Providing Confidentiality and Integrity in Ultra Low Power IoT Devices

Emanuele Valea, Mathieu da Silva, Marie-Lise Flottes, Giorgio Di Natale, Sophie Dupuis, Bruno Rouzeyre

Providing Confidentiality and Integrity in Ultra Low Power IoT Devices

Emanuele Valea[1], Mathieu Da Silva[1], Marie-Lise Flottes[1], Giorgio Di Natale[2], Sophie Dupuis[1], Bruno Rouzeyre[1]
[1]LIRMM (Université de Montpellier – CNRS), Montpellier, France
[2]Univ. Grenoble Alpes, CNRS, Grenoble INP*, TIMA, Grenoble, France

***Abstract*—IoT devices for low power applications often rely on energy harvesting techniques. In these implementations, IoT nodes must be aware of the amount of available energy. When the energy is about to run out, they need to halt and save the context of the CPU, which is restored as soon as the energy is available again. Some of the existing techniques rely on the transfer of the scan chains content to a Non-Volatile Memory (NVM). However, the saved state of the device might be the target of malicious attacks, including tampering and theft of confidential data. One solution is to provide the encryption and the integrity of context data. In this paper, we propose the implementation of an additional module interposed between the CPU and the NVM, which provides both these security features.**

Keywords—IoT, Hardware Security, Context Saving, PUF

I. INTRODUCTION

The *Internet of Things* (IoT) has known an extraordinary fast dissemination in the last years. Nowadays, we see more and more often the commercialization of IoT devices dealing with numerous every-day-life applications. Very often, these devices integrate security functions because they manage encrypted communications over the network, or because they deal with confidential and private data [1].

Energy consumption is a very tight design constraint in IoT-oriented designs. Consequently, techniques have been developed that allow IoT nodes to be independent from power supply sources, both wired and battery-based. *Energy harvesting* makes devices self-powered resorting to environmental sources. In this scenario, the availability of the power source is not always guaranteed. For this reason, these devices need to be designed in order to be robust against intermittent power supply.

Some techniques have been developed in the last years to deal with this problem. In most solutions the objective is to save the state of the computation on a *Non-Volatile Memory* (NVM) at specific times. This way, whenever the power supply cannot be granted, the running process is halted and its state is saved into the NVM. This is resumed as soon as the energy is available again by moving back the content of the NVM into the CPU. This mechanism is called *context saving*.

A recently proposed technique for context saving relies on the presence of scan chains inside the CPU [2]. Scan chains are always present inside microprocessor designs for testing purposes. For the purpose of context saving, the focus is on a properly designed scan chain that traverses all the registers that must be saved. When the computation is halted due to a lack of energy, all the content of the scan chain is transferred to a dedicated NVM.

In [3], a security issue related to context saving was introduced. The need for security in many emerging IoT applications is in contrast to existing context saving techniques that expose the content of the internal registers of the CPU to an external NVM without implementing security criteria. In fact, NVMs can be observed and even tampered with by any attacker that has physical access to the device [4]. Since the IoT infrastructures are usually very wide, having physical access to even only a single device is not an unlikely assumption in most applications. This access can be a way to perform attacks that affect the whole infrastructure. In such cases, the interruption of the power supply can be maliciously triggered by the attacker in order to force the CPU to expose its content timely during computation. Reading and modifying the content of many kinds of NVMs can be successfully performed according to the implemented technology and the kind of NVM used. Content read-out can lead to steal secret keys or, more generally, to have knowledge of contents that are usually stored into secure memories when the system is regularly turned off.

In this paper, we propose the *Secure Context Saving Unit* (SCSU), which can be plugged into any SoC design and activated whenever the user wants the system to perform a secure context saving in the case of supply unavailability. Confidentiality is provided via stream cipher based encryption, while the integrity against memory tampering is provided via a *Message Authentication Code* (MAC) derived from the saved context.

