, secure, and passes all National Institute of Standards and Technology (NIST) SP 800-22 tests with better results rather than a Speck algorithm.

The remaining portions of this work are organized as follows. Section 2 presents an overview of the related work. Section 3 describes the encryption mechanism in detail. Section 4 goes through the proposed encryption technique in detail. Section 5 addresses the system's performance and security. Finally, in section 6, there is a succinct conclusion.

## 2. RELATED WORK

Speck and Salsa20 are discussed separately in several papers. Almazrooe *et al.* [13] proposes a Salsa20 modification based on the logistic map to improve the speed of the Salsa20-based encryption. After the second cycle, an XOR network is used to raise the level of unpredictability by modifying each of the 64 bytes in the sequence, as well as to address the statistical leakage. It generates a 32-byte sequence as a secret key. There is only 1-bit of difference at the inputs of any two sequential blocks of Salsa20. However, there are 33-bits of differences in the proposed chaotic Salsa system. The enlargement of the differences in the inputs can strengthen the system against different types of attacks. It is possible to achieve good diffusion and a faster speed. Against differential attacks, the method performed admirably.

Fukushima *et al.* [14] proposed the ChaCha and Salsa20 algorithm to describe an incorrect injection attack to obtain an X (20) matrix. By skipping add-ons and attacking the initial array, it was able to extract the key. The proposed strategy is evaluated using a small countermeasure. In [15], reported the first quantum attack that uses the cipher's diffusion to estimate add-rotate-XOR (ARX) round differences. At 8 rounds of Salsa with a 256-bit key sequence length, the results were faster. The work is contained in [16]. This algorithm was the first to employ a hybrid strategy (block and stream). It uses a 16-bit input and a 256-bit key to perform twenty rounds. Hummingbird-2 accepts 64-bit input with a key length of 128 bits and is optimized for low-end microprocessors. While it outperforms present, it does have a few disadvantages (initialization before encryption and decryption, distinct encryption, and decryption functions). In [17], Mouha *et al.*'s framework has concentrated on and created an auto-differential coding study of ARX block ciphers for XOR variance. The suggested method significantly decreases search time and allows for the discovery of differential properties of ARX block zeros with high word sizes, such as  $n=48.64$ . When calculating several features, it takes into consideration the differential effect and finds that the differential probability increases by a factor of 416 for Speck and more than 210 for lightweight encryption algorithm (LEA). It demonstrated efficiency by improving Speck and LEA attacks, which attack 1, 1, 4, and 6 more rounds of Speck48, Speck64, Speck96, and Speck128, respectively, and two more rounds of LEA than earlier work.

AES block ciphers are used in the internal functionality of the current format-preserving standard encryption, FF1 and FF3-1. The approach is implemented by altering the cipher to lightweight block ciphers LEA and Speck to improve the speed of FF1 and FF3-1, according to the research. By splitting it into high-performance computing environments and low-performance internet of things (IoT) environments, the encryption speed is studied and compared with the present encryption speed. In comparison to FF1 and

FF3-1, the results revealed an increase in encoding speed. It will be easier to use format-preserving ciphers across multiple systems if their coding speed is improved.

The evaluation is conducted using physical area (GEs): energy, latency, and throughput [8] and found that the Speck software-based ciphers consume the least energy (1.6), have the highest throughput (471.5), and have the lowest latency [8] are algorithms that are software-efficient and lightweight. Speck, Simon, PRIDE, ITUbee, and IDEA are the top five. Algorithms with low latency Speck, Simon, PRIDE, Hummingbird-2, and ITUbee are the top five [8]. The performance of random Speck exceeds the acceptable success rate; we used SSDIC to solve the difficulties of the Speck algorithm.

### 3. BACKGROUND

#### 3.1. Salsa20 algorithm

Salsa20 is a highly reliable stream cipher algorithm that encrypts quickly with a key size of 128 or 256 bits [18] that was submitted to eSTREAM, the encrypt stream cipher project. The hash function is used in Salsa20, which takes 64-byte inputs and outputs 64 bytes [18], [19]. This hash function is implemented as a stream cipher in counter mode [18], [19].

