GS3: A Lightweight Method of Generating Data Blocks With Shuffling, Scrambling, and Substituting Data for Constrained IoT Devices

Francisco Alcaraz-Velasco, Jose M. Palomares, *Senior Member, IEEE*, and Joaquin Olivares

***Abstract*—The enabling devices and sensors of the Internet of Things (IoT) are characterized by devices with limited resources, where the computation and the energy consumption should be optimized. Application fields, such as healthcare or multimedia content, bring up security and privacy issues. Therefore, data security is critical. However, to obtain it, high-computing resources are required. To avoid it, in this work, we propose a lightweight method to protect data transmissions in sensor devices. We present the *GS3* method, it is based on a procedure set by Generating the data block, Shuffling, Scrambling, and applying Substitution boxes on the data. Our experimental results will show that *GS3* introduces a minimal overhead, of just two bytes corresponding to the cyclic redundant control 16 (CRC16) integrity control in the length of the messages concerning the original data. According to the execution time with respect of other encryption–based methods, even a 50% less than *Chacha20* algorithm, as fewer steps and simpler computation procedures are required.Therefore, *GS3* is a good choice to be used in resource-constrained *IoT* devices in which data integrity and security are required, taking into account the data freshness.**

***Index Terms*—Chaotic system, integrity, Internet of Things (IoT), scrambling, security, shuffling, wireless sensor networks (WSNs).**

I. Introduction

NOWADAYS, the evolution and growing up of the Internet of Things (IoT) is allowing those smart and small devices with reduced resources to share, send data, and be connected to the Internet. However, these advantages may conduce to security gaps. So, on the one hand, it is necessary to develop lightweight cryptographic methods to protect private information and guarantee the integrity of the exchanged information on the network. On the other hand, it is a challenge the selection of the proper lightweight cryptographic algorithm [1]. As a consequence, there is a tradeoff between implementation performance and security [2]. The security and performance of *IoT* will be a balancing act [3].

Examples of devices where lightweight cryptography should be considered are the following: radio frequency identification (RFID), wireless sensor networks (WSNs), and industrial control equipment, among many other fields. WSN [4] are networks which are created ad-hoc. They are composed of hundreds of sensors, interconnected by lightweight wireless protocols, such as *Zigbee*, *IEEE 802.15.4* [5], or others, to sense the environment. The collected data are sent to base–stations from where data could be sent to the Internet through a gateway using a lightweight protocol, such as *LoRa* [6]. Afterward, data can be processed and analyzed by computers with more resources using the cloud computing paradigm. Finally, *IoT* devices, such as smartphones or tablets, can access and request this information. Fig. 1 represents this scheme of data transmission and the interconnection between WSNs and *IoT* world.

Nevertheless, this technology is exposed to threats. Lata et al. [7] presented a comprehensive analysis of security threats against WSN and *IoT*. Also, they describe strategies for preventing, detecting, and mitigating those threats. Ahmad et al. [8] carried out a case use of underwater WSN (UWSN). They analyze the threats that UWSN are exposed to and some of the countermeasures for those threats. Wand and Chen [9] researched on data security immunity in WSN based on trust management, clustered communication, and *LEACH* protocol. Other recent cases of use are: Rezaeibagha et al. [10] presented an efficient and provably secure scheme, which is a novel cryptographic accumulator based on their novel authenticated additive homomorphic encryption which can collect and accumulate data from *IoT* wireless wearable devices. According to Manjula et al. [11] a major limitation to the use of WSNs in asset monitoring applications is the privacy of source location information. For this reason, they develop two phantom routing-based solutions to provide source location privacy for the case of multisource/asset scenario, which is a case that has received very little attention in the literature. Another problem presented in WSN and *IoT* is related to processing delay and energy efficiency. Peng et al. [12] proposed a multifunctional and multidimensional secure data aggregation (DA) scheme to strike the balance between data availability and privacy.

The number of connected *IoT* devices is estimated to be about 125 billion by 2030 year. Thus, data security is a

Manuscript received 25 October 2023; revised 11 January 2024 and 24 April 2024; accepted 27 April 2024. Date of publication 30 April 2024; date of current version 25 July 2024. This work was supported in part by the Spanish Ministry of Science, Innovation, and Universities Grant Intelligent Distributed Processing Architectures in Fog Level for the IoT Paradigm (Smart-Fog) under Grant RTI2018-098371-B-I00, and in part by the FoCRA Project (FEDER Andalucia 2014–2020) under Grant 1380974-F. *(Corresponding author: Jose M. Palomares.)*

The authors are with the Department of Electronic and Computer Engineering, Universidad de Córdoba, 14071 Córdoba, Spain (e-mail: francisco.alcaraz.velasco@juntadeandalucia.es; jmpalomares@uco.es; olivares@uco.es).

Digital Object Identifier 10.1109/JIOT.2024.3395543

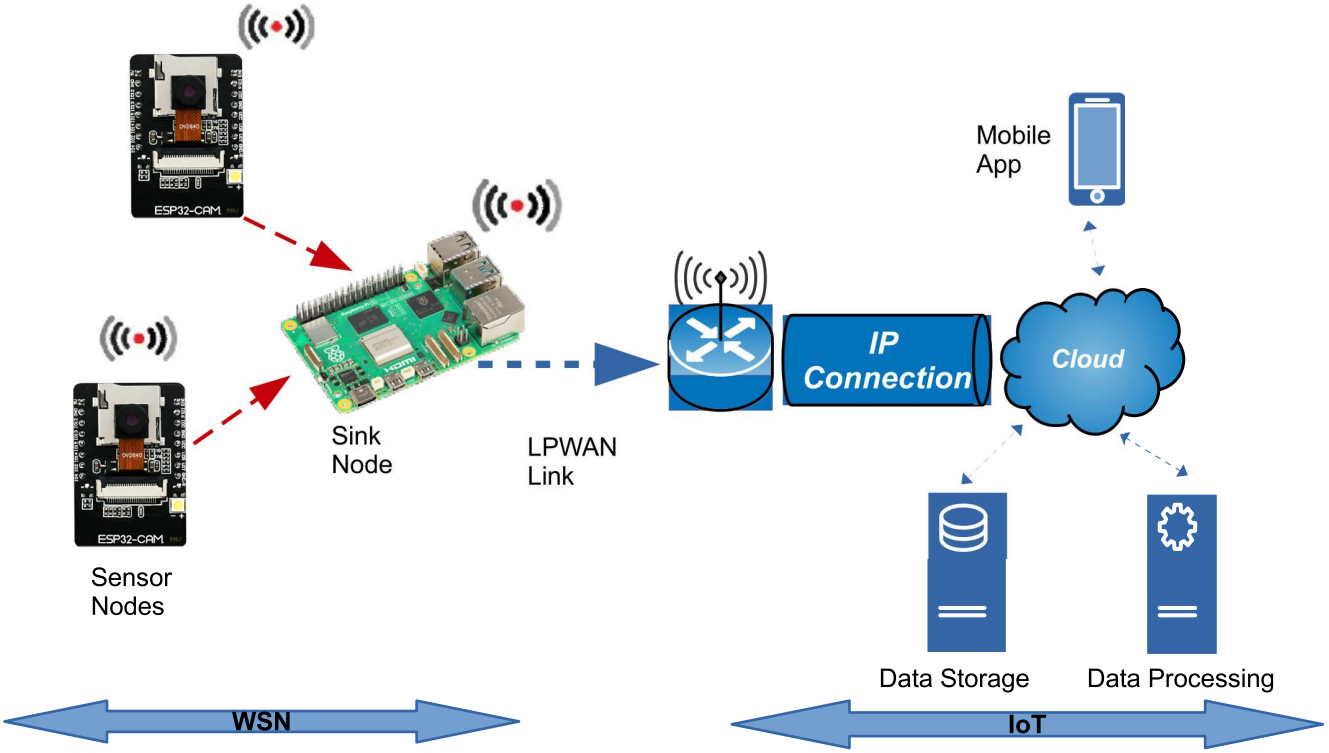

Fig. 1. General scheme of communication between sensors and cloud.

required feature With such a big scope along with related threats and security issues. Besides, these devices usually send sensitive data.

Thus, security solutions are necessary to protect private data. Although, there are security mechanisms focused on (WSN and IoT) devices, not all of these solutions are suitable for them due to the computational overhead introduced [13]. For example, classical cipher algorithms, such as the advanced encryption standard (AES) [14], which is included as a reference in the *IEEE 802.15.4* standard [15], the *PRESENT* cipher [16], or even newer ciphers, such as *Salsa20* [17] and *Chacha20* [18], have very demanding memory and computational requirements that cannot be assured by most IoT devices. Therefore, it is necessary to research developing lightweight secure protocols and mechanisms. Our working environment is a communication model where the data are sensed and sent to a base station, where a decision is made using the received data. After that, those data are discarded within a small time window and have no further utility. For example, IoT applications to detect thieves or sound alarms and face recognition of people to prevent situations where armed people appear. All of these events only have a brief period of validity. Encryption techniques and cipher algorithms provide extremely high privacy at the expense of large computing consumption. A tradeoff between security and computing power must be found. Therefore, a lightweight security protocol that assures the integrity, confidentiality, and authenticity of data, is interesting.

In this work, we propose the GS3 method. This new method is an evolution from our earlier research [19] on the same subject because of the following three features: 1) a pseudo-random shuffle method using a chaotic system as pseudo-random number generator enhances the generation of the data block proposal described in [19]; 2) a scrambling method to achieve the confusion rule is included; and 3) finally, it is added a third confusion step through *S-boxes* substitutions. All of these procedures do not require any special hardware or functional units, as most operations are memory movement–related and basic mathematical operations. Therefore, *GS3* is very suitable in resource–constrained *IoT* devices.

Thus, our main contribution is a lightweight security method, which does not use encryption as the global information obfuscation method. Our proposal uses some computational operations which are much lighter than the encryption, in terms of processing time.

This article is presented as follows: we first present an introduction about these issues in Section I. Section II presents a brief vision of the lightweight block and stream ciphers along with their operation modes. In Section III, the proposed *GS3* method is described. Then, Section IV shows the hardware and software platform used in this project. Our approach is evaluated through several statistical tests, described in Section V. In Section VI, the results of our proposal are shown together with the algorithms reviewed previously in foundations. Section VII includes a comparative analysis of the results. Finally, Section VIII concludes this scientific work.

II. Foundations

This section presents the cipher algorithms that have been reviewed to develop our proposal. Fig. 2 represents a general

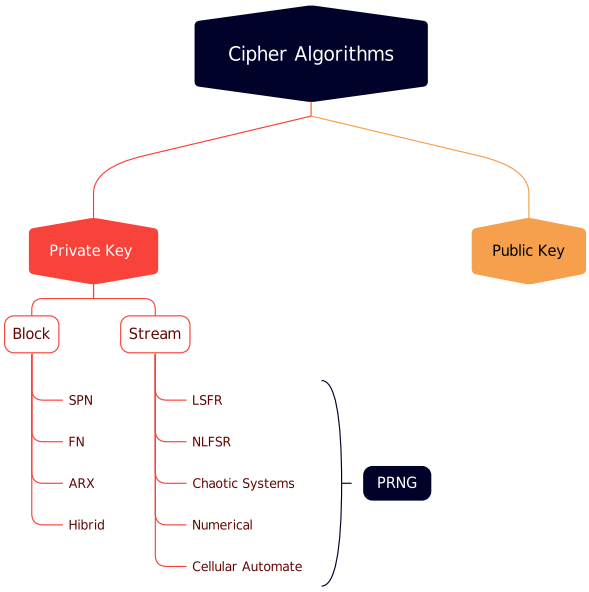

Fig. 2. Categories of cipher algorithms.

classification of these cipher algorithms. On the one hand, in public key ciphers, each entity has a pair of keys (one public and the other public). This scheme provides confidentiality, no-repudio, integrity, and authenticity. The keys are in the order of thousands of bits (1024 bits or more), providing a high-security level. However, the computational costs are also very high. Therefore, it makes this scheme less suitable for sensor devices. Nevertheless, some research based on electric curves cryptography (ECC), [20] or more recently based on *Quantum Cryptography* [21], have reduced the computational costs, but require highly specialized hardware.

On the other hand, private key ciphers ensure confidentiality, integrity, and authenticity. The sender and the receiver share a key, which is in the range between 80 and 256 bits. These two issues are the main drawbacks of private key cryptography. However, the main advantage is that the cipher is much faster and requires fewer computational resources than the public key ones. For these reasons, our proposed method is based on private key or symmetric cryptography [1].

Besides, symmetric cryptography can be classified into two types of operations: 1) block and 2) stream cipher, respectively. In the first one, plain data are divided into blocks of the same size (64 bits or more) and processed using some of these structures: substitution–permutation network (SPN), feistel network (FN), add rotate XOR (ARX) operations, or a hybrid structure.

In a stream cipher, the plain text is ciphered bit–by–bit or byte–by–byte with an encryption sequence, which should have a random-like appearance. The main issue in the stream cipher is how the encryption sequence is generated. There are some methods below the pseudo random number generators (PRGNs) paradigm, such as linear feedback shift register (LFSR) or no linear (NLFSR), *Chaotic Systems*, *Numerical Methods*, or cellular automata (CA). The final result is obtained by a XOR operation between the plain text and the encryption sequence. This function is chosen because is balanced, so there is exactly a 50% chance for any value of the input bits (0 or 1) to be changed. Besides, it is an invertible function, so that the decryption process can be carried out.

A. Lightweight Stream Ciphers

The security provided by a cryptographic algorithm is directly related to the randomness of the generated output. A *PRNG* algorithm should produce an indistinguishable sequence of bits, given an initial seed. Therefore, there must not be any algorithm executable in polynomial time, which can decide if the given sequence was random or calculated. There are some empirical tests to assess the randomness of a given sequence. Some examples of randomness tests are Solomon [22] or the 800-22-rev1 statistical suite test [23] published by the national institute of standards and technology (NIST).

1) Feedback Shift Register Sequences: One of the mechanisms to generate a key stream is LFSR. Regarding the required hardware, it is composed of a clock storage element (such as a flip-flop) with 0 or 1 value and a feedback path that connects the output to certain flip-flops and to the input through an XOR function. LFSRs are often specified by polynomials, which should be primitives to get the maximum period. However, its linearity is a security drawback. For example, a known plain-text attack based on *Berlekamp–Massey's* algorithm [24] allows to obtain the polynomial.

