Efficient Lightweight Cryptography Algorithm in IoT Devices with Real-time Criteria

Paul D. Rosero-Montalvo[1][a] and Vanessa E. Alvear-Puertas[2,3][b]
[1]*Computer Science Department, IT University of Copenhagen, Copenhagen, Denmark*
[2]*Computer Science Department, Universidad de Salamanca, Salamanca, Spain*
[3]*Instituto Superior Tecnológico 17 de Julio, Ibarra, Ecuador*

Keywords: Computer Network Security, Cryptography, Embedded Systems, Internet of Things.

Abstract: Cryptographic algorithms are used to ensure the communication channel between the transmitter and receiver. However, these algorithms are focused on processing data blocks that consume a lot of computational resources. Therefore, they have some constraints to be used in IoT devices. This work presents a lightweight AES cryptographic algorithm designed for IoT devices with real-time operative system criteria to improve the time response and with threads that can be suspended to leverage RAM resources for another task. As a result, we design an AES algorithm with a cipher key updating process that uses 11k bytes of Flash, 820 bytes of RAM, and a time response of around 14.5 us in real scenarios.

1 INTRODUCTION

Internet of Things (IoT) is a network of embedded devices with microcontrollers, microprocessors, specific software, and sensors connected through the Internet to collect and exchange data (Gehrmann and Gunnarsson, 2019). Therefore, IoT provides shared distributed resources and services to different organizations, sites, or homes with the possibility to send data to anyone, any device, and anytime (Latif et al., 2020). IoT uses these embedded devices like tiny computers with defined functionalities and integrates a large variety of sensors with their principal aim to convert a physical phenomenon or human behavior into an electric signal (Rosero-Montalvo et al., 2021). Thus, IoT brings us the possibility to acquire a large amount of data, that through machine learning algorithms, people can get the intrinsic knowledge from the data to make our life more comfortable and constructive (Islam et al., 2019).

However, IoT requires robust protection against possible attacks or vulnerabilities. The security of IoT devices becomes important when linked to human life or industrial procedures. Also, standard security techniques such as cryptography algorithms with user authentication are built to run in traditional computers (end-to-end manner) (Khan et al., 2021). Besides, cryptography algorithms are developed to send a considerable amount of information called ”blocks”, and compiling them requires many computational resources which IoT devices can not run due to complexity. Also, the data acquired for the IoT device does not fit in these blocks leaving empty spaces and wasting memory. Therefore, these mentioned algorithms can provide overhead the computational resources of the IoT devices. Since they can only compile lightweight software with a smaller number of bits at a time with computational restrictions such as RAM and Flash sizes, power consumption, processing speed, and time response (Prakash et al., 2019).

Besides, IoT devices have different wireless protocols to establish communication between them with a low data transfer rate to the cloud, which brings latency issues in real-time applications if the system has a long time response due to run cryptography algorithms (Shah and Venkatesan, 2018).

Currently, there is a new tendency to implement Trusted Execution Environments (TEE) to isolate users on the cloud computing side. At the same time, microprocessors are developed with a trusted platform module (TPM) to protect specific software from the rest of the system operative. Otherwise, in an IoT environment, we can consider that the entire microcontroller is trusted because the device runs

[a] https://orcid.org/0000-0003-1995-400X
[b] https://orcid.org/0000-0003-2332-6073

Rosero-Montalvo, P. and Alvear-Puertas, V.
Efficient Lightweight Cryptography Algorithm in IoT Devices with Real-time Criteria.
DOI: 10.5220/0010922800003194
In *Proceedings of the 7th International Conference on Internet of Things, Big Data and Security (IoTBDS 2022)*, pages 103-109
ISBN: 978-989-758-564-7; ISSN: 2184-4976

only one application without multi-user security concerns (Szymanski, 2017). With this assumption, the leading security flaws happen outside the IoT device. Therefore, the communication channel can be a security concern since most IoT applications are designed in standard wireless protocols such as 802.11 with many vulnerabilities (Naru et al., 2017).

