MINAR International Journal of Applied Sciences and Technology
ISSN: 2717-8234
Article type: Research Article

Received: 19/05/2023 **Accepted:** 22/06/2023 **Published:** 01/09/2023

A LIGHTWEIGHT ENCRYPTION ALGORITHM TO SECURE IOT DEVICES

Ruah Mouad Alyas AL_AZZAWI [1]
University of MosuL, Iraq
Sufyan Salim Mahmood AL-DABBAGH[2]
University of MosuL, Iraq

Abstract

Lightweight cryptography is an effective solution for ensuring confidentiality and privacy in resource-constrained environments, particularly in Internet of Things (IoT) systems. These systems consist of a large number of IoT devices, such as sensor nodes and embedded devices that handle sensitive data and are connected via the Internet, making them susceptible to various types of attacks. Considering the constrained processing capability, memory size, and energy of these devices, this article proposes a lightweight block cipher that aims to provide security for IoT devices, with a focus on achieving efficient software implementation, particularly in devices with 32-bit processors. The structural design of the suggested algorithm is based on Type-2 Generalized Feistel Networks (GFN), which are based on four branches of 32-bit and simple key scheduling. To enhance the diffusion and strength of the GFN structure, a partial permutation layer has been added to the cipher. Moreover, to reduce the implementation cost, a simple round function has been designed using low-cost operations like Addition, XOR, and Rotation. Security of the suggested algorithm has been validated through security analysis, and the results show meeting the criteria for avalanche effect with 50.97% Key Sensitivity and 50.58% Plaintext Sensitivity, and the correlation coefficient test results indicate a weak correlation between the plaintext and ciphertext, this makes it difficult to retrieve the plaintext from the ciphertext. When designing lightweight algorithms, one of the primary objectives is to enhance performance. To evaluate the algorithm's performance, we tested its execution time, code size, memory consumption, and throughput on real IoT devices, such as ESP8266 and ESP32, as an embedded cryptographic module. Based on the experimental results and security analysis, the suggested algorithm showed a good grade of security and higher performance in encryption and decryption operations than other lightweight algorithms, which makes it capable of providing a framework for securing data in the restricted IoT environment.

Keywords: *Block Cipher, IoT Devices, Lightweight Cryptography, Performance Evaluation, Security.*

http://dx.doi.org/10.47832/2717-8234.16.3

[1] ruaa.moayad@uomosul.edu.iq
[2] drsufyan.salim@uomosul.edu.iq

Introduction

With the large deployment of resource-constrained devices in IoT applications, an important research challenge is providing security services in an environment with limited resources. The term "Internet of Things" refers to all devices that connect to the Internet, either directly or through other devices. Several of these devices are often used within networks, such as Wireless Sensor Networks (WSN) (Pereira et al., 2020). The authors (Lachner & Dustdar, 2019) define resource-constrained IoT devices as devices with low processing capability but have built-in Ethernet or WiFi abilities and a USB interface that enables easy programming, providing a clear distinction between devices often utilized in WSN. These devices have many limitations, not only in terms of computational power but also in the available power that may depend on batteries or energy collection methods. Although these limitations, these devices are commonly utilized in many fields such as Smart Cities, Smart Industry, Smart Health, and many applications (Sutar & Mekala, 2022) because of their low cost or smaller space requirements. IoT systems are spreading and growing more common in our everyday life, where many devices collaborate to sense, analyze and store various types of data. This presents several security and privacy issues, particularly when dealing with sensitive data.

Given the presence of unsecured network services with the ease of physical access to the end nodes devices like sensor nodes and RFID tags, as well as the possibility of hacking the devices' web interfaces, the requirements for securing these devices are an important issue. Security mechanisms ensure the availability of an IoT ecosystem's services. Physical devices or end nodes are the primary attack surface for attackers in IoT systems (Noor & Hassan, 2019). One of these security mechanisms is encryption. The IoT ecosystem's node devices as they were manufactured have many restrictions such as lower computation capabilities, limited battery charge, and limited storage, making embedding conventional encryption not suitable. Lightweight cryptography addresses these limitations and has gained popularity in this domain (Thakor et al., 2021). According to NIST, Lightweight cryptography (LWC) is a cryptosystem with characteristics that have been designed to satisfy the needs of devices with various limitations, particularly those with limited resources (McKay et al., 2017). For cryptographic applications like data privacy, integrity, and confidentiality, a block cipher is used as a crucial primitive and ensures end-to-end security. Over the past few years, there has been a lot of interest in the study of designing and evaluating Lightweight Block Ciphers (LBC) (Hatzivasilis et al., 2018).

Several LBCs were suggested throughout the years in the literature, regarding their structure, block ciphers can be classified into: Substitution Permutation Networks (SPN), Feistel Networks (FN), Generalized Feistel network (GFN), ARX (Addition, Rotation, and XOR), and hybrid. Although many LBCs have been proposed, each algorithm differs from the other in terms of its suitability for a specific platform in terms of cost and performance, its difference from one application to another, and the level of security required. Table 1 summarizes the basic cryptographic primitives of various LBC algorithms with different structures.

Table 1

Cryptographic Primitives of LBC

Algorithm	**Key size in bits**	**Block size in bits**	**Rounds**	**Inner Structure**	**Target Environment**
AES (Daemen & Rijmen, 1998)	128/192/256	128	10/12/14	SPN	H.W, S.W
HIGHT (Hong et al., 2006)	128	64	32	GFN +ARX	H.W, S.W
Piccolo (Shibutani et al., 2011)	80/128	64	25/30	GFN	H.W
Chaskey (Mouha et al., 2014)	128	128	8	ARX	S.W
LEA (Hong et al., 2014)	128/192/ 256	128	24/28/32	GFN +ARX	H.W, S.W
SIMON (Beaulieu et al., 2015)	64/72/96/128 /144/192/256	32/48/64/ 96/128	32/36/42/44/ 52/54/68/69/ 72	FN	H.W, S.W
SPECK (Beaulieu et al., 2015)	64/72/96/128 /144/192/256	32/48/64/ 96/128	22/23/26/27/ 28/29/32/33/ 34	FN +ARX	H.W, S.W
SIMECK (Yang et al., 2015)	64/96/128	32/48/64	32/36/44	FN	H.W, S.W
RECTANGLE (Zhang et al., 2015)	80/128	64	25	SPN	H.W, S.W
RoadRunner (Baysal & Şahin, 2016)	80/128	64	10/12	FN	S.W
CHAM (Koo et al., 2017)	128/256	64/128	80/96	GFN +ARX	H.W, S.W
GIFT (Banik et al., 2017)	128	64/128	28/40	SPN	H.W, S.W
LiCi (Patil et al., 2017)	128	64	31	FN	H.W, S.W