To reduce the linearity, several LFSRs are combined to get a non-LFSR. The *A5/1* stream cipher uses three LFSR. This cipher is used in global system for mobile communication (GSM) phones for secure communication. However, this cipher can be attacked with several different methods [25]. We have tested this algorithm based on a Python implementation [26].

Another algorithm based on three LFSRs is *Geffe* CITA. Also, we have tested this algorithm based on the implementation of [26].

Although the schemes based on LFSRs are not new, proposed methods recently use LFSR. In [27], the output of a LFSR is XOR with the output of a chaotic system. *Trivium* [28] is a synchronous stream cipher based on 288 circular flips–flops, the length of each feedback shift register (FSR) is 93, 84 and 111 shift registers. This algorithm has more hardware than software profile and it was a finalist in phase 3 of the project *the ECRYPT Stream cipher Project* [29]. We test this cipher based on the released source code [30].

2) Cellular Automata: CA are mathematical models for a dynamic system that progresses in discrete steps. Cryptography is an application field of the CAs. Wolfram [31] was the first to propose CAs to generate *PRNG*. Poornima et al. [32] presented a survey of CA. A CA consists of an array of cells each of which can be in one of a finite number of possible states that are updated synchronously according to a rule. A simple example of CA is known as rule 30, 1-D and with a *radio* $= 1$ of the surrounding (adjacent) cells. In this case, there will be $2^8 = 256$ rules, but some of them show

chaotic behavior. The evolution of the rule 30 is given by (1), where $s_i(t)$ is the state of cell i at time t

$$s_i(t+i) = s_{(i-1)}(t) \oplus \big(s_i(t) + s_{(i+1)}(t)\big). \tag{1}$$

An improved CA, named *Pentavium* [33], is proposed. It is a 1-D CA-based which makes use of the strength of greater radii CA. In this case, a 5–neighborhood is used to improve the cryptographic properties of the *Trivium* stream cipher. The diffusion and randomness of the cipher are also increased but at the cost of increased computational complexity. There are 2^{32} possible rules. *Pentavium* uses the rule set (1452976485, 1721342310, 2523490710, 1520018790, 1721342310, 1452976485, 1520018790, 2523490710), some of which are linear, and others nonlinear rules which have shown good cryptographic properties.

3) Chaotic Systems: The analysis of the dynamical properties of the chaotic systems through tools, such as the bifurcation diagrams and *Lyapunov's* exponents, is considered an essential security analysis, allowing the designer to discover regions of the parameters of design where the chaotic system shows windows of periodic-like behavior. Therefore, initial conditions of the chaotic system must avoid values where the system should conduct in a periodic-like behavior because it would reduce the randomness of the output, which is the most crucial issue of a *PRNG*.

An example of a 1-D chaotic system is the logistic map (LM). The discrete mathematical model of the LM system is defined by

$$x_{(n+1)} = bx_n(1 - x_n), b \in (0, 4] \tag{2}$$

$$x_{(n+1)} = f(x_n). \tag{3}$$

The general expression of a chaotic system is defined by (3). The bifurcation diagram is shown in Fig. 3 and *Lyapunov's* exponent in Fig. 4. Three relevant intervals can be observed in Fig. 3, independently of the initial value x_n.

1) *Stable Interval:* When $b = [0, 1)$, $f(x_n)$ values converge quickly to 0. If $b = (1, 3]$, $f(x_n)$ values tend to a stable value, which is $b - 1/b$. Therefore, observing Fig. 4, *Lyapunov's* exponent takes negative values because $f(x_n)$ converges to a value.
2) *Periodic Interval:* As b increments the value, the system's behavior is more chaotic. If $b = 3$, a first bifurcation emerges with period 2^1, therefore $f(x_n)$ generates two different values alternatively. Around $b = 3.449$, it appears another bifurcation with period 2^2. In these intervals, *Lyapunov's* exponent is zero or negative.
3) *Chaotic Interval:* This region appears when $b > 3.569$. $f(x_n)$ generates aperiodic values, so a chaotic behavior is shown. Therefore, *Lyapunov's* exponent takes values higher than zero. Nevertheless, periodic windows can appear in this interval, for example, when $b = 3.8339$ or $b = 3.4704$.

Lee et al. [34] studied that there is a correlation between positive sign *Lyapunov's* exponents (λ) and good chaotic random number sequences. This exponent provides information about how the chaotic system is sensitive to initial conditions. So, for two similar (but not equal) initial conditions, the output of (3) should be different. Therefore, there is a divergence between close initial conditions, x_0. This divergence is defined by

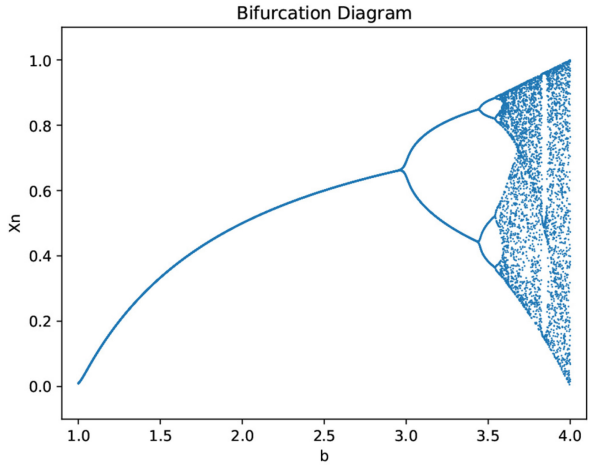

Fig. 3. Bifurcation diagram.

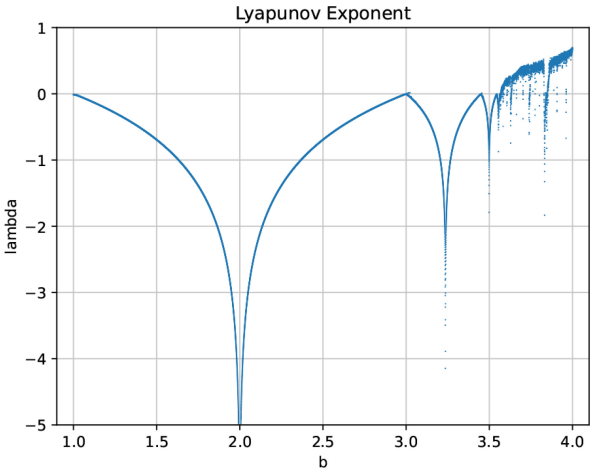

Fig. 4. Lyapunov's exponent.

$$d_{(n)} = |f(x_{n+\epsilon}) - f(x_n)|. \tag{4}$$

It is expected that this divergence is large, although $\epsilon \to 0$. Therefore, (4) can be expressed by

$$d_{(n)}/\epsilon = |f(x_{n+\epsilon}) - f(x_n)|/\epsilon \tag{5}$$

where (5), can be approximated by

$$e^{\lambda n}. \tag{6}$$

Finally, (7) is used to calculate λ value when $\epsilon \to 0$ and $n \to \infty$. It is the average of the logarithm of the derivatives of $f(x_n)$

$$\lambda = \lim_{n\to\infty} 1/n \sum_{i=0}^{n} \ln|f'(x_i)|. \tag{7}$$

In Fig. 4, λ evolution with respect *Lyapunov's Exponent* is shown. If $f(x_n)$ is evaluated with two close values and $f(x_n)$ generates different paths or orbits, then $|f'(x_n)| > 1$ and therefore $\ln|f'(x_n)| > 0$, so we can conclude that λ in

Fig. 4 is positive and the system presents a chaotic behavior. In both Figs. 3 and 4, when control parameter $b > 3.5$ in $f(x_n)$, pseudo-random outputs are generated.

Some related research on chaos-systems has been published recently. Dridi et al. [27] proposed an encryption/decryption procedure operating in cipher block chaining (CBC) mode. The system is produced by a pseudo-random key stream generated by a *PRGN* based on a chaotic system. First, in the confusion phase, *S-boxes* are generated by combining four 1–D chaotic systems, *(PWLCM, skew tent, logistic, and 3-D Chebyshev maps)* and LFSR, which are weakly coupled by a predefined coupling matrix. Second, in the diffusion phase, a three steps process is carried out. This second phase consists of a permutation layer based on the modified *2–D–Cat* map and a horizontal–vertical addition diffusion.

Alshammari et al. [35] modified the well–known *AES* with a new *S-box* generated by the *Lorenz* chaotic map [36]. Its software implementation is done on *TelosB* hardware.

Zhu et al. [37] proposed a cipher schema based on a combined chaotic system between logistic–map and tent–map to generate two chaotic sequences, which are used in the confusion and diffusion steps, respectively. Combining both types of maps (Logistic and Tent), *Lyapunov's Exponent* and *bifurcation diagram* improve.

4) Numerical PRNG: In this section, stream ciphers based on different techniques than the ones from Sections II-A1, II-A2, or II-A3 are shown.

The first of them is the well–known Rivest cipher (RC4) or *(ARC4)* cipher [38]. It was designed in 1987 by *Ronald Linn Rivest*. The use of RC4 in thetransport layer security (TLS) was prohibited in February 2015. This algorithm is initialized with a key in the range $1 <= key < 256$ bytes. It works to the byte level in each cycle. *RC4* has two subproccesses called Key-scheduling algorithm (KSA) and Pseudo-random generation algorithm (PRGA). It is a simple and fast cipher.

Several improvements of *RC4* have been published. The modified ARC4 (MARC) is one of them [39]. It enhances the security of *RC4* by modifying its key scheduling algorithm (KSA) and improves the performance by modifying the *PGRA* process, because four bytes of keystream are generated in each cycle.

In [40], a combination of elliptic curve cryptography (ECC), *RC4*, and *SHA–256 hash* is presented to protect sensitive data of IoT–based irrigation systems. The *ECC* is used to encrypt the key of the *KSA* subprocess and the result of the *PRGA* subprocess.

The *Salsa20* [17] algorithm was proposed by Daniel J. Bernstein and presented in the *eSTREAM* project in 2005. *Salsa20* has 512-bits internal state, which is initialized with 8-bytes arbitrary numbers, 16-bytes constants, 8-bytes counters, and 32-bytes keys. Then the algorithm generates 512-bits key–stream in each cycle, iterating 20 times the quarter–round (QR) function, which only uses the *Add*–XOR–*Rotate* operations.

The *Chacha20* [18] cipher is similar to *Salsa20*. It was proposed by Daniel J. Bernstein in 2008. *Chacha20* modifies each word twice in each QR while the *Salsa20* modifies only once per QR. Besides, the QR function operates with the columns and diagonals of the internal state; whilst, *Salsa20* operates by rows and columns.

B. Lightweight Block Ciphers

This section presents the lightweight block ciphers that have been revised and classified according to every operation mode in Fig. 2.

1) Substitution–Permutation Networks: The *AES* is one of the most well–known and important encryption algorithms in use due to its high–level security and reduced execution time. The *NIST* announced the *Rijndael* algorithm [14] in the 2000 year as the winner among the finalists. *AES* divides messages into 16-bytes size blocks with keys of 128, 192, or 256-bits sizes. Moreover, *AES* works with 2^8 *Galois Field* polynomials.

Another cipher based on *SPN* is *PRESENT* [16]. It was developed in 2007. It has a block size of 64 bits, a key length of 64 or 128 bits, and 31 rounds. It was developed having in mind devices with reduced computational resources. In each round, the internal state (64 bits) is XORed with the key, then a confusion step is carried out with a defined *S-box* $S : F_2^4 \rightarrow F_2^4$. Finally, a permutation layer moves the *i*-bit of the state according to a permutation table. Its source code is very compact, about the 50% smaller than *AES* cipher

2) Feistel Networks: The data encryption standard (DES) [41] was published as a standard in 1977 and developed by the international business machines (IBM) Corporation. In 1998, it was broken in 56 h. *DES* is the reference cipher operating in *FN* balanced mode, with blocks with 64-bit size, and keys of 56 bits during 16 rounds. Its strength resides in a confusion step with 8 *S-Boxes*. To break these *S-Boxes* by brute force attack, it would require 2^{256} operations. However, to break the key it would be necessary only 2^{56} operations. *New Light-Weight Crypto Algorithms for RFID (DESL)* [42] cipher has been proposed. It is a reduced version of *DES* using only one *S-box*. It tries to reduce the execution time and the amount of gate equivalents, GE.

Simon [43] is also based on balanced *FN* with *n-bits* words and $2 \cdot n$-bit block size. It can operate with different combinations of blocks sizes ([32–128] bits), key sizes ([64–256] bits), and rounds ([32–72]) to get more flexibility taking in mind constrained devices, where the simplicity of design is important. Its design was optimized for performance in hardware implementations. It was developed by the national security agency (NSA) in 2013.

3) Add–Rotate–XOR Operations: Speck [43] is a software–orientated cipher and it is based on *ARX* operations. *Speck* was released by the *NSA* in June 2013. It can work with different combinations of block size ([32–128] bits), key size ([64–256] bits), and number of rounds ([32–72]). These combinations provide a variety of implementations in order to get flexibility. A block is formed with two words and in each round the key is expanded by rotating the left word to the right, adding the right word to the left word, XORing the key into the left word, rotating the right word to the left, and finally, XORing the left word into the right word.

4) S-Boxes: A Substitution–box, *S-box*, is a component of the symmetric key algorithms that changes the values of

TABLE I
DATA BLOCK

Message-1
Message-2
...
Message-n

the data with substitutions. Its objective is to obfuscate the relationship between the key and the ciphertext, thus providing Shannon's property [44] of confusion. *S-box* must pass several tests, such as bijectivity, nonlinearity, strict avalanche criteria, and equiprobable input/output XOR distribution, to resist linear and differential cryptanalysis. Farah et al. [45] presented a mechanism based on two chaotic maps and Teaching–Learning-Based-Optimization, *TLBO*. The results of the performance test show that their method presents good cryptography properties and can resist several different attacks. The mechanism proposed by Artuğer and Özkaynak [46] defined a technique to provide *S-box* with the best performance criteria for all chaotic system classes. Islam and Liu [47] used a 4D 4-wing hyperchaotic system to construct S-boxes, which is proven to have good cryptographic strength. These S-boxes constructions are an alternative to the classical algebraic techniques.

