EasyChair Preprint

№ 813

SIA: Secure Intermittent Architecture for Off-the-Shelf Resource-Constrained Microcontrollers

Daniel Dinu, Archanaa Santhana Krishnan and Patrick Schaumont

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

March 5, 2019

SIA: Secure Intermittent Architecture for Off-the-Shelf Resource-Constrained Microcontrollers

Daniel Dinu
Virginia Tech
Blacksburg, VA, USA
ddinu@vt.edu

Archanaa S. Khrishnan
Virginia Tech
Blacksburg, VA, USA
archanaa@vt.edu

Patrick Schaumont
Virginia Tech
Blacksburg, VA, USA
schaum@vt.edu

***Abstract*—Recent advancements in energy-harvesting techniques provide an alternative to batteries for resource-constrained IoT devices and lead to a new computing paradigm, the intermittent computing model. In this model, a software module continues its execution from where it left off when an energy shortage occurred. Enforcing security of an intermittent software module is challenging because its power-off state has to be protected from a malicious adversary in addition to its power-on state, while the security mechanisms put in place must have a low overhead on the performance, resource consumption, and cost of a device.**

In this paper, we propose SIA (Secure Intermittent Architecture), a security architecture for resource-constrained IoT devices. SIA leverages low-cost security features available in commercial off-the-shelf microcontrollers to protect both the power-on and power-off state of an intermittent software module. Therefore, SIA enables a host of secure intermittent computing applications such as self-attestation, remote attestation, and secure communication. Moreover, our architecture provides confidentiality and integrity guarantees to an intermittent computing module at no cost compared to previous approaches in the literature that impose significant overheads. The salient characteristic of SIA is that it does not require any hardware modifications, and hence, it can be directly applied to existing IoT devices.

We implemented and evaluated SIA on a resource-constrained IoT device based on an MSP430 processor. Besides being secure, SIA is simple and efficient. We confirm the feasibility of SIA for resource-constrained IoT devices with experimental results of several intermittent computing applications. Our prototype implementation outperforms by two to three orders of magnitude the secure intermittent computing solution of Suslowicz *et al.* presented at IGSC 2018.

***Index Terms*—secure intermittent architecture, resource-constrained microcontroller, energy harvesting, intermittent computing.**

I. Introduction

The brisk development of the Internet of Things (IoT) brings computing systems in all aspects of our lives, from home and building automation to the clothes we wear, or even life-critical implantable medical devices. Many of these IoT devices, commonly referred to as *smart devices* or *smart things*, are expected to operate autonomously for a long period of time after their deployment. However, some characteristics of batteries, such as their relatively large size, heavy weight, limited lifetime, and the need for a replacement, stay in the way of this goal [18]. Yet, recent advancements in energy-harvesting technologies provide a viable alternative to traditional batteries. Energy-harvesting techniques collect energy from the environment and store it in a supercapacitor. Although energy harvesters exploit a virtually inexhaustible resource, environmental conditions influence the resource availability (e.g., clouds block sun rays from reaching a solar panel). Moreover, the amount of energy that can be stored is limited. Therefore, IoT devices powered by harvested energy have to work in a new execution model, in which every fraction of the available energy has to be judiciously used to ensure forward progress of computational tasks across unpredictable power failures. *Intermittent computing* provides a means to resume execution of a task after a power loss without redoing the entire computation, but only that part whose result was not saved in the non-volatile memory prior to the power loss.

System security is among the technical challenges that stem from the absence of a continuous power source in the *intermittent computing model*, especially because the attack surface of devices that operate on harvested energy increases compared to the attack surface of battery-powered devices. Attack vectors specific to intermittent systems exploit the transition of a system from task execution to power-off state and in particular the memory and register content when the system is not powered. In this way, an attacker can affect the confidentiality, integrity, and availability of processed data. Hence, intermittent systems must leverage security mechanisms to protect against these new types of attacks in addition to the attacks that threaten the security of a battery-powered device. Moreover, besides being effective, security mechanisms should have a low overhead in terms of area, memory, speed, and energy. With the market of energy-harvesting devices estimated to grow to $2.6 billion by 2024 [34], the security of intermittent computing devices is of paramount concern.

Two key characteristics of IoT devices are network connectivity and software extensibility [30], [31]. These two factors enable a large spectrum of applications and at the same time create opportunities for remote software attackers. Mirai and other IoT botnets are eloquent examples of malware that successfully infected low-end devices [2], [23], [26]. The lack of advanced memory management units and privilege levels is a hindrance to the security of resource-constrained embedded

devices against remote software attacks [30], [31]. To solve this security issue, researchers from academia and industry proposed various architectures tailored to the security goals of IoT devices. In particular, *trusted computing architectures* provide a certain level of confidence to users, owners, or software vendors that the software executed on a device is genuine and behaves as expected. The core idea of many security architectures is to enforce some form of *software isolation*. Depending on how software isolation is enforced, a trusted computing architecture can be based on hardware, software, or hardware/software codesign [28]. A security architecture may provide local or remote *attestation*. Attestation is a crucial security service that allows an authorized party (verifier) to check if a software module is in a specific state. The verifier sends a challenge to the prover and then checks the authenticity of the attestation report received from the prover. Typically, the attestation report consists of a cryptographic signature or a message authentication code (MAC) over the challenge and a measurement (e.g., a hash) of the binary code to be attested.

The predicted growth of IoT devices powered by harvested energy [13], [34] brings new security challenges, many of which are at the intersection of intermittent computing and security architectures. On the one hand, with a few exceptions, the large body of literature on intermittent computing for low-end devices is mainly focused on reliability and efficiency of intermittent computing techniques and ignores security issues specific to this computing model. On the other hand, to the best of our knowledge, there is no architecture tailored to the security requirements of resource-constrained IoT devices powered by harvested energy in the wide range of security architectures proposed in the literature. More precisely, the security guarantees of existing solutions do not hold in the intermittent computing model, mainly because they can not be enforced when there is no power. Therefore, the power-off state of a device is vulnerable to attacks that modify or replace its content [27].

