Lightweight Stream Cipher Scheme for Resource-Constrained IoT Devices

Hassan Noura, Raphael Couturier, Congduc Pham, Ali Chehab

Lightweight Stream Cipher Scheme for Resource-Constrained IoT Devices

Hassan Noura
Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon

Raphaël Couturier
FEMTO-ST Institute,
Univ. Bourgogne Franche-Comté,
Belfort, France

Congduc Pham
University of Pau, LIUPPA laboratory
Pau, France

Ali Chehab
Electrical and Computer Engineering
American University of Beirut (AUB)
Beirut, Lebanon

***Abstract*—The Internet of Things (IoT) systems are vulnerable to many security threats that may have drastic impacts. Existing cryptographic solutions do not cater for the limitations of resource-constrained IoT devices, nor for real-time requirements of some IoT applications. Therefore, it is essential to design new efficient cipher schemes with low overhead in terms of delay and resource requirements. In this paper, we propose a lightweight stream cipher scheme, which is based, on one hand, on the dynamic key-dependent approach to achieve a high security level, and on the other hand, the scheme involves few simple operations to minimize the overhead. In our approach, cryptographic primitives change in a dynamic lightweight manner for each input block. Security and performance study as well as experimentation are performed to validate that the proposed cipher achieves a high level of efficiency and robustness, making it suitable for resource-constrained IoT devices.**

***Index Terms*—Lightweight cryptography; key-dependent encryption, security, IoT.**

I. Introduction

The Internet of Things (IoT) systems introduced new smart applications such as smart houses/ buildings/ cities, environment monitoring, traffic monitoring, and health monitoring, among others. For most of IoT applications, the devices are resource-constrained and are used to monitor and to collect data from the physical environment.

IoT systems are constantly facing dangerous security and privacy threats. The different types of threats target various security services such as confidentiality (data confidentiality and privacy), integrity (device system integrity) and authentication (device/user and data origin authentication), as well as availability (data and system). Therefore, in order to ensure the appropriate security measures, two types of solutions are considered, cryptographic and non-cryptographic. In general, data confidentiality, data integrity, and data origin authentication are ensured by cryptographic algorithms. On the other hand, user/device authentication can be ensured by using a cryptographic protocol that can be based on cryptographic algorithms such as an encryption algorithm or a hash function. When IoT applications communicate sensitive information, confidentiality may be simply breached via eavesdropping and traffic analysis. The eavesdropper will be able to extract the message contents, while the traffic analysis is able to recover useful information (privacy issues) from the traffic such as source and destination from the header of the communicated messages.

A. Problem Formulation

Based on the characteristics of IoT devices and applications, the existing security solutions are not suitable for delay-sensitive applications, nor for tiny devices that have a limited battery lifetime and limited computational power. Moreover, different IoT applications have stringent QoS requirements. As such, there is a critical need for new security solutions that are compatible with the limitations and requirements of IoT devices and applications.

B. Related Work

Traditional cryptographic algorithms such as the Advanced Encryption Standard (AES) [1] require several iterations over a round function, which introduces relatively a large overhead in terms of latency and required resources. The minimum required number of rounds for a traditional block cipher is 4 as it is for the Hummingbird2 cipher [2]. Therefore, such cryptographic

algorithms would result into a poor performance in the context of IoT networks. Recently, several lightweight ciphers such as Simon and Speck [3] have been proposed and they require less computation and resources compared to AES. Speck has a lower overhead compared to Simon and it was shown to be suitable for tiny devices. However, Speck still uses the multi-round structure, although the round function is simple and optimized.

On the other hand, the chaotic cryptographic algorithms also suffer from different limitations such as floating-point computations and conversion operations, finite periodicity and complex hardware implementation [4]. Also, they are based on the multi-round structure. Accordingly, a new paradigm emerged for cryptographic algorithms, which are referred to as "lightweight" since they exhibit low latency and overhead [5], [6]. Lightweight cryptographic algorithms that are based on the dynamic key approach have been proposed in [2], [7]–[9]. The cipher schemes described in [7]–[9] require two iterations of a round function, while [2] requires a single iteration of a round function, and it processes 2 blocks at a time, which makes it faster than the one in [7]–[9]. These solutions use the dynamic key-dependent approach to reduce the required computations and resources while preserving a high security level.

