Securing Data in Low-Resource Environments: Lightweight Block Ciphers

Fernando Ramirez Arredondo
Computer Science
Universidad Católica San Pablo
Arequipa, Perú
fernando.ramirez@ucsp.edu.pe

Yván Jesús Túpac Valdivia
Computer Science
Universidad Católica San Pablo
Arequipa, Perú
ytupac@ucsp.edu.pe

***Abstract*—Ensuring communication security for devices with limited resources is vital in the contemporary landscape. This work classifies seven state-of-the-art lightweight block ciphers designed for such devices. These ciphers offer efficient encryption while minimizing resource consumption. This study focuses on the two main categories of lightweight block ciphers, Substitution-Permutation Networks and Feistel Networks. The experiments reveal that Feistel Networks excel with limited resources and when decryption implementation is needed, while Substitution-Permutation Networks excel at execution time.**

***Index Terms*—iot, feistel, substitution-permutation, resource-constrained, lightweight, cryptography**

I. Introduction

The development of Internet of Things (IoT) technology initiated an era of interconnected devices, collecting data from the physical world and transforming our interaction with it. This connectivity presents a security challenge when the devices generating or transmitting sensitive data lack the resources to perform up-to-standard encryption processes. As Thakor et al. [1] categorize, IoT devices fall into two resource-based groups, resource-rich devices like servers, personal computers, and smartphones, and resource-constrained devices like sensor nodes, radio-frequency identification (RFID) tags, and actuators. See Table I for the categorization.

Table I
Cryptographic Approaches on Different Devices [1]

Device	Cryptography
Servers, Desktops and Smartphones	Conventional Cryptography
Embedded Systems and RFID	Lightweight Cryptography

This challenge is further amplified by the evolution of IoT, which has transitioned from smaller networks to wide area networks, leading to a rapid growth in the number of interconnected devices. As IoT applications increase, a substantial amount of sensitive data is exchanged, raising concerns about privacy and security. Increased device interconnectivity further intensifies these concerns, potentially exposing new vulnerabilities with each additional device. In a landscape with billions of interconnected devices, a single security flaw can be exploited on a massive scale. Just as physical security measures like locks, safes, and opaque envelopes protect our valuables, cryptography serves as the digital equivalent, safeguarding the integrity and confidentiality of data in our online activities. This work aims to contribute to the application of cryptography for resource-constrained devices in the ever-expanding IoT landscape, where robust security solutions are essential to trust the activities and processes we reproduce or create from scratch in the digital world.

Traditional cryptography, often computationally intensive, is impractical for resource-constrained devices. Lightweight cryptography emerges as a modern solution, specifically designed for devices with limitations in memory, processing power, and energy consumption. It prioritizes low computational complexity and a small memory footprint [2]. Ciphers are divided into two main categories, asymmetric and symmetric. While asymmetric ciphers offer enhanced security features, they require greater computational resources and often have higher implementation costs. Consequently, they are less suitable for resource-constrained devices. Lightweight ciphers lean into the symmetric approach due to being generally faster and requiring less computational power [3]. Following this same line of thinking, the vast majority of these ciphers are designed to operate on fixed-sized blocks of data rather than streams of data, as can be seen in NIST latest Lightweight Cryptography Standardization Process [4]. It is for this reasons that this study focuses on block ciphers, specifically on two fundamental structures employed in lightweight block ciphers (LBC), Feistel Network (FN) and Substitution-Permutation Network (SPN).

The Feistel Network offers a modular and efficient approach to encryption, employing multiple rounds of data processing. In each round, a half-block of data is processed using a round function, often incorporating substitutions and permutations. The output of this function is then combined with the other half-block using an XOR operation, creating a dependency between rounds that improves the overall security [5].

The Substitution-Permutation Network relies on a series of substitution and permutation stages. Substitution box (S-box) replaces data bits, introducing non-linearity to confuse potential attackers. Permutation box (P-box) rearranges the data bits, making it challenging to discern the original data [6]. Refer to Fig. 1 for a general view of both FN and SPN structures.

