LESCA: LightwEight Stream Cipher Algorithm for Emerging Systems

Hassan Noura[1], Ola Salman[2], Raphaël Couturier[1] and Ali Chehab[2]

[1]Univ. Bourgogne Franche-Comté (UBFC), FEMTO-ST Institute, CNRS, Belfort, France
[2]American University of Beirut, Electrical and Computer Engineering Department, Lebanon

***Abstract*—Recently, there has been a dire need for lightweight cryptographic solutions, which exhibit low computational complexity and require few resources. In this paper, we present LESCA, a novel dynamic key-dependent lightweight stream cipher, which consists of two main functions, a typical round function based on cryptographic primitives, and a function that updates these primitives. The update is performed in a selective (partial) manner while encrypting a block, and in full after each δ blocks/iterations. As such, LESCA consumes minimal resources and introduces a very low latency. The originality of this solution stems from the fact that the cryptographic primitives get updated even when encrypting the same message. Several performance and security tests were performed to confirm that the proposed cipher is robust and efficient, especially for limited devices and real-time applications. The proposed cipher achieves a high throughput; for example, when implemented on a Raspberry Pi (RPI4) device, LESCA provides an enhancement of at least 343% when compared to the Advanced Encryption Standard (AES), 72% over a recent one-round cipher scheme, and 43% improvement compared to a recent LORCA stream cipher that outperforms the Simon and Speck algorithms.**

***Index Terms*—Dynamic cryptography; lightweight encryption; variable cryptographic primitives; Security and performance analysis;**

I. Introduction

Emerging systems, especially Internet of Things (IoT) networks, are enabling the online interconnection of different types of devices and in large numbers. These large-scale networks offer users a myriad of applications by leveraging the massive quantity of gathered data. However, this gives rise to new Quality of Service (QoS) challenges, in addition to dangerous security threats. These systems are vulnerable to the typical threats associated with traditional networks, in addition to new types of threats that might compromise their availability, in addition to breaching the privacy and security of their data. Data confidentiality and privacy are critical security services, and the challenge is intensified by the huge volume of data and limited resources of some IoT devices. In general, symmetric encryption algorithms are used to achieve data confidentiality, which in some scenarios achieve data privacy as well, such as the case of encrypted surveillance recordings that prevent attackers from identifying individuals.

A. Problem Formulation

Several applications transmit and process sensitive data that must be protected by adopting the best practices in terms of data confidentiality and privacy. However, most current symmetric encryption algorithms are computationally costly, mainly due to the use of a round function that 1) consists of multiple operations, some of which are complex, and 2) needs to be iterated multiple times. As a result, they are not appropriate for tiny devices (e.g., micro-controllers and constrained IoT devices) and/or real-time applications. This led to the emergence of "lightweight" cryptographic algorithms in an attempt to reduce the encryption computational complexity, and hence, to decrease the latency and energy consumption, which are directly related to the time spent in data processing and transmission.

Traditional encryption algorithms such as AES iterate a round function that includes complex operations with static (fixed) cryptographic primitives. The secret key is solely used to derive the sub-keys used in the iteration process. Such algorithms exhibit a high computational complexity, and they are vulnerable to side channel attacks because of their static approach. In response to a NIST call for lightweight cipher solutions, mainly to cater for constrained IoT devices, the shortlisted algorithms adopted the static approach, and focused on 1) the design of a simplified round function (e.g., SIMON and SPECK by eliminating the substitution table) and/or 2) reducing the number of iterations of the round function. One possible solution for reducing the number of iterations is to adopt the dynamic approach whereby

the cryptographic primitives are no longer fixed, instead they get updated frequently, which in turns provides random physical properties and consequently immunity against side channel attacks.

B. Motivation

The recent lightweight ciphers that adopt the static approach still require a relatively large number of rounds [1], such as ASCON and Elephant that require at least 6 and 18 rounds, respectively. In this work, we propose a novel stream cipher scheme based on the dynamic cryptographic approach that uses a minimum number of operations, and thus, minimizes the associated latency, while maintaining the required security level. Such a solution is appropriate for restricted devices and real-time applications. For each new input message, the encryption is performed using a different set of cryptographic primitives. The key-stream is generated by iterating two functions, "round" and "update". The round function is iterated once to generate a new key-stream block, and the update function changes the cipher primitives such as the permutations tables, and it could be configured to perform the update for each δ blocks or messages. For example, in the case of multimedia contents, these could be divided into α blocks, each encrypted using a different set of cryptographic primitives. The two functions consist of simple, yet effective operations that enable them to satisfy the required cryptographic properties.

In summary, the proposed scheme is designed to outperform the NIST shortlisted algorithms that adopt the static approach, and at the same time provide immunity against side channel attacks.

C. Contributions

The construction of cipher schemes, based on the dynamic approach, was proposed to reduce the number of rounds [2]–[6]. These schemes were the first to offer a good balance between performance and the security level. However, they still exhibit a relatively high cost when re-generating the dynamic keys and cryptographic primitives, or when updating the cryptographic primitives after each input block. In this work, we address this issue by simplifying the update process using just a permutation operation. Moreover, to increase the security level, we propose to update all cryptographic primitives (three permutation tables) for each new δ blocks of a message. This design enhances the performance without degrading the security level.

LESCA consists of a round function that is basically a key-stream generation function, and the update process of the cryptographic primitives takes place in a selective manner after each iteration, and in full after a number of input blocks, δ. The update process itself is lightweight when compared to other existing solutions.

LESCA exhibits the following properties:

- **Flexibility:** A block to be encrypted has a variable length of h words, where h depends on the application requirements and device limitations. The block length is $(h \times W)$ bits, where W represents the word's size (precision).
- **Efficiency**: LESCA consists of a simple round function with simple operations that can process h words in parallel, in addition to a simple update process for the cryptographic primitives. Hence, the scheme minimizes the resource utilization and computational complexity.
- **Simple hardware and software implementations:** This is possible since the scheme uses the "XOR" logical operation, a simple and efficient PRNG, as well as look-up tables for the selection process of permutation tables.
- **Error Tolerance**: Compared to [3], as a stream cipher, LESCA has a higher channel error resistance. In an encrypted block, a bit error affects only a single byte, which corresponds to the byte in error.
- **Session Key-Dependent Approach:** LESCA relies on dynamic and key-dependent cryptographic primitives, which can vary in a pseudo-random manner for each block. After each δ blocks or messages, all cryptographic primitives used in the encryption and decryption processes are updated. This renders analytic and implementation attacks very challenging [4], [7], [8].
- **Dynamic Cryptographic Primitives**: Existing ciphers employ fixed cryptographic primitives to encrypt all blocks in a message. However, LESCA uses the concept of variable cryptographic primitives, which makes the relation among the produced key-streams more complex and random, and ensures its resistance to various analytic attacks.

D. Organization

The paper is structured as follows: in Section II, we review the existing lightweight cipher algorithms, and we highlight their limitations. The dynamic key generation and the construction of the cryptographic primitives are described in Section III. Section IV describes the LESCA stream cipher and its main components. In Section V,

we perform a security analysis to assess the proposed cipher against the required cryptographic properties. In Section V-D, the cipher's resistance to various types of attacks is assessed, while the efficiency of LESCA is proved in Section VI. Finally, we conclude the paper in Section VII.

II. Related Work

Encryption can be done either at the block level or stream level. An input data block is combined with a pseudo-random key-stream in the case of a stream cipher, whereas input data is divided into fixed-size blocks in the case of a block cipher (usually 128 bits) and encrypted according to a specific mode of operation. Typically, a block cipher relies on an invertible round function that is iterated r times, for each block [9]. A block cipher algorithm can be used to generate a key-stream sequence by adopting the Output FeedBack (OFB) or CounTeR (CTR) modes of operation. In these cases, the block cipher could be used as a stream cipher [10].

Traditional block ciphers exhibit a high computational cost since they require a large number of rounds, in addition to a diffusion operation within the round function [3]. The minimum number of rounds for existing block ciphers is 4, which is the case of the Hummingbird2 cipher. Such an overhead is not appropriate for some emerging systems [3]. Recently, there has been an interest in designing new lightweight cryptographic algorithms that exhibit much lower overhead in terms of latency and required resources.