C. Motivation and Contributions

This paper focuses on the design of a new efficient cipher scheme for IoT devices; it requires a single iteration and it provides a better performance and security level compared to the previous dynamic key-dependent ciphers and recent static lightweight ciphers [3].

The proposed solution follows the same logic and results in a flexible, simple lightweight stream cipher scheme (LSC) with 2 simple functions, a round function and an update function that are iterated only once to produce a key-stream. These functions are designed with the minimum possible number of operations to preserve the desirable cryptographic performance. To accomplish this objective, a new dynamic key is generated for each input message, which can be an audio, an image or even a video message. The dynamic key is produced as a function of a secret key and a nonce, which makes the cryptographic primitives non-static and unknown to attackers and hence, introducing a higher complexity for such attackers. The substitution and permutation tables are dynamic and key-dependent, and they are based on the methods proposed in [2]. They are respectively based on the Key Setup Algorithm (KSA) and modified KSA of RC4. These techniques have been validated to ensure a good cryptographic performance in a dynamic manner according to [2]. Having said that, the novelty of this work stems from the encryption algorithm and how it makes use of the dynamic key and cryptographic primitives. The advantages of LSC compared to [2] are related to the excellent balance between the security level and performance for IoT devices:

1) Minimum effect of error propagation: LSC encrypts 1 block at a time instead of 2 blocks to reduce the effect of error propagation
2) Low overhead: LSC requires fewer operations and does not apply the block permutation operation to reduce delay and memory consumption. LSC also avoids chaining and diffusion operations to further reduce the computational complexity.
3) Simpler implementation: [2] cannot be applied to resource-constrained tiny devices, such as Arduino boards, due to the need for a large memory capacity.
4) Variable cipher primitives: LSC updates the cryptographic primitives after each encrypted/decrypted block to provide a higher security level.

D. Organization of the paper

The rest of the paper is organized as follows. The proposed key derivation algorithm along with the proposed cipher construction primitives are described in Section II. Section III presents the proposed lightweight stream cipher (LSC) scheme. Then, extensive security analysis is performed in Section IV to prove the robustness of the scheme. Section V investigates the immunity of LSC against different kinds of existing attacks. The effectiveness of LSC is then validated in Sections VI and VII. Conclusions are derived in Section VIII.

II. Proposed Key Derivation Function

In this section, the proposed key derivation function is described. All the notations used are shown in Table I. Figure 1 shows all the steps of the proposed dynamic key generation technique, where the input is a shared secret session key (SK) between two legal entities. This session key can be renewed after each new session, depending on the IoT application. Key management among IoT devices are beyond the scope of this paper and readers can refer to [10] for more details about possible key management approaches in IoT systems.

A dynamic key (DK) is produced for each new input message by hashing the secret key SK with a nonce that can be produced in a synchronous manner between both entities. This procedure allows any secure cryptographic hash function to be used at this step. In this paper, SHA-512 [11] is used and the output dynamic key is 64 bytes long: $DK = hash_{SHA-512}(SK \oplus nonce)$. The produced dynamic key is therefore different for

Table I: Table of notations

Symbol	Definition
SK	A shared secret Session Key
$nonce$	A dynamic nonce which can be changed for each input message
DK	A Dynamic Key that is updated for each input message
k_{S1} and k_{S2}	First and second substitution sub-Keys
S_1 and S_2	First and second dynamic substitution tables
π	Dynamic permutation table
k_{RM}	Seed for a stream cipher to produce RM and IM
RM and IM	Two pseudo-random blocks
k_{PRM}	A permutation sub-Key and it is used to produce the permutation table π_{RM}
len	length of input message after reshaped to a table form.
nb	Number of blocks in one input message and it is equals to $\rceil \frac{len}{h} \lceil$
h	Number of bytes in one block message
M	The original message
m_i	The i^{th} original plain block
C	The encrypted message
c_i	The i^{th} encrypted block