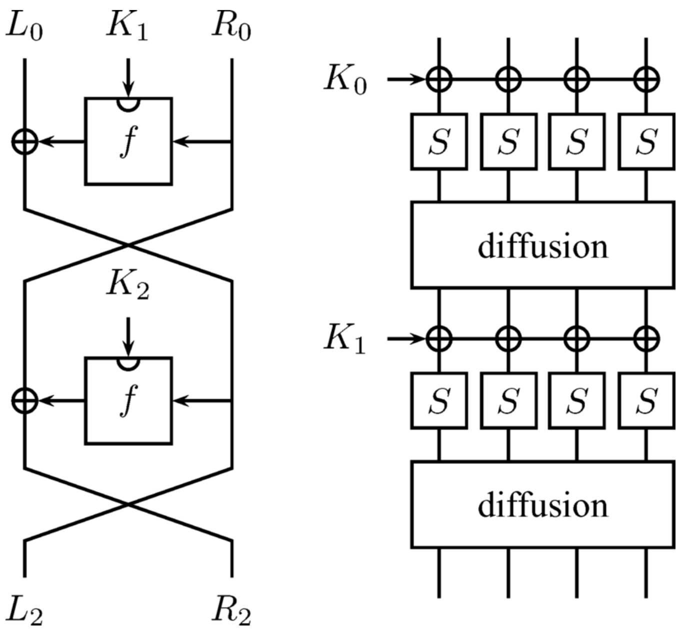

Figure 1. Generalization of a Feistel Network (left) and a Substitution-Permutation Network (right) [7].

Both FN and SPN structures offer advantages in lightweight cryptography. FN structures are known for their modularity and ease of implementation, while SPN structures provide inherent parallelism, potentially leading to faster encryption and decryption on certain architectures. This paper makes an exploration of FN and SPN structures in the context of lightweight cryptography. We will provide analysis of their design principles, operational workflows, and suitability for resource-constrained devices.

Section II. explores seven LBCs categorized by structure, Section III. details the step by step of SAND and GIFT, Section IV. details the implementation and the metrics used to judge performance, Section V. presents the results of the experiments and Section VI. presents the conclusions of the work.

II. State of the Art

This section explores lightweight block ciphers, introducing them based on their structure and the techniques that generate their state-of-the-art performance for resource-constrained devices.

A. SPN-based solutions

Dynamic Ultra-Lightweight Block Cipher (**DULBC**) employs a dynamic SPN structure in which the specific encryption process changes based on the round keys, generating a non-fixed round function. Each round selects one of four functions based on the first two bits of the round key, enhancing security without significant hardware overhead. The proposed S-box consists of three NOR and XOR, one NAND and XNOR operations, leading to a simple inverse, efficient implementation, and good cryptographic properties with minimal cost. The authors highlight successful Field Programmable Gate Array (FPGA) and Application-Specific Integrated Circuit (ASIC) implementations of DULBC, demonstrating high throughput (number of operations processed per unit time) and relatively low area requirements (number of transistors needed to build the hardware that performs the encryption operations). Security analysis confirms the resistance of DULBC to various attacks, including differential attacks, linear attacks, and side-channel attacks [8].

Adomnicai et al. proposed a new representation of the LBC **GIFT**. This new representation, called fixslicing, allows for efficient software implementation. It improves upon the original GIFT by replacing the transpositions, shuffle and swaps with a more efficient way of representing the round operations in a bitsliced manner. By operating on multiple bits at once, in bit-slicing substitution, the larger data element is sliced into smaller pieces. Each slice is then independently subjected to the substitution operation using the S-box. This enables the processing of multiple bits at once, leveraging the architecture of the processor. This new representation does some light modifications over the original Key Scheduling Algorithm (KSA) for the round keys to fit the new round function [10].

InVolutive Lightweight Block Cipher (**IVLBC**) is a LBC designed for unified encryption and decryption in resource-limited environments. The design prioritizes high diffusion (influence of a single bit change in the plaintext over the ciphertext) for strong encryption, involution (components return to their original state after two applications) for efficient implementation and flexibility in hardware and software implementations. To achieve these goals, IVLBC utilizes a FN structure with a tree-based design for a compact and efficient S-box, by using a tree-based structure with logic operations, the S-box in IVLBC can achieve the same functionality as a traditional table-based S-box while requiring less memory and resulting in faster processing. Additionally, it employs a nibble-based involutive permutation to enhance diffusion. Using nibble-based operations can further enhance diffusion by disrupting patterns across larger chunks of data compared to single-bit permutations. Decryption can reuse the same circuitry as encryption, leading to lower resource requirements [11].

