Implementation of Password Hashing on Embedded Systems with Cryptographic Acceleration Unit

Holman Montiel A, Fredy Martínez S, Edwar Jacinto G
Facultad Tecnológica
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia

***Abstract*—In this modern world where the proliferation of electronic devices associated with the Internet of Things (IoT) grows day by day, security is an imperative issue. The criticality of the information linked to the various electronic devices connected to the Internet forces developers to establish protection mechanisms against possible cyber-attacks. When using computer equipment or servers, security mechanisms can be applied without having problems with the number of resources associated with this activity; the opposite is the case when implementing such mechanisms on embedded systems. The objective of this document is to implement password hashing on a FRDM-K82F development board with ARM® Cortex™-M4 processor. It describes the basic criteria necessary to aim at moderate levels of security in specific purpose applications; that can be developed taking advantage of the hardware cryptographic acceleration units that these embedded systems have. Performance analysis of the implemented hash function is also presented, considering the variation in the number of iterations performed by the development board. The validation of the correct functioning of the hashing scheme using the SHA-256 algorithm is carried out by comparing the results obtained in real-time versus an application developed in Python software using the PyCryptodome library.**

Keywords—Cryptography; password hashing; embedded systems; cryptographic acceleration hardware; SHA-256

I. Introduction

One of the current priorities with the boom and great demand for devices associated with the Internet of Things (IoT) [1], in the face of multiple interconnectivity environments is security [2],[3]. While it is true that day by day the development of specific solutions or electronic control units associated with communication processes; establishes a great demand by this society interconnected to the web [4]; as developers, it must be considered that there are a wide variety of tools both software and hardware [5]; which allow to improve and optimize security against the handling of critical information [6],[7]. The possible vulnerability and impact on the integrity of the information make security a key point in any electronic development; thus, establishing a primordial factor in the selection criteria of the possible users of these technological solutions [8].

A fundamental characteristic that must be considered when implementing IoT on embedded systems is that most of these devices have limitations associated with computational power and speed [9], [10]. Although this could be a limitation when developing electronic control units with cryptographic functions [11]; many applications are being developed that make use of hardware coprocessors that facilitate the implementation of various cryptographic algorithms and hash functions [12], [13]. This allows the use of these embedded systems to be applied not only in encryption and decryption tasks [14], but also in hashing and authentication in complex environments with limited computing resources [15],[16]. The current literature shows us a great variety of security implementations on embedded systems in which performance analysis is performed for both symmetric and asymmetric algorithms [17], [18]; this shows us that cryptographic hardware continues in a constant process of evolution due to the great demand for efficient [19], reliable, portable [20], and secure IoT technological products [21].

In this complex scenario of IoT interconnectivity, some organizations have fallen prey to cyber-attacks in which they have been exposed thanks to vulnerabilities exploited by poor password protection practices. The exploitation of this vulnerability has managed to expose many user accounts and credentials; significantly affecting the reliability of the use of web applications and embedded solutions used in home automation and industrial areas [6],[12],[14]. One way to counteract this phenomenon and at the same time guarantee a high level of security related to user accounts is through the implementation of password hashing schemes (PHS) [13].

Considering the above, this work aims to show the implementation and validation of a password hashing on the FRDM-K82F embedded system. The document intends to develop in a simple and practical way a first approach to the use of Arm Mbed TLS libraries; thus, providing a basic and functional solution for those who make their first approach to the use of processors with cryptographic acceleration units. A performance analysis associated with the variation in the number of iterations used for the generated summary function is also shown.

This contribution is presented in the following sections, which are organized as follows: Section II describes the basic theoretical concepts associated with the use of embedded systems with cryptographic acceleration unit. Section III describes the implementation and development of the proposal. Section IV presents the results obtained, and finally, the conclusions and future work are presented in Section V.

II. Methodology

The constant evolution of digital devices in terms of their cryptographic modules has allowed a lot of opportunities;

associated with the linking of security parameters in the development of specific purpose applications. Using the FRDM-K82F development platform, a step-by-step implementation of password hashing is carried out, making use of the Cryptographic Acceleration Unit (CAU). The validation and verification of the SHA-256 algorithm and the respective results are performed by means of the Python PyCryptodome library; this to verify the correct operation of the proposed solution on the embedded system used.