The reminder of this paper is organized as follows. In Section II, we provide a background on existing context saving techniques and the related security issues. In Section III, we present the implementation of our solution. Section IV presents experimental results. Finally, we draw some conclusions in Section V.

II. BACKGROUND

A. Context Saving Techniques

Context saving techniques are divided into two categories, depending on whether the state of the processor core is saved *in-place* (InP) or moved *out-of-place* (OoP). InP techniques are based on *state retentive flip-flops*, which

*Institute of Engineering Univ. Grenoble Alpes

are able to store their state even when the power supply is turned off [5][6]. In OoP techniques, which are the scope of this paper, the content of the CPU is moved towards an external retaining memory. OoP methods are generally preferred over InP because they require less design efforts and they consume less dynamic power [2]. Generally, using off-chip memories (e.g. flash memories) is too expensive from an energetic point of view. For this reason, in-chip memories are the preferred solution for OoP context saving techniques.

OoP context saving techniques that have been proposed in the literature handle the problem at different levels of the digital stack. The power supply can be periodically monitored by specific software procedures that are generated in the compilation phase [7]. Alternatively, an interrupt mechanism can be employed, in order to trigger the context saving as soon as the supply voltage goes under a predefined threshold [2][8]. The technique presented in [8] relies on interrupt routines that transfer the context to the NVM. On the other side, the technique presented in [2] relies on a context saving procedure that is entirely managed at hardware level (i.e. the scan chain content is transferred to the NVM).

Another critical aspect for the comparison of these techniques is the kind of NVM that is used as target of the saved context. In [7], the context is saved into an external flash memory. Alternatively, the authors of [8] and [2] proposed the implementation of a *Ferromagnetic RAM* (FRAM) inside the chip, in order to host the saved context. FRAM memories have high efficiency and low power consumption during the write process. For this reason, the context saving procedure is far more energy efficient, making FRAM and similar technologies preferred for context saving.

B. Security issue

The use of NVMs based on emerging technologies, such as FRAMs, poses a strong vulnerability with respect to *physical attacks*. Some recent works have addressed possible security issues of such emerging NVMs. For instance, the authors of [9] and [10] have studied the effect of active attacks on the *Oxide-based Resistive Random Access Memory* (OxRRAM). These works showed that OxRRAM cells could be easily attacked by laser injections. More specifically, it is quite easy to change even a single bit of such memories from '1' to '0'. The physical phenomenon that allows such an attack (i.e., the localized increase of the temperature) can also be exploited in other types of memories, such as *Phase Change Memories*. Another work [11] showed how *Spin Transfer Torque Magnetic Memories* (STT-RAM) could be affected by external magnetic fields, thus allowing the control of the memory elements. In general, it has been demonstrated that physical attacks are possible for emerging NVM technologies, and they might be even easier to perform than fault attacks targeting traditional memories (e.g. flip-flops, SRAMs, DRAMs, flash memories). Traditional memory technologies, such as flash memories, generally provide integrated security features. It is not the same for emerging memory technologies. For this reason, we believe that the employment of this new kind of memories for applications such as context saving deserves the attention of security researchers.

In order to tamper with the device, the attacker can either observe the internal behavior of the device or can actively modify its state. When context saving techniques are implemented, this boils down to a physical attack on the NVM. Several techniques and means exist to inject a fault into the device in order to alter the content of the memory elements [12]. This modification could lead to an error that can be exploited to retrieve a secret key (e.g., differential fault injection as in [13]), to the modification of the device state with a possible privilege escalation or to the bypass of security checks. For instance, faults can be injected to change the flow of an executed software code. For code implementing cryptographic algorithms, changes in the program flow can be a security threat.

To summarize, recent works have shown that NVMs that are used in the next generation of IoT devices might clearly lack of security in at least two aspects: (i) their content can be read, and (ii) their content can be tampered with. Because of these vulnerabilities, it is important to protect the data stored into the NVM against physical attacks, in order not to allow privilege escalation or lack of confidentiality.