B. FN-based solutions

SCENERY is a LBC that makes use of bit-slicing in the round function design to achieve low-cost hardware and efficient software implementations. The main benefits of this implementation are speed improvements on hardware that supports parallel execution and simpler logic circuits in hardware design, potentially reducing area and power consumption compared to traditional designs. While bit-slicing increases code complexity, lightweight ciphers are not complex by definition, so this is not a significant issue if executed correctly. To address the slow diffusion issue that can occur with a FN, SCENERY incorporates a 32x32 binary matrix used to operate in the 64-bit blocks that can be implemented efficiently and improves diffusion speed. The authors propose a new KSA

Table II
LIGHTWEIGHT BLOCK CIPHERS

Algorithm	Key size	Block size	N. of rounds	Structure
DULBC (Yang et al., 2022) [8]	80/128	64	25/30	SPN
GIFT (Adomnicai et al., 2020) [9] [10]	128	64/128	28/40	SPN
IVLBC (Huang et al., 2023) [11]	80/128	64	29	SPN
LBC-IoT (Ramadan et al., 2021) [12]	80	32	32	Feistel
SAND (Chen et al., 2021) [13]	128	64/128	48/54	Feistel
LBCCS (Zhu et al., 2022) [14]	128	128	20	Feistel
SCENERY (Feng and Li et al., 2022) [15]	80	64	28	Feistel

to tackle the issue of backward derivation of the master key. While not a complete solution, this method makes it more difficult compared to the key scheduling of PRESENT [16] and enhances the security of key expansion [15].

Lightweight Block encryption algorithm based on Combined Chaotic System (**LBCSS**) is a LBC that uses a combined chaotic system for encryption and decryption. This means that instead of relying on a single mathematical function that exhibit chaotic behavior, LBCSS uses multiple functions together. This chaotic system is used to create secure S-boxes, P-boxes, and keys for the encryption process. LBCSS makes use a structure that enhances diffusion compared to the traditional FN, but maintains a concise design. The round function is created using bit-slicing techniques, where operations like substitutions, rotations, or bitwise operations introduce non-linear transformations on the data. The round function remains consistent across all encryption rounds to maintain low complexity. To prevent these consistent rounds from becoming too predictable, round constants are introduced to add variation. The authors argue that LBCSS offers sufficient security based on various tests, including resistance to different cryptanalysis methods and statistical testing [14].

SAND is a LBC that it employs AND, Rotation, and XOR operations (AND-RX) within a FN. Traditionally, evaluating the security of AND-RX ciphers can be complex because these operations often work on individual bits within the data, making it difficult to apply existing analysis methods typically used for S-boxes. SAND allows for easier security evaluation compared to other AND-RX ciphers by utilizing AND-RX operations within small data units (nibbles). This allows security researchers to use existing methods designed for analyzing S-boxes, which are a more familiar and well-studied component in cryptography. The authors highlight the benefits of SAND in terms of both security and performance. SAND achieves strong security under various attack scenarios, including differential and linear attacks. The paper also discusses the motivations behind the design of SAND. Existing ARX-based ciphers often rely on large S-boxes, which are not ideal for hardware implementation. SAND provides an AND-RX design with efficient hardware implementation and strong security guarantees through its novel S-box transformation approach [13].

LBC-IoT achieves high diffusion by employing a combination of techniques including a compact S-box with high non-linearity properties for data substitution and dedicated P-boxes for bit permutation, further obfuscating the relationship between the original and encrypted data. It is well known that FN structures can lack diffusion due to the round function being applied to only half of the data each round. The solution of LBC-IoT to this problem is to apply a different P-box to each half of the data at the end of the rounds. The cipher utilizes simple operations in the core function and keeps the S-box size small. This approach minimizes the amount of processing power and memory required for implementation. Security-wise, LBC-IoT is resistant to brute-force attacks by adhering to NIST recommendations for key length. It uses an 80-bit key, which is too complex to crack through exhaustive searching methods. The authors ensure that LBC-IoT offers a good balance between robust security and efficient implementation [12].

