LoRCA: Lightweight Round Block and Stream Cipher Algorithms for IoV Systems

Hassan N. Noura¹, Ola Salman², Raphaël Couturier¹ and Ali Chehab²

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

Abstract—The Internet of Vehicles (IoV) is a disruptive technology that has a great impact on people’s lifestyle and activities. Fully autonomous vehicles are the next generation of connected cars. However, IoV systems suffer from several security threats that could offset the intended advantages. To address the security and privacy issues, several cryptographic and non-cryptographic security solutions should be adopted. However, traditional security solutions might burden the IoV network with further complexity and computational overhead. This would translate into additional performance issues in a network already suffering from big data and large-scale challenges. In this context, lightweight security solutions are needed to reduce the required resources in terms of computations, power and memory, and to optimize the network performance in terms of latency and bandwidth. In this paper, we propose a new lightweight cipher scheme, LoRCA, with dynamic key-dependent structure to provide data confidentiality with minimum resources’ requirements. To validate our solution’s robustness and efficiency within the IoV context, several performance and security tests were performed. The results show that the proposed solution strikes a good balance between the security level and performance. The proposed LoRCA ciphers achieve a high throughput with an enhancement of at least 100% improvement as compared to the Advanced Encryption Standard (AES), 358% compared to Simon, 388 % improvement for Speck and 24% improvement compared to our previous one round cipher scheme.

Index Terms—Lightweight encryption; dynamic encryption; key-dependent primitives; security analysis; performance analysis.

I. INTRODUCTION

The Internet of Vehicles (IoV) is emerging as an extension of the Internet of Things (IoT) paradigm to interconnect all types of vehicles and to allow online access. This results into a large-scale network with a heterogeneous set of inter- connected devices, while enabling innovative applications to benefit from the huge amount of collected data. However, this will burden the network with new QoS challenges and security issues. In addition to the threats associated with traditional networks, IoV systems gave rise to new types of threats that could have a drastic impact on the privacy and security of data and systems. The security threats affect the different types of security services such as confidentiality, integrity, authentication, and availability (see Figure 1-a). To defend against these security threats, there are some existing solutions that can be divided into two types: cryptographic and non-cryptographic, as shown in Figure 1-b. Devices with good computational resources and reasonable memory capacity rely on traditional cryptographic algorithms to ensure the required security services. Specifically, data confidentiality and user privacy are essential requirements of any communication system such as IoV. Typically, data confidentiality is ensured by applying encryption, while user privacy can be achieved by using data minimization and pseudonymity features [1]–[5]. In general, data confidentiality is based on Symmetric Key Cryptography (SKC) since it is more efficient than Asymmetric Key Cryptography (AKC) (see Figure 2). The SKC Algorithms [6], [7] are based on a secret key shared between two entities.

Moreover, encryption algorithms can be performed at the block level or in stream mode. In stream cipher mode, the data is mixed (xor-ed) with a pseudo-random stream called "key-stream", usually at the byte level; for every byte of data, a random byte is generated and then, both bytes are xor-ed, and the output byte is transmitted as the cipher-byte. However, in the case of block cipher, the data is divided into blocks of fixed size, usually 128 bits. Then, a block cipher employs a reversible round function that is iterated r times on each block [8]. The round function should ensure two essential cryptographic properties according to the famous information theorist “Claude Shannon” [9]. These properties are:

• Confusion: It obscures the relationship between the secret key and cipher-text. Typically, the confusion property is satisfied by employing a substitution table, also called S-box or through the use of a nonlinear transformation.
• **Diffusion**: It ensures that the influence of one plaintext symbol impacts many cipher-text symbols, with the goal of hiding the statistical properties of the plain-text. The diffusion property can be ensured by using a Permutation table (P-box) such as the case of the Data Encryption Standard (DES) or by using an invertible diffusion matrix such as the “Matrix Distance Separate (MDS)” [10], that is adopted in AES in the “Mix-Column” operation.

The diffusion operation is a linear transformation, while the substitution operation provides a local avalanche effect at the byte (or word) level. Furthermore, the diffusion operation combined with substitution ensure the global avalanche effect after several iterations.

Additionally, a block cipher can be used as a stream cipher when the block cipher is employed to produce a key-stream sequence, which is the case of Output Feedback (OFB) and Counter (CTR) operation modes [11]. The security of a stream cipher is based on different metrics that depend on the quality of the produced key-stream sequences, which should exhibit high non-linearity, long periodicity, high level of randomness, and high uniformity degree.

Furthermore, a message authentication algorithm is necessary to guard against threats related to data integrity and source authentication. Message authentication algorithms can be based on a keyed hash function such as HMAC [12] or a block cipher with authentication
operation mode, such as CMAC and GMAC [7]. Note that in some cases, data secrecy reinforces privacy since encrypted surveillance videos, for example, may prevent attackers from identifying people.

Due to the fact that traditional SKC algorithms follow a static structure, the substitution and diffusion primitives should be chosen according to their maximum cryptographic performance. For example, the static AES S-box was designed to provide minimum differential probability (to defend against differential attacks) and minimum linear probability (to defend against linear attacks). Also, SKC algorithms use static diffusion operation with a high linear branch number, and a low number of fixed points.

A. Problem Formulation

Many IoV systems and applications are based on sensitive information that must be protected through the adoption of best practices in terms of data confidentiality. However, most of the existing encryption algorithms, as indicated previously, are computationally expensive ($r$ rounds and multi-operations per round) and hence, they are not appropriate for IoV real-time applications and for in-vehicle constrained devices (e.g., micro-controllers).

Even though the existing symmetric ciphers, which are based on a static structure, have shown resistance against linear and differential cryptanalysis attacks, such fixed structures lend themselves to future potential attacks [13], [14]. These attacks would target their fixed structure to recover the secret key. In this case, these existing solutions would have to increase the number of rounds leading to even higher latency and resources overhead. This would reduce further their performance and makes them not suitable for some future systems and applications such as IoV ones.

Accordingly, it is important to design secure and efficient cryptographic algorithms that optimize the trade-off between the security level and the performance overhead. One possible way to achieve this target is to modify the existing SKC structure by using the dynamic key-dependent approach to achieve higher efficiency and robustness against powerful future attacks.

B. Motivation

As shown in Figure 3, an IoV network consists of IoV devices that are connected to a data center (or application servers) through a gateway. The communication between IoV devices and the gateway is wireless. The IoV devices can be interconnected through several gateways. Therefore, a secure end-to-end encryption should be put in place between the IoV devices and the data center or applications servers.

In this context, this paper proposes a new cipher scheme that is suitable for constrained IoV devices, exhibiting the minimum possible amount of computations and latency, while maintaining the required security level. The proposed cipher ensures the data confidentiality security service. Furthermore, the structure of the proposed LoRCA block and stream ciphers consist of iterating two functions (round and update functions) for one round to generate a key-stream block. These functions consist of simple, yet efficient operations allowing them to achieve the needed cryptographic properties. The dynamic key approach is adopted in a way that a
different dynamic key is generated and used for each input message (audio, video or image).

C. Contributions

In previous works, we proposed cipher schemes based on the dynamic key approach and with a low number of rounds [15]–[18]. In this paper, the novelty of the proposed cipher schemes lies in the ability to achieve an excellent balance between performance and security for IoV devices, with the main advantages as listed below:

1) Minimum error propagation rate: compared to [18], LoRCA is applied at the block level, which reduces the error propagation rate.
2) Minimum overhead: LoRCA requires a small number of operations. Moreover, no block permutation operation is required, which minimizes delay and memory consumption. Also, compared to [15]–[17], LoRCA reduces the computational complexity by avoiding diffusion/chaining operations.
3) Easier implementation: the solution presented in [18] cannot be deployed on small devices such as Arduino boards or resource-restricted devices since it requires double the memory overhead.
4) Dynamic cipher primitives: LoRCA updates two cryptographic primitives (X and RM that are defined in Section III) after each encrypted/decrypted block. This operation is designed to consume a low overhead in terms of computational resources and latency, yet it offers a high security level.