Researchers proposed several lightweight ciphers to address the issue of computational complexity, such as RECTANGLE, which is based on the Substitution–Permutation Network (SPN) [11]. Similarly, other lightweight ciphers that use the Feistel networks were presented recently such as AKF [12], Simon and Speck [13].

The Simon algorithm has been optimized for performance in hardware implementations, while Speck is better suited for software implementations. The main idea of the two techniques (Speck and Simon) is to reduce the complexity of the round function, but they both use the multi-round structure. Next, the SIMECK algorithm, a combination of Speck and Simon algorithms, was proposed in [14]. However, the authors of [15] proved SIMECK to be vulnerable to the random byte attack and the bit-flip attack.

Elliptic curve cryptography was adopted by researchers to design lightweight cipher schemes [16]. TWINE is a primary work in this direction, and it was described in [17], [18]. In recent works such as QTL [19], and the Substitution–permutation Feistel Network (SP-FN) [20], the authors merged SPN with Feistel Network (FN) to leverage the benefits of both. However, the techniques are still based on static cryptographic primitives, and a round function that needs to be iterated multiple times. Thus, these schemes are not efficient enough for constrained devices or for real-time applications.

Another approach in the same direction is based on chaotic cryptography. However, the proposed algorithms suffer from different security and performance challenges. They are vulnerable to some attacks and they exhibit performance difficulties due to the need for conversion operations, and floating-point calculations. Furthermore, these ciphers employ a multi-round structure, as seen in [21].

Stream cipher algorithms are typically faster and require less computational and memory resources when compared to block ciphers. Many lightweight cipher schemes have been proposed such as Grain [22] and Grain-128 [23]. These schemes, however, still require r rounds, and they employ a round function with fixed cryptographic primitives, and thus, reducing their number of rounds renders them vulnerable to a range of analytical attacks.

In summary, the need persists for cipher schemes that are well suited for real-time applications and that can be adopted by devices with limited computational power, memory capacity, and battery life [24], [25].

To that end, researchers are targeting 1) the design of lightweight round functions that use simple operations, and 2) the use of a dynamic key-dependent cryptographic technique to minimize the number of rounds. The authors of this paper have been focusing on the design of such techniques, and they presented in [3], [26], a scheme with with a low number of rounds with a high level of security. Another technique presented in [2], requires two rounds, and the one proposed in [3] needs a single round, and it processes two blocks at once.

In this paper, the objective is to enhance the prior works through the design of a robust and more efficient stream cipher that uses variable cryptographic primitives with minimal costs associated with the construction of

Table I: Table of notations

Symbol	Definition
SK	Shared secret session key
DV	The hashed of SK by using a cryptographic hash function such as SHA-512 and updated per session
kp_i	The i^{th} sub-key of permutation that will be used to construct the i^{th} permutation table π_i for i =1, 2, 3 and 4
π_i	The i^{th} dynamic permutation table, which will be updated after each δ encrypted/decrypted blocks
k_S	PRNG Seed
V	a sequence of pseudo-random keystream values.
V_j	a block of h words from V produced at iteration j, which is updated after each δ $(mod(j, \delta) == 0)$ encrypted/decrypted blocks.
len	Input message length
$\lceil x \rceil$	x gets the next integer higher than its existing value
h	Number of words per block message
Wp	The word precision and it can be equal to 64 bits
nb	The number of blocks per input message and it is equal to $\lceil \frac{len}{h \times Wp} \rceil$
M	Plaintext message
m_i	i^{th} original plain block
C	Ciphertext message
c_i	i^{th} encrypted block
$MKSA$	RC4's modified KSA was presented in [3].
$[Y, X] = XorShift64(X)$	The XorShift PRNG is iterated with h words as input block X to generate a pseudo-random block Y (round key). The i^{th} element of the input word is considered as seed for the XorShift PRNG and its corresponding output is stored in the i^{th} index in Y.
$x << n$	The bits in x shifted to the left by n times.
$V_j(\pi_i)$	means that the produced vector is the permutation vector of the j^{th} V is ordered according to the i^{th} permutation table
$x >> n$	Right shift operator.

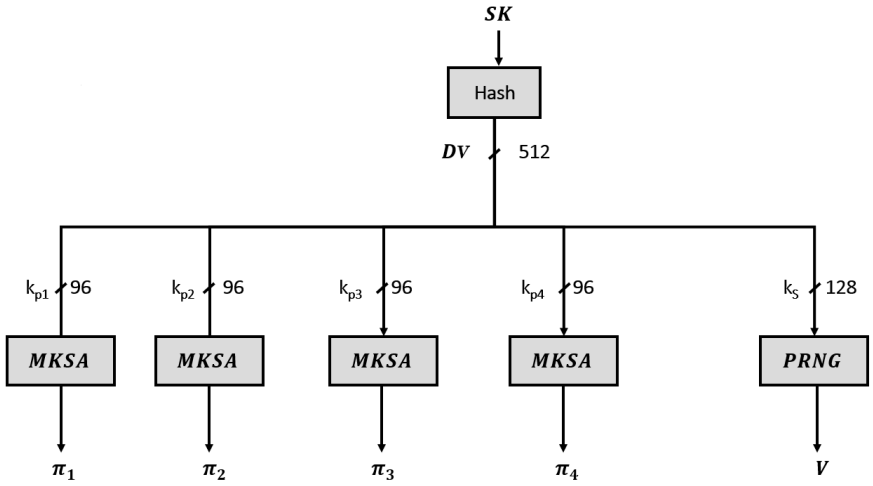


Figure 1: Generation technique of the dynamic keys and cryptographic primitives

these primitives. We show that the proposed scheme outperforms AES and the recent lightweight dynamic ciphers of [3], [5], [6].

In order to better highlight the contribution of this work, we list below the advantages of the proposed solution as compared to our previous works [5], [6]:

- LESCA uses a round function that can process h words in parallel instead of using a single word that is iterated recursively to generate a block of h words.
- In LESCA, all operations are performed at the word level (64 bits), while the prior techniques perform some operations at the byte level such as the substitution operation.
- In LESCA, a key-stream could be generated independently of other ones, while the prior works start with a word and perform an iterative approach in a chaining mode.
- In prior works, all the cryptographic primitives are updated after an iteration, while LESCA performs selective update of the primitives. A full update takes place only after δ blocks. Increasing δ will further reduce the computational overhead.
- LESCA avoids the substitution operation and ensures the confusion property by performing bit rotation, which reduces the computational overhead and associated latency.

III. Proposed Key Derivation Function

This section describes the proposed construction technique of the dynamic cryptographic primitives. Table I lists all notations used in this paper, and the construction process is illustrated in Figure 1.

We assume that every IoT device (Di) has a built-in secret $(S(Di))$ that is shared with the application server(s). The process starts when a new session is to be established between an IoT device (Di) and an application server. First, a new shared secret session key (SK) is generated and shared by adopting any protocol for key management as in [27]. Next, the common secret $(S(Di))$ is used as a seed to a pseudo-random generator to generate a Nonce (No) simultaneously at the IoT device and at the application server; the Nonce (No) is not transmitted. Then, (SK) and $(S(Di))$ are XOR-ed, and the result is fed to a cryptographic hash function such as SHA-512 to produce a 512-bit dynamic vector, DV.

Please note that this paper does not focus on lightweight hash functions, which have been investigated by many researchers and several such algorithms have been proposed and evaluated for their lightweight implementations. The majority of these algorithms are based on the Sponge construction [28], such as Spongent [29], PHOTON [30], GLUON [31], Keccak [32], Quark [33], Neiva [34], and others.

The proposed algorithm requires a secure Hash function only as an initialization step at the beginning of a session. The reference to SHA-512 has been made to indicate the required size to be generated (512 bits), and it could be instead replaced, for example, by Keccak that is used in very lightweight scenarios such is the case of RFID.