A. *Cryptographic Acceleration Unit*

A wide variety of embedded systems can be found in the market that has this type of module incorporated in their architecture, see Fig. 1. In general, terms, the Cryptographic Acceleration Unit (CAU) is a ColdFire® coprocessor that is accessed by the CPU using specialized hardware operations [21], [22]. The purpose of this unit is to increase the performance of software-based hashing and encryption functions, thus guaranteeing acceleration and high performance when using algorithms such as DES, 3DES, AES, MD5, SHA, among others.

Also available are some Kinetis® MCU processors that have the memory-mapped cryptographic acceleration unit (mmCAU), a coprocessor that is connected to the processor's private peripheral bus (PPB), as shown in Fig. 2. These units are focused on improving the performance of software-based security encryption/decryption operations.

B. *Password Hashing*

One way to increase security in any communication and information transfer process associated with applications that link user accounts with their respective access passwords; is to move from storing passwords in plain text to using a hash function on the respective password, as shown in Fig. 3. A hash function makes it possible to encrypt a password by taking advantage of the fact that this type of algorithm; takes any size of data and converts it into a fixed length of information [13]. To be more precise, the aim is to make it impossible to recover the password from the generated hash.

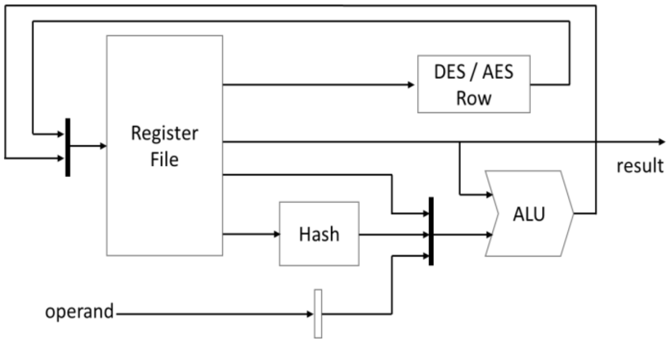


Fig. 1. Block Diagram of the CAU Module [22].

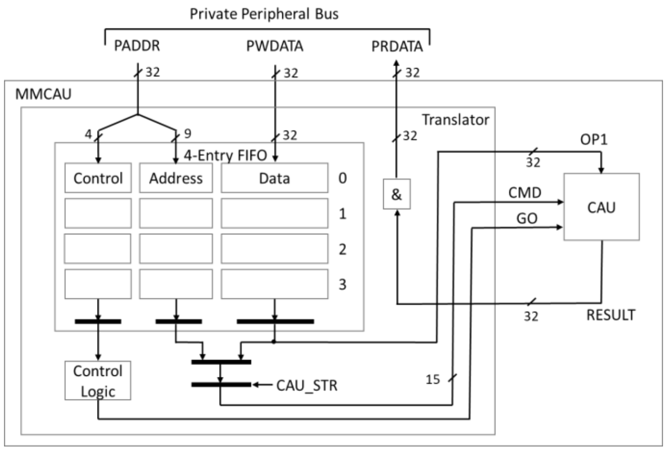


Fig. 2. mmCAU Block Diagram [23].

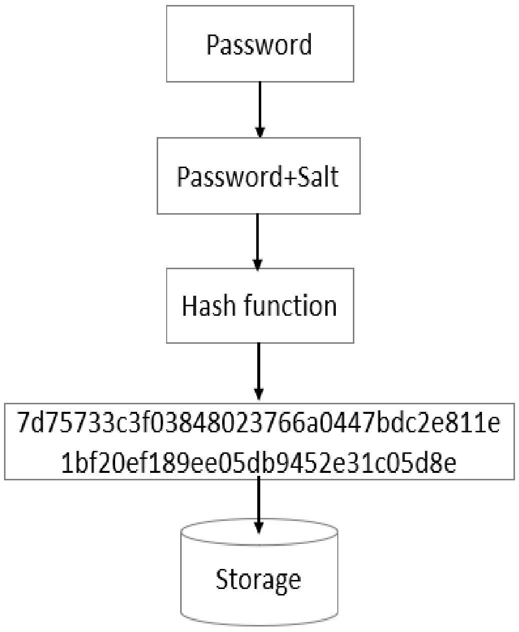

Password
Password+Salt
Hash function
7d75733c3f03848023766a0447bdc2e811e
1bf20ef189ee05db9452e31c05d8e
Storage