III. Secure Context Saving Unit

The solution proposed in this paper adds security features to the context saving technique proposed in [2]. The memory elements of the CPU are traversed by two different scan chains: (i) one involving the *volatile domain* (i.e. the CPU content that can be lost when the power supply is off); (ii) the other involving the *non-volatile domain* (i.e. the set of memory elements that must be preserved when the energy supply is not available). The transfer of the non-volatile domain content to the NVM and vice versa through the scan chain is managed by a *Context Saving Controller*. It performs the *Context Storing Procedure* (CSP) when the device is about to be turned off, and the *Context Loading Procedure* (CLP) when the computation is restored. A *supervisor* manages a higher level control. It receives an interruption from the energy supply when this is going below a specific threshold. After that, the supervisor sends an interruption to the CPU that runs a software routine in order to enter into a predefined sleep mode. At this point, the CSP procedure is performed at hardware level and the scan chain content is transferred to the target NVM. When the system wakes up from the sleep mode, the supervisor checks if an image to restore is present in the NVM. In the affirmative case, the CLP is started by the controller and the CPU state is restored. The *Secure Context Saving Unit* (SCSU) is serially connected between the scan chain and the target NVM. In order to cope with the security threats described in the previous section, the SCSU provides both confidentiality and integrity of context data that are going to be stored into the target NVM. Figure 1 shows the high-level architecture of the SCSU module.

In order to provide confidentiality, data are encrypted before being stored into the NVM and then decrypted when

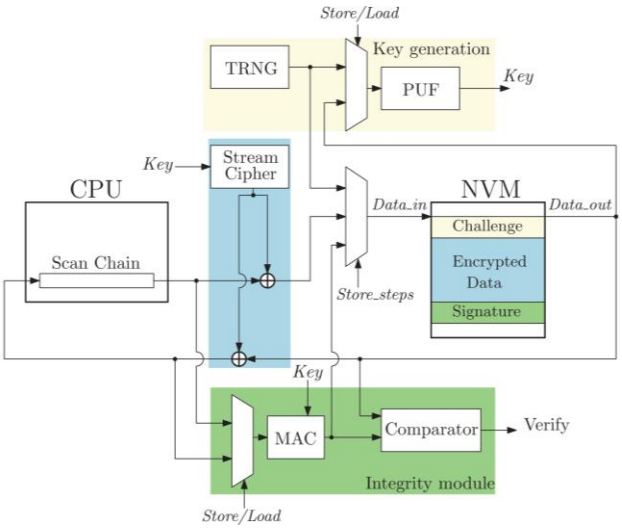

Fig. 1 Architecture of the SCSU

loaded back into the CPU. If the encryption method is secure, the attacker is not able to understand the content of the NVM and to extract any information out of it. The SCSU performs the encryption resorting to a stream cipher. The stream cipher performs a simple XOR operation between an input bitstream and a random sequence of bits, called *keystream*. The keystream is generated pseudo-randomly from a seed k and is denoted as $S(k)$. The encryption function is $E(m,k) = m \oplus S(k)$, while the decryption function is $D(c,k) = c \oplus S(k)$. If the generated keystream is unpredictable, the stream cipher will be considered secure.

The encryption does not prevent the attacker from modifying the content of the NVM. This can generate a vulnerability even if the attacker does not understand the data content. For this reason, the SCSU also performs integrity verification resorting to a MAC module. The MAC takes as input the plaintext context data and a secret key. It produces a signature at the output, which is stored into the NVM, together with the encrypted context data. When the context is loaded back to the CPU, the decrypted data are sent to the MAC module in order to compute the signature again and compare it with the one stored into the NVM. If they are equal, the content of the NVM is intact, thus the computation can start again. On the contrary, if some tampering occurred, the supervisor must reset the content of the scan chain and the computation must start from scratch and/or a security alarm should be raised.