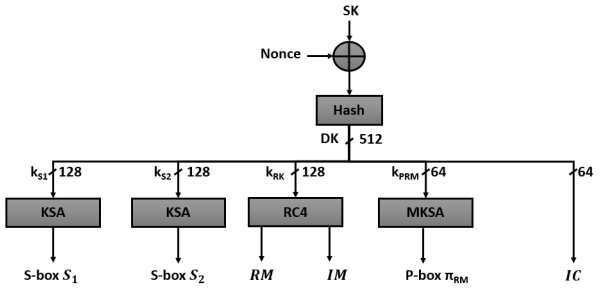


Figure 1: Proposed dynamic key derivation function and construction cipher primitives

each input message and the secure cryptographic hash function ensures a high resistance against collision. In our approach, the dynamic key is divided into four sub-keys: $DK = \{k_{RM}, k_{PRM}, k_{S1}, k_{S2}\}$. Each sub-key has a length of 128 bits (16 bytes). These sub-keys will be employed for different purposes:

- **Pseudo-Random Key** k_{RM} consists of the first most significant 16 bytes and is used to construct a pseudo-random vector RM and an initial vector IM. Both have a length equal to h^2 bytes. These two matrices can be generated by using any stream cipher scheme. In this paper, we use RC4 [12] with k_{RM} as a seed to produce $2 \times h^2$ bytes key-stream. The first h^2 bytes and the next h^2 bytes are reshaped to respectively form the RM and IM vector.
- **Permutation sub-key** k_{PRM} consists of the next most significant 16 bytes of DK and is used to construct a flexible permutation table π_{RM} of length h^2 by using the modified key setup algorithm of RC4 which was presented in [2]. The values of the elements in the permutation table π_{RM} range from 1 to h^2.
- **Substitution sub-key** k_{S1} consists of the next 16 bytes of DK and is used to construct the first substitution table S_1 by using the key setup algorithm of RC4 as described in [2]. The substitution operation is done at the byte level and the elements in table S_1 have values between 0 and 255.
- **Substitution sub-key** k_{S2} consists of the next 16 bytes of DK and is used to construct the second substitution table S_2 similar to S_1.

By construction, all cipher primitives are related to any bit of difference in the secret key or nonce, and will provide a different dynamic key. Therefore, LSC ensures high key sensitivity since all cipher primitives are related to the dynamic key.

III. Lightweight Stream Cipher Scheme (LSC)

LSC is based on the dynamic key dependence approach which means that a different dynamic key is used for each input message, increasing randomness of ciphertext and making cryptanalysis approaches more difficult to be applied. Therefore, for each input message the various cipher primitives are updated to encrypt the next message. The encryption and decryption algorithms will be described in the next paragraphs.

A. Encryption algorithm

The 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. h can be configured according to the IoT applications. A smaller value of h is preferable for real-time applications.

LSC produces a new keystream block for each iteration as the different stream cipher primitives are changed for each input message. The i^{th} ciphertext block $c_i = m_i \oplus R_i$ of each message is obtained by mixing the i^{th} keystream block R_i with the i^{th} plain block m_i and $i = \{1\ ,\ 2,\ \ldots, nb\}$.

To recover the original i^{th} block we compute $m'_i = c_i \oplus R_i$ where the i^{th} ciphertext block c_i is "XORed" with the same R_i.

As explained previously, LSC is divided into two sub-functions: RoundFunction (RF) and $\mathsf{Update-RM-vectorFunction}$ (URM). RF is iterated to produce a required key-stream block R_i. However, this requires that RM is updated which can be achieved by calling the URM.

RoundFunction *(RF)*

RF produces the i^{th} keystream block by applying the following five steps:

1) Update the pseudo-random vector RM as described below.

2) Iterate the selected Pseudo-Random Generator (PRG) for only once. Any Pseudo-random number generator (PRNG) can be used at this step. In this paper, a XorShift64 PRNG is used to produce h bytes for each iteration. The output of XorShift64 is a 64 bits word so XorShift64 should be iterated for $\lceil \frac{h}{8} \rceil$. For example, for h=16 or 32, XorShift64 will be respectively iterated for 2 and 4 times. The PRNG is iterated in a recursive manner where the output IC becomes the next input.
3) Mixing the updated RM with the PRNG output and the initial vector M through the use of XOR.
4) Substitute the output by using the two substitution tables (S_1 and S_2) to produce the i^{th} keystream R_i. In this step, the proposed substitution technique uses the second substitution table S_2 to substitute the bytes with odd indexes and the first substitution table S_1 to substitute the bytes with even indexes.
5) Initial vector IM is updated and becomes equal to R_i.

