Surplus and Scarce Energy: Designing and Optimizing Security for Energy Harvested Internet of Things

Archanaa Santhana Krishnan

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Patrick R. Schaumont, Chair

Leyla Nazhand-ali

Dong Ha

May 09, 2018

Blacksburg, Virginia

Keywords: Embedded systems, Internet of Things, Security, Precomputing, Intermittent computing

Surplus and Scarce Energy: Designing and Optimizing Security for Energy Harvested Internet of Things

Archanaa Santhana Krishnan

(ABSTRACT)

Internet of Things require a continuous power supply for longevity and energy harvesting from ambient sources enable sustainable operation of such embedded devices. Using self-powered power supply gives raise two scenarios, where there is surplus or scarce harvested energy. In situations where the harvester is capable of harvesting beyond its storage capacity, the surplus energy is wasted. In situations where the harvester does not have sufficient resources, the sparse harvested energy can only transiently power the device. Transiently powered devices, referred to as intermittent computing devices, ensure forward progress by storing checkpoints of the device state at regular intervals. Irrespective of the availability of energy, the device should have adequate security.

This thesis addresses the security of energy harvested embedded devices in both energy scenarios. First, we propose precomputation, an optimization technique, that utilizes the surplus energy. We study two cryptographic applications, namely bulk encryption and true random number generation, and we show that precomputing improves energy efficiency and algorithm latency in both applications. Second, we analyze the security pitfalls in transiently powered devices. To secure transiently powered devices, we propose the Secure Intermittent Computing Protocol. The protocol provides continuity to underlying application, atomicity to protocol operations and detects replay and tampering of checkpoints. Both the proposals together provide comprehensive security to self-powered embedded devices.

Surplus and Scarce Energy: Designing and Optimizing Security for Energy Harvested Internet of Things

Archanaa Santhana Krishnan

(GENERAL AUDIENCE ABSTRACT)

Internet of Things(IoT) is a collection of interconnected devices which collects data from its surrounding environment. The data collected from these devices enable emerging technologies like smart home and smart cities, where objects are controlled remotely. With the increase in the number of such devices, there is a demand for self-powered devices to conserve electrical energy.

Energy harvesters are suitable for this purpose because they convert ambient energy into electrical energy to be stored in an energy buffer, which is to be used when required by the device. Using energy harvesters as power supply presents us with two scenarios. First, when there is sufficient ambient energy, the surplus energy, which is the energy harvested beyond the storage capacity of the buffer, is not consumed by the device and thus, wasted. Second, when the harvested energy is scarce, the device is forced to shutdown due to lack of power. In this thesis, we consider the overall security of an energy harvested IoT device in both energy scenarios. We optimize cryptographic algorithms to utilize the surplus energy and design a secure protocol to protect the device when the energy is scarce. Utilizing both the ideas together provides adequate security to the Internet of Things.

Acknowledgments

I would like to thank my advisor, Dr. Patrick Schaumont, for his guidance and support. All the lessons I learned from him greatly helped in drafting this thesis and making me a better researcher.

I am thankful to Dr. Leyla Nazhandali and Dr. Dong Ha, for serving on my thesis committee.

Special thanks to Dr. Peter Athanas, Qualifying Examination committee chair, for helping with the dual use of my thesis.

Many thanks to the present members and alumni of the Secure Embedded Systems Lab for helping me in my research and making this an enjoyable experience.

And thank you to all my friends and family for their support.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC Analog-to-Digital Converter

AEAD Authenticated-Encryption with Associated-Data

AES Advance Encryption Standard

AES-CTR Advance Encryption Standard in Counter Mode Encryption

API Application Programming Interface

CAESAR Competition for Authenticated Encryption: Security, Applicability, and Robustness

CCS Code Composer Studio

CKP Checkpoint

CKPX X^{th} Checkpoint

CPU Central Processing Unit

CTPL Compute Through Power Loss utility

DCO Digitally Controller Oscillator

DINO Death Is Not an Option

DMA Direct Memory Access

ECDSA Elliptic Curve Digital Signature Algorithm

EDP Energy Delay Product

FRAM Ferroelectric Random Access Memory

HW Hardware

I/O Input/Output

IoT Internet of Things

JTAG Joint Test Action Group

KCP Keccak Code Package

LPM Low Power Mode

MAC Medium Access Control Layer

MSP Mixed Signal Processor

NVM Non-Volatile Memory

OPT One-Time Pad

PC Program Counter

PHY Physical Layer

PRNG Pseudo Random Number Generator

RAM Random-Access Memory

RFID Radio-Frequency Identification

RNG Random Number Generation

SIC Secure Intermittent Computing

SICP Secure Intermittent Computing Protocol

SMART Secure and Minimal Architecture for establishing a dynamic Root of Trust

SRAM Static Random-Access Memory

SS State Save packet

SW Software

TI Texas Instrument

TRNG True Random Number Generation

VLO Very-low-frequency Oscillator

Preface

This thesis is composed of the following manuscripts:

[1] Charles Suslowicz, Archanaa S. Krishnan, Patrick Schaumont. Optimizing Cryptography in Energy Harvesting Applications. In Proceedings of the 2017 Workshop on Attacks and Solutions in Hardware Security (ASHES 17). ACM, New York, NY, USA, 17-26.

[2] Archanaa S. Krishnan , Charles Suslowicz, and Patrick Schaumont. Secure Intermittent Computing: Protecting State Across Power Loss. Under review

It is divided into two chapters. Chapter 2 includes the manuscript [1], where I worked on the case study for bulk encryption. It involved computing the energy delay product improvement for precomputed AES counter mode encryption. Chapter 3 includes the manuscript from [2], where I designed the Secure Intermittent Computing Protocol to secure the checkpoints of an intermittent system.

Chapter 1

Introduction

Internet of Things (IoT) is a network of embedded devices that collect and process data about their surrounding environment to support emerging technologies like smart homes and smart grids. They are placed in hard to access locations to collect data that is beyond human reach, where traditional power-up using socket and cord is inconvenient. Battery powered devices, which are an alternative to grid-connected devices, are limited in scope because of the bulkiness of batteries and the need for replacing/recharging batteries at frequent intervals. Also, the chemicals used in batteries can cause environmental damage if the batteries are not disposed safely.

With the increase in the number of IoT devices, there is an increasing demand for ambient energy harvesting to enable energy autonomous operations. By using energy harvesters, IoT devices can operate indefinitely by drawing energy from surrounding environment. Various ambient energy sources are available to be harvested, including solar energy, thermal gradient, vibration energy, wind energy and electromagnetic sources. The energy harvested from ambient energy sources is either in microwatt scale for vibration and wind energy [1, 2] or in milliwatt scale for solar and thermal electric energy [4, 7], which is more than sufficient to power IoT devices that operate at low duty cycles or at infrequent intervals of time. Compared to batteries, energy harvesters are low maintenance circuits that can be deployed in the field without replacement or recharging for the their lifetime.

1.1 Intermittent Computing

Even though energy harvesters provide energy-autonomy to IoT devices, the energy that is harvested is not abundant. A typical energy harvester converts ambient energy into electrical energy to be collected in a small energy buffer, such as a supercapacitor. Once the buffer has accumulated sufficient energy, the device begins operation until the buffer is drained. On one hand, when the buffer is empty, the device shuts down and waits for the buffer to be recharged again. The device restarts when the buffer accumulates sufficient energy again. Transience in input power forces the device to restart frequently, leading to an *intermittent computing model*. On the other hand, when the device is idle, but the harvester continues to harvest beyond the storage capabilities of the buffer, the surplus energy is wasted. The energy harvesting circuits typically use a discharge circuit to dissipate this energy, while the device performs normal tasks in the *continuous execution model*.

The presence of surplus energy from the harvester does not affect the device. Whereas, the transient power supply due to energy scarcity inhibits continuous execution. When the device shuts down, it is forced to restart upon subsequent power-up, failing to execute completely until sufficient power is available. To address the transient nature of harvested power supply, state-of-the-art solutions incorporate a checkpointing technique known as intermittent computing. In intermittent computing, the device stores a snapshot of the system and application state in non-volatile memory. The stored checkpoint data can then be used upon power-up to restore the device to its last known state, inadvertently resuming execution of the application.

Irrespective of the available energy, the device has to continue operation with adequate security. IoT devices are not equipped with extensive computational capabilities and storage power like conventional computers, but at the same time, their security is as important as

that of computers because they collect and process sensitive and private information that needs to be protected from adversaries. They typically contain 8, 16 or 32-bit processors that are built for low power applications. So, it is challenging to implement conventional security measures on small processors without using all their computational power and time to provide the necessary security. These resource constrained devices require lightweight cryptography which can be implemented on top of existing applications. The unpredictable nature of intermittent systems brings in two interesting scenarios that must be considered when thinking about implementing security on self-powered devices. First, when there is surplus harvested energy, and second, when the energy is scarce.

1.2 Security for Surplus Energy

The surplus energy, which is harvested even after the storage buffer has reached its maximum capacity [6], is not usually consumed for useful operations. Instead of wasting the surplus energy, it can be used to compute something useful that can be used later. This process, know as *precomputing*, computes input data independent variables in advance to be used when needed. In regular monolithic execution, the device performs all operations online once the input data is available. Whereas, if input independent operations can be performed in advance, and their results can be stored for future use, the latency of the algorithm is reduced to just input dependent operations.

Cryptographic algorithms are ideal for precomputation as they can be divided into input data independent and dependent parts. Figure 1.1 demonstrates this feature using an example of a Diffie-Hellman key exchange between Alice and Bob. In Figure 1.1 the first two steps, generating a random point and its multiplication with the generator, G, are input data independent operations. Alice executes all three steps of the key exchange only after she

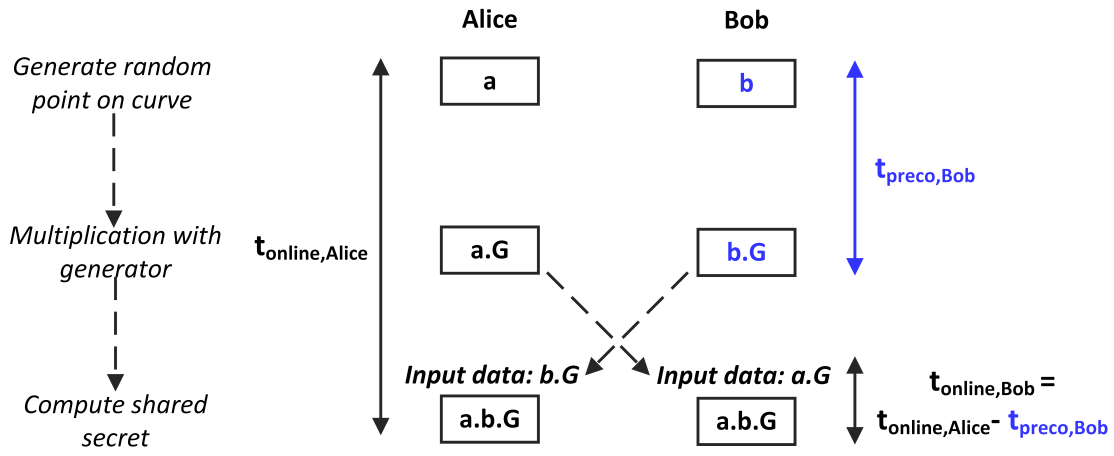

Figure 1.1: Diffie-Hellman Key Exchange between Alice, with monolithic execution, and Bob, with precomputation techniques

receives the input, $b.G$, from Bob. Whereas, Bob precomputes b and $b.G$ with the surplus energy and stores them as coupons in non-volatile memory. When Bob is presented with input, $a.G$, from Alice he directly proceeds to compute the shared secret. This reduces the runtime cost of the key exchange for Bob from Alice's monolithic operation, $t_{online,Alice}$, $t_{online,Alice}$, to just the time taken to compute the shared secret, $t_{online,Alice}$ - $t_{preco,Bob}$. In this thesis, we show that precomputation can improve runtime latency and energy efficiency of the application.

1.3 Security for Scarce Energy

Intermittent computing solutions make it feasible for long running applications to be implemented on energy harvested IoT devices [3, 5]. The checkpoint, which is a snapshot of system and application state, contains the necessary information, like the program counter, stack and other application variables that are needed to ensure forward progress of the application after power loss. During the power-off period, the checkpoint is stored as plain text in

non-volatile memory. Existing intermittent computing solutions do not consider power-loss as a threat vector. When the adversary can control power, they can alter, copy or replay checkpoints. If the device is running a cryptographic application, like in Figure 1.1, then the checkpoints will contain the intermediate state of the cryptographic operation, which could include the secret a or $a.b.G$. Sensitive data stored in non-volatile memory can be copied or altered after power-loss. Secure storage of checkpoints is a concern which is addressed in this thesis.

1.4 Contributions and Outline

In this work, we develop a secure intermittent computing device that utilizes the surplus energy and protects itself when the harvested energy is scarce. In Chapter 2, we detail the advantages of precomputation to cryptography with two supporting case studies, bulk encryption and random number generation. Then in Chapter 3, we treat power-loss as an adversarial event and analyze the security risks involved in existing intermittent computing solutions. We then present the Secure Intermittent Computing Protocol, which can be used on top of any generic checkpointing scheme, to protect the device after power loss, followed by conclusions in Chapter 4.

Chapter 2

Optimizing Cryptography for Energy Harvesting Applications

[1] Charles Suslowicz, Archanaa S. Krishnan, Patrick Schaumont. Optimizing Cryptography in Energy Harvesting Applications. In Proceedings of the 2017 Workshop on Attacks and Solutions in Hardware Security (ASHES 17). ACM, New York, NY, USA, 17-26.

2.1 Introduction

Devices within the Internet of Things (IoT) are expected to maintain a ubiquitous network connection. This presents a significant challenge in its implementation as many devices lack access to a continuous and uninterrupted power supply. Energy harvesting devices resolve this problem by recharging their local power reservoir, often a supercapacitor or a rechargeable battery, via energy available in their surroundings. This improves IoT logistics but creates a challenge in the computing domain through the introduction of unexpected and difficult to predict power loss.

The domain of intermittent computing contains a significant amount of work to address power loss during the operation of a device including techniques such as DINO, Clank, or Hibernus [18] [12] [4]. In all of these cases, there is an assumption that the device will be

doing more work than there is energy available, and this is reflected in their design to preserve the system state gracefully or avoid ever reaching a state where power loss is detrimental to the device's computations. In this chapter, we analyze the less addressed case that an IoT device will have excess power during periods where there is little to no work to be done.

Computing devices have long periods of idle activity before executing their necessary task. These idle periods are common enough to lead to the development of power management features to reduce the amount of power wasted on non-productive CPU cycles. Energy harvested systems face similar problems and many technologies exist, such as the low power modes of the MSP430 chip family, to reduce power draw when a system is idle. However, energy harvested systems are bounded on the other extreme by the maximum amount of harvested energy they can store in their local battery or supercapacitor. As a result, remaining idle during a period where the storage medium is full and additional energy is collected by the energy harvested results in a complete waste of useful energy.