III. METHODOLOGY

In this section, the proposed method is represented in Fig. 5. Section III-A process is called *Generating Data Block*, where the messages are transformed in data block structures. In Section III-B phase, every data block is shuffled, therefore the initial positions bytes are modified. Then, in the *Internal State*, shown in Section III-C phase, data are scrambled with a pseudo-random sequence of bits. Section III-D phase, which is the last one, is called *Applying S-boxes*, where our method introduces a nonlinear substitution mapping function.

In all these phases, we take into account the tradeoff between the provided security level and the consumption of the computational sources. Since the main parts are Generating the data block, Shuffling, Scrambling, and applying Substitution boxes, we propose GS3 as the name for this method.

A. Generating the Data Block

As input, each message can store several different environment variables r, which are divided into n data chunks c with equal byte size. Equation (8) defines it mathematically

$$M = (m_{11}, \ldots, m_{1c}, \ldots, m_{r1}, \ldots, m_{rc}). \tag{8}$$

The logical organization of these data messages it is shown in Table I. We name this organization b data block.

This b block is defined in (9), as a M matrix with $(n \times m)$ values, which includes $(n-1) \times (m-1)$ data of b block. The row $(n-1)$ and column $(m-1)$ include frame control sequence (FCS) values of $(b-1)$ data block to reduce the internal correlation between data and *FCS* values

$$M^b_{n,m} = \begin{pmatrix} M^b_{1,1} & \cdots & M^b_{1,m} \\ \vdots & \ddots & \vdots \\ M^b_{n,1} & \cdots & M^b_{n,m} \end{pmatrix} \tag{9}$$

TABLE II
SHUFFLED DATA BLOCK

FCS-1-Sw	Msg 2-m	..	FCS-b-Sw	Msg n-a
Msg n-m	FCS-a-Sw	..	Msg 1-b	Msg n-b
..	..	..	..	..
Msg 2-b	Msg 2-a	..	Msg 1-m	FCS-mn-Sw
Msg 1-a	FCS-n-Sw	..	FCS-2-Sw	FCS-n-Sw

$$m^b_{i,m} = CRC16^{b-1}_{h_i} \quad \forall i = 1 \ldots n-1 \tag{10}$$

$$m^b_{n,j} = CRC16^{b-1}_{v_j} \quad \forall i = 1 \ldots m-1 \tag{11}$$

$$m^b_{n,m} = CRC16^{b-1}_{hv}. \tag{12}$$

Our method includes an integrity data control for each row and column, Frame Check Sequence *FCS* is calculated. Specifically, the *Cyclic Redundant Code* is computed, in (11) and (12). It allows the detection of errors in data transmission and attacks, such as tampering or forgery of data. These issues have already been studied in deep [19].

B. Shuffle Algorithm

The objective of this phase is to get Shannon's property [44] of diffusion. It is obtained using a permutation S-box (P-Box), to swap the data position. At the end of this phase, a new data block is produced. Table II shows a shuffled data block, where the highest amount of data chunks is expected to change from their initial original positions. This issue will be examined in Section V-C.

Table II is defined as a S matrix in

$$S^b_{n,m} = \begin{pmatrix} S^b_{1,1} & \cdots & S^b_{1,m} \\ \vdots & \ddots & \vdots \\ S^b_{n,1} & \cdots & S^b_{n,m} \end{pmatrix}. \tag{13}$$

Each element of the S^b_{ij} matrix is calculated by applying the

$$S^b_{n,m} = \sum_{i=1}^{n} \sum_{j=1}^{m} \left(m^b_{(i,j)} \cdot \Pi_{(i,j)} \right). \tag{14}$$

These Π permutation matrices have the following properties: a unique 1 for each (i, j) position. Besides the sum of these permutation matrices is the matrix with all elements equal to 1 value. Equation (15) defines Π matrices

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \Pi(i,j) = \begin{pmatrix} 1 & \ldots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \ldots & 1 \end{pmatrix}. \tag{15}$$

Nevertheless, the product of matrices is a mathematical operation with computational requirements. Therefore, we propose a lightweight shuffle method represented in Fig. 6, which has the following steps.

1) Sender and receiver must share private seed values. The value of the b parameter in (2) must avoid ranges of values where the LM shows periodic behavior.
2) The data block is transformed into a data array.
3) Our shuffle method uses a chaotic system to calculate two pseudo-random values. More specifically, we have selected the known discrete LM because it is simple,

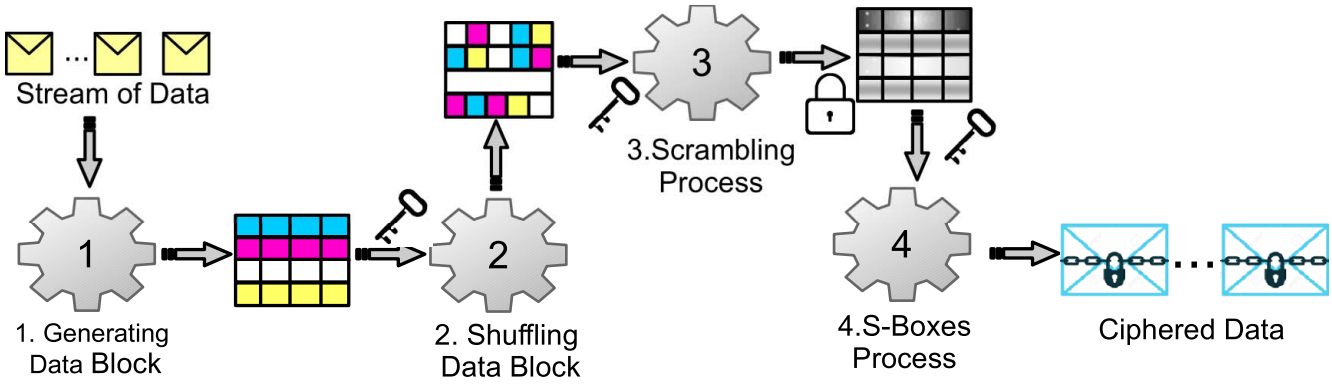

Fig. 5. Proposed method.

requires low-computational resources, and has a large enough period. These pseudo-random values are used to find out three points of cutting the data array. Therefore, four subarrays are obtained P_1, P_2, P_3, P_4.

4) The previous subarrays are swapped with four possible sequences, (P_4, P_1, P_2, P_3), (P_2, P_1, P_4, P_3), (P_3, P_1, P_4, P_2) and (P_4, P_3, P_2, P_1). The sequences are selected according to a pseudo-random number that the chaotic LM generates. It is represented in the fourth step in Fig. 6.
5) In step 5, the data block is read diagonally. There are four options for reading, which are chosen depending on the sequence number of the data block and the generated pseudo-random value provided by the chaotic system. Therefore, the way of reading is pseudo-random. This issue is shown in Fig. 6 with the key symbol.
6) After reading the data block diagonally, it is created a new data array.
7) Finally, the data array obtained from the previous step is converted into a data block structure as the original to get the shuffled data block. It will be used as the input for the "*3. Scrambling Process*" step described in Fig. 5.
8) The steps 1–7 may be repeated to achieve higher obfuscation levels. This issue is evaluated in Section V-C.
9) Values of the initial conditions in (2) are updated between each data block sent.

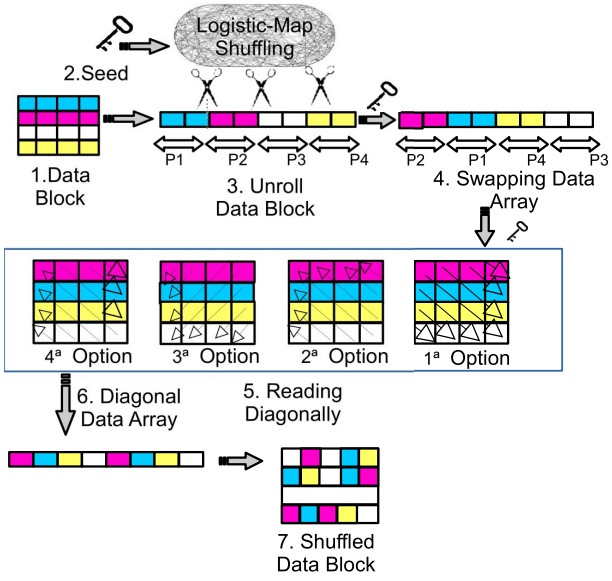

Fig. 6. Lightweight shuffling method.

C. Scrambling Process

This section describes the proposed method to scramble the previously shuffled data block. Fig. 7 represents the general scheme. It has the following steps.

1) The internal state IS generates a 288-bit pseudo-random value in each iteration. This IS has the following input parameters: pseudo-random values generated by the chaotic system, constants, and keys. The value of the b parameter in (2) must avoid ranges of values where the LM shows periodic behavior. A complete description of this IS is shown in Fig. 8. The initial state is organized as follows.
   a) Each X_i element of the IS has a size of 4 bytes.
   b) In the initial state I_0, several bytes are reserved: 16 bytes for the four key values, (K_1, K_2, K_3, K_4), 8 bytes for storing two constants (C_1, C_2), and 12 bytes to store three pseudo-random values (S_1, S_2, S_3) generated by a chaotic system.
2) The shuffled data block is subdivided into 288-bit chunks, which are scrambled using XOR operation with a 288-bit pseudo-random value generated by the *Internal State*.
3) The scrambled bits are stored in a data array.
4) The 288-bit pseudo-random value of the *Internal State* is generated with the followings operations:

$$\begin{aligned} Y_{i3} &= X_{i3} \otimes (X_{i1} + X_{i2}) \\ Y_{i2} &= X_{i2} \otimes (Y_{i3} + X_{i1}) \\ Y_{i1} &= X_{i1} \otimes (Y_{i2} + Y_{i3}). \end{aligned} \tag{16}$$

After generating the 288-bit pseudo-random value and scrambling 288 bits of the data block, the (Y_{11}, Y_{22}, Y_{33}) elements are updated with pseudo-random values generated by a chaotic system according to the following operations:

$$Y_{11} = Y_{11} + \text{ChaoticMap}$$

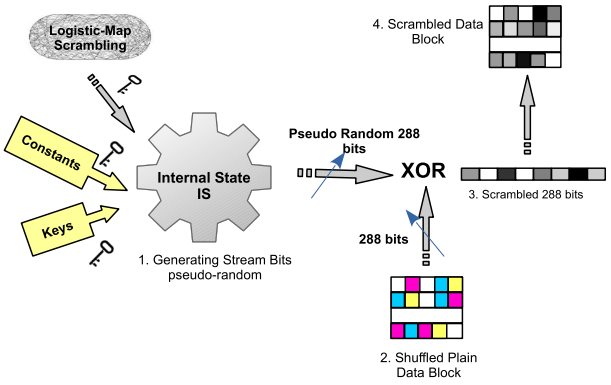

Fig. 7. Internal state.

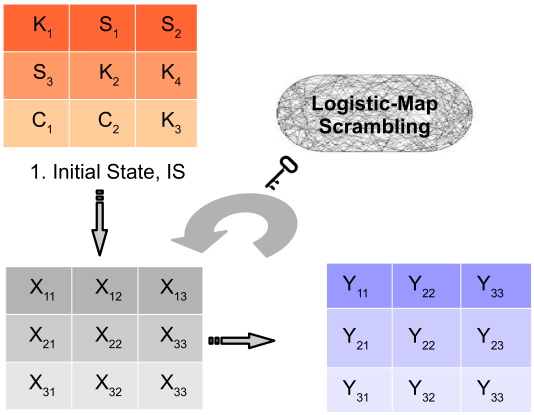

Fig. 8. Generating the internal state.

$$Y_{22} = Y_{22} + \text{ChaoticMap}$$
$$Y_{33} = Y_{33} + \text{ChaoticMap}. \quad (17)$$

With the operations described by (17), we are, including the concept of *key expansion*, which is common in block ciphers to avoid repeating the same key in each round.

5) The steps 2–4 are iterated until the data block is completely scrambled.
6) Finally, the obtained scrambled array is converted into a scrambled data block, which is the input to the *(S-boxes Process)*, as it is shown in Fig. 5.
7) Values of the initial conditions in (2) are updated between each data block sent.

D. Applying the S-Boxes

To increase the security level with a nonlinear transformation and get Shannon's property [44] of confusion, the scrambled data block is obtained in the *Scrambling Process* described in Fig. 5. In this section, it is described how the proposed method uses the *S-Boxes* substitutions. The proposed mechanism takes advantage of the *S-boxes* designed by Farah et al. [45], Artuğer and Özkaynak [46], and Islam and Liu [47]. Fig. 9 represents the *S-boxing Process*, which has the following steps.

1) The process has the following input parameters: pseudo-random values generated by the chaotic system, an array of S-Boxes (four S-Boxes), and the scrambled data block obtained in the *(Scrambling Process)* step, as it is shown in Fig. 9. The value of the b parameter in (2) must avoid ranges of values where the LM shows periodic behavior. Each $S-Box_i$ is a data array with a size of 256 bytes. In (18), the first second values and the last second ones of the *S-Box* designed by Farah et al. [45] are shown

$$\begin{bmatrix} 0 & 1 & \cdots & 254 & 255 \\ 150 & 189 & \cdots & 39 & 206 \end{bmatrix}. \quad (18)$$

2) Every 288 bits of the scrambled data block are substituted by the values of a $S-Box_i$. The specific *S-Box* is selected depending on the pseudo-random values generated by the chaotic system.
3) Finally, once the $S-Box_i$ is selected, each byte of the value of the data block ($value_i$) is used to index inside of the $S-Box_i$, then this value is changed by the input of $S-Box_i[value_i]$.
4) Values of the initial conditions in (2) are updated between each data block sent.

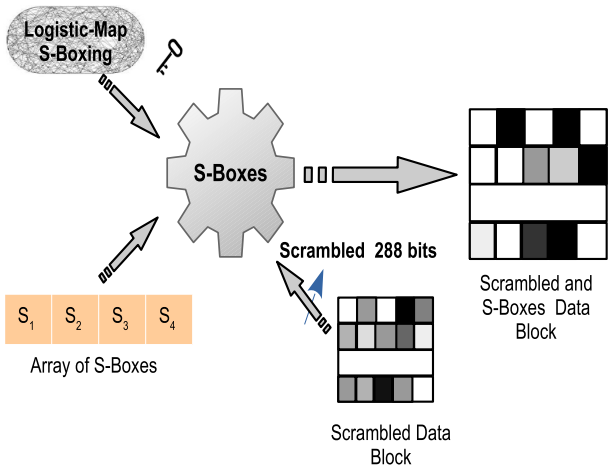

Fig. 9. S-boxes process.

E. Decrypt Process

In this section, the decrypt process is described. Fig. 10 depicts this process. It has the following steps.

1) Sender and receiver must share private seed values. The value of the b parameter in (2) must avoid ranges of values where the LM shows periodic behavior.
2) First, the receiver transforms the ciphered data stream into a ciphered data block.
3) Second, the *Inverse-SBoxes Process* is executed. The receiver selects the specific $S-Box_i$ depending on the pseudo-random values generated by the chaotic system. Then, each byte of the value of the ciphered data block ($value_i$) is used to index inside of the $S-Box_i$. This value is changed by the input of $S-Box_i[value_i]$. In (19) it is shown the inverse *S-Box* of (18). For example, the 39 value should be substituted by the 254 value

$$\begin{bmatrix} .. & 39 & .. & 150 & .. & 189 & .. & 206 & .. \\ .. & 254 & .. & 0 & .. & 1 & .. & 255 & .. \end{bmatrix}. \quad (19)$$

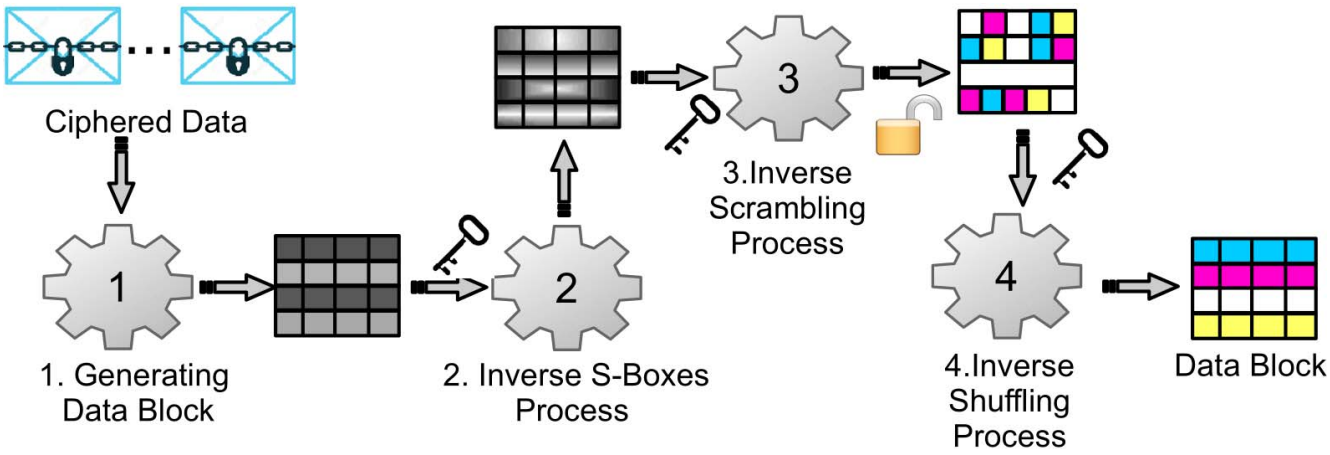

Fig. 10. Decrypt process.

4) In the third step, the *Inverse Scrambling Process* is run. The *Interval State* must be iterated according to (16) and (17). After that, each 288-bit is decrypted with the invertible XOR operation.
5) *Inverse Shuffling:* In Fig. 11 it is shown this process. First, the chaotic *Logistic-Map* is run to find out the four options for diagonally reading the data block and the points of cutting the data array. These pseudo-random values are stored in a *Pseudo-Random Array*. We highlight that this array is accessed in reverse order by the *3. Reading Diagonally*, *4. Unroll Shuffled Data Block*, and *5. Swapping Data Array* subprocesses. These last subprocesses are computed according to the stabilized number of the shuffled iterations. After that, the plain data block is obtained.
6) *Verify Integrity Data:* the last step verifies the integrity data using the FCS by rows and columns.
7) Values of the initial conditions in (2) are updated between each data block received.

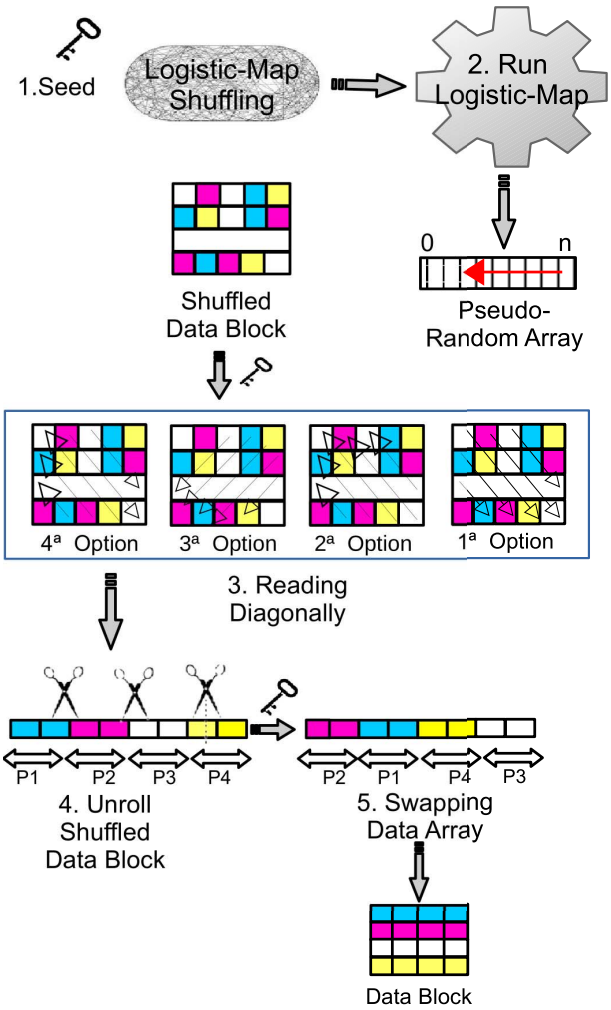

Fig. 11. Lightweight inverse shuffling method.

F. Performance Parameters

In order to test the performance and the security of the proposal, some experiments will be carried out using images as input data. Images are highly suitable for the proposal, as they have high spatial and value correlation. Thus, partitioning, shuffling, and scrambling the pixel values provide a good experimental setup to test the security of the proposal. Besides, it allows a quick visual review of the results. In particular, the image used for the test was the well-known *Lake* [48] image in its grayscale version. Three sizes of image $(64^2, 128^2, 256^2)$ bytes with 8-bit resolution were used. We named these images as *Lake_64, Lake_128, Lake_256*.

The performance parameters and security results selected to assess and compare the proposed method with the reviewed ciphers in Section II, are the following.

1) *Visual Analysis:* The original image is compared after shuffling and scrambling, using some well-known and synthetic images for the comparison.
2) *Histogram:* The histogram of the scrambled image is computed. The ideal result should be a uniform histogram.
3) *Shuffle Process:* The proposed shuffling method is analyzed to get a tradeoff between the number of iterations and the number of swapped bytes.

TABLE III
SOME OF THE TYPICAL SENSOR PLATFORMS

Platform	CPU	Memory
Raspberry Pi 4 Model-B	ARM 1.8 GHz	1-8 GB RAM
PyCom Fipy	Xtensa 32–bit, 600 DMIPS	4 MB RAM
TelosB	8 MHz	10 KB RAM
Arduino Uno	ATmega328P 16 MHz	2KB SRAM, 32KB FLASH
MicaZ	ATmega128L	128 KB

4) *Sensitive Analysis:* The proposed method should be sensitive to light changes in both bits of plain data and key. To detect this behavior, two parameters are calculated number of pixel change rate (NPCR), unified average changing intensity (UACI).
5) *Entropy:* The entropy indicator is calculated to measure the randomness.
6) χ^2 *Value:* This statistic is calculated to test the uniformity of the ciphered data.
7) *Randomness Tests:* To check the randomness of the internal state, the *800-22-rev1* and *Ent* tests are used.
8) *Execution Time:* The main parameter to test and compare our method with other ciphers is the execution time. It is the mean time of 100 executions for each cipher.

IV. IMPLEMENTATION

In this section, the common hardware and software of sensor platforms are presented. From the older, such as *(TelosB or MicaZ)*, to newly realized *(Pycom or Raspberry Pi)*. The last they allow to design and create *WNS* with a smaller size footprint, low cost, better energy efficiency, and computing power. Most of the previous papers used simulations to assess the efficiency and strength of their algorithms. However, the real implementation of encryption algorithms on real nodes provides more realistic results.

Table III shows some examples of sensor platforms. The *Raspberry and PyComs* platforms have more computing power and memory than the rest of the analyzed devices. However, all the involved platforms have limited resources. Therefore, the development of any security protocol or algorithm should take into account both the computing power and the energy consumption. A comparison between *Raspberry Pi* platform and (*TelosB, MicaZ*, and *Iris*) platforms is presented in [49], showing that *Raspberry Pi* devices have some strengths: processing power, memory, connectivity, and flexibility. Two recent use cases to design a *WSN* using *Raspberry Pi* as a sensor platform are: in [50], it is created a low-cost *WSN* for monitoring and measuring the leaf area index (LAI), which is an important parameter for forestry vegetation canopy structure investigation and ecological environment model study. In [51], it is analyzed the performance of the deep learning algorithms in compressing medical images and the efficiency of the Raspberry Pi WSN in transmitting the compressed images across the WSN nodes. Another recent platform is the *PyCom* ecosystem which makes IoT development easier. An IoT prototype device is presented in [52], using *Pycom* as a sensor platform and data shipped with *LoRa* and *Sigfox* communication protocol.

The *CM5000* mote is an IEEE 802.15.4 [15] compliant wireless sensor node based on the original opensource *TelosB* platform [53] using *TinyOs* as an operative system and *nesC* programming language [54]. An accurate energy model (EM) for wireless sensor devices (WSDs) [55] based on power consumption measurement having a *TelosB* device is presented. Also, a procedure for the characterization and optimization of the power consumption and reliability in sensor networks [56] is presented. Another platform is *Arduino*. This platform is proposed together with sensors and cameras in Smart Farming IoT [57].

Finally, we select the following hardware and software.

1) *Hardware: Raspberry Pi* platform described in Table III is used as the *IoT* device due to the reasons expose in [49]. A laptop with an *i3* CPU with 6 GB of RAM, running a *Debian 9* operating system is used to develop the source code and the *Visual Studio* as *IDE* programming.
2) *Software: Python* [58] is used. The selection of this language is because of its large support of *IoT* platforms and its versatility, lightweight consumption resources, and growing-up use.

V. EXPERIMENTATION

In this section, we describe the experiments (or benchmark tests) to assess and validate our proposed method. First, we suppose that our *IoT* devices transmit multimedia data, such as the use cases in [50] or [51]. Therefore, some of the tests that we carried out are histogram analysis, X^2, and Entropy tests. These tests are commonly used when images are studied statistically. Second, a visual analysis to make a graphic understanding of the method is carried out. Third, it is shown the histograms of the original and the scrambled images. In the fourth place, the shuffle process is evaluated either from a visual perspective and also from an analytic point of view. Then, the proposed method is tested against plain–text attacks with the values of NPCR and unified average change in intensity (UACI). In the fifth step, the computation time of the shuffle and scramble process is shown. Then, the entropy values of the original and scrambled images are computed. Next, the X^2 test is calculated. Also, the *Env test* and *800–22-rev1* suite randomness tests are executed to analyze the bits sequence of the internal state. Finally, the proposed method is studied in resistance to brute force attacks.

A. Visual Analysis

The analysis of the proposed method from the point of view of the performance and the cryptography properties is done using well-known and synthetic images rather than using environmental data, which is harder to manage and compare visually. In this section, the results of applying the proposed method to the grayscale version of the *Lake* image are shown.

Fig. 12. Image of *Lake*.

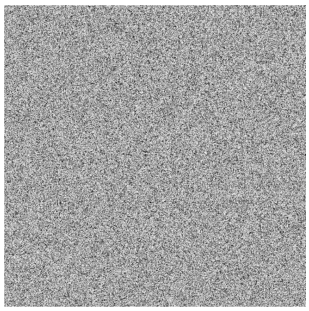

Fig. 13. Image of lake after running *GS3* method.

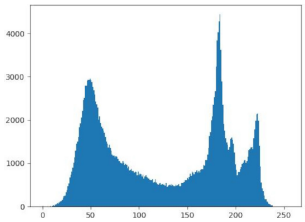

Fig. 14. Histogram of the image of *Lake*.

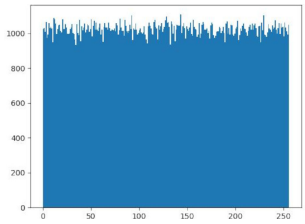

Fig. 15. Histogram of the scrambled image of *Lake*.

Fig. 12 is the original image of *Lake*. Fig. 13 shows the final result after applying all the processes of shuffling, scrambling, and S-boxes as described in Section III.

B. Histogram

In this section, the histograms of both the original well-known image of *Lake* and the scrambled version of it are shown. Fig. 14 shows the histogram of the image. Fig. 15 is the histogram of the ciphered image of *Lake*.

C. Shuffled Bytes

First, we are going to study the proposed mechanism of shuffling from a visual point of view using the gray-scale

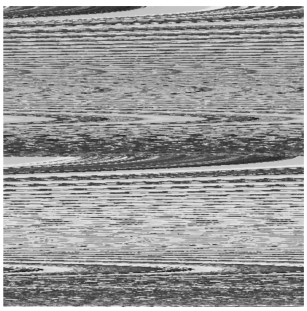

Fig. 16. Image of *Lake* after shuffling with one iteration.

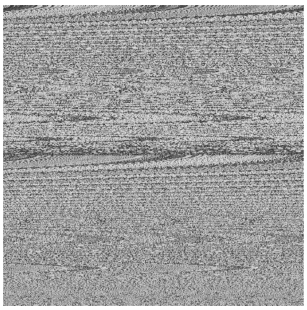

Fig. 17. Image of *Lake* after shuffling with two iterations.

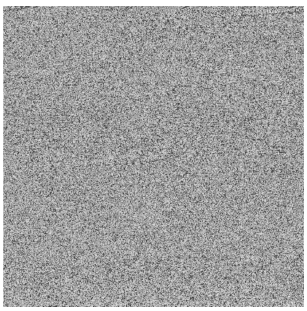

Fig. 18. Image of *Lake* after shuffling with three iterations.

version of the well-known *Lake* image. Second, the number of pixels that were not swapped by the shuffling algorithm is computed using a synthetic image.

Fig. 12 is the 256×256 size image of *Lake*. Fig. 16 shows the achieved image after executing the shuffling algorithm with just one iteration. From a visual point of view, this image is completely different from the initial one. Nevertheless, as the number of iterations increases, the degree of obfuscation rises up. This behavior is shown in Figs. 17 and 18, with 2 and 3 shuffling iterations, respectively.

To calculate the number of snowcapped pixels, a synthetic image with 65536 pixels is created. Each one has a different value (representing a color) in the range of $[0 - 65535]$. This method allows us to find out easily which pixels have not been shuffled. The original image is represented in Fig. 19. In Fig. 20, the shuffled image after just one iteration is presented. In that, it can be observed that the obtained image differs completely from the original one. However, it is detected that the pixels have been shuffled within a short distance among them. For this reason, some forms of waves appear in the shuffled image. Nevertheless, those waves are reduced as the

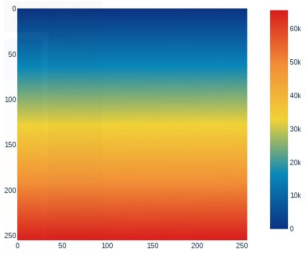

Fig. 19. Color scale image.

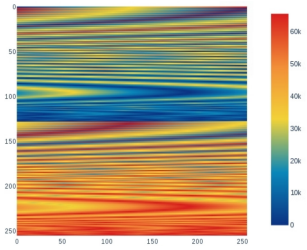

Fig. 20. Shuffling with one iteration.

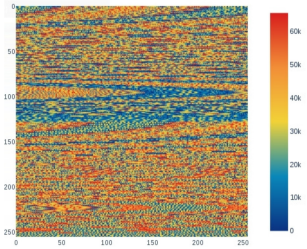

Fig. 21. Shuffling with two iterations.

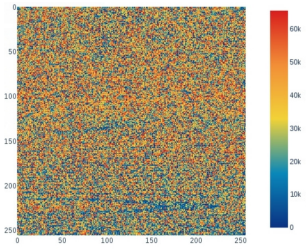

Fig. 22. Shuffling with three iterations.

iteration number increases. This can be observed in Figs. 21 and 22 with 2 and 3 shuffled iterations.

Table IV shows the quantitative results for the number of nonshuffled bytes and the percentage of nonshuffled values of the shuffling step, depending on the size of the image (in bytes) and the applied number of iterations, $[1-3]$.

D. Small Input Variation Analysis

The first analysis is going to test the strength of the proposed method against plain–text attacks, which are based on the differential analysis between plain and cipher data. Therefore, small changes in plain data must provoke greatly different ciphered data. To measure this sensibility, two indicators are calculated, the NPCR and UACI. These indicators are frequently used in image encryption schemes [59]. The Formulas

TABLE IV
QUANTITATIVE RESULTS OF SHUFFLING

Size	1 Iteration	2 Iterations	3 Iterations
64^2	2 Bytes 0.05%	0 Bytes 0%	2 Bytes 0.05%
128^2	2 Bytes 0.05%	0 Bytes 0%	1 Byte 0.01%
256^2	4 Bytes 0.1%	0 Bytes 0%	2 Bytes 0%
512^2	1 Byte 0%	0 Bytes 0%	1 Byte 0%

TABLE V
NPCR AND UACI. PLAIN DATA. IN BOLDFACE, BEST RESULT FOR (NPCR) AND (UACI)

Parameter	1 bit	2 bits	3 bits	+3 bits
NPCR	99.585	99.607	99.60	**99.62**
UACI	33.511	33.53	33.509	**33.35**

TABLE VI
NPCR AND UACI. SENSIBILITY KEY. IN BOLDFACE, BEST RESULT FOR (NPCR) AND (UACI)

Parameter	1 bit	2 bits	3 bits	+3 bits
NPCR	99.61	99.61	99.546	**99.619**
UACI	33.67	33.53	**33.364**	33.31

of both terms are defined in (20) and (22), respectively. The optimum values of these parameters are 99.61%, 33.46% for (NPCR) and (UACI), respectively

$$\text{NPCR} = \frac{1}{(N \times M)} \sum_{i=1}^{N} \sum_{j=1}^{M} D(i,j) \times 100\% \tag{20}$$

$$D_{(i,j)} = \begin{cases} 1 & C_1(i,j) \neq C_2(i,j) \\ 0 & C_1(i,j) = C_2(i,j) \end{cases} \tag{21}$$

$$\text{UACI} = \frac{1}{(N \times M)} \sum_{i=1}^{N} \sum_{j=1}^{M} |C_1(i,j) - C_2(i,j)| \times 100\%. \tag{22}$$

First, a byte is randomly chosen to carry out the tests. Using that random byte, the four more significant bits (MSbs) of it are modified. They are indicated in Table V as *1* bit, *2* bits, *3* bits, and *+3* bits, respectively. Table V shows the obtained results. We have highlighted in boldface the best results for (NPCR) and (UACI).

The second analysis is to test the behavior of the proposed method when the same data block is scrambled with small changes in the seeds or keys. In this case, the most significant 1 bit, 2 bits, 3 bits, and + 3 bits of the initial value of the parameter b in (2) are modified. The b parameter operates like a private key. Table VI shows the obtained values.

E. Execution Time

In this section, the execution times of the proposed mechanism are shown. Table VII presents the results. The first column, *size*, represents the data block size: four different sizes of data block have been used, ranging from 4906 to 65 536 bytes. The second column shows the overhead of the shuffling process, computed using one, two, and three iterations. The

TABLE VII
EXECUTION TIME (SECONDS). IN BOLDFACE, THE LOWEST RESULT FOR EACH SIZE OF DATA

Size	Shuffling		Scrambling	$\sum$ Time (s)	
64^2	1 Ite	**0.030**	0.078	1 Ite	**0.108**
	2 Ite	0.031		2 Ite	0.109
	3 Ite	0.032		3 Ite	0.110
128^2	1 Ite	**0.06**	0.4	1 Ite	**0.46**
	2 Ite	0.07		2 Ite	0.47
	3 Ite	0.11		3 Ite	0.51
256^2	1 Ite	**0.17**	1.39	1 Ite	**1.56**
	2 Ite	**0.17**		2 Ite	**1.56**
	3 Ite	0.18		3 Ite	1.57
512^2	1 Ite	**0.53**	6.53	1 Ite	**7.06**
	2 Ite	0.64		2 Ite	7.17
	3 Ite	0.73		3 Ite	7.26

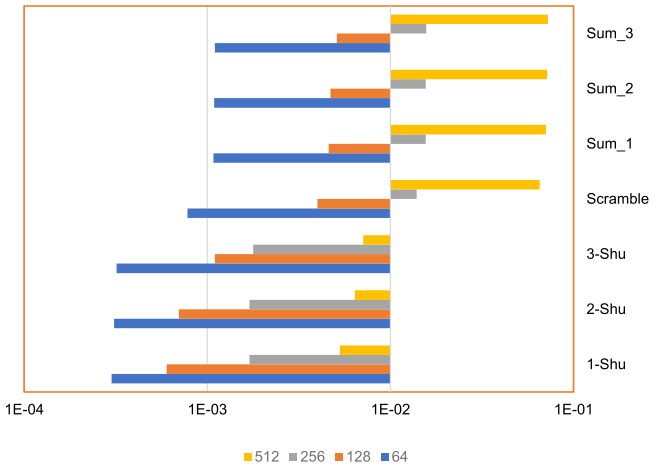

Fig. 23. Execution time (seconds).

third column is the computation time taken to execute the scrambling process. The last column is the sum of all the previous times.

The results of Table VII are represented in Fig. 23. The X-axis is shown in the logarithm scale. The Y-axis represents the computation time of shuffling, and scrambling processes, and the sum of all the previous times from each data block $size^2$.

F. Entropy Values

The *Entropy* is an indicator to measure the randomness or unpredictability level of a data source [44]. It is defined by

$$H(S) = -\sum_{i=1}^{N} P(s_i)\log_2 P(s_i) \tag{23}$$

where S variable is random and it produces output values between $(s_1, .., s_N)$ and $P(s_i)$ is the probability of the occurrence of the s_i output. For example, in the case of data with 8-bit precision, such as the grayscale images, there are $2^8 = 256$ possible different values. In this case, an ideal value of *Entropy* should be 8, which represents that all pixels are equally probable, and, therefore, the probability is completely random.

In Table VIII, the *Entropy* values are shown for the *Lake* image according to four different sizes of data and three different data structures: 1) plain (unmodified); 2) shuffled; and 3) scrambled data. We have highlighted in boldface the highest *Entropy* values after the Scrambling process.

TABLE VIII
ENTROPY VALUES. IN BOLDFACE, THE HIGHEST RESULT FOR COLUMN *Scrambled*

Size	Plain Data	Shuffled	Scrambled
64^2	7.41	7.41	7.96
128^2	7.45	7.45	7.98
256^2	7.44	7.44	**7.99**
512^2	7.44	7.44	**7.99**

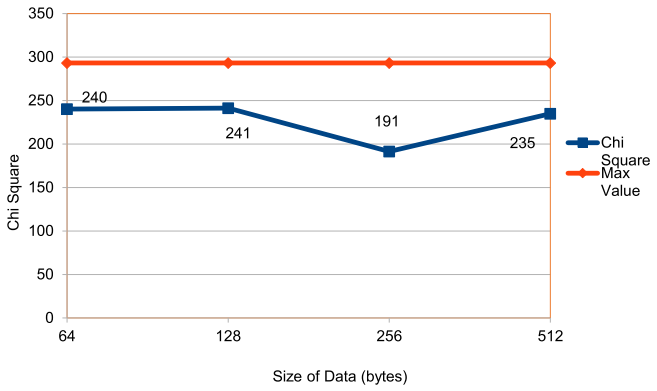

Fig. 24. χ^2 values.

G. χ^2 Test

A way to test quantitatively the uniformity of the ciphered data is the χ^2 test. It is defined by (24). As a data block, we consider a grayscale image with 8-bits per pixel. Therefore, $k = 2^8$, o_i is the observed probability and e_i is the expected probability of the level gray k_i, respectively. The null hypothesis H_0 is to consider the histogram of the ciphered data block to be uniform and the H_1 hypothesis, on the contrary case. The significance level is $\alpha = 0.05$, so the percentage of acceptance of the null hypothesis is 95%. According to the 255 degrees of freedom χ^2 table and $\alpha = 0.05$, the critical value of χ^2 is $\chi^2 = 293.247$

$$X^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i} \tag{24}$$

$$X^2 : \begin{cases} \text{Yes}, & \chi^2_{\text{test}} < 293.24 \\ \text{No}, & \chi^2_{\text{test}} >= 293.24. \end{cases} \tag{25}$$

Fig. 24 shows the computed χ^2 values for four different sizes of a gray-scale image, concretely: 1) (64^2; 2) 128^2; 3) 256^2; and 4) 512^2). The orange line is the value of reference in this example 293.247, the blue line represents the results after scrambling the data. In all the cases, the null hypothesis stands as the χ^2 values are below the critical value.

H. Randomness Tests Ent Test

A Statistical Test Suite for Random and Pseudo-random Number Generators for cryptographic applications is the *Ent test* suite [60]. Table IX shows the randomness of the generated sequence by the proposed internal state. It has been

TABLE IX
ENT TEST

Statistic	Value	Optimum
Entropy	1	1
χ^2	0.66	
Mean	0.499	0.5
MonteCarlo	3.14162889	π
Correlation	0.000069	0

TABLE X
800-22-REV1 TEST

Test	P-Value	Result
monobit	0.75697	PASS
frequency-within-block	0.26798	PASS
run	0.65342	PASS
longest-run-ones-in-block	0.65562	PASS
binary-matrix-rank	0.82868	PASS
dft	0.0	FAIL
non-overlapping-template-matching	0.99999	PASS
overlapping-template-matching	0.00508	FAIL
maurers-universal	0.37574	PASS
linear-complexity	0.59878	PASS
serial	0.33591	PASS
approximate-entropy	0.37353	PASS
cumulative-sums	0.89654	PASS
random-excursion	0.28323	PASS
random-excursion-variant	0.11324	PASS

generated with a binary file about 10^6 bits, which represents five data blocks of size 512^2 bytes.

The output of this test is the following.

1) Entropy = 1.00 bits per bit. Optimum compression would reduce the size of this 10484640 bit file by 0%.
2) χ^2 distribution for 10484640 samples is 0.10, and randomly would exceed this value 75.70% of the times.
3) Arithmetic mean value of data bits is 0.50, where 0.5 means random.
4) Monte Carlo value for Pi is 3.138799615 (error 0.09%).
5) Serial correlation coefficient is 0.000069 (totally uncorrelated should be 0.0).

I. Randomness Tests 800-22-Rev1

Another test suite is the 800-22-rev1 suite, published by NIST [23]. To use this test with our proposal, 100 sequences of 100 000 bits, as suggested by [23], have been generated. Afterward, each sequence has been shuffled and scrambled. Finally, these shuffled and scrambled sequences have been tested in the 800-22-rev1 suite. The results are shown in Table X.

Only 2 out of the 15 tests that compose the suite fail to assess the randomness of the proposal.

J. Key Space Analysis

Key space is defined as the total number of different keys that can be generated to encrypt data. The recommendations [61], [62] for minimal length key for symmetric algorithms is 112 bits. The proposed method achieves the following key space.

1) *Chaotic System:* The subprocesses *Shuffling Algorithm, Scrambling*, and *S-Boxes* of the proposed method, Section III, makes use of the LM. This chaotic system has two parameters, b and X_0 in (2), which are considered private keys. These parameters are initiated with different values in each subprocess. If it is considered a precision of 64 bits for each parameter in (2), there are 2^{384} total possible permutations. Therefore, each LM would achieve 2^{128} permutations.
2) *Internal State:* The Internal State size is 288 bits in Fig. 8. Although the initial state is set up with 128 bits for the key values, depending on security level requirements, it is possible to use 288 bits for key values. Therefore, the possible permutations are bounded by 2^{128} and 2^{288}.

So, taking into account previous issues, we can evaluate that the key space could reach $2^{544} \approx (5.75 \times 10^{163})$ that is much larger than 2^{112}, which makes a brute-force attack to be unfeasible.

K. Brute Force Attack

In this section, it is studied an attack scenario against *Shuffle Process* from two perspectives. The first one is named *Brute Force Attack by full matrices* and the second *Brute Force Attack with 1-D Data Vectors*. Also, we suppose that the intruder is able to crack the security of *Scrambling* and *S-boxing* processes.

1) Brute Force Attack by Full Matrices: The attacker does not have any knowledge about the sent data. Therefore, the positions of the cells are unknown and the only way to break the privacy is by guessing the ordering. Besides, the cycle redundant control 16 (CRC16) value is used as integrity control *FCS*. First, the intruder must save on memory all the possible combinations of the $S^b_{(n\times m)}$ shuffled data block, in (9). Second, *CRC16* values by each row and column must be calculated. Finally, when the intruder receives the $S^{(b+1)}_{(n\times m)}$ data, must compare the calculated *CRC16* values with the received *CRC16* values. In the case of finding out a coincidence, the intruder could guess that has discovered a correct combination of the value data by row or column. The number of possible combinations is defined by the following equation:

$$V_{(n.m),(n-1.m-1)} = \frac{(n.m)!}{(n+m-1)!}. \tag{26}$$

An example of a brute force attack is shown in Table XI. We suppose 2-bytes size for each element $S^b_{i,j}$ in (9). The second column represents the matrix size (bytes). The third column is the number of different matrices. The fourth column is the memory consumption to save these matrices. Finally, the execution time is measured with an enhanced software implementation of the *CRC16* value in [63], which is indicated in the fifth column.

The results of Table XI are displayed in Fig. 25. It can be observed comparatively the memory consumption and the execution time of the *CRC16* value depending on the number of matrices. The *Y*-axis is represented in a logarithmic scale.

For example, for an 8×8 matrix sending 128 bytes, this payload fits in a single message for common *WSN* and

TABLE XI
BRUTE FORCE ATTACK BY FULL MATRICES

	Size (bytes)	Matrices	Memory (bytes)	Time (s)
2 x 2	8	4	32	6.93^{-10}
3 x 3	18	3024	52432	5.23^{-7}
5 x 5	50	4.27^{19}	2.14^{19}	7.41^{9}
7 x 7	98	9.77^{52}	9.57^{54}	1.69^{43}
8 x 8	128	9.7^{76}	1.24^{79}	1.68^{67}
10 x 10	200	7.67^{140}	1.53^{143}	1.33^{131}

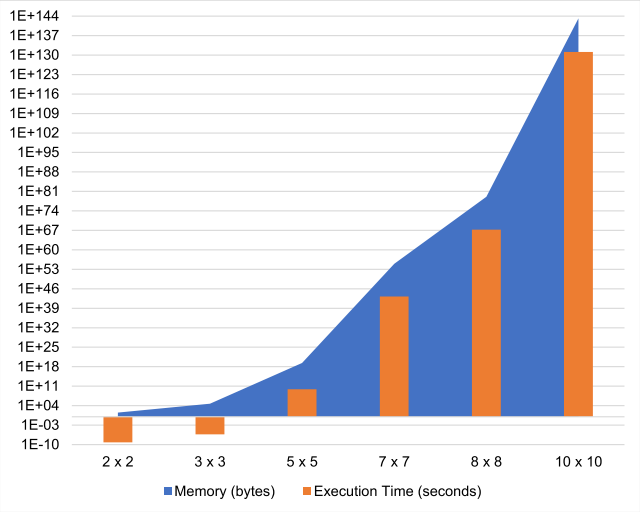

Fig. 25. Brute force attack by full matrices.

TABLE XII
BRUTE FORCE ATTACK WITH 1-D DATA VECTORS

	1st Step Vectors	2nd Step Matrices	Memory (bytes)	Execution Time (s)
2 x 2	4	1	24	3.25^{-10}
3 x 3	72	2	468	9.88^{-9}
5 x 5	303600	24	30.37^{5}	7.861^{-5}
7 x 7	1.0068^{7}	720	1.409^{11}	3.9266
8 x 8	3.1309^{12}	5040	5.009^{13}	1424.57
10 x 10	6.902^{17}	362880	1.38^{19}	403.81^{6}

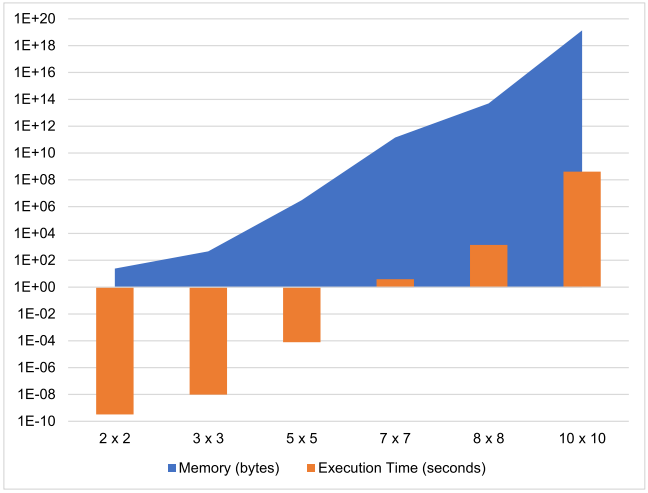

Fig. 26. Brute force attack with 1-D data vectors.

IoT protocols. An attacker would calculate 9.7^{76} matrices. However, this immense number of matrices would require a huge amount of memory, about 24.1×10^6 exabytes. For instance, a 4-GHz computer running an efficient *CRC16* software implementation would take more than 39.52×10^6 years of computation.

2) Brute Force Attack With 1-D Data Vectors: In this case, although the attack is done by brute force attack, the intruder has a bit more intelligence, with the objective being to reduce the execution time and memory consumption. The procedure is the following.

1) All possible combinations of data rows are generated, instead of generating all combinations of matrices. Equation (27) defines the number of combinations of data rows

$$V_{(n.m),(m-1)} = \frac{(n.m)!}{(n.m - m - 1)!}. \tag{27}$$

2) The *CRC16* value is calculated for each r_i row. When the $(b+1)$ data block is received, the *CRC16* calculated in the previous step are sought in the $(b+1)$ data block. If there is a success, it can be supposed that the r_i data row appeared in the b data block.
3) To find out the correct order of each found r_i row, it is necessary to calculate the *CRC16* by columns. The number of these combinations is given

$$V_{(n-1),(n-1)} = (n-1)!. \tag{28}$$

In Table XII it is shown the number of data vectors, number of matrices, memory consumption (bytes), and execution time (seconds) for different data block sizes.

The memory (bytes) and execution time (seconds) columns of Table XII are shown in Fig. 26. *Y*-axis is represented in a logarithmic scale. Taking as an example a data block with (8×8) size, the memory consumption would be about 45 terabytes to save vectors and matrices. The execution time could be about 23 min.

VI. COMPARISON

The proposed method is evaluated in this section by comparing it with the ciphers presented in Section II. This comparison is based on the following metrics.

1) *Execution Time:* This parameter is commonly used to compare the consumption of resources of different algorithms. However, it is crucial in resource-constrained *IoT* devices.
2) *Entropy Test:* This value measures the randomness of the ciphered data. Therefore, if all value data have the same or similar probability, the entropy value is maximum.
3) χ^2 *Test:* This metric is used to confirm the results of the entropy test. The higher the uniformity of the ciphered data is, the harder an intruder could obtain useful knowledge about those data.

A. Execution Time

The execution timings have been calculated taking into account: 1) the *Raspberry IoT* device, named *RPi* described

TABLE XIII
EXECUTION TIME (SECONDS). IN BOLDFACE, THE LOWEST RESULT FOR EACH PLATFORM AND IMAGE SIZE

Method	64^2	128^2	256^2
Proposed method			
GS3	0.08	0.32	1.21
Lightweight Stream ciphers			
A5/1	3.18	11.8	48.8
Geffe	23.6	289	∞
Trivium	1.25	4.08	15.2
Pentavium	31.2	119	443
RC4	0.06	0.26	1.04
RC4-Marc	0.04	**0.2**	0.75
Salsa20	3.2	12	47.4
Chacha20	0.26	0.6	1.94
Lightweight Block ciphers			
AES	0.11	0.45	1.77
Present	0.62	4.44	18.3
DES	1.52	6.11	24.3
Simon	0.05	0.21	0.85
Speck	**0.03**	0.14	**0.57**

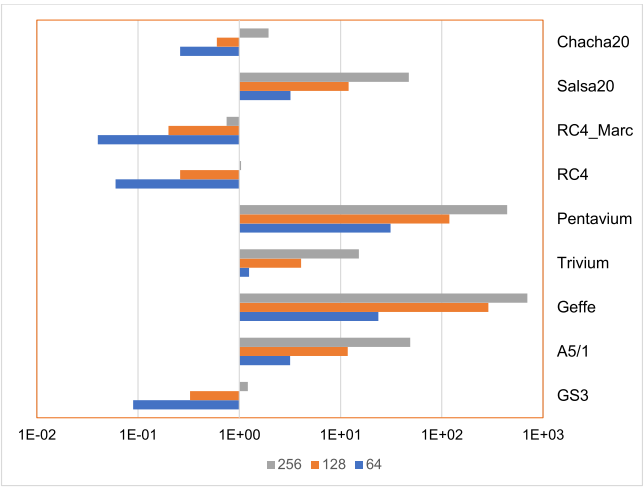

Fig. 27. Comparing execution time of stream ciphers.

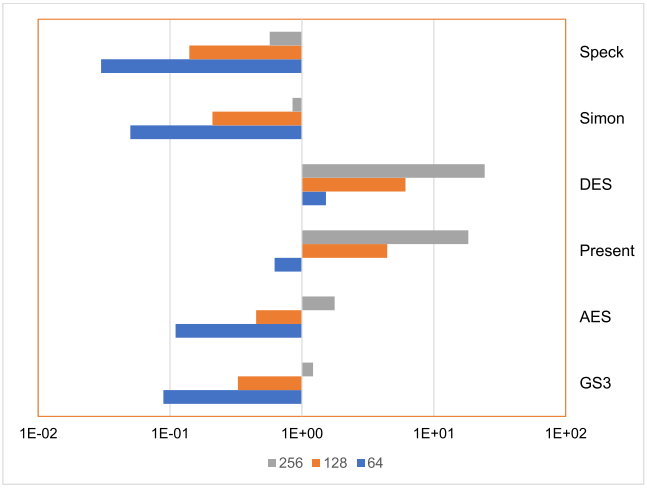

Fig. 28. Comparing execution time of block ciphers.

TABLE XIV
COMPARING ENTROPY. IN BOLDFACE, THE BEST RESULTS FOR EACH IMAGE SIZE

Method	64^2	128^2	256^2
Initial Entropy	7.411	7.455	7.44
Proposed method			
GS3	**7.96**	**7.98**	**7.99**
Lightweight Stream ciphers			
A5/1	**7.96**	7.98	**7.99**
Geffe	7.95	**7.98**	**7.99**
Trivium	**7.96**	7.98	**7.99**
Pentavium	7.95	**7.99**	**7.99**
RC4	7.95	**7.99**	**7.99**
RC4-Marc	**7.96**	7.98	7.98
Salsa20	7.95	7.98	**7.99**
Chacha20	7.94	7.98	**7.99**
Lightweight Block ciphers			
AES	7.95	7.98	**7.99**
Present	7.95	7.88	**7.99**
DES	7.95	7.98	**7.99**
Simon	7.95	7.98	**7.99**
Speck	7.94	7.98	**7.99**

in Table III shows its model and characteristics and 2) three different sizes of data blocks and using gray-scale images.

Table XIII shows the execution timings in seconds. The first column is called *Method*, and it represents the analyzed cipher algorithm. The second, third, and, fourth columns represent the execution time regarding the data block size in bytes. Three subgroups are represented in Table XIII. The first of one is the execution time of the proposed GS3 method. The second group is the *Lightweight Stream ciphers*, described in Section II-A. The third group is *Lightweight Block ciphers*, reviewed in Section II-B. We have highlighted the lowest result for each image size in boldface.

The results indicated in Table XIII are shown with a bar chart in Fig. 27 comparing GS3 with the stream ciphers, and in Fig. 28 making the comparison with block ciphers. The execution time in seconds is represented in logarithmic scale in the X-axis, and the size of images in bytes (64^2, 128^2, 256^2) are represented with three colors, whilst the reviewed ciphers are indicated in the Y-axis. For each cipher and size of the data block, the execution time is depicted with a bar. The GS3 proposed method has also been executed in the platform to develop the source code, Intel i3, obtaining an execution time of 0.11, 0.51, and 1.57 s for each image size, respectively. Therefore, comparing these times with Table XIII, it can be figured out the computing power of the *Raspberry Pi* platform.

B. Entropy Test

This section compares the entropy values. According to Section III-F, three sizes of data blocks are used. The initial entropy values for these images are 7.41, 7.45, and 7.44, respectively, as shown in Table XIV. In this case, the optimum results should be close to 8.00. We have highlighted the highest results for each image size in boldface.

For each cipher and size of the image, the entropy is calculated. After that, it is calculated the percentage of ciphers that provide that same entropy value. Table XIV is graphically depicted in Figs. 29 and 30, splitting the Stream and the Block ciphers for the sake of clarity. In the Y-axis, the entropy

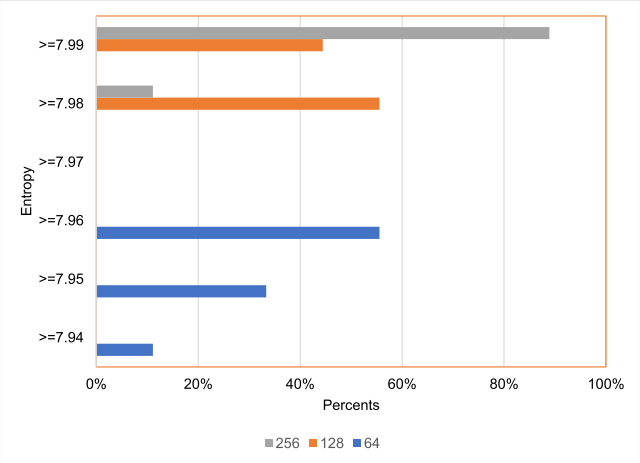

Fig. 29. Percentage of stream ciphers by entropy values.

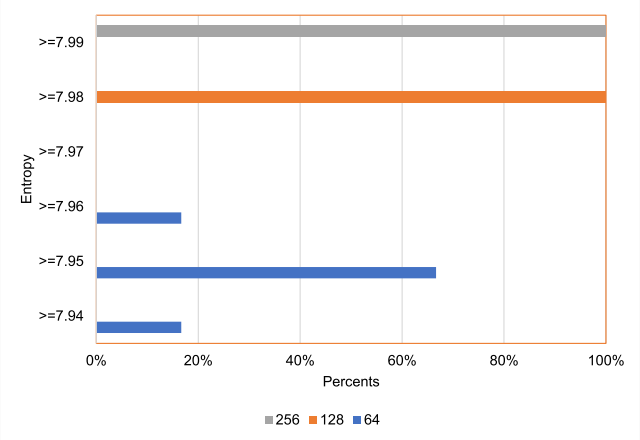

Fig. 30. Percentage of block ciphers by entropy values.

values are represented. They are classified in six intervals. The percentage value of ciphers for each given entropy value is represented in the *X*-axis.

It is worth remarking that the entropy value for the *Lake_64* image is lower than the ones from *Lake_128 and Lake_256* images in both Figs. 29 and 30.

C. χ^2 Test

This section compares the χ^2 values obtained by the ciphers and the GS3 method, taking into account the *Performance Parameters* and the χ^2 *test*, described in Sections III-F and V-G. We have highlighted in boldface the lowest results for each image size.

A visual representation of Table XV it is shown in Figs. 29 and 32, for Stream and Block ciphers, respectively. For each cipher and size of the image, the χ^2 test is computed. After that, the χ^2 values are grouped and the percentage of ciphers in a given interval is computed. In the *Y*-axis, the χ^2 values are depicted and classified into four intervals $[0-199]$, $[199-230.3]$, $[230.3, 261.3]$, $[261.3, 293]$. The *X*-axis shows the percentage of ciphers with χ^2 test values laying in these intervals.

TABLE XV
COMPARING χ^2. IN BOLDFACE, THE LOWEST RESULTS FOR EACH IMAGE SIZE

Method	64^2	128^2	256^2
Initial χ^2	28092.72	7121.81	1892.62
Proposed method			
GS3	**220.25**	249.81	242.19
Lightweight Stream ciphers			
A5/1	246.25	253.68	248.81
Geffe	255.85	223.15	239.98
Trivium	225	271.25	291.25
Pentavium	236	**225.59**	260.48
RC4	267.12	265.1	**198.65**
RC4-Marc	233.25	229.31	264.18
Salsa20	237.15	249.93	252.69
Chacha20	292.25	399.84	648.35
Lightweight Block ciphers			
AES	254.25	273.45	256.89
Present	226.75	285.4	299
DES	226.25	259	259.53
Simon	246.75	270.9	270.7
Speck	280.12	262	233.7

VII. DISCUSSION

After the exposition of the results provided in Section VI, in this section, we make a more in-depth analysis of them, comparing the results, and trying to determine all the facts affecting them. First, all the ciphers based on *NLFRS* described in Section II-A1 *(A5/1, Geffe, Trivium,* and *Pentavium)* take extremely large execution times. Second, the *RC4* and *RC4-Marc* are the stream ciphers with the lowest execution time, followed by our proposed GS3 method and by the *Chacha20* cipher. Nevertheless, *RC4* cipher should not be used as it was prohibited in *TLS* by RFC 7465 published in February 2015, due to the identified vulnerabilities, such as brute force and statistical attacks [64]. Besides, *RC4* does not take into account Shannon's cryptography principle of diffusion. On the other hand, *GS3* takes into consideration this principle. Moreover, *GS3* performs a double confusion phase, the first of them is done in the scrambling process and the second is carried out with the *S-box* process. Concerning the *ChaCha20* encryption algorithm, although it was designed to provide a combination of speed and security, it uses 10 iterations of the double round with a state size of 512 bits. On the other hand, *Scrambling subprocess* of the *GS3* uses only 3 iterations of the simple round with a state size of 288 bits. Besides, *GS3* executes the lightweight *Shuffling* and *S-Boxing* subprocesses to enlarge its cryptographic strength.

Related to the block cipher category, *Speck, Simon,* and *AES* ciphers require much less execution time than *Present* and *DES* ones. Both of them take extremely large execution times on *IoT* environments. Our proposed GS3 method requires about 31% less time than *AES* in the *RPi* platform. However, *Simon* and *Speck* take less execution time than GS3 using *RPi* platform. Nevertheless, *GS3* holds a key space that can reach more than 2^{320}, which is large enough to make brute-force attacks infeasible. Besides, it is larger than the key space of all analyzed ciphers in Section VI.

Analyzing the results of Table VIII for the *Entropy* test and Figs. 29 and 30, after the scrambling process, all the stream,

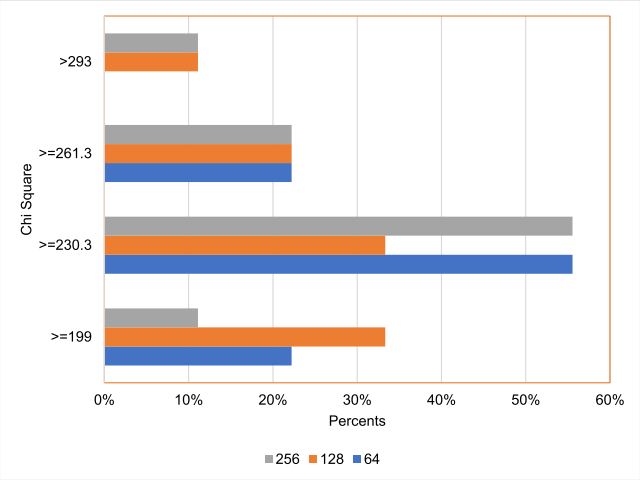

Fig. 31. Percentage of stream ciphers by χ^2 values.

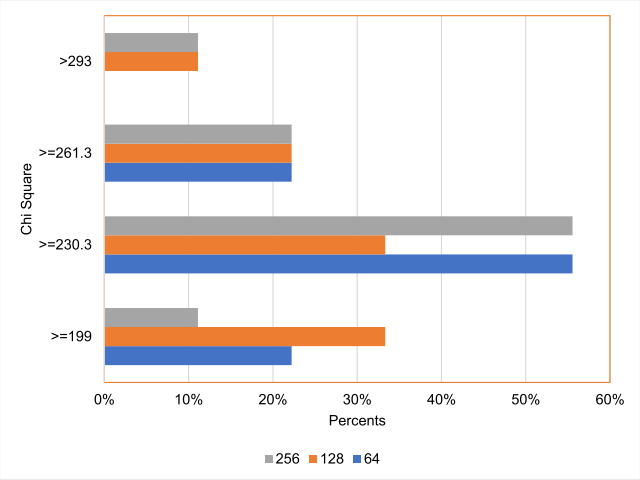

Fig. 32. Percentage of block ciphers by χ^2 values.

block, and GS3 ciphers provide entropy values very close to 8. Therefore, the obtained randomness level is close to the maximum. If we compare GS3 method with stream ciphers for the *Lake_256* image, it is classified at the top with an entropy value of 7.99. For the smallest size image, *Lake_64*, the proposed GS3 method provides the greatest entropy value, 7.96. In this case (*Lake_64*), GS3 obtains the highest entropy value among all the block ciphers. Moreover, for *Lake_128 and Lake_256* images, the GS3 method is classified in the top percentages where entropy values are maximum in their ranges, (7.98, 7.99) respectively. Thus, it can be confirmed that the proposed GS3 method provides the highest level of entropy among either the Lightweight Stream ciphers or the Lightweight Block ciphers.

We are analyzing the results of the χ^2 test in Table XV and Figs. 31 and 32. The initial χ^2 values for these images are much greater than 293.247, therefore, the null hypothesis *H*0 should be rejected. However, after the scrambling process, only two ciphers fail the test. The *Salsa20* cipher fails the test for *Lake_128* and *Lake_256*. Besides, the *Present* cipher does not pass χ^2 test for the *Lake_256* image. On the contrary, the proposed GS3 method passes the χ^2 tests for all the image sizes. Furthermore, GS3 provides better test values than *AES, Salsa20, DES,* and *Simon* ciphers for the three image sizes. In most cases, the GS3 method has better χ^2 test values than the rest of the ciphers.

In Fig. 31, the proposed GS3 method versus stream ciphers are assessed for χ^2 test values. Only 22% of the ciphers are classified in the [190, 230.1) range for *Lake_64* image. It is worth remarking that, for the χ^2 test, the lower the value, the better. Our proposed GS3 method stays in that lowest interval. In the case of *Lake_128* image, 33.3% of the ciphers provide χ^2 values in the second–level quality interval, which is in the range of [230.3, 261.3). GS3 method is located in that interval for *Lake_128* image. For *Lake_256* images, 55% of the ciphers get a χ^2 value between [230.3, 261.3), where our proposed method is located. However, about the 11% of the ciphers do not pass the test. In particular, *Chacha20* cipher cannot guarantee the fulfillment of the null hypothesis, an therefore, the randomness test is rejected for *Lake_128* and *Lake_256* images.

The analysis of the key space determines that the *GS3* method can reach at minimal the amount of 2^{320}, which is suitable with the recommendations in [61] and [62], 2^{112}. Besides, supposing the attacker has broken the *Scrambling* and *Sboxing* processes, the analysis of the brute force attack for the *Shuffle* subprocess in the worst case for an attacker, reveals that computation time would overcome millions of years. Nevertheless, a smarter attacker would execute the attack *Brute Force Attack with 1-D Data Vectors* described in Section V-K2. Taking as an example a data block size of (8×8), the memory consumption would be about 45 terabytes. This amount of memory could be available in super-computers or cloud-computing paradigms. An execution time of 23 min is a low value, if the nodes sent data with intervals of less than 23 min, although the attacker would achieve original data, the freshness property of data is not accomplished. Therefore, these data would not have utility.

VIII. Conclusion

In this research, a lightweight shuffling and scrambling method, named GS3, is proposed, which is strong enough to keep data safe meanwhile the data is used and takes less time than the most secure encryption mechanisms. This method fulfills *Shannon's* cryptography principles of diffusion and confusion [44] in *IoT* environments. Because *IoT* devices have constrained computational resources, GS3 introduces minimal overhead either in the length of the message or the execution time to provide privacy to exchanged data. Besides, using the *Raspberry Pi* platform, the proposed GS3 method can shuffle and scramble data taking about 30% less time than using *AES* cipher, and about 50% less than using the *Chacha20* cipher.

On the one hand, the analysis of the statistic parameters *Entropy, χ^2, Histogram, NPCR, UACI,* and *Randomness tests* exposes that our proposal passes all of these tests. On the other hand, our pseudo-random number generator (*PRNG*) of the *Internal State*, described in Section III-C, has been proved with the "*Ent Test*" and the *800-22-rev1* tests, described in the

sections *H* and *I*, respectively. In both cases, the results reveal the good behavior of the proposed method.

Both, the key space in Section V-J and the brute force attack in Section V-K studies reveal the strength of the security *GS3*.

To summarize, the proposed method is a lightweight software mechanism suitable for *IoT* environments, which can protect the communications between end–nodes and the sink node, taking into account the tradeoff between the security level and the consumption of computational resources. This research improves the approach described in [19], introducing a confusion step with scrambling and substitution steps.

IX. Future Works

GS3 has a clear weakness if a hacker captures the initial seed value. Therefore, two modifications will be considered in the following version of the *GS3* method: use a (previously shared) symmetric key encryption to share the seed between each sender and receiver, and to send a new seed after a given number of interchanged messages. Although we avoid using the encryption operation for all the data in the message, because of the computational complexity of this method, it will be applied to a few data. Therefore, the time penalization is very limited and can be affordable for a resource–constrained device.

Shortly, we will test our proposal on *Jetson One and ESP32* platforms. Besides, we will implement the *GS3* method in a different programming language than *Python* because metrics, such as execution time or memory consumption, are highly influenced by interpreted programming languages.

Data transfer usually represents the main part of energy consumption in *IoT*. So, data reduction should always be applied if possible. In future works, we will combine data reduction techniques [65] with the GS3 obfuscation method in a real deployment, analyzing energy consumption and computation on distributed scenarios [66].

Acknowledgment

Funding for open access charge: Universidad de Córdoba/CBUA.

References

[1] N. Homma, K. Aoki, and T. Iwata, *Cryptographic Technology Guideline*. Cryptogr. Res. Eval. Comm., Tokyo, Japan, 2017.

[2] 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.

[3] L. E. Kane, J. J. Chen, R. Thomas, V. Liu, and M. Mckague, "Security and performance in IoT: A balancing act," *IEEE Access*, vol. 8, pp. 121969–121986, 2020.

[4] R. P. França, Y. Iano, A. C. B. Monteiro, and R. Arthur, "Intelligent applications of WSN in the world: A technological and literary background," in *Handbook of Wireless Sensor Networks: Issues and Challenges in Current Scenarios*, P. K. Singh, B. K. Bhargava, M. Paprzycki, N. C. Kaushal, and W.-C. Hong, Eds. Cham, Switzerland: Springer, 2020, pp. 13–34.

[5] P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards," *Comput. Commun.*, vol. 30, no. 7, pp. 1655–1695, 2007.

[6] "Low range LoRa alliance." Accessed: Apr. 1, 2024. [Online]. Available: https://lora-alliance.org

[7] S. Lata, S. Mehfuz, and S. Urooj, "Secure and reliable WSN for Internet of Things: Challenges and enabling technologies," *IEEE Access*, vol. 9, pp. 161103–161128, 2021.

[8] I. Ahmad et al., "Analysis of security attacks and taxonomy in underwater wireless sensor networks," *Wireless Commun. Mobile Comput.*, vol. 2021, pp. 1–15, Dec. 2021.