A. Research Contributions

In this paper, we address the above-mentioned security challenges at the intersection of intermittent computing and security architectures. To this end, we propose *SIA (Secure Intermittent Architecture)*, a security architecture for resource-constrained IoT devices. SIA makes use of low-cost security features available in resource-constrained microcontrollers to enable secure intermittent execution of virtually any application. Our secure intermittent architecture is the first of its kind and is very useful for devices powered by harvested energy. Potential applications include sensor nodes for monitoring and detection of building and bridge stresses, air pollution, forest fires, pending landslides, worn bearings, and wing vibration [1], [13], [25]. In addition to the secure intermittent computing framework which can be applied to any software module, SIA provides three security services, namely self-attestation, remote attestation, and secure communication. A distinguishing feature of SIA is that it can be directly applied to off-the-shelf microcontrollers because it does not require any hardware modifications. Moreover, SIA can be applied even to IoT devices already deployed in remote locations via a secure over-the-air update.

Our research contributions can be summarized as follows:

- We propose SIA, a secure intermittent architecture that guarantees the security of both power-on and power-off states of a software module executed on a resource-constrained microcontroller. In addition to this, SIA provides three security services: self-attestation, remote attestation, and secure communication.
- We implement SIA on an IoT device based on an MSP430 microcontroller which is widely-used for ultra-low-power applications. Our implementation does not require hardware modifications, nor a custom toolchain.
- We evaluate our implementation of SIA by using several intermittent computing applications. Experimental results show that our secure intermittent architecture outperforms by two to three orders of magnitude the secure intermittent computing solution of Suslowicz *et al.* [38].

B. Outline

In Section II we formulate the problem addressed in this paper, then we describe the attacker model and the desired security properties. We present the design of our secure intermittent architecture in Section III, our prototype implementation in Section IV, and our experimental results in Section V. In Section VI we discuss related work. Finally, we conclude the paper in Section VII.

II. Problem Statement

A. Intermittent Computing Systems

In this work, we consider a resource-constrained IoT device that is powered by harvested energy and that operates in the intermittent computing model. For example, we can imagine a sensor node deployed at a distant location from its owner to gather some sensory information, do some processing on acquired data, and transmit updates to a gateway. The next generation of smart-metering sensors to be deployed by utility companies are a real-world example of low-end intermittent computing devices. The metering system mounted on a pipe to measure the water consumption of a household can be powered by the energy harvested from the fluid flow [8], [19], [24]. Similarly, a gas metering sensor can be powered by the energy harvested from the gas flow in a pipe [39]. Smart metering is just one domain in which low-end intermittent computing devices powered by harvested energy are a viable replacement of battery-powered devices. For many other application examples, we refer the reader to [1], [13], [25].

We assume an intermittent computing system that has network connectivity and that may support software updates from one or more software providers. It can be any low-end device that has a resource-constrained embedded processor which does not feature an advanced memory management unit, hypervisor, or other advanced memory protection mechanism found in more powerful processors.

In our settings, a software module executed by an intermittent computing system must be protected from unauthorized

access and modification by other modules. These requirements apply to both power-on and power-off state of an intermittent computing system. Moreover, the provider of a software module should be able to detect any unauthorized modification of its module.

In this paper, we design, implement and evaluate a secure intermittent architecture that provides a solution to the above-mentioned security requirements. We aim for a secure intermittent architecture that is easy to implement and deploy and has a low impact on resource utilization.

B. Attacker Model

We assume a strong attacker that has the following two capabilities. First, the attacker can sniff and modify the network traffic between the intermittent computing system and software providers. Essentially, the attacker has control of the communication network. This type of attacker is referred to as an input/output attacker in the model of Piessens and Verbauwhede [33].

Second, the attacker can manipulate all the software modules deployed to an intermittent computing system. More concretely, the attacker can modify or replace an existing software module, or even install new modules. For example, the attacker can modify the data stored by a metering system [40]. These capabilities are attributed to the code attacker in the model of Piessens and Verbauwhede [33].

Compared to [33], Schaumont and Montuschi [35] consider a third attacker, a hardware attacker. We assume that the attacker has no physical access to the hardware. Consequently, the attacker can not place probes on the memory bus, inject faults, measure the energy consumption of the microprocessor, or remove components of the system. Hardware attacks are out of scope for this work because there is a distinct research area that focuses on the design and implementation of protection mechanisms against this type of attacks. However, the application of protections against hardware attacks is strongly encouraged since it does not affect our security architecture but improves its overall security.

We adopt the Dolev-Yao attacker model [12] for the cryptographic algorithms used by the intermittent computing system. More precisely, the attacker can perform protocol-level attacks but can not break the cryptographic algorithms.

This attacker model is very realistic since it encompasses all the major threats to the security of an intermittent system. Similar assumptions on the attacker capabilities were made by the designers of Sancus [30], [31].

C. Security Properties

Based on the characteristics of the intermittent system and the attacker model, we formulate a set of security properties that our secure intermittent architecture provides:

- *Secure intermittent computing.* A software module is protected from unauthorized read or write during both the power-on and power-off state of the intermittent computing system. In other words, the software module is isolated. This property holds even if the software provider does not enable intermittent execution of the software module.
- *Remote intermittent attestation.* A software provider can verify that a specific module loaded by the intermittent system is exactly the same to the one deployed. The whole process is secure and can be executed across periods of power loss. *Intermittent self-attestation* is an extension of this property in which the software module can check whether it was correctly loaded and its content is genuine.
- *Secure communication.* The communication channel between a software provider and a deployed software module is protected by a cryptographic algorithm to ensure confidentiality, integrity, authenticity, and freshness of exchanged data.
- *Hardware breach confinement.* In the unlikely event that an attacker breaks the hardware security mechanisms of a device, all other intermittent systems running the same software module are not affected.

III. Design of SIA

A. Overview

Our aim is to design a simple and efficient secure intermittent architecture that provides the desired security properties and can be easily applied to off-the-shelf microcontrollers. The main design challenge is to build the security architecture using only the security features available in low-end devices and no hardware modifications at all. The reward for this effort is a solution that can be immediately applied to existing IoT devices, even to those already deployed. Furthermore, our architecture does not require a custom toolchain to build software modules. Considering the resource constraints of intermittent computing systems, we decided to use a lightweight symmetric cryptographic engine to support remote attestation and secure communication.

Initially, a software provider deploys a software module to an intermittent system. Then, the provider communicates with the intermittent system to get some data from the node or to instruct it to perform certain tasks. Since the communication channel is not secure, the message exchange has to be encrypted and authenticated. Once in a while, the software provider may verify the integrity of the software module or update it.

Software modules are binary files which consist of several sections, each section storing a different part of the module. Three important sections are `.text`, `.rodata`, and `.data`. They contain the machine code executed by the microprocessor, constants used by the code, and global and static variables, respectively.