The dynamic vector, DV, is split into 5 components (sub-keys), $DV = \{kp_1,\ kp_2,\ kp_3,\ kp_4,\ k_S\}$, which are used to generate the cryptographic primitives and their corresponding update primitives. Note that DV and the generated primitives are unknown to attackers, which reinforces the security level of the proposed scheme when compared to the existing cryptographic techniques that rely on the session key.

The first four sub-keys of the dynamic vector $\{kp_1,\ kp_2,\ kp_3,\ kp_4\}$ have each a size of 96 bits (12 bytes), and the last sub-key, k_S, has a size of 128 bits (16 bytes). These sub-keys are used as follows:

- **Permutation sub-key** kp_i: it is employed to build the i^{th} permutation table π_i and it consists of the i^{th} set of the most significant 12 bytes of DV, for i =1, 2, 3 and 4. Each permutation table is generated using the modified key setup algorithm of RC4, as described in [3]. A permutation table is used as a selection table, and its values range between 1 and h, where h is the number of words in an input block.

- **Pseudo-random sub-key** k_S: it represents the least significant 16 bytes of DV, and it is employed to produce a pseudo-random vector, V, of size h words. k_S is used as a seed in any pseudo-random generator (PRNG) or stream cipher, to produce the vector V. The word size is decided according to the devices' characteristics. Note that, for each new input block, V is updated during the encryption process, and all cryptographic primitives are changed for every set of (δ) input blocks or messages.

All cryptographic primitives are extremely sensitive to any slight modification in any bit of the session key, which results into a completely different DV, and consequently, different cryptographic primitives. Also, this improves the ciphertext randomness, uniformity, and independence, and makes the cryptanalysis very challenging.

IV. LESCA Stream Cipher Scheme

An input message M is divided into nb blocks $M = m_1, m_2, \ldots, m_{nb}$, where each block has a

length of h words. The value of h affects the security level and the impact on performance; for real-time applications, a low value of h is preferable.

As shown in Figure 2, the j^{th} ciphertext block (c_j) is obtained by mixing the j^{th} plaintext block (m_j) with the j^{th} produced key-stream block V_j. The process is detailed in Algorithm 1, and expressed in Eq. 1:

$$c_j = E_K(m_j) = m_j \oplus V_j \; ; \;\; j = 1,\ 2,\ \ldots,\ nb \quad (1)$$

the produced key-stream (Vector V) is updated iteratively: the $(j+1)^{th}$ vector is updated as a function of the j^{th} vector.

The proposed stream cipher consists of two sub-functions:

1) Round Function (RF): applied once to initialize the key-stream V, which afterwards gets updated in an iterative manner. The $(j+1)^{th}$ encrypted block c_{j+1} is obtained by mixing (xor) the $(j+1)^{th}$ plaintext block with the $(j+1)^{th}$ key-stream V.
2) Update PrimitivesFunction (UPF): applied once every δ blocks or messages, to update all the cryptographic primitives used in encryption (V, π_1, π_2, and π_3) except for π_4.

A. The Round Function (RF)

The first step is to convert each input block of plaintext or ciphertext to word precision Wp for example, a size of 32 or 64 bits). Each block will have h words, which is a common value between the source and destination. In this work, we set h to 32 and the word precision Wp to 64 bits. Hence, the plain or ciphertext blocks have the same representation as V, which consists also of h elements.

Next, to produce the j^{th} key-stream block V_j, the **Round Function (RF)** is iterated as follows:

1) Compute X by mixing three different elements of V, two of them are selected according to the first and the second permutation tables, and expressed as:
$$X = V[w] \oplus V[\pi_1[w]] \oplus V[\pi_2[w]]$$
for $w = 1,\ 2,\ \ldots,\ h$.
2) Iterate an efficient PRNG once, for each element of vector X, using the elements as seeds, and store each output in its corresponding index in vector V; this updates the vector V. As a proof of concept, we set the word precision, Wp, to 64, and we choose XorShift64 as a PRNG due to its low computational complexity. Note that any other secure and efficient PRNG can be used instead of XorShift64.
3) Rotate right each output word of V using the ROR_{Wp} function, and this operation is controlled by the values of the first permutation table, that is the first element of V is rotated $\pi_1[1]$ times. This operation increases the non-linearity, randomness, and uniformity, and it was defined in Speck and Simon Cipher [13] for a word precision Wp of 64 and 128 bits. The output of this operation represents the key-stream of the j^{th} iteration.
4) XOR the jth plaintext block m_j with its corresponding key-stream word V_j.

The same process is adopted for the construction of all the required key-streams.

Algorithm 1 The proposed LESCA round function

Input: j^{th} Plaintext block m_j, Key-stream vector (V), two permutation tables (π_1, π_2)
Output: i^{th} ciphertext block c_i

procedure LESCA_SC($V,\ \pi_1,\ \pi_2,\ \pi_3,\ \pi_4$)
  for $j = 1 \rightarrow nb$ **do**
    if $j\%\delta = 0$ **then**
      $Update\_Primitives(V,\ \pi_1,\ \pi_2,\ \pi_3,\ \pi_4)$
    end if
    $X = V_{j-1} \oplus V_{j-1}(\pi_1) \oplus V_{j-1}(\pi_2)$
    $V_j \leftarrow PRNG(X)$
    $V_j \leftarrow ROR64(V_j, \pi_1)$
    $c_j \leftarrow m_j \oplus V_j$
  end for
  return $c_1||c_2||\ldots||c_{nb}$
end procedure

All the plaintext blocks are encrypted with their corresponding key-stream blocks, to produce the ciphertext $C = c_1||c_2||\ldots||c_{nb}$.

The decryption algorithm follows the same processes for the dynamic key production and the construction of required cryptographic primitives, in addition to the key-stream blocks generation V_j; the latter are "xor-ed" with the ciphertext blocks c_j to obtain the plaintext message block m_j. That is, to recover the j^{th} plain block m'_j, the j^{th} ciphertext block c_j is mixed (XOR) with the j^{th} produced key-stream word V_j, as expressed in Eq. 2:

$$m'_j = E_K(c_j) = c_j \oplus V_j \;\; , \; j = 1,\ 2,\ \ldots,\ nb \quad (2)$$

On the other hand, the non-linearity property of LESCA is ensured based on two factors: dynamicity (variable cryptographic primitives) and the bit rotation

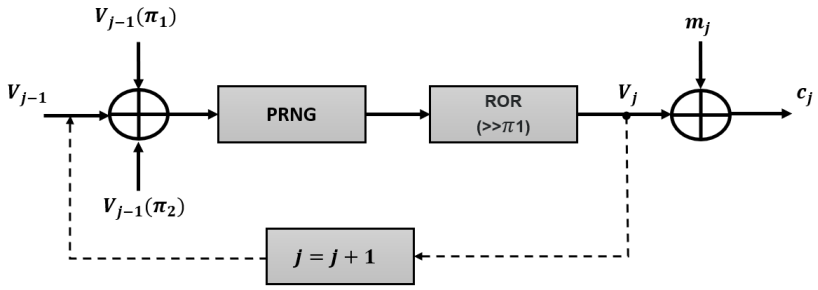


Figure 2: Structure of the proposed round function (RF) of LESCA stream cipher for the j^{th} block (iteration)

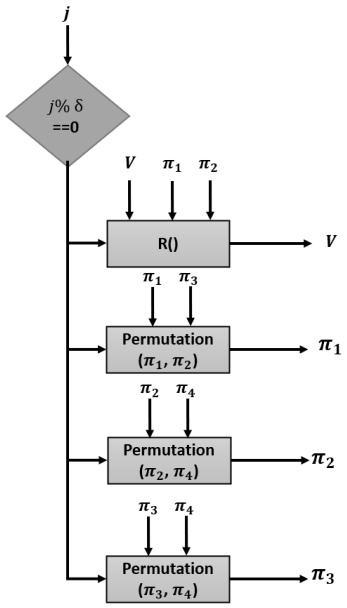


Figure 3: Structure of the proposed update function ($Update\_Primitives$) of LESCA stream cipher called in Algorithm 1.

operation. The vector V is changed after each iteration, and each word is diffused first with two other words, and confused using the bit rotation operation of the PRNG, and another final rotation using the right bit rotation. The diffusion operation is needed to ensure the global confusion property and consequently the key avalanche effect.