Both the stream cipher and the MAC module need a secret key in order to securely perform their operations. Moreover, these keys, which are produced in the CSP phase, must be recovered in the CLP phase. However, there are two important aspects to take into account: *1)* using the same key to perform the encryption of data more than once is not a secure choice when dealing with a stream cipher; *2)* used keys must be saved after the CSP phase in order to be used in the CLP phase. This cannot be done simply by saving the keys into the NVM, because we assume that the attacker is able to access the NVM and read its content.

The key generation is performed by a *True Random Number Generator* (TRNG) and a *Physical Unclonable Function* (PUF). In particular, the TRNG generates a random value, representing a challenge for the PUF. The keys for the stream cipher and the MAC module are extracted by the response of the PUF to the challenge. The challenge is also stored into the NVM. The saved challenge is used during the CLP to recover the key via the PUF. If an attacker reads the content of the NVM, he/she will know the challenge, but context data cannot be decrypted nor the signature tampered with. The only way to obtain the keys is to know the behavior of the PUF. However, a PUF relies on unique physical characteristics of each device sample, making its response unpredictable. The TRNG is employed in order to generate a different challenge for each context saving session and avoid key repetition.

The controller is in charge of executing the CLP and the CSP procedures. The CSP is managed by the *Store* signal and the CLP is managed by the *Load* signal. During the CSP, the controller also selects the multiplexer driving the *Data_in* port of the NVM. The context saving controller implements a *Finite State Machine* (FSM), described in Fig. 2. The FSM is divided into two main flows, one implementing the CSP, the other implementing the CLP. By default, the controller is in IDLE state. The controller receives an input from the supervisor determining the procedure to start, the CSP or the CLP. Then, the FSM executes each step of the procedure, depending on the value of the counter *cnt*. The counter is reset after each step. At the end of the executed procedure, the controller goes back to the IDLE state and waits for new instructions from the supervisor. In the next subsections, we detail the different steps of the two procedures.