Hash functions include the quarter round (QR), row round (RR), column round (CR), and double round functions (DR) as illustrated in algorithm 1 pseudocode and Figure 1. It accepts as input a 256-bit key ( $k_0, k_1 \dots k_7$ ), a 64-bit counter ( $t_0, t_1$ ), a 64-bit nonce ( $v_0, v_1$ ), and 128-bit constants ( $c_0, c_1 \dots c_3$ ). Salsa20 operates on 32-bit words and maps inputs to a 4x4 matrix, as in (1) to (5) [18], [19]. The QR ( $a, b, c, d$ ) transformation updates the matrix  $X$  four 32-bit words as below [18], [19]. Where the symbol  $\ll$  represents the rotation to the left,  $+$  is arithmetic addition and  $\oplus$  represents a bitwise XOR. There are various types of rounds; for example, Salsa20/12 and Salsa20/8 are considered to be the fastest of the other stream cipher algorithms. Salsa20 algorithm is faster than the AES cipher algorithm and is therefore recommended for typical cryptosystems where speed is more important than confidence [18], [19].

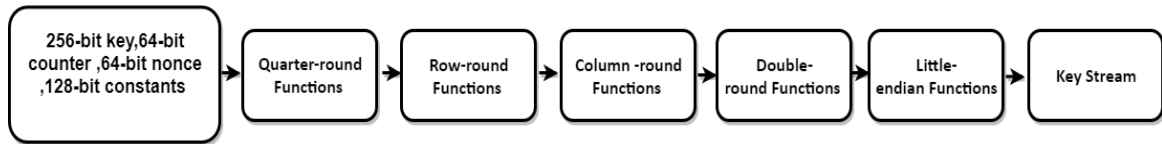


Figure 1. Salsa20 algorithm hash function operating [18], [19]

$$X = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix} = \begin{bmatrix} \sigma_0 & k_0 & k_1 & k_2 \\ k_3 & \sigma_1 & v_0 & v_1 \\ t_0 & t_1 & \sigma_2 & k_4 \\ k_5 & k_6 & k_7 & \sigma_3 \end{bmatrix} \quad (1)$$

$$QR = \begin{cases} b = b \oplus [(a + d) \lll 7] \\ c = c \oplus [(b + a) \lll 9] \\ d = d \oplus [(c + b) \lll 13] \\ a = a \oplus [(d + c) \lll 18] \end{cases} \quad (2)$$

$$RR = \begin{cases} QR(x_0, x_4, x_8, x_{12}) \\ QR(x_5, x_9, x_{13}, x_1) \\ QR(x_{10}, x_{14}, x_2, x_6) \\ QR(x_{15}, x_3, x_7, x_{11}) \end{cases}, CR = \begin{cases} QR(x_0, x_1, x_2, x_3) \\ QR(x_5, x_6, x_7, x_4) \\ QR(x_{10}, x_{11}, x_8, x_9) \\ QR(x_{15}, x_{12}, x_{13}, x_{14}) \end{cases} \quad (3)$$

$$DR = (X) = RR(CR(X)) \quad (4)$$

$$\text{Keystream} = X + DR^r(X) \quad (5)$$

Algorithm 1: Salsa20 keystream generator [18], [19]

Input: key ( $k$ ), block counter ( $c$ ) and nonce ( $n$ )

Output: keystream

1:  $X \leftarrow \text{IntMatrix}(k, c, n)$

2: for  $i=0$  to  $r=9$  do

```

3:      (x0, x4, x8, x12) ← QR(x0, x4, x8, x12)           ▷ Column Round (CR)
4:      (x5, x9, x13, x1) ← QR((x5, x9, x13, x1)
5:      (x10, x14, x2, x6) ← QR(x10, x14, x2, x6)
6:      (x15, x3, x7, x11) ← QR(x15, x3, x7, x11)
7:      (x0, x1, x2, x3)      ← QR((x0, x1, x2, x3)           ▷ Row Round (RR)
8:      (x5, x6, x7, x4)      ← QR((x5, x6, x7, x4)
9:      (x10, x11, x8, x9)     ← QR(x10, x11, x8, x9)
10:     (x15, x12, x13, x14) ← QR(x15, x12, x13, x14)
13:     DR ← RR(CR(X))
14:     Keystream ← X + [[DR]^r(X)
15: end for
    
```

### 3.2. Speck algorithm

The Speck algorithm is one of the lightweight block ciphers. It can handle a large number of different blocks and key sizes [12], [20]. While there are numerous lightweight block ciphers available, the majority is designed for a single platform, and it is not intended to perform well across a range of devices. The purpose of Speck is to address the need for secure and high-performance computing on hardware and software platforms [12], [20] while remaining flexible enough to support a range of implementations on a given platform in a variety of devices running lightweight applications [12], [20].