These steps to produce the i^{th} keystream block R_i are illustrated in Figure 2 and described in Eq. 1.

$$\begin{aligned} IC &= XorShift64(IC) \\ RM &= updateRM(RM, S_1, S_2, \pi_{RM}) \\ R_i &= S(IV \oplus RM \oplus IC) \\ IM &= R_i \end{aligned} \tag{1}$$

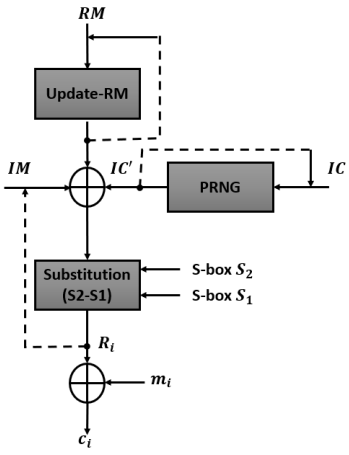


Figure 2: LSC architecture

All plain blocks will be encrypted to form the encrypted message C, which will be securely sent to the desired destination, or to be safely stored locally. The URM function is now presented to explain how RM is updated.

Update − RM − vectorFunction *(URM)*

RM is updated first before being permuted by using the permutation table π_{RM}. Then, the output will be substituted by using the first table S_1 to substitute the bytes with even indexes and the second table S_2 to substitute the bytes with odd indexes.

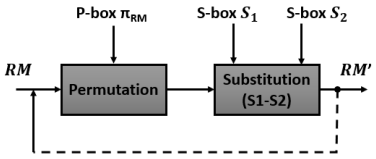


Figure 3: Updating RM for each iteration

Xorshift *PRNG*

Xorshift is a PRNG class that represents linear-feedback shift registers (LFSRs) such as the one described in Algorithm 1. In addition, Xorshift allows an efficient implementation without the need of excessively using sparse polynomials. This makes them extremely fast on any modern computer architecture. Similar to LFSRs, the parameters must be chosen with extreme cautiousness in order to achieve a long period [13]. However, Xorshift generators do not have non-linear steps which can make some statistical tests to fail [13]. Otherwise, Xorshift generators do have numerous advantages including a lower execution time with a very simple implementation.

Algorithm 1 Xorshift64 code

```
__device__ inline
ulong xorshift64(ulong t)
{
  ulong x = t;
  x ^= x >> 12;
  x ^= x << 25;
  x ^= x >> 27;
  return x;
}
```

B. Decryption algorithm

The decryption algorithm uses the same steps to produce the same key-stream sequence and recover the original message by mixing the key-stream with the cipher-text.

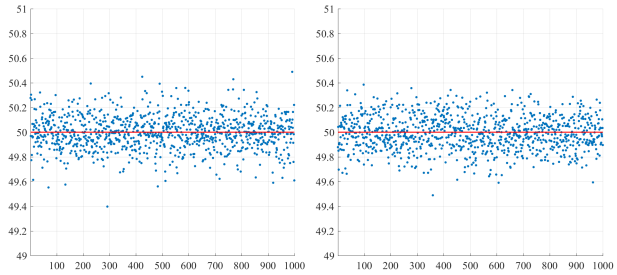
IV. Security Analysis

The proposed cipher scheme should resist different kinds of analytic attacks such as statistical and algebraic

attacks, as well as brute-force attacks [14], [15]. We use the security tests already applied in [2] to validate the cryptographic level and consequently the immunity of LSC against crypatanalysis attacks. We consider input messages filled with zeros.