B. Software Module Protection

Software modules deployed to an intermittent system must be protected from interacting with each other in undesired ways. This can be enforced using a memory protection mechanism that restricts read and write access to a software

module. Our security architecture leverages the basic memory protection features available in many low-cost microcontrollers. This type of feature is available in a wide range of low-end microcontrollers under different names. The 8-bit PIC16F184xx family of microcontrollers for sensor network applications from Microchip features a Memory Access Partition (MAP) [29]. Some 16-bit MSP430 microprocessors from Texas Instruments have a Memory Protection Unit (MPU) [20] and support Intellectual Property Encapsulation (IPE) [32]. The 32-bit ARM Cortex-M3 includes an optional memory protection unit [3]. All these features are a form of program-counter based memory access control [37] and can be used to isolate software modules.

Typically, memory isolation is enabled by writing some registers. The code, constant, and data sections of a module are placed in special sections inside the protected memory region. We use `.sia_text`, `.sia_const`, and `.sia_data` to denote the three protected sections of a software module. Once the memory isolation is activated, read and write operations inside the protected memory region are not possible from outside the region. Hence, the software module is protected when the device is powered on as well as when the device is powered off.

Following a defense-in-depth approach, all debug and test interfaces of a device should be disabled. If all these interfaces to the device are locked, then the software module has an additional protection layer and one more guarantee that the memory isolation can not be reverted.

C. Secure Intermittent Computing

A checkpoint is a snapshot of an intermediate state of an intermittent software module which is used to resume the execution of the module after a power loss. Typically, a checkpoint includes the content of the general purpose registers, the execution stack, global variables, and the configuration of peripherals. Checkpoints are periodically created during the execution of the module and stored in non-volatile memory. After a power loss, the most recent checkpoint is restored and the execution of the software module continues from that point. This is useful for long running tasks or in settings where the power-on cycles are not long enough to support completion of a task.

Our architecture provides secure intermittent computing if checkpoints of a module are stored inside the protected memory region of that module. The code, constant, and data sections of the checkpointing routines have to be placed in the protected memory region. If these requirements are fulfilled, the module isolation provides confidentiality and integrity guarantees to the created checkpoints.

D. Cryptographic Primitives

To provide remote attestation and secure communication, our architecture requires three cryptographic primitives, namely a key derivation function, a message authentication code (MAC), and an encryption algorithm. All these cryptographic operations can also be provided by a single primitive, authenticated encryption with associated data (AEAD).

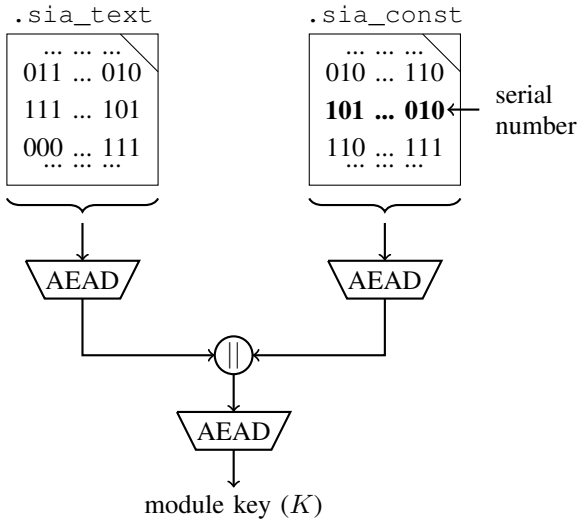

Fig. 1. Generation of the module key.

The key derivation function is used to generate a *module key*, a key that is unique for each software module and each intermittent computing device. This key is derived from the contents of the `.sia_text` and `.sia_const` sections of the protected module as shown in Fig. 1. To ensure uniqueness of the derived key, the software provider must set a different *serial number* for each module and device that executes the module. A simple way to achieve this is to define and instantiate a constant stream of bytes in the `.sia_const` section of the protected module. Thanks to the way the key derivation is designed, any software update determines a change of the module key. This key rotation mechanism should be activated by a software update whenever the software provider suspects a potential leak or compromise of the module key. Software provider and its deployed module compute the module key using the same algorithm. However, the protected software module may execute this operation in an intermittent fashion. Finally, the module key is used as the secret key of the MAC and encryption algorithms.

Our secure intermittent architecture uses a message authentication code to compute a tag over some data that has to be authenticated with the module key. First, the software module uses a MAC to authenticate a challenge sent by its provider. The software provider verifies the response to this challenge using the same cryptographic primitive and the module key. Second, all messages exchanged between a module and its provider are authenticated to protect their integrity and ensure their authenticity.

An encryption algorithm is required to protect the confidentiality of the communication between the software provider and its module. Messages are encrypted and decrypted under the same symmetric key.

Authenticated encryption with associated data (AEAD) is a cryptographic primitive that combines the above-mentioned cryptographic functions into a single algorithm. Typically, AEAD schemes are more efficient than a set of algorithms that

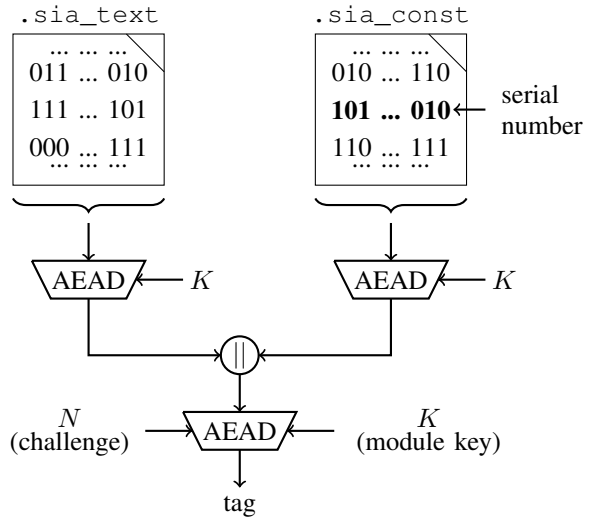

Fig. 2. Generation of the slow attestation tag.

together achieve the same cryptographic properties. Hence, lightweight AEAD algorithms are appropriate for low-end intermittent computing devices. However, if confidentiality is not desired, one can replace the AEAD primitive with a hash-based message authentication code. Some hash functions have efficient implementations for small microcontrollers [11].

E. Remote Attestation

Our security architecture gives a software provider two ways to authenticate its software modules. Depending on the time required to compute the attestation report, we distinguish between *slow attestation* and *fast attestation*.