On the other hand, the contributions of the proposed cipher solution in terms of robustness and efficiency are summarized as follows:

Performance Contributions:

- Efficiency: LoRCA avoids diffusion and requires a low number of operations to reduce both resources usage and computational complexity.
- Flexibility: LoRCA handles dynamic block lengths (N bytes), where N can be configured depending on the application and device constraints.
- Simple hardware/software implementations: Hardware (HW) and Software (SW) implementations of LoRCA are simple and efficient with the use of "Exclusive-OR" logical operation and look-up substitution and permutation operations.
- Error Tolerance: Compared to [15], [18], [19], LoRCA provides better resistance to channel errors. Each byte error in any encrypted block affects only the byte itself and does not propagate to other bytes in the corresponding block. In general, the proposed cipher variant is well suited for noisy communication channels since it exhibits low error propagation rate when compared to existing standard block ciphers such as the Advanced Encryption Standard (AES).

Security Contributions:

- Dynamic Key-Dependent Approach: Being a dynamic key-dependent cipher scheme, LoRCA re-
lies on cryptographic primitives that change in a pseudo-random manner for every input message. This hardens the statistical/implementation attacks, especially with the use of dynamic encryption keys [15], [19]–[21].

- **Dynamic Cipher Primitives**: Traditional block ciphers use static cipher primitives, which are the same for all input blocks, whereas LoRCA uses dynamic pseudo-random blocks (X and RM). This ensures that the relation among the encrypted blocks is more complex and random, which guarantees its immunity against different analytic attacks since for each input message block, the encryption/decryption is dynamic and variable.

**D. Organization**

This paper is organized as follows. In Section II, we review the exiting lightweight stream and block cipher schemes along with their limitations. Section III describes the generation of the dynamic key and the associated cryptographic primitives. The variants of LoRCA (stream cipher and block cipher) are detailed in Section IV. Extensive security analysis is presented in Section V to prove that the proposed cipher variants exhibit the desired cryptographic properties. Section VI assesses the robustness of the proposed cipher variants against various types of attacks, while the effectiveness of LoRCA is presented in section VII. Finally, this work is concluded in section VIII with directions for future work.

**II. Related Work**

As indicated previously, most of the traditional block cipher methods exhibit a high overhead in terms of computations, memory, and power consumption. For example, AES [22] with counter mode requires a relatively large number of rounds with diffusion operations within the round function [18]. Considering other traditional block ciphers, such as the Hummingbird2 cipher [18], the minimum number of required rounds is 4. Therefore, within the IoV context, the traditional cryptographic algorithms lead to poor performance [18]. On the other hand, the scientific interest towards lower overhead cryptography is not only confined to the IoV domain as indicated in [23].

In the ongoing effort to address the computational complexity issue, several lightweight ciphers have been proposed such as LED (PHOTON family) [24], [25], ITUbee [26], RECTANGLE (Substitution–Permutation Network (SPN) based) [27], AKF (Feistel based) [28], Simon and Speck [29]. The Speck algorithm exhibits a lower overhead than the Simon’s algorithm, making it more suitable for tiny devices. However, a multi-round structure is still being used by Speck, but with a simple round function. SIMECK, a combination of Speck and Simon, was proposed in [30]. In [31], SIMECK was proven to be vulnerable to bit-flip and random-byte attacks. More recently, other lightweight block ciphers were proposed including LiCi (SPN based) [32], BORON [33], PRESENT [34], GIFT [35], and CHAM [36].

Researchers also used elliptic curve to design lightweight cipher schemes [37]. A primary work, TWINE, was presented in [38], [39]. By combining SPN and the Feistel Network (FN), recent works were proposed including QTL [40], Substitution–permutation Fiestel Network (SFN) [41], to benefit from the advantages of both approaches (SPN and FN). However, using the traditional round function structures (SPN, FN and SFN) with static cryptographic primitives still requires a high number of rounds and operations. Therefore, these ciphers, by design, are not really suitable for limited IoV devices or real-time IoV applications.

Alternatively, chaotic cryptographic algorithms have been proposed to address this issue. However, these algorithms are prone to various security and performance issues such as the need for conversion operations, floating-point computations, and a finite periodicity, in addition to a complex hardware implementation. Moreover, these solutions also use a multi-round structure [42], [43].

On the other hand, stream cipher algorithms, considering only one block of data at a time, are by nature more lightweight than block ciphers, and they require less computational and memory overhead. In this context, different lightweight stream cipher solutions with software and hardware acceleration were proposed such as Grain [44], Trivium [45], Grain-128 [46], Salsa20 [47], Sosemanuk [48], MICKEY [49], Chacha [50], Encoro-80 [51], Encoro-128 [52], SNOW3G [53], A2U2 [54], and Quavium [55]. In the second generation of lightweight stream cipher cryptography, widely adopted stream ciphers include WG-8 [56], Sprout cipher [57], Fruit [58], Plantlet [59], Espresso [60], and Lizard [61]. However, as indicated previously, these solutions are based on a static round function that iterates for r rounds. Thus, reducing the number of rounds will make them prone to several statistical and analytical attacks.

In summary, the existing encryption algorithms are not designed to be applied for real-time IoV applications, and with tiny devices that are constrained in terms of computational power, resources, and battery lifespan. IoV applications and devices with stringent QoS re-
Table I: List of Notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SK</td>
<td>Shared secret session key</td>
</tr>
<tr>
<td>nonce</td>
<td>Nonce that changes per input message</td>
</tr>
<tr>
<td>DK</td>
<td>Dynamic key updated per input message</td>
</tr>
<tr>
<td>k_{S1} and k_{S2}</td>
<td>Substitution sub-keys</td>
</tr>
<tr>
<td>S_1 and S_2</td>
<td>Substitution tables</td>
</tr>
<tr>
<td>k_p</td>
<td>Permutation sub-key</td>
</tr>
<tr>
<td>k_{PRK}</td>
<td>PRNG Seed</td>
</tr>
<tr>
<td>R_M and IV</td>
<td>Pseudo-random blocks, where IV is required for chaining operation modes (CBC and CTR).</td>
</tr>
<tr>
<td>k_p_R_M</td>
<td>Permutation sub-Key used to produce the permutation table π_{R_M}</td>
</tr>
<tr>
<td>X</td>
<td>PRNG seed, which is updated after each PRNG iteration (each input block)</td>
</tr>
<tr>
<td>len</td>
<td>Length of input message</td>
</tr>
<tr>
<td>n_b</td>
<td>Blocks number per input message and is equals to ⌈len_n\⌉</td>
</tr>
<tr>
<td>h</td>
<td>Bytes number per block message</td>
</tr>
<tr>
<td>M</td>
<td>Original message</td>
</tr>
<tr>
<td>m_i</td>
<td>i^{th} original plain block</td>
</tr>
<tr>
<td>c_i</td>
<td>i^{th} encrypted block</td>
</tr>
<tr>
<td>KSA</td>
<td>Key Setup Algorithm of RC4</td>
</tr>
<tr>
<td>MKSA</td>
<td>Modified KSA of RC4 presented in [18].</td>
</tr>
<tr>
<td>S(m_i, S_1, S_2)</td>
<td>Substitute the bytes of m with odd and even indices by using first substitution table S_1 and S_2, respectively.</td>
</tr>
<tr>
<td>Y, X</td>
<td>XorShift^{64}(X) iterates the XorShift PRNG with an input value (block) X to produce a pseudo-random block y (round key).</td>
</tr>
<tr>
<td>x&lt;&lt;n</td>
<td>Left shift operator</td>
</tr>
<tr>
<td>x&gt;&gt;n</td>
<td>Right shift operator</td>
</tr>
</tbody>
</table>

Requirements need new security measures that cater to their limitations. Such algorithms and protocols must be “lightweight”: they should exhibit a low overhead in terms of resources, computational power, latency and overhead [62], [63].