Fig. 3. Suggested Password Storage.

As a good security practice, it is not recommended to store passwords in unencrypted text; since any attacker could access them and obtain all the information directly. By storing the hash of these passwords and taking advantage of the fact that these functions are not reversible, the security vulnerability of the respective system is significantly reduced. The validation process of this technique is simple; initially, the input data is taken, the same hash function is executed, and then it is analyzed if this result matches the information stored in the password store, see Fig. 4.

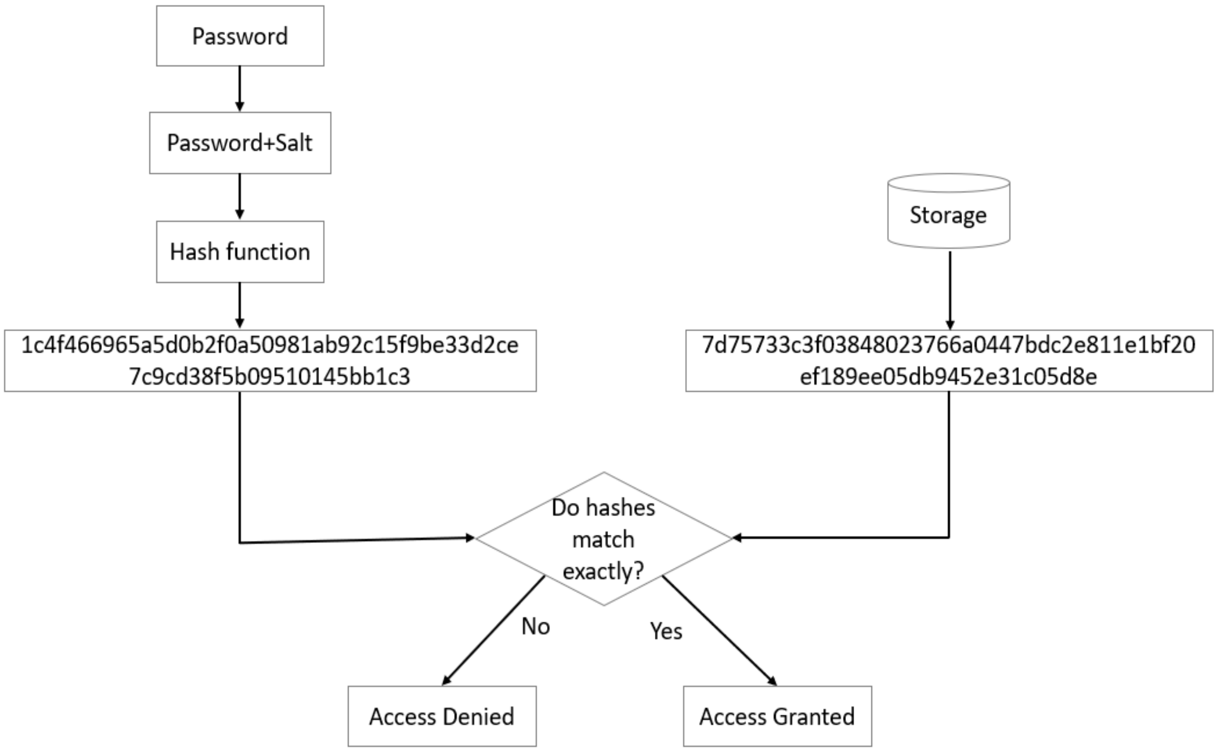
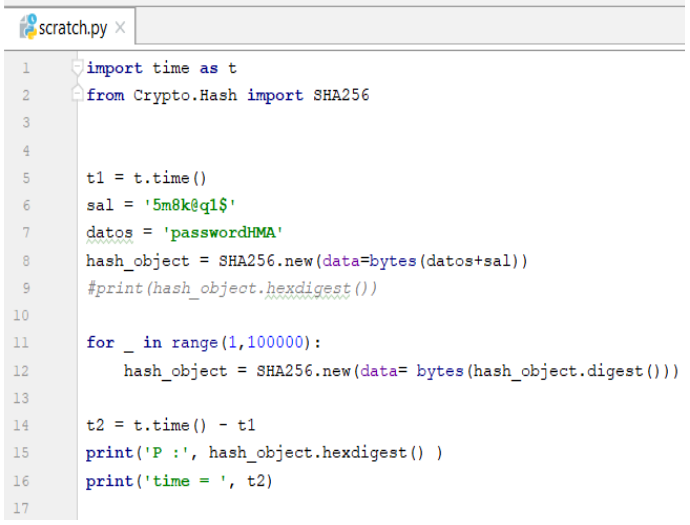