The expectation of excess energy for energy harvested systems is not unreasonable. Work by Simjee and Chou showed that a solar cell could rapidly, within a few minutes, recharge a supercapacitor while powering a sensor node and that there were large periods of time during a multi-day stress test where the supercapacitor was fully charged during sensor operation [26].

We propose the alteration of an energy harvested system's cryptographic algorithms to exploit the energy wasted when the harvester continues to collect energy after the storage medium is full. Specifically, we show the device can use this excess energy to generate *coupons* for future cryptographic operations. These *coupons* consist of the offline portions of cryptographic operations that do not rely on the runtime inputs. Examples of this type of precomputation include: generating the full hash chain of a Winternitz one time signature [3], generation and storage of random numbers, and the expansion of a key schedule [1].

These operations must be completed for the cryptographic operation to be successful, but they do not need to be done at the exact moment the operation is requested. Previous work has exploited this relationship to improve performance in many fields [6] [29] [25] [22]. We explore this capability to improve the energy efficiency of devices with may have excess, or free, energy available for use.

Both side effects of precomputation: the reduction in runtime latency and the reduction in energy required for the runtime operation, benefit energy harvested devices. In energy harvested devices the ability to power precomputation efforts with energy that would otherwise be unused is valuable and unique. Our work demonstrates a *coupon* precomputation scheme which allows the system to execute AES-CTR encryptions for up to 28 times less energy at runtime and generate random numbers for over 100 times less energy at runtime compared to the energy required to execute the entire operation.

2.1.1 Contributions

In this chapter, we present the following contributions for the optimization of cryptographic operations in energy harvesting applications:

1. *Precomputation as an Energy Optimization*: We demonstrate the expansion of precomputation from a latency optimization to an energy optimization in cases where an energy harvester can collect more energy than can be stored locally. This energy optimization allows system designers to service more requests with an identical device, reduce the size of the necessary energy store to meet a designated worst case operational capacity, and increase the device's security against hardware attacks.

2. *Identify Algorithms that Benefit from Precomputation*: We demonstrate two different algorithms that specifically benefit from this method of precomputation and highlight

the features of the algorithms that make them good candidates for *coupon* precomputation. We describe empirical findings in the case of AES-CTR mode encryption on MSP430 and ARM-Cortex M4, and a true random number generator (on MSP430.

3. *Metric for Comparison*: We present a framework of metrics for the comparison of algorithms and the effect of precomputation on their performance in terms of energy consumed, cycle count, operational delay, and the Energy-Delay Produce (EDP) of their execution. This framework enables an effective judgement on the suitability of precomputation for a particular implementation and provides insight into the potential performance of a device utilizing precomputed *coupons* for cryptographic operations.

The rest of the chapter is structured in the following manner. Section 2 discusses previous work in energy harvested systems, precomputation of cryptographic algorithms, scaling within the IoT, and our threat model. Section 3 details our core concepts: the computation of *coupons* and our framework of metrics for comparison between precomputed and non-precomputed algorithms. Section 4 contains the two case studies and their related analysis. Section 5 and Section 6 present future work and our conclusions respectively.

2.2 Background

Neither energy harvested systems nor precomputation are new ideas or paradigms. Significant previous work has outlined the growth and operation of energy harvested systems and the difficulties created in intermittent computing operations. Additionally, precomputation has been discussed as an optimization technique for decades in cryptography [6]. Here we discuss these previous works, and how their contributions enable our work to optimize the operation of energy harvested devices.

Table 2.1: Data and Energy Retention Time

Technology	Format	Retention Time (~$20°C$)
Supercapacitor [19]	Energy	5.5 days
Li-Ion Battery [27]	Energy	1-2 years
FRAM [28]	Data	100 years

2.2.1 Energy Harvested System Operations

Energy harvested systems are a class of transiently powered devices that gather energy from the surrounding environment to power their operation. The methods used range from solar cells, to the RFID PHY and MAC layer, to motion and vibration via piezoelectric circuits [7] [22]. In all cases, the energy harvested device uses this ambient available energy to power its operation and often fill a local energy store in the form of a rechargeable battery or supercapacitor.

The nature of energy harvested devices leads to the possibility that power will be lost at any point during an operation. A growing body of work on intermittent computing provides potential solutions to this problem. For our study, we assume one of these solutions from Mementos to QuickRecall or a hardware enabled solution like Clank is sufficient to resolve the loss of power mid-computation [23] [14] [12]. It must be noted that without such a solution, it is possible for the device to land in an undefined state as data has been written to non-volatile memory by a partially completed operation, and subsequent operations will fail due to these faulty or unexpected inputs [18] [9].

The volatile nature of power for energy harvested systems highlights the stable nature of data stored in non-volatile memory compared to the retention of energy stored in a battery or supercapacitor. Energy within a supercapacitor will discharge based on the leakage current and surrounding circuitry at a relatively quick rate. A rechargeable battery will retain the same energy for a longer period of time, but will also eventually discharge even if the device

has not executed any operations but no additional energy is provided [27] [19].

When that energy is converted to a *coupon* and stored in non-volatile memory, it can be maintained in FRAM for 100 years at room temperature (20° C) and 10 years in extreme conditions (85° C). The stability of this data is illustrated in Table 2.1 and is a strong argument for precomputation when energy is available as the loss of energy in the future will have little effect on data stored in a non-volatile memory [28].

The existence of this excess energy is a unique benefit of energy harvested devices. Work in the mid-2000s by Kansal et al. and Hsu et al. showed the potential to increase or decrease the duty cycle of energy harvested devices to match the energy available from a harvester. When additional energy was available, energy harvested systems could consume that energy to activate more frequently while maintaining a neutral energy balance, and thereby conducting more operations than a similar system not making use of the increased energy available from the harvester [13] [16]. In this chapter, we explore using this excess energy to precompute *coupons* and improve the efficiency of later cryptographic operations rather than increase the sample or measurement rate of a sensor.

2.2.2 Previous Work in Precomputation

The concept of doing work ahead of time for an operation has been used throughout history for complex techniques in the form of lookup tables and references. This process is illustrated in Figure 2.1 highlighting the separation of a process into an offline, precompute, portion and an online, runtime, portion. The application of this to cryptographic operations is a straightforward adaptation and underlies the concept of rainbow tables and other optimization techniques [20] [6]. Additionally, precomputation has proven an effective optimization tool in other fields, such as Quality of Service routing within large networks, where some

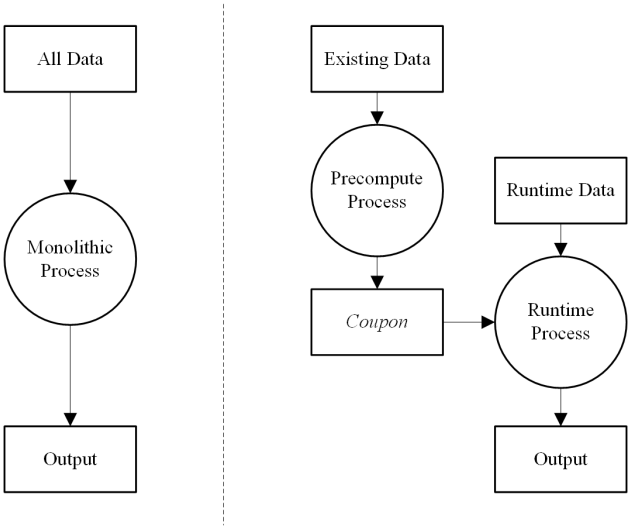

Figure 2.1: The process for a single operation, shown on left, and the precomputed operation, shown on right. When combined, the *coupon* and runtime data, available only immediately before execution, allow the generation of an output identical to the single, monolithic, process.

parameters of a problem are known ahead of time and latency is a critical metric [21].

Precomputation does not reduce operational latency without introducing its own challenges. The energy cost of the precomputation itself must be accounted for, the precomputed values must be kept secure, even during potential power loss, and the algorithms must be partitioned in such a way that the runtime operation is sufficiently faster to warrant the data storage expense imposed by *coupons*. In the case of energy harvested systems, the ability to employ excess energy reduces the energy cost of the precomputed *coupons* to zero. This leaves only the security of *coupons* and algorithmic partitioning as challenges to address in the implementation of precomputation for energy harvested systems.

Within the IoT, previous work has identified the value of precomputation for resource constrained devices. Ateniese et al. identified and demonstrated the potential benefits for the precomputation of ECDSA signatures in wireless sensor nodes in [1] and further expanded on their work in [2] in 2017. This work highlighted the applicability of precomputation for

IoT devices and a cryptographic operation. We show here a more general concept for the utilization of the excess energy generated by energy harvested devices and its effectiveness across two very different cryptographic primitives.

2.2.3 Scaling Within the Internet of Things

Neither precomputation nor energy harvesting would be valuable avenues of consideration if IoT devices scaled in the same manner as traditional computers. Unfortunately, the nature of the IoT is to deploy many small devices, too many to easily manage or service, across a large area over a long period of time [24]. This paradigm leads to cheap devices that are expected to operate for as long as possible without additional human interaction or support [8].

Batteries, if they scaled in the same manner as silicon, would provide the perfect power source for such devices. Unfortunately, batteries do not scale in a manner similar to Moore's Law, and often make up the majority of mass in modern electrical equipment to provide only a short period of power before recharging is required. Energy harvested devices provide a solution as a device with its own recharging mechanism paired with an energy store, either a battery or supercapacitor depending on the application. This improves the scaling of IoT devices by allowing each device to remain small, and cheap while staying operational without human intervention for far longer than a normal battery's lifetime [26].

A challenge of energy harvested systems is the likelihood that at some points there will be no energy available for the system and at other times there will be excess energy unused by the system. Previous work on intermittent computing addresses the former case and provides a backstop to ensure proper behavior when power is limited [18] [12]. Our work provides an opportunity to exploit the latter case of excess energy. This is illustrated in Figure 2.2

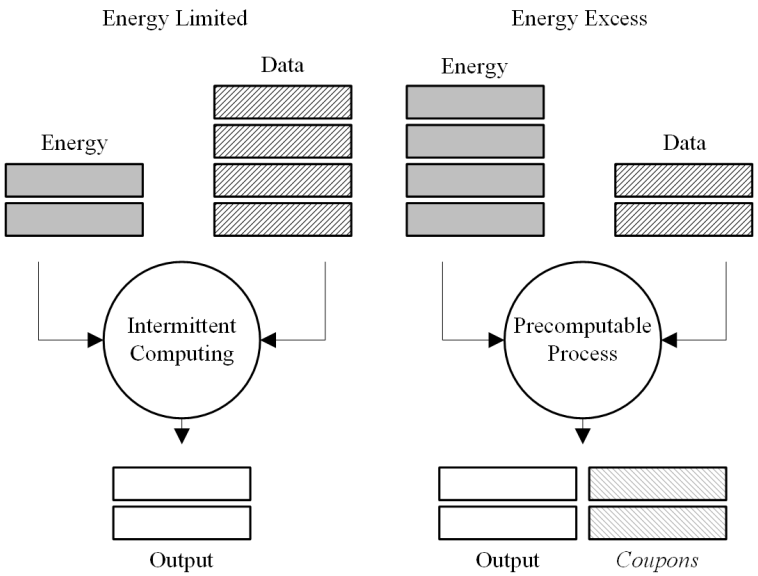

Figure 2.2: Intermittent computing ensures that as much output as possible is created during a scarcity of energy. Our work ensures that excess energy is utilized to improve the efficiency of future operations with *coupons*.

which highlights the ability of an intermittent process to produce as much output as there is energy available, while a precomputation enabled process produces as much output as there is data available and generates *coupons* with any excess energy.

Other scaling solutions for the IoT have been proposed, but they require much steeper trade-offs in operational flexibility and cost for their improvements in IoT device performance. For example, bespoke processors take a very different approach to operational efficiency and have shown dramatic improvements in energy usage. A bespoke processor is a microcontroller that has been modified by removing all capability not required to properly execute its expected program. This provides a significant energy cost improvement as all unnecessary hardware components of the processor have been removed but incurs additional costs as the device is no longer reconfigurable and must be custom manufactured for a specific implementation [8].

Our proposal for precomputation with energy harvested devices provides a solution to

the scaling problem facing new IoT devices without the drawbacks demonstrated by recent hardware proposals and with full interoperability with existing intermittent computing paradigms.

2.2.4 Threat Model

A multitude of threats exists for energy harvested systems, especially those deployed in remote and unsecured areas. In recognition of this, our threat model includes adversaries that can physically access and control the environment around the device. Additionally, it is assumed that adversaries can control the input and output of the system during operation and take physical measurements of the system during operation. We do not expect an adversary to be able to view on-chip memory or register values during operation and expect this to be beyond the capability of a competent adversary when the proper configuration recommendations are observed (JTAG locking enabled, no debugging port available, etc). This is a key consideration of our *coupon* precomputation scheme as an adversary that can view on-chip memory during device operation would be able to observe precomputed values prior to their use in cryptographic operations and bypass all reasonable attempts to secure the operation of the system.

2.3 Precomputation, Energy Harvested Devices, and Cryptography

How can the latency and efficiency of cryptographic operations on these platforms be improved? First, we evaluate the limiting factors of energy harvested platforms for cryptographic operations, Second, we evaluate the partitioning of cryptographic algorithms, the

use of intermediate value *coupons* and their effect on process execution. Finally, we consider a framework of metrics for evaluating the benefit to a particular implementation in terms of energy efficiency.

In all of our considered cases, the underlying premise is the opportunity to exploit excess energy collected by an energy harvester. We propose utilizing this excess energy to precompute *coupons* consisting of non-input related computations for future cryptographic operations. Their use has ramifications for the design of future algorithms within this space according to our analysis and the results of our case studies in Section 4.

2.3.1 Intermittent Computing and Cryptography

In this chapter, we focus on methods to exploit the case where an abundance of energy is available, but all of our proposed solutions should be implemented in conjunction with an intermittent computing paradigm (checkpointing, idempotent processing, etc) to ensure the proper operation of the device during periods of low energy when *coupons* are most likely to be consumed and performance benefits realized.

2.3.2 *Coupons* and the Precomputation of Algorithms

A *coupon* is some amount of data generated during a period of excess energy in preparation for a future cryptographic operation. It must be stored in a secure location (in our case studies on-chip non-volatile memory) and be readily available for the runtime operation in order to maximize the *coupon's* reduction of the runtime operation's latency and energy cost.

The generation of a *coupon* will be unique to each cryptographic operation, but in all cases, it represents a function that accepts some input data not dependent on runtime parameters

and some amount of energy to produce an intermediate data block in the operation. This process converts energy that would normally be stored in an energy storage medium, a supercapacitor or rechargeable battery, into data that can be stored on silicon. By executing this conversion, the energy harvesting system converts energy into data for a future operation thereby reducing the energy required to produce the final output at runtime as illustrated by the equations in (1).

$$\begin{aligned} E_{original} &\leq E_{precomputation} + E_{runtime} \\ E_{runtime} &< E_{original} \end{aligned} \tag{2.1}$$

The value of this transformation can be seen when considering the EDP of the final computation. A difference in the EDP of the runtime operation shows that the energy efficiency improvements were not achieved strictly through a reduction in processing speed or increased latency. The EDP improvements demonstrated in Section 4 makes it clear that cryptographic operations utilizing *coupons* provide better energy efficiency and performance than those without.