The bit rotation operation is inspired by 1) the RC6 non-linear function concept, which is used for that purpose, and 2) its low computational and memory consumption. The dynamicity property is ensured at two different levels: the variable initial words V for each new iterated plaintext block, and the update of all cryptographic primitives except π_4 after δ blocks, which further enhances the non-linearity level.

Next, we describe the update process of the cryptographic primitives.

B. Update_Primitives *Function*

For each δ input blocks (or messages), all cryptographic primitives are updated, except the last permutation table π_4, which is used only to update the third permutation table. The latter controls the update process of the first two permutation tables used in the key-stream generation algorithm (see Algorithm 1).

The update process starts by updating V using the R function, as described in SIMON/Speck [13], and it is controlled by the first and second permutation tables (π_1 and π_2) as follows:

$$V_{j,t} = R(V_{j,t}, V_{j,\pi 1[t]}, V_{j,\pi 2[t]}),\ \text{t= } 1,\ 2,\ \ldots,\ h \quad (3)$$

Each new element of V is updated using three current elements of V; one element is selected according to π_1, and the second using π_2. Then, the first three permutation tables are updated according to the following equations:

$$\pi 1 = Permutation(\pi 1,\ \pi 3) \quad (4)$$
$$\pi 2 = Permutation(\pi 2,\ \pi 4) \quad (5)$$
$$\pi 3 = Permutation(\pi 3,\ \pi 4) \quad (6)$$

Next, we provide an example on the use of "Xor-Shift64" as a PRNG.

1) Xorshift PRNG: To generate the vectors V, a PRNG is iterated, starting from the seed value X. For a word precision of $Wp = 64$, we selected "XorShift64", which provides an output block of 64 bits. The PRNG is iterated h times; in each iteration, an input seed value $X[w]$ is used, and the output corresponds to the element of V at index w.

Xorshift is a successor of the Linear-Feedback Shift Registers (LFSR) family, which is explained in Algorithm 2. The recent implementation of Xorshift makes it very fast since it avoids the use of excessively sparse polynomials [35], and non-linear operations. It is highly efficient, however, it fails certain statistical tests [35]. In the proposed scheme, this issue is resolved through the use of dynamic bit rotation to achieve the desired non-linearity degree. Note that any PRNG or a combination of different PRNGs can be used instead of Xorshift.

The following sections discuss the security and performance analysis needed to evaluate the robustness and efficiency of the proposed stream cipher.

Algorithm 2 xorshift64 PRNG

Input: 64-bit word of state t

Output: A produced random number word x with 64-bits length

1: **procedure** XORSHIFT64(t)
2: $x \leftarrow t$;
3: $x \leftarrow x \oplus (x >> 12)$;
4: $x \leftarrow x \oplus (x << 25)$;
5: $x \leftarrow x \oplus (x >> 27)$;
6: **return** x;
7: **end procedure**

V. Security Analysis

A cryptographic solution is designed to resist all sorts of analytical attacks such as statistical, brute-force, linear, and differential attacks [27], [36]. In this section, we assess LESCA's immunity against such attacks. We consider the worst-case scenario with input messages consisting of "all-zero" bytes and an attacker trying to compromise the secret key from the produced key-stream. We select a message length of 10,240 bytes, $h = 16$, and $\delta = 64$. Several tests such as the difference and sensitivity tests were repeated and averaged for 1,000 times.

A. Resistance Against Statistical Analysis

Statistical attacks can be avoided if the encrypted messages exhibit high uniformity and randomness levels. Hence, the produced key-stream should satisfy these properties, in addition to exhibiting a high periodicity [37].

The produced ciphertexts were tested using the typical benchmark statistical tests, PractRand [38] and TestU01 [39], [40], which are considered as the most difficult ones, and they guarantee that the generated key-streams satisfy the appropriate randomization and uniformity levels, and they confirm that no periodic pattern exists within the produced keystream(s). The results of the generated key-streams confirmed that the proposed stream cipher effectively passes these randomness tests, TestU01 and PractRand, with all of the examined seeds.

On the other hand, the visual results of the key-stream for a size of 10,240 bytes are illustrated in Figure 4-a), which illustrates the histogram of the key-stream with a random SK, and its corresponding recurrence in Figure 4-b). The results show that the generated key-stream exhibits a random recurrence since the values are (distributed over the whole space), and it matches the uniform distribution (all symbols have the same probability of occurrence). More details on the implementation of these tests are described in [3].

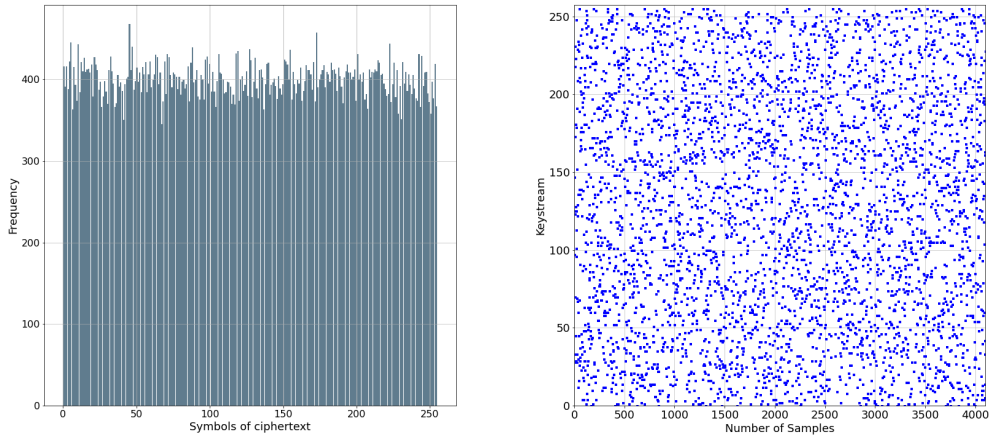
B. Key Sensitivity Test

The proposed stream cipher relies on the dynamic key-dependent approach, where all cryptographic primitives and their corresponding update primitives are changed for each new session. Moreover, the proposed round function updates the vector V for each input block. The update process guarantees the generation of different key-streams, which increases the non-linearity degree and extends their periodicity.

The key sensitivity test quantifies the percentage difference between the produced key-streams for a bit difference in the secret session key. In the proposed solution, the secret session key is hashed using (SHA-512) to produce DV, which guarantees a high sensitivity for a bit change in the input. Hence, if the same plaintext is encrypted twice, the algorithm generates two different ciphertexts as different cryptographic primitives are used. Figure 5 shows the proposed stream cipher's key sensitivity for 1,000 random dynamic keys and key-stream length of 10,240 bytes. The difference in the produced cipher-texts (key-streams) is very close to 50% and hence, this validates the key avalanche effect.

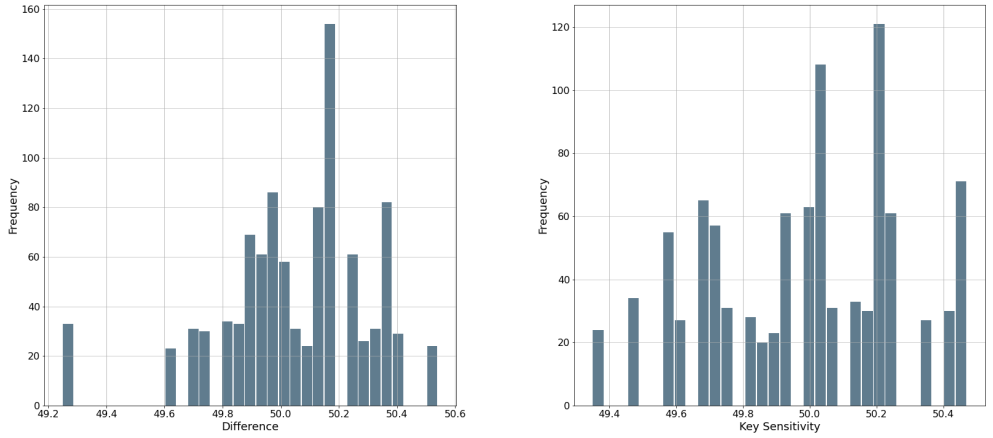
C. High Periodicity

The key-stream periodicity is strongly linked to the PRNG being used, the periodicity of the permutation tables, the block length h, and the word precision Wp.