To this end, researchers tend to design new lightweight round functions that consist of simple operations, or to reduce the number of rounds. Recently, various lightweight cryptographic algorithms have been proposed in [15], [16], [18], [64]. These solutions rely on the dynamic key approach to reduce the required number of rounds and to maintain a high-security level. For example, the cipher schemes described in [15]–[17] require two rounds, while the scheme described in [18] requires only one round and processes two blocks at a time, making it more efficient compared to the previous solutions [15]–[17].

In this work, we aim at enhancing the work done in [18] by designing new block and stream cipher variants that exhibit lower latency and memory consumption in addition to using variable cryptographic primitives (dynamic key-streams: X and R_M). The proposed cipher variants achieve a better throughput compared to [18], in addition to reinforcing the security level.

In the following, we describe the proposed key derivation function and the employed techniques to construct the required cryptographic primitives. Then, the proposed block and stream cipher are presented.

### III. Proposed Key Derivation Function

The used notations in this section are listed in Table I, and the steps of the dynamic key derivation are illustrated in Figure 4. The input to this function is a shared secret session Key (SK) and a nonce (a unique block that is used for only one session). SK can be renewed per session (or sub-session) according to the IoV application requirements and constraints. Note that the key management between the different IoV entities is outside the scope of this paper. For more information about possible key management solutions, please refer to [7], [65].

For every new input message (or a set of messages, depending on the configuration), a new dynamic key is generated by first xor-ing SK and a nonce and then, hashing the result, as indicated in Eq. 1:

\[
DK = hash_{SHA-512}(SK \oplus nonce) \quad (1)
\]

where the nonce can be generated by the communicating parties. The size of the nonce is 512 bits, with a corresponding space of potential nonces of \(2^{512}\), and hence a very low probability of nonce collision. Also, we use the secure hash function SHA-512 [66], which is highly resistant against collisions to produce the required dynamic key with a length of 512 bits (64 bytes). Figure 5 shows a numerical example of this process where all values are represented as bytes with decimal values.
Figure 4: Proposed technique to generate the dynamic key and cryptographic primitives

\[ SK \]

\[
\begin{array}{cccccccccccccccc}
129 & 157 & 236 & 68 & 225 & 16 & 240 & 139 & 212 & 156 & 247 & 229 & 103 & 159 & 164 & 250 \\
\end{array}
\]

\[ \text{Nonce} \]

\[
\begin{array}{cccccccccccccccc}
33 & 126 & 77 & 80 & 206 & 244 & 186 & 9 & 123 & 88 & 139 & 250 & 100 & 68 & 141 & 28 \\
\end{array}
\]

\[ SK \oplus \text{Nonce} \]

\[
\begin{array}{cccccccccccccccc}
160 & 227 & 161 & 20 & 47 & 228 & 74 & 130 & 175 & 196 & 124 & 31 & 3 & 210 & 41 & 230 \\
\end{array}
\]

\[ DK = hash(Secretkey \oplus \text{Nonce}) \]

\[
\begin{array}{cccccccccccccccc}
165 & 198 & 230 & 248 & 146 & 180 & 14 & 229 & 228 & 113 & 11 & 10 & 45 & 141 & 173 & 31 \\
176 & 231 & 125 & 118 & 80 & 163 & 79 & 61 & 183 & 236 & 126 & 235 & 169 & 168 & 219 & 162 \\
0 & 218 & 26 & 92 & 55 & 216 & 227 & 58 & 163 & 142 & 144 & 195 & 5 & 137 & 96 & 8 \\
\end{array}
\]

Figure 5: A numerical example of the dynamic key generation process
Figure 6: A numerical example of the generation of substitution tables (Sbox $S_1$ and Sbox $S_2$)

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>118</td>
<td>17</td>
</tr>
<tr>
<td>89</td>
<td>109</td>
</tr>
<tr>
<td>49</td>
<td>42</td>
</tr>
<tr>
<td>79</td>
<td>48</td>
</tr>
<tr>
<td>1</td>
<td>118</td>
</tr>
<tr>
<td>217</td>
<td>25</td>
</tr>
<tr>
<td>133</td>
<td>32</td>
</tr>
<tr>
<td>35</td>
<td>294</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
</tr>
<tr>
<td>53</td>
<td>204</td>
</tr>
<tr>
<td>136</td>
<td>3</td>
</tr>
</tbody>
</table>

On the other hand, NIST recommends 4 secure Deterministic Random Bit Generator (DRBG) algorithms that can be used for the generation of nonces or session keys. These algorithms are detailed in the NIST SP800-90A report [67], where two of them are based on hash functions, one being key-based and the other is unkeyed. The unkeyed DRBG hash-based technique can be used for nonce generation with LoRCA since it requires low overhead compared to the keyed one. This unkeyed technique ensures that the generated nonces have a high periodicity and low collision probability. Moreover, in practical scenarios, a new session key is produced for each new session. Hence, even if the same nonce is regenerated in a different session, a different dynamic key will be produced since it also depends on the session secret key. Note that, in the same session, the produced dynamic key $DK$ is constantly changing since the nonce can be changed per input message or for a set of messages (depending on the configuration). Consequently, the attacker should know which nonce and session key are used for each message (or a set of messages), which complicates the attacker task tremendously.

The produced dynamic key is divided into three subkeys $DK = \{k_{S1}, k_{S2}, k_R\}$. $k_{S1}$ and $k_{S2}$ have a size of 128 bits, while $k_R$ is 256 bits, as illustrated in Figure 4 and described next.
• **Substitution sub-key** $k_{S1}$: is used to construct the first substitution table $S_1$, and it consists of the first most significant 16 bytes of $DK$. The substitution table is generated by using KSA, which is described in the next subsection (see Algorithm 1). In fact, the substitution is applied at the byte level, with the values of elements in the tables $S_1$ or $S_2$ ranging between 0 and 255.

• **Substitution sub-key** $k_{S2}$: this sub-key consists of the next most significant 16 bytes of $DK$, it is used to construct the second substitution table, $S_2$, also by applying KSA but with a slight modification: $S_1$ is used as the initial substitution table, as described in the next subsection. As such, $S_2$ depends on the first 32 most significant bytes of $DK$. Figure 6 provides a numerical example for the generation of tables $S_1$ and $S_2$.

• **Pseudo-random sub-key** $k_R$: it consists of the least significant 32 bytes of $DK$, and it is used to generate a key-stream $R$ of $(3 \times h)$ bytes length. $R$ will be divided later into three equal parts, where each one has a length of $h$ bytes:
  1) Pseudo-random vector;
  2) An initial vector $IV$;
  3) An initial Condition $X$;

In this step, to generate $R$, we use RC4 [68], with $k_R$ as a seed and $S_2$ as initial substitution table. The first, second and third $h$ bytes of $R$ are used to form the $RM$, $IV$ and $X$ vectors, respectively. Thus, $RM$, $IV$ and $X$ depend on all bits of $DK$.

We reduce, modulo $h$, each byte of $X$ to produce the permutation key $k_{PRM}$, which will be used as a seed with the modified KSA algorithm [18] to generate the permutation table $\pi_{RM}$. The only required modification compared to the original KSA of RC4 is changing the length of the state table $L_S$ from 256 to $h$. The permutation table elements have values ranging from 0 to $h - 1$. As such, $\pi_{RM}$ depends on all the dynamic key bits. Note that $k_{PRM}$ has also $h$ elements with values varying between 0 and $h - 1$.