The Speck is a 2n-bit block that denotes a wn-bit key. 2n/wn, where n must be a minimum of (16, 24, 32, 48, 64) bits in length, and the key must be a minimum of m bits in length. m must be in the range (2, 3, 4) and is version dependent [12], [20]. In the case of Speck 64/128, for instance, 64 block size divide two-part (32-bit) words representing in (x, y) are encrypted using a key 128 divide four-part (32-bit) word representing in  $k=(l[2], l[1], l[0], k[0])$  Beaulieu *et al.* [12] as illustrated in Table 1 [20]. As illustrated in Algorithm 2 pseudocode and Figure 2 of Speck algorithm consists of two iterative components: Figure 2(a) a key schedule and Figure 2(b) a round function denoted by  $R$ , both of which require a round count ( $T$ ) [12], [20]. The main components include bitwise XOR ( $\oplus$ ), addition modulo 2 n (+), and rotation operations ( $S^{\alpha}$ ,  $S^{\beta}$ ), Speck gets its nonlinearity from the modular addition operation, below encryption and decryption (6) to (9) [12], [20]. where ( $S^{\beta}$ ) is left-shift circular and ( $S^{-\alpha}$ ) is right-shift circular, where  $\beta$  and  $\alpha$  are equal to 2 and 7 for block sizes equal to 32; and 3 and 8 for all other block sizes. Where  $k$  is the round key,  $K$  can be denoted by  $(l_{m-2}, \dots, l_0, k_0)$  [12].

$$R(x, y) = \left( (s^{-\alpha}x + y) \oplus k, s^{\beta}y \oplus (s^{-\alpha}x + y) \oplus k \right) \tag{6}$$

$$R_k^{-1}(x, y) = \left( s^{\alpha}((x \oplus k) - s^{-\beta}(x \oplus y)), s^{-\beta}(x \oplus y) \right) \tag{7}$$

$$l_{i+m-1} = (K_i + s^{-\alpha}l_i) \oplus i \tag{8}$$

$$K_{i+1} = s^{\beta}k_i \oplus l_{i+m-1} \tag{9}$$

Table 1. Speck parameters

Block size(2n)	Key size (m*n)	Word size (n)	Key word (m)	Rot $\alpha$	Rot B	Round T
32	64	16	4	7	2	22
48	72,96	24	3,4	8	3	22,23
64	96,128	32	3,4	8	3	26,27
96	96,144	48	2,3	8	3	28,29
128	128,192,256	64	2,3,4	8	3	32,33,34

#### Algorithm 2. Speck key schedules, Speck encryption, and decryption [20]

Input: ( $k_0, l_0, \dots, l_{(m-2)}$ ), round number ( $t$ ), plaintext ( $x, y$ )

Output: ( $k_0, k_{10}, \dots, k_{(t-1)}$ ), ciphertext ( $x, y$ )

```

1: for i=m to t-1 do           ▷ Speck key schedules
2:   [[ l ]_(i+m-1) ← (K_i+s^(-α) l_i) ⊕ i
3:   K_(i+1)      ← s^β k_i ⊕ l_(i+m-1)
4: end for
5: for i=0 to t-1 do         ▷ Speck encryption
6:   x ← (s^(-α) x+y) ⊕ k
7:   y ← s^β y ⊕ x
8:   y ← S^(-β) (X+Y)
9:   x ← S^α ((X⊕K)-Y)
10: end for           ▷ Speck decryption
    
```