For slow attestation, the prover computes an authentication tag over a challenge and the entire memory of a module, except for the data section, as shown in Fig. 2. This operation is resource-intensive and therefore it should have support for intermittent execution. The software module protection guarantees that tag generation can be executed intermittently without any security risk.

The computation of the fast attestation report is very efficient compared to the generation of the slow attestation tag because the prover authenticates only the challenge received from the software provider.

Both slow and fast attestation use the module key to authenticate the attestation report. One difference between the two is that slow attestation authenticates the `.sia_text` and `.sia_const` sections of the module in addition to the received challenge. Consequently, slow attestation requires more computational resources than fast attestation.

The security of the attestation stems from the fact that no other party, except for the software module and its vendor, can get the module key because it is protected during the power-on and power-off state of the intermittent computing system. An attacker can not read the module from the memory and can not modify it. Moreover, any modifications made by an input/output attacker that may exploit a software vulnerability of the module itself (e.g., a buffer overflow attack) can be detected by a slow attestation request.

We recommend a combination of both slow and fast attestation to exploit the trade-off between trustworthiness of the attestation report and resource consumption. For example, the software provider may request a slow attestation report once a week and fast attestation reports twice a day.

Self-attestation is similar to key generation or slow attestation without a key and a challenge. The software module recomputes its key and verifies if the computed value matches the module key stored in its protected memory. This should not be confused with the module key generation because the computed value does not replace the stored module key. Key generation is executed only after the module receives a software update from its provider.

Slow attestation, fast attestation, and self-attestation are secure in the intermittent computing model thanks to the memory isolation enforced by our security architecture.

F. Secure Communication

Our secure intermittent architecture provides confidentiality, integrity, authenticity, and freshness guarantees to the data exchanged between the software provider and its software module. All these guarantees can be realized with the help of an authenticated encryption scheme.

Each message includes a nonce for freshness. The role of the nonce is to protect the communication against replay attacks. The authenticated encryption algorithm encrypts the message to protect its confidentiality and generates an authentication tag. The authentication tag is used by the receiver to verify the integrity and authenticity of the encrypted message.

An important security requirement of many authenticated encryption algorithms is that the same nonce and key should not be used more than once because this leads to weaknesses in the cryptographic algorithm that can be easily exploited. For example, if two different messages are encrypted under the same key and nonce using AES-GCM [41], then the XOR of their plaintexts is leaked. Nonce-misuse resistant algorithms do not suffer from this weakness. Our security architecture assumes that authenticated encryption algorithms do not have nonce-misuse resistance. Therefore, a software provider should not repeat a nonce under the same key for secure communication. A simple solution to this problem is to use a counter value.

IV. Implementation

We implemented our secure intermittent architecture on an MSP430FR5994 LaunchPad development kit [21], a low-end device from Texas Instruments. This board is equipped with a 16-bit MSP430 microcontroller which is widely-used for ultra-low-power applications. The memory of MSP430FR5994 comprises 256 kB of non-volatile FRAM (Ferroelectric Random Access Memory) which makes this microcontroller suitable for intermittent computing applications thanks to its high speed write access, high endurance, and low-power consumption [21]. The memory space also includes 8 kB of on-chip SRAM (Static Random Access Memory).

Typically, a software module designed to operate in an intermittent computing model allocates its code, constants, global and static variables in FRAM. The module stack is placed in SRAM because it is frequently accessed by the microprocessor. This configuration ensures maximum application performance [16].

SIA consists of several building blocks. First, we describe how to enable memory isolation of a software module using the standard security features of the target microcontroller. Second, we introduce the utility that enables intermittent execution of a software module. Third, we present the four cryptographic engines implemented to support self-attestation, remote attestation, and secure communication. Finally, we briefly describe the communication protocol used between a software provider and a software module. We tested our implementation of SIA with the help of this communication protocol.

A. Software Module Protection

1) Intellectual Property Encapsulation (IPE): When enabled, this feature can be used to protect critical pieces of code, data, and constants (e.g., secret keys) allocated in FRAM from being accessed [16]. Only the code inside the IPE region can access the data variables and constants stored in the protected memory region [32]. In other words, the content of the IPE region can not be read or written by another software module. Moreover, direct memory access (DMA), the JTAG interface, or the bootstrap loader (BSL) can not view or write the protected memory region [16]. Hence, IPE enforces memory isolation of a software module. The only way to remove IPE is with a special mass erase command [32].

In our implementation of SIA, IPE isolates a protected module. This includes all functions from the `.sia_text` section, all data variables from the `.sia_data` section, and all constants from `.sia_const` section. Only a few functions from a protected module are callable from outside the module. Before returning from those functions, SIA clears the content of the general-purpose registers, hardware module registers, and RAM to prevent leakage of sensitive information [32]. Interrupts are disabled during the execution of the protected module. To set IPE, we wrote a signature at address `0xff88` that points to a configuration structure (allocated inside the protected module) which specifies the memory address boundaries of the module.

2) Disabling the JTAG Interface: It is a good security practice to disable all the test and debug interfaces to a device that is deployed in the field. An intermittent computing device which uses SIA should disable the JTAG interface to add an additional layer of security to the system. For our microcontroller, the JTAG/SBW interface can be locked by blowing an electronic fuse. This requires setting a signature at address `0xff80`. The bootstrap loader (BSL) can unlock the JTAG/SBW. However, BSL is password protected and a single incorrect password triggers a mass erase of the device [32].

B. Secure Intermittent Computing

The Compute Through Power Loss (CTPL) utility from Texas Instruments [22] allows a software module to create and restore checkpoints. A checkpoint includes the context of the software module and device peripheral state. By default, a checkpoint is saved in FRAM when power loss is detected and restored upon wakeup. In this way, a software module resumes its execution from where it left off.

We modified the standard CTPL API to include a function that creates a checkpoint when called. This allows us to choose where to place checkpoints and facilitates testing of our implementation. The CTPL code, data, and constants are part of SIA. Moreover, checkpoints are created inside the FRAM memory of the protected module. Therefore, SIA supports secure intermittent computing of any software module.

TABLE I
MAIN CHARACTERISTICS OF THE CRYPTOGRAPHIC PRIMITIVES USED FOR SIA'S CRYPTOGRAPHIC ENGINE.