Figure 7 illustrates a numerical example for the construction of $\pi_{RM}$, along with $RM$, $IV$ and $X$.

All cipher primitives are highly sensitive to any bit change in the secret key, nonce, and also the dynamic key. Hence, a different nonce, session key or dynamic key will lead to totally different cipher primitives. As such, the proposed block and stream cipher algorithms exhibit a high sensitivity (key avalanche effect) in terms of session, nonce, and dynamic key. This also increases the randomness of the ciphertext and makes the cryptanalysis attacks even more difficult.

After detailing the key and sub-keys derivation, in the next sub-sections, we present the construction of the cryptographic primitives.

### A. Cryptographic Primitives Construction

In this work, we use simple techniques to construct the required cryptographic primitives that are based on the RC4 stream cipher algorithm or one of its corresponding steps, namely (KSA).

(KSA) and the Pseudo-Random Generator Algorithm (PRGA) are steps of RC4, that have to be implemented sequentially as shown in Figure 8.

KSA is used to produce the dynamic substitution tables, $S_1$ and $S_2$. The modified KSA is used to produce the required dynamic permutation table, $\pi_{RM}$. Also, RC4 is used to produce the key-stream, $R$, which is divided to form the remaining cryptographic primitives ($RM$, $IV$ and $X$).

1) **Substitution and Permutation Tables Construction:**

According to RC4, KSA (see Algorithm 1) has a variable key length, which ranges between 64 and 256 bits. KSA produces a state table $S \triangleq \{S[0], \ldots, S[L_S - 1]\}$ with $L_S = 256$ elements, ranging from $S[0]$ to $S[255]$. In Algorithm 1, the parameter $L_K$ is the length of the initial key in bytes, and $i$ and $j$ are iteration variables. For a secure use of the key setup of RC4, the input sub-key size should be set at least to 128 bits ($L_K = 16$).

The original implementation of KSA initializes the state table elements to their corresponding indices, such as $S[i] = i$ for $i = 0, 1, \ldots, 255$. However, in this paper, we propose a new state table initialization method, which sets the current state table to the previous one. This helps to reach the dynamic key avalanche effect. However, for the first iteration of (KSA) (case of the construction of the first substitution table $S_1$), the initialization of the original (KSA) is used, which means that $S[i] = i$ and $i = 0, 1, \ldots, 255$.

As indicated previously, we use the modified (KSA) to construct the permutation table $\pi_{RM}$, that requires to modify $L_S$ from 256 to $h$ compared to the original (KSA) [18].

### B. $X$, $RM$ and $IV$ Construction

The pseudo-code of the PRGA steps of RC4 is included in Algorithm 2. To generate the required key-stream, first, we iterate (KSA) with $S_2$ as initial state and $K_R$ as a seed to produce the update state, which is used in PRGA to produce the key-stream $R$ that is divided into three parts, as described previously.
In the next section, the proposed LoRCA stream and block ciphers will be described in detail. LoRCA stream and block ciphers use the produced dynamic cryptographic primitives \((S_1, S_2, RM, IV, X)\) for each input message or a set of messages (depending on the configuration).

IV. LoRCA Block and Cipher Schemes

For each input message, LoRCA uses a new dynamic key along with an updated set of cipher primitives \((S_1, S_2, RM, IV, \pi_{RM})\). The common part between the block and stream algorithms is that an input message \(M\) is divided into \(nb\) blocks \(M = m_1, m_2, \ldots, m_{nb}\), where each block has a length of \(h\) bytes, with \(h\) being configured based on the IoV application security and performance requirements. A small \(h\) value is typically used for real-time applications.

Note that the block cipher scheme uses the dynamic Electronic Code Book (ECB) mode of operation without degrading the security level since the same input message (plaintext) produces a different ciphertext due to the different cryptographic primitives used with the two messages.

In the block cipher case, the \(i^{th}\) ciphertext block \(c_i\) is obtained by applying the proposed encryption algorithm \(E_K\) (see Figure 9), which is detailed in Algorithm 3, and described in Eq. 2:

\[
c_i = E_K(m_i) ; \quad i = 1, 2, \ldots, nb
\]

The proposed cipher consists of two sub-functions, Round Function (RF) and UpdateRM Function (URM). RF is iterated once to produce a ciphertext \(c_i\), while RM should be updated for each input block iteration. This is done by calling the UpdateRM Function (URM). Similarly, \(X\) is also updated for each input block iteration by using the same update process.

A. LoRCA Block Cipher Round Function (RF)

The LoRCA block cipher Round Function (RF) produces the \(i^{th}\) ciphertext via the following six steps (Algorithm 1):

1) Iterating the Pseudo-Random Generator (PRG) for one round. Note that the PRNG is iterated recursively, meaning that the output \(X'\) becomes the next input \(X\).

![Diagram](image-url)

**Figure 8:** RC4 stream cipher algorithm

---

**Algorithm 1** KSA algorithm of RC4

Input: \(L_K\) key length; \(K \triangleq \{k_1, k_2, \ldots, k_{L_K}\}\);
\(L_S\) length of state array \(S\), an initial substitution table \(S \triangleq \{S[0], S[1], \ldots, S[L_S - 1]\}\)

Output: The updated substitution table \(S \triangleq \{S[0], S[1], \ldots, S[L_S - 1]\}\)

1: procedure \(S = \text{RC4-KSA}(S, K, L_K, L_S)\)
2: \(j \leftarrow 0\)
3: for \(i \leftarrow 0\) to \(L_S - 1\) do
4: \(j \leftarrow j + S[i] + k[i] \mod L_K\) \(\mod L_S\)
5: \(\text{SWAP}(S[i], S[j]) \triangleright \text{swap values of } S[i] \text{ and } S[j]\)
6: end for
7: return \(S\)
8: end procedure

**Algorithm 2** PRG for RC4

Input: \(L_R\) represents the required number of key-stream bytes,
\(L_S\) represents the length of the state array \(S\);
\(S \triangleq \{s[0], \ldots, s[L_S - 1]\}\)

Output: Key-stream \(R \triangleq \{R[0], R[1], \ldots, R[L_R - 1]\}\) with \(L_R\) bytes length

procedure \(R = \text{PRNGA}(S, L_S)\)

\(i \leftarrow 0\)

\(j \leftarrow 0\)

for \(w \leftarrow 0\) to \(L_R - 1\) do

\(i \leftarrow i + 1 \mod L_S\)

\(j \leftarrow j + S[i] \mod L_S\)

\(\text{SWAP}(S[i], S[j]) \triangleright \text{swap values of } S[i] \text{ and } S[j]\)

\(R[w] \leftarrow S[i] + S[j] \mod L_S\)
end for
return \(R\)
end procedure
2) Mixing the input block $m_i$ with the PRNG output using the logical "XOR" operation.
3) Using two substitution tables, $S_1$ and $S_2$, to substitute the output to produce the $i^{th}$ substituted block $T$. In the proposed solution, $S_1$ is used to substitute the bytes with odd indices, while $S_2$ is used for substituting those with even indices. For example, $S_1$ is used to substitute the bytes with indices 1, 3, 5, … and $S_2$ is used to substitute the bytes with indices 2, 4, 6, ….
4) Updating the pseudo-random vector $RM$.
5) Mixing the updated $RM$ with the substituted output block $T$, using also the logical "XOR".
6) Substituting the output using two substitution tables ($S_2$ and $S_1$) to produce the $i^{th}$ ciphertext, $c_i$. In this step, $S_1$ is used to substitute the bytes with even indices, and $S_2$ is used to substitute the bytes with odd indices.