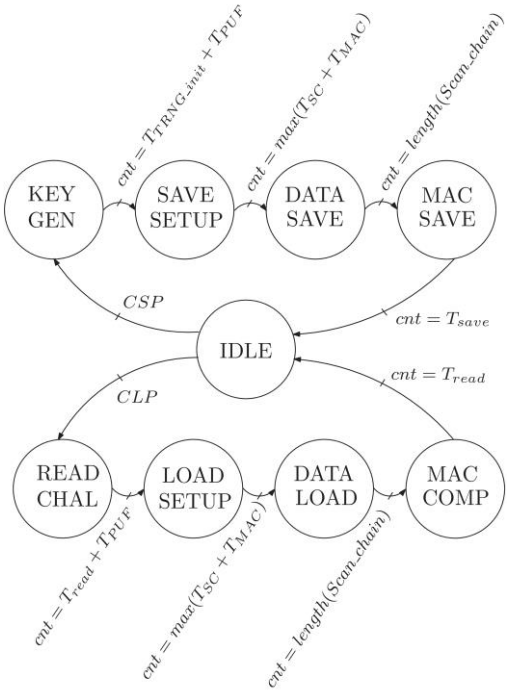

Fig. 2 FSM of the context saving controller

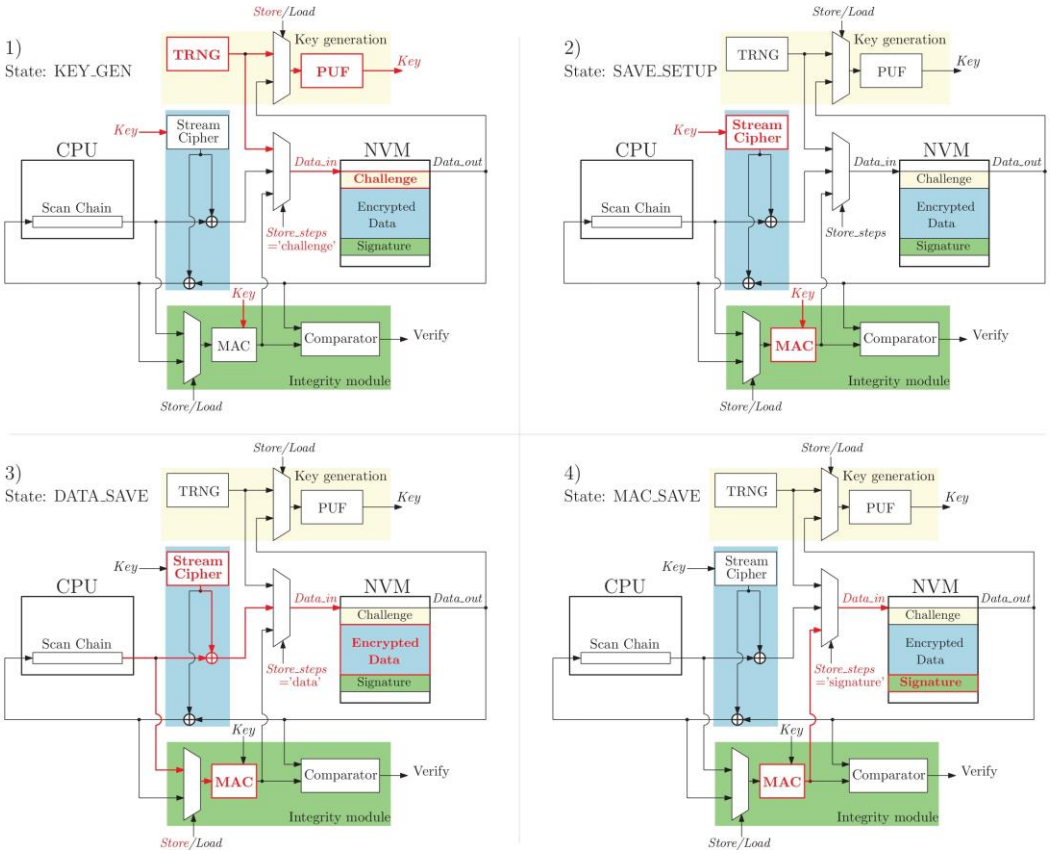

Fig. 3 Context Saving Procedure

A. *Context Saving Procedure*

As soon as the supervisor sends the request to save the context, the system enters in the CSP phase. The CSP consists in four steps, described in Fig. 3: 1) the generation of the keys for the stream cipher and the MAC; 2) the setup of the stream cipher and the MAC; 3) the scan chain content storing; 4) the signature storing.

The TRNG starts its initialization as soon as the CSP begins. Firstly, the FSM enters the KEY_GEN state, where a random number is generated. The TRNG requires an initial setup time, T_{TRNG_init}, to reach sufficient entropy in order to generate random numbers properly. Once the TRNG is initialized, the controller sets the *Store_steps* signal controlling the multiplexer at the NVM input in order to select the TRNG output. The multiplexer at the PUF input is also connected to the TRNG output via the *Store* signal. The random number generated by the TRNG serves as a challenge for the PUF. This challenge is saved into the NVM in order to be used for restoring the context data in the CLP. At the same time as the challenge is saved, the PUF generates the response.

This response is used as key for the stream cipher and the MAC. The stream cipher and MAC setups are performed in parallel in the SAVE_SETUP state. Once both modules are initialized, the context data can be saved into the NVM.

The FSM enters the DATA_SAVE state. The controller selects the datapath connecting the scan chain to the multiplexer at the NVM input. The context data contained in the scan chain are shifted out. During data shifting, the encryption is performed by the stream cipher, and the result is saved into the NVM. At the same time, the same data are processed in parallel by the MAC module.