2.3.3 Metrics for Comparison

To properly evaluate the effectiveness of *coupon* precomputation we considered the energy required to complete an operation, the operation's cycle count, an operation's delay, and the EDP of the computation. This framework of metrics allows evaluation of the benefit of precomputing a particular algorithm. Additionally, these metrics support the comparison between different implementations of cryptographic algorithms on energy harvested devices and show definitively that the proper implementation of a *coupon* precomputation scheme

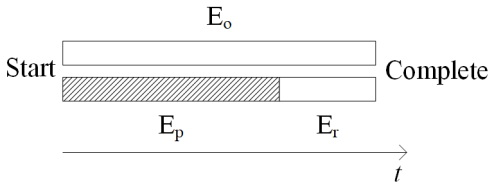

Figure 2.3: Illustration of the energy required for a monolithic computation versus seperation into a precomputation and runtime operation.

can be beneficial.

The first set of metrics considered are for a non-precomputed, or standard, operation. These are taken as the energy (E_o), the cycle count (C_o), the delay (D_o), and the EDP (EDP_o), computed as $E_o \times D_o$. These are compared with the separated metrics for precomputation, identified with a subscript p, and for the runtime only operation, identified with a subscript r. In general, it is expected that the following relationships are true:

$$\begin{aligned} E_o &\leq E_p + E_r \\ C_o &\leq C_p + C_r \\ D_o &\leq D_p + D_r \end{aligned} \tag{2.2}$$

This follows from the most efficient separation of an algorithm is an exact split without any supporting logic for data manipulation. In the majority of cases, the sum of the precomputation and runtime operations will be slightly greater than a monolithic execution of the operation. Despite this, we are able to show tremendous gains in operational efficiently because the runtime operational parameters (E_r, C_r, D_r) are much smaller than the monolithic or original operations.

The EDP of the operation is an important metric to identify improvements in operational efficiency when a device is able to reduce its energy consumption and work more slowly on an operation or perform the opposite. By taking the EDP we are able to show that

the precomputed operations are significantly more efficient than the monolithic operations regardless of operating mode for the device.

Finally, when analyzing a specific implementation we consider the ratios of the runtime operation to the monolithic operation as the following terms:

$$\begin{aligned} \text{Speedup} : C_i &= \frac{C_o}{C_r} \\ \text{Energy Improvement} : E_i &= \frac{E_o}{E_r} \\ \text{Latency Improvement} : D_i &= \frac{D_o}{D_r} \\ \text{EDP Improvement} : EDP_i &= \frac{EDP_o}{EDP_r} \end{aligned} \tag{2.3}$$

By considering a ratio of the original computation to the runtime computation, which utilizes a *coupon*, we are able to measure the benefit conferred by precomputing a portion of the algorithm. The cost of computing a *coupon*, E_p, is less valuable than the ratio of E_o and E_r because the *coupon* computation is executed during periods of excess energy. The Energy Improvement, E_i, provides a comparison of the unavoidable energy costs associated with the operation despite a precomputation scheme and supports analysis on the value of precomputation for that specific cryptographic operation.

Similarly, precomputation delay, D_p, is not considered when analyzing a specific implementation because the *coupon* computation should be executed when no other tasks are pending. However, it should be noted that both E_p and D_p are non-zero and limit a system's performance if additional operations are required after all precomputed *coupons* have been consumed. This will prevent a system from permanently executing at the upper limit, $\frac{1}{D_r}$, shown in Figure 2.4.

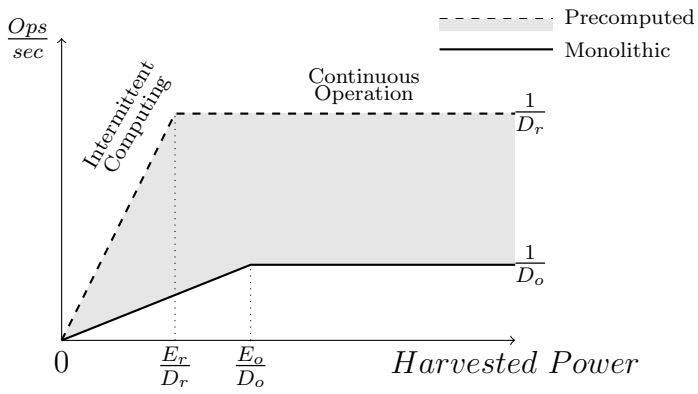

Figure 2.4: Operations per second as a function of Energy influx into the system. When *coupons* are available the device is able to execute more operations within a given time period until limited by the latency of the minimum runtime computation (D_r).

2.3.4 Conversion of Energy to Data via Precomputation

The value of converting excess energy to data via precomputation deserves additional examination. The size of the system's energy store defines an upper-bound on the number of consecutive operations that can be done without harvesting additional energy from the environment. Ultimately, this serves as a limit on the designs maximum capacity for concurrently requested operations, including cryptographic operations, forcing either a limitation on its expected performance or an increase in the size of the energy store. By precomputing elements of necessary cryptographic operations as *coupons*, it is possible to transform a portion of this energy storage requirement to a data storage requirement. As discussed in the Background, modern non-volatile storage technologies such as FRAM provide a more efficient and stable storage medium for data than current battery or supercapacitor technologies provide for energy.

The transformation of energy to *coupons* for future use allows us to exploit the improved data storage capacity of modern energy harvesting systems and improve the runtime performance of our cryptographic operations. This is illustrated in Figure 2.4 which highlights the

potential to improve the performance of an energy harvested device, measured in completed operations per second.

The solid line represents the operation of a system without precomputation, with a maximum value, where the number of operations executed per second is limited by the execution latency (delay, D_o) of the operation. With a precomputation method in place, the new theoretical maximum number of operations per second is limited by the delay of the runtime only computation, D_r, which may be orders of magnitude shorter than the original operation depending on the algorithm. In reality, the theoretical limit, the dotted line, will not be reached since it requires an infinite number of precomputed *coupons*. Instead the device will operate within the highlighted area between the lower bound of operations lacking any precomputation and an upper bound where all operations have been precomputed, changing position depending on the number of *coupons* the device was able to generate and store during periods of excess energy availability.

The inflection points for the two bounds are the points at which the available power, P, is equal to the energy required for an operation divided by the operation's delay. This is the point at which sufficient power is available for the system to run the operation continuously and the limiting factor changes from power to latency. The points are highlighted in Figure 2.4 as $\frac{E_o}{D_o}$ for the original operation and $\frac{E_r}{D_r}$ for the runtime operation with *coupons*.

2.3.5 Effect of Precomputation on Security

The security of the device is also improved through the implementation of a *coupon* precomputation scheme. As previously discussed the energy required for the completion of a cryptographic operation and the actual number of processor cycles needed to complete an operation are reduced when compared to a normal operation. This has side effects includ-

ing reduced latency as observed by the distant end of communications, reduced emanations susceptible to side-channel analysis, temporal separation of data dependent operations, and improved resilience to denial of service attacks.

Denial of Service

In all cases, the device is still susceptible to an adversary denying its operation through physical destruction or disconnection. If no energy is available to the energy harvester, then no operations will be completed with or without a precomputation scheme in place. However, with a precomputation scheme in place, the device will recover from such an attack faster if any *coupons* remain in non-volatile memory from before such an attack began. In this work, we assume such *coupons* are still valid since they are stored on-chip and therefore would require an adversary well outside our threat model to effectively access and compromise these coupons without destroying the device. Effectively, such a denial of service attack is only a threat to the availability of *coupons* but not a threat to their integrity or confidentiality. This is still an improvement over a non-precomputed case since work can resume more quickly once the device is available.

Temporal Separation of Data Dependent Operations

For some cryptographic operations, a *coupon* precomputation scheme can temporally separate data dependent operations. If a key schedule is computed as a coupon, it is more difficult for an adversary to determine when this is occurring and attempt to observe the device. Similarly, in our first case, study we show that AES-CTR can be precomputed up to the one-time pad (OTP) byte stream to be XOR'd with input data. This limits an attacker to observing only the interaction of the attacker provided input and the OTP byte stream

rather than the entire AES-CTR operation. To bypass this, an attacker must now determine when *coupons* are being created and which specific *coupon* is being processed to observe the activity.

Reduced Risk of Side Channel Leakage

Precomputing brings two advantages from the perspective of side-channel attacks. First, the reduction in cycle count for the runtime operation increases the difficulty for an attacker to properly identify the effects of the cryptographic operation on the device's side channels. Second, precomputing allows uncoupling the generation of keystreams from their usage. Device-level master secrets will ideally only be accessed during the precomputation phase, and the device will not generate external input/output operations during that time. This eliminates straightforward differential power analysis. And by using only precomputed keystreams during the online phase, differential power analysis becomes harder for the online phase as well.

Reduced Operational Latency

By reducing the operational latency of our device we further limit attackers in their ability to hijack communications or protocols dependent on the completion of cryptographic operations. Communications with a device utilizing precomputation can utilize larger key sizes or stronger ciphers that are more resistant to compromise than those available to a device unable to precompute portions of its cryptographic operations. For example, in our TRNG case study, we demonstrate the dramatic reduction in runtime latency, over 2000 times faster, to access a 256-bit random value when a *coupon* is used compared collecting the necessary entropy via oscillator jitter at runtime. This is an extreme case, but any level of improvement

can be directly applied to an increased computational complexity in the security protocol employed for the device, providing a proportional amount of increased protection against attacks.

2.4 Case Studies

The following case studies examine the effects of precomputation on two cryptographic primitives. First, we analyze the precomputation of *coupons* for the key schedule and OTP for AES in Counter mode (AES-CTR) and the benefit they bring to the execution of the runtime encryption. Second, we analyze a true random number generator as one of the best cases for the precomputation of *coupons*. Energy, delay and cycle count measurements from the two case studies are for generating cipher text or a random number, the case studies do not include measurements for the communication overhead which would appear in a remote energy harvested node.

Table 2.2: Key features of MSP430FR5994 and MSP432P401R

Features	**MSP430FR5994**	**MSP432P401R**
Core	16 bit RISC	32 bit ARM Cortex M4
Memory	8kB SRAM	up to 64kB SRAM
NVM	256kB - FRAM	256kB - Flash
AM [1] current	100 μA/MHz	80 μA/MHz
HW accelerators	AES/CRC/MPY	AES/CRC
Operating mode	AM, various LPM[2]	AM, various LPM
DMA	3-channel	8-channel

[1] AM : Active mode
[2] LPM : Low Power Mode

2.4.1 Experimental setup

We have used the Texas Instruments(TI) MSP430FR5994 and the TI SimpleLink MSP432P401R launchpad development kits in our case studies. Different styles of TRNG were implemented on the MSP430FR5994 and AES-CTR mode was implemented on both the devices. Table 2.2 lists some important features that make the selected devices ideal to be used as an energy harvested node. The code was developed using Code Composer Studio (CCSv7) and the energy profile was measured using the integrated EnergyTrace technology. The principle of energy measurement of EnergyTrace is based on counting charge cycles of a switched-mode power-supply [10]. The two devices have specialized debug circuitry to work with EnergyTrace.

2.4.2 AES counter mode

AES as a block cipher can be used in different modes of operation to encrypt messages that are longer than one block of data. In AES-CTR, a counter value is encrypted first. The encrypted counter value, also known as OTP, is then XOR'd with the message block to generate the cipher text. Decryption proceeds by XORing again with a synchronized keystream. In AES-CTR mode, the actual block cipher operation is independent of the input message, making it a good candidate for parallelizing the encryption/decryption process. Similar to how the key schedule of one block of AES can be precomputed offline [17], OTPs in AES-CTR can also be precomputed offline. Figure 2.5 shows the two inputs needed for offline encryption, $\mathcal{E}_K$, are key ,K, and counter value ,IV. When a message ,m_n, is available at runtime, it can be XOR'd with the precomputed OTP which provides the resultant cipher text ,c_n, . Based on these features AES-CTR was chosen to demonstrate how precomputing can optimize both, the energy required at runtime and latency of the algorithm.

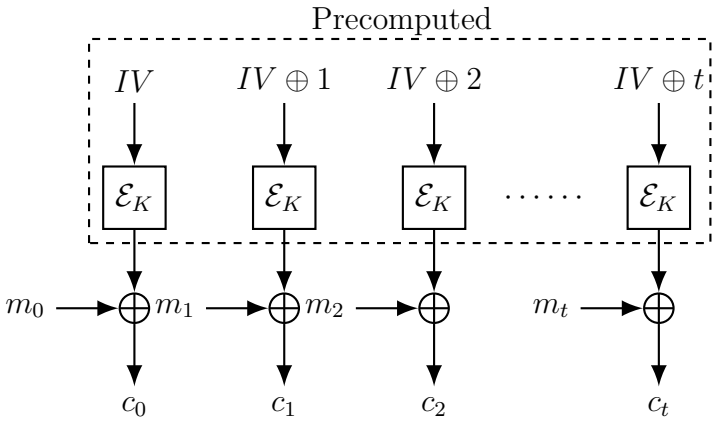

Figure 2.5: Block diagram of counter mode operation [15] with precomputable portion highlighted.

Since both the chosen microcontrollers have a dedicated AES encryption and decryption co-processor, we have chosen to experiment on both software and hardware implementations of AES. TI provides a C library for 128-bit encryption and decryption which was incorporated along with the hardware AES module in AES-CTR mode. We also implemented AES-CTR mode using a software implementation of T-box based encryption on the MSP432 [11]. In the following experiments, we have considered a 128-byte message (8 blocks of 16 bytes each) to be encrypted using a 128-bit key.

AES-CTR as a monolithic block

When no precomputation is involved, whole encryption of the message using AES-CTR mode would be performed at runtime. This requires a node to perform the code sequence in Figure 2.6 to encrypt a message.

The *aes_encrypt()* function first performs key expansion and then encrypts the counter for every block of message.

When the whole encryption is done in one online stage, we measured a delay of 6055

```
1    char *aes_ctr_monolithic(char *key, char *ctr,
2        char *PT) {
3      while(blocks < 8) {
4        aes_encrypt(char *ctr,char *key);
5        increment_counter(char *ctr);
6        xor_mask(char *PT, char *OTP, char *CT);
7      }
8      return CT;
9    }
```

Figure 2.6: Pseudo-code for Monolithic AES-CTR

Table 2.3: Cost of Monolithic AES-CTR encryption

Device	**Test case**	C_o $Cycles$	E_o μJ	D_o μs	EDP_o $10^{-12}J$
MSP432	SW T-box	18474	75.0	6055	454125.0
	SW S-box	94981	384.2	31405	12065801.0
	HW	10995	44.6	3605	160783.0
MSP430	SW S-box	153989	244.4	165746	40508322.4
	HW	13043	17.8	12370	220186.0

μs to finish encrypting a 128-byte block using T-box implementation of software AES in MSP432P401R (Table 2.3). This delay is proportional to the latency of algorithm at runtime.