Password
Password+Salt
Hash function
1c4f466965a5d0b2f0a50981ab92c15f9be33d2ce 7c9cd38f5b09510145bb1c3
Storage
7d75733c3f03848023766a0447bdc2e811e1bf20 ef189ee05db9452e31c05d8e
Do hashes match exactly?
No
Yes
Access Denied
Access Granted

Fig. 4. Password Validation.

III. Implementation

Initially, use is made of the PyCryptodome libraries, which is an autonomous package of low-level cryptographic primitives supported by Python. This software tool allows an external validation of the operation of the cryptographic process to be implemented. It can be said that first the operational validation of the process will be performed by means of software simulation; to later compare the results obtained with the implementation on hardware with the FRDM-K82F platform, see Fig. 5.



```
import time as t
from Crypto.Hash import SHA256


t1 = t.time()
sal = '5m8k@q1$'
datos = 'passwordHMA'
hash_object = SHA256.new(data=bytes(datos+sal))
#print(hash_object.hexdigest())

for _ in range(1,100000):
    hash_object = SHA256.new(data= bytes(hash_object.digest()))

t2 = t.time() - t1
print('P :', hash_object.hexdigest() )
print('time = ', t2)
```

Fig. 5. Source Code used in Python for Validation of Results.

As the fundamental objective of the application is to make use of the cryptographic acceleration unit (CAU) of the FRDM-K82F card; which has an ARM® Cortex®-M4 core running at up to 150 MHz, with KB256 of Flash and 256 KB of RAM. The source code is developed using the Mbed OS compiler, which provides a comprehensive SSL/TLS solution called Arm MbedTLS [24]. This library simplifies the integration of cryptographic solutions because it is compact and generic; it should be noted that it can only be used in ColdFire and Kinetis devices with CAU or mmCAU hardware coprocessors. The following encryption/decryption algorithms and hash functions can be used with this library: AES128, AES192, AES256, DES, 3DES, MD5, SHA1, and SHA256.

This implementation was carried out using the Mbed-Studio compiler; this has a large repository of examples associated with the security schemes [25]. It must be considered that at the moment of creating the source code and loading the libraries; the compiler must initially evaluate if the processor has the cryptographic acceleration unit required for the use of the respective libraries; based on this concept, some components of the source code would be as follows:

```
#include "mbed.h"

#include "mbedtls/sha256.h" /* SHA-256 only */

#include "mbedtls/md.h" /* generic interface */

#include < cstdio>

#if DEBUG_LEVEL > 0

#include "mbedtls/debug.h"
```

```
#endif

#include "mbedtls/platform.h"

#include < string.h>
```

The call to the Hash function is quite simple, the definition of the respective input and output variables must be considered; for this case it must be considered that the output associated to a SHA256 function is of 32 bytes, with respect to the input it must be remembered that it can be of any length. The parameters and input variables would be:

```
static const char dato_input[] = "passwordHMA5m8k@q1$";

static const unsigned char *input_buffer = (const unsigned
char *) dato_input;

static const size_t dato_len = strlen(dato_input);
```