Many studies have been presented to benchmark the performance of various LBCs (Makarenko et al., 2020) (Dinu et al., 2018). The trade-off between cost, security, and performance is the main factor that determines how a lightweight cryptography algorithm is designed. According to Shannon's definitions, to design a secure cipher, two properties must verify confusion and diffusion. Confusion means maintaining the relation between the cipher text and the secret key obscured, while diffusion means making the relationship between the cipher text and the plain text obscured. For encryption and decryption purposes, a symmetric block cipher employs the same secret key. The fixed bit-lengths (block size) of the plaintext and ciphertext are both predetermined. For a preset number of rounds, different operations are applied to these blocks in each round. Despite the high-security level of the SPN cipher, its implementation is complicated and costly due to the use of non-linear component S-boxes. In the FN cipher, a plaintext is split in half and a round function is performed to one half before the result is added to the other half. All operations in the FN structure, are symmetrical both encryption and decryption, but in reverse order, so there is no need to design a decryption function. This type of inner structure reduces half the design time for hardware implementation. Despite this advantage, FN requires more rounds to achieve strong diffusion. An improvement for Feistel structure is GFN, which divides the plaintext into K sub-blocks where K is greater than 2. The Type-2 GFN is a common form of GFN (Suzaki & Minematsu, 2010), where each pair of consecutive sub-blocks is exposed to a Feistel transformation, and then the sub-blocks are permuted. Although this type is easy to implement, its diffusion property is low, and it is also affected by the type of permutation and the value of k if it is large. Compared to the previous types of structures mentioned, the ARX structure cipher has a small RAM footprint and a smaller code size (Dinu et al., 2018). This article suggested a lightweight devices data encryption algorithm (LWDDEA) to secure data in a restricted IoT environment. The main objectives of the suggested algorithm:

1. LWDDEA is designed for software implementation, especially to be suitable for execution on devices with 32-bit processors and limited memory capabilities. The proposed algorithm can be used to design a security framework for an application that utilizes devices with limited resources and ensures end-to-end data security from the deployment point through data storage.
2. LWDDEA has a GFN structure. This type of construction will be an asset in designing a lightweight round function where the encryption process can be converted to a decryption process in a simple manner and at minimum cost.
3. The balance between higher security and higher implementation can be achieved by using an iterative block cipher structure with a configurable number of rounds and simple ARX operations in each round. To satisfy confusion, modular addition is used to archive non-linearity. For linearity, a partial permutation layer was added to improve the diffusion property of GFN.

4. The fundamental computations operate on complete words of data at once, the word length allows the algorithm to be adaptable to different processors. The overall structure supports the algorithm being adaptive to many types of processors, the 128-bit block size can be split into different word sizes, which enhances the performance.

5. LWDDEA based on simple key scheduling with key sizes of 128 bits provides the algorithm resistance to thwart brute-force attacks and MITM attacks, in addition to using key whitening to thwart weak key attacks.

The article is structured in the following order: Section 2 addresses current research related to the design and performance evaluation of LBC algorithms. Section 3 introduces the methodology used for designing and specification the suggested algorithm, and Section 4 demonstrates the security criteria used to assess the security of the suggested algorithm. Section 5 will clarify the performance evaluation of the suggested algorithm and list the benchmarking metrics that will be used in this evaluation to compare the performance results of the suggested algorithm with other lightweight algorithms. The software and hardware used in the evaluation experiments are also mentioned in this section. Lastly, Section 6 showed the conclusion.

2. Related Works

This section discusses papers related to the design of lightweight block ciphers. The papers take two directions either adapting existing cryptographic algorithms to meet the requirements of resource-constrained devices or designing new lightweight block cipher algorithms specifically for the constrained environment. In 2017, (Koo et al., 2017), suggested a new LBC called CHAM based on the ARX operations was introduced for optimal hardware and software performance. CHAM family consists of three cipher standards, CHAM-64/128, CHAM-128/128 with 80 rounds, and CHAM-128/256 with 96 rounds. CHAM cipher is designed to get a higher degree of effectiveness than SIMON and SPECK in constrained environments. In 2021, (Thabit et al., 2021), introduced a new LBC named NLCA to enhance data confidentiality in the cloud computing environment. NLCA structure follows a hybrid of FN and SPN. For data transmission security in cloud services, they use 128-bit for block size and 128-bit for key size, Many operations including XOR, XNOR, AND, OR left shift, substitution with S-boxes, and swaps, are used in each round to complicate the encrypted data. Several suggestions for enhancing the lightweight block ciphers with Feistel structure have demonstrated exemplary performance. A new version of Speck called Speck-R has been suggested (Sleem & Couturier, 2020), it is based on a hybrid structure that combines the ARX structure with a layer of dynamic substitution to strengthen the security level, this layer varies depending on the dynamic key, Speck-R decreased the Speck algorithm's rounds from 26 to 7 which reduce the execution time. Slim cipher (Aboushosha et al., 2020) adopts a conventional Feistel structure that uses 4-bit S-Boxes to achieve confusion in the round function, resulting in reasonable security and superior hardware performance. Despite the improvements in the cipher structure, the need to minimize performance drawbacks through appropriate analysis of resource-constrained devices should be emphasized. Seok and Lee (2019) proposed a method known as two-way modular arithmetic to enhance the efficiency of executing a lightweight block cipher with an ARX structure. this technique improved the performance of SPARX 64/128 and CHAM 64/128 lightweight algorithms on a 32-bit processor. Another important aspect of designing lightweight cryptographic algorithms is related to evaluating the performance on different platforms. On the performance evaluation of LBC, different approaches for security and performance evaluation of LBC based on restricted environment (platform), and target applications have been discussed and show their experimental results, in 2020,(Makarenko et al., 2020), the authors compare the throughput, using energy and, the utilized RAM and ROM of several selected lightweight block ciphers and conventional block ciphers (AES, CLEFIA, DES, Triple DES, TEA, XTEA, IDEA, PRESENT, SEA, SPECK, and TWOFISH) to find the most suitable cryptographic schema for IoT devices. Experimental results were obtained using the Cooja simulator using z1 motes using MSP430F2617 microcontroller (16-bit) architecture. In comparison to the other algorithms, it has been found that PRESENT is considerably less performance and SPECK cipher shows better results in all metrics, and the software performance of lightweight encryption algorithms can be affected by bit-level operations and increasing the number of rounds. Due to resource-constrained devices relying on low-end microcontrollers, Software optimization techniques are necessary to improve low-level performance. In 2021, (Panahi et al., 2021) compare the evaluation metrics: RAM and ROM consumption, execution time, throughput, and energy consumption for ten LBCs: AES, PRESENT, LBlock, Skipjack, SIMON, XTEA, PRINCE, Piccolo, HIGHT, and RECTANGLE by using Raspberry Pi 3(64-bit ARM Cortex processor) and Arduino Mega 2560 (ATmega2560 8-bit microcontroller) as a restricted environment for IoT applications to find the most suitable lightweight algorithm for this IoT devices.

3. Specification of the Proposed LWDDEA Algorithm

LWDDEA is a lightweight block cipher that utilizes a key of 128-bit, block size of 128-bit, and 16 rounds. The fundamental structure for the Proposed Algorithm is a Type-2 GFN with four branches of 32- bits. For explanation, Table 2 lists the notations used for the suggested algorithm and Figure 1 shows the structure of the suggested algorithm.