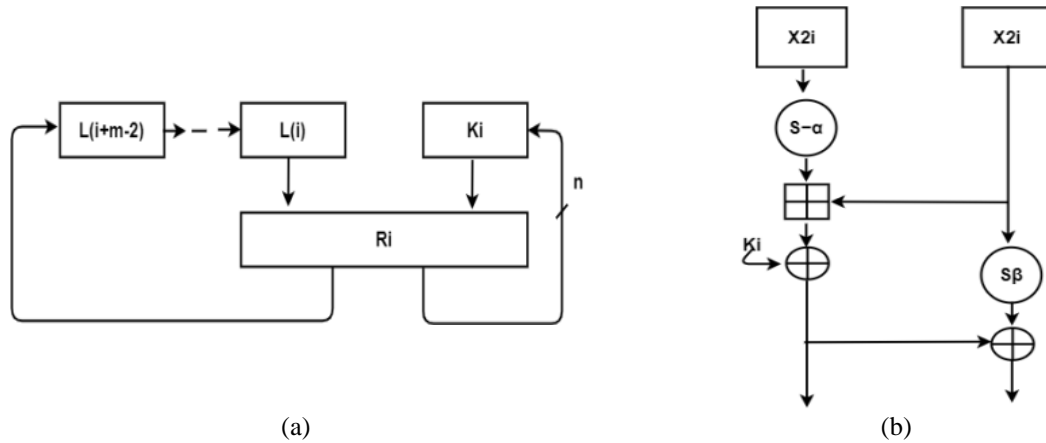


Figure 2. Speck algorithm consists of two iterative components: (a) a key schedule and (b) a round function denoted by  $R$ , both of which require a round count ( $T$ ) [20]

**4. SPECK-SALSA20 FOR DATA INTEGRITY AND CONFIDENTIALITY**

The performance of random Speck exceeds the acceptable success rate, indicating that the algorithm’s sequence is insecure [21], [22]. Because Speck requires the key to be supplied alongside the plaintext to be encrypted, as well as the good attack point in a side-channel attack is where the plaintext directly mixes with the key. The first XOR operation of the round function in Speck is where the plaintext directly mixes with the key. In [23] have already demonstrated that using random plaintext to attack the first round of XOR operation can recover the right half of the original key using a correlation power analysis (CPA) attack, because the round key used in the first round is the right half of the original key [24], as illustrated in Figure 2.

When an attacker can recover the right half of the original key of the Speck method using a CPA attack the Speck method is not secure, as it allows for predictability and provides cryptanalysts with partial understanding as part of the known key. This proposed paper addresses the issue of non-random Speck as well as restoring the right half of the original key by combining Speck and the Salsa20 algorithm as a block cipher and stream cipher called SSDIC method, respectively, and exploiting the strengths of the two algorithms. It is optimized for lightweight applications to ensure performance is applied during software and hardware implementation. The Speck algorithm’s key will not be used; instead, the Salsa20 stream’s keys will be used as the SSDIC algorithm’s key to circumvent the Speck algorithm’s weakness. As a result, Figure 1 illustrates the key generated using the Salsa20 hash function.

The ten Speck variants are designated as Speck  $2n/wn$ . For instance, Speck 128/256 denotes the Speck block cipher with a block size of 128 bits and a key size of 256 bits. Thus,  $n=64$ ,  $w=4$ ,  $\alpha=8$ ,  $\beta=3$  and  $T=34$  is obtained from Table 1. Because no Speck algorithm key is used, a Salsa20 algorithm key must be generated to overcome a weakness, as previously indicated. As a result, Salsa20 must generate a key with a length commensurate with the lengths of the keys listed in Table 1. In this case, Salsa20 must generate 256 keys and then pass them to the Speck algorithm, as illustrated in algorithm 3 pseudocode. Also, Figure 3 shows that the SSDIC method which consists of two iterative components: Figure 3(a) a key schedule and Figure 3(b) a round function denoted by  $R$ .

As illustrated in Figure 1, the keystream is generated using Salsa20 hash functions. It accepts a  $((m*n)/16)*8$ -bit key ( $k0, k1, \dots, k7$ ), a  $((m*n)/16)*2$ -bit counter ( $t0, t1$ ), a  $((m*n)/16)*2$ -bit nonce ( $v0, v1$ ), and  $((m*n)/16)*4$ -bit constants ( $c0, c1, c2, c3$ ) as input. Equation maps the inputs to a  $4 \times 4$  matrix (1). As shown in (1) to (5) are used to generate a keystream with a length of  $(m*n)$ . All inputs to Salsa20 are displayed in Table 2. Following the process of generating the key for use in encryption and decryption using the Speck algorithm, the encryption and decryption (6) and (7) were modified to eliminate the restoring of the right half of the original key, as illustrated in Figure 3. A benefit of any stream cipher-based system is its ease of implementation. However, the strength of such systems is entirely dependent on how the keystream is generated. The ten instances of Speck have been designed to provide excellent performance in both hardware and software, but have been optimized for performance on microcontrollers [20].

$$R(X, Y) = \left( (S^{-\alpha}X + (X \oplus Y)) \oplus K, (S^{\beta}(X \oplus Y)) \oplus (S^{-\alpha}X + (X \oplus Y)) \oplus K \right) \tag{10}$$

$$R_K^{-1} = \left( S^{\alpha}((X \oplus K) - S^{-\beta}(X \oplus Y)), S^{-\beta}(X \oplus Y) \oplus S^{\alpha}((X \oplus K) - S^{-\beta}(X \oplus Y)) \right) \tag{11}$$