The encryption steps to produce the $i^{th}$ ciphertext block, $c_i$, are illustrated in Figure 9 and described in detail in Algorithm 3.

**Algorithm 3 LoRCA block cipher variant**

**Input:** Plaintext block $m_i$, substitution tables ($S_1$ and $S_2$), $RM$, $\pi_{RM}$, $X$

**Output:** $i^{th}$ ciphertext block $c_i$

1: **procedure** ENCR($m_i$, $S_1$, $S_2$, $RM$, $\pi_{RM}$, $X$)
2: $RM \leftarrow UpdateRM(RM, \pi_{RM})$
3: $X' \leftarrow XorShift64(X)$
4: $T \leftarrow S(m_i \oplus X', S_1, S_2)$
5: $TR \leftarrow RM \oplus T$
6: $c_i \leftarrow S(m_i \oplus TR, S_2, S_1)$
7: $X \leftarrow X'$
8: **end procedure**

As such, all the plaintext blocks are encrypted to produce $C$, which is the encrypted message that will be either stored locally, or securely communicated with the desired destination.

To recover the original message, the decryption process follows the same steps as the encryption but in the reverse order, and using the inverse substitution tables. Hence, the original block $m_i'$ is recovered by decrypting the $i^{th}$ ciphertext block $c_i$, using the corresponding decryption algorithm $D$, that is described in detail in Algorithm 4 and presented in Eq. 3:

$$m_i' = D(C_i) ; \ i = 1, 2, \ldots, nb$$  \hspace{1cm} (3)

**Algorithm 4** The corresponding decryption algorithm $D$ of the proposed block cipher variant

**Input:** Ciphertext block $c_i$, inverse substitution tables ($S_1^{-1}$ and $S_2^{-1}$), $RM$, $\pi_{RM}$, $X$

**Output:** The $i^{th}$ plaintext block $m_i$

1: **procedure** DECR($c_i$, $S_1^{-1}$, $S_2^{-1}$, $RM$, $\pi_{RM}$, $X$)
2: $RM \leftarrow UpdateRM(RM, \pi_{RM})$
3: $X' \leftarrow XorShift64(X)$
4: $TR \leftarrow S(c_i, S_2^{-1}, S_1^{-1})$
5: $T \leftarrow RM \oplus TR$
6: $O \leftarrow S(T, S_1^{-1}, S_2^{-1})$
7: $m_i \leftarrow O \oplus X'$
8: $X \leftarrow X'$
9: **end procedure**

**B. LoRCA Stream Cipher Round Function (RF)**

To produce the $i^{th}$ key-stream block, the LoRCA stream cipher Round Function (RF) consists of the following six steps:

1) Iterating the selected PRNG once with a seed vector $X$. Similarly to block cipher, the selected PRNG should be iterated recursively, with the output $X'$ becoming the next input $X$. 

![Figure 9: Structure of the proposed LoRCA block cipher](image-url)
The proposed LoRCA stream cipher round function

Algorithm 5: The proposed LoRCA stream cipher round function

**Input:** Plaintext block \( m_i \), Initial Vector (IV), two substitution tables \( (S_1 \text{ and } S_2) \), \( RM \), \( \pi_{RM} \), \( X \)

**Output:** \( i^{th} \) ciphertext block \( c_i \)

1. **procedure** \( UP_{\text{ABPRIM}}(IV, S_1, S_2, RM, \pi_{RM}, X) \)
2. \( RM \leftarrow \text{UpdateRM}(IV, S_1, S_2, RM, \pi_{RM}, X) \)
3. \( X \leftarrow \text{PRNG}(X) \)
4. \( T \leftarrow S(IV \oplus X, S_2, S_1) \)
5. \( R \leftarrow RM \oplus T \)
6. \( c_i \leftarrow m_i \oplus R \)
7. **end procedure**

In a similar manner, all the plaintext blocks are encrypted to produce \( C \), the encrypted message that is either stored locally, or securely communicated with the desired destination.

On the other hand, to recover the original message, the decryption algorithm follows the same steps to generate the same key-stream sequence that will be mixed with the ciphertext. Thus, to recover the \( i^{th} \) original block \( m_i' \), the \( i^{th} \) ciphertext block \( c_i \) should be mixed (exclusive or) with the same produced \( R_i \), as presented in Eq. 5:

\[
m_i' = c_i \oplus R_i; \quad i = 1, 2, \ldots, nb
\]

Next, we describe the URM function to explain the update process of \( RM \).

**C. Update Pseudo-Random Vector Function**

\( RM \) is updated by using a deterministic PRNG, that uses \( RM \) as a seed. For example, we use XorShift64 as PRNG since it requires low computational and resource requirements. This variant is similar to the update \( X \) approach.

To update \( X \), we use the XorShift64 PRNG, which has an output of 64 bits. Therefore, XorShift64 should be iterated for \( \lceil \frac{h}{8} \rceil \) times to get the desired number of bytes \( h \) (length of \( X \)). For example, XorShift64 should be iterated twice or 4 times to get \( h = 16 \) or \( 32 \), respectively. XorShift64 belongs to the Linear-Feedback Shift Registers (LFSR) PRNG functions and is described in Algorithm 6. In modern processor architectures, XorShift64 is extremely fast due to its efficient implementation without the excessive use of sparse polynomials [69]. Due to the fact that XorShift64 generators do not include non-linear steps, they are prone to fail certain statistical tests [69]. However, they still have numerous advantages such as simple implementation and low execution time.

**Algorithm 6:** C code of 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. \( \text{return } x \)
7. **end procedure**
Figure 11: Tests Results of the generated key-stream showing (a) Amplitude variation, (b) recurrence, (c) probability density function, and (d) entropy; for 1,000 random dynamic keys and $h = 16$.

V. SECURITY ANALYSIS

To be secure, the proposed stream and block cipher schemes should resist all types of analytical attacks, including statistical, chosen/known plaintext/ciphertext, brute-force and algebraic attacks [6], [7]. In this section, the immunity of the proposed schemes against these attacks is evaluated and analysed. We consider input messages with "all-zero" bytes.
A. Resistance Against Statistical Analysis

Statistical attacks are typically prevented when the encrypted message exhibits high uniformity and randomness levels, as well as a high periodicity [6], for the stream cipher case.

Several statistical tests including PractRand and TestU01 were carried out on the ciphertext. It is important to state that these tests are among the most difficult randomness tests. The purpose is to validate that the ciphertext achieves the desired randomness and uniformity properties. According to the obtained results, the ciphertext of both schemes successfully passes the TestU01 [70] and "PractRand" [71] tests with all the tested seeds. This confirms that the proposed ciphers can resist statistical attacks.

Moreover, visual results of the ciphertext Probability Density Function (PDF) for a random dynamic key (see Figure 11c), in addition to the entropy of the produced key-stream for 1,000 dynamic keys are presented in Figure 11d. These results show that the produced key-stream follows the required uniform distribution. Furthermore, for the same random dynamic key, the recurrence results of the produced key-stream (see Figure 11a) is presented in Figure 11b), show that the produced key-stream has a uniform random recurrence. Accordingly, the block cipher exhibits similar statistical results in terms of uniformity, recurrence, and sensitivity. To avoid redundancy, only the stream cipher results are included in this paper. Note that more details about the conducted tests can be found in [18], [72].

Figure 12 shows the variation of the correlation coefficient ($\rho$) and the difference percentage between the original and encrypted messages ($Dif$) for 1,000 random dynamic keys. Let us indicate that the red line in these figures represents the ideal value, which is 0 for the correlation coefficient test and 50% for the difference percentage test. The results confirm that there is no detectable relation between the plaintext and ciphertext.