Table 2

The proposed LWDDEA notations Description

Symbol	Description
W	Word size (bits)
R	Round number
Mk	Master Key
Wk	Whitening key
Pt	Input Plaintext block 128-bits
Ct	Ciphertext output block 128-bits
Pt_i	sub-block Plaintext 32-bits
Ct_i	sub-block Ciphertext 32-bits
Sk_j	32-bit subkeys or round keys for j_{th} round
$+$	addition modulo 2^w
$-$	subtraction modulo 2^w
$\times$	multiplication modulo 2^w
$\oplus$	Bitwise exclusive – OR
$\ll$	Shift left
a, b	W-bit variables
$a \lll b$	Cyclic rotation **a** to the left by a given amount of **b**
$a \ggg b$	Cyclic rotation **a** to the right by a given amount of **b**
$a \parallel b$	concatenation **a** with **b**

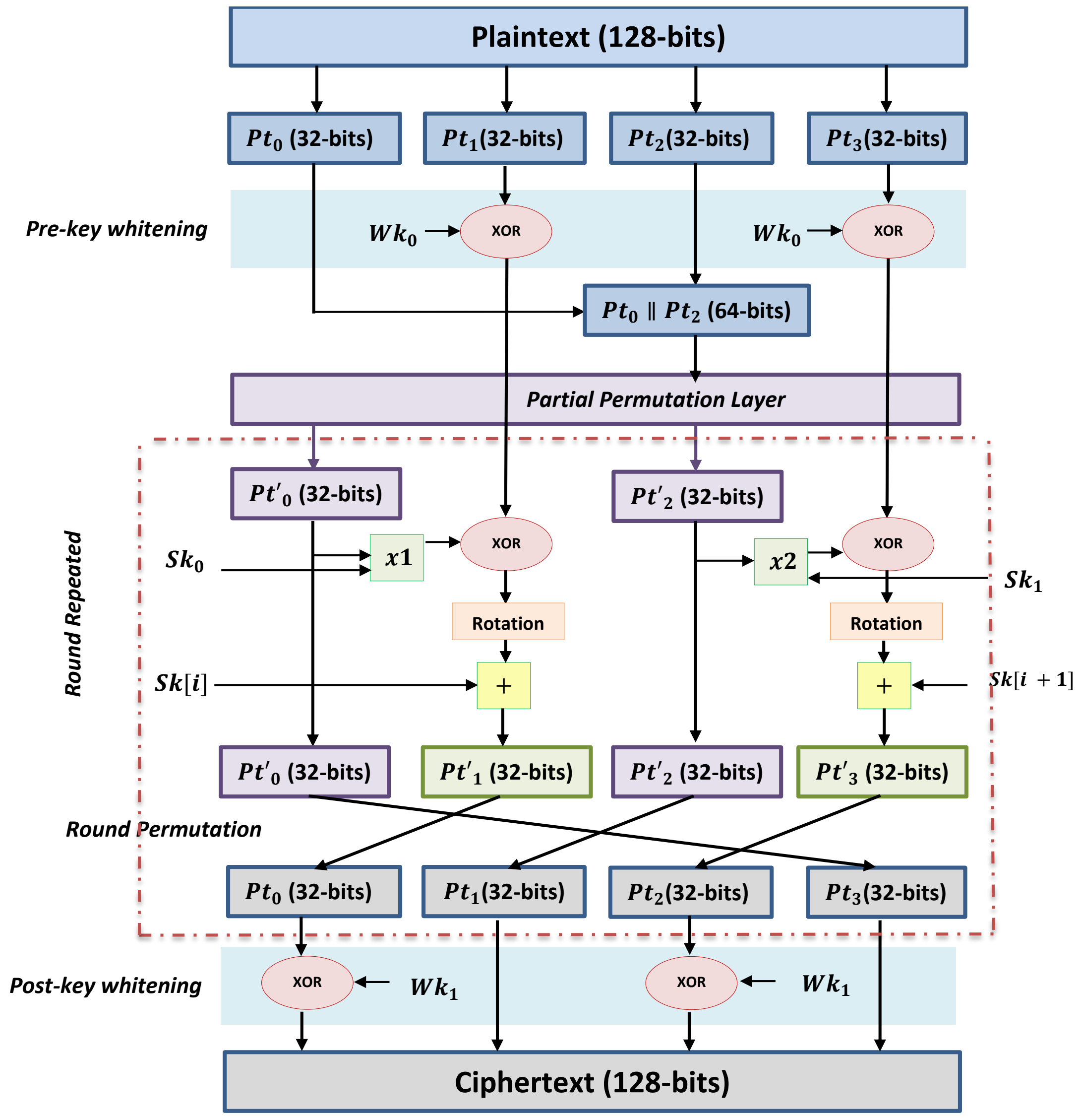

Figure 1. The proposed Encryption algorithm

3.1 SubKeys Generation

The SubKeys Generation algorithm included two parts, firstly, whitening keys $(\mathrm{Wk}_0, \mathrm{Wk}_1)$ 32 bits are extracted from the user-supplied master key (Mk) of 128-bit, and these two keys will be used in pre-key whitening and post-key whitening steps. The second part is the Key generation process, where 34 subkeys Sk_0_Sk_{33} of 32 bits are derived from the master key (Mk) that is utilized for the encryption and decryption processes. The SubKeys Generation algorithm is explained in Algorithm 1. The value of $c3$ represented the shift-left amount parameter and was determined by the five lower-order bits from the part of the master key Mk_2 .

Algorithm 1 SubKeys Generation algorithm

```
Input: 128-bits Master Key $(Mk)$
Output: 32-bit subkeys $[Sk_0, Sk_1, Sk_2, \dots, Sk_{33}]$
1:    Partitioning 128-bits Master Key $(Mk)$ into 4 equal parts
$(Mk_0, Mk_1, Mk_2, Mk_3) \leftarrow$ Master Key $(Mk)$
2:    Extract 32-bit whitening key $Wk_0$, $Wk_1$,
$Wk_0 = Mk_0$, $Wk_1 = Mk_2$
3:    Generate 32-bit subkeys $[Sk_0, Sk_1, Sk_2, \dots, Sk_{33}]$
4:    $for\ j = 1\ \ to\ \ (2R + 2)\ \ do$
5:    $A = Mk_0 \ll c3$
6:    $B = (Mk_2 + Mk_3) + 1$
7:    $C = (Mk_0 + Mk_1) + 3$
8:    $Mk\,[j \bmod 4] = C + (A \oplus B)$
9:    $Sk\,[j] = Mk[j \bmod 4]$
10:   End for
11:   return subkeys $[Sk_0, Sk_1, Sk_2, \dots, Sk_{33}]$
```

3.2 The Proposed Encryption Algorithm

In the encryption algorithm, firstly, a plaintext Pt of 128 bits is loaded into an internal state of 128-bit to be utilized in the process of encrypting which produces a ciphertext Ct of 128-bit that is loaded into an internal state of 128-bit to be utilized for decryption process, where $\mathrm{Pt}, \mathrm{Ct} \in \{1,0\}^{128}$. The internal state of the plaintext Pt is divided into four sub-block with equal size, $\mathrm{Pt}_i, \mathrm{Ct}_i \in \{1,0\}^{32}$ where $(0 \le i < 4)$, such that $\mathrm{Pt} = \mathrm{Pt}_0 \parallel \mathrm{Pt}_1 \parallel \mathrm{Pt}_2 \parallel \mathrm{Pt}_3$, also, the internal state of the ciphertext Ct is divided into four sub-block- such that $\mathrm{Ct} = \mathrm{Ct}_0 \parallel \mathrm{Ct}_1 \parallel \mathrm{Ct}_2 \parallel \mathrm{Ct}_3$ for the decryption process. For implementation on a 32-bit processor, the word size is equal to 32 bits. The master key (Mk) is 128-bit and for each round R, there are $\mathrm{Sk}_j \in \{1,0\}^{32} (0 \le j < 2R + 2)$ subkeys or round keys extracted from Mk by the SubKeys Generation algorithm. $\mathrm{Wk}_0, \mathrm{Wk}_1$ are extracted from Mk and subsequently used as whitening keys in the pre-key whitening step and post-key whitening step, respectively. The Key whitening does not offer protection against analytical attacks, but it can be used to improve security by strengthening a cipher key space. The addition of these two steps helps to disguise the exposed from the plaintext, which was input to the first round and also disguises the exposed from the ciphertext, which was input to the last round.

To increase the security of the GFN structure a diffusion layer is added. The partial permutation layer is used to improve the security against cryptanalysis by utilizing an 8-bit word instead of a 32-bit word to unbind the 32-bit word structure. As shown in Figure 2 first, Pt_0 and Pt_2 are concatenated to constructed 64-bit block and then, partitioned into 8-bit words $(x_0 \parallel x_1 \parallel x_2 \parallel x_3 \parallel \cdots \parallel x_7)$, after that permutation is done similarly to (Shibutani et al., 2011). Finally, the new values of Pt_0, Pt_2 are combined again with Pt_1, Pt_3 to enter the round function.

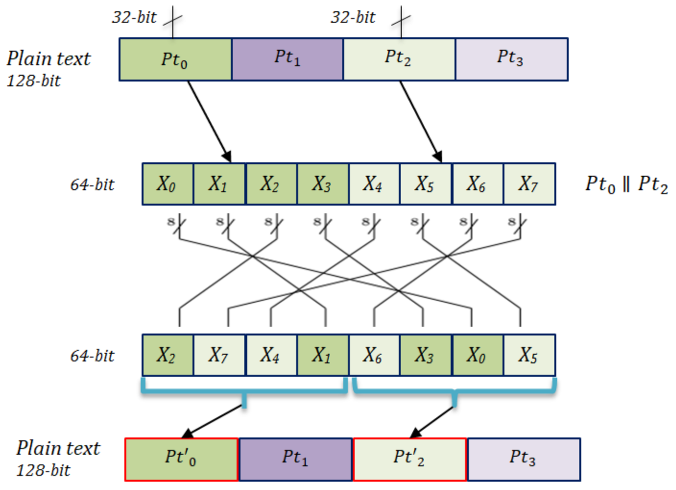

Figure 2. Partial Permutation layer

Algorithm 2 describes the pseudo-code of the LWDDEA encryption. The values of $c1$ and $c2$ represent the cyclic rotation amount parameters and are determined by the five lower-order bits from the results of $x1$ and $x2$ respectively. In the decryption process, all steps are similar to the encryption process but are performed in reverse order. Subtraction modulo 2^w is used instead of addition modulo 2^w operations in the decryption process, in addition, the sequence in which the whitening keys are used in the decryption process is inverted.

Algorithm 2 The proposed LWDDEA Encryption

```
Input: Plaintext 128-bits, 32-bit subkeys $[Sk_0, Sk_1, Sk_2, \dots, Sk_{33}]$
Output: 128-bits Ciphertext
1:   partitioning 128-bit plaintext Pt into 4 equal parts
2:   $(Pt_0, Pt_1, Pt_2, Pt_3) \leftarrow$ plaintext(Pt)
3:   Pre-key whitening:
4:   $Pt_1 = Pt_1 \oplus Wk_0$
5:   $Pt_3 = Pt_3 \oplus Wk_0$
6:   $Pt'_0, Pt'_2 \leftarrow$ Partial Permutation ( $Pt_0 \parallel Pt_2$)
7:   for i = 2 to 2R do
8:   $x1 = \left(Pt'_0 \times ((Sk_0 \oplus Pt'_0) + 2)\right) \ggg 5$
9:   $x2 = \left(Pt'_2 \times ((Sk_1 \oplus Pt'_2) + 2)\right) \ggg 5$
10:  $Pt'_1 = \left((Pt_1 \oplus x1) \ggg c2\right) + Sk[i]$
11:  $Pt'_3 = \left((Pt_3 \oplus x2) \ggg c1\right) + Sk[i+1]$
12:  Perform Round Permutation:
13:  $Pt_0 = Pt'_1$
14:  $Pt_1 = Pt'_2$
15:  $Pt_2 = Pt'_3$
16:  $Pt_3 = Pt'_0$
17:  i=i+2
18:  Post-key whitening
19:  $Pt_0 = Pt_0 \oplus Wk_1$
20:  $Pt_2 = Pt_2 \oplus Wk_1$
21:  return Ciphertext (Ct)$\leftarrow (Pt_0 \parallel Pt_1 \parallel Pt_2 \parallel Pt_3)$
```

4. Security Criteria

This section demonstrates the simulations used to evaluate the security of the suggested LWDDEA algorithm. To meet confusion and diffusion properties, the produced ciphertext is evaluated based on the following security criteria :

4.1 Avalanche Test

This test is used to estimate the strength of the cipher against attacks such as brute force attacks. If the average of 50% of the output bits differs from one input bit change in the plain text or the secret key, then the avalanche effect is satisfying optimally. The state, when the avalanche effect is unsatisfied, refers to poor randomization for the cipher. This test has been applied in measuring the security of block ciphers (Encarnacion, 2020), the following equation is used to calculate the avalanche effect:

$$\text{Avalanche } \% = \frac{\text{no.of the changing bits in the ciphertext}}{\text{no.of bits in the ciphertext}} * 100 \quad (1)$$

Two different methods are used in this evaluation (Banani et al., 2021) (Zakaria et al., 2022), the plaintext sensitivity test and the key sensitivity test. To evaluate the produced ciphertext, the plaintext sensitivity test is used. The master key remains fixed, whereas the plaintext undergoes a one-bit flip. The key sensitivity test measures how much the ciphertext changes due to the key's single-bit change. It can be explained by the fact that if the key sensitivity is close to the optimal value, then the change in the ciphertext will be close to 50%. An algorithm is considered key-sensitive if it is impossible to retrieve the original data when the key differs only a bit from the original key.

4.2 Correlation Coefficient Test

This test is used to examine the non-linear feature. By using the cipher text generated from specific input data, the correlation coefficient R is calculated from equation (2), (Zakaria et al., 2022).

$$R = \frac{\sum_{I=1}^{L}(pt_i - AE)(ct_i - AE)}{\sqrt{\sum_{I=1}^{L}(pt_i - AE)^2 \sum_{I=1}^{L}(ct_i - AE)^2}} \quad (2)$$

Where L is the length of the plaintext or the ciphertext. pt_i and ct_i are the i^{th} places of the bits in *Pt* and *Ct*, respectively, and *i* ranging from (0 . . . 128). AE is the avalanche effect between the cipher text and the plain text and is calculated according to equation (1). The acceptable result ranges for the correlation coefficients, which have values between -1 and +1, are shown in Table 3 (Zakaria et al., 2022).