Lightweight block ciphers are categorized based on their genereal metrics and structure. The performance of each algorithm is evaluated based on the following metrics.

- **Block Size**: To save on processing power and battery life in IoT devices, block ciphers chop up the data they encrypt into small, pre-defined chunks. The size of these chunks directly impacts how much energy the encryption process consumes. As shown in Table II, GIFT, SAND and LBCCS have the largest block size of 128 bits while LBC-IoT adopts the smallest block size of only 32 bits. Most surveyed ciphers employ 64-bit blocks.
- **Key Size**: The number of bits in a cryptographic key, known as the key size, directly affects security. Bigger keys are like more complex locks, offering stronger protection however, they require more effort (processing power and energy) to use. NIST suggests a minimum key size of 80 bits for basic defense against brute-force attacks, where attackers try every possible key combination [17]. Reflecting this trade-off, none of the ciphers in Table II exceed 128 bits or fall below 80 bits, opting for one of these two key sizes.
- **Number of rounds**: Compared to conventional ciphers, lightweight block ciphers prioritize simplicity to work on devices with limited resources. Unlike conventional ciphers, they achieve strong security through multiple rounds of processing. While each extra round makes the cipher harder to crack, it also slows down the encryption process.

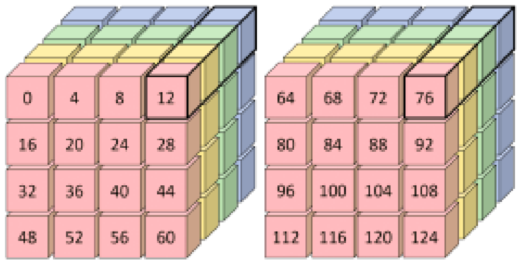

Figure 2. Cubic representation of the main state of GIFT-128. The black cuboid is where an S-box is applied for both matrices [10].

III. Techniques to implement

This section details two of the lightweight block ciphers introduced earlier, SAND and GIFT. The remainder of this section will outline their encryption and decryption processes, along with the rationale behind chosing them for implementation. NOTE: Implementations will follow NIST Recommendation for Block Cipher Modes of Operation [3].

A. GIFT

GIFT has two variants, GIFT-64, a 64-bit block, 28 round SPN cipher, and GIFT-128, a 128-bit block, 40-round SPN cipher. Due to its larger block size offering potentially stronger security, GIFT-128 will be implemented for this project [10].

1) Round function: GIFT starts with an initialization phase and continues with 40 rounds of operations, each round consisting of three steps, SubCells, PermBits and AddRoundKey. The decryption process consists of performing the same steps with inversed P-box and S-box.

Initialization. The 128-bit plaintext is loaded into the cipher state S which is represented as four 32-bit segments. A matrix representation of the cipher state S can be found in Fig. 2 where each color represents a segment, as described in eq. 1.

$$S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \leftarrow \begin{bmatrix} b_{124} & \cdots & b_8 & b_4 & b_0 \\ b_{125} & \cdots & b_9 & b_5 & b_1 \\ b_{126} & \cdots & b_{10} & b_6 & b_2 \\ b_{127} & \cdots & b_{11} & b_7 & b_3 \end{bmatrix} \tag{1}$$

SubCells. The substitution layer of 32 identical 4-bit S-boxes is applied to S as shown in eq. 2.

$$\begin{aligned} S_1 &\leftarrow S_1 \oplus (S_0 \wedge S_2) \\ S_0 &\leftarrow S_0 \oplus (S_1 \wedge S_3) \\ S_2 &\leftarrow S_2 \oplus (S_0 \vee S_1) \\ S_3 &\leftarrow S_3 \oplus S_2 \\ S_1 &\leftarrow S_1 \oplus S_3 \\ S_3 &\leftarrow \neg S_3 \\ S_2 &\leftarrow S_2 \oplus (S_0 \wedge S_1) \\ \{S_0, S_1, S_2, S_3\} &\leftarrow \{S_3, S_1, S_2, S_0\} \end{aligned} \tag{2}$$

Where $\wedge$, $\vee$ and $\neg$ are logical AND, OR and NOT operations respectively.