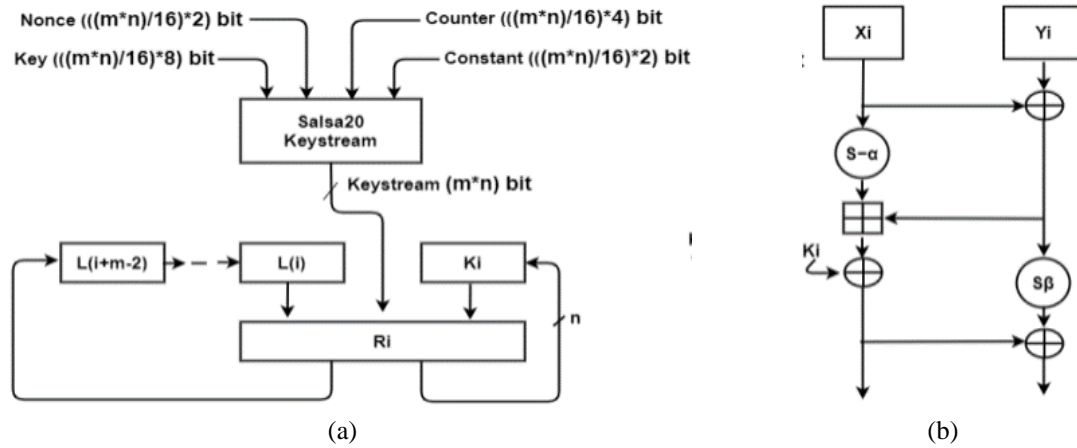


Figure 3. SSDIC method consists of two iterative components: (a) a key schedule of SSDIC and (b) a round function of SSDIC

Table 2. Table parameters input Salsa20 in a bit

Key Stream	Key in bit (k0, ..., k7)	Counter in bit (b0, b1)	Nonce in bit (v0, v1)	Constant in bit (c0, ..., c3)
32	64	16	4	7
48	72, 96	24	3, 4	8
64	96, 128	32	3, 4	8
96	96, 144	48	2, 3	8
128	128, 192, 256	64	2, 3, 4	8

**Algorithm 3. SSDIC algorithm**

**Input:** key (k), block counter (c), nonce (n), (k<sub>0</sub>, l<sub>0</sub>, ..., l<sub>m-2</sub>), round number (t), plaintext (x, y)

**Output:** keystream(KG), (k<sub>0</sub>, k<sub>10</sub>, ..., k<sub>t-1</sub>), ciphertext(x, y)

```

1: X ← IntMatrix(k, c, n)                                ▷ SSDIC generating key
2: for i=0 to r=9 do
3:   (x0, x4, x8, x12) ← QR(x0, x4, x8, x12)             ▷ Column Round (CR)
4:   (x5, x9, x13, x1) ← QR((x5, x9, x13, x1)
5:   (x10, x14, x2, x6) ← QR(x10, x14, x2, x6)
6:   (x15, x3, x7, x11) ← QR(x15, x3, x7, x11)
7:   (x0, x1, x2, x3) ← QR((x0, x1, x2, x3)             ▷ Row Round (RR)
8:   (x5, x6, x7, x4) ← QR((x5, x6, x7, x4)
9:   (x10, x11, x8, x9) ← QR(x10, x11, x8, x9)
10:  (x15, x12, x13, x14) ← QR(x15, x12, x13, x14)
11:  DR ← RR(CR(X))
12:  KG ← X + DRr (X)
13: end for
14: for i=m to t-1 do                                    ▷ SSDIC key schedules
15:  li+m-1 ← (KGi + s-αli) ⊕ i
16:  KGi+1 ← sβKGi ⊕ li+m-1
17: end for
18: for i=0 to t-1 do
19:  x ← ((s-αX + (X ⊕ Y)) ⊕ K                            ▷ SSDIC encryption
20:  y ← (sβ (X ⊕ Y)) ⊕ x
21:  y ← (sα ((X ⊕ K) - s-β (X ⊕ Y))                    ▷ SSDIC decryption
22:  x ← s-β (X ⊕ Y) ⊕ y
23: end for
    
```

**5. RESULTS AND DISCUSSION**

In this paper, we propose a hybrid algorithm called SSDIC that combines a lightweight stream with a lightweight block cipher (Speck and Salsa20) to achieve data integrity and confidentiality to create a flexible and secure hybrid cipher, a remarkable fusion that combines the qualities of both the Speck algorithm and the Salsa20 algorithm. We proposed SSDIC to improve the vulnerabilities of the Speck algorithm. Speck fails to pass an acceptable success rate in the statistical analysis because the generated sequence is not random, and the right half is retrieved from Speck's original main algorithm. Although a lot

of research has been done to exploit Speck's security in terms of linear and differential cipher analysis, few have attempted to address non-randomness, a basic need of all encryption methods.

The SSDIC method is built and implemented in a Python 3.9.7 environment, on a machine with an Intel(R) Xeon(R) CPU E3-1545M v5 running at 2.90 GHz and 8 GB of RAM running Windows 10, Intel(R) Xeon(R) CPU E3-1545M v5 running at 2.90 GHz. The run-time is determined by a timer running in the visual studio code environment. The run-time for the SSDIC and the Speck is 0.0019991 and 0.0010006  $\mu$ s, respectively. Where a small difference in execution time is observed between SSDIC and the Speck algorithm, but it has high randomness and security based on NIST test results shown in Table 3.

Table 3. NIST statistical test suite

NIST Tests	Speck	Proposed Algorithm
Frequency	0.2888	0.4795
Block frequency	0.2888	0.4795
Cumulative sums	0.6548	0.8920
Runs	0.2509	0.4507
Longest run	0.5126	0.9693
Rank	0.1601	0.3601
FFT	0.8711	0.5741
Non-overlapping	0.9999	0.9999
Overlapping template	0.1515	0.9891
Universal	0.2248	0.4455
Approximate entropy	0.4999	0.9999
Random excursions	0.6999	0.7924
Random excursions variant	0.9999	0.9999
Serial	0.4989	0.7134
Linear complexity	0.1746	0.5434

Several statistical tests are also available to evaluate the randomness features of cryptographic algorithms. The statistical analysis is evaluated using NIST SP 800-22. Based on the significance value, the NIST tests determine whether the sequence ratio is random. When the P-value is less than 0.01, the sequence is considered random or vice versa and is called a non-random sequence [25], [26]. The SSDIC method and Speck encryption algorithm are subject to each of the fifteen NIST tests [26]. Test results will also be discussed below.

- Frequency (Monobit) test: passing this test is required for all subsequent tests [25]. In this test, the SSDIC method is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.1907 more than the Speck algorithm, according to NIST tests.
- Frequency block test: in this test, the SSDIC is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.1907 more than the Speck, according to NIST tests.
- Runs test: in this test, SSDIC is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.1998 more than the Speck algorithm, according to NIST tests.
- Longest run test: in this test, SSDIC is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.4567 more than the Speck, according to NIST tests.
- Binary matrix rank test: in this test, SSDIC is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.2 more than the Speck, according to NIST tests.
- Discrete Fourier transform test: in this test, SSDIC is generally less than the Speck, as shown in Table 3. SSDIC decreases nearly 0.297 more than the Speck, according to NIST tests.
- Non-overlapping template matching test in this test, SSDIC is generally equal to the Speck, as shown in Table 3.
- Overlapping template matching test: in this test, SSDIC is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.8376 more than the Speck, according to NIST tests.
- Maurer's "Universal Statistical" Test: In this test, SSDIC is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.2207 more than the Speck, according to NIST tests.
- Linear complexity test: in this test, SSDIC is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.3688 more than the Speck, according to NIST tests.
- Serial test: in this test, SSDIC is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.2145 more than the Speck, according to NIST tests.
- Approximate entropy test: in this test, SSDIC is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.5 more than the Speck, according to NIST tests.
- Cumulative Sums (Cusum) test: in this test, SSDIC is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.0925 more than the Speck, according to NIST tests.

- Random excursions variant test: in this test, the SSDIC method is equal to the Speck, as shown in Table 3 according to NIST tests.
- Random excursions test: in this test, SSDIC is generally superior to the Speck, as shown in Table 3. SSDIC increases nearly 0.0925 more than the Speck, according to NIST tests.