AES-CTR with precomputation

The above program in Figure 2.6 is optimized by precomputing the functions *aes_encrypt()* and *increment_counter()* in the offline stage. Precomputed OTPs can then be stored as *coupons* in non-volatile memory such as FRAM in MSP430FR5994 or flash in MSP432P401R. The AES block cipher operation is then confined to the offline stage and removed from the critical path of the online process. The only remaining function to be executed during runtime is *xor_masking()*, as shown in Figure 2.7, which greatly reduces the runtime energy

```
1    char *aes_ctr_online(char *PT,
2        char *precomp-coupons) {
3      while(blocks < 8) {
4         xor_mask(char *PT, char *precomp-coupons,
5              char *CT);
6      }
7      return CT;
8    }
```

Figure 2.7: Pseudo-code for precomputed AES-CTR

Table 2.4: Runtime Cost of AES-CTR with precomputed OTP

Device	**Test case**	C_r $Cycles$	E_r μJ	D_r μs	EDP_r $10^{-12}J$
MSP432	XOR masking	3455	13.8	1105	15249.0
MSP430	XOR masking	6904	8.7	6312	54914.4

requirement.

Table 2.4 gives a clear picture of the cost of XOR masking in both MCUs. Since AES block cipher operations are precomputed in the offline stage, the runtime latency arises from retrieving precomputed *coupons* from the non-volatile memory and XORing the plain text message with those *coupons*. The energy required for fetching *coupons* and XOR masking in MSP432P401R is 13.8 μJ.

Table 2.5: Improvements in AES-CTR with precomputation

Device	**Test case**	$\frac{C_o}{C_r}$	$\frac{E_o}{E_r}$	$\frac{D_o}{D_r}$	$\frac{EDP_o}{EDP_r}$
MSP432	SW T-box	5.4	5.4	5.5	29.8
	SW S-box	27.5	27.8	28.4	791.3
	HW	3.2	3.2	3.3	10.5
MSP430	SW S-box	22.3	28.1	26.3	737.7
	HW	1.9	2.1	2.0	4.0

Discussion

By partitioning the AES-CTR algorithm, it can be optimized for latency and energy. Excess energy from the harvester can be utilized for precomputing OTPs which are needed for XOR masking. This precomputation can be continued as long as there is excess energy to compute OTPs and memory available to store them. Even if only 10 % of non-volatile memory is allocated for *coupon* storage, both devices can store almost 25.6kB of *coupons*. When the MSP432P401R is programmed to encrypt messages in AES-CTR mode using a software S-box implementation, it can store 1600 OTPs, enough to encrypt the same number of message blocks with a latency reduction by a factor of 27.5 for each message encryption. Instead of consuming 76.9 mJ of energy for encrypting 1600 blocks (monolithic encryption), a precomputed algorithm would require only 2.76 mJ of energy at runtime to compute the same amount of cipher text. This energy consumption improvement from precomputation, a factor of 28, could be utilized to reduce the required size of attached energy storage or allow more executions per charge. These values are also applicable for the decryption process as AES-CTR works in the same way for both encryption and decryption. From a security point of view, the encryption/decryption operations performed using precomputed OTPs are protected from side-channel analysis since the AES computations are performed during an offline stage. Power traces of the online stage will not reveal any information related to the key or counter value.

It can be seen that there is a vast improvement in runtime latency, energy requirement and security in AES-CTR mode when OTPs are precomputed. The EDP improvement for the AES-CTR implementations using hardware co-processors is lower than other implementations listed in Table 2.5. This is because the hardware co-processors are already optimized and they do not contribute to much of the energy and delay values of the algorithm.

Table 2.6: TRNG Structures and Labels

Label	**Structure**
`osc_clksft`	Oscillator jitter with clock frequency shifting
`osc_noclksft`	Oscillator jitter with a Von Neumann extractor and XOR compression
`sram_aes`	SRAM values processed with a HW AES coprocessor
`sram_swaes`	SRAM values processed with a SW AES implementation
`sram_sha256`	SRAM values processed through a SHA256 hash function
`sram_xor16cvn`	SRAM values processed with a 16 to 1 XOR and a Von Neumann extractor
`sram_xor32cvn`	SRAM values processed with a 32 to 1 XOR and a Von Neumann extractor

Table 2.7: TRNG Measurements and Precomputation

RNG Structure	**Monolithic Computation** C_o *cycles*	E_o μJ	D_o μs	EDP_o $10^{-12}Js$	**Improvement with Precomputation** $\frac{C_o}{C_r}$	$\frac{E_o}{E_r}$	$\frac{D_o}{D_r}$	$\frac{EDP_o}{EDP_r}$
`sram_aes`	142285	81.7	94.0	7680.9	209.6	118.2	132.6	15680.2
`sram_swaes`	178747	118.9	130.0	15462.7	263.3	172.1	183.5	31566.6
`sram_xor16cvn`	251140	196.8	301.3	59295.8	369.9	284.8	425.0	121050.5
`sram_xor32cvn`	450498	382.7	406.8	155692.7	663.5	553.8	573.9	317841.6
`sram_sha256`	1791832	1677.2	1752.4	2939160.5	2638.9	2427.2	2472.1	6000200.7
`osc_clksft`	9603395	3131.6	1709.0	5352070.4	14143.4	4531.9	2410.9	10926077.8
`osc_noclksft`	3233803	2955.9	3241.5	9581641.5	4762.6	4277.6	4572.8	19560609.8

Precomputation	C_r	E_r	D_r	EDP_r
`Read from FRAM`	679	0.691	0.709	0.490

2.4.3 Hardware Random Number Generator

This case study analyzes a true random number generator as a possible best case situation for the precomputation of *coupons*. An RNG is a possible best case example because all random number generation can be completed and securely stored before it is required by a runtime operation. This generally reduces the request for a random value to a single memory access to retrieve the next pre-generated random number. We implement two different styles of TRNG on an MSP430FR5994 one which derives entropy from the jitter between two on-chip oscillators and one which extracts entropy from the start-up values of an 8 kB SRAM. For all examples considered in this case study, the random number generators were used to generate a 256-bit random value stored in non-volatile memory (FRAM).

Generator Structure

The first type of TRNG implemented was an oscillator based RNG constructed on an MSP430FR5994 following the recommendations from Texas Instruments [30]. This oscillator based TRNG generated a random value based on the jitter between two separate oscillators, the very-low-frequency oscillator (VLO) and the digitally controlled oscillator (DCO), and included a number of techniques to avoid any bias that might be present on the device and influence the resulting random value. The second TRNG constructed was SRAM based, and extracted a random value from the startup state of the MSP430FR5994's 8kB SRAM. A number of different techniques were measured for their energy and latency efficiency when extracting a random value from the startup state of the SRAM. In all cases, the resultant random values were tested with the NIST Statistical Test Suite to validate the randomness the results [5]. Table 2.6 identifies the specific TRNGs used in the case study and the label associated with that TRNG's results throughout our collected data.

True Random Number Generation Without Precomputation

In normal operation, when a program requests a random value execution is handed off to a TRNG process or a cryptographically secure pseudorandom number generator (PRNG) that has been seeded with a truly random value of sufficient entropy. This process then generates the random value to provide to the requesting program. Depending on the implementation, the TRNG may block execution until sufficient entropy is harvested from the environment or a computation completes. The oscillator based TRNGs tested here would work very well in this style of implementation. They are able to generate an arbitrary number of random bits, simply requiring a longer collection time for larger bit strings. The SRAM based TRNGs are more difficult to employ in this manner because the device must be turned off or placed in a Low Power Mode, which removes power from the SRAM modules, in order to collect additional entropy. This places an additional delay burden on the non-precomputed versions of the SRAM based TRNG implementations that is not reflected in our results. If included this delay would only serve to further amplify the benefits of precomputation for this structure of TRNG.

Random Number Generation With Precomputation

When precomputation is available to an energy harvested system, the energy, and latency cost of random number generation is reduced to a non-volatile memory access to retrieve the next viable random number. For the MSP430FR5994, we calculated 679 clock cycles were required to copy a 32-byte, 256-bit, value from FRAM to SRAM, requiring 0.691 μJ of energy, and causing a delay of 0.709 μs. This is a multiple order of magnitude improvement for all of the TRNG implementations, in line with the dramatic reduction in complexity and difficulty when changing the operation to a simple memory access and copy. Copying

the data from the *coupon* into SRAM was chosen as the precomputed case because it was a representative of another operation accessing the random value in FRAM, via a normal extended memory access, and writing a value into SRAM for use in any application specific operations.

Discussion

This case shows the best possible situation for the precomputation of a cryptographic operation when excess energy is readily available. The algorithm does not need to be partitioned as all operations except reading the result can be executed during the precomputation and stored as a *coupon*. Additionally, the algorithm can be executed as often as possible until the data storage area for *coupons* is filled.

Given these favorable conditions, it is not surprising that the improvements seen between the monolithic operation and the runtime operation are tremendous. Depending on the speed and resources required by the specific TRNG structure we observed multiple order of magnitude improvements in latency and energy required to produce a 256-bit random value. For an energy harvested device, computing strong random values as *coupons* during periods of excess energy will dramatically improve the rate at which cryptographic operations can be executed during runtime operations. Additionally, by precomputing random values, it is much easier to exploit more efficient entropy sources such as SRAM startup values that are otherwise awkward or impossible to access in the middle of a larger computation.

It should be noted that the methods reviewed in this section were for true random number generators and did not specifically address PRNG techniques. It is possible to construct a hybrid PRNG for a system that uses one of the analyzed TRNGs to generate a seed value and then executes a less energy intensive computation for each iteration of the PRNG. Ultimately,

this technique would still benefit from precomputation and would also result in an energy and latency cost equivalent to a single pointer update after the use of a *coupon*. Due to the similarity of these results, we have highlighted only the TRNG case in this study.

2.5 Future Work

Developing a standard method for the identification of precomputable algorithms used within the IoT is a clear next step in our work. Additionally, exploration of the extent to which our precomputation methods can be combined with developments from the intermittent computing research to create an IoT device that behaviors favorably in all conditions would provide additional insight into the optimization of cryptographic operations in this realm. A detailed study of the effects *coupon* computation has on the resistance of IoT devices to side channel analysis would also strengthen our understanding of these techniques and the level of security improvement they provide. Analysis of *coupon* storage costs is also necessary before implementation in production systems. Finally, it is critical to define the points at which precomputation is not worthwhile for future developers to bracket their operations and ensure future devices are always executing in the most efficient manner.

2.6 Conclusion

This chapter presented an effective method for exploiting the excess energy available to energy harvested devices to improve the efficiency of cryptographic operations. We explored the underlying concepts of this method, the conversion of excess energy into *coupons* via precomputation, and the utilization of *coupons* to improve the efficiency of cryptographic operations executed at a later time. The security benefits of precomputation were iden-

tified and explored as a countermeasure against hardware attacks made at runtime on an IoT device. Finally, we demonstrated the effectiveness of this method with two different cryptographic operations, AES-CTR, and a true random number generator, as concrete examples of the energy efficiency improvements available to energy harvested systems when precomputation is employed to exploit their access to excess energy.

Acknowledgement

The authors like to thank the anonymous reviewers for their valuable comments and helpful suggestions. This work is made possible in part by the the Semiconductor Research Corporation under Task No.: 2712.019 and the National Science Foundation under Grant No.: 1704176.

Bibliography

[1] Giuseppe Ateniese, Giuseppe Bianchi, Angelo Capossele, and Chiara Petrioli. Low-cost standard signatures in wireless sensor networks: a case for reviving pre-computation techniques? In *Proceedings of NDSS 2013*, 2013.

[2] Giuseppe Ateniese, Giuseppe Bianchi, Angelo T. Capossele, Chiara Petrioli, and Dora Spenza. Low-cost standard signatures for energy-harvesting wireless sensor networks. *ACM Trans. Embed. Comput. Syst.*, 16(3):64:1–64:23, April 2017.

[3] Aydin Aysu and Patrick Schaumont. Precomputation methods for faster and greener post-quantum cryptography on emerging embedded platforms. Cryptology ePrint Archive, Report 2015/288, 2015. http://eprint.iacr.org/2015/288.

[4] 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. *IEEE Embedded Systems Letters*, 7(1):15–18, March 2015.

[5] Lawrence E. Bassham, III, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Elaine B. Barker, Stefan D. Leigh, Mark Levenson, Mark Vangel, David L. Banks, Nathanael Alan Heckert, James F. Dray, and San Vo. Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Gaithersburg, MD, United States, 2010.

[6] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David B. Wilson. *Fast Exponentiation with Precomputation*, pages 200–207. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[7] Michael Buettner, Richa Prasad, Alanson Sample, Daniel Yeager, Ben Greenstein, Joshua R. Smith, and David Wetherall. Rfid sensor networks with the intel wisp. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 393–394, New York, NY, USA, 2008. ACM.

[8] Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. Bespoke processors for applications with ultra-low area and power constraints. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 41–54, New York, NY, USA, 2017. ACM.

[9] Alexei Colin and Brandon Lucia. Chain: Tasks and channels for reliable intermittent programs. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2016, pages 514–530, New York, NY, USA, 2016. ACM.

[10] H. Diewald, G. Zipperer, P. Weber, and A. Brauchle. Electronic device and methods for tracking energy consumption.

[11] Viktor Fischer and Miloš Drutarovský. *Two Methods of Rijndael Implementation in Reconfigurable Hardware*, pages 77–92. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[12] Matthew Hicks. Clank: Architectural support for intermittent computation. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 228–240, New York, NY, USA, 2017. ACM.

[13] J. Hsu, S. Zahedi, A. Kansal, M. Srivastava, and V. Raghunathan. Adaptive duty cycling for energy harvesting systems. In *ISLPED'06 Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, pages 180–185, Oct 2006.

[14] H. Jayakumar, A. Raha, and V. Raghunathan. Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 330–335, Jan 2014.

[15] Jérémy Jean. TikZ for Cryptographers. https://www.iacr.org/authors/tikz/, 2016.

[16] Aman Kansal, Jason Hsu, Mani Srivastava, and Vijay Raghunathan. Harvesting aware power management for sensor networks. In *Proceedings of the 43rd Annual Design Automation Conference*, DAC '06, pages 651–656, New York, NY, USA, 2006. ACM.

[17] Bin Liu and Bevan M. Baas. Parallel aes encryption engines for many-core processor arrays. *IEEE Trans. Computers*, 62:536–547, 2013.

[18] Brandon Lucia and Benjamin Ransford. A simpler, safer programming and execution model for intermittent systems. In *Proceedings of the 36th ACM SIGPLAN Conference*

on Programming Language Design and Implementation, PLDI '15, pages 575–585, New York, NY, USA, 2015. ACM. DINO.

[19] Maxwell Technologies. *Datasheet: HC Series Ultracapacitors*, 2013.