Table 3

Results Indicators for the Correlation Coefficient (R)

Conditions	Correlation Coefficient(R) Results Indicators
R = 0	Non-linear relation
(0 < R ≤ 0.3) or (-0.3 ≤ R < 0)	Weakly (positive/negative) linear relation
(0.3 < R < 0.7) or (-0.7 < R < -0.3)	Medium (positive/negative) linear relation
(0.7 ≤ R < 1) or (-1 < R ≤ -0.7)	Highly (positive/negative) linear relation
R = ±1	Strongly (positive/negative) linear relation

4.5 Security Analysis Results

For the plaintext sensitivity test, a random 16-byte plaintext input encrypted with a 16-byte fixed key is used. By flipping one bit at a time, the plaintext was used to make several states, and the avalanche effect between these states was then calculated. Table 4 explains the outcomes of the diffusion range and average diffusion for a different state when one bit is flipped in plaintext. From the results in Table 4, more than half of the ciphertext bits are modified, resulting in an average avalanche effect of 50.58 percent for the proposed algorithm.

In the key sensitivity test, a random 16-byte key input is used to encrypt 16 bytes of fixed plaintext. By flipping one bit at a time, the key was used to make several states, and the avalanche effect between these states was then calculated. The outcomes of the key sensitivity test for the proposed algorithm are shown in Table 5. It shows that the algorithm achieves the criteria related to key sensitivity. The results summarize the diffusion range and average diffusion of 50.97 percent for the proposed algorithm when the plaintext is kept constant and the secret key undergoes a change by flipping one bit.

Table 4

Avalanche effect using variable single-bit flipped plaintext with fixed secret key = $(000102030405060708090a0b0c\ 0d0e0f)_{16}$

State Number	Plaintext (Hexadecimal)	Ciphertext (Binary) 128-bit
1	**00000000000000000000000000000000**	10110011010000011000101100001001010001010 01000101000101110011110100001111000100011 11000100011101101100001010000001100001000 11111
2	**00000000000000000000000000000001**	01000010001110010110001111000011110100100 00011001111101101101100010010000111010100 01110001000111101010000000011110010101000 10010

3.	**00000000000000000000000000000002**	01010111111100111111010000111010101101001
		01000000110001101011101100100010001011000 1
		01000000101111011111010111110000111000101 1
		01011

4	**00000000000000000000000000000003**	00001110101010101001010101110111001100100 01100101011100100111110110111000000100010 11100101111000010111000011001011101110110 00001
5	**00000000000000000000000000000004**	01000110011010111100010000101000101001000 01110100010001110111010111111111000110110 01001011011001011110010000000011100110100 11111
6	**00000000000000000000000000000005**	11001110111010000111001011101010011011011 11111100011101111101100011001100101001101 01010001101110101101100011010011101100110 10101

7	**00000000000000000000000000000006**	01111001110101111110110101111000000100110 1
		01111110101000100101001011010101110010010
		11001011010000110111101001111010010100111
		11010

8	**00000000000000000000000000000007**	11011111100100010010101100010110110000110 0
		11100011000110111100100100101011111111010
		00000011010101001001011100010110101010000
		11011

Number of changing bits	1 with 2	1 with 3	1 with 5	2 with 4	3 with 4	4 with 8	[illegible]	6 with 8	7 with 8	2 with 6	3 with 7	5 with 6
	72	**57**	51	69	66	64	6 1	68	73	63	73	60

Diffusion Range = 51 – 73, Average diffusion = 64.75 (50.58%)

Table 5

Avalanche effect using variable single-bit flipped key with fixed plain text = $(00000000000000000000000000000000)_{16}$

State Number	key (Hexadecimal)	Ciphertext (Binary) 128-bit
1	**001122334455667788992a2b2c2d2e2f**	1110001100011011110011001011101101011001001100011011001010000001010111101001010100100100100110101111000110001100011010101010011110
2	**011122334455667788992a2b2c2d2e2f**	1111011110101000100001101000111100101111010110111110000010010011011011010001101001011110100011101010010011101101101 00000010100000
3	**021122334455667788992a2b2c2d2e2f**	01010001100111100011011100111110000111101100111011111000001100110001110001011011011110010101000010001101001000111101000010100 1101
4	**031122334455667788992a2b2c2d2e2f**	11011111101100100100110010100011100100101000000110000101011100010111011001011001111001101100011010100001000111100011 01000110101101
5	**041122334455667788992a2b2c2d2e2f**	10001110000000100111010101000100100010011101000011011000110010001010111111011101101110100111010111000010001011111111 10000110110111
6	**051122334455667788992a2b2c2d2e2f**	01001010111101001010101010010100101000000001101111110001110011000100100100101100001000001011111011011100010110001011 11101011010001
7	**061122334455667788992a2b2c2d2e2f**	10010101011000011001100100100011001110101100100011000000011001001010010100110011001110011101110101010010101100000101101101110011
8	**071122334455667788992a2b2c2d2e2f**	00001010101100101100011000101111011010000010001101111111001000101100101011101001001100000000101010001101110001100111 11100110011111

Number of changing bits	1 with 2	1 with 3	1 with 5	2 with 4	3 with 4	4 with 8	5 with 7	6 with 8	7 with 8	2 with 6	3 with 7	5 with 6
	62	75	65	57	72	62	70	58	69	67	65	61

Diffusion Range = 57 – 75 , Average diffusion = 65.25 (50.97%)

For the Correlation Coefficient test, 1000 randomly generated plaintexts and 5 random keys generated have been used to evaluate the Correlation Coefficient. Analysis results of the Correlation Coefficient based on plaintext and ciphertext are explained in scatter charts as shown in Figure 3. A set of 1000 plaintexts is encrypted using one of the random keys, and the resulting R-value is classed according to how the plaintext correlates with the ciphertext. Table 6 classifies the results of R-values into ranges and shows the average value of the five groups. The results indicate that the majority of the R-values vary from 0 to 0.3 and -0.3 to 0, referring to a weak linear relation between the plaintext and ciphertext.