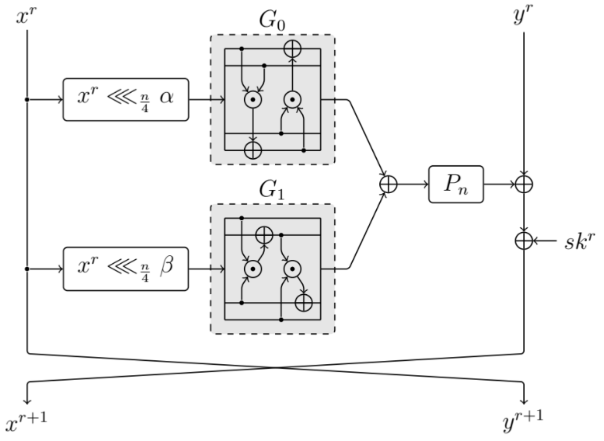
x^r
G_0
y^r
$x^r \lll_{\frac{n}{4}} \alpha$
G_1
P_n
sk^r
$x^r \lll_{\frac{n}{4}} \beta$
x^{r+1}
y^{r+1}

Figure 3. Representation of the encryption round function of SAND [13].

Permbits. Different 32-bit permutations are applied to each S_i independently. The specifications for GIFT-128 bit permutation table are detailed in the main article [10].

AddRoundKey. This step consists of adding the round key and round constant. Two 32-bit segments U, V are extracted from the key state as the round key $RK = U||V$. U and V are XORed to S_2 and S_1 as can be seen in eq. 3.

$$\begin{aligned} S_2 &\leftarrow S_2 \oplus U \\ S_1 &\leftarrow S_1 \oplus V \end{aligned} \tag{3}$$

For the addition of round constant, S_3 is updated as shown in eq. 4.

$$S_3 \leftarrow S_3 \oplus 0x800000XY \tag{4}$$

2) Reasons for implementation: The GIFT block cipher, known for efficient hardware implementation, had a slow software implementation due to its heavy operations. However, this new representation of GIFT allows for extremely efficient software bitslice implementation using only a few rotations while maintaining the security that has GIFT as one of the go-to LBCs.

B. SAND

SAND is a family of two AND-RX block ciphers with Feistel structure, SAND-64, a 64-bit block, 48-round cipher, and SAND-128, a 128-bit block, 54-round cipher. Due to its larger block size offering potentially stronger security, SAND-128 will be implemented for this project [13].

1) Round function: SAND starts with a state loading phase and continues with 54 iterations of the round function F, where G_0 and G_1 are non-linear functions, and P_n is a permutation providing diffusion. The decryption process is very similar to encryption, but the round keys are applied in reverse order. This allows for easy decryption at almost zero extra cost. Refer to Fig. 3 for a representation of the round function.

State Loading. The plaintext is represented as two halves $P = (Pl, Pr)$, the left 64-bit half $Pl = (Pl_{63}, ..., Pl_1, Pl_0)$ and the right 64-bit half $Pr = (Pr_{63}, ..., Pr_1, Pr_0)$ are represented as two array of bits as shown in eq. 5.

$$Pl = \begin{bmatrix} Pl_{63} & \cdots & Pl_7 & Pl_3 \\ Pl_{62} & \cdots & Pl_6 & Pl_2 \\ Pl_{61} & \cdots & Pl_5 & Pl_1 \\ Pl_{60} & \cdots & Pl_4 & Pl_0 \end{bmatrix} = \begin{bmatrix} x^0 & \{3\} \\ x^0 & \{2\} \\ x^0 & \{1\} \\ x^0 & \{0\} \end{bmatrix}, \quad Pr = \begin{bmatrix} Pr_{63} & \cdots & Pr_7 & Pr_3 \\ Pr_{62} & \cdots & Pr_6 & Pr_2 \\ Pr_{61} & \cdots & Pr_5 & Pr_1 \\ Pr_{60} & \cdots & Pr_4 & Pr_0 \end{bmatrix} = \begin{bmatrix} y^0 & \{3\} \\ y^0 & \{2\} \\ y^0 & \{1\} \\ y^0 & \{0\} \end{bmatrix} \tag{5}$$

The input state (x, y) of the encryption is loaded from P in a row-by-row manner.