[20] Philippe Oechslin. *Making a Faster Cryptanalytic Time-Memory Trade-Off*, pages 617–630. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[21] A. Orda and A. Sprintson. Precomputation schemes for qos routing. *IEEE/ACM Transactions on Networking*, 11(4):578–591, Aug 2003.

[22] S. Pelissier, T. V. Prabhakar, H. S. Jamadagni, R. VenkateshaPrasad, and I. Niemegeers. Providing security in energy harvesting sensor networks. In *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 452–456, Jan 2011.

[23] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: System support for long-running computation on rfid-scale devices. *SIGARCH Comput. Archit. News*, 39(1):159–170, March 2011.

[24] Mastooreh Salajegheh, Shane S Clark, Benjamin Ransford, Kevin Fu, and Ari Juels. Cccp: Secure remote storage for computational rfids. In *USENIX Security Symposium*, pages 215–230, 2009.

[25] Weidong Shi, H. S. Lee, M. Ghosh, Chenghuai Lu, and A. Boldyreva. High efficiency counter mode security architecture via prediction and precomputation. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 14–24, June 2005.

[26] Farhan Simjee and Pai H. Chou. Everlast: Long-life, supercapacitor-operated wireless sensor node. In *Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, ISLPED '06, pages 197–202, New York, NY, USA, 2006. ACM.

[27] Chester Simpson. *Characteristics of Rechargeable Batteries*. Texas Instruments, 2011.

[28] Shan Sun and Scott Emley. *Data Retention Performance of 0.13-um F-RAM Memory*. Cypress Semiconductor Corp., 7 2015. Rev. *A.

[29] Patrick P. Tsang and Sean W. Smith. *Secure Cryptographic Precomputation with Insecure Memory*, pages 146–160. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[30] Lane Westlund. Random number generation using the msp430, 2006.

Chapter 3

Secure Intermittent Computing: Protecting State Across Power Loss

3.1 Introduction

In a conventional embedded system, power loss results in the loss of volatile state variables, and when power is restored the system reboots from the initial state. A more recent development in embedded computing, called intermittent computing, recognizes power loss as a fact of life in applications that use ultra-low power computers and energy-harvesting power sources [2, 11, 14, 16]. Such systems create *checkpoints*, snapshots of the intermediate program state. When the power is restored, the program state is reconstructed from the checkpoint and the program continues execution. Emerging non-volatile memory technologies enable the practical realization of intermittent computing scenarios and a broad range of instant-on applications.

Intermittent computing systems support long-running computations and computations that use a long-lived dynamic state. They deal with unreliable power sources such as harvested energy from solar, thermal, vibrational or electromagnetic sources [19, 21]. Intermittent computing opens up new applications for tiny computing platforms operating on such unstable energy sources. In the domain of cryptographic engineering, examples of long-running applications that may benefit from this operational mode include the creation of public-key

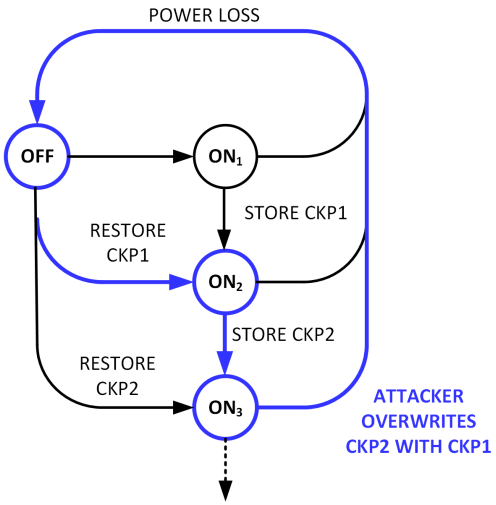

Figure 3.1: State diagram of intermittent computing system under replay attack. An adversary is able to repeatedly overwrite the new `CKP2` with the stale `CKP1` forcing continued re-execution of the code between ON_2 and ON_3.

key pairs and the generation and testing of high-quality entropy. Intermittent computing is also a suitable solution for cases that use long-lived dynamic state, such as stateful hash based signatures [18] or key-rolling encryption applications [26].

Risks of Checkpoints Current state-of-the-art literature in intermittent computing does not address the security requirements of checkpoint data [2, 11, 14, 16]. However, if we treat power loss as an adversarial event, it is clear that unprotected checkpoints pose a significant risk. First, when the *same* checkpoint can be restored infinitely, the adversary can use power loss as an effective method to force repeated execution of the same section of code. This is helpful to design implementation attacks, including fault injection and side-channel measurement campaigns. Figure 3.1 illustrates this situation. A system moves through a sequence of activities symbolized through ON-states. During the ON-state, a checkpoint is computed and stored in non-volatile memory. When power is restored subsequent to a power loss, the system resumes operation from the most recent checkpoint. If an adversary records and restores an old checkpoint, the same functionality is repeated.

Second, when the adversary has direct access to the checkpoint data, the checkpoint may reveal a snapshot of the entire internal state, leading to loss of confidentiality. The checkpoint data can be accessed by the adversary, for example, through the microcontroller debug interface. The inspection of a checkpoint in this manner is similar to a cold-boot attack [10]. Another significant risk is that a checkpoint also contains the control flow state, including the program counter and the stack pointer. Tampering with the checkpoint may enable the attacker to grab control over the embedded system after power is restored. Therefore, checkpoint data has a confidentiality as well as an integrity requirement.

Objectives In this chapter we propose the Secure Intermittent Computing Protocol, SICP, to address these risks. We describe the objectives of the SICP using Figure 3.2. A microcontroller runs on an intermittent power supply and may need to be turned off every now and then. To ensure continuity after power is restored, the microcontroller creates checkpoints and stores these checkpoints in non-volatile memory. The precise assumptions on the attacker capabilities are described further in the chapter. For Figure 3.2, it suffices to understand that checkpoint data could be observed or tampered by the attacker. First, SICP creates encrypted checkpoint data using a scheme that provides integrity, confidentiality, and authenticity. Second, SICP offers checkpoint freshness. SICP ensures that every checkpoint can only be restored once, and that replay can be detected by the microcontroller application. Following Figure 3.2, after checkpoint `CKP1` is created, the only valid checkpoint that can be restored is `CKP1`. Furthermore, the internal design of SICP allows `CKP1` to be restored successfully only once. Third, SICP creates and restores checkpoints atomically. SICP is protected against power loss during checkpoint generation and restoration. Finally, SICP provides application continuity. All checkpoints are cryptographically linked, ensuring every checkpoint represents the entire history of the long-running application over all possible power interruptions.

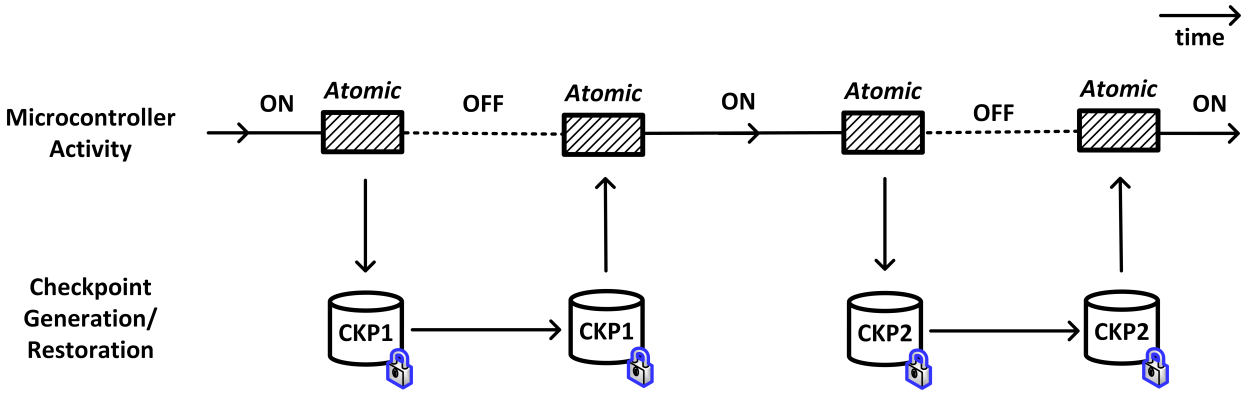

Figure 3.2: Illustration of the basic requirements for secure checkpoint generation and restoration. The operations to create and restore checkpoints must be atomic to be resilient to power loss during their execution, and when properly implemented will provide continuity for the underlying application through periods of power loss.

Related Work We now differentiate the unique properties of SICP from other work and discuss several concepts related to this work. So far, none of the intermittent computing proposals has considered checkpoint security [11, 14, 16, 23]. Some authors have proposed in-band memory encryption techniques for non-volatile memories, including i-NVMM [7] and sneak-path encryption [15]. These techniques introduce a continuous encryption overhead, which cannot be handled by resource constrained devices, and they ignore the specific attack vector, power loss, targeted by our approach. One recently proposed scheme introduced encryption of stored checkpoints in intermittent computing systems [9], but that proposal only considers confidentiality, and does not offer replay detection.

Power loss is also a design consideration in smart card environments. Smart card tearing is the removal of a smart card from a card reader mid-operation, potentially leaving the smart card in an inconsistent state [13]. Platforms such as Java Card provide transactional semantics to ensure that critical operations are cleaned up after an abrupt power loss [1]. However, the solutions to smart card tearing are not applicable to intermittent systems, since smart cards do not create checkpoints. While SICP relies on similar transactional semantics as used against smart card tearing, the context is very different.

We are not aware of efforts in the area of trusted embedded computing which consider power loss, or which consider maintaining trustworthiness features across power cycles [17]. Trusted computing environments provide strong guarantees on the integrity of code execution and the data stored in memory, as long as power is available. Power loss implies the loss of the trustworthy data and control flow integrity information. For example, SMART, a primitive that offers attestation of dynamic program state, explicitly states that memory has to be cleared after every reboot [8]. The attestation data of SMART cannot be carried over across power loss. We argue that SICP offers a solution to save and restore critical program information in those environments.

Contributions The contributions of our work are as follows.

- We propose the Secure Intermittent Computing Protocol, referenced as the SIC Protocol or SICP. The SIC Protocol protects checkpointed program state in intermittent computing systems. SICP itself is resistant to power loss attacks, through atomic checkpoint generation and restoration.

- SICP maintains application continuity across power loss such that every checkpoint is a reflection of the control flow of the entire program. SICP introduces freshness and ensures the information security of every checkpoint.

- We demonstrate SICP through an implementation on an MSP430 microcontroller and we present a preliminary performance evaluation.

Organization Section 3.2 enumerates four essential properties achieved by SICP. Section 3.3 explains the design and operation of SICP, while Section 3.4 describes its detailed implementation. Section 3.5 demonstrates the feasibility of the SIC Protocol through a pro-

totype implementation on an MSP430 microcontroller, followed by a detailed evaluation in Section 3.6. We conclude with Section 3.7.

3.2 Essential Properties

We identify four fundamental properties that will be achieved by the SIC Protocol.

Continuity Application continuity is the assurance that an application will resume execution from where it left off after a power loss without modification. Lacking this assurance, existing security properties fail to provide value to an intermittent system. We utilize cryptographic chaining of checkpoints to meet the continuity requirement, which preserves the security properties from previous on-states and carries them forward to future on-states.

Atomicity An atomic operation is one that either finishes execution or fails with no effect on the system. To prevent unexpected behavior and potential security shortfalls within our protocol, we require that each portion of the protocol's operation be atomic in nature. At a low level, this requirement is served by the byte-level write atomicity of our chosen hardware [29] and prevents inconsistent state within the non-volatile memory. At a higher level, we utilize multiple stored states to ensure that updates can be completed without corrupting the current state information if power loss occurs mid-operation. Together, these capabilities ensure that each critical operation either commits fully to the state of the microcontroller or, if it is interrupted, has no effect on the operational state of the device.

Freshness By definition, freshness is something that is newly obtained or recent. Without freshness, it is not possible to identify if a datum is more or less recent than any other that

meets a system's structural requirements. Freshness is incorporated into checkpoints in the form of a nonce to differentiate checkpoints with respect to time and to ensure that if a checkpoint is restored it is indeed the latest checkpoint available to the system. Critically, uniqueness and freshness are the main requirements for the nonce used within the SIC Protocol.

Information Security Information security relates to the confidentiality, integrity, and authenticity of a system's data. For this work, these are critical properties for the protection of a system's stored checkpoints.

3.3 Design

Previously, we highlighted the lack of security in the majority of the state-of-the-art intermittent computing solutions. In this section, we define our threat model and the architectural assumptions which are followed by an overview of the protocol design.

Threat Model We assume an I/O attacker model, with two capabilities. First, the adversary has complete control of the power supply to the device. It gives the adversary the ability to arbitrarily stop execution of the target program. Second, the adversary has access to the majority of the device memory during power-off periods except for a small portion of non-volatile memory, which is tamper free. The adversary can view and modify the device memory to read secret information in checkpoints, tamper checkpoints, or replay stale checkpoints except within the tamper free region.

We assume a protected embedded software execution environment which provides execution integrity and memory protection when the device is powered on. The feasibility of this

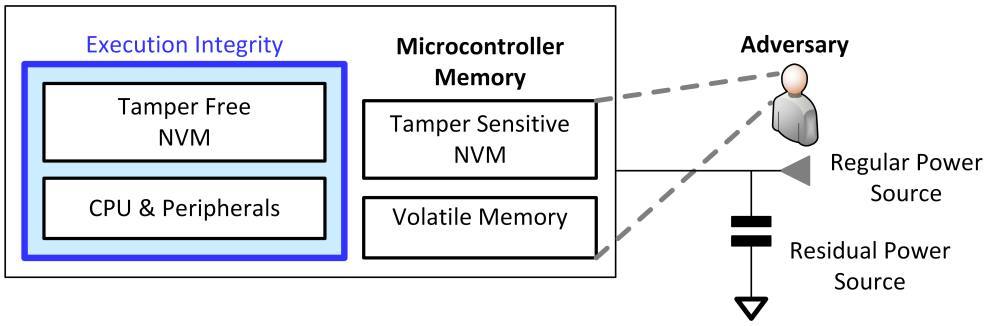

Figure 3.3: The architectural assumptions and memory model for SICP illustrating the assumed attacker model. Information stored in tamper free non-volatile memory cannot be observed by the adversary while the device is powered off and is protected by an execution integrity environment while the device is powered on.

assumption has been demonstrated by recent efforts in attestation and isolation for microcontrollers [17, 20]. We do not deal with the mitigation of side channel and fault injection attacks which is beyond the scope of this work.