B. Plaintext Sensitivity Test

This test is needed only for the block cipher algorithm to check the difference between the obtained ciphertexts, when two plaintext messages with a slight difference are encrypted with the same key. However, our proposed block cipher relies on the dynamic key approach. Thus, different keys are used for different messages. Hence, different cipher primitives are employed compared to the previous or next messages, and the different dynamic keys will lead to different ciphertexts for the same plaintext. Considering the same plaintext with 1,000 different keys, the difference between the obtained ciphertexts is...
close to 50%, as seen in Figure 13. As such, the proposed LoRCA cipher variants achieve the required plaintext sensitivity (avalanche effect).

C. Key Sensitivity Test

The nonce and key sensitivity tests aim at validating the key avalanche effect. In these tests, we quantify the percentage difference between the produced ciphertexts, for any bit-difference in the nonce or the secret key.
According to the structure of the proposed solution, a one-bit difference in the secret key or nonce would lead to the generation of a new dynamic key. Also, different cipher primitives will be generated and hence, different ciphertexts will be obtained. Figure 13 shows the secret key, nonce and dynamic key sensitivity results of the proposed stream cipher for 1,000 random instances. The results are very close to the ideal value of 50% difference at the bit level. Similar sensitivity results are obtained for the proposed block cipher scheme.

D. High Periodicity

Based on the dynamic key approach, the proposed stream cipher can be considered as a perturbation technique. The generated periodicity of the key-stream is directly related to the PRNG generator and the RM update process. Therefore, using a secure cryptographic PRNG along with the proposed RM update process leads to a high periodicity level, while minimizing the probability of repeated dynamic keys.

VI. CRYPTOANALYSIS

The proposed block and stream ciphers satisfy the confusion and diffusion properties with the lowest possible number of rounds and operations. These ciphers require a single round that consists of simple addition, substitution and permutation operations. The proposed ciphers avoid the use of the diffusion operation to avoid an increase in the latency.

In order to validate the security and robustness of the proposed LoRCA block and stream cipher variants, we discuss and analyze these schemes in the context of different attacks, namely, statistical attacks, differential attacks, brute force attacks and related key attacks.

We assume that the adversary has full knowledge of the protocols used for transmission, the proposed cipher algorithm, and the ciphertexts that are exchanged between the IoT/IIoV and application server(s). The only secret information is the session key.

A. Statistical Attacks

To ensure high resilience against this type of attacks, the cipher scheme should produce highly random and uniform ciphertexts. Based on the security tests presented in the previous section, both of the proposed schemes proved to have a high level of randomness and uniformity via the statistical tests, "Testu01" and "PractRand.

The security tests results confirmed that the encrypted messages are uniformly distributed and have highly dispersed recurrence plots. The schemes also satisfy the independence property since the bit-difference between the input plaintext and the corresponding ciphertext is always close to the desired value of 50%, in addition to a low correlation coefficient as shown in Figure 12.

On the other hand, the statistical results, presented in Table II, and the visual results of the produced ciphertexts validate these conclusions (see Figure 11), and confirm the strong resistance against statistical attacks.

B. Linear and Differential Attacks

In linear and differential attacks, an adversary tries to exploit the relationship between two encrypted messages and their corresponding ciphertexts, to extract the cipher primitives, encryption keys or any useful information. To overcome this issue, we proposed the dynamic update process, which changes the cipher primitives for every input message (or a set of messages). The update process can be based on the re-generation of a dynamic key or in a lightweight manner by simply permuting the cryptographic primitives.

Hence, the produced ciphertexts are uncorrelated, random and independent compared to plaintexts (see Figure 12). This has also been validated using the key sensitivity test for the secret key, nonce and dynamic secret key, as shown in Figure 13. More specifically, the results showed that when the same message is encrypted using two different keys, the resulting ciphertexts have at least 50% bit difference. As a result, no valuable information can be revealed from the encrypted data.
Table II: Statistical results of the proposed stream cipher for 1,000 random keys.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Mean</th>
<th>Max</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secret Key Sensitivity (KS)</td>
<td>49.4977</td>
<td>50.0011</td>
<td>50.5125</td>
<td>0.1413</td>
</tr>
<tr>
<td>Nonce Sensitivity (NS)</td>
<td>49.5797</td>
<td>50.0045</td>
<td>50.4664</td>
<td>0.1356</td>
</tr>
<tr>
<td>DK sensitivity</td>
<td>49.48</td>
<td>50.001</td>
<td>50.43</td>
<td>0.138</td>
</tr>
<tr>
<td>Diff</td>
<td>49.59</td>
<td>50.004</td>
<td>50.43</td>
<td>0.14</td>
</tr>
<tr>
<td>(\rho)</td>
<td>-0.0245</td>
<td>0.00006</td>
<td>0.026</td>
<td>0.0081</td>
</tr>
<tr>
<td>Entropy (H) of ciphertext</td>
<td>7.984</td>
<td>7.988</td>
<td>7.99</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Moreover, the update of cryptographic primitives per input message further complicates the algebraic, linear and differential attacks. Thus makes linear and differential attacks are ineffective in this case.

The proposed schemes are also secure against chosen/known plaintext/ciphertext attacks, which are a sub-set of linear and differential attacks.

C. Brute Force Attacks

In both cipher schemes, the pre-shared secret key can have a length of 128, 196, or 256 bits. Based on the size of the secret key, the size of the nonce is adjusted (padding) since both should have the same size in order to perform the XOR operation. On the other hand, the size of the generated dynamic key is 512 bits since the SHA-512 hash function is utilized. Such key-size is acceptable for preventing brute force attacks.

D. Related Key Attacks and Weak Keys

From the key sensitivity test results, it is evident that the desired key sensitivity percentage (50\%) is achieved (see Figure 13), hence, proposed stream and block cipher schemes are able to resist any related-key attack. In these schemes, a dynamic key is generated from a nonce and a secret key. This key is then, divided into a number of smaller sub-keys, each used to derive a specific cipher primitive. To increase the security of the proposed key generation function, a secret key is combined with the nonce to decrease the probability of successfully deriving the dynamic key and the cipher primitives. The secret key is only known to the legitimate user. Moreover, each cryptographic primitive is generated in a way that any bit change in the dynamic key will lead to a different set of cryptographic primitives.

Acquiring the dynamic key or sub-keys is rather a hard task for illegitimate users, since they should be able to, correctly, estimate the nonce and secret session key. Unlike existing symmetric cipher schemes which consist of static cipher primitives, the proposed solution updates the cipher-primitives, frequently and pseudo-randomly. Therefore, the obtained encrypted messages are independent and uncorrelated. The proposed approach complicates the process of recovering transmitted data. Any vulnerability in any of the dynamic keys will not affect the previously processed or the following messages. Moreover, as seen in Figure 13, the sensitivity of the dynamic key, secret session key and nonce are satisfied, which complicates the key-related attacks and make them unfeasible. On the other hand, the proposed solution is also resilient to weak keys.

The existing analytic attacks are unable to break the cipher scheme with the dynamic key-dependent approach since they are designed to break ciphers with static primitives, which is not the case of the proposed cipher scheme. This makes the proposed cipher extremely hard to break by any analytic or implementation attacks such as side channel attacks since dynamic cryptographic primitives lead to different physical properties. As a conclusion, the proposed block and stream ciphers are immune against the different well-known attacks and possibly future ones.

VII. PERFORMANCE ANALYSIS

In this section, LoRCA’s performance is analyzed to quantify its effectiveness. In the following, we present two essential metrics, the error propagation rate and the encryption/decryption computational delay.