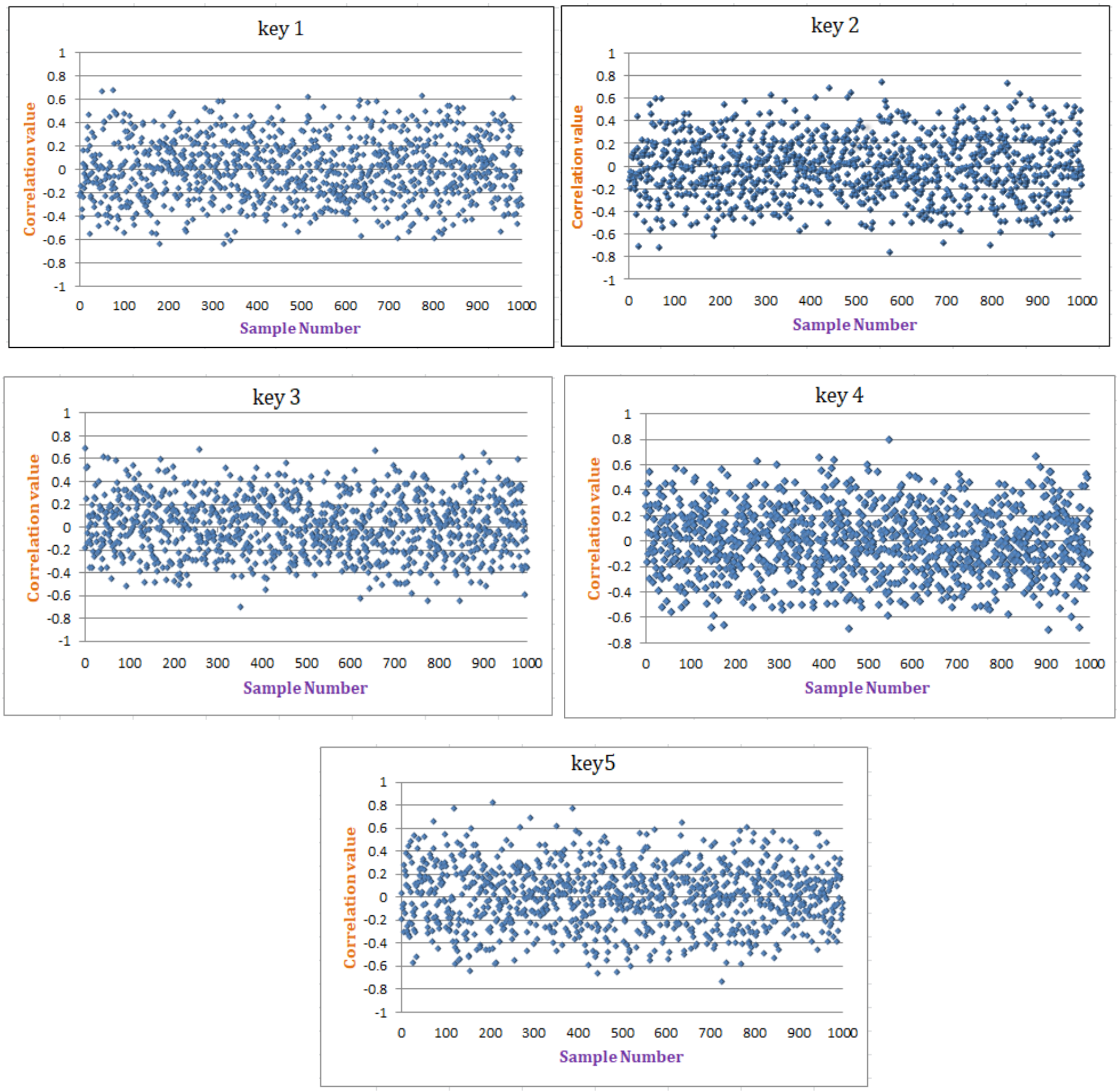

Figure 3. Correlation Coefficient analysis

Table 6

Correlation Coefficient analysis outcomes between plaintext and ciphertext

Random Key	$R=0$	$0 < R \leq 0.3$	$-0.3 \leq R < 0$	$0.3 < R \leq 0.7$	$-0.7 \leq R < -0.3$	$0.7 < R < 1$	$-0.7 < R < -1$	$R=1$ or $R=-1$
Key1	13	424	440	70	53	0	0	0
Key2	13	423	426	66	72	0	0	0
Key3	12	451	432	59	46	0	0	0
Key4	10	421	427	63	78	1	0	0
Key5	12	447	404	65	71	1	0	0
Average	**1.2**	**43.32**	**42.58**	**6.46**	**6.4**	**0.04**	**0**	**0**

5. Performance Evaluation Results

For Performance evaluation, we chose two widely used embedded devices, NodeMCU ESP8266, and WROOM-32 ESP32, to implement and evaluate the suggested algorithm's performance, these devices have been employed as microcontrollers for sensor nodes in different IoT applications (Caraveo-Cacep et al., 2023). Table 7 lists some of the main specifications of these devices, the two devices have a 32-bit processor that is compatible with 32-bit word size, although the ESP32 has dual processing cores, we only used one since most lightweight cryptographic algorithms are built for a single core. The proposed algorithm has been compiled in the Arduino development environment by using ESP32 Arduino Core and ESP8266 Arduino Core boards.

Researches on lightweight cryptographic algorithms take three directions: hardware, software, and hardware/software implementations (Hatzivasilis et al., 2017). According to the implementation method hardware or software, several metrics are utilized to evaluate the performance of lightweight algorithms. Since the suggested algorithm is targeted at software implementation, we will concentrate on software implementation only. Among the primary objectives of software implementations, is to maintain RAM/ROM and CPU requirements as minimal as possible to reduce power consumption and the cost of the devices. Therefore, good performance depends on decreasing the number of CPU cycles and having a small memory footprint. Accordingly, the following benchmarking criteria have been utilized in evaluating the performance of the suggested algorithm:

- **Execution Time**

In IoT applications, a lower execution time is requested, therefore, the execution time of the algorithm must be taken into consideration during the design phase, with maintaining the necessary level of security. In the valuation process, the execution time has represented the time taken by an algorithm to encrypt and decrypt a specific block of data. To measure the execution time, we take the difference between the time before and after the operations.

• **Memory usage**

An important consideration for the restricted IoT environment is the limited memory that the devices have. Many program execution errors occur when the limited memory capacity is exceeded. RAM usage represents how many bytes are required to store the intermediate values used in all operations, while ROM usage represents how many bytes are required to store the code size of the algorithm and static data.

• **Throughput**

Based on the processor's frequency, throughput is the fraction of the data size in bytes to the total execution time in bytes per second. For any encryption system, high throughput is required with maintaining an acceptable level of security.

To set a comparison between the performance of the suggested algorithm to other presented lightweight benchmarked algorithms, we chose two lightweight block ciphers, AES (Daemen & Rijmen, 1998) and SPECK (Beaulieu et al., 2015). AES is tailored for hardware and software implementations. Even though it is not a lightweight cipher, a lot of IoT devices use this cipher. SPECK is specifically optimized for software implementation. several studies presented comparisons related to the performance of lightweight encryption algorithms on different platforms and different targeted applications, the SPECK algorithm was chosen to compare its performance with the performance of our proposed algorithm as it outperformed other lightweight algorithms in terms of throughput and energy use (Makarenko et al., 2020) (Pei et al., 2018). Table 8 explains the test vectors that will be used in our performance evaluation experiments.

Table 7

Specifications of NodeMCU ESP8266 and ESP32

	NodeMCU ESP8266	**WROOM-32 ESP32**
CPU	Xtensa Single-core	Xtensa Dual-Core
Bus Width	32	32
Clock Speed (MHz)	80 MHz	240 MHz
SRAM	128 KB	512KB
Flash memory	4MB	4MB

Table 8

Test vectors for performance evaluation

Pt	646f6e746261636b7468697320776179
Mk	0f0e0d0c0b0a09080706050403020100
Ct	7a6dc9ba57f50e1708b611c00ee286d9

To test the available memory in the device, the memory utilization of the proposed algorithm was analyzed. Table 9 summarizes the memory consumption on the two target IoT devices. From the results in Table 9, the suggested algorithm consumes a small amount of the total device memory. Even the highest SRAM consuming uses only 32% of the whole SRAM, leaving enough effective space for other applications and making it convenient for its deployment in an IoT environment.