Once the content of the whole scan chain is shifted out, the FSM enters the MAC_SAVE state. The controller connects the MAC output to the NVM input through the *Store_steps* signal. The signature computed by the MAC is saved into the NVM. At this point, the CSP is finished and the controller goes back to the IDLE state.

B. *Context Loading Procedure*

The CLP is also carried out in four steps, described in Fig. 4: 1) the recovering of the keys used during the CSP by reading out the saved challenge; 2) the initialization of the stream cipher and the MAC module; 3) the loading of the decrypted context; 4) the verification of the integrity.

When the CLP begins, the FSM enters the READ_CHAL state. The controller selects the multiplexer driving the input of the PUF through the *Load* signal, in order to retrieve the challenge from the memory. The challenge saved into the NVM is then fetched and sent to the PUF. The PUF response corresponds to the keys used by the stream cipher and the MAC during the CSP.

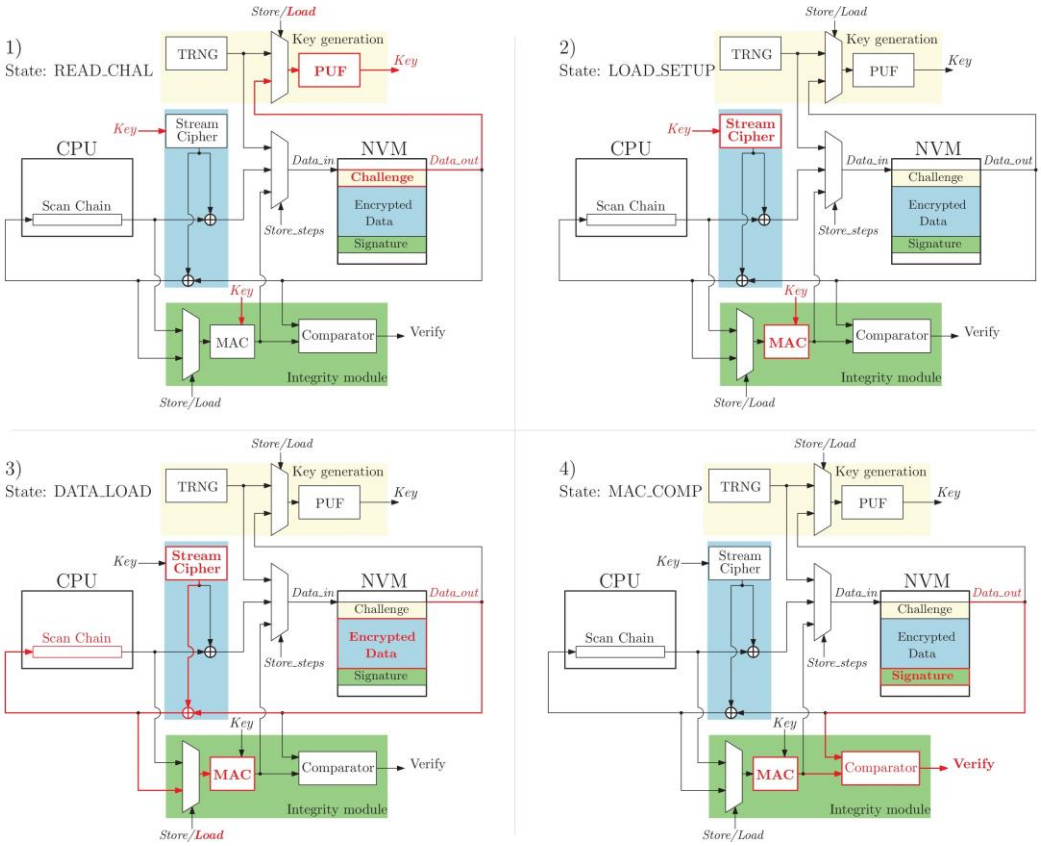

Fig. 4 Context Loading Procedure

Once the keys are generated, the stream cipher and the MAC module are initialized in the LOAD_SETUP state. When both setups have finished, the FSM goes into the DATA_LOAD state. The encrypted context data are read out and decrypted by the stream cipher, generating the same keystream used in the CSP. At the same time, the decrypted data are processed by the MAC module, while the scan chain of the CPU is loaded in parallel. However, the context of the CPU is not resumed until the end of the MAC_COMP state.