A. Error Propagation Rate

In the stream cipher case, any bit error that occurs in the encrypted block, \(c_i\), will affect only the block’s corresponding bit in the decrypted message. However, in the block cipher case, the block’s corresponding byte will be affected in the decrypted message. As such, the stream cipher variant exhibits a lower error rate compared to the block cipher. Furthermore, the LoRCA block cipher variant has lower error propagation compared to traditional block cipher algorithms such as AES, Simon and speck, since in these algorithms, the plain block avalanche effect is achieved by using the traditional multi-round function structure with diffusion operations. Note that the message avalanche effect is achieved differently in
this work by benefiting from the dynamic key approach. Consequently, LoRCA cipher variants are efficient and can resist better channel errors compared to state-of-the-art ciphers.

B. Encryption/Decryption Computational Delay

The main objective of this work is to achieve a high level of security with minimum computational complexity to reduce the encryption time and the resources overhead, especially in terms of energy consumption. To assess the total computational overhead, the delays associated with the required operations are quantified as follows:

1) $T_S$ stands for the execution time of substituting an $N$-byte block.
2) $T_{xor}$ is the execution time for the XOR operation between two $N$-byte blocks.
3) $T_{PRNG}$ is the time to iterate the selected PRNG.
4) $T_P$ is the time needed to permute a block of $N$ bytes.

As a result, the total Computational Delay ($CD$) to encrypt a single block with the proposed block cipher is:

$$CD = 2 \times T_S + 2 \times T_{xor} + T_{PRNG} + T_P$$

(6)

While, the required ($CD$) to encrypt a single block with the proposed stream cipher is:

$$CD = T_S + 3 \times T_{xor} + T_{PRNG} + T_P$$

(7)

On the other hand, the total computational delay to encrypt a single block using the standard AES in [22] is:

$$CD_{AES} = rT_S + (r + 1)T_{xor} + (r - 1)T_D + rT_{SR}$$

(8)

Where $T_D$ represents the AES Mix-column operation time, presenting the highest delay among the other AES operations. Moreover, $T_{SR}$ represents the AES “Shiftrows” operation delay, and $r$ represents the number of rounds. For a 128-bit secret key, the minimum value of $r$ is 10. As a result, the minimum AES computational delay is given by:

$$CD_{AES(r=10)} = 10T_S + 11T_{xor} + 9T_D + 10T_{SR}$$

(9)

This clearly shows that the AES computational time is much higher compared to the proposed block and stream ciphers where we intentionally avoided any diffusion operation. Moreover, for AES with 192 and 256-bit length secret key, $r$ is equal to 12 and 14, respectively. This requires even higher execution times compared to the 128-bit secret key case. Note that the proposed stream cipher requires a lower computational complexity compared to the proposed block cipher.

On the other hand, the delay associated with the derivation function of the key and cryptographic primitives ($CD_{KDF}$) is quantified below and can be described as follows:

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

$$CD_{KDF} = T_{xor} + T_H + 2T_{KSA} + T_{MKSA}(h) + T_{PRNG}$$

(10)

It can be noticed that the key derivation function introduces computational overhead and thus, a different key derivation strategy should be adopted for low data rate applications communicating small-sized messages. For low data rate applications, the dynamic key and cipher primitives should be updated after $\delta$ small-sized messages, or after $\tau$ data bytes. Thus, until reaching this threshold, all cipher primitives remain constant, except for the two substitution tables, $S_1$ and $S_2$, which will be updated for each message, as follows:

$$Temp = S_1$$
$$S_1 = S_1(S_2)$$
$$S_2 = S_2(Temp)$$

(11)

As a result, decreasing $\tau$ can increase the security level along with the required resources and delays and vice-versa. In fact, $\delta$’s configuration depends on the application requirements and the needed security level.

C. Proposed Ciphers Throughput

In this section, the average throughput of the proposed block and stream ciphers is quantified by conducting several experiments (1,000 times) on a real hardware platform (“Rapsberry Pi”), which can be used in IoV applications. Also, several Raspberry device classes are considered in these experiments, namely, RPI0, RPI3, and RPI4. In the following, only the encryption throughput is analysed since the decryption throughput is very close to the encryption one; this is due to the fact that the decryption algorithm for both ciphers requires the same number of operations as the encryption algorithm.

Figure 15 illustrates the variation of the throughput (MB/s) as a function of $h$ for the proposed block and stream ciphers. We can see that the value of $h$ has a
visible and different impact on the encryption performance on different Raspberry Pis (RPI) devices; the best performance, for the RPI0 and RPI3 for both LoRCA ciphers is achieved for \( h = 32 \) and 8, respectively. While in the case of RPI4, the best performance for the stream cipher variant is achieved for \( h = 64 \) and 8 for the block cipher variant. As such, \( h \) can be fixed to the optimal value that is obtained from the performance analyzing of LoRCA for the device to be used, as shown in Figure 15.

The throughput of the proposed LoRCA cipher schemes is compared to that of AES (OpenSSL implementation). Note that the proposed LoRCA ciphers are implemented using the C programming language, whereas the AES OpenSSL implementation code is well optimized in assembly language. Figure 16 compares the throughput of the proposed ciphers and that of AES OpenSSL with CTR and CBC operation modes. The results indicate that LoRCA always outperforms AES on RPIs. Numerical results are given in Tables III and IV for the throughput ratio between the proposed ciphers as compared to AES OpenSSL. The best performance of LoRCA over AES is obtained with RPI0. However, the performance is slightly reduced with RPI3 and RPI4 since the OpenSSL implementation is optimized in assembly. In summary, considering RPI0, the encryption/decryption throughput of the proposed LoRCA cipher variants is higher compared to AES with or without optimization as indicated in Table III and IV. A significant gain can be achieved by optimizing LoRCA cipher variants using assembly as in AES. This will further improve the throughput for RPI3 and RPI4.

Moreover, the proposed LoRCA variants are compared to state-of-the-art IoT cipher algorithms such as Simon and Speck. Table V presents the throughput results of the proposed cipher variants against (AES, Simon, and Speck) for different Raspberry Pi classes, and Table VI and Table VII show the ratio of the throughput results. Based on these results, Speck seems to require more execution time compared to Simon. According to Table VI, Simon and Speck require at least, 4.87 and 1.67 times, respectively, the overhead of the proposed LoRCA block cipher. Also, according to Table VII, Simon and Speck require at least 2.71 and 1.18 times, respectively, the overhead of the proposed stream cipher. Moreover, the proposed block cipher throughput outperforms, by a factor of 24.113%, 60.05% and 65.38%, the previous one of [18] with RPI0, RPI3, and RPI4, respectively. According to Table VI and VII, both variants present a higher throughput compared to AES, SPECK, SIMON [29] and [18].

This test was also applied on tiny devices that use an ARM CPU such as teensy3.6 (ARM Cortex M4), teensy 4.0 (ARM Cortex-M7) in addition to ESP 32. These devices were selected since they have a cryptographic acceleration unit (AES-NI). The obtained results in Figure 17 demonstrate the efficiency and suitability of the proposed stream cipher solution with this type of architecture when compared to AES.

According to Figure 18, AES with NI instructions require at least 28.5% and 38.5% additional overhead time, respectively, compared to the proposed LoRCA ciphers. Similarly, LoRCA exhibits lower execution time between 55% and 64% as compared to Teensy 3.6, while the time overhead reduction varies between 32.3% and 41.9% against ESP 32.

D. Memory Consumption

To encrypt one block in both proposed ciphers, we need the input block and two substitution tables \((S1 \text{ and } S2)\), each with 256 bytes, in addition to two blocks \((X \text{ and } RM)\) of \( h \) bytes. Therefore, the required memory consumption of both ciphers is \((512 + 3 \times h)\) bytes. For limited memory devices, we can use only one substitution table instead of two to reduce the required memory consumption to \(256 + 3 \times h\). Compared to AES, one substitution table (256 bytes) is required in addition to the input key and round key (32). Note that \( h \) can be increased, if the devices possess more memory capacity.