Architectural Assumptions The SIC Protocol is not tied to a particular microcontroller, but, based on the above threat model, it requires certain basic capabilities from the microcontroller architecture, illustrated in Figure 3.3. The SICP architecture contains three types of memory. *Volatile memory* holds program state, and is erased upon power loss. *Tamper-sensitive non-volatile memory* stores secure checkpoints created from runtime program state. This non-volatile memory does not possess any tamper-resistance and represents the vast majority of the system's non-volatile memory. SICP also requires a small *tamper-free non-volatile memory* to store SICP variables that need tamper-free storage. At a 128-bit security level, SICP needs 48 bytes of tamper-free storage for two 128-bit nonces and a 128-bit secret key. The size of tamper free memory must be minimized to reduce hardware cost and complexity. Finally, we assume that the microcontroller has a residual power source which provides a small, finite energy supply when the regular power source is interrupted. This residual power source can, for example, be provided through power conditioning capaci-

Table 3.1: Variables Used in SICP and Their Information Security Requirements

Variable	Description	Memory Type	C[1]	I[2]	Purpose
K	Device Unique Key	Tamper Free	Yes	Yes	Secrecy
R_i	Nonce	Tamper Free	No	Yes	Freshness
S_i	Encrypted State	Tamper Sensitive	No	No	Confidentiality
T_i	Authentication Tag	Tamper Sensitive	No	No	Authenticity, Continuity

[1] Confidentiality
[2] Integrity

tors. We assume that the residual power source can be physically protected and can provide the minimum required energy to finish a write to non-volatile memory and wipe sensitive program state.

Table 3.2: Building Blocks of SICP

Function	Purpose
factory_reset()	Restore device to manufacturer setting and load key, K
nonce()	Generate a unique and fresh nonce
abort()	Flag a violation of the SIC Protocol
$AEAD_{encr}(P, D, N, K)$	Encryption function to generate ciphertext, C
$AEAD_{auth}(C, D, N, K)$	Authentication function to compute the authentication tag, T
$AEAD_{decr}(C, T, D, N, K)$	Decryption function to generate plaintext, P

3.3.1 SICP Operation

This section provides a brief overview of the SIC Protocol and its major components. Table 3.1 lists all the variables that are used in the protocol. To satisfy the security goals, a nonce, R_i, is assigned to each power on-state, *STATE*, of the application. The nonce introduces freshness to the state, even if the application state is identical to a previous power cycle. It is stored in tamper free non-volatile memory to prevent checkpoint replay. To provide confidentiality, the microcontroller state is encrypted using the device unique key,

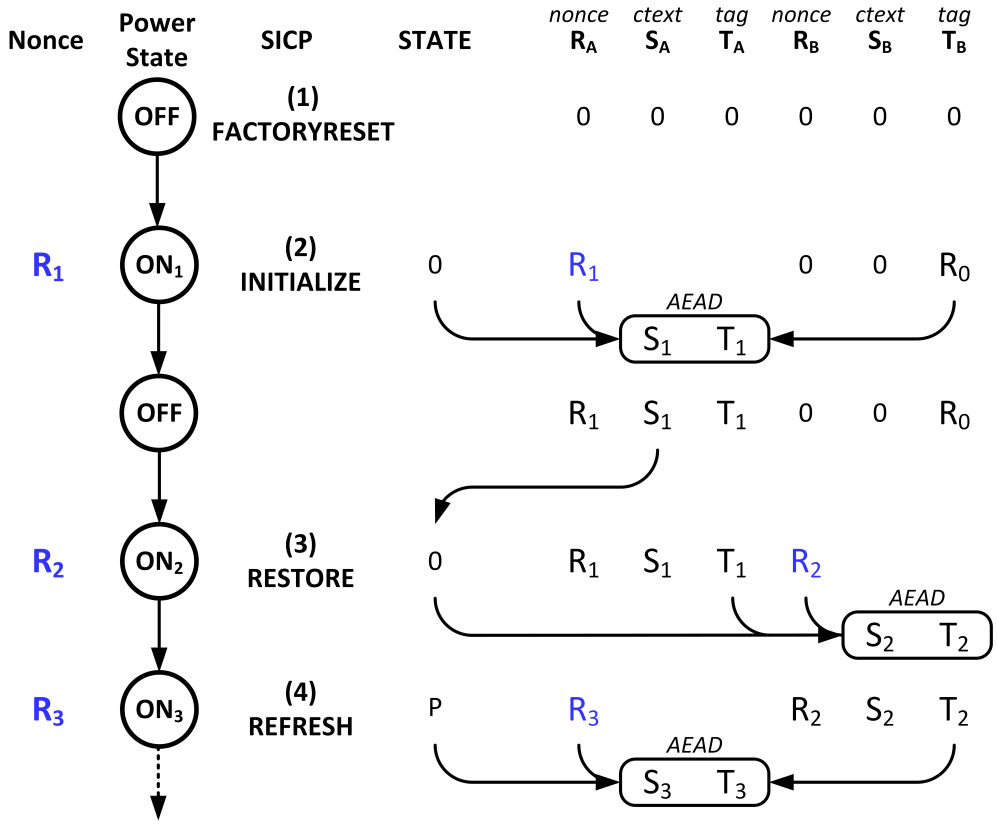

Figure 3.4: Example of the SIC Protocol. (1) The system is cleared by the *factory_reset()* operation. A fresh nonce is associated with each power-on state. (2) The first valid state save packet is SS_1, created by the *INITIALIZE* function. (3) After a power cycle, *RESTORE* validates the latest state save packet, SS_1, restores the program state, and generates a new state save packet SS_2. (4) During program execution, *REFRESH* is called to create a new checkpoint SS_3, overwriting the oldest state save packet, SS_1.

K, and stored only in ciphertext format, as S_i, within the device's tamper sensitive non-volatile memory. The authentication tag, T_i, is calculated over the encrypted microcontroller state to ensure the authenticity and integrity of the checkpoint. These tags are also used to cryptographically chain all checkpoint together in chronological order.

With this structure, every new checkpoint will contain a unique nonce, R_i, the current encrypted state of the microcontroller, S_i, and an authentication tag, T_i, generated over R_i and S_i. We call this a state save packet, SS_i. Two state save packets, SS_A and SS_B, are

required to allow atomic generation and restoration of state save packets. Keeping one packet valid at all times and updating them in an alternating manner, as shown in Figure 3.4, makes SICP resilient against power loss. We use Authenticated-Encryption with Associated-Data (AEAD) [24] to generate the state save packets, the details of which are described further in the chapter.

Continuity is achieved by using the authentication tag from the previous state save packet as the associated data in an AEAD operation to generate the next state save packet. For example, in Figure 3.4, T_1 is used as associated data to compute T_2, which in turn is used as associated data to compute T_3. This process is referred to as *tag-chaining*. The authentication tags serve multiple purposes within SICP. They provide authentication of the encrypted states, and they can be used to detect power loss and in turn detect the number of times a checkpoint was restored. This assures the system of its control flow since the previous *factory_reset()*.

Cryptographic Functions Table 3.2 enumerates the primitive functions that are the building blocks of the SIC Protocol. We use an AEAD primitive to support checkpoint information security. Given a plain text message, P, a nonce, N, non-confidential associated data, D, and a secret key, K, an AEAD scheme encrypts P into a cipher text, C, while computing an authentication tag, T, over the four components C, N, D, and K. The exact use of the associated data and nonce during the encryption process is dependent on the the choice of AEAD scheme.

$AEAD_{encr}()$ generates only the ciphertext and it does not generate the authentication tag. $AEAD_{auth}()$ generates the required authentication tag. The standard AEAD interface for encryption returns both of these values. They are separated here into encryption and tag calculation, to provide clarity in protocol description, but this protocol can be easily imple-

Algorithm 1 Initialize

1: **procedure** INITIALIZE() ▷ Create first SS_A
2: $Q \leftarrow nonce()$
3: $T_B \leftarrow nonce()$
4: $STATE \leftarrow 0$
5: $R_A \leftarrow Q$
6: $S_A \leftarrow AEAD_{encr}(STATE, T_B, R_A, K)$ ▷ T_B used as associated data
7: $T_A \leftarrow AEAD_{auth}(S_A, T_B, R_A, K)$
8: **end procedure**

mented with the standard monolithic encrypt and decrypt functions of an AEAD scheme.

3.4 SIC Protocol

In this section we provide a detailed description of the operating principle of the protocol. We also motivate our design choices. SICP is defined as a collection of four algorithms: *INITIALIZE, REFRESH, RESTORE* and *WIPE*. The algorithms are defined as follows.

INITIALIZE: This function is called the first time the device is powered on and is detailed in Algorithm 1. It is used to generate the first state save packet of the application, SS_A, following a *factory_reset()*. Since the first state save packet has no previous authentication tag to use for associated data, the tag, T_B, is initialized with a nonce before it is used to generate SS_A. This ensures a unique chain of tags after each *factory_reset()*. Finally, *STATE* is overwritten with zeros to ensure future tests for *factory_reset()* fail and *INITIALIZE* is only executed once after a *factory_reset()*.

REFRESH: Algorithm 2 defines the procedure to securely checkpoint the current state of the microcontroller. It must be called only after *INITIALIZE* has finished generating the first valid state save packet. *REFRESH* can be called at any point by the application when

Algorithm 2 Refresh

```
1: procedure REFRESH()                                   ▷ Create SS of system state
2:     Q ← nonce()
3:     if T_A = AEAD_auth(S_A, T_B, R_A, K) then          ▷ SS_A is valid
4:         R_B ← Q
5:         S_B ← AEAD_encr(STATE, T_A, R_B, K)
6:         T_B ← AEAD_auth(S_B, T_A, R_B, K)              ▷ Now, SS_B is valid
7:     else if T_B = AEAD_auth(S_B, T_A, R_B, K) then
8:         R_A ← Q
9:         S_A ← AEAD_encr(STATE, T_B, R_A, K)
10:        T_A ← AEAD_auth(S_A, T_B, R_A, K)
11:    else
12:        abort()
13:    end if
14: end procedure
```

the device is powered-on. When this algorithm is called, the microcontroller will determine which is the latest state save packet. If SS_A is currently valid, then SS_B is updated or vice versa. At the end of *REFRESH*, the microcontroller will have a new valid state save packet and the previously valid packet will be defunct.

A state save packet is valid if it satisfies the following conditions. First, its nonce, R_i, must match the nonce used in the $AEAD_{encr}$ and $AEAD_{auth}$ operations. Second, the associated data used in $AEAD_{encr}$ and $AEAD_{auth}$ must match the tag of the previously valid state save packet. This ensures that at any point, only one state save packet is valid.

For example, if SS_A was the latest packet to be refreshed, then when *REFRESH* is called again, line 3 in Algorithm 2 would be true. Correspondingly, the microcontroller will start updating SS_B by first updating R_B and then S_B. As long as T_B is not updated, SS_B does not yet contain a valid state save packet and SS_A remains valid. As soon as T_B is updated, it simultaneously invalidates SS_A and makes SS_B the latest valid state save packet. This write to T_B makes *REFRESH* atomic.

The implementation of SICP makes an explicit assumption regarding the tag update in lines 6 and 10 of Algorithm 2. The tag update must be an atomic operation. The feasibility of this assumption is discussed in Section 3.5.1.

RESTORE: *RESTORE*, which is called upon every power-up, except immediately after a *factory_reset()*, is used to restore the latest valid *STATE* of the microcontroller. This function operates in the same manner as *REFRESH* with a slight difference shown on lines 4 and 9 of Algorithm 3. The $AEAD_{decr}()$ function decrypts the ciphertext to restore the latest valid state save packet in *STATE*. The state restoration happens only if there is a valid state save packet, otherwise, the program is aborted. After restoring the valid state save packet, the other state save packet is updated with the current state, a fresh nonce, Q, and the tag generated during the creation of the latest valid state save packet. This feature can be used to keep track of the number of times the microcontroller is restored to a particular *STATE*.

If the stored state is tampered during power-off, the conditional checks on lines 3 and 8 fail, which flags a security exception and calls *abort()*. At a minimum, this function should either end program execution or restart the device.

WIPE: Wiping *STATE* is necessary to ensure that checkpoint data is not accessible to the adversary during power-off periods. Transient information such as the program variables stored as plain text in volatile and non-volatile memory should be cleared to prevent access during power-off and ensure confidentiality. The residual power source must have sufficient power to finish this operation otherwise the protocol will fail to satisfy the stated security goals. *WIPE*, which is detailed in Algorithm 4, must be called as soon as power loss is detected and the device is about to shut down, i.e. before the microcontroller stops execution.

Algorithm 3 Restore

```
1: procedure RESTORE()
2:     Q ← nonce()
3:     if T_A = AEAD_auth(S_A, T_B, R_A, K) then
4:         STATE ← AEAD_decr(S_A, T_A, T_B, R_A, K)        ▷ Restore decrypted state
5:         R_B ← Q
6:         S_B ← AEAD_encr(STATE, T_A, R_B, K)
7:         T_B ← AEAD_auth(S_B, T_A, R_B, K)
8:     else if T_B = AEAD_auth(S_B, T_A, R_B, K) then
9:         STATE ← AEAD_decr(S_B, T_B, T_A, R_B, K)
10:        R_A ← Q
11:        S_A ← AEAD_encr(STATE, T_B, R_A, K)
12:        T_A ← AEAD_auth(S_A, T_B, R_A, K)
13:    else
14:        abort()
15:    end if
16: end procedure
```

Algorithm 4 Wipe

```
1: procedure WIPE()                                        ▷ Wipe sensitive data
2:     STATE ← 0
3:     V ← 0                                               ▷ Clear volatile memory
4: end procedure
```

The function must securely overwrite the sections of non-volatile memory that contain sensitive data, *STATE*. The application programmer must identify variables in the application that will contain sensitive information and store them in a separate marked section of tamper sensitive memory on the microcontroller. As a precaution volatile memory, V, is also wiped to prevent cold-boot style attacks [10]. The specific implementation of *WIPE* will be device dependent. Section 3.5.2 outlines our implementation's approach.

3.4.1 Discussion

Simplifications to our protocol will easily result in flawed designs. We illustrate this by means of two examples. Figure 3.5(a) depicts a naive example that only stores a single

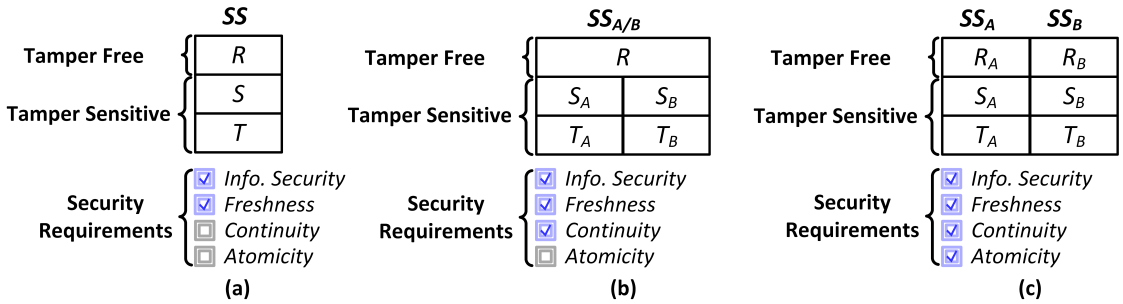

Figure 3.5: Need for two state save packets. Only (c) fully meets the atomicity and continuity security properties of SICP. (a) and (b) require less storage and may appear simpler, but they are unable to support SICP.

copy of the state save packet. This solution does not satisfy the protocol's requirements for continuity or atomicity. When the microcontroller loses power during the update of any one of the three components, R, S, or T, it is left with an inconsistent checkpoint, illustrating its lack of atomicity. It also lacks continuity because it can only store a single checkpoint and therefore cannot identify an order between checkpoints. The example in Figure 3.5(b) is an improvement over Figure 3.5(a) by providing application continuity. It contains a single copy of the nonce, but two copies each of the encrypted state and tag, enabling identification of a checkpoint order. The two copies can be updated in an alternating manner, similar to the proposed protocol. The microcontroller can identify which state and tag corresponds to the latest state save packet by matching the authentication tag with the nonce stored in R. Although it appears operational, the 3.5(b) still fails to enable atomicity in the protocol's operation. Power loss during an update of the nonce, R, or before the update of R following a tag update would leave the device with inconsistent checkpoints. Ensuring atomic protocol operations requires the use of two state save packets, as shown in Figure 3.5(c). Through the employment of two state save packets and tag-chaining, SICP is able to provide atomicity to the protocol operation and continuity to the underlying application.