In the MAC_COMP state, the saved signature is fetched from the NVM and compared with the one produced by the MAC module. The result of the integrity control is sent to the supervisor through the *Verify* signal. After that, the CLP is finished and the FSM goes back to the IDLE state, waiting for another request from the supervisor.

IV. Results

In this Section, we provide an implementation of the SCSU, in order to evaluate its cost and performance. The evaluation of the proposed solution has been carried out measuring the area footprint, the impact on the context saving time and the energy consumption.

We have chosen to implement a TRNG from the Synopsys DesignWare IP library [14] and an Arbiter PUF for the key generation, the Trivium stream cipher for the encryption/decryption of the context data and the HMAC algorithm (based on the SHA-256 hash function) for the signature computation. Table I reports the area cost for each module composing the proposed solution. Globally, the area cost is equal to 28295 gate equivalent. The additional hardware is a serious overhead, since the application is intrinsically extremely low power. However in the IoT domain, it is very common that an attack on a node of the network can compromise the entire infrastructure and lead to serious problems. Therefore, this area overhead is the cost to pay in order to ensure a higher security for the whole infrastructure.

The proposed solution also brings an overhead on the performances. Indeed, the CSP and CLP executions are modified in order to ensure confidentiality and integrity. Consequently, they require more time to execute. The execution of the CSP is composed by the initialization of the TRNG, the PUF computation, the setup of the stream cipher

TABLE I. Area cost of SECCS module

Submodule	**Area (*gate equivalent*)**
TRNG	15000
PUF	516
HMAC	10763
Stream cipher	2016

and the MAC module (both processed in parallel), the shifting out of the whole scan chain of the CPU, and the saving of the signature. The saving of the challenge and the encrypted data are not considered in the execution time of the procedure, since this is performed in parallel to the PUF computation. The storing of the encrypted data is performed when the content of the scan chain is shifted out.

Concerning the CLP, its execution is composed of the reading of the challenge from the NVM, the PUF computation, the parallel setup of the stream cipher and the MAC, the filling of the scan chain with the decrypted context data, and the reading of the signature from the NVM. The reading out of the encrypted data is not considered in the execution time, since the data are read while the scan chain is filled.

More formally, T_{CSP} and T_{CLP} respectively denote the time to execute both procedures, T_{TRNG_init} the time to initialize the TRNG, T_{PUF} the time to obtain the response from the PUF, T_{SC} the time for the stream cipher setup, T_{MAC} the time for the HMAC module setup, T_{read} and T_{save} the time to access the NVM for reading and writing respectively and $length(Scan\_chain)$ the length of the scan chain. T_{CSP} is defined as:

$$T_{CSP} = T_{TRNG\_init} + T_{PUF} + \max(T_{SC}, T_{MAC}) + \text{length}(Scan\_chain) + T_{save} \quad (1)$$

T_{CLP} is defined as:

$$T_{CLP} = T_{read} + T_{PUF} + \max(T_{SC}, T_{MAC}) + \text{length}(Scan\_chain) + T_{read} \quad (2)$$

Concerning power consumption, previous works have proven that hardware techniques for context saving represent a smaller overhead with respect to software procedures. In [2], the energy consumption to transfer context data from the scan chains of the CPU to the target NVM has been measured being 41pJ/b on a 65nm standard library. The security module proposed in this paper produces an additional energy consumption of 3.64pJ/b for the stream cipher encryption and 16pJ/b for the MAC signature generation on a similar technology library. This represents an overhead of 48% on the energy required to store the context into the NVM. This is an acceptable overhead when dealing with the security of the whole IoT infrastructure.