## 6. CONCLUSION

The disadvantage of Speck's algorithm is that the performance of randomization exceeds the allowable success rate. The Speck also requires that the key be supplied with the plaintext to be encrypted, as well as part of the key and part of the known text in the first round. The Speck indicates that the algorithm sequence is not secure because it allows for predictability and gives cryptanalysts partial understanding of the key and plaintext. To increase the non-randomness, this study introduces the SSDIC algorithm to improve the vulnerability of the Speck method by using the Salsa20 algorithm key instead of the Speck algorithm key. Also, change the Speck round function and the key schedule to process recovery of the right half of the original key and plaintext of the Speck algorithm. Random cipher performance was tested using 15 NIST statistical tests, which were created to evaluate pseudo-random numbers of cryptographic applications and successfully bypass the randomness of the SSDIC algorithm.

## REFERENCES




- [1] O. Z. Akif, S. Ali, R. S. Ali, and A. K. Farhan, "A new pseudorandom bits generator based on a 2D-chaotic system and diffusion property," *Bulletin of Electrical Engineering and Informatics (BEEI)*, vol. 10, no. 3, pp. 1580–1588, Jun. 2021, doi: 10.11591/eei.v10i3.2610.
- [2] D. Khwailleh and F. Al-balas, "A dynamic data encryption method based on addressing the data importance on the internet of things," *International Journal of Electrical & Computer Engineering (IJECE)*, vol. 12, no. 2, 2022, doi: 10.11591/ijece.v12i2.pp2139-2146.
- [3] M. D. Aljubaily and I. Alshawi, "Energy sink-holes avoidance method based on fuzzy system in wireless sensor networks," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 12, no. 2, pp. 1776–1785, Apr. 2022, doi: 10.11591/ijece.v12i2.pp1776-1785.
- [4] I. S. Alshawi, Z. A. Abbood, and A. A. Alhijaj, "Extending lifetime of heterogeneous wireless sensor networks using spider monkey optimization routing protocol," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 20, no. 1, pp. 212–220, Feb. 2022, doi: 10.12928/telkomnika.v20i1.20984.
- [5] A. H. Jabbar and I. S. Alshawi, "Spider monkey optimization routing protocol for wireless sensor networks," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 3, pp. 2432–2442, Jun. 2021, doi: 10.11591/ijece.v11i3.pp2432-2442.
- [6] I. S. Alshawi, A.-K. Y. Abdulla, and A. A. Alhijaj, "Fuzzy dstar-lite routing method for energy-efficient heterogeneous wireless sensor networks," *Indonesian Journal of Electrical Engineering and Computer Science (IJECS)*, vol. 19, no. 2, pp. 906–916, Aug. 2020, doi: 10.11591/ijeecs.v19.i2.pp906-916.
- [7] B. Kumar, M. Hussain, and V. Kumar, "BRRC: A hybrid approach using block cipher and stream cipher," in *Advances in Intelligent Systems and Computing*, vol. 563, Singapore: Springer Singapore, 2018, pp. 221–231.
- [8] V. A. Thakor, M. A. Razzaque, and M. R. A. Khandaker, "Lightweight cryptography algorithms for resource-constrained IoT devices: A review, comparison and research opportunities," *IEEE Access*, vol. 9, pp. 28177–28193, 2021, doi: 10.1109/ACCESS.2021.3052867.
- [9] A. Sevin and A. A. O. Mohammed, "A survey on software implementation of lightweight block ciphers for IoT devices," *Journal of Ambient Intelligence and Humanized Computing*, no. 0123456789, 2021, doi: 10.1007/s12652-021-03395-3.
- [10] H. H. Al-Badrei and I. S. Alshawi, "Improvement of RC4 security algorithm," *Advances in Mechanics*, vol. 9, no. 3, pp. 1467–1476, 2021.
- [11] A. Bogdanov *et al.*, "PRESENT: An ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems - CHES 2007*, vol. 4727, Springer Berlin Heidelberg, 2007, pp. 450–466.
- [12] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The Simon and Speck lightweight block ciphers," in *Proceedings of the 52nd Annual Design Automation Conference*, Jun. 2015, vol. 2015, pp. 1–6, doi: 10.1145/2744769.2747946.
- [13] M. Almazrooie, A. Samsudin, and M. M. Singh, "Improving the diffusion of the stream cipher salsa20 by employing a chaotic logistic map," *Journal of Information Processing Systems*, vol. 11, no. 2, pp. 310–324, 2015, doi: 10.3745/JIPS.02.0024.
- [14] K. Fukushima, R. Xu, S. Kiyomoto, and N. Homma, "Fault injection attack on salsa20 and ChaCha and a lightweight countermeasure," in *2017 IEEE Trustcom/BigDataSE/ICSS*, Aug. 2017, pp. 1032–1037, doi: 10.1109/Trustcom/BigDataSE/ICSS.2017.348.
- [15] A. Q. Cruz, "Conditional differential cryptanalysis of the post-quantum ARX symmetric primitive salsa20," Univeristé Denis Diderot Paris 7, 2018.
- [16] D. Engels, X. Fan, G. Gong, H. Hu, and E. M. Smith, "Hummingbird: Ultra-lightweight cryptography for resource-constrained devices," in *Financial Cryptography and Data Security*, Springer Berlin Heidelberg, 2010, pp. 3–18.
- [17] L. Song, Z. Huang, and Q. Yang, "Automatic differential analysis of ARX block ciphers with application to Speck and LEA," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9723, Springer International Publishing, 2016, pp. 379–394.
- [18] D. J. Bernstein, "The salsa20 family of stream ciphers," in *New Stream Cipher Designs*, vol. 4986, Springer Berlin Heidelberg, 2008, pp. 84–97.
- [19] Z. M. J. Kubba and H. K. Hoomod, "A hybrid modified lightweight algorithm combined of two cryptography algorithms PRESENT and salsa20 using chaotic system," in *2019 First International Conference of Computer and Applied Sciences (CAS)*, Dec. 2019, pp. 199–203, doi: 10.1109/CAS47993.2019.9075488.