A. Resistance against statistical analysis

Statistical attacks can be prevented if the encrypted message reaches a high randomness and uniformity level in addition to a high periodicity [14]. Several hard statistical tests were carried out (`TestU01` [16] and `practrand` [17]) on the produced keystream to validate that it reaches the required uniformity and randomness properties. The most difficult scenario where we use the same message with constant values is tested and we found that the produced keystream successfully passed all the tests of `TestU01` and `practrand` with more than 100 different seed values. It is worth mentioning that these statistical tests are the hardest ones. We also found that the produced keystream reaches the required uniformity and randomness levels. These results are summarized in Figure 4-a-b-c to show the probability density function and the uniform distribution of the produced keystream. More details on these tests can be found in [2], [18].



(a) KS (b) NS

Figure 5: Key (a) and nonce (b) sensitivity against 1,000 random keys.

B. Key Sensitivity Test

The sensitivity test is used to validate the key avalanche effect by quantifying the difference (percentage) between the produced keystream for a given difference in the secret key or nonce. If one bit differs in the secret key or nonce, this will produce a new dynamic key, and consequently different cipher primitives and different keystreams. The desired value is 50% difference at the bit level. Figure 5 shows the key and nonce sensitivity for 1,000 random runs. We can see that the difference between produced keystreams for both sensitivity tests is very close to the desired value.

C. High periodicity

LSC is based on the dynamic key dependence approach and can be considered as a perturbation technique since different cipher primitives are updated. Moreover, as the nonce has a long periodicity in addition to updating initial and random matrices in a recursive manner LSC exhibits high periodicity.

V. Discussions on analytic and brute force attacks

We discuss in this section on how LSC can resist to some well-known attacks. First, LSC proposes a new way of designing stream cipher and to reach the desired cryptographic properties such as confusion and diffusion. We found above that LSC reaches a high level of randomness and uniformity according to hard statistical tests such as `TestU01` and `practrand`. In addition, the independence among produced keystreams is ensured.

Then, as both dynamic key's sensitivity and nonce's sensitivity are achieved as shown in Figure 5, this makes key-related attacks much more difficult to succeed. Furthermore, as the dynamic key changes for each input message, algebraic, linear and differential attacks will also become very hard to succeed. Each collected message is encrypted differently with a different dynamic key, and consequently with different cipher primitives making LSC harder to break. All analytic attacks will be unable to break LSC since they are designed to break static ciphers with static cipher primitives.

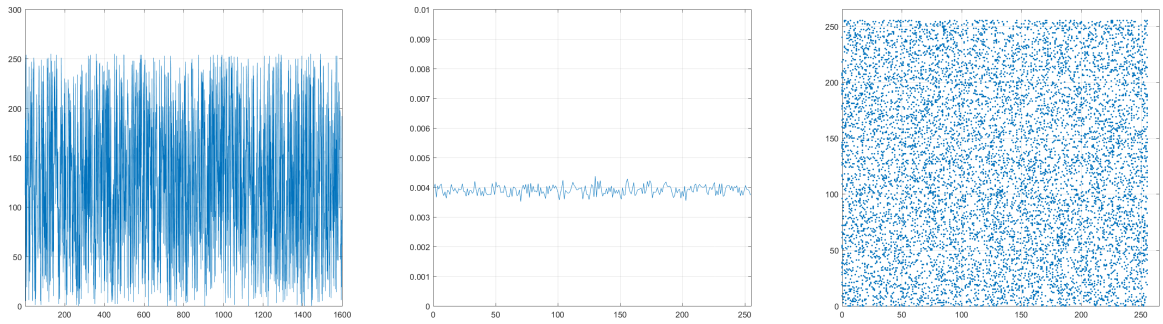
Finally, the size of the secret key can be set to 128, 196, or 256 bits, whereas the size of the nonce and dynamic key is 512 bits. These sizes are large enough to make brute force attacks unfeasible.

VI. Performance Analysis

In this section, we analyze the performance of the proposed cipher scheme towards quantifying its effectiveness. Two important metrics are presented in details, which are the effect of error propagation and the associated encryption/decryption time.

A. Effect of error propagation

The effect of any bit error in the encrypted block c_i will only affect its corresponding bits in the decrypted block.



(a) (b) (c)

Figure 4: Amplitude variation of the produced keystream (a) in addition to its corresponding probability density function (b), recurrence for a random key and for $h = 16$.