 VIII. CONCLUSION & FUTURE WORK

In this paper, we proposed LoRCA, an efficient lightweight block and stream cipher schemes for tiny IoT devices. LoRCA relies on the dynamic key-dependent approach to strike the right balance between the security level and the devices’ performance. Two different lightweight round functions were proposed, one for block cipher and a simpler one for stream cipher. Each round function needs to be applied for a single iteration, which drastically reduces the computational and resources overhead when compared to standard ciphers such as AES, which requires a higher number of rounds, \( r \). Finally, security and performance analysis were performed to prove LoRCA’s effectiveness and robustness. We believe that such work opens a new way for the definition and design of modern lightweight cipher schemes.

COMPLIANCE WITH ETHICAL STANDARDS

• Funding: This research was partially supported by funds from the Maroun Semaan Faculty of Engineering and Architecture at the American University of Beirut and by the EIPHI Graduate School (contract "ANR-17-EURE-0002").
Figure 15: Throughput variation (MB/s), with a colored Lena image of size $(512 \times 512 \times 3)$, as a function of $h$ on: (a) RPI0, (b) RPI3, and (c) RPI4.

Table III: Speedup results of LoRCA block cipher over AES

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Protocol comparison</th>
<th>16B</th>
<th>64B</th>
<th>512B</th>
<th>1KB</th>
<th>4KB</th>
<th>16KB</th>
<th>64KB</th>
<th>256KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPI0</td>
<td>LoRCA BC vs AES CBC</td>
<td>23.12</td>
<td>8.20</td>
<td>4.34</td>
<td>3.64</td>
<td>3.40</td>
<td>3.57</td>
<td>3.72</td>
<td>3.70</td>
</tr>
<tr>
<td></td>
<td>LoRCA BC vs AES CTR</td>
<td>16.51</td>
<td>7.77</td>
<td>4.44</td>
<td>3.91</td>
<td>3.69</td>
<td>3.90</td>
<td>3.67</td>
<td>3.68</td>
</tr>
<tr>
<td>RPI3</td>
<td>LoRCA BC vs AES CBC</td>
<td>20.72</td>
<td>9.96</td>
<td>3.46</td>
<td>3.10</td>
<td>2.89</td>
<td>2.61</td>
<td>3.0</td>
<td>2.84</td>
</tr>
<tr>
<td></td>
<td>LoRCA BC vs AES CTR</td>
<td>15.40</td>
<td>8.35</td>
<td>3.14</td>
<td>2.54</td>
<td>2.31</td>
<td>1.99</td>
<td>2.24</td>
<td>2.29</td>
</tr>
<tr>
<td>RPI4</td>
<td>LoRCA BC vs AES CBC</td>
<td>19.02</td>
<td>7.05</td>
<td>3.94</td>
<td>3.99</td>
<td>3.75</td>
<td>3.67</td>
<td>3.67</td>
<td>3.69</td>
</tr>
<tr>
<td></td>
<td>LoRCA BC vs AES CTR</td>
<td>14.70</td>
<td>5.68</td>
<td>3.34</td>
<td>3.21</td>
<td>3.00</td>
<td>2.89</td>
<td>2.89</td>
<td>2.88</td>
</tr>
</tbody>
</table>

- **Conflict of interest:** The authors declare that they have no conflict of interest.
Figure 16: Comparison of the encryption throughput of the proposed block and stream ciphers (LoRCA) and AES OpenSSL (in CTR and CBC mode) on (a) RPI0, (b) RPI3, and (c) RPI4, as a function of the buffer size (in bytes) for $h=16$.

### Table IV: Speedup results of LoRCA stream cipher over AES

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Protocol comparison</th>
<th>Message Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>16B</td>
</tr>
<tr>
<td>RPI0</td>
<td>LoRCA SC vs AES CBC</td>
<td>16.27</td>
</tr>
<tr>
<td></td>
<td>LoRCA SC vs AES CTR</td>
<td>11.62</td>
</tr>
<tr>
<td>RPI3</td>
<td>LoRCA SC vs AES CBC</td>
<td>15.81</td>
</tr>
<tr>
<td></td>
<td>LoRCA SC vs AES CTR</td>
<td>11.74</td>
</tr>
<tr>
<td>RPI4</td>
<td>LoRCA SC vs AES CBC</td>
<td>13.14</td>
</tr>
<tr>
<td></td>
<td>LoRCA SC vs AES CTR</td>
<td>9.84</td>
</tr>
</tbody>
</table>

21
Figure 17: Variation of the encryption throughput of the proposed block and stream ciphers (LoRCA) and AES OpenSSL (in CTR and CBC mode) on ESP32 (a), teensy3.6 (b), and teensy 4.0 (c) in function of the buffer size (in bytes or kilobytes) for $h=16$.

Table V: Encryption throughput in MB/s of SPECK and SIMON ciphers [29], one round of [18] and the proposed ciphers on different Raspberry Pi devices with a colored Lena image of size $(512 \times 512 \times 3)$

<table>
<thead>
<tr>
<th>Cipher(Key size, block size)</th>
<th>Raspberry Pi0</th>
<th>Raspberry Pi3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speck(256, 128)</td>
<td>14.60</td>
<td>28.12</td>
</tr>
<tr>
<td>Speck(64, 32)</td>
<td>5.32</td>
<td>9.25</td>
</tr>
<tr>
<td>Simon(256, 128)</td>
<td>4.99</td>
<td>9.98</td>
</tr>
<tr>
<td>Simon(64, 32)</td>
<td>4.09</td>
<td>7.40</td>
</tr>
<tr>
<td>One round [18] (block cipher)</td>
<td>19.62</td>
<td>43.16</td>
</tr>
<tr>
<td>Proposed One (Stream Cipher)</td>
<td>28.98</td>
<td>85.14</td>
</tr>
</tbody>
</table>
Figure 18: Variation of the encryption reduction ratio, as a function of $h$, of LoRCA and AES OpenSSL in CTR mode on ESP32, teensy3.6, and teensy4.

Table VI: Throughput ratio of the proposed block cipher and those of Speck and Simon.

<table>
<thead>
<tr>
<th>Cipher Size (Key, Block)</th>
<th>Raspberry Pi0</th>
<th>Raspberry Pi3</th>
<th>Raspberry Pi4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speck(256, 128)</td>
<td>1.26</td>
<td>2.69</td>
<td>5.78</td>
</tr>
<tr>
<td>Speck(64, 32)</td>
<td>3.76</td>
<td>8.01</td>
<td>17.198</td>
</tr>
<tr>
<td>Simon(256, 128)</td>
<td>3.13</td>
<td>6.67</td>
<td>14.33</td>
</tr>
<tr>
<td>Simon(64, 32)</td>
<td>5.17</td>
<td>11.02</td>
<td>23.64</td>
</tr>
</tbody>
</table>

Table VII: Throughput ratio of the proposed stream cipher and those of Speck and Simon.

<table>
<thead>
<tr>
<th>Cipher (Key size, block size)</th>
<th>Raspberry Pi0</th>
<th>Raspberry Pi3</th>
<th>Raspberry Pi4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speck(256, 128)</td>
<td>1.18</td>
<td>2.88</td>
<td>4.5</td>
</tr>
<tr>
<td>Speck(64, 32)</td>
<td>3.25</td>
<td>8.58</td>
<td>13.39</td>
</tr>
<tr>
<td>Simon(256, 128)</td>
<td>2.71</td>
<td>7.15</td>
<td>11.16</td>
</tr>
<tr>
<td>Simon(64, 32)</td>
<td>4.47</td>
<td>11.81</td>
<td>18.41</td>
</tr>
</tbody>
</table>

- Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

REFERENCES