Non-linear Functions. Let the 64-bit variable x be the input value of G_0 and G_1, which is denoted as the concatenation of four 16-bit words $x\{3\}||x\{2\}||x\{1\}||x\{0\}$. Let $y = y\{3\}||y\{2\}||y\{1\}||y\{0\}$ denote the output value. For G_0 refer to eq. 6.

$$\begin{aligned} y\{0\} &= x\{3\} \odot x\{2\} \oplus x\{0\}, \\ y\{3\} &= y\{0\} \odot x\{1\} \oplus x\{3\}, \\ y\{2\} &= x\{2\}, \\ y\{1\} &= x\{1\}. \end{aligned} \tag{6}$$

As to the function G_1, the output is calculated as shown in eq. 7.

$$\begin{aligned} y\{2\} &= x\{3\} \odot x\{1\} \oplus x\{2\}, \\ y\{1\} &= y\{2\} \odot x\{0\} \oplus x\{1\}, \\ y\{3\} &= x\{3\}, \\ y\{0\} &= x\{0\}. \end{aligned} \tag{7}$$

Bit Permutation. This operation can be represented as the parallel application of a 16-bit permutation table. The bit permutation table p_{16} for SAND-128 is illustrated in the main article [13].

2) Reasons for implementation: SAND implements an approach that restricts AND-RX operations to work within 4-bit nibbles. This allows SAND to be represented equivalently using a 4x8 synthetic S-box (SSb). This conversion unlocks the advantage of employing established S-box-based security evaluation methods. The authors claim that the bitslice structure combined with AND-RX operations positions SAND among the top contenders for efficient software implementation. The simplicity of employing AND-RX operations and the robustness of well-developed S-box-based security evalution methods and tools appear to be a promising advancement.

IV. Implementation

This section details the implementation of the SAND-128 and GIFT-128 lightweight ciphers in software. The source code is writen out in C. The metrics used to judge the performance of the implementations are execution time, code size, memory usage, total cycles, latency, efficiency, and throughput, plus a synthetic metric. The rest of the section describes the process and key considerations for the software implementation.

A. Algorithm Specification and Code Development

The specifications of SAND-128 and GIFT-128 were obtained from their respective academic papers. Following specifications from the articles, the encryption and decryption functions for both ciphers were implemented in C.

B. Testing and Validation

The implemented ciphers were proven valid against the test vectors in NIST Special Publication 800-38A 2001 ED [3], this article provides recommendations for block cipher modes of operation. The selected encryption mode of operation is the electronic codebook (ECB). The main drawback lies in a limited diffusion capability, as it does not effectively conceal data patterns when encrypting identical plaintext blocks, resulting in identical ciphertext blocks. This critical disadvantage can be ignored when the plaintext blocks are selected in a testing environment, leaving ECB as the optimal encryption mode because of simplicity. The tests ensure that the software implementations produce correct outputs for known inputs.

C. Performance Evaluation

The performance of the software implementation was evaluated based on the following metrics.

- **Execution Time**: The time taken to perform encryption operations was measured using high-resolution timers.
- **Code Size**: Memory size to store the cipher code and constants, measured in kilobytes (kB).
- **Memory Usage**: The memory consumption during the encryption process was analyzed for both RAM and ROM.
- **Throughput**: The number of operations processed per unit time was calculated to determine the efficiency of the implementation. Throughput, denoted as T, can be defined as:
$$T = \frac{D}{s}$$
where D is the data size in megabytes (MB) and s is the time in seconds or as t:
$$t = \frac{d}{ms}$$
where d is the data size in kilobits (kb) and ms is the time in milliseconds.
- **Total Cycles**: The total number of clock cycles required for the encryption process.
- **Latency**: Latency denotes the number of clock cycles required to compute each plaintext/ciphertext block. Latency, denoted as L, can be defined as:
$$L = \frac{N_b}{C}$$
where N_b is the number of 128-bit blocks and C is the number of cycles.