Cryptographic primitive	State size (B)	Block size (B)	Key size (B)	Nonce size (B)	Tag size (B)
AES-EAX	16	16	16	16	16
KETJE JR	25	2	12	10	12
KETJE SR	50	4	16	16	16
NORX	64	48	16	16	16

C. Cryptographic Engines

We implemented four different cryptographic engines for SIA. All of them use an authenticated encryption with associated data (AEAD) primitive to provide the cryptographic services required by SIA for self-attestation, remote attestation, and secure communication.

We selected three lightweight AEAD schemes submitted to the CAESAR competition [10] that use simple operations and have efficient software implementations. The three cryptographic primitives are KETJE JR [6], KETJE SR [6], and NORX [4]. We ported the reference implementations provided by the designers of these CAESAR candidates. We also used the AES-EAX [5] implementation from the Cifra library [7] because it leverages the AES hardware accelerator of our device. A summary of the main characteristics of these four cryptographic primitives is given in Table I.

The cryptographic engines are part of SIA. Their functions, data, and constants are part of the `.sia_text`, `.sia_data`, and `.sia_const` section, respectively. We placed a checkpoint at the core of each cryptographic algorithm; it is created when a specific amount of data has been processed.

D. Communication Protocol

We implemented a simple communication protocol over UART (Universal Asynchronous Receiver/Transmitter) to test our implementation of SIA, which includes all the abovementioned building blocks. The communication protocol is part of the protected module and is protected by SIA.

The protocol works as follows. The software provider sends a request (e.g., remote attestation with a given challenge) to the protected module, which processes the request and then sends a response to its provider. The provider has the binary content of the module and performs the same computation the protected module does. Hence, the software provider can

check if the response received from the protected module (e.g., attestation report) matches the expected value.

V. Evaluation

Our secure intermittent architecture is designed for low-end devices powered by harvested energy. Therefore, it is important to verify that our implementation of SIA meets the resource constraints of an intermittent computing system. We consider three relevant metrics for our evaluation, namely binary size (the `.sia_text` and `.sia_const` sections), execution time, and energy consumption. We conduct our analysis for each of the four cryptographic engines (i.e., authenticated encryption with associated data algorithms) that support SIA. The four AEAD algorithms are AES-EAX, KETJE JR, KETJE SR, and NORX. AES-EAX employs the AES hardware accelerator of the MSP430FR5994 LaunchPad development board used in our evaluation. The other cryptographic algorithms are implemented only in software.

To measure the binary size of various sections we used the `msp430-elf-objcopy` utility included in the MSP430 toolchain. Time and energy measurements were taken with a Tektronix DPO3034 oscilloscope. We used a GPIO trigger to identify the execution of relevant operations. The execution time of an operation is given by the time difference between the two edges of the trigger signal. Energy measurements are obtained by integrating the voltage measured across a 1 kΩ shunt resistor inserted in the power line of the evaluation board. A regular power supply provided the 3.5 V supply voltage. The MSP430 microcontroller was clocked at 8 MHz.

A. SIA

First, we analyze the binary size of the entire security architecture with and without support for intermittent computing. The binary size of SIA for each of the four cryptographic engines is given in Table II.

Our implementation of SIA based on the AES-EAX engine has the lowest binary size thanks to the hardware implementation of AES which has no influence on this metric. At the other extreme, the implementation of SIA that uses NORX for the cryptographic operations has the biggest binary size. The ratio between the binary size of the NORX-based SIA and AES-EAX based SIA is 2.6. The binary size of the implementations based on KETJE JR and KETJE SR is a little bit higher than the binary size of the security architecture based on AES-EAX, but significantly lower than that of NORX-based SIA.

The overhead of checkpointing the execution of SIA is negligible, between 0.22% and 1.35% of the binary size of a non-intermittent version of SIA. The smallest and largest overhead corresponds to the implementation of SIA based on AES-EAX and NORX, respectively. The overhead of checkpointing KETJE JR and KETJE SR is roughly three times higher than the overhead of checkpointing AES-EAX and almost half of the penalty of adding checkpoints to NORX.

All implementations of SIA fill less than 15% of the memory space of our microcontroller, leaving enough memory for the actual code of the software module. The smallest implementation of our security architecture, SIA using AES-EAX, fills only 5.55% of the memory space.

TABLE II
OVERHEAD OF CHECKPOINTING ON SIA'S BINARY SIZE (`.sia_text` AND `.sia_const` SECTIONS) FOR EACH CRYPTOGRAPHIC ENGINE.

Crypto engine	Binary size (B)		Overhead (%)
	no checkpointing	with checkpointing	
AES-EAX	14,520	14,552	0.22
KETJE JR	18,628	18,740	0.60
KETJE SR	18,388	18,500	0.60
NORX	37,680	38,188	1.35

TABLE III
TIME AND ENERGY REQUIRED TO EXECUTE DIFFERENT INTERMITTENT OPERATIONS, NAMELY SLOW ATTESTATION, FAST ATTESTATION, AND SECURE COMMUNICATION.

Crypto engine	Slow Attestation		Fast Attestation		Secure comm.	
	t (ms)	E (μJ)	t (ms)	E (μJ)	t (ms)	E (μJ)
AES-EAX	151	330	6.240	9.687	6.844	12.500
KETJE JR	1,786	4,489	7.208	14.062	12.448	23.437
KETJE SR	1,196	3,097	9.196	18.750	13.968	29.687
NORX	779	1,799	6.428	12.500	8.800	17.343

Second, we present in Table III the execution time and energy consumption required for a slow attestation, a fast attestation, and a secure message exchange. These operations are executed in an intermittent fashion (i.e., with checkpointing). The protected module includes only the secure intermittent architecture and its features.

Almost all operations are executed in a fraction of a second. Slow attestation requires more time and energy compared to fast attestation because it computes an authentication tag over a challenge, the `.sia_text` and `.sia_const` sections of the module. Hence, the execution time of slow attestation depends on the binary size of the protected module given in Table II. For all engines, fast attestation has a constant execution time and energy consumption because it authenticates only a fixed-length challenge. Secure communication has a variable execution time and energy consumption depending on the request and response size. For our evaluation, we set the size of the two messages to the block size of the AEAD algorithm.

For all operations considered, the implementation of SIA based on AES-EAX has the fastest execution time and the lowest energy consumption. The NORX-based implementation of SIA is the forerunner for all operations and for both execution time and energy consumption. On the third place, KETJE JR and KETJE SR have a similar execution time and energy consumption. KETJE JR is more efficient than KETJE SR when a single block of data is processed, i.e., fast attestation and secure communication. However, for large amounts of data, i.e., slow attestation, KETJE SR is superior to KETJE JR.