(a) (b)

Figure 4: The corresponding (a) histogram, and (b) recurrence, of the produced key-stream for a random dynamic key ($h = 16$).



(a) NS (b) KS

Figure 5: Difference (a) and Key (b) sensitivity results for the proposed stream cipher considering 1,000 random keys.

The proposed cipher uses a perturbation technique by updating V (after each input block) and the permutation tables (π_1, π_2, and π_3), after each δ blocks; and the update process uses four permutation tables to maximize this effect. Hence, this results into a high periodicity, and a minimal probability of repeated key-streams.

D. Analytic and Brute Force Attacks

With a single round and simple operations, the proposed cipher satisfies the confusion and diffusion properties. The round function includes addition, PRNG, mixing, and bit rotation operations. As previously stated, LESCA exhibits high degrees of

randomness and uniformity, and the visual results of the ciphertexts reflect these results, as well as the strong immunity to statistical attacks. Also, LESCA achieves a high key sensitivity. Hence, the scheme is immune against key-related attacks.

Note that the update process makes algebraic, linear, and differential attacks very challenging. All tested analytical attacks on the stream cipher were unsuccessful. On the other hand, the secret key size is either 128, 196, or 256 bits, and the hash output, DV, is 512 bits long, which is sufficient to prevent brute force attacks.

Given that every new session uses a different key, that vector V is updated for every new block, and the primitives are updated regularly, the same plaintext/ciphertext will be encrypted/decrypted differently either within the same session and in different sessions. Hence, the proposed scheme is immune to analytic attacks.

VI. Performance Analysis

The performance of LESCA is evaluated in this section to validate its efficiency through two important metrics, the error propagation rate and the computational delay. The experimental set up is based on values for $h = 16$ and $\delta = 64$, and a message length varying between 2^6 and 2^{18} bytes. The throughput measures were collected for 1,000 times on actual hardware devices including Raspberry Pi, Teensy, and ESP32.

A. Error Propagation Rate

Being a stream cipher, a bit error in an encrypted block, c_i, will only impact the block's corresponding bit in the decrypted block. As such, LESCA is very suitable for noisy wireless channels.

B. Encryption Computational Delay

One of the main objectives of this work is to achieve a high degree of security with minimal computational complexity and energy consumption. We computed the delays associated with all of LESCA's components to get the overall delay:

1) T_{xor} is the time it takes to perform an XOR operation between two blocks, each with a length of h words.
2) T_{PRNG} is the time it takes to iterate the selected PRNG to produce a word.
3) T_{ROR} is the time needed to rotate right the bits of one word.
4) T_P is the time required to permute h elements of a block.

The total Computational Delay (CD_{KSGW}) to produce a single key-stream word of Wp bits is:

$$CD_{KSGW} = (3 \times T_{xor} + T_{PRNG} + T_{ROR}) \quad (7)$$

To encrypt a single block of h words, the required CD is:

$$CD = h \times (CD_{KSGW} + T_{xor}) \quad (8)$$

The delay associated with the construction of cryptographic primitives (CD_{CCP}) is given by the following equation:

$$CD_{CCP} = T_H + 4T_{MKSA}(h) + T_{PRNG} \quad (9)$$

where:

1) T_H stands for the time needed to hash an N-byte block.
2) $T_{MKSA}(x)$ stands for the needed execution time of the modified KSA of RC4 for a table with x elements.
3) T_{PRNG} is the needed execution time of a PRNG to generate h seeds.

It is worth noting that lowering the parameter ($delta$) increases the security level while also increasing the delays, and vice versa. The value of $delta$ is configured depending on the application needs and the required security level.

C. The Cipher Throughput

The average throughput of the proposed stream cipher is computed by conducting 1,000 runs with a message length varying between 2^6 and 2^{18} bytes on the hardware platforms, Raspberry Pi, Teensy, and ESP32 that may be employed in emerging systems. The tests were performed on Raspberry device classes (RPI0, RPI3, and RPI4), and two Teensy classes (Teensy 3.6 and Teensy 4), and the (characteristics of these devices are listed in Table III). We consider only the key-stream throughput since it accurately represents the encryption/decryption throughput and it involves the same number of operations.

Based on the experimental results, the best performance is achieved for h=16. Figure 6 shows the throughput results of the proposed cipher as a function of the message length, for different Raspberry Pi classes, and with $h = 16$ and $\delta = 64$. The results are compared to those of AES OpenSSL in CTR operation mode (making it a stream cipher), LORCA Stream Cipher (SC) [6], which exhibits a better throughout

Table II: Statistical Results of security sensitivity tests for $h = 16$

Security Tests	min	Avg	max	STD
ρ	-0.044	-0.0019	0.0486	0.0156
H_C	7.941	7.954	7.967	0.004
Dif	49.258	50.051	50.54	0.253
KS	49.35	49.983	50.469	0.287

Table III: Employed devices during bench-marking

Device	Characteristics
RPI0	• RAM: 512MB • Processor: 1GHZ, Broadcom BCM2835 (32bits)
RPI3	• RAM: 1GB • Processor: 1.2GHz ARM Cortex-A53 (64bits)
RPI4	• RAM: 4GB • Processor: 1.5GHz ARM Cortex-A72 (64bits)
Teensy 3.6	• RAM: 256MB • Processor: 180MHz Cortex-M4F (32bits)
Teensy 4	• RAM: 1GB • Processor: 600MHz Cortex-M7 (64bits)
ESP32	• RAM: 320MB • Processor: 160MHz Tensilica Xtensa LX6 (32bits)

compared to recent lightweight cipher schemes such as the enhanced one round of [4] and Simon and Speck [13].We shall refer to the former as AES-CTR, and the latter as One-Round. Similar results were obtained for the decryption process.

Note that the optimized implementation of AES with OpenSSL is possible only on devices that support operating systems such as Raspier-Pi devices. For other devices such as Teensy, we used another optimized implementation of AES.

The throughput results in Figure 6 show that LESCA outperforms AES-CTR, and the recent LORCA-SC stream cipher, on all devices versus different message lengths. The numerical results of the throughput ratio for the proposed cipher against AES-CTR, the recent One-Round cipher scheme [4], LORCA-SC [6], finalist NIST Grain-128 v2 [41] stream cipher, and the Lightweight Stream Cipher (LSC) of [5] are presented in Tables IV and V for different hardware platforms. Furthermore, Figure 7 provides a visual representation of the results versus AES-CTR and LORCA.

The throughput ratio of LESCA over that of AES-CTR varies between 3.61 and 8.72 for RPI0, between 3.66 and 8.61 for RPI3, and between 4.12 and 8.70 for RPI-4. Also, it varies between 1.72 and 1.87 for Teensy 3.6, between 3.0 and 3.85 for Teensy 4, and between 2.224 and 2.315 for ESP32. Moreover, the throughput ratio of LESCA over that of the Enhanced One-Round [4] varies between 1.12 and 2.13 for RPI0, between 1.50 and 2.917 for RPI3, and between 1.724 and 2.92 for RPI4. It varies between 1.30 and 1.45 for Teensy 3.6, between 1.40 and 1.85 for Teensy 4, and between 1.195 and 1.568 for ESP32. When compared to LORCA-SC [6], the throughput ratio varies between 1.01 and 1.33 for RPI0, between 1.02 and 1.93 for RPI3, and between 1.43 and 1.76 for RPI4. Higher throughput ratio of LESCA is obtained with the finalist stream cipher GRAIN 128 v2 on all RPI devices, varying between 13.98 and 37.94. These results indicate clearly the efficiency of the proposed solution compared to all recent techniques.