3.5 Implementation

To evaluate the SIC Protocol, we establish an intermittent system and implement SICP to manage and secure the system's checkpoints. We implement a proof-of-concept solution using two different AEAD schemes, one of which is tested with both a software and a hardware implementation. It was created on an MSP430FR5994 Launchpad development board [30] with all code built against the recent `msp430-elf-gcc` compiler [25, 28].

MSP430FR5994 is equipped with non-volatile ferroelectric RAM (FRAM) and multiple peripheral modules suitable for modern embedded tasks. Additionally, this device's unified memory model, collapsing all SRAM, FRAM, and memory mapped peripherals in a single globally mapped memory, simplified the implementation of SICP by providing a common interface for all data locations.

3.5.1 Secure Intermittent Computing Support

The underlying intermittent computing solution used in the proof-of-concept is a modified version of TI's *Compute Through Power Loss* (CTPL) utility [29]. The CTPL utility was modified to compile on the `msp430-elf-gcc` compiler, to support user declared system checkpoints, and to invoke a security function within the checkpoint and startup process as necessary. Compatibility with `msp430-elf-gcc` required the modification of all preprocessing references to non-volatile data structures and recomposition of all supporting assembly code to comply with GCC's idioms rather than those of TI's `cl430` compiler.

State Save Packet Creation User declarable checkpoints require the addition of a user callable checkpoint function to the CTPL library. This function must create the checkpoint, including a copy of the program counter (PC) and stack, and properly store the checkpoint

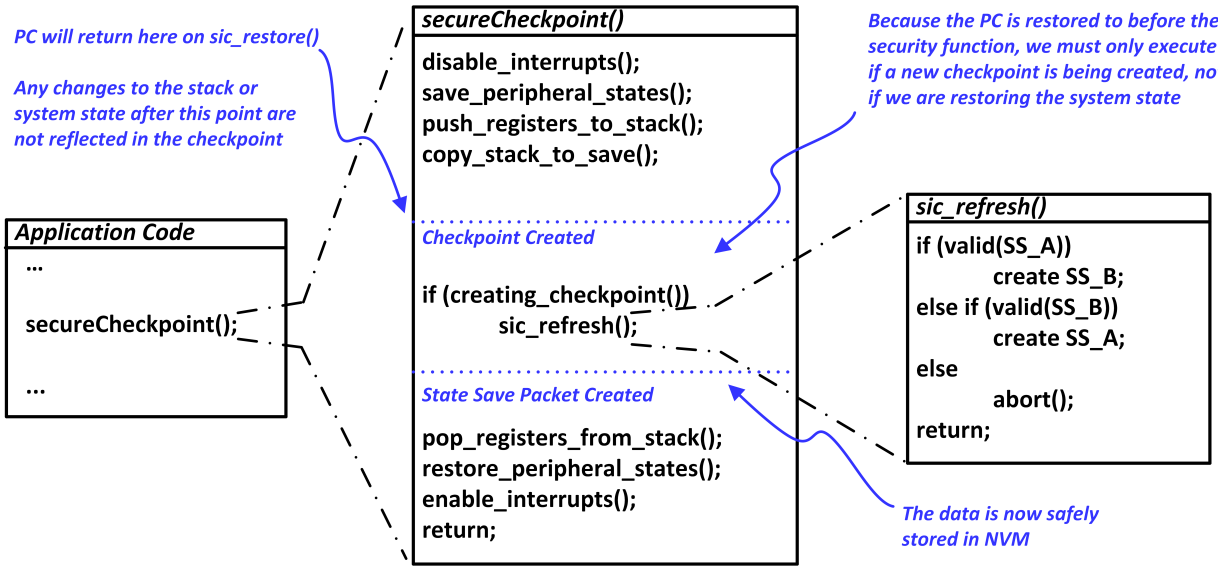

Figure 3.6: The control and data flow for the creation of a checkpoint and subsequent state save packet. Because the creation of a checkpoint mangles the stack and PC, we must skip `sic_refresh()` when a checkpoint is restored but not when one is created. This logic is shown here as `create_checkpoint()`, within the implementation it is handled by multiple conditional checks.

within a state save packet for restoration at a later time. To support this structure, the `sic_refresh()` function is constructed such that it can be called after the checkpoint is created but before the stack and associated system data are restored. This is accomplished by instantiating it as a `void` function without parameters and basing its conditional decisions solely on the information present in non-volatile memory. Figure 3.6 shows the major tasks executed while creating a checkpoint and storing it in a state save packet, it also illustrates how `sic_refresh()` and `sic_restore()` are invoked after the PC and stack are stored within a checkpoint. This prevents the security functions from relying on information passed via the stack and allows everything to be properly restored before returning execution to the application code. These modifications to the CTPL library create a utility capable of efficiently recording the system state whenever the checkpoint function is invoked.

SICP Support A simple user API, consisting of only a `secureCheckpoint()` function, is supported by the ability to call a security function within the checkpoint creation process. The platform specific implementations of the SICP functions, respectively `sic_initialize()`, `sic_refresh()`, `sic_restore()` and `sic_wipe()`, do not need to be directly invoked by the supported application. Instead, the SICP operations execute transparently when security critical data is properly annotated and the `secureCheckpoint()` function is called in locations where the developer wishes to save the system state. This is illustrated in Figure 3.6 where the application code only calls `secureCheckpoint()` which invokes `sic_refresh()` as needed. The other three SICP functions are either called automatically during system startup in the case of `sic_initialize()` and `sic_restore()`, or triggered by a detection of power loss in the case of `sic_wipe()`.

Atomicity Support The atomicity of the `secureCheckpoint()` function is ensured through the following considerations. All changes in non-volatile memory are made to locations that are ignored until the newly computed tag is ready. Once the new tag computation is complete and stored in a temporary buffer, the `sic_copyTag()` function is called to overwrite the previous tag and set the newly created checkpoint as the only valid checkpoint in an atomic operation. This is achieved by disabling all interrupts for the copy duration of 48 cycles and relying on the residual energy of the device and the FRAM's atomic byte write capability to ensure that even if power is lost, the copy operation will complete before the system stops operating. With this structure, it is possible for the developer to treat calls to `secureCheckpoint()` as atomic operation. The function either has no effect on the system, if power is lost before the tag update, or completes the checkpoint creation without incident.

The development of the SICP API is agnostic of the underlying cryptographic kernel selected to enforce the protocol's security guarantees, allowing different AEAD schemes to be

evaluated, and providing flexibility for future implementations.

AEAD Integration Two different AEAD schemes are implemented for evaluation. First, a software implementation of EAX [3], provided by the Cifra [6] cryptographic library, is constructed. EAX is a well established two-pass AEAD scheme that fits a number of development parameters for the proof-of-concept. The two-pass structure provides an opportunity to avoid unnecessary decryption operations when a tag fails authentication in `sic_refresh()` or `sic_restore()`. Tag failure occurs on half of the calls to these two functions since the state save packet authentication is used to determine which packet is valid and which should be overwritten. The block-cipher based nature of EAX enables the use of a hardware accelerated version by modifying the code to employ the MSP430FR5994's AES accelerator [27].

Second, Ketje v2, specifically Ketje Sr, from the Keccak Code Package (KCP) [4] is integrated to test an alternative lightweight AEAD scheme. Ketje Sr is an authenticated encryption scheme designed for resource constrained devices that derives its security from the underlying Keccak-f permutation. It is one of four functions of the Ketje AEAD scheme and is the larger of the two lightweight Ketje functions with a state size of 400 bits [5]. Ketje Sr provides an opportunity to observe if a one-pass AEAD scheme generates more overhead compared to the EAX implementation. Additionally, Ketje Sr complies with the CEASAR competition API, providing only a `crypto_aead_encrypt()` and `crypto_aead_decrypt()` interface.

3.5.2 System Construction

The complete implementation of SICP was accomplished by wrapping the modified checkpointing system with the secure intermittent computing functions to generate valid state

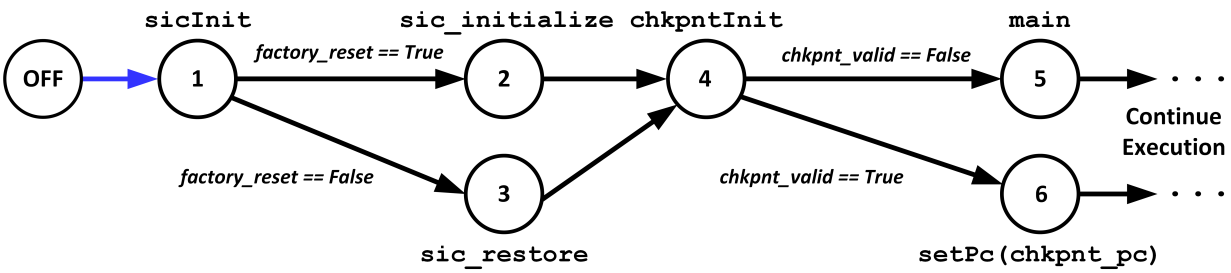

Figure 3.7: Startup sequence for a device implementing SICP. (1) SICP checks for *factory_reset()* and calls (2) `sic_initialize()` or (3) `sic_restore()` to populate *STATE* in non-volatile memory. (4) the checkpointing system inspects *STATE* for a valid checkpoint, restoring the checkpoint (6) if one is found or invoking `main()` (5) if one does not exist. Program execution will then continue normally until power is lost or another checkpoint is created.

save packets.

System Startup Figure 3.7 shows the startup process for a system employing SICP. A random number generator is used to generate the nonce for T_B during the `sic_initialize()` function. This nonce is generated randomly to ensure that no two different uses of a device create the same pattern of tags, even if the exact same code is executed following a *factory_reset()* [12, 22]. All other nonces employed throughout the protocol are provided via a 128-bit counter that is initialized to zero during the `sic_initialize()` function and incremented each time a new nonce is requested.

A portion of the non-volatile memory region containing *STATE* is then checked for the factory reset bit pattern. This is used to determine if a *factory_reset()* has occurred and `sic_initialize()` should be invoked, or a normal boot sequence with `sic_restore()` should occur. In either case, the appropriate SICP function is executed overwriting the location with either 0's, for `sic_initialize()`, or the authenticated and decrypted system state, for `sic_restore()`.

Checkpoint Location and Contents The memory section containing *STATE* is separately declared in the device linker file, enabling easy identification by the system and forcing its location within tamper sensitive memory. This organization solves a number of key implementation challenges for our protocol. It provides a single known location for the `sic_wipe()` operation to target, discussed in detail later in this section. A guaranteed memory location also allows a straightforward check on the existence of a *factory_reset()* operation. It provides the application developer a much simpler declaration interface, enabling the use of GCC's variable attributes, marked with the `__attribute__` keyword, instead of a complex variable registration interface and tracking data structure. The dedicated memory section also simplifies the storage of processor state information, such as the stack or heap, by allocating space within the segment for their storage during the creation of a checkpoint. Finally, it reduces the complexity of the checkpoint process. To create a new checkpoint, the state information in this memory segment must be updated, then `sic_refresh()` must be called to wrap the segment up in a valid state save packet.

The final state save packet contains the necessary information for the intermittent system to restore the previously saved system state. This includes a copy of all CPU registers, the stack, peripheral states, and any developer identified variables that are necessary to restore the program to its previous point of execution. If the checkpointing system determines that no valid checkpoint exists, such as on the first boot after a *factory_reset()*, it will invoke `main()` as would be expected in a standard system startup.

By conducting the tests for checkpoint restoration during the early boot process and limiting the additional storage locations to a specific memory location it is possible to allow the developer to enable or disable SICP at compile time without other changes to the code base.

WIPE The implementation of the *WIPE* operation requires proper detection of power loss by monitoring the device's V_{cc}. This is accomplished with the MSP430FR5994's ADC12_B analog-to-digital converter, measuring V_{cc} against the system's V_{ref} as described in TI's FRAM Utilities [29]. The MSP430FR5994's V_{ref} is supported by a band gap voltage reference circuit, and remains stable even when the system is suffering a power loss. Direct-memory-access (DMA) overwrite of the *STATE* and SRAM is triggered when V_{cc} falls below V_{ref}. The MSP430FR5994 development board's unmodified implementation, including one 10μF capacitor and three 100nF capacitors, has sufficient residual energy to consistently overwrite up to 16kB of memory following the trigger [31].

3.6 Evaluation

In this section, we demonstrate SICP's feasibility and measure the cost in terms of space and processing overhead incurred. The performance effects of the different AEAD primitives on our implementation are measured and their feasibility for use in real-world applications are considered. Initial tests verify the restoration of program state after a power loss in our proof-of-concept implementation.

The comparison between the performance of the different AEAD schemes is specific to this implementation and should not be taken as an evaluation of the different AEAD constructions themselves. We have utilized reference implementations for both the software and hardware-accelerated AEAD designs. We emphasized correctness of the system integration over optimization of the individual primitives. This implementation serves as a baseline for the performance of the protocol.

All measurements used a state size of 2kB, a reasonable region for applications on a resource limited device.

Table 3.3: Overhead incurred by the SIC Protocol

(a) Checkpoint Overhead

Function	Cycle Count
`init` *(no saved state)*	128
`restore`	8679
`checkpoint`	10050

(b) Executable Size Overhead

Component	Size (B)
Checkpoint Support	2532
RNG Support	418
SICP API (EAX/Ketje Sr) [a]	1164/1504
EAX (HW)	2774
EAX (SW)	3262
Ketje Sr	3336

[a] Size differences occur within the SICP API because of the changes between the EAX and Ketje Sr calling conventions.

Checkpoint Overhead In all cases, the overhead incurred by the checkpointing system is constant. The cycle count of the checkpoint operations was measured when SICP is disabled to gather overhead data. This provides the processing cost incurred when copying the CPU state, stack, and device peripheral state into the secure non-volatile memory section and marking the data as valid during a *REFRESH* or *RESTORE* operation without any additional security operations. Table 3.3a shows the cycle cost for saving and restoring a 2kB checkpoint without security.