If we aggregate the results from Table II and Table III, we infer that the implementation of SIA based on AES-EAX is the most suitable for our evaluation device. The implementation of SIA based on NORX is the second-best option thanks to its low execution time and energy consumption, despite its large binary size.

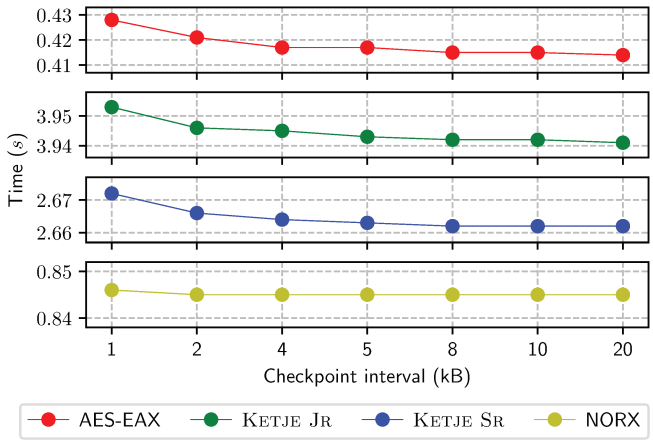

Fig. 3. Time required to attest 40 KB for different checkpoint intervals.

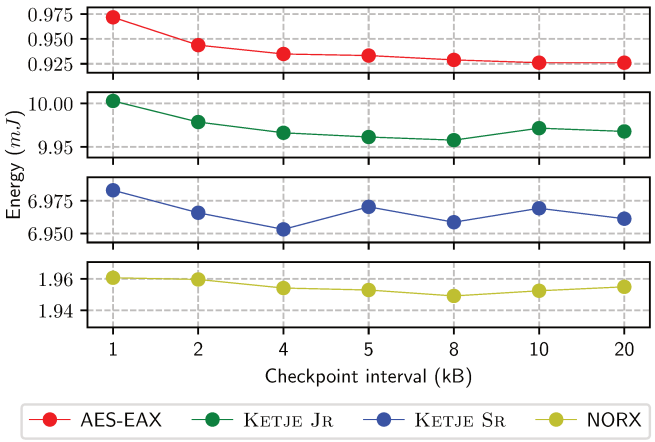

Fig. 4. Energy required to attest 40 KB for different checkpoint intervals.

B. Remote Attestation

We analyze the time (Fig. 3) and energy (Fig. 4) required to create an attestation report for a secure intermittent module of 40 kB using the slow attestation operation and different checkpoint intervals. By a checkpoint interval, we understand the amount of data processed by the cryptographic engine before a new checkpoint is created.

We can see in Fig. 3 that the execution time improves for all cryptographic engines when the checkpoint interval is increased from 1 kB to 2 kB and from 2 kB to 4 kB. Placing checkpoints at intervals larger than 4 kB does not influence the overall execution time of slow attestation. However, large checkpoint intervals lead on average to the re-execution of more instructions after a power loss than small checkpoint intervals do. Therefore, a good strategy is to create a checkpoint after each 1 kB, 2 kB, or 4 kB of data processed by the cryptographic engine.

The energy consumption for different checkpoint intervals is shown in Fig. 4. We notice that energy consumption does not have a clear trend as execution time does for different values of the checkpoint interval. Essentially, the energy consumption of the overall operation is not influenced by the frequency of checkpoints. This is explained by the insignificant energy required to create a checkpoint compared to the overall energy necessary for the slow attestation operation. However, the observation made for execution time evaluation applies to energy consumption as well. More precisely, the closer the interval between two checkpoints, the lesser energy is wasted on re-execution after a power failure.

C. Secure Intermittent Computing

We present in Fig. 5 the execution time and energy spent on creating a secure checkpoint for different values of the stack size. Module stack is saved in a checkpoint and therefore the stack size has an impact on checkpoint creation. The global and static variables used by the module are not checkpointed because they are stored in the non-volatile memory of the secure intermittent system.

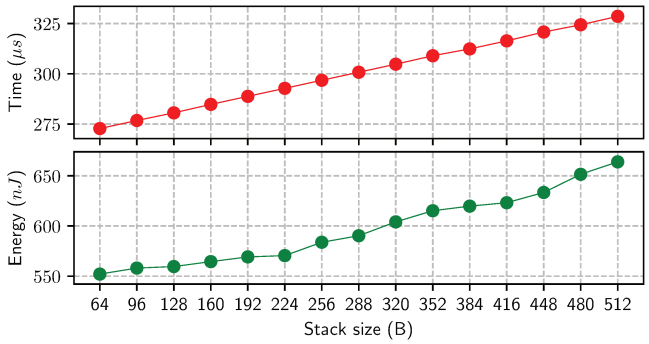

Fig. 5. Time and energy required to create a checkpoint for different stack sizes.

One can notice the very low time and energy spent on the creation of a checkpoint. Execution time and energy consumption grow linearly with the stack size. Creation of a checkpoint that includes a stack of 128 B requires only 280.6 μs and 559.6 nJ. The low-energy required to create a checkpoint is provided by the power conditioning capacitors when a power loss occurs during checkpoint creation. Hence, the creation of a checkpoint is an atomic operation in SIA. As a result, the intermittent system never ends in an invalid state which may affect its availability. Moreover, the entire software module, including its checkpoint, is protected by SIA.

Compared to the work of Suslowicz *et al.* [38], SIA does not keep two copies of the state and does not have to run a cryptographic algorithm over the entire module state twice to create a new checkpoint. This is why the creation of a checkpoint with SIA is between two and three orders of magnitude faster and more energy efficient than the same operation in their work. For example, the secure application continuity solution of Suslowicz *et al.* [38] requires 42.96 μJ and 17,095 μs to create a new checkpoint for a state of 512 B. In contrast, SIA creates a checkpoint for a stack of 512 B using 663.9 nJ and 328.6 μs. Hence, SIA requires 64.7 times less energy and is 52 times faster than their secure

application continuity solution. The advantage of SIA over the secure application continuity solution of Suslowicz *et al.* [38] increases if the stack size is smaller for SIA or the state size is bigger for their work.