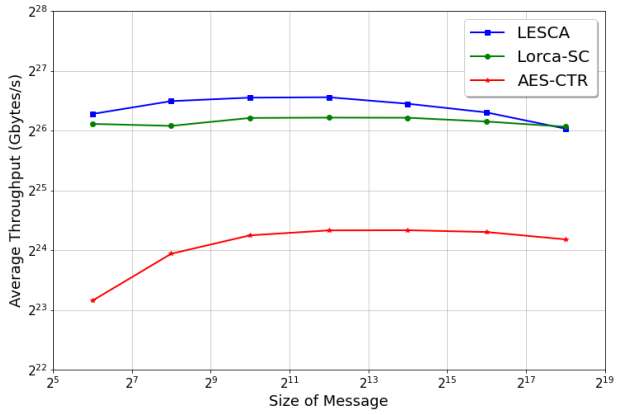
The maximum throughput of LESCA over AES-CTR, LORCA-SC and other listed stream ciphers is achieved on RPI4. However, the performance is slightly reduced on RPI3, and more so on RPI0. Note that the optimized AES-CTR implementation leverages the hardware enhancement of RPI4 compared to previous classes of RPI.

On the other hand, LESCA was implemented in C, but AES-CTR is written in assembly language, and it is highly optimized. A significant gain can be attained by optimizing LESCA using assembly language in a similar manner to AES-CTR.

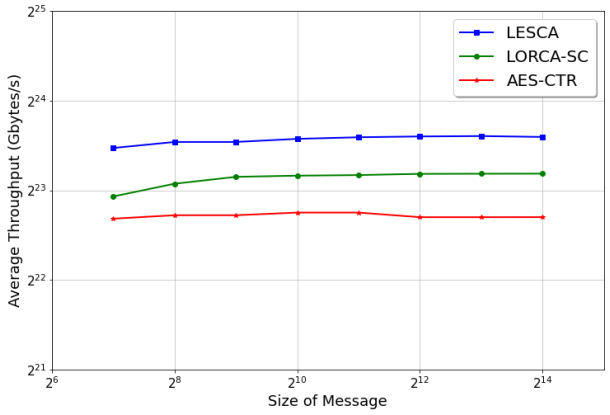
Note that One-Round has been validated in terms of its efficiency, and it was compared to the state-of-the-art lightweight cipher algorithms such as Simon and Speck. In this work, the results demonstrate that the proposed LESCA scheme outperforms One-Round. The numerical results in Table IV and Table V validate this conclusion.

D. Memory Consumption

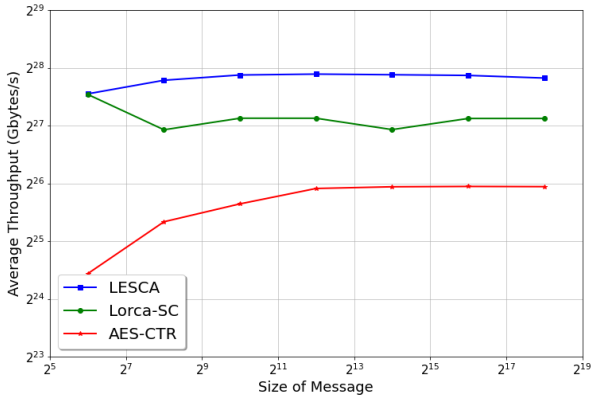
The encryption process of LESCA involves one input block (h words), a vector V (h words), and two permutations tables (π_1 and π_2), each with h elements varying between 0 and $(h-1)$. For the update process, the scheme requires two permutation tables (π_3 and π_4) of h elements. Thus, the required memory consumption



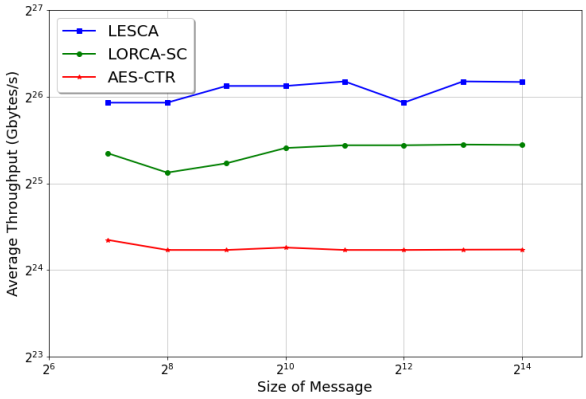
(a) RPI0



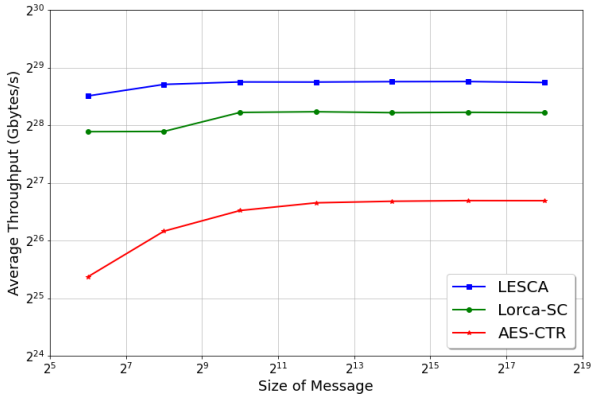
(b) Teensy 3.6



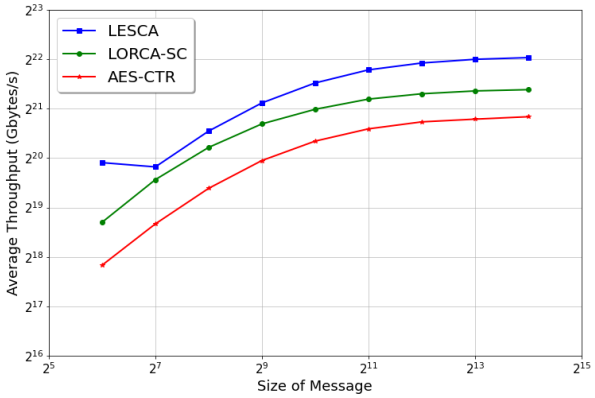
(c) RPI3



(d) Teensy 4

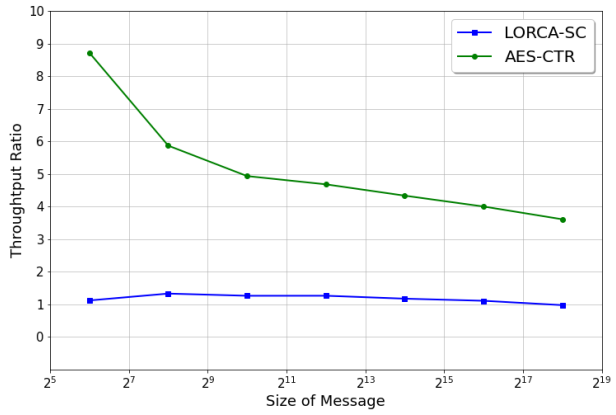


(e) RPI4

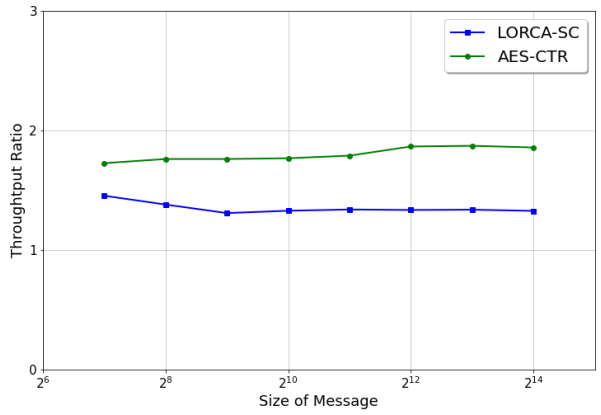


(f) ESP32

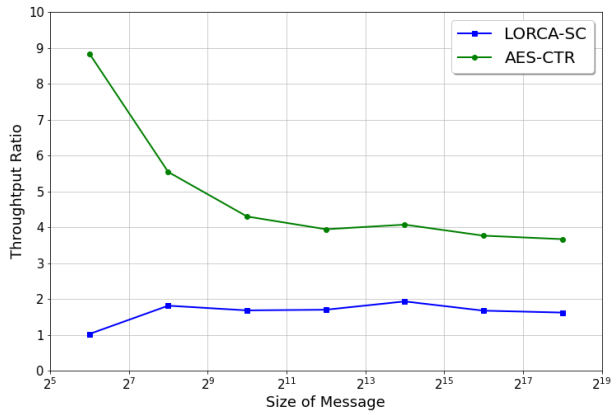
Figure 6: Encryption throughput (GBytes/s) versus message size on (a) RPI0, (b) Teensy 3.6, (c) RPI3, (d) Teensy 4, (e) RPI4, and (f) ESP32, for LESCA, LORCA-SC [6], and AES-CTR, with $h = 16$ and $\delta = 64$.