With respect to the execution of the SHA function, four functional requirements must be considered. These are: Data buffer, buffer length, Output buffer and a parameter defining whether to use the full SHA-256 or the SHA-224 variant. In this case, this value must be 0 (to use SHA-256).

```
unsigned char output1[32]; /* SHA-256 outputs 32 bytes */

unsigned char output2[32];

t.start();

mbedtls_sha256(input_buffer, dato_len, output1, 0);

for (int i=1; i<=99999; i++){

mbedtls_sha256(output1, 32, output1, 0);

  }

  t.stop();
```

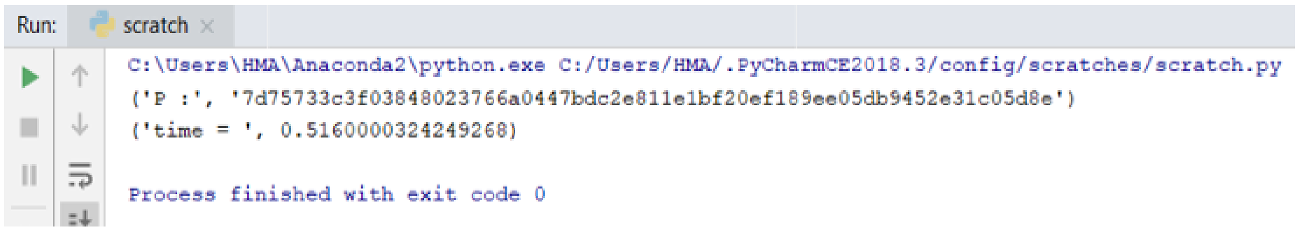
IV. Results

To validate the effectiveness of the password hashing implementation on the selected hardware, we proceeded to compare the result obtained with the implementation done entirely on the PC using pyCryptodome, (see characteristics in Table I). The execution times were measured both in the PC implementation and in the embedded system, performing a variation between the number of iterations associated with the selected hash function.

As general concepts, a test password of 11 bytes in length was used, encrypted with a SHA256 algorithm (password + salt). The salt used was eight bytes long. For the information associated with the salt, a test constant was used and there was no code segment associated with its generation by the embedded system. The validation and verification were performed by comparing the output of the simulation performed in Python versus the result given by the serial port of the development board. Comparisons of up to 100,000 iterations were performed, demonstrating the full functionality of the implementation, see Fig. 6 and Fig. 7.

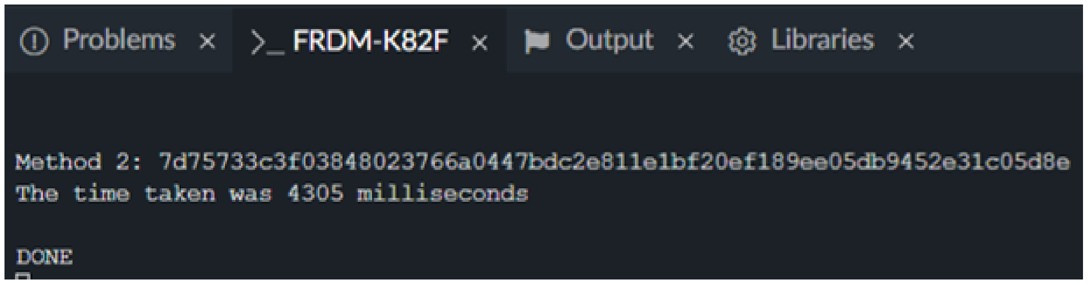
TABLE I. Characteristics of the Devices used