Table III
COMPARISON OF SOFTWARE PERFORMANCE FOR ENCRYPTION WITH 2MB OF MESSAGES.

Algorithm	Execution Time	Code Size	Memory Usage	Throughput	Total Cycles	Latency	Efficiency	Synth. Metric
GIFT-128	**47650 ms**	10.1 kB	20.2 kB	**20.85**	**164469482**	**1255**	16912.49	12977670
SAND-128	132980 ms	**2.2 kB**	**8.2 kB**	15.04	458841876	3501	**56002.95**	**7886345**

Table IV
COMPARISON OF SOFTWARE PERFORMANCE FOR ENCRYPTION AND DECRYPTION WITH 2MB OF MESSAGES.

Algorithm	Execution Time	Code Size	Memory Usage	Throughput	Total Cycles	Latency	Efficiency	Synth. Metric
GIFT-128	**84099 ms**	12.4 kB	23.3 kB	**11.81**	**288633765**	**2202**	7792.72	28186891
SAND-128	262752 ms	**2.6 kB**	**8.5 kB**	7.61	908490955	6931	**23982.84**	**18453723**

- **Efficiency**: Efficiency evaluates the relationship between performance and implementation cost. Generally, higher efficiency values are preferable. Software efficiency follows.
$$E = \frac{t}{S}$$
where t is throughput in kilobits per second (kbps) and S is the code size in kilobytes (kB).
- **Synthetic Metrics**: A combined metric such as Code Size $\times$ Cycle Count / Block Size.

V. EXPERIMENTS AND RESULTS

In this experiments, the key schedule is not taken into account as the round keys are assumed to be precomputed and stored in RAM. The benchmark consists in measuring the time required to encrypt and decryption 2MB data making use of the ECB mode of operation. All benchmarking results can be found in Table III & IV and all the tests are accomplished on a PC with 2.7GHz Intel(R) Core i7-7500U CPU using gcc 13.2.0.

In Table III can be observed that the GIFT execution time is almost three times shorter than the SAND execution time for encryption only. This can be attributed in part to the number of rounds and the complexity of functions. The code size is 10.1 kB for GIFT and 2.2 kB for SAND. This metric is important when implementation space is limited and code size needs to be minimal. It is important to note that the code size of GIFT will inevitably increase if decryption is also implemented, while the SAND code size will remain the same. Memory usage is significantly smaller for SAND, which is crucial when resources are limited and memory usage must be minimal. Throughput, referring to the MB of data processed per second, is better for GIFT at 20.85 MB/s compared to 15.04 MB/s for SAND. In terms of clock cycles, GIFT employs 1255 cycles per 128-bit block of data, while SAND uses 3501 cycles per 128-bit block of data. Efficiency evaluates the relationship between performance and implementation cost in terms of throughput and code size, SAND is almost four times as efficient as GIFT. The synthetic metric reveals that even though these ciphers operate in very distinct ways, considering the previously mentioned metrics, SAND is clearly favorite in terms of code size and cycles per block.

Results in Table IV show that adding decryption to the expermient doubles the execution time for both implementations, the code size for SAND barely increases while GIFT gains 2.3 kB. Memory usage increases proportionaly for both ciphers and throughput is reduced to half. It can be noted that SAND is the one that handles the addition of the decryption operations better.

VI. CONCLUSIONS

Feistel networks and Substitution-Permutation networks are both fundamental structures for designing lightweight block ciphers. While they share some similarities, they have distinct advantages and disadvantages.

Feistel networks offer invertibility, meaning decryption is straightforward even if the round function itself is not invertible. This simplifies design and analysis. However, these ciphers also have drawbacks. The overall security depends heavily on a strong Feistel function, and each round introduces processing overhead, potentially slowing down encryption and decryption.

Substitution-Permutation networks achieve higher throughput due to their inherent parallelism, making them attractive for implementations with multiple processing units, and confusion by employing S-boxes that perform non-linear operations on the data, making the link between plaintext, key, and ciphertext highly complex. Additionally, P-boxes ensure diffusion by rearranging the data bits, such that a single change in the plaintext has a cascading effect on the final ciphertext. Ultimately, the number of encryption rounds and the design of the S-boxes and P-boxes play a critical role in determining the strength and security of an Substitution-Permutation network cipher.