B. Encryption/Decryption time

The main objective of the proposed cipher approach is to reach a high level of security with the minimum number of operations. This requires reducing the computational complexity, encryption/decryption time and resources (especially energy) for the data confidentiality process. The execution time of the proposed cipher with and without chaining operation mode, is presented and quantified. To assess the total associated overheads, we quantify several delays as follows:

1) T_S denotes the required substitution execution time for a block of N bytes.
2) T_{xor} denotes the required "XOR" execution time between two blocks of N bytes.
3) T_{PRNG} denotes the required time to iterate the employed PRNG.
4) T_P denotes the required time to permute a block of bytes.

Therefore, the total Computational Delay (CD) of the proposed scheme to encrypt one block is:

$$CD = 3 \times T_S + 2 \times T_{xor} + T_{PRNG} + T_P \quad (2)$$

while the total computation delay of the standard AES described in [1] to encrypt one block is:

$$CD_{AES} = rT_S + (r+1)T_{xor} + (r-1)T_D + rT_{SR} \quad (3)$$

where T_D represents the required delay for the AES "mix-column" operations (for all 4 columns), which has a very high delay compared to other AES operations. T_{SR} represents the required delay for the AES "shift-rows" operations, and r represents the number of rounds. The minimum value of r is 10 for 128 bits secret key and the minimum AES computation delay is given by:

$$CD_{AES(r=10)} = 10T_S + 11T_{xor} + 9T_D + 10T_{SR} \quad (4)$$

Consequently, the AES computation time is larger compared to our proposed solution with or without relying on the chaining operation mode. In addition, our proposed solution avoids any diffusion operation such as the "mix-column" operations of AES in order to reduce the delay: the delay of the XOR and substitution operations are far less than that of the "mix-column" diffusion of AES. Accordingly, our proposed scheme requires a lesser computational complexity compared to the AES standard cipher with 128 bits length secret key. For 192-bit and 256-bit secret keys, r are equal to 12 and 14 respectively which requires much more execution time compared to the 128-bit secret key.

VII. Experimentations

In this section we present results from additional experimentations conducted on real hardware platforms used in many IoT deployments: low-cost 8-bit AVR ATmega328P MCU at 8MHz (which is used on the well-known Arduino ProMini board and many other similar boards) and a 32-bit Cortex-M4 (MK20DX256VLH7) ARM MCU at 48MHz (which is used on the Teensy32 board for instance). We implemented LSC and integrated it into our LoRa IoT framework [19]. In all the tests, we compared the encryption time of LSC with an efficient implementation of 128-bit AES for resource-constrained devices [20] and Speck. We varied the message size from 16 bytes to 240 bytes.

Figure 6 compares the encryption time of our LSC algorithm with AES and Speck. On the ProMini, LSC outperforms both AES and Speck. On the Teensy32, LSC

also outperforms AES but is slightly slower than Speck, probably due to higher optimization of bit rotation operations on Cortex architecture. In Figure 6(bottom), the data tags from 10us to 115us are for LSC while the tags from 8us to 109us are for Speck. Figure 7 shows the corresponding encryption time ratio of AES and Speck compared to LSC.