Table 9

Memory usage and the total code size comparison of the proposed algorithm on the embedded IoT devices

MCU Modules	Operations	Total Code Size (bytes)	Usage Space (%)	Used SRAM (bytes)	Usage SRAM (%)
Node MCU ESP8266	Encryption+ Key generation	264120	25%	26976	32%
	Decryption+ Key generation	264088	25%	26976	32%
	Encryption + Decryption + Key generation	264408	25%	26992	32%
ESP32	Encryption+ Key generation	199472	15%	13200	4%
	Decryption+ Key generation	199460	15%	13200	4%
	Encryption + Decryption + Key generation	199696	15%	13216	4%

To compare the results of the performance evaluation with AES and Speck algorithms in a fair way, both algorithms are compiled in the Arduino IDE and tested using the Arduino Cryptography Library (Sarker et al., 2020). We chose exclusively 128-bit encryption algorithms to match our 128-bit key size, so AES128 and Speck128 tests were chosen. Similar to our algorithm, the Electronic Codebook (ECB) mode was chosen for the AES128 and Speck128 tests due to its linearity and lower complexity. Table 10 shows the comparison results in terms of the execution time (us/byte) for the proposed algorithm's encryption, decryption, and SubKeys Generation with ASE and Speck algorithms. The results from Table 10 show that key setup time takes the largest amount of time compared to other operations. Furthermore, the times required for encryption and decryption are close together in both devices because the same operations are used but in reverse order and operations only involve static-length data. Figure 4 and Figure 5 compare the key setup, encryption, and decryption times between the algorithms on the ESP8266 and ESP32, respectively. The results of the implementation on the ESP32 showed that the algorithm required more time in the key generation phase compared to the AES algorithm, but it overcame the encryption and decoding times, this is due to the dedicated piece of hardware cryptography accelerator inside the ESP32 chip. Compared to the key setup of the Speck algorithm, the proposed algorithm showed a lower execution time on both devices. In both Figure 4 and Figure 5, we can see that the encryption and decryption times of the suggested algorithm are faster than the ASE and Speck algorithms on both MCUs and more quickly on the ESP32 microcontroller. The minimum execution time consumed was 0.14 us for encryption and decryption operations on the ESP32 MCU, this is a very small amount of time that can be consumed in a constrained IoT environment.

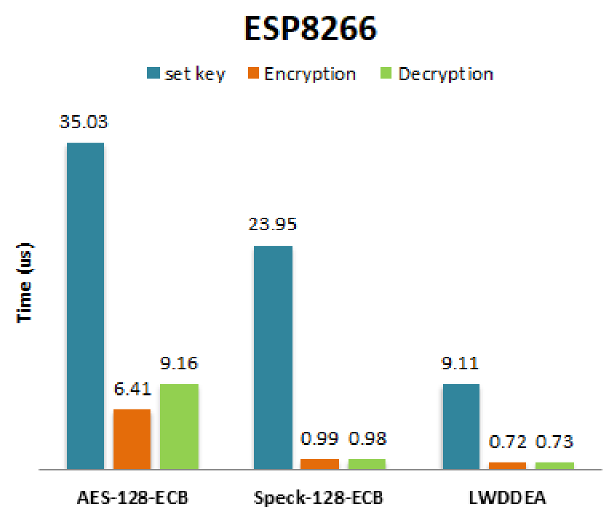

Figure 4. Run-time performance evaluation of AES, SPECK, and the proposed LWDDEA algorithm on NodeMCU ESP8266

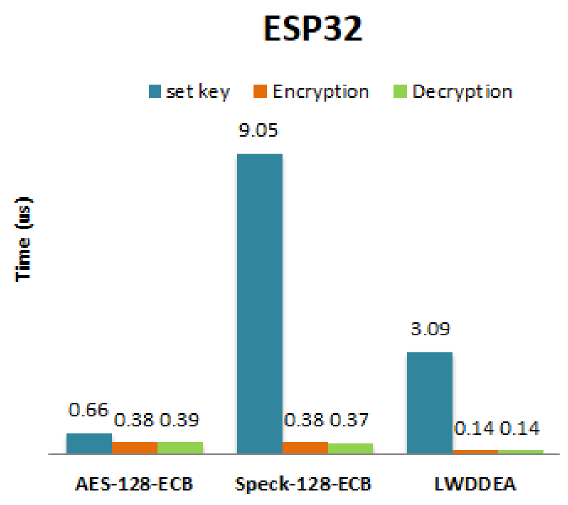

Figure 5. Run-time performance evaluation of AES, SPECK, and the proposed LWDDEA algorithm on ESP32

Table 10

Implementation comparison regarding execution time (us per byte) and throughput (bytes per second) on embedded IoT devices

Operation	Algorithm	ESP8266	ESP32
Key setup Time *(us)*	AES-128-ECB	35.03	0.65
	Speck-128-ECB	23.96	9.05
	LWDDEA	9.15	3.09
Encryption Time *(us/byte)*	AES-128-ECB	6.41	0.38
	Speck-128-ECB	0.99	0.38
	LWDDEA	0.72	0.14
Decryption Time *(us/byte)*	AES-128-ECB	9.16	0.39
	Speck-128-ECB	0.98	0.37
	LWDDEA	0.73	0.14
Encryption Throughput *(byte/second)*	AES-128-ECB	156,064.98	2646,465.31
	Speck-128-ECB	1006,200.71	2645,677.62
	LWDDEA	1387,612.09	7080,272.59
Encryption Throughput *(block/second)*	AES-128-ECB	9,754	165,404
	Speck-128-ECB	62,887	165,355
	LWDDEA	86,726	442,517

6. Conclusion

This paper suggests a cryptographic security solution related to the constrained IoT environment. Given the IoT devices' resource restrictions, which include memory utilization, processor speed, and cost, conventional cryptographic techniques are inappropriate for the IoT environment. A lightweight encryption algorithm was introduced to be compatible with the limited resources of IoT devices. The algorithm is based on a combination of Type-2 GFN and ARX structures. To enhance the strength of Type-2 GFN, a diffusion layer was added to the inner structure. The round function was based on low-cost ARX operations to give the algorithm fast encryption and decryption with low memory consumption. According to the outcomes of the security analysis, the algorithm meets the avalanche effect criteria and passes the plain sensitivity and key sensitivity tests. For the correlation coefficient test, the algorithm produced R-values ranging from 0 to 0.3 in the positive direction and from 0.3 to 0 in the negative direction, these values indicated a weak correlation between the plaintext and ciphertext. The performance was evaluated and compared to AES and SPECK algorithms on two real IoT devices with 32-bit processors. The results of the evaluation showed small execution time and high throughput compared to other algorithms. The results also showed that the proposed algorithm has small memory consumption, commensurate with the limited resources of these devices. The overall structure supports the algorithm's ability to be adaptive to many types of processors, where the 128-bit block can be divided into different word sizes. For further research, it is suggested to adapt the word length to other processors with 8-bit or 16-bit processors, which enhance the performance.

REFERENCES

Aboushosha, B., Ramadan, R. A., Dwivedi, A. D., El-Sayed, A., & Dessouky, M. (2020).