V. Conclusion

This paper proposes a solution for processor-based IoT devices requiring security. We consider a system performing context saving to external NVMs, activated when the available energy is not sufficient to continue the execution of the operations. At the same time, we want to prevent attacks on the saved state of the device, including tampering and fault attacks. For this reason, we proposed a Secure Context Saving Unit that provides a hardware module easy to implement inside a System on Chip (SoC). This module provides both confidentiality and integrity to all the CPU content that is saved into the target NVM. We have shown that the proposed solution can have a large impact on the area overhead of the system. However, the need for security cannot be ignored in many applications. For this reason, we believe that scarifying a part of the power budget can be necessary for higher security.

References

[1] A. Mosenia and N. K. Jha, "A Comprehensive Study of Security of Internet-of-Things," in *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 586-602, Oct.-Dec. 1 2017.

[2] P. A. Hager, H. Fatemi, J. P. de Gyvez and L. Benini, "A scan-chain based state retention methodology for IoT processors operating on intermittent energy," *Design, Automation & Test in Europe Conference & Exhibition* (DATE), 2017, Lausanne, 2017, pp. 1171-1176.

[3] E. Valea, M. Da Silva, G. Di Natale, M. Flottes, S. Dupuis, B. Rouzeyre, "SECCS : SECure Context Saving for IoT Devices," 2018 *13th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era* (DTIS), Taormina, 2018.

[4] K. Shamsi and Y. Jin, "Security of emerging non-volatile memories: Attacks and defenses," 2016 *IEEE 34th VLSI Test Symposium* (VTS), Las Vegas, NV, 2016, pp. 1-4.

[5] Z. Liu and V. Kursun, "New MTCMOS Flip-Flops with Simple Control Circuitry and Low Leakage Data Retention Capability," 2007 *14th IEEE International Conference on Electronics, Circuits and Systems*, Marrakech, 2007, pp. 1276-1279.

[6] Portal, J.M. & Bocquet, M & Moreau, Mathieu & Aziza, Hassen & Deleruyelle, Damien & Zhang, Yue & Kang, Wang & Klein, Jacques-Olivier & Zhang, Youguang & Chappert, Claude & ZHAO, Weisheng. (2014). "An Overview of Non-volatile Flip-Flops Based on Emerging Memory Technologies". *Journal of Electronics Science and Technology*. 12. 173-181.

[7] Ransford, Benjamin & Sorber, Jacob & Fu, Kevin. (2011). Mementos: System Support for Long-Running Computation on RFID-Scale Devices. Sigplan Notices - SIGPLAN. 47. 159-170. 10.1145/2248487.1950386.

[8] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli and L. Benini, "Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems," in *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 15-18, March 2015.

[9] A. Krakovinsky *et al*., "Impact of a laser pulse on HfO2-based RRAM cells reliability and integrity," *2016 International Conference on Microelectronic Test Structures (ICMTS)*, Yokohama, 2016, pp. 152-156.

[10] A. Krakovinsky, M. Bocquet, R. Wacquez, J. Coignus and J. Portal, "Thermal laser attack and high temperature heating on HfO2-based OxRAM cells," *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, Thessaloniki, 2017, pp. 85-89.

[11] S. Ghosh, "Spintronics and Security: Prospects, Vulnerabilities, Attack Models, and Preventions," in *Proceedings of the IEEE*, vol. 104, no. 10, pp. 1864-1893, Oct. 2016.

[12] R. Piscitelli, S. Bhasin and F. Regazzoni, "Fault attacks, injection techniques and tools for simulation," *2015 10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, Naples, 2015, pp. 1-6.

[13] A. Barenghi, L. Breveglieri, I. Koren and D. Naccache, "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures," in *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056-3076, Nov. 2012.

[14] Synopsys. (2015). DesignWare True Random Number Generator Core.