Thereby, new IoT devices are designed to avoid security flaws concerning authentication, such as Arduino IoT products incorporating a crypto chip and specific libraries for the cloud services connection (Sung et al., 2018). The crypto chip is focused on connecting the device to the cloud. However, in wireless sensor networks, IoT devices constantly send information between them, and just one (considered a central node) is in charge of sending the information to the cloud (Fotovvat et al., 2021). As a result, the information sent to the cloud is protected. Nevertheless, this process does not ensure if a malicious user has tampered with the IoT network.

The cryptography application in IoT devices also brings constraints related to system time response. In this scenario, it is essential to improve the code with techniques to run in parallel instructions to use the entire processor at once. A real-time operative system carries on this process, inserting flaws to prioritize procedures and time slots to run each instruction. Thereby, the processor is running the crypto library, at the same time, is checking the battery, calibrating sensors, and sending information to other devices.

With the above mentioned, this article presents a hybrid cryptography algorithm with a real-time operative system and memory optimization criteria. This process is carried on to send encrypted information on a wireless sensor network without exceeding the computational resources of a traditional IoT device and leaving memory space to compile other functionalities. To accomplish this goal, we assumed that the IoT device and sensors are trusted, and only the wireless channel (802.11 n) is not. Also, we define at first the average of RAM and Flash requirement for the algorithm AES 128 from the lightweight crypto library (8 bits) supported by Arduino to measure the RAM increasing to deploy this stage. Later, we design the same application with software optimization techniques and a real-time schedule to improve the system time-respond. Also, we developed a secure way to update the key without external messages from the central node. The top results are the RTOS implementation in the AES encryption/decryption algorithm with only 820 bytes of RAM and 11K bytes of Flash and the response time around 8 uS.

The rest of the manuscript is structured as follows: Section II shows representative works related to cryptographic algorithms implemented in IoT devices and WSN. Section III presents the experimental setup with the assumptions to determine the RAM and Flash requirements. Results are illustrated in section IV with the real-time schedule and optimization criteria. Finally, section VI shows relevant conclusions and future works.

2 LITERATURE REVIEW

Lightweight cryptographic algorithms for IoT are an emergent research field due to their relation to sensitive information collected from human behavior. However, the benefits of IoT come with many security challenges, and traditional cryptographic algorithms need to adapt in hostile environments with constrained computational resources. Therefore, (Khan et al., 2021; Fotovvat et al., 2021), present an extensive comparison of lightweight cryptographic algorithms implemented in several scenarios where the limitations of the IoT device restrict the possible security implementations. Furthermore, (Guo et al., 2019) designed a complexity reduction of the block encryption and their benefits in a complex task such as image encryption. (Jalaly Bidgoly and Jalaly Bidgoly, 2019) presents a novel chaining encryption algorithm for the LPWAN IoT network and the statistical proprieties of its method. Also, (Dang et al., 2021) encrypts the information in LORA communication in a WSN. Works such as (Naru et al., 2017; Prakash et al., 2019; Ramesh and Govindarasu, 2020; Prakash et al., 2019; Hijawi et al., 2021) present novel implementations on lightweight cryptographic modifying the process of traditional algorithms. Specifically, in real-time applications, works like (Gope and Hwang, 2016; Islam et al., 2019) use robust hardware (FPGA or dual ARM processors) than the traditional IoT devices. Finally, (Tsai et al., 2018) presents to AES 128 secure channel in LORAWAN communication for IoT environments.

In conclusion, all these works have presented solutions for data privacy in IoT environments. However, this is not seen integrally with the primary objective of sending information to the Cloud, considering the IoT device an extension of the trusted Cloud. For this reason, the same security trends must be adopted within the IoT environment so that later the integration of these IoT devices is transparent to more extensive networks. As a result, there are open issues like designing a trusted environment in IoT environments with specific assumptions, optimizing the time response, and implementing lightweight cryptography algorithms on particular scenarios such as WSN and IoT.

3 EXPERIMENTAL SETUP

This section presents the description and configuration of Advanced Encryption Standard. Then, we present the assumptions and specific scenarios to determine which is trusted and what is not in the IoT device.