- [20] A.-V. Duka and B. Genge, "Implementation of Simon and Speck lightweight block ciphers on programmable logic controllers," in *2017 5th International Symposium on Digital Forensic and Security (ISDFS)*, 2017, pp. 1–6, doi: 10.1109/ISDFS.2017.7916501.
- [21] Y. S. S. Risqi and S. Windarta, "Statistical test on lightweight block cipher-based PRNG," in *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, Oct. 2017, vol. 2018, pp. 1–4, doi: 10.1109/TSSA.2017.8272925.
- [22] R. A. F. Lusto, A. M. Sison, and R. P. Medina, "Performance analysis of enhanced Speck algorithm," in *Proceedings of the 4th International Conference on Industrial and Business Engineering*, Oct. 2018, pp. 256–264, doi: 10.1145/3288155.3288196.
- [23] H. Gamaarachchi, H. Ganegoda, and R. Ragel, "Breaking Speck cryptosystem using correlation power analysis attack," *Journal of the National Science Foundation of Sri Lanka*, vol. 45, no. 4, pp. 393–404, Dec. 2017, doi: 10.4038/jnsfsl.v45i4.8233.
- [24] J. Tang, K. Iokibe, T. Kusaka, and Y. Nogami, "An approach for attacking speck on microcontroller with correlation power analysis," in *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, Nov. 2020, pp. 368–372, doi: 10.1109/CANDARW51189.2020.00076.
- [25] L. E. Bassham III *et al.*, "Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications." National Institute of Standards & Technology, 2010.
- [26] E. A. Luengo and L. J. G. Villalba, "Recommendations on statistical randomness test batteries for cryptographic purposes," *ACM Computing Surveys*, vol. 54, no. 4, May 2021, doi: 10.1145/3447773.

## BIOGRAPHIES OF AUTHORS



**Lamia A. Muhalhal**    is a student M.Sc. in the Computer Science Department, College of Computer Science and Information Technology, University of Basra, located in his hometown of Basra, Iraq. She received a B.Sc. degree in Computer Science at Shatt Al-Arab College, Basra, Iraq, in 2018. Recently she has been interested in security and wireless sensor networks. She can be contacted at email: allalemyaa@gamil.com.



**Imad S. Alshawi**    received B.Sc. and M.Sc. degrees in computer science from the College of Science, University of Basrah, Basrah, IRAQ. He received a Ph.D. degree in wireless sensor networks at the School of Information Science and Technology, Information and Communication System Department, Southwest Jiaotong University, Chengdu, China. Mr. Alshawi has been a Prof. of Computer Science and Information Technology at, the University of Basrah, for 20 years. He serves as a frequent Referee for more than fifteen journals. He is the author and co-author of more than 40 papers published in prestigious journals and conference proceedings. He is a member of the IEEE, the IEEE Cloud Computing Community, and the IEEE Computer Society Technical Committee on Computer Communications. He can be contacted at email: emad.alshawi@uobasrah.edu.iq.