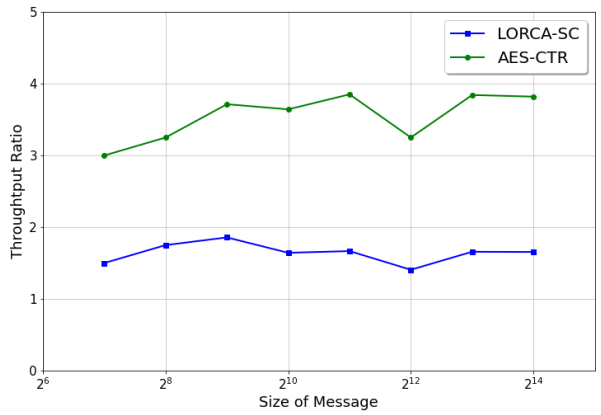
(a) RPI0



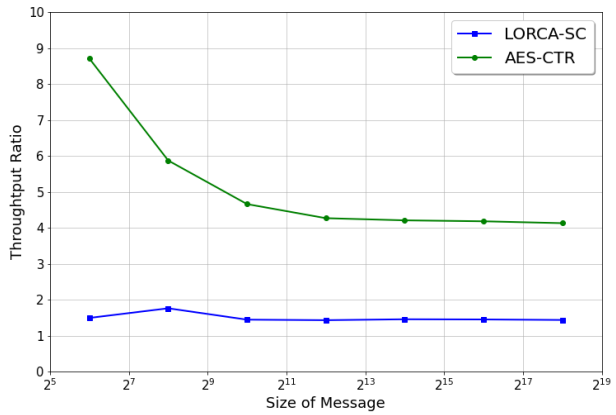
(b) Teensy 3.6



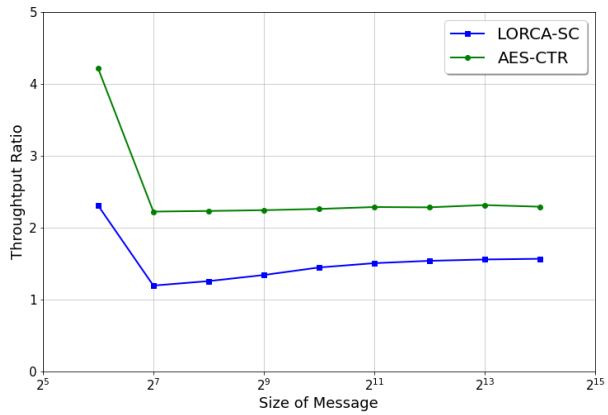
(c) RPI3



(d) Teensy 4



(e) RPI4



(f) ESP 32

Figure 7: Throughput Ratio variation (MB/s) in function of message size on (a) RPI0, (b) Teensy3.6, (c) RPI3, (d) Teensy4, (e) RPI4, and (f) ESP32, for $h = 16$ and $\delta = 64$, for LESCA, LORCA-SC [6] and AES-CTR.

Table IV: Speedup comparison of LESCA versus AES-CTR, LORCA-SC [6], enhanced One-Round of [4], finalist NIST Grain-128 v2 [41] stream cipher and Lightweight Stream Cipher (LSC) of [5], on a set of RPI devices

Hardware	Protocol comparison	Message Length (Bytes)						
		2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}
RPI0	LESCA vs Enhanced One-Round	1.81	2.13	2.06	1.56	1.34	1.24	1.12
	LESCA vs LSC	1.32	1.46	1.53	1.55	1.44	1.39	1.33
	LESCA vs LORCA-Stream cipher	1.12	1.33	1.27	1.27	1.18	1.11	1.01
	LESCA vs GRAIN-128	23.43	20.98	20.01	19.74	18.39	16.74	13.98
	LESCA vs AES-CTR	8.72	5.87	4.94	4.68	4.34	4	3.61
RPI3	LESCA vs Enhanced One-Round	1.87	2.92	2.58	1.82	1.61	2.18	1.5
	LESCA vs LSC	1.92	2.15	2.64	2.25	2.28	2.27	2.22
	LESCA vs LORCA-Stream cipher	1.02	1.81	1.68	1.7	1.93	1.67	1.62
	LESCA vs GRAIN-128	37.94	34.12	33.73	33.53	32.96	32.6	31.54
	LESCA vs AES-CTR	8.62	5.54	4.3	3.94	4.07	3.76	3.67
RPI4	LESCA vs Enhanced One-Round	1.72	2.72	2.9	2.07	1.88	1.85	1.82
	LESCA vs LSC	1.82	2.09	2.12	2.11	2.13	2.14	2.11
	LESCA vs LORCA-Stream cipher	1.49	1.76	1.44	1.43	1.45	1.45	1.43
	LESCA vs GRAIN-128	32.21	32.75	31.16	30.82	30.93	30.42	30.21
	LESCA vs AES-CTR	8.71	5.87	4.66	4.27	4.21	4.18	4.13

Table V: Speedup comparison of LESCA versus AES-CTR, and LORCA-SC [6], on Teensy 3.6, Teensy 4, and ESP32

Hardware	Protocol comparison	Message Length (Bytes)							
		2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}
Teensy 3.6	LESCA vs LORCA-SC	1.45	1.38	1.31	1.33	1.34	1.34	1.34	1.33
	LESCA vs AES-CTR	1.73	1.76	1.76	1.77	1.79	1.87	1.87	1.86
Teensy 4	LESCA vs LORCA-SC	1.5	1.75	1.86	1.64	1.67	1.41	1.66	1.65
	LESCA vs AES-CTR	3	3.25	3.71	3.64	3.85	3.25	3.84	3.82
ESP32	LESCA vs LORCA-SC	1.2	1.26	1.34	1.45	1.51	1.54	1.56	1.57
	LESCA vs AES-CTR	2.22	2.23	2.24	2.26	2.29	2.28	2.32	2.29

to encrypt/decrypt a block of h words is $(2 \times h)$ words + $(4 \times h)$ elements. In the case of devices with limited memory capacity, two permutations tables are preferred instead of 4 to reduce the required memory consumption. In this case, we can update π_1 in function of π_2 and π_2 in function of the updated π_1. This reduces the required memory to $(2 \times h)$ words + $(2 \times h)$ elements. Note that AES-CTR requires an input key of size (16, 24 or 32 bytes) and r round keys ($r \times 16$ bytes), in addition to 256 bytes for the substitution table. As for the one-round cipher scheme, it requires two substitution tables (256 bytes each), a permutation table with nb elements, in addition to two sets of round keys.

VII. Conclusion & Future Work

In this paper, we presented LESCA, a lightweight stream cipher scheme suitable for constrained devices and real-time applications. It consists of two lightweight functions: a round function and another function to update the cryptographic primitives. The two functions are applied once, and they make use of simple logical operations (a bit rotation and "exclusive-or"), a lightweight PRNG, and a permutation operation. LESCA adopts the dynamic key-dependent structure to achieve an optimal balance between the security level and performance. The update process is performed after every $(delta)$ blocks to update all cryptographic primitives used in the key-stream generation of the round function. When compared to traditional ciphers like AES and to recent one-round cipher schemes, LESCA reduces significantly the execution time and required resources. The security tests and performance evaluations confirmed LESCA's efficacy and robustness when compared against recent lightweight ciphers and the optimized implementation of AES. Finally, this work follows the new trend in the design of lightweight cryptographic algorithms for an optimized trade-off between the security and performance levels.