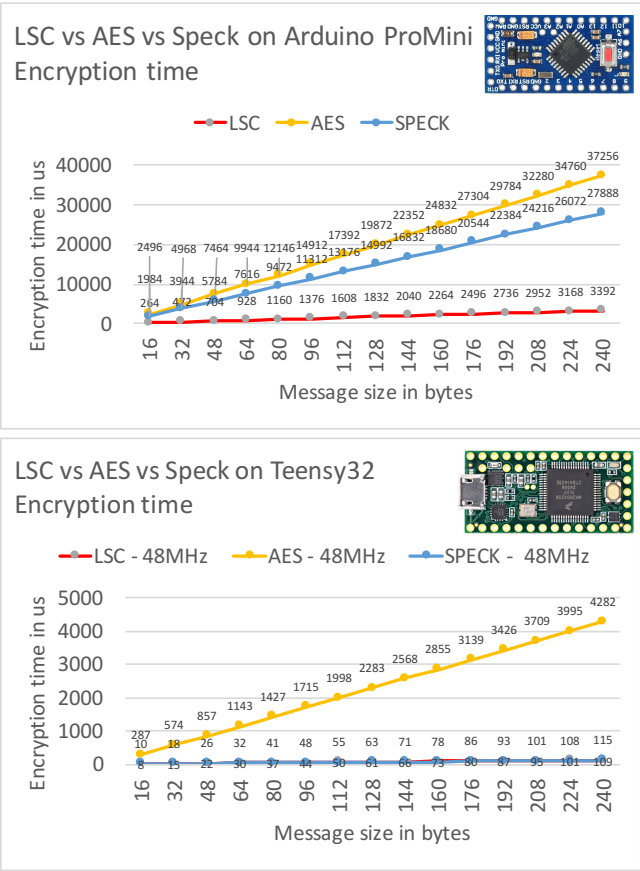


Figure 6: Encryption time

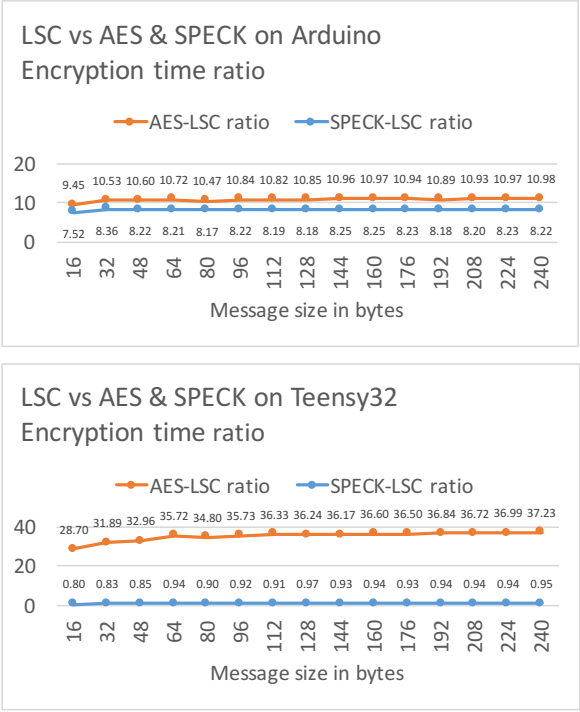


Figure 7: Encryption time ratio

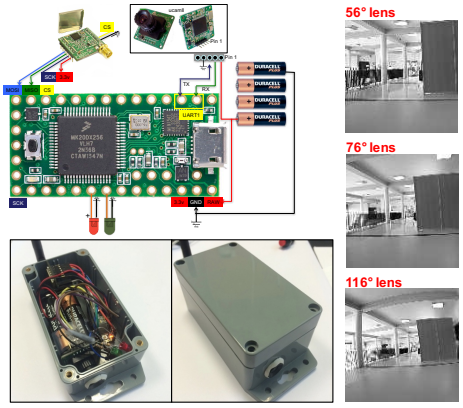


Figure 8: Long-range image IoT

Even if the encryption time may not be important for some IoT applications because of non real-time constraint, a higher encryption time also means a higher energy consumption. As there is an increasing interest in multimedia IoT, especially image IoT, where small images can be transmitted from IoT devices, we developed a long-range image sensor using LoRa radio and a Teensy32 board for surveillance applications [21] as shown in Figure 8.

When transmitting 1 image every hour (about 8 packets of 240 bytes per image) and a mean power consumption of 35mA while in active state and 5μA in deep sleep mode, the lower encryption time of our LSC algorithm provides an additional estimated autonomy of more than 1 month: for instance 730 days compared to 686 days.

VIII. Conclusions

In this paper, an efficient lightweight stream cipher scheme (LSC) was proposed for tiny IoT devices that are limited in terms of energy, resources, and sometimes real-time requirements. The existing standard ciphers are not adapted for these devices since a higher number of round iterations is required to reach the desired security level. In addition, a static round function is usually

applied for each iteration which is why existing approaches use a larger number of rounds r. Our proposed solution LSC reduces r to one iteration and requires less computation and resource overheads. LSC is based on the dynamic key dependence approach to reach a good balance between security level and device's performance. The statistical tests and the experimentations on real IoT hardware show that LSC is a promising candidate for resource-constrained IoT as it exhibits high randomness and uniformity level in addition to a high periodicity in the worst-case scenario, while outperforming traditional AES in terms of encryption/decryption time as well as the more recent Speck algorithm. In addition to the obvious latency reduction, significant energy saving have been quantified when encryption is performed on small images.