Characteristics of the processors used		
Platform:	**Embedded System**	**PC**
Reference:	FRDM-K82F	ROG GL553VD
Processor:	Kinetis MK82FN256VLL15 (ARM® Cortex™-M4)	Intel® Core™ i7-7700HQ
Clock frequency:	150 MHz	2.8 GHz
RAM:	256 KB	12 GB



```
C:\Users\HMA\Anaconda2\python.exe C:/Users/HMA/.PyCharmCE2018.3/config/scratches/scratch.py
('P :', '7d75733c3f03848023766a0447bdc2e811e1bf20ef189ee05db9452e31c05d8e')
('time = ', 0.5160000324249268)

Process finished with exit code 0
```



```
Method 2: 7d75733c3f03848023766a0447bdc2e811e1bf20ef189ee05db9452e31c05d8e
The time taken was 4305 milliseconds

DONE
```

Fig. 6. Comparison of Software vs. Hardware Results.

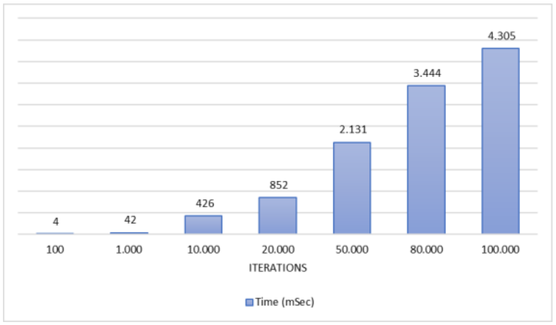


Fig. 7. Execution Time on the FRDM-K82F Board.

V. Conclusion and Future Work

This document presents simply and easily a password hashing scheme that can be implemented on any embedded system with cryptographic acceleration hardware; it provides the key concepts so that people who are starting in the subject of embedded cryptography can begin to incorporate these security measures for their respective developments. It shows that it is possible to make use of a technological solution that offers a moderate level of security using electronic devices with limited computational resources. Although in most IoT applications, communication between devices is short term and there is not a high rate of information transfer; the protection of session passwords becomes a fundamental objective in terms of security. It is hoped that this type of examples will encourage developers to incorporate new methodologies for protecting information on the design of IoT devices.

As future work, we intend to develop specific application hardware to enable digital signatures by implementing hash functions and lightweight encryption algorithms.

Acknowledgment

This work was supported by the Universidad Distrital Francisco José de Caldas, in part through CIDC, and partly by the Technological Faculty. The views expressed in this paper are not necessarily endorsed by the university. The authors thank the research group ARMOS for supporting the development of the code and the implementation on hardware.

References

[1] S. Surendran, A. Nassef and B. D. Beheshti, "A survey of cryptographic algorithms for IoT devices," *2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pp. 1-8, 2018.

[2] B. Vinayaga Sundaram, Ramnath M., Prasanth M. and Varsha Sundaram J., "Encryption and hash based security in Internet of Things," *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*, pp. 1-6, 2015.

[3] M. El-Haii, M. Chamoun, A. Fadlallah and A. Serhrouchni, "Analysis of Cryptographic Algorithms on IoT Hardware platforms," *2018 2nd Cyber Security in Networking Conference (CSNet)*, pp. 1-5, 2018.

[4] P. Flood and M. Schukat, "Peer to peer authentication for small embedded systems: A zero-knowledge-based approach to security for the Internet of Things," *The 10th International Conference on Digital Technologies 2014*, pp. 68-72, 2014.

[5] C. Profentzas, M. Günes, Y. Nikolakopoulos, O. Landsiedel and M. Almgren, "Performance of Secure Boot in Embedded Systems," *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 198-204, 2019.

[6] R.V. Rashmi and A. Karthikeyan, "Secure boot of Embedded Applications - A Review," *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 291-298, 2018.

[7] A. Mundra, A., & H. Guan, "Secure Boot on Embedded Sitara Processors". *Texas Instrument*, 2018.

[8] N. B. Silva, D. F. Pigatto, P. S. Martins, & K.C.Branco, "Case studies of performance evaluation of cryptographic algorithms for an embedded system and a general purpose computer", *Journal of Network and Computer Applications*, Vol. 60, pp. 130-143, 2016.

[9] N. J. G. Saho, & E. C. Ezin, "Survey on Asymmetric Cryptographic Algorithms in Embedded Systems", *IJISRT*, vol *5*, no. 12, pp. 544-554, 2020.