Funding:

The EIPHI Graduate School (contract "ANR-17-EURE-0002") provided funding for this work.

References

[1] M. S. Turan, K. McKay, D. Chang, C. Calik, L. Bassham, J. Kang, J. Kelsey *et al.*, "Status report on the second round of the

nist lightweight cryptography standardization process," *National Institute of Standards and Technology Internal Report*, vol. 8369, no. 10.6028, 2021.

[2] Noura, Hassan N and Noura, Mohamad and Chehab, Ali and Mansour, Mohammad M and Couturier, Raphaël, "Efficient and secure cipher scheme for multimedia contents," *Multimedia Tools and Applications*, vol. 78, no. 11, pp. 14 837–14 866, 2019.

[3] H. Noura, A. Chehab, L. Sleem, M. Noura, R. Couturier, and M. M. Mansour, "One round cipher algorithm for multimedia iot devices," *Multimedia tools and applications*, vol. 77, no. 14, pp. 18 383–18 413, 2018.

[4] H. N. Noura, A. Chehab, and R. Couturier, "Efficient & secure cipher scheme with dynamic key-dependent mode of operation," *Signal processing: Image communication*, vol. 78, pp. 448–464, 2019.

[5] H. Noura, R. Couturier, C. Pham, and A. Chehab, "Lightweight stream cipher scheme for resource-constrained iot devices," in *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2019, pp. 1–8.

[6] H. N. Noura, O. Salman, R. Couturier, and A. Chehab, "Lorca: Lightweight round block and stream cipher algorithms for iov systems," *Vehicular Communications*, p. 100416, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214209621000851

[7] P. Zhang, Y. Jiang, C. Lin, Y. Fan, and X. Shen, "P-coding: secure network coding against eavesdropping attacks," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[8] L. Pradeep and A. Bhattacharjya, "Random key and key dependent s-box generation for aes cipher to overcome known attacks," in *International Symposium on Security in Computing and Communication*. Springer, 2013, pp. 63–69.

[9] T. Kwon, H. Lee, S. Choi, J. Kim, D.-H. Cho, S. Cho, S. Yun, W.-H. Park, and K. Kim, "Design and implementation of a simulator based on a cross-layer protocol between mac and phy layers in a wibro compatible. ieee 802.16 e ofdma system," *Communications Magazine, IEEE*, vol. 43, no. 12, pp. 136–146, 2005.

[10] M. Dworkin, "Nist sp 800-38a: Recommendation for block cipher modes of operation," *National Institute of Standards and Technology (NIST), USA*, 2001.

[11] W. Zhang, Z. Bao, D. Lin, V. Rijmen, B. Yang, and I. Verbauwhede, "Rectangle: a bit-slice lightweight block cipher suitable for multiple platforms," *Science China Information Sciences*, vol. 58, no. 12, pp. 1–15, 2015.

[12] F. Karakoç, H. Demirci, and A. Harmancı, "Akf: A key alternating feistel scheme for lightweight cipher designs," *Information Processing Letters*, vol. 115, no. 2, pp. 359–367, 2015.

[13] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "Simon and speck: Block ciphers for the internet of things." *IACR Cryptology ePrint Archive*, vol. 2015, p. 585, 2015.

[14] G. Yang, B. Zhu, V. Suder, M. D. Aagaard, and G. Gong, "The simeck family of lightweight block ciphers," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 307–329.

[15] V. Nalla, R. A. Sahu, and V. Saraswat, "Differential fault attack on simeck," in *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems*, 2016, pp. 45–48.

[16] S. Kim and I. Lee, "Iot device security based on proxy re-encryption," *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, no. 4, pp. 1267–1273, 2018.

[17] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "Twine: A lightweight, versatile block cipher," in *ECRYPT Workshop on Lightweight Cryptography*, vol. 2011, 2011.

[18] Y. Wei, P. Xu, and Y. Rong, "Related-key impossible differential cryptanalysis on lightweight cipher twine," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 2, pp. 509–517, 2019.

[19] L. Li, B. Liu, and H. Wang, "Qtl: a new ultra-lightweight block cipher," *Microprocessors and Microsystems*, vol. 45, pp. 45–55, 2016.

[20] L. Li, B. Liu, Y. Zhou, and Y. Zou, "Sfn: A new lightweight block cipher," *Microprocessors and Microsystems*, vol. 60, pp. 138–150, 2018.

[21] H. Noura, C. Guyeux, A. Chehab, M. Mansour, and R. Couturier, "Efficient Chaotic Encryption Scheme with OFB Mode," *International Journal of Bifurcation and Chaos*, vol. 29, no. 05, 2019.

[22] M. Hell, T. Johansson, and W. Meier, "Grain: a stream cipher for constrained environments," *IJWMC*, vol. 2, no. 1, pp. 86–93, 2007.

[23] M. Hell, T. Johansson, A. Maximov, and W. Meier, "A stream cipher proposal: Grain-128," in *2006 IEEE International Symposium on Information Theory*. IEEE, 2006, pp. 1614–1618.

[24] K. A. McKay, L. Bassham, M. S. Turan, and N. Mouha, "Report on lightweight cryptography," *NIST DRAFT NISTIR*, vol. 8114, 2016.

[25] A. Y. Poschmann, "Lightweight cryptography: cryptographic engineering for a pervasive world," in *PH. D. THESIS*. Citeseer, 2009.

[26] R. Melki, H. N. Noura, M. M. Mansour, and A. Chehab, "An efficient ofdm-based encryption scheme using a dynamic key approach," *IEEE Internet of Things Journal*, 2018.

[27] W. Stallings, *Cryptography and network security: principles and practice*. Pearson Upper Saddle River, NJ, 2017.

[28] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Sponge functions," in *ECRYPT hash workshop*, vol. 2007, no. 9. Citeseer, 2007.

[29] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, "Spongent: the design space of lightweight cryptographic hashing," *IEEE Transactions on Computers*, vol. 62, no. 10, pp. 2041–2053, 2012.

[30] J. Guo, T. Peyrin, and A. Poschmann, "The photon family of lightweight hash functions," in *Annual cryptology conference*. Springer, 2011, pp. 222–239.

[31] T. P. Berger, J. D'Hayer, K. Marquet, M. Minier, and G. Thomas, "The gluon family: a lightweight hash function family based on fcsrs," in *International Conference on Cryptology in Africa*. Springer, 2012, pp. 306–323.

[32] E. B. Kavun and T. Yalcin, "A lightweight implementation of keccak hash function for radio-frequency identification applications," in *International Workshop on Radio Frequency Identification: Security and Privacy Issues*. Springer, 2010, pp. 258–269.

[33] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "Quark: A lightweight hash," *Journal of cryptology*, vol. 26, no. 2, pp. 313–339, 2013.

[34] K. Bussi, D. Dey, M. Kumar, and B. Dass, "Neeva: A lightweight hash function," *Cryptology ePrint Archive*, 2016.

[35] F. Panneton and P. L'ecuyer, "On the xorshift random number generators," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 15, no. 4, pp. 346–361, 2005.

[36] C. Paar and J. Pelzl, *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[37] W. Stallings, "Cryptography and network security: principles and practice," *Practice (6th Edition)*, vol. 9, p. 09685, 2008.

[38] C. Doty-Humphrey, "Practrand," 2014. [Online]. Available: http://pracrand.sourceforge.net/

[39] P. L'Ecuyer and R. J. Simard, "Testu01: A c library for empirical testing of random number generators," *ACM Trans. Math. Softw*, vol. 33, no. 4, pp. 22:1–22:40, 2007. [Online]. Available: http://doi.acm.org/10.1145/1268776.1268777

[40] L. Sleem and R. Couturier, "Testu01 and practrand: Tools for a randomness evaluation for famous multimedia ciphers," *Multimedia Tools and Applications*, vol. 79, no. 33, pp. 24 075–24 088, 2020.

[41] M. Hell, T. Johansson, A. Maximov, W. Meier, J. Sönnerup, and H. Yoshida, "Grain-128aeadv2-a lightweight AEAD stream cipher," *Information Technology, Laboratory COMPUTER SECURITY RESOURCE CENTER*, 2019.