SICP Overhead Measuring the overhead created by the SICP functions requires observing their execution time without the subsequent checkpointing system calls. This also allows the device to be tested without requiring a restart between each iteration since the SICP functions' computational costs only depend on the validity of the stored states, SS_A and SS_B, and their associated tags. Additionally, the overhead of SICP functions must be measured separately when SS_A and when SS_B are the valid state because the authenticity of SS_A is always checked first in the protocol. As a result, a computational benefit exists for two-pass AEAD schemes when SS_B is the valid state and the decryption of SS_A can be

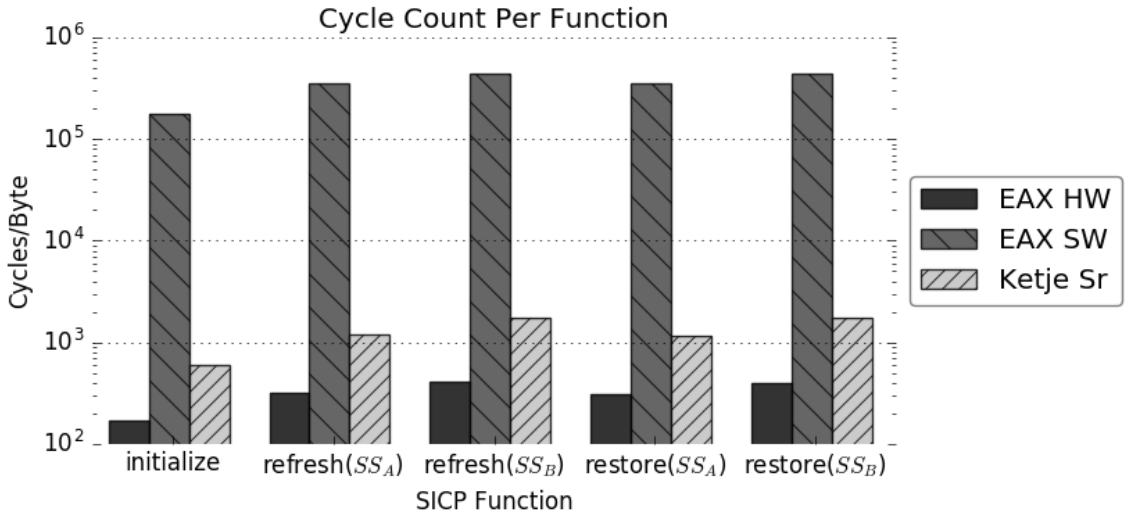

Figure 3.8: Graph of the cycles per checkpointed byte necessary to complete each SIC function. *REFRESH* and *RESTORE* are shown for each case when SS_A or SS_B is the valid state save packet. Since the packet validity is always checked in order, there is a measurable difference in computation required depending on which packet is valid. The unoptimized software implementations clearly take too many cycles to be employed in a real world embedded application. This is in agreement with our argument that an actual deployment of SICP must be supported by a hardware accelerated cryptographic primitive.

skipped after a failed tag check.

The executable size overhead, when compiled with the `-Os` option, for all components are listed in Table 3.3b. It provides an estimate of the expected growth of a program's memory footprint when support for each component is added to the system. The SICP entry represents the executable size overhead of the protocol functions, while the supporting cryptographic kernels are listed separately. The code supporting checkpoints, listed first, is constant between tests and are therefore listed only once.

AEAD Performance The processing overhead of the three different AEAD schemes used in SICP's implementation are depicted in Figure 3.8. It shows the cycle count for computing `sic_initialize()` only once for each scheme since it is a constant operation, always creating

a valid SS_A and returning to normal execution. The other two functions are shown twice, once for each state save packet's validity, to highlight the reduced computational cost when a two-pass AEAD scheme is able to abort a decrypt operation after identifying an invalid tag. This is clearly visible for both implementations of EAX in `sic_refresh(`SS_B`)` and `sic_restore(`SS_B`)` where SS_B is valid, it is able to skip the decryption of invalid state, S_A, after checking the tag, T_A. These values are only 1.255 times larger than when SS_A is the valid state save packet compared to the 1.495 increase seen by Ketje Sr, a one-pass AEAD scheme. This behavior is specific to our two-pass EAX implementation. If the EAX implementation did not contain this short circuit, we would expect to see behavior identical to the Ketje Sr implementation.

3.6.1 Interpretation

From these results, it is possible to identify a number of design considerations for future implementations of a secure intermittent computing protocol. First, two-pass AEAD schemes that can identify an invalid tag before the decryption operation is complete may reduce overhead on resource constrained devices. Second, a hardware accelerated or lightweight AEAD scheme is a must for deployment of this protocol in real-world applications. SICP requires two or three encryption operations for each *REFRESH* or *RESTORE*, depending on which state save packet is valid, and while the reference implementation of Ketje Sr was substantially faster than the reference software implementation of EAX, it still required slightly more than one second for each encryption operation. This is a high price to pay for devices that are truly resource constrained, especially if frequent checkpoints are requested by the developer to ensure forward progress in their program.

All experiments were conducted with a 2kB secure state. The need for hardware supported

AEAD schemes increases as secure state size grows. If a larger secure state is required, possibly due to large user data structures, then the expected processing time for the non-hardware accelerated implementations would be larger and likely untenable for intermittent computing systems.

Minimum powered time for forward progress is an unexpected design consideration exposed by the implementation of the protocol. The SICP functions do not leave a valid state save packet until they are fully executed, a side effect of satisfying the atomicity requirement. As a result, processing the SICP functions requires a minimum powered time for the supported program to ever progress. For example, if the `sic_restore()` function requires one second of computation to complete then the system will only begin to make forward progress after at least one second of continuous power and will have a minimum powered time of one second. Reducing the processing cost of the SICP functions, through a hardware accelerated or lightweight AEAD scheme, directly reduces this minimum powered time requirement. Unfortunately, it is impossible to completely remove this requirement and guarantee forward progress while meeting our security goals.

This proof-of-concept implementation demonstrated the feasibility of the SIC protocol and highlighted the design considerations that future implementations will face. A hardware supported AEAD primitive is necessary for resource constrained systems and the minimum powered time of the intermittent power source should be considered when determining the size of the secure state and choosing which AEAD scheme is best suited to the task.

3.7 Conclusion

This chapter was motivated by the lack of appropriate security features incorporated in existing intermittent systems. The Secure Intermittent Computing Protocol addresses the

security of intermittent systems across periods of power loss and its implementation highlights the heavy computational cost required to secure a stored system state. Our work demonstrated the need for future work in lightweight cryptographic kernels that can support AEAD schemes. Possibly, a platform could be developed that features both FRAM, or an equivalent low latency non-volatile memory, and the necessary hardware acceleration to support the storage of larger system states. Although we met our goal of securing intermittent systems, the performance of our solution is an area were additional progress can be made. It is our hope that this work triggers a discussion on the security of intermittent devices and drives future work in the development of efficient cryptographic primitives and platforms suitable for the growing intermittent computing domain.

Bibliography

[1] J. Andronick. Formally proved anti-tearing properties of embedded c code. In *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA 2006)*, pages 129–136, Nov 2006.

[2] Domenico Balsamo, Alex S. Weddell, Geoff V. Merrett, Bashir M. Al-Hashimi, Davide Brunelli, and Luca Benini. Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems. *Embedded Systems Letters*, 7(1):15–18, 2015.

[3] Mihir Bellare, Phillip Rogaway, and David Wagner. *The EAX Mode of Operation*, pages 389–407. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[4] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, and Ronny Van Keer. Keccak code package. https://github.com/gvanas/KeccakCodePackage, 2017.

[5] Guido Bertoni, Joan Daemen, Michal Peeters, Gilles Van Assche, and Ronny Van Keer. Caesar submission: Ketje v2. *CAESAR competition (round 3)*, 2016.

[6] Joseph Birr-Pixton. Cifra: Cryptographic primitive collection. https://github.com/ctz/cifra, 2017.

[7] Siddhartha Chhabra and Yan Solihin. i-NVMM: A secure non-volatile main memory system with incremental encryption. In *38th International Symposium on Computer Architecture (ISCA 2011), June 4-8, 2011, San Jose, CA, USA*, pages 177–188, 2011.

[8] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele 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*, 2012.

[9] Z. Ghodsi, S. Garg, and R. Karri. Optimal checkpointing for secure intermittently-powered IoT devices. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 376–383, Nov 2017.

[10] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 45–60, 2008.

[11] Matthew 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, 2017.

[12] D. E. Holcomb, W. P. Burleson, and K. Fu. Power-up SRAM state as an identifying

fingerprint and source of true random numbers. *IEEE Transactions on Computers*, 58(9):1198–1210, Sept 2009.

[13] Engelbert Hubbers and Erik Poll. Reasoning about Card Tears and Transactions in Java Card. In Michel Wermelinger and Tiziana Margaria-Steffen, editors, *Fundamental Approaches to Software Engineering*, pages 114–128, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[14] Hrishikesh Jayakumar, Arnab Raha, Woo Suk Lee, and Vijay Raghunathan. QuickRecall: A HW/SW Approach for Computing across Power Cycles in Transiently Powered Computers. *JETC*, 12(1):8:1–8:19, 2015.

[15] S. Kannan, N. Karimi, O. Sinanoglu, and R. Karri. Security Vulnerabilities of Emerging Nonvolatile Main Memories and Countermeasures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(1):2–15, Jan 2015.

[16] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. Intermittent Computing: Challenges and Opportunities. In *2nd Summit on Advances in Programming Languages, SNAPL 2017, May 7-10, 2017, Asilomar, CA, USA*, pages 8:1–8:14, 2017.

[17] Pieter Maene, Johannes Götzfried, Ruan de Clercq, Tilo Müller, Felix C. Freiling, and Ingrid Verbauwhede. Hardware-Based Trusted Computing Architectures for Isolation and Attestation. *IEEE Trans. Computers*, 67(3):361–374, 2018.

[18] Dr. David A. McGrew, Michael Curcio, and Scott Fluhrer. Hash-Based Signatures. Internet-Draft draft-mcgrew-hash-sigs-11, Internet Engineering Task Force, April 2018. Work in Progress.

[19] P. D. Mitcheson, E. M. Yeatman, G. K. Rao, A. S. Holmes, and T. C. Green. Energy

Harvesting From Human and Machine Motion for Wireless Electronic Devices. *Proceedings of the IEEE*, 96(9):1457–1486, Sept 2008. http://dx.doi.org/10.1109/JPROC.2008.927494.

[20] Job Noorman, Jo Van Bulck, Jan Tobias Mühlberg, Frank Piessens, Pieter Maene, Bart Preneel, Ingrid Verbauwhede, Johannes Götzfried, Tilo Müller, and Felix Freiling. Sancus 2.0: A Low-Cost Security Architecture for IoT Devices. *ACM Trans. Priv. Secur.*, 20(3):7:1–7:33, July 2017.

[21] M. Penella-López and Manuel Gasulla-Forner. *Optical Energy Harvesting*, pages 81–123. Springer Netherlands, Dordrecht, 2011. http://dx.doi.org/10.1007/978-94-007-1573-8_5.

[22] Amir Rahmati, Mastooreh Salajegheh, Dan Holcomb, Jacob Sorber, Wayne P. Burleson, and Kevin Fu. TARDIS: Time and Remanence Decay in SRAM to Implement Secure Protocols on Embedded Devices without Clocks. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 221–236, Bellevue, WA, 2012. USENIX.

[23] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: system support for long-running computation on rfid-scale devices. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2011, Newport Beach, CA, USA, March 5-11, 2011*, pages 159–170, 2011.

[24] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107, 2002.

[25] Richard M. Stallman and GCC DeveloperCommunity. *Using The Gnu Compiler Collection: A Gnu Manual For Gcc Version 4.3.3*. CreateSpace, Paramount, CA, 2009.

[26] M. Taha and P. Schaumont. Key Updating for Leakage Resiliency With Application to AES Modes of Operation. *IEEE Transactions on Information Forensics and Security*, 10(3):519–528, March 2015.

[27] Texas Instruments. *AES Accelerator*, 2012. Rev. E.

[28] Texas Instruments. *MSP430 GCC User's Guide*, 2016.

[29] Texas Instruments. *MSP MCU FRAM Utilities*, 2017.

[30] Texas Instruments. *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide*, 2017.

[31] Texas Instruments. *MSP430FR599x, MSP430FR596x Mixed-Signal Microcontrollers*, 2017.

Chapter 4

Conclusions

In this work, we proposed to secure energy harvested IoT devices by exploiting the surplus energy and designing secure protocols. We presented the advantages of precomputing cryptographic algorithms using the surplus energy available from the harvester. We supported this argument by computing energy delay product improvement of two applications, bulk encryption using AES-CTR mode and true random number generation. The improvements from these numbers show how precomputation can improve secuirty of IoT devices by enabling extensive operations to be partitioned into data independent and data dependent operations. We then highlight the security pitfalls in existing intermittent computing solutions and bring out the need for considering security of intermittent systems. We proposed the Secure Intermittent Computing Protocol that provides continuity to the application running on the device and atomicity to protocol primitives. Used in conjunction, precomputation and secure intermittent computing will provide adequate security to energy harvested IoT devices.

Bibliography

[1] Fei Fei, John D. Mai, and Wen Jung Li. An Indoor Air Duct Flow Energy Conversion System: Modeling and Experiments. pages 75–80, 2013.

[2] Vikram Gupta, Arvind Kandhalu, and Ragunathan (Raj) Rajkumar. Energy Harvesting from Electromagnetic Energy Radiating from AC Power Lines. In *Proceedings of the 6th Workshop on Hot Topics in Embedded Networked Sensors*, HotEmNets '10, pages 17:1—-17:6, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0265-4. doi: 10.1145/1978642.1978664. URL http://doi.acm.org/10.1145/1978642.1978664.

[3] H. Jayakumar, A. Raha, and V. Raghunathan. Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 330–335, Jan 2014. doi: 10.1109/VLSID.2014.63.

[4] Shuguang Li and Hod Lipson. Vertical-Stalk Flapping-Leaf Generator for Wind Energy Harvesting, 2009. URL http://dx.doi.org/10.1115/SMASIS2009-1276.

[5] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: System support for long-running computation on rfid-scale devices. *SIGARCH Comput. Archit. News*, 39(1): 159–170, March 2011. ISSN 0163-5964. doi: 10.1145/1961295.1950386. URL http://doi.acm.org/10.1145/1961295.1950386.

[6] Farhan Simjee and Pai H. Chou. Everlast: Long-life, supercapacitor-operated wireless sensor node. In *Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, ISLPED '06, pages 197–202, New York, NY, USA, 2006. ACM. ISBN 1-59593-462-6. doi: 10.1145/1165573.1165619. URL http://doi.acm.org/10.1145/1165573.1165619.

[7] Nagarajan Sridhar and Dave Freeman. A Study of Dye Sensitized Solar Cells under Indoor and Low Level Outdoor Lighting: Comparison to Organic and Inorganic Thin Film Solar Cells and Methods to Address Maximum Power Point Tracking. *Proceedings of 26th European International Conference on Photovoltaic Solar Energy*, pages 232–236, 2011.