Our experimental results show that the overhead of secure intermittent computing is very low. The resources spent on re-execution of a large part of a task clearly exceed the resources spent on creation of checkpoints and re-execution of a tiny fraction of the task in the event of a power loss. Therefore, a software module should use the secure checkpointing feature of SIA to optimize resource utilization.

VI. Related Work

Our work addresses the challenge of providing security services to an intermittent computing system, a problem that lies at the intersection of intermittent computing and security architectures. In this section, we briefly discuss the research that is closely related to SIA.

In the intermittent computing literature, there are only a few papers that provide some security properties to the state of an intermittent software module. Moreover, they only provide a subset of the security services provided by SIA at the cost of a high resource-utilization overhead.

Suslowicz *et al.* [38] proposed a solution for secure application continuity in intermittent systems. Their work adds integrity, authenticity, and freshness guarantees only to the power-off state of an intermittent software module. SIA secures both the power-on and power-off state of an intermittent software module and is between two to three orders of magnitude more efficient than the work of Suslowicz *et al.* [38].

Ghodsi, Garg, and Karri [15] proposed to encrypt the checkpoints of an intermittent system and optimized the checkpointing policy using machine learning. Their solution makes hardware modifications to the processor of the intermittent system. Compared to this work, SIA provides stronger security guarantees and does not require hardware modifications.

In the field of trusted computing, there are many security architectures that protect low-end embedded devices against software-level attacks. However, none of these security architectures provides guarantees in the intermittent computing model, mainly because they are not designed to consider this execution model. In particular, the power-off state of an intermittent software module is not protected.

Sancus [30], [31] is a low-cost security architecture for IoT devices that enforces isolation of software modules and provides several security properties such as remote attestation and secure communication. Sancus requires hardware modifications of the microprocessor and a custom software development toolchain. In contrast, SIA leverages security features available in low-end microcontrollers and does not require hardware modifications, nor a custom toolchain. Therefore, SIA is directly applicable to existing IoT devices, unlike Sancus which requires new hardware. Admittedly, Sancus supports more security properties through the custom toolchain and its memory access control is fine grained.

Soteria [17] builds on Sancus to provide offline software protection by encrypting the code and data of a protected module. However, Soteria does not support intermittent execution.

SMART [14] uses hardware-software codesign to build a lightweight software architecture which provides remote attestation of a memory range chosen by the verifier. TyTAN [9] is a trusted hardware architecture that has real-time guarantees and supports isolation between tasks, and secure interprocess communication (IPC).

Maene *et al.* [28] evaluated 14 hardware-based trusted computing architectures, including the above-mentioned four lightweight architectures with respect to different security properties. SIA provides five of the seven security properties considered by Maene *et al.* [28], namely isolation, attestation, dynamic root of trust, code confidentiality, and protection against software side-channels targeting memory access patterns. Additionally, SIA provides support for secure intermittent computing of virtually any application and its security properties hold in the intermittent computing model.

SWATT [36] is a software-based attestation technique based on the running time of the verification procedure. Since it does not use any form of isolation, SWATT is vulnerable to time-of-check time-of-use (TOCTOU) attacks.

VII. Conclusion

We proposed SIA, the first secure intermittent architecture for resource-constrained IoT devices. SIA builds on security features available on commercial off-the-shelf microcontrollers to protect both the power-on and power-off state of an intermittent software module. Our security architecture supports secure intermittent computing of virtually any application and provides three security services, namely self-attestation, remote attestation, and secure communication. SIA can be applied to existing IoT devices, even to those already deployed, because it does not require any hardware modifications.

We implemented and evaluated SIA on an MSP430 microprocessor. Our results confirm the feasibility of SIA for low-end devices powered by harvested energy. Moreover, experimental results show that intermittent computing should be enabled for all software modules protected by SIA because the overhead of checkpointing is negligible.

SIA is a simple and efficient solution to the problem of secure intermittent computing on low-end devices and provides support for remote intermittent attestation. In view of the predicted growth of the market for devices powered by harvested energy, the security features of SIA are crucial for the next-generation of IoT devices.

VIII. Acknowledgements

This research was supported in part by the Semiconductor Research Corporation (Task 2712.019) and by the National Science Foundation (Grant 1704176).

References

[1] F. Ambroglini. Energy harvesting application, July 2017. Lecture at the NiPS Summer School 2017 – Energy Harvesting: models and applications, Gubbio, Italy. Available at http://www.nipslab.org/sites/nipslab.org/files/Wisepower_SummerSchool_MicroEnergy2017_building.pdf.

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. Understanding the Mirai botnet. In E. Kirda and T. Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 1093–1110. USENIX Association, 2017.

[3] ARM. Memory Protection Unit (MPU). Available at http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0179b/CHDFDFIG.html.

[4] J.-P. Aumasson, P. Jovanovic, and S. Neves. CAESAR submission: NORX v3.0, September 2016. Available at https://competitions.cr.yp.to/round3/norxv30.pdf.

[5] M. Bellare, P. Rogaway, and D. A. Wagner. The EAX mode of operation. In B. K. Roy and W. Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004.

[6] G. Bertoni, J. Daemen, M. Peters, G. van Assche, and R. van Keer. CAESAR submission: Ketje v2, September 2016. Available at https://competitions.cr.yp.to/round3/ketjev2.pdf.

[7] J. Birr-Pixton. Cifra – A collection of cryptographic primitives targeted at embedded use. Available at https://github.com/ctz/cifra.

[8] S. Boisseau, A.-B. Duret, M. Perez, E. Jallas, and E. Jallas. Water flow energy harvesters for autonomous flowmeters. *Journal of Physics: Conference Series*, 773(1):012019, 2016.

[9] F. F. Brasser, B. E. Mahjoub, A. Sadeghi, C. Wachsmann, and P. Koeberl. TyTAN: tiny trust anchor for tiny devices. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*, pages 34:1–34:6. ACM, 2015.

[10] CAESAR committee. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. Available at https://competitions.cr.yp.to/caesar.html.

[11] H. Cheng, D. Dinu, and J. Großschädl. Efficient implementation of the SHA-512 hash function for 8-bit AVR microcontrollers. In J. Lanet and C. Toma, editors, *Innovative Security Solutions for Information Technology and Communications - 11th International Conference, SecITC 2018, Bucharest, Romania, November 8-9, 2018, Revised Selected Papers*, volume 11359 of *Lecture Notes in Computer Science*. Springer, 2018.

[12] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Trans. Information Theory*, 29(2):198–207, 1983.