SLIM: A Lightweight Block Cipher for Internet of Health Things. *IEEE Access*, *8*, 203747–203757. https://doi.org/10.1109/access.2020.3036589

Banani, S., Thiemjarus, S., Wongthavarawat, K., & Ounanong, N. (2021). A Dynamic Light-Weight Symmetric Encryption Algorithm for Secure Data Transmission via BLE Beacons. *Journal of Sensor and Actuator Networks*, *11*(1), 2. https://doi.org/10.3390/jsan11010002

Banik, S., Pandey, S. K., Peyrin, T., Sasaki, Y., Sim, S. M., & Todo, Y. (2017). GIFT: A Small Present. *Lecture Notes in Computer Science*, 321–345. https://doi.org/10.1007/978-3-319-66787-4_16

Baysal, A., & Şahin, S. (2016). RoadRunneR: A Small and Fast Bitslice Block Cipher for Low Cost 8-Bit Processors. *Lecture Notes in Computer Science*, 58–76. https://doi.org/10.1007/978-3-319-29078-2_4

Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., & Wingers, L. (2015). The SIMON and SPECK lightweight block ciphers. *Proceedings of the 52nd Annual Design Automation Conference*. https://doi.org/10.1145/2744769.2747946

Caraveo-Cacep, M. A., Vázquez-Medina, R., & Hernández Zavala, A. (2023). A survey on low-cost development boards for applying cryptography in IoT systems. Internet of Things, 22, 100743. https://doi.org/10.1016/j.iot.2023.100743

Daemen, J., & Rijmen, V. (2000). The Block Cipher Rijndael. *Lecture Notes in Computer Science*, 277–284. https://doi.org/10.1007/10721064_26

Dinu, D., Corre, Y. L., Khovratovich, D., Perrin, L., Großschädl, J., & Biryukov, A. (2018). Triathlon of lightweight block ciphers for the Internet of things. *Journal of Cryptographic Engineering*, *9*(3), 283–302. https://doi.org/10.1007/s13389-018-0193-x

Encarnacion, P. C. (2020). Modified Round Function of SIMECK 32/64 Block Cipher. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*(1.3), 258–266. https://doi.org/10.30534/ijatcse/2020/3991.32020

Hatzivasilis, G., Fysarakis, K., Papaefstathiou, I., & Manifavas, C. (2018). A review of lightweight block ciphers. *Journal of Cryptographic Engineering*, *8*(2), 141–184. https://doi.org/10.1007/s13389-017-0160-y

Hong, D., Lee, J. K., Kim, D. C., Kwon, D., Ryu, K. H., & Lee, D. G. (2014). LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors. *Information Security Applications*, 3–27. https://doi.org/10.1007/978-3-319-05149-9_1

Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B. S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., & Chee, S. (2006). HIGHT: A New Block Cipher Suitable for Low-Resource Device. *Lecture Notes in Computer Science*, 46–59. https://doi.org/10.1007/11894063_4

Koo, B., Roh, D., Kim, H., Jung, Y., Lee, D. H., & Kwon, D. (2017). CHAM: A Family of Lightweight Block Ciphers for Resource-Constrained Devices. In *Lecture Notes in Computer Science* (pp. 3–25). Springer Science+Business Media. https://doi.org/10.1007/978-3-319-78556-1_1

Lachner, C., & Dustdar, S. (2019). A Performance Evaluation of Data Protection Mechanisms for Resource Constrained IoT Devices. 2019 IEEE International Conference on Fog Computing (ICFC). https://doi.org/10.1109/icfc.2019.00015

Makarenko, I., Semushin, S. A., Suhai, S., Kazmi, S. M. A., Oracevic, A., & Hussain, R. (2020). A Comparative Analysis of Cryptographic Algorithms in the Internet of Things. 2020 International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTeC). https://doi.org/10.1109/monetec49726.2020.9258156

McKay, K. A., Bassham, L., Turan, M. S., & Mouha, N. (2017). Report on lightweight cryptography. https://doi.org/10.6028/nist.ir.8114

Mouha, N., Mennink, B., Van Herrewege, A., Watanabe, D., Preneel, B., & Verbauwhede, I. (2014). Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers. *Selected Areas in Cryptography -- SAC 2014*, 306–323. https://doi.org/10.1007/978-3-319-13051-4_19

Noor, M. M., & Hassan, W. H. (2019). Current research on Internet of Things (IoT) security: A survey. Computer Networks, 148, 283–294. https://doi.org/10.1016/j.comnet.2018.11.025

Panahi, P., Bayilmis, C., Çavuşoğlu, Ü., & Kaçar, S. (2021). Performance Evaluation of Lightweight Encryption Algorithms for IoT-Based Applications. *Arabian Journal for Science and Engineering*, *46*(4), 4015–4037. https://doi.org/10.1007/s13369-021-05358-4

Patil, J., Bansod, G., & Kant, K. S. (2017). LiCi: A new ultra-lightweight block cipher. *2017 International Conference on Emerging Trends & Innovation in ICT (ICEI)*. https://doi.org/10.1109/etiict.2017.7977007

Pei, C., Xiao, Y., Liang, W., & Han, X. (2018). Trade-off of security and performance of lightweight block ciphers in Industrial Wireless Sensor Networks. *EURASIP Journal on Wireless Communications and Networking*, *2018*(1). https://doi.org/10.1186/s13638-018-1121-6

Pereira, F., Correia, R. A., Pinho, P., Lopes, S. M., & Carvalho, N. B. (2020). Challenges in Resource-Constrained IoT Devices: Energy and Communication as Critical Success Factors for Future IoT Deployment. Sensors, 20(22), 6420. https://doi.org/10.3390/s20226420

Seok, B., & Lee, C. (2019). Fast implementations of ARX-based lightweight block ciphers (SPARX, CHAM) on 32-bit processor. *International Journal of Distributed Sensor Networks*, *15*(9), 155014771987418. https://doi.org/10.1177/1550147719874180

Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., & Shirai, T. (2011). Piccolo: An Ultra-Lightweight Blockcipher. *Cryptographic Hardware and Embedded Systems – CHES 2011*, 342–357. https://doi.org/10.1007/978-3-642-23951-9_23

Sleem, L., & Couturier, R. (2020). Speck-R: An ultra light-weight cryptographic scheme for Internet of Things. *Multimedia Tools and Applications*, *80*(11), 17067–17102. https://doi.org/10.1007/s11042-020-09625-8

Sutar, S., & Mekala, P. (2022). Extensive review on IoT security challenges and LWC implementation on tiny hardware for node level security evaluation. International Journal of Next-generation Computing. https://doi.org/10.47164/ijngc.v13i1.424

Suzaki, T., & Minematsu, K. (2010). Improving the Generalized Feistel. *Fast Software Encryption*, 19–39. https://doi.org/10.1007/978-3-642-13858-4_2

Thabit, F., Alhomdy, A. P. S., Al-Ahdal, A. H., & Jagtap, P. D. S. (2021). A new lightweight cryptographic algorithm for enhancing data security in cloud computing. Global Transitions Proceedings, 2(1), 91–99. https://doi.org/10.1016/j.gltp.2021.01.013

Thakor, V., Razzaque, M. A., & Khandaker, M. R. A. (2021). Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities. *IEEE Access*, *9*, 28177–28193. https://doi.org/10.1109/access.2021.3052867

Yang, G., Zhu, B., Suder, V., Aagaard, M. D., & Gong, G. (2015). The Simeck Family of Lightweight Block Ciphers. *Lecture Notes in Computer Science*, 307–329. https://doi.org/10.1007/978-3-662-48324-4_16

Zakaria, A. A., Ab Halim, A. H., Ridzuan, F., Zakaria, N. H., & Daud, M. (2022). LAO-3D: A Symmetric Lightweight Block Cipher Based on 3D Permutation for Mobile Encryption Application. *Symmetry*, *14*(10), 2042. https://doi.org/10.3390/sym14102042

Zhang, W., Bao, Z., Lin, D., Rijmen, V., Yang, B., & Verbauwhede, I. (2015). RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. *Science China Information Sciences*, *58*(12), 1–15. https://doi.org/10.1007/s11432-015-5459-7