3.1 Advanced Encryption Standard

Advanced Encryption Standard (AES) is a symmetric block cipher for encryption/decryption where a block of plaintext is treated as a single block and is used to obtain a block of ciphertext of the same size. AES supports 128, 192, or 256 encryption keys. Also, AES has five modes, Electronic Code Block (ECB), Cipher Block Chaining mode (CBC mode), Cipher Feedback mode (CFB mode), Output FeedBack mode (OFB mode), and Counter mode (CTR mode) (Serra et al., 2021). However, CBC mode is the most used because CBC encrypts the result to the ciphertext block. In the next block, the encryption results combine with xor gate with plaintext block until the last block. Therefore, even if this mode encrypts the same plaintext block, it will get a different ciphertext block (Borges et al., 2021).

3.2 Existing Libraries

Many libraries implement AES in computers and servers in common programming languages to encrypt/decrypt big datasets. However, they are not compressed into 8 or 16-bit architecture. In this scenario, a few contributions exist to demonstrate the functionality of AES into IoT devices uncommonly send 16 bytes of data, such as in environmental conditions or water quality analysis (Rosero-Montalvo et al., 2021). In this article, we use two popular libraries as `Lib one` (Van Heesch, 2018) and `Lib two` (Landman,). However, the real-time operative system has not been implemented in encryption libraries yet.

3.3 Assumptions and Specific Scenario

The assumptions to design the AES encryption/decryption algorithm are: (i) the IoT device trusts that data acquired from sensors have not been tampered with in any form, (ii) all the software running into the IoT device is trusted, (iii) the RAM must bein optimized to run complex tasks regarding their computational constraints, (iv) the communication scheme is in star; slaves nodes send the information to a single central node, (v) the central node already has the encryption key before the system is implemented, and (vi) the IoT device does not use 16 bytes in sensors data acquisition.

AES Description: is a synchronic cipher algorithm that the transmitter and the receiver need to know the cipher key. Therefore, updating the cipher key is a complex task (asynchronous cipher). However, the encryption key can be updated if two random numbers are generated, the first to determine how many times the encryption key is to be used. When the data send number reaches this number, an XOR gate is executed between the encryption key and the second random number. Later, this second number is inserted in a specific position in the ciphertext that only the central node knows to update its cipher key in the following shipments.

Allocating the key in the Flash memory can be risky. Therefore, we use the EEPROM to hide the key how as possible. To leverage RAM, the Rijndael S-box is allocated into the flash. Finally, RTOS is implemented to improve the time-respond performance emulating symmetric multiprocessing into the IoT device.

Specific Scenario: The IoT system developed is an autonomous robot working in greenhouses acquiring data from environmental conditions. Therefore, the sensors used are SCD30 for temperature, humidity, and temperature. In addition, the VEML6075 is used to measure levels of UV rays. Also, it is necessary to acquire data from motors to get information about their velocity and gyration. Fig. 1 shows the system working on the greenhouse.

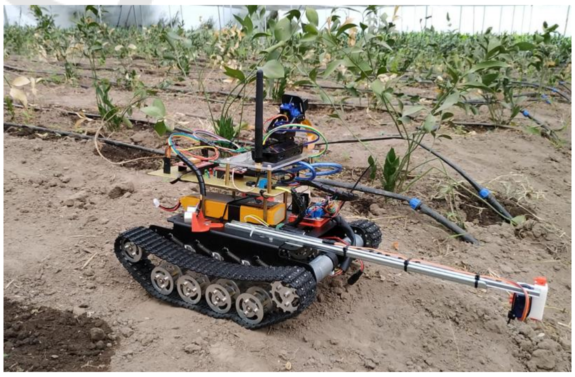


Figure 1: A robot working in the greenhouse.

RTOS Procedures: The autonomous robot has the following steps: Motor control, receiving motor parameters, checking the battery, acquiring data, encrypting data, sending information. Hence, this research only presents the encryption data. The encryption data has the stages: `Plain text`: 16 hexadecimal bytes, `Data rotation`: byte substitution (x-box), and shift rows. `Round key`: circular byte left shift, byte substitution (x-box), adding round constant. `Cipher-text`: xor gate between Data rotation and Round key.

4 RESULTS AND TESTS

This section shows the memory consumption of each library used in this work and the way of the ROS. Then, the summary of RAM, flash, and response time are presented.