Making the best choice between a Feistel network and a Substitution-Permutation network depends on the specific application and design goals. If ease of decryption and hardware limitations are key concerns, a Feistel network might be preferable. If maximizing encryption speed on parallel hardware is a priority, a Substitution-Permutation network might be a better choice.

Bit-slicing allows for parallel processing on hardware that supports it, potentially leading to significant speed improvements. It is particularly beneficial for lightweight ciphers designed for resource-constrained environments. However, bit-slicing can increase code complexity, so it is important to find a balance between efficiency and implementation effort. Some S-boxes might be designed with low complexity for compact hardware implementations, while others might prioritize high non-linearity for enhanced security.

The development of lightweight cryptography seems to be a continuing search for balance between security and efficiency. On one hand, lightweight ciphers need to be strong enough to resist attacks, but on the other hand, they must also be efficient in terms of resource consumption (memory, processing power) to run on devices with limited capabilities.

References

[1] V. Thakor, M. A. Razzaque, and M. Khandaker, "Lightweight cryptography algorithms for resource-constrained iot devices: A review, comparison and research opportunities," *IEEE Access*, vol. 9, pp. 28 177–28 193, 01 2021.

[2] Y. Zhong and J. Gu, "Lightweight block ciphers for resource-constrained environments: A comprehensive survey," *Future Generation Computer Systems*, 2024.

[3] M. Dworkin, "Recommendation for block cipher modes of operation," *NIST special publication*, vol. 800, p. 38B, 2001.

[4] K. McKay, L. Bassham, M. Sönmez Turan, and N. Mouha, "Report on lightweight cryptography," National Institute of Standards and Technology, Tech. Rep., 2016.

[5] V. Nachef, J. Patarin, and E. Volte, "Feistel ciphers," *Cham: Springer International Publishing*, 2017.

[6] H. M. Heys and S. E. Tavares, "Substitution-permutation networks resistant to differential and linear cryptanalysis," *Journal of cryptology*, vol. 9, no. 1, pp. 1–19, 1996.

[7] C. De Canniere, A. Biryukov, and B. Preneel, "An introduction to block cipher cryptanalysis," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 346–356, 2006.

[8] J. Yang, L. Li, Y. Guo, and X. Huang, "Dulbc: A dynamic ultra-lightweight block cipher with high-throughput," *Integration*, vol. 87, pp. 221–230, 2022.

[9] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "Gift: A small present: Towards reaching the limit of lightweight encryption," in *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Springer, 2017, pp. 321–345.

[10] A. Adomnicai, Z. Najm, and T. Peyrin, "Fixslicing: A new gift representation," *Cryptology ePrint Archive*, 2020.

[11] X. Huang, L. Li, and J. Yang, "Ivlbc: An involutive lightweight block cipher for internet of things," *IEEE Systems Journal*, 2022.

[12] R. A. Ramadan, B. W. Aboshosha, K. Yadav, I. M. Alseadoon, M. J. Kashout, and M. Elhoseny, "Lbc-iot: Lightweight block cipher for iot constraint devices." *Computers, Materials & Continua*, vol. 67, no. 3, 2021.

[13] S. Chen, Y. Fan, L. Sun, Y. Fu, H. Zhou, Y. Li, M. Wang, W. Wang, and C. Guo, "Sand: An and-rx feistel lightweight block cipher supporting s-box-based security evaluations," *Designs, Codes and Cryptography*, pp. 1–44, 2022.

[14] D. Zhu, X. Tong, Z. Wang, and M. Zhang, "A novel lightweight block encryption algorithm based on combined chaotic system," *Journal of Information Security and Applications*, vol. 69, p. 103289, 2022.

[15] J. Feng and L. Li, "Scenery: a lightweight block cipher based on feistel structure," *Frontiers of Computer Science*, vol. 16, no. 3, p. 163813, 2022.

[16] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9*. Springer, 2007, pp. 450–466.

[17] E. Barker and A. Roginsky, "Transitioning the use of cryptographic algorithms and key lengths," National Institute of Standards and Technology, Tech. Rep., 2018.