[9] S. Wang and Y. Chen, "Optimization of wireless sensor network architecture with security system," *J. Sens.*, vol. 2021, pp. 1–11, Nov. 2021.

[10] F. Rezaeibagha, Y. Mu, K. Huang, and L. Chen, "Secure and efficient data aggregation for IoT monitoring systems," *IEEE Internet Things J.*, vol. 8, no. 10, pp. 8056–8063, May 2021.

[11] R. Manjula, T. Koduru, and R. Datta, "Protecting source location privacy in IoT-enabled wireless sensor networks: The case of multiple assets," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10807–10820, Jul. 2022.

[12] C. Peng, M. Luo, P. Vijayakumar, D. He, O. Said, and A. Tolba, "Multifunctional and multidimensional secure data aggregation scheme in WSNs," *IEEE Internet Things J.*, vol. 9, no. 4, pp. 2657–2668, Feb. 2022.

[13] M. El-Hajj and A. Fadlallah, "Analysis of lightweight cryptographic algorithms on IoT hardware platforms," in *Proc. 32nd Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, 2022, pp. 121–126.

[14] J. Nechvatal et al., "Report on the development of the advanced encryption standard (AES)," *J. Res. Nat. Inst. Stand. Technol.*, vol. 106, no. 3, pp. 511–577, 2001.

[15] *IEEE Standard for Low-Rate Wireless Networks*, IEEE Standard 802.15.4, Accessed: Apr. 1, 2024. [Online]. Available: http://www.ieee802.org/15/pub/TG4.html

[16] A. Bogdanov et al., "PRESENT: An ultra-lightweight block cipher," in *Proc. Int. Workshop Cryptogr. Hardw. Embed. Syst.*, 2007, pp. 450–466.

[17] "The eSTREAM project, Salsa20." Accessed: Apr. 1, 2024. [Online]. Available: https://www.ecrypt.eu.org/stream/salsa20pf.html

[18] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," Internet Eng. Task Force, RFC 7539, 2015.

[19] F. Alcaraz Velasco, J. M. Palomares, and J. Olivares, "Lightweight method of shuffling overlapped data-blocks for data integrity and security in WSNs," *Comput. Netw.*, vol. 199, Nov. 2021, Art. no. 108470.

[20] S. R. Moosavi, E. Nigussie, S. Virtanen, and J. Isoaho, "An elliptic curve-based mutual authentication scheme for RFID implant systems," *Procedia Comput. Sci.*, vol. 32, pp. 198–206, Jan. 2014.

[21] S. Shamshad, F. Riaz, R. Riaz, S. S. Rizvi, and S. Abdulla, "An enhanced architecture to resolve public-key cryptographic issues in the Internet of Things IoT, employing quantum computing supremacy," *Sensors*, vol. 22, no. 21, p. 8151, 2022.

[22] G. Solomon, *Shift Register Sequences: Secure and Limited-Access Code Generators, Efficiency Code Generators, Prescribed Property Generators, Mathematical Models*. Singapore: World Sci., 2017.

[23] L. E. Bassham et al., "A statistical test suite for random and Pseudorandom number generators for cryptographic applications," U.S. Dept. Comm. Nat. Inst. Stand. Technol., Gaithersburg, MD, USA, Rep. SP 800-22, 2010.

[24] N. B. Atti, G. M. Diaz-Toca, and H. Lombardi, "The Berlekamp–Massey algorithm revisited," *Appl. Algebra Eng., Commun. Comput.*, vol. 17, no. 1, pp. 75–82, 2006.

[25] P. K. Gundaram, A. N. Tentu, and S. N. Allu, "State transition analysis of GSM encryption algorithm A5/1," *J. Commun. Softw. Syst.*, vol. 18, no. 1, pp. 36–41, 2022.

[26] N. Bajaj. "Linear feedback shift register: 1.0.6." Accessed: Apr. 17, 2023. [Online]. Available: https://pypi.org/project/pylfsr/

[27] F. Dridi, S. El Assad, W. El Hadj Youssef, M. Machhout, and R. Lozi, "Design, implementation, and analysis of a block cipher based on a secure chaotic generator," *Appl. Sci.*, vol. 12, no. 19, p. 9952, 2022.

[28] C. De Cannière and B. Preneel, *New Stream Cipher Designs: The eSTREAM Finalists*, M. Robshaw and O. Billet, Eds. Berlin, Germany: Springer, 2008.

[29] "The eSTREAM project." Accessed: Mar. 1, 2024. [Online]. Available: https://www.ecrypt.eu.org/stream/finallist.html

[30] U. Yudha. "Source code of trivium." Accessed: Jan. 4, 2024. [Online]. Available: https://github.com/uisyudha/Trivium

[31] S. Wolfram, "Random sequence generation by cellular automata," *Adv. Appl. Math.*, vol. 7, no. 2, pp. 123–169, 1986.

[32] I. G. A. Poornima, B. Paramasivan, K. M. Pitchai, and M. Bhuvaeswari, "A survey on cellular automata with the application in pseudo random number generation," *J. Netw. Inf. Secur.*, vol. 5, no. 2, pp. 1–11, 2017.

[33] A. John, B. C. Nandu, A. Ajesh, and J. Jose, "PENTAVIUM: Potent trivium-like stream cipher using higher radii cellular automata," in *Proc. Int. Conf. Cell. Automata Res. Ind.*, 2021, pp. 90–100.
[34] P. H. Lee, Y. Chen, S. C. Pei, and Y. Y. Chen, "Evidence of the correlation between positive Lyapunov exponents and good chaotic random number sequences," *Comput. Phys. Commun.*, vol. 160, no. 3, pp. 187–203, 2004.
[35] B. M. Alshammari, R. Guesmi, T. Guesmi, H. Alsaif, and A. Alzamil, "Implementing a symmetric lightweight cryptosystem in highly constrained IoT devices by using a chaotic S-box," *Symmetry*, vol. 13, no. 1, p. 129, 2021.
[36] E. N. Lorenz, "Deterministic nonperiodic flow," *J. Atmos. Sci.*, vol. 20, no. 2, pp. 130–141, 1963.
[37] S. Zhu, X. Deng, W. Zhang, and C. Zhu, "A new one-dimensional compound chaotic system and its application in high-speed image encryption," *Appl. Sci.*, vol. 11, no. 23, 2021, Art. no. 11206.
[38] B. Schneier and P. Sutherland, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed. Hoboken, NJ, USA: Wiley, 1995.
[39] J. Zheng and J. Li, "MARC: Modified ARC4," in *Foundations and Practice of Security*, J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, A. Miri, and N. Tawbi, Eds. Berlin, Germany: Springer-Verlag, 2013, pp. 33–44.
[40] S. K. Mousavi, A. Ghaffari, S. Besharat, and H. Afshari, "Security of Internet of Things using RC4 and ECC algorithms (case study: Smart irrigation systems)," *Wireless Pers. Commun.*, vol. 116, pp. 1713–1742, Feb. 2021.
[41] "Data encryption standard DES." Accessed: Apr. 1, 2024. [Online]. Available: https://csrc.nist.gov/CSRC/media/Publications/fips/46/archive/1977-01-15/documents/NBS.FIPS.46.pdf
[42] A. Poschmann, G. Leander, K. Schramm, and C. Paar, "New light-weight Crypto algorithms for RFID," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, New Orleans, LA, USA, 2007, pp. 1843–1846.
[43] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK families of lightweight block ciphers," Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2013/404, 2024. [Online]. Available: https://eprint.iacr.org/2013/404
[44] C. E. Shannon, "Communication theory of secrecy systems," *Bell Syst. Tech. J.*, vol. 28, no. 4, pp. 656–715, 1949.
[45] T. Farah, R. Rhouma, and S. Belghith, "A novel method for designing S-box based on chaotic map and teaching–learning-based optimization," *Nonlinear Dyn.*, vol. 88, pp. 1059–1074, Apr. 2017.
[46] F. Artuğer and F. Özkaynak, "A method for generation of substitution box based on random selection," *Egypt. Informat. J.*, vol. 23, pp. 127–135, Mar. 2022.
[47] F. U. Islam and G. Liu, "Designing S-box based on 4D-4Wing hyperchaotic system," *3-D Res.*, vol. 8, no. 1, p. 9, 2017.
[48] "Data base images." Accessed: Mar. 2, 2024. [Online]. Available: https://ccia.ugr.es/cvg/dbimagenes/
[49] V. Vujović and M. Maksimović, "Raspberry Pi as a wireless sensor node: Performances and constraints," in *Proc. 37th Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, 2014, pp. 1013–1018.
[50] H. Wang, Y. Wu, Q. Ni, and W. Liu, "A novel wireless leaf area index sensor based on a combined U-net deep learning model," *IEEE Sensors J.*, vol. 22, no. 16, pp. 16573–16585, Aug. 2022.
[51] S. A. Deepthi, E. S. Rao, and M. N. Giriprasad, "Secure MRI brain image transmission using IOT devices based on hybrid autoencoder and restricted Boltzmann approach," *J. Sensors*, vol. 2022, pp. 1–11, May 2022.
[52] O. Elijah, S. Rahim, M. Musa, Y. Salihu, M. Bello, and M.-Y. Sani, "Development of LoRa-Sigfox IoT device for long distance applications," in *Proc. 4th Int. Conf. Disrupt. Technol. Sustain. Develop.*, 2022, pp. 1–5.
[53] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. 4th Int. Symp. Inf. Process. Sens. Netw.*, 2005, pp. 364–369.
[54] P. Levis and D. Gay, *TinyOS Programming*, 1st ed. Cambridge, U.K.: Cambridge Univ., 2009.
[55] R. Randriatsiferana, F. Alicalapa, R. Lorion, L. Rajaoarisoa, B. Ravelo, and C. Moy, "Energy modeling based on power profiling of wireless sensor device," *IEEE Sensors J.*, vol. 22, no. 23, pp. 22754–22769, Dec. 2022.
[56] J. M. Castillo-Secilla, P. C. Aranda, F. J. B. Outeiriño, and J. Olivares, "Experimental procedure for the characterization and optimization of the power consumption and reliability in ZigBee mesh networks," in *Proc. 3rd Int. Conf. Adv. Mesh Netw.*, 2010, pp. 13–16.
[57] K. Phasinam, T. Kassanuk, and D. M. Shabaz, "Applicability of Internet of Things in smart farming," *J. Food Qual.*, vol. 2022, pp. 1–7, Feb. 2022.
[58] S. W. Bray, *Implementing Cryptography Using Python*. Hoboken, NJ, USA: Wiley, 2020.
[59] Y. Wu, J. P. Noonan, and S. S. Agaian, "NPCR and UACI randomness tests for image encryption," *Cyber J. Multidiscipl. J. Sci. Technol., J. Sel. Areas Telecommun.*, vol. 1, no. 2, pp. 31–38, 2011.
[60] J. Walker. "A pseudorandom number sequence test program." Accessed: Mar. 4, 2024. [Online]. Available: https://www.fourmilab.ch/random/
[61] E. Barker, "Recommendation for key management, part 1: General," U.S. Dept. Comm., Nat. Inst. Stand. Technol., Gaithersburg, MA, USA, Rep. SP 800-57, 2020.
[62] *Cryptographic Mechanisms: Recommendations and Key Lengths. Version, 2023–1*, Fed. Office Inf. Secur., Bonn, Germany, 2023.
[63] V. Gopal, E. Ozturk, J. Guilford, and W. Feghali, "Choosing a CRC polynomial and associated method for fast CRC computation on intel ®processors," Intel Corp., Santa Clara, CA, USA, White Paper, 2012.
[64] A. Mughaid, A. Al-Arjan, M. Rasmi, and S. AlZu'bi, "Intelligent security in the era of AI: The key vulnerability of RC4 algorithm," in *Proc. Int. Conf. Inf. Technol. (ICIT)*, 2021, pp. 691–694.
[65] F. León-García, J. M. Palomares, and J. Olivares, "D2R-TED: Data—Domain reduction model for threshold-based event detection in sensor networks," *Sensors*, vol. 18, no. 11, p. 3806, 2018.
[66] F. León-García, F. J. Rodríguez-Lozano, J. Olivares, and J. M. Palomares, "Data communication optimization for the evaluation of multivariate conditions in distributed scenarios," *IEEE Access*, vol. 7, pp. 123473–123489, 2019.

Francisco Alcaraz-Velasco was born in Cordoba, Spain, in 1977. He received the B.Sc. degree in computer science engineering from the Universidad de Córdoba, Córdoba, Spain, in 1998, the M.Sc. degree from Universidad de Málaga, Málaga, Spain, in 2003, the master's degree in computer and network engineering from the Universidad de Sevilla, Seville, Spain, in 2011, the master's degree in communication, networks and contents management from the Universidad Nacional de Educación a Distancia, Madrid, Spain, in 2016, and the master's degree in intelligent systems from the Universidad de Córdoba in 2016.

His research interests are in the area of securing a wireless sensor networks and embedded systems.

Jose M. Palomares (Senior Member, IEEE) was born in Motril, Spain, in 1975. He received the B.Sc., M.Sc., and Ph.D., degrees in computer engineering from the Universidad de Granada, Granada, Spain, in 1996, 1998, and 2011, respectively.

Since 2000, he has been a Lecturer, an Assistant, an Associate Professor, and, currently, as a Professor with the Universidad de Córdoba, Córdoba, Spain. He is Co-Founder of the Advanced Informatics Research Group, Universidad de Córdoba. He has research interests in image and video processing, real–time systems, wireless sensor networks, Internet of Things, and computer architecture.

Dr. Palomares is a member of the ACM since 2005.

Joaquin Olivares received the B.S. and M.S. degrees in computer sciences in artificial intelligence and the M.S. degree in electronics engineering from the Universidad de Granada, Granada, Spain, in 1997, 1999, and 2003, respectively, and the Ph.D. degree was a patented chip for motion estimation in video coding designed in FPGA from the Universidad de Córdoba, Córdoba, Spain, in 2008.

From 2000 to 2001, he was a Software Architect with the Orangee Group, Rome, Italy. He is a Full Professor with the Electronic and Computer Engineering Department, Universidad de Cordoba, where he was hired in 2001, and the Founder and the Head of the Advanced Informatics Research Group. His research interests are the Internet of Things, embedded systems, computer vision, and high-performance computing.