4.1 Memory Consumption

To determine the initial resources to compile the AES algorithm, we define the memory and time-respond of lib one and lib two. Then, we implemented the algorithm with the majority of the variables in the RAM without any optimization (`Lib AES v1`). Later, we define the right size of each variable and put some of them (considered as constants) in the FLASH (`Lib AES v2`). Next, we define code blocks to make threads, put them in RTOS architecture, and measure the time-respond and memory. Finally, regarding the mentioned assumptions, we allocated the encryption key in the ROM, hidden of the principal code with aleatory updates explained in the next section (`AES-RTOS v3`).

4.2 RTOS Implementation

AES algorithm has four threads defined, `Plain text`: 16 hexadecimal bytes, `Data rotation`: byte substitution (x-box), and shift rows. `Round key`: circular byte left shift, byte substitution (x-box), adding round constant. `Cipher-text`: xor gate between Data rotation and Round key. Therefore, we updated the current design to fit into IoT devices as follows: `Plaintext`: data acquisition from sensors, filling empty spaces in hexadecimal code (the system uses 8 bytes). Also, the key-updated works as follows: the encryption key is stored before the system runs the first time (the central node also has the encryption key for each IoT node), and one flag is high. When the IoT device runs the first time and looks at the flag in high, the IoT device runs the first random number to determine how many times the ciphertext will send with the same cipher key. When this number comes to an end, the cipher key is updated, running an XOR gate with a second 8-bit number. Then, this second number is allocated in the Plaintext to be encrypted and sent to the central node. Finally, the first random number is generated again to repeat this process. The system generates the first random number without issues if a hard reset is generated, even if the flag is low. The difference between the standard AES process with the improved AES-RTOS is shown in Fig. 2.

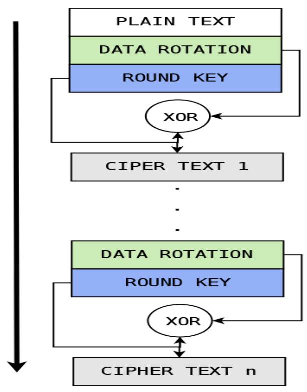
Table V shows the summary of RAM, Flash, and response time for each library. For example, lib one uses much RAM, and lib two takes too long to encrypt the text. Otherwise, Lib AES V1 and V2 need less computational resources with better response time than the last libraries. Otherwise, Lib AES-RTOS V3 consumes more Flash than the others; each thread has an average of 60 bytes. However, RTOS has better management of resources; priorities threads make it feasible to suspend and reactivate them, freeing up RAM. The library is available in: AES-RTOS Lib. The pseudocode is presented in 1

Table 1: Summary of the Computational resources consumption for each AES library.

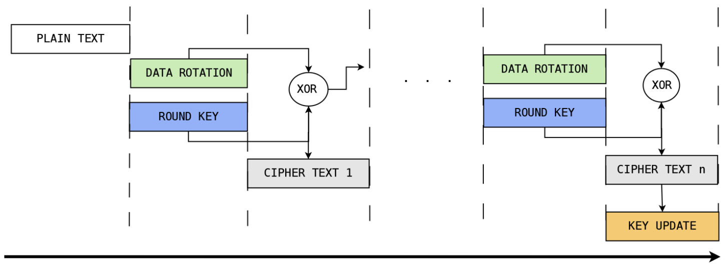
Libraries	FLASH	RAM	Time to respond
Lib one	6888	1065	3.05 us
Lib two	7468	800	14.36 us
Lib AES V1	5056	607	10.36 us
Lib AES V2	4626	353	5,10 us
Lib AES-RTOS V3	11228	818	8.36 us

4.3 Real Scenario

The proposed algorithm has been validated in a real scenario by sending data from environmental conditions inside the greenhouses. The data is acquired by a network of sensors installed in autonomous robots (1). Therefore, it is assumed that its design and function have been checked. The designed data frame is sent through the wireless LoRa protocol with a 4-bytes preamble, a 4-bytes header, and a 2-bytes end of frame and error control. The lightweight library is designed to compile data with 8bits size to compile in 8 bits architecture such as many microcontrollers used to developt IoT devices. Therefore, the designed payload (16-bytes) includes the robot's identification, the sensor network's data, motor movement parameters, and, if necessary, the update of the key as follows: CO2: 2 Bytes, Temperature: 2 bytes, Humidity:2 bytes, UV rays: 2 bytes, Battery state: 2 bytes, motor A velocity: 1 byte, motor B velocity: 1 byte, motor A gyration:1 byte, motor B gyration: 1 byte, node identification: 1 byte, and key updated: 1 byte. We have to add an extra function to divide one parameter into two bytes, requiring less memory to use 16 bits.



(a) Traditional AES.



(b) AES-RTOS implementation.

Figure 2: AES execution time comparison, execution time: ($\rightarrow$).

Consequently, the AES algorithm was implemented into the IoT device to get performance measurements and determine if the server has issues decrying the information when the data frame traveled for wireless communication. We tested the system with the RTOS implementation and without them. As a result, the system with RTOS has a time response of around 14.5 us, and just encrypting the data and sending it to the server takes 21.4 us; this is because, with RTOS, we can run in parallel stages to improve the performance. Also, we can stop the machine efficiently by changing the priority of each task or suspending them. For this reason, the battery consumption decreased around 15

5 CONCLUSIONS AND FUTURE WORKS

Cryptographic algorithms allow us to ensure that the data sent reaches its receiver safely. However, this computational cost is essential in implementing a more significant amount of code to strengthen the application, such as machine learning algorithms. It has been seen that the previous libraries have an adequate consumption for microcontrollers with 8-bit architectures. However, they are not intended for their genuine use in real applications. For this reason, the implementation of RTOS allows to determine processes and know important information such as their response time and memory size required to compile them.

In addition, it helps code optimization by freeing up RAM when suspending a thread. We were able to show that despite optimally implementing AES, we left much of the Flash and RAM free for other processes by the microcontroller. For example, our library, compiled on an Arduino Uno, occupies 32% of Flash and 39% of RAM with response times between 5 and 8 us. Also, if we use an ESP8266 board, AES-RTOS uses 25% of the Flash and 33% of the RAM with response times between 7 and 12 us. In addition, the receiver with the highest computational capacity can efficiently manage the exchange of keys, and there was no loss of packets due to the update of the encryption key. As a result, we leave space available for future work to implement machine learning algorithms for on-site decision-making.

ACKNOWLEDGEMENT

Novo Nordisk Fonden supports this research with the project: Privacy through Co-Design for Real-World Data Analytics in the Cloud.

REFERENCES

Borges, M., Paiva, S., Santos, A., Gaspar, B., and Cabral, J. (2021). Azure rtos threadx design for low-end nb-iot device. In *2020 2nd International Conference on Societal Automation (SA)*, pages 1–8.

Dang, T.-P., Tran, T.-K., Bui, T.-T., and Huynh, H.-T. (2021). Lora gateway based on soc fpga platforms. In *2021 International Symposium on Electrical and Electronics Engineering (ISEE)*, pages 48–52.

Fotovvat, A., Rahman, G. M. E., Vedaei, S. S., and Wahid, K. A. (2021). Comparative performance analysis of lightweight cryptography algorithms for iot sensor nodes. *IEEE Internet of Things Journal*, 8(10):8279–8290.

Gehrmann, C. and Gunnarsson, M. (2019). An Identity Privacy Preserving IoT Data Protection Scheme for Cloud Based Analytics. *Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019*, pages 5744–5753.

Algorithm 1: AES Implementation,

```
Require: Queue, s-box[], input [4][4], output[16], cipher[4][4], key[16], random
  procedure PLAIN TEXT:(priority: 3, stack size: 80)
      for $j \leftarrow 0, 16$ do
          if $j < sensors$ then
              input[:] ← data
          else
              input[:] ← 0x00
          end if
          if r == random then
              input[:sensors] ← flag
              random function
          end if
      end for
  end procedure
  procedure DATA ROTATION:(priority: 2, stack size: 80)
      for $i \leftarrow 0, 10$ do
          input ← S-box(input[:])
          for $k \leftarrow 1, 4$ do
              input[:i] ← input[:i] >> k
              Round constant input[:]
          end for
          Queue ← i
      end for
  end procedure
  procedure ROUND KEY:(priority: 2, stack size: 80)
      cipher ← key
      for $l \leftarrow 0, 10$ do
          g(cipher[3])
          S-box(cipher[3])
          cipher[:] ← cipher[:] ⊕ S-box(cipher[:3])
          for $n \leftarrow 1, 4$ do
              cipher[:n]← cipher[:n-1] ⊕ cipher[:n]
          end for
      end for
  end procedure
  procedure CIPHER TEXT:(priority: 1, stack size: 80)
      if Queue ≠ 0 ∧ Queue ≠ 10 then
          out [:] ← input[:] ⊕ cipher[:]
      else if Queue == 10 then
          Send → out[:]
      end if
  end procedure
```

Gope, P. and Hwang, T. (2016). A Realistic Lightweight Anonymous Authentication Protocol for Securing Real-Time Application Data Access in Wireless Sensor Networks. *IEEE Transactions on Industrial Electronics*, 63(11):7124–7132.

Guo, X., Hua, J., Zhang, Y., and Wang, D. (2019). A complexity-reduced block encryption algorithm suitable for internet of things. *IEEE Access*, 7:54760–54769.

Hijawi, U., Unal, D., Hamila, R., Gastli, A., and Ellabban, O. (2021). Lightweight kpabe architecture enabled in mesh networked resource-constrained iot devices. *IEEE Access*, 9:5640–5650.

Islam, M. S., Verma, H., Khan, L., and Kantarcioglu, M. (2019). Secure real-time heterogeneous IoT data management system. *Proceedings - 1st IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications, TPS-ISA 2019*, pages 228–235.

Jalaly Bidgoly, A. and Jalaly Bidgoly, H. (2019). A novel chaining encryption algorithm for lpwan iot network. *IEEE Sensors Journal*, 19(16):7027–7034.

Khan, M. N., Rao, A., and Camtepe, S. (2021). Lightweight Cryptographic Protocols for IoT-Constrained Devices: A Survey. *IEEE Internet of Things Journal*, 8(6):4132–4156.

Landman, D. Arduino library for aes encryption.

Latif, M. A., Ahmad, M. B., and Khan, M. K. (2020). A Review on Key Management and Lightweight Cryptography for IoT. *2020 Global Conference on Wireless and Optical Technologies, GCWOT 2020*.

Naru, E. R., Saini, H., and Sharma, M. (2017). A recent review on lightweight cryptography in iot. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 887–890.

Prakash, V., Singh, A. V., and Kumar Khatri, S. (2019). A New Model of Light Weight Hybrid Cryptography for Internet of Things. *Proceedings of the 3rd International Conference on Electronics and Communication and Aerospace Technology, ICECA 2019*, pages 282–285.

Ramesh, S. and Govindarasu, M. (2020). An efficient framework for privacy-preserving computations on encrypted iot data. *IEEE Internet of Things Journal*, 7(9):8700–8708.

Rosero-Montalvo, P. D., López-Batista, V. F., Arciniega-Rocha, R., and Peluffo-Ordóñez, D. H. (2021). Air Pollution Monitoring Using WSN Nodes with Machine Learning Techniques: A Case Study. *Logic Journal of the IGPL*. jzab005.

Serra, L. F. D., Gonçalves, P. G. B., Lopes Frazão, L. A., and Antunes, M. J. G. (2021). Performance analysis of aes encryption operation modes for iot devices. In *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6.

Shah, T. and Venkatesan, S. (2018). Authentication of IoT Device and IoT Server Using Secure Vaults. *Proceedings - 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications and 12th IEEE International Conference on Big Data Science and Engineering, Trustcom/BigDataSE 2018*, pages 819–824.

Sung, B.-Y., Kim, K.-B., and Shin, K.-W. (2018). An aes-gcm authenticated encryption crypto-core for iot security. In *2018 International Conference on Electronics, Information, and Communication (ICEIC)*, pages 1–3.

Szymanski, T. H. (2017). Security and Privacy for a Green Internet of Things. *IT Professional*, 19(5):34–41.

Tsai, K.-L., Huang, Y.-L., Leu, F.-Y., You, I., Huang, Y.-L., and Tsai, C.-H. (2018). Aes-128 based secure low power communication for lorawan iot environments. *IEEE Access*, 6:45325–45334.

Van Heesch, D. (2018). Arduino cryptography library.