ACKNOWLEDGMENTS

This work is supported by (a) the WAZIHUB project funded by EU Horizon 2020 program under grant agreement No 780229, (b) the Maroun Semaan Faculty of Engineering and Architecture at the American University of Beirut and (c) by the EIPHI Graduate School (contract ANR-17-EURE-0002). We also thank the supercomputer facilities of the Mésocentre de calcul de Franche-Comté.

REFERENCES

[1] Daemen, Joan and Rijmen, Vincent, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[2] Noura, Hassan and Chehab, Ali and Sleem, Lama and Noura, Mohamad and Couturier, Raphaël and Mansour, Mohammad M, "One round cipher algorithm for multimedia IoT devices," *Multimedia Tools and Applications*, vol. 77, no. 14, pp. 18 383–18 413, 2018.

[3] Beaulieu, Ray and Shors, Douglas and Smith, Jason and Treatman-Clark, Stefan and Weeks, Bryan and Wingers, Louis, "SIMON and SPECK: Block Ciphers for the Internet of Things." *IACR Cryptology ePrint Archive*, vol. 2015, p. 585, 2015.

[4] Noura, Hassan, "Conception et simulation des générateurs, crypto-systèmes et fonctions de hachage basés chaos performants," Ph.D. dissertation, université de Nantes, 2012.

[5] McKay, Kerry A and Bassham, Larry and Turan, Meltem Sönmez and Mouha, Nicky, "Report on lightweight cryptography," *NIST DRAFT NISTIR*, vol. 8114, 2016.

[6] Poschmann, Axel York, "Lightweight cryptography: cryptographic engineering for a pervasive world," in *PH. D. THESIS*. Citeseer, 2009.

[7] Noura, Hassan and Sleem, Lama and Noura, Mohamad and Mansour, Mohammad M. and Chehab, Ali and Couturier, Raphaël, "A new efficient lightweight and secure image cipher scheme," *Multimedia Tools and Applications*, vol. 77, no. 12, pp. 15 457–15 484, Jun 2018.

[8] Noura, Hassan and Courousse, Damien, "Method of encryption with dynamic diffusion and confusion layers," December 2017.

[9] 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*, pp. 1–30, 2019, to appear.

[10] Roman, Rodrigo and Alcaraz, Cristina and Lopez, Javier and Sklavos, Nicolas, "Key management systems for sensor networks in the context of the Internet of Things," *Computers & Electrical Engineering*, vol. 37, no. 2, pp. 147–159, 2011.

[11] T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, T. Lehman, and B. Schott, "Comparative analysis of the hardware implementations of hash functions sha-1 and sha-512," in *International Conference on Information Security*. Springer, 2002, pp. 75–89.

[12] Paul, Goutam and Maitra, Subhamoy, *RC4 stream cipher and its variants*. CRC press, 2011.

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

[14] Paar, Christof and Pelzl, Jan, *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

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

[16] Pierre L'Ecuyer and Richard J. Simard , "TestU01: A C library for empirical testing of random number generators ," *ACM Trans. Math. Softw* , 2007.

[17] Doty-Humphrey, C., "PractRand," 2014. [Online]. Available: {http://pracrand.sourceforge.net/}

[18] Noura, Hassan and Martin, Steven and Al Agha, Khaldoun and Chahine, Khaled, "ERSS-RLNC: Efficient and robust secure scheme for random linear network coding," *Computer networks*, vol. 75, pp. 99–112, 2014.

[19] C. Pham, "DIY low-cost LoRa IoT framework," 2016. [Online]. Available: {https://github.com/CongducPham/LowCostLoRaGw}

[20] Gerben den Hartog, "Efficient AES implementation for Arduino," 2016. [Online]. Available: {\url{https://github.com/Ideetron/RFM95W_Nexus/tree/master/LoRaWAN_V31}}

[21] C. Pham, "Low-cost, low-power and long-range image sensor for visual surveillance," in *SmartObjects@MobiCom*. ACM, 2016, pp. 35–40.