[10] A. J. Acosta, T. Addabbo, & E. Tena-Sánchez, "Embedded electronic circuits for cryptography, hardware security and true random number generation: an overview", *International Journal of Circuit Theory and Applications*, vol. *45, no.* 2, pp.145-169, 2017.

[11] Z. Musliyana, T. Y. Arif, & R. Munadi, "Security enhancement of advanced encryption standard (AES) using time-based dynamic key generation", *ARPN Journal of Engineering and Applied Sciences*, vol *10, no.* 18, pp. 8347-8350, 2015.

[12] W. Wang et al. "XMSS and Embedded Systems". In: Paterson K., Stebila D. (eds) Selected Areas in Cryptography – SAC 2019. SAC 2019. Lecture Notes in Computer Science, vol 11959. Springer, Cham, pp. 1-33, 2020.

[13] G. Hatzivasilis, I. Papaefstathiou, C. Manifavas, I. Askoxylakis, "Lightweight Password Hashing Scheme for Embedded Systems", In: Akram R., Jajodia S. (eds) Information Security Theory and Practice. WISTP 2015. Lecture Notes in Computer Science, vol 9311. Springer, Cham, 2015.

[14] A. Flores-Vergara, E. Inzunza-González, E. E. García-Guerrero, O. R. López-Bonilla, E. Rodríguez-Orozco, J. M. Hernández-Ontiveros, E. Tlelo-Cuautle, "Implementing a chaotic cryptosystem by performing parallel computing on embedded systems with multiprocessors", *Entropy*, vol. *21, no.* 3, pp. 1-28, 2019.

[15] M. Mozaffari-Kermani, K. Tian, R. Azarderakhsh and S. Bayat-Sarmadi, "Fault-Resilient Lightweight Cryptographic Block Ciphers for Secure Embedded Systems," in *IEEE Embedded Systems Letters*, vol. 6, no. 4, pp. 89-92, Dec. 2014.

[16] P. Branco, L. Fiolhais, M. Goulão, P. Martins, P. Mateus, and L. Sousa, "ROTed: Random Oblivious Transfer for embedded devices", *TCHES*, vol. 2021, no. 4, pp. 215–238, Aug. 2021.

[17] L. Baldanzi, L. Crocetti, F. Falaschi, M. Bertolucci, J. Belli, L. Fanucci, and S. Saponara, "Cryptographically Secure Pseudo-Random Number Generator IP-Core Based on SHA2 Algorithm" *Sensors, vol.* 20, no. 7, 1869, pp. 1-13, 2020.

[18] S. Falas, C. Konstantinou and M. K. Michael, "A Hardware-based Framework for Secure Firmware Updates on Embedded Systems," *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 198-203, 2019.

[19] Z. He, W. Chen, X. Xu, L. Harn, & M. Wan, "Reliable and efficient PUF-based cryptographic key generator using bit self-tests", *Electronics Letters*, vol. *56, no.*16, pp. 803-806, 2020.

[20] Z. Gu, G. Han, H. Zeng and Q. Zhao, "Security-Aware Mapping and Scheduling with Hardware Co-Processors for FlexRay-Based Distributed Embedded Systems," in IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 10, pp. 3044-3057, 1 Oct. 2016.

[21] K. Q. Ye, M. Green, N. Sanguansin, L. Beringer, A. Petcher, and A. W. Appel, "Verified Correctness and Security of mbedTLS HMAC-DRBG". *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*, 2017.

[22] L. Casado, "Soluciones de seguridad de Freescale parte III: aceleradores criptográficos en la familia de procesadores Coldfire", *Revista española de electrónica*, (649), pp. 76-80, 2008.

[23] Freescale, "Kinetis MCUs Securing the Internet of Tomorrow", Rev2, pp. 1-12, 2015.

[24] ARM mbed,"mbed TLS v2.16.1 source code documentation", 2021.

[25] ARM mbed, "Tutorial and official examples", Repository Mbed OS 6, 2021.