[13] J. Donovan. New applications for energy harvesting. Available at https://www.mouser.com/applications/energy-harvesting-new-applications/.

[14] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito. SMART: secure and minimal architecture for (establishing dynamic) root of trust. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[15] Z. Ghodsi, S. Garg, and R. Karri. Optimal checkpointing for secure intermittently-powered IoT devices. In S. Parameswaran, editor, *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, CA, USA, November 13-16, 2017*, pages 376–383. IEEE, 2017.

[16] W. Goh and A. Dannenberg. MSP430 FRAM technology – how to and best practices. Technical report, Texas Instruments, June 2014. Available at http://www.ti.com/lit/an/slaa628/slaa628.pdf.

[17] J. Götzfried, T. Müller, R. de Clercq, P. Maene, F. C. Freiling, and I. Verbauwhede. Soteria: Offline software protection within low-cost embedded devices. In *Proceedings of the 31st Annual Computer Security Applications Conference, Los Angeles, CA, USA, December 7-11, 2015*, pages 241–250. ACM, 2015.

[18] M. Hicks. Clank: Architectural support for intermittent computation. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*, pages 228–240. ACM, 2017.

[19] D. Hoffmann, A. Willmann, R. Göpfert, P. Becker, B. Folkmer, and Y. Manoli. Energy harvesting from fluid flow in water pipelines for smart metering applications. *Journal of Physics: Conference Series*, 476(1):012104, 2013.

[20] T. Instruments. MSP430FR58xx, MSP430FR59xx, and MSP430FR6xx family user's guide, October 2012. Revised December 2017. Available at http://www.ti.com/lit/ug/slau367o/slau367o.pdf.

[21] T. Instruments. MSP430FR5994 LaunchPad development kit (MSP-EXP430FR5994), March 2016. Revised April 2016. Available at http://www.ti.com/lit/ug/slau678a/slau678a.pdf.

[22] T. Instruments. MSP MCU FRAM utilities user's guide, October 2017. Version 3.10.00.10. Available at http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/FRAM_Utilities/latest/exports/FRAM-Utilities-UsersGuide.pdf.

[23] G. Kambourakis, C. Kolias, and A. Stavrou. The Mirai botnet and the IoT zombie armies. In *2017 IEEE Military Communications Conference, MILCOM 2017, Baltimore, MD, USA, October 23-25, 2017*, pages 267–272. IEEE, 2017.

[24] K.-B. Kim, C. I. Kim, Y. H. Jeong, J.-H. Cho, J.-H. Paik, S. Nahm, J. B. Lim, and T.-H. Seong. Energy harvesting characteristics from water flow by piezoelectric energy harvester device using Cr/Nb doped Pb (Zr, Ti) O3 bimorph cantilever. *Japanese Journal of Applied Physics*, 52(10S):10MB01, 2013.

[25] A. Kingatua. The how and why of energy harvesting for low-power applications, June 2016. Available at https://www.allaboutcircuits.com/technical-articles/how-why-of-energy-harvesting-for-low-power-applications/.

[26] C. Kolias, G. Kambourakis, A. Stavrou, and J. M. Voas. DDoS in the IoT: Mirai and other botnets. *IEEE Computer*, 50(7):80–84, 2017.

[27] A. S. Krishnan and P. Schaumont. Exploiting security vulnerabilities in intermittent computing. In A. Chattopadhyay, C. Rebeiro, and Y. Yarom, editors, *Security, Privacy, and Applied Cryptography Engineering - 8th International Conference, SPACE 2018, Kanpur, India, December 15-19, 2018, Proceedings*, volume 11348 of *Lecture Notes in Computer Science*. Springer, 2018.

[28] P. Maene, J. Götzfried, R. de Clercq, T. Müller, F. C. Freiling, and I. Verbauwhede. Hardware-based trusted computing architectures for isolation and attestation. *IEEE Trans. Computers*, 67(3):361–374, 2018.

[29] Microchip. PIC16F18446 product family. Available at https://www.microchip.com/promo/pic16f184xx-product-family.

[30] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. V. Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In S. T. King, editor, *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 479–494. USENIX Association, 2013.

[31] J. Noorman, J. V. Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. C. Freiling. Sancus 2.0: A low-cost security architecture for IoT devices. *ACM Trans. Priv. Secur.*, 20(3):7:1–7:33, 2017.

[32] K. Pier. MSP code protection features. Technical report, Texas Instruments, December 2015. Available at http://www.ti.com/lit/an/slaa685/slaa685.pdf.

[33] F. Piessens and I. Verbauwhede. Software security: Vulnerabilities and countermeasures for two attacker models. In L. Fanucci and J. Teich, editors, *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, pages 990–999. IEEE, 2016.

[34] A. Poor. Reaping the energy harvest [resources]. *IEEE Spectrum*, 52(4):23–24, 2015.

[35] P. Schaumont and P. Montuschi. The rise of hardware security in computer architectures. *IEEE Computer*, 51(8):4–5, 2018.

[36] A. Seshadri, A. Perrig, L. van Doorn, and P. K. Khosla. SWATT: software-based attestation for embedded devices. In *2004 IEEE Symposium on Security and Privacy (S&P 2004), 9-12 May 2004, Berkeley, CA, USA*, page 272. IEEE Computer Society, 2004.

[37] R. Strackx, F. Piessens, and B. Preneel. Efficient isolation of trusted subsystems in embedded systems. In S. Jajodia and J. Zhou, editors, *Security and Privacy in Communication Networks - 6th Iternational ICST Conference, SecureComm 2010, Singapore, September 7-9, 2010. Proceedings*, volume 50 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 344–361. Springer, 2010.

[38] C. Suslowicz, A. S. Krishnan, D. Dinu, and P. Schaumont. Secure application continuity in intermittent systems. In *Proceedings of the 9th International Green and Sustainable Computing Conference, Pittsburgh, PA, USA, October 22-24*, 2018.

[39] M. Taghavi, A. Sadeghi, B. Mazzolai, L. Beccai, and V. Mattoli. Triboelectric-based harvesting of gas flow energy and powerless sensing applications. *Applied Surface Science*, 323:82–87, 2014.

[40] P. Thanigai. Closing the security gap with TI's MSP430 FRAM-based microcontrollers, September 2014. Available at http://www.ti.com/lit/wp/slay035/slay035.pdf.

[41] U.S. Department Of Commerce/National Institute of Standards and Technology. Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. *NIST Special Publication 800-38D*, pages 1–39, November 2007. Available at http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf.