# Secure Intermittent Computing: Precomputation and Implementation

Charles E. Suslowicz

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

Alan J. Michaels

Cameron D. Patterson

May 7, 2018

Blacksburg, Virginia

Keywords: intermittent computing, embedded systems, random number generation

# Secure Intermittent Computing: Precomputation and Implementation

Charles E. Suslowicz

(ABSTRACT)

This thesis explores the security of intermittent devices, embedded systems designed to retain their state across periods of power loss, for cases both when the device has an excess of available energy and when power loss is unavoidable. Existing work with intermittent systems has focused on the problems inherent to the intermittent paradigm and ignored the security implications of persistent state across periods of power loss. The security of these devices is closely linked to their unique operational characteristics and are addressed here in two studies. First, the presence of an energy harvester creates an opportunity to use excess energy, available when additional energy is harvested after the local energy reservoir is filled, to precompute security related operations. Precomputation powered by this excess energy can reduce the cost of expensive tasks during periods of energy scarcity, potentially enabling the use of expensive security operations on traditionally unsecured devices. Second, when energy is limited and intermittent operation is required, the secure storage of checkpoints is a necessity to protect against adversary manipulation of the system state. To examine the secure storage of checkpoints a protocol is implemented to ensure the integrity and authenticity of a device's checkpoints, and evaluated for its energy overhead and performance. The cost of properly ensuring the integrity and authenticity of these checkpoints is examined to identify the overhead necessary to execute intermittent operations in a secure manner. Taken together, these studies lay the groundwork for a comprehensive view of the current state of intermittent device security.

# Secure Intermittent Computing: Precomputation and Implementation

Charles E. Suslowicz

(GENERAL AUDIENCE ABSTRACT)

This thesis explores two unique aspects of the intermittent computing paradigm, the precomputation during periods of excess energy and the security of system checkpoints. Intermittent systems are a class of embedded device that lack a classic, consistent, energy source and instead rely on transient energy collected from their surroundings. This removes the need for connection to a power grid or battery management, but introduces challenges in operation since the device can lose power at any time. Additionally, excess energy is available to these systems when they have filled their local energy reservoir, a capacitor or small rechargeable battery, and additional energy can still be collected form the environment. In this case, it is possible to begin precomputing energy intensive operations to enable more operations at a later time on a limited energy budget. Since their power source is inconsistent, intermittent systems checkpoint their current state to allow execution to resume at the beginning of the next power cycle. The security ramifications of saving the current system state into a checkpoint have not been considered in the state of the art. This thesis implements a protocol to properly secure system checkpoints and evaluates its performance to identify the energy overhead required for a secure checkpointing scheme. The results demonstrate the need for the development of more efficient solutions within the domain. Together, the two approaches presented in this thesis provide case studies on the behavior of intermittent devices when provided with either an excess or a dearth of energy. The optimization and improvement of modern intermittent devices will need to address both of these extremes as the field is further improved.

# Dedication

*This work is dedicated to my wife, Arena, whose patience has been boundless throughout its creation.*

# Acknowledgments

I would like to thank my committee: Dr. Schaumont, Dr. Michaels, and Dr. Patterson for their guidance and oversight. I would like to acknowledge and thank all the members of the Secure Embedded Systems group for their help, thoughts, and advice over the last two years.

I would especially like to thank the other members of the Energy Harvesting team: Archanaa S. Krishnan and Daniel Dinu for their dedication to our shared projects.

The members of both the Virginia Tech IT Security Lab and Hume Center were instrumental in my initial research attempts and provided excellent guidance in my early work. I owe them a special thanks for helping bring me back to an academic mindset and knock the rust of my learning faculties.

Finally, I must offer my deepest gratitude to Dr. Schaumont for his patience and understanding throughout my time under his direction.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

AES Advanced Encryption Standard

CBC Cipher Block Chaining

CMAC Cipher-Based Message Authentication Code

CRC Cyclic Redundency Check

CT Cipher Text

CTPL Compute Thru Power Loss

CTR Counter

DCO Digitally Controlled Oscillator

DMA Direct Memory Access

DPA Differential Power Analysis

ECDSA Elliptic-Curve Digital Signature Algorithm

EDP Energy-Delay Product

FRAM Ferro-electric Random Access Memory

GPIO General Purpose Input Output

IV Initialization Vector

JTAG Debug Port Connection (Joint Test Action Group), IEEE Standard 1149.1-1990

LPM Low Power Mode

MAC Message Authentication Code

NIST National Institute of Standards and Technology

NVM Non-volatile memory

OMAC One-Key Message Authentication Code

OTP One-Time Pad

PRNG Pseudo-Random Number Generator

PT Plain Text

RISC Reduced Instruction Set Computer

RNG Random Number Generator

SRAM Static Random Access Memory

TRNG True Random Number Generator

VLO Very-Low Frequency Oscillator

XOR Exclusive Or

# Chapter 1

# Introduction

The expansion of the Internet of Things (IoT) and rapid growth of embedded systems applications has created a requirement for low power devices operating without significant or frequent support. Traditional embedded systems operate on a consistent power source, often batteries that are either rechargeable or replaced during maintenance, but leave much to be desired for some classes of lightweight or remote applications.

Energy harvested devices have developed as a potential solution to this problem, operating off energy collected from their immediate environment without additional resources, maintenance, or incurring additional load on existing energy sources. The unique challenge for the operation of energy harvested devices is the unreliable nature of their harvested energy, often leading to abrupt power loss and interruption of the current system task. If all system tasks were sufficiently atomic that they could be completed during the same power cycle, this would not be an issue for the employment of energy harvested devices. Unfortunately, most tasks are not this small, even on very simple embedded devices, and a new operational paradigm is necessary for their successful application [11].

A second challenge unique to energy harvested systems stems from their direct connection to a constantly fluctuating energy source. This connection creates a unique operating paradigm where the device is either charging or discharging its energy reservoir depending on the state of the harvester and its operation. Previous work has focused on improving the device efficiency and extending the period during which a device can operate when no additional

energy is provided via the energy harvesting circuit [7, 9, 10, 12]. Very little work has been done to consider the case where the energy reservoir is filled and additional energy is still produced by the energy harvesting circuit.

A wide ranging assumption within current literature is that a harvester cannot gather enough energy to both run a device and continue charging an energy reservoir. This is already unrealistic, as research by Simjee and Chou [13] showed it was possible with a solar harvester, and energy harvesting technologies only improved since their publication. Chapter 2, *Optimizing Cryptography in Energy Harvesting Applications*, explores a potential application for this otherwise wasted energy in the precomputation of cryptographic operations.

Intermittent systems take their name from their ability to operate in the opposite condition, when insufficient energy is present to run the device and execution must be paused until additional energy is gathered. Normal embedded systems stop when their energy reserve is exhausted and then restart, from a set initial state, their software when additional energy is collected. An intermittent device is capable of storing the current state of the system in a checkpoint before shutting down and using this checkpoint to restore the system to its previous state when power is restored.

By storing the system state in non-volatile memory (NVM), sensitive data can be exposed to tampering by an adversary interacting with the device. Ensuring the integrity and authenticity of intermittent system checkpoints is a new and largely ignored concern for intermittent systems. This security relationship is explored in Chapter 3 *The Price of Continuity in Intermittent Systems* through the implementation of a simple checkpointing scheme, similar to the one presented by Jayakumar *et al.* [7], which is then extended to validate the integrity and authenticity of the system's checkpoints. The energy overhead required for these operations is then studied and presented for both hardware-supported and software-only versions of the system.

## 1.1 Precomputation

Chapter 2 of this thesis explores the use of excess energy to precompute portions of expensive cryptographic operations and thereby reduce their runtime energy costs. Central to its approach is the creation of *coupons* through the precomputation of portions of a cryptographic operation for later use to reduce the energy required to complete the task. For some operations, this reduced energy cost is significantly lower than the normal operational cost and can result in significant energy savings when a *coupon* is consumed at runtime. To benefit from this arrangement the *coupon* must be created during a period of abundant energy and be used to reduce the runtime energy cost of the cryptographic operation.

Identifying operations well suited for precomputation is a challenge, but the technique has been employed within the cryptographic community previously [5, 6]. The application of precomputation to energy harvested devices is a different challenge as the state of the energy harvester determines when precomputation or *coupon* usage should occur. Some work has been done to employ precomputation to reduce energy consumption [1, 2], but these solutions were not applied to energy harvested devices and their unusual energy storage characteristics.

Chapter 2 presents the unique challenges faced in identifying when precomputation should be employed to support an energy harvested device, a metric for identifying operations well suited for precomputation, and two case studies exploring the use of precomputation to reduce the energy required for a cryptographic operation on an such a device. The results of the case studies show that it is exceptionally worthwhile to precompute *coupons* for use at runtime, reducing energy consumption by more than 10x for some operations, and we conclude that future solutions for energy harvested devices should consider precomputation of complex operations during periods of excess energy.

## 1.2 Secure Intermittent Operation

Chapter 3 discusses the opposite conditions from those suited for precomputation, when insufficient energy is available for continuous operation of a device and intermittent operation is a necessity. The ability to save and restore the system state when an energy reservoir is exhausted serves as the underlying capability for intermittent devices, continuing the same execution across multiple power cycles.

To enable this unique behavior, the majority of intermittent systems create and store checkpoints of their operational state during normal execution. These checkpoints are then used to restore the system state after a period of power loss and enable the continued execution of a long running task as though no interruption had occurred.

The effective creation and restoration of checkpoints has been a very active research area for the past decade. Determining when, where, and how to record a system's operational state creates challenges in maintaining a program's control flow and data consistency that have lead to a variety of proposed solutions. The earliest attempts by Ransford *et al.* [12] created checkpoints when the monitored energy store was determined to cross a hard coded voltage threshold. This process was further improved in work by Balsamo *et al.* with their Hibernus solution [3, 4] which employed ferro-electric RAM (FRAM) to dramatically reduce the hibernation cost of their systems.

These and other solutions all took a simple approach to safely interrupting program execution, leaving it to the developer to reason about the side effects of partially completed operations. Work by Lucia *et al.* presented an alternative programming paradigm, DINO [9], that offered an alternative approach for structuring an embedded systems software. Following their methods, it is clear, within the source code itself, what tasks are eligible locations for checkpoints to occur and guarantees can be made about the state of the system

when it is restored. An improvement and extension of these concepts was demonstrated by Woude and Hicks with their introduction of Ratchet [16], which automatically breaks a program into idempotent sections during compilation and ensured that checkpoints only occur on the boundaries of these sections. This prevents any data corruption from occurring in non-volatile memory through the repeated execution of side-effect causing instructions.

Despite this extensive body of work, which all strictly focused on the successful execution of such intermittent operations, the security considerations necessary for operation in a real world environment have been completely ignored. By writing the information necessary to restore execution into a checkpoint, traditionally private information, present only in volatile memory, is written to NVM and available for inspection by an adversary.

Chapter 3 of this thesis explores the energy cost necessary to secure the integrity and authenticity of intermittent device checkpoints through the implementation of a straightforward checkpointing scheme and extensive measurement of the energy costs associated with ensuring the integrity and authenticity of the resultant checkpoints.

## 1.3 Contributions

This thesis examines the effect of these two opposing conditions on the security of embedded systems. Chapter 2 examines the potential for precomputation of cryptographic operations with the excess energy occasionally available via an energy harvester. Through the use of otherwise wasted energy, it is possible to dramatically reduce the runtime energy cost of certain cryptographic operations, laying the groundwork for future efforts to employ stronger cryptographic primitives on embedded systems reliant on energy harvesters for power. Chapter 3 explores the energy costs incurred to verify the integrity and authenticity of intermittent system checkpoints. Our measurements show that ensuring the integrity and authenticity of

system checkpoints is a necessary but expensive task. The use of a hardware accelerated or lightweight permutation is necessary for even moderately reasonable overhead. Future work may be able to further reduce this computational overhead, possibly through the integration of a precomputation scheme or other optimizations unique to the intermittent computing domain.

## 1.4 Attribution

The manuscripts presented in this thesis are the combined effort of multiple individuals. For both manuscripts I served as the primary author, structuring the outline, drafting the abstract, compiling references, and constructing the overall document.

I developed the primary argument and application for the precomputation techniques discussed in Chapter 2 with my advisor, Dr. Schaumont, and validated their validity through the development and execution of the True Random Number case study presented in the manuscript. The random number generators used within the case study were constructed specifically for this study and tested to pass the NIST Statistical Test Suite before their inclusion in the case study. The AES-CTR mode case study, created by my co-author, is the only element of this paper I did not create. This manuscript was accepted and presented at *Attacks and Solutions in Hardware Security (ASHES) 2017* and can be referenced as the following:

- Charles Suslowicz, Archanaa S. Krishnan, and Patrick Schaumont. Optimizing cryptography in energy harvesting applications. In *Proceedings of the 2017 Workshop on Attacks and Solutions in Hardware Security - ASHES 2017*. ACM Press, 2017. doi: 10.1145/3139324.3139329. URL https://doi.org/10.1145/3139324.3139329

The intermittent system used within the second manuscript, Chapter 3, is built on a secure intermittent operation library I developed. This library supports both the checkpointing and security operations discussed throughout the manuscript. The checkpointing functionality is a heavily modified version of the Texas Instruments *Compute Thru Power Loss* library [15], including rewrites of all low level code to support a different compiler, addition of developer initiated checkpoints, integration of a secure program section, and the capability to call a security function from within the checkpointing process. Finally, I developed the debugging and test suite code used within the evaluations presented in Chapter 3. Two major elements of Chapter 3 I did not create are the Secure Intermittent Computing Protocol theory and the oscilloscope automation and measurement software created by my co-authors. This manuscript has been submitted to *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED) 2018* and is pending review.

# Chapter 2

# Optimizing Cryptography in Energy Harvesting Applications

Charles Suslowicz[1], Archanaa S. Krishnan[1], Patrick Schaumont[1]

[1]Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 34061 USA

## 2.1 Abstract

The Internet of Things will need to support ubiquitous and continuous connectivity to resource constrained and energy constrained devices. To this end, we consider the optimization of cryptographic protocols under energy harvesting conditions. Traditionally, computing using energy harvesting power sources is handled as a case of intermittent-computing: working towards the completion of a goal under uncertain energy supply. In our work we consider the often ignored case when there is excess harvested energy available, but there are no useful operations to complete. In cryptographic protocols, this can occur while the protocol waits for the next message. To avoid waste, we partition cryptographic algorithms into an offline portion and an online portion, where only the online portion has a real-time dependency to the availability of data. The offline portion is precomputed with the result stored as a *coupon* for the remaining online operation. We show that this structure brings

multiple benefits including decreased response latency, a smaller energy store requirement, and reduced energy waste in a harvester supported system. We present a case study of two canonical cryptographic applications: true random number generation and bulk-encryption. We analyze the precomputed implementations on an MSP430 with ferroelectric RAM and an ARM Cortex M4 with nonvolatile flash memory. Our solutions avoid energy waste during the offline phase, and they offer gains in energy efficiency during the online phase of up to 28 times for bulk-encryption and over 100 times for random number generation.

## 2.2 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 a devices operation 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 paper, 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.

### 2.2.1 Contributions

In this paper 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 MSP-430 and ARM-Cortex M4, and a true random number generator (on MSP-430).

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

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 |

consumed, cycle count, operational delay, and the Energy-Delay Product (EDP) of their execution. This framework enables effective judgement on the suitability of pre-computation for a particular implementation and provides insight into the potential performance of a device utilizing precomputed *coupons* for cryptographic operations.

The remaining portions of this chapter are structured in the following manner. Section 2.3 discusses previous work in energy harvested systems, precomputation of cryptographic algorithms, scaling within the IoT, and our threat model. Section 2.4 details our core concepts: the computation of *coupons* and our framework of metrics for comparison between precomputed and non-precomputed algorithms. Section 2.5 contains the two case studies and their related analysis. Section 2.6 and Section 2.7 present future work and our conclusions respectively.

## 2.3 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.

### 2.3.1 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 and 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) [28]. The stability of this data is illustrated on 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 paper, 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.

### 2.3.2 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 straight forward adaptation and underlies the concept of rainbow tables and a 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 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

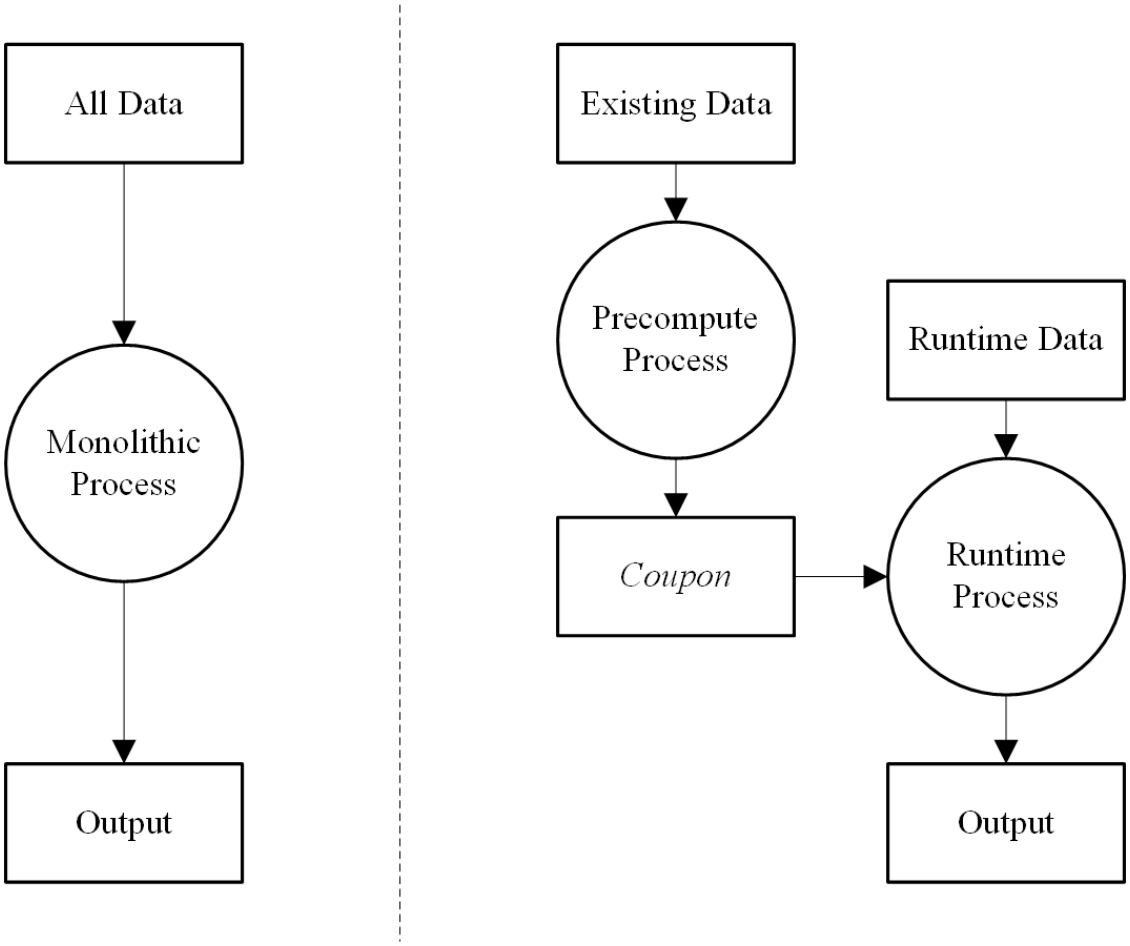


Figure 2.1: The process for a single operation, shown on left, and the precomputed operation, shown on right. When combined, the *coupon* and runtime data, available only immediately before execution, allow the generation of an output identical to the single, monolithic, process.

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.

### 2.3.3 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, 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.

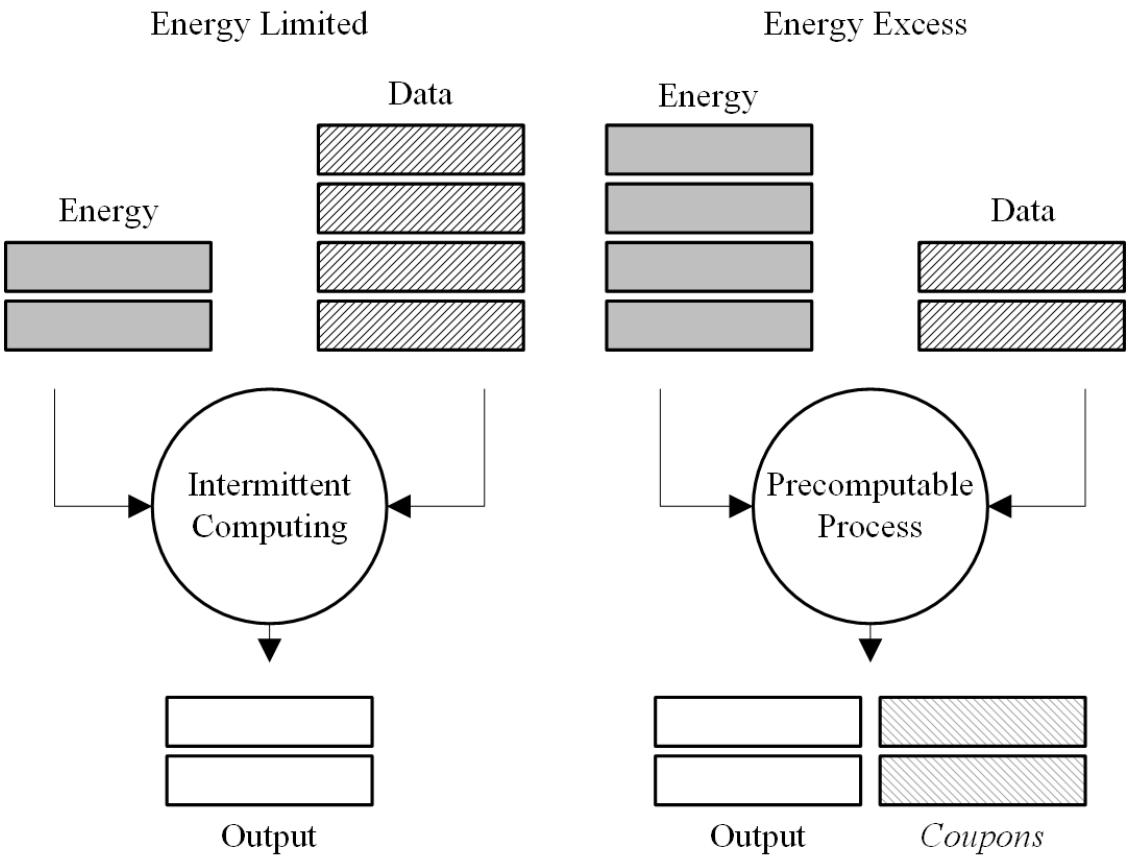


Figure 2.2: Intermittent computing ensures that as much output as possible is created during a scarcity of energy. Our work ensures that excess energy is utilized to improve the efficiency of future operations with *coupons*.

### 2.3.4 Threat Model

A multitude of threats exist 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 systems inputs and outputs 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.

## 2.4 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 en-

ergy 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 2.5.

### 2.4.1 Intermittent Computing and Cryptography

In this paper, 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 proper operation when during periods of low energy when *coupons* are most likely to be consumed and performance benefits realized.

### 2.4.2 *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).

$$E_{original} \leq E_{precomputation} + E_{runtime}$$
$$E_{runtime} < E_{original} \tag{2.1}$$

The value of this transformation can be seen when considering the energy-delay product (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 the Section 4 make it clear that cryptographic operations utilizing *coupons* provide better energy efficiency and performance than those without.

### 2.4.3 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 Energy-Delay Product (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 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

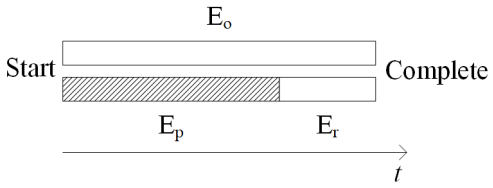


Figure 2.3: Illustration of the energy required for a monolithic computation versus separation into a precomputation and runtime operation.

$r$. 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 being 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.

### 2.4.4 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 upperbound on the number

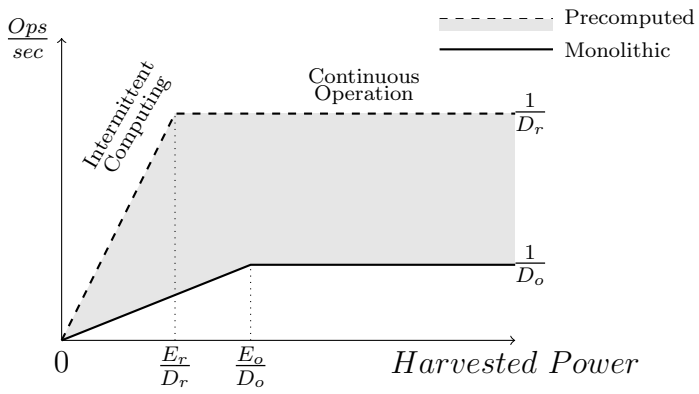


Figure 2.4: Operations per second as a function of Energy influx into the system. When *coupons* are available the device is able to execute more operations within a given time period until limited by the latency of the minimum runtime computation ($D_r$).

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 by 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*.

### 2.4.5 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 including 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 to uncouple 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/ouput 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.

## 2.5 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 $\mu$A/MHz | 80 $\mu$A/MHz |
| **HW accelerators** | AES/CRC/MPY | AES/CRC |
| **Operating mode** | AM, various LPM[2] | AM, various LPM |
| **DMA** | 3-channel | 8-channel |

### 2.5.1 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 makes the selected devices ideal to be used as an

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

energy harvested node. 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.

### 2.5.2 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 counter mode (AES-CTR), a counter value is encrypted first. The encrypted counter value - also known as one-time pad (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 energy required at runtime and latency of the algorithm.

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

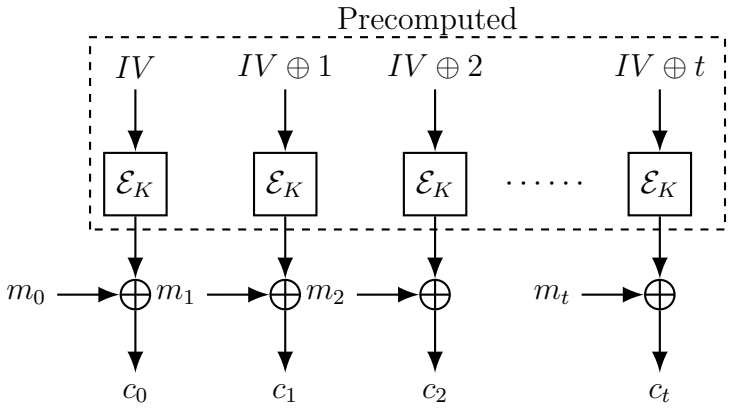


Figure 2.5: Block diagram of counter mode operation [15] with precomputable portion highlighted.

```
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

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

Table 2.3: Cost of Monolithic AES-CTR encryption

| **Device** | **Test case** | $C_o$ $Cycles$ | $E_o$ $\mu J$ | $D_o$ $\mu s$ | $EDP_o$ $10^{-12}J$ |
|---|---|---|---|---|---|
| **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 |

every block of message.

When the whole encryption is done in one online stage, we measured a delay of 6055 $\mu$s 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 run-time.

### 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 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 non-volatile memory and XORing the plain text message with those *coupons*. The energy required for fetching *coupons* and XOR masking in

```
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** | $C_r$ $Cycles$ | $E_r$ $\mu J$ | $D_r$ $\mu s$ | $EDP_r$ $10^{-12}J$ |
|---|---|---|---|---|---|
| **MSP432** | XOR masking | 3455 | 13.8 | 1105 | 15249.0 |
| **MSP430** | XOR masking | 6904 | 8.7 | 6312 | 54914.4 |

MSP432P401R is 13.8 $\mu$J.

### 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*.

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 |

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 |

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 2.7: TRNG Measurements and Precomputation

| RNG Structure | Monolithic Computation $C_o$ *cycles* | $E_o$ $\mu J$ | $D_o$ $\mu s$ | $EDP_o$ $10^{-12} Js$ | 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 |

### 2.5.3 Hardware Random Number Generator

This case study analyzes a true random number generator as a possible best case situation for the precomputation of *coupons*. A 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 within 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 $\mu J$ of energy, and causing a delay of 0.709 $\mu s$. 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 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 pseudorandom number generation (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.

## 2.6 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.

## 2.7 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 in to *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 identified 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.

# 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, Graham Harvey, Brandon Lucia, and Alanson P. Sample. An energy-interference-free hardware-software debugger for intermittent energy-harvesting systems. *SIGOPS Oper. Syst. Rev.*, 50(2):577–589, March 2016.

[10] H. Diewald, G. Zipperer, P. Weber, and A. Brauchle. Electronic device and methods for tracking energy consumption, 2013. US Patent Application US20160077138.

[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

# The Price of Continuity in Intermittent Systems

Charles Suslowicz[1], Archanaa S. Krishnan[1], Daniel Dinu[1], Patrick Schaumont[1]

[1]Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 34061 USA

## 3.1 Abstract

Intermittent systems operate embedded devices without a source of constant reliable power. They overcome this limitation by retaining and restoring system state as checkpoints across periods of powerloss. Previous works have addressed a multitude of problems created by the intermittent paradigm. In this chapter, we address the security concerns created through the introduction of checkpoints to an embedded device. When the non-volatile memory that holds checkpoints can be tampered, the checkpoints can be replayed or duplicated. We define *application continuity* as a defense against this attack. Continuity is the assurance that the application continues where it left off upon powerloss. Our solution to support continuity is to add integrity and authenticity to the checkpoints. We develop two solutions for our secure checkpointing design. The first solution uses a hardware accelerated implementation of AES, while the second one is based on a software implementation of a lightweight cryptographic

algorithm, Chaskey. We analyze the feasibility and overhead of these designs in terms of energy consumption, execution time, and code size across several application configurations. We compare this overhead to a non-secure checkpointing system similar to QuickRecall. We conclude that application continuity does not come cheap, and that it increases the overhead of checkpoint restoration from 3.79 $\mu J$ to 42.96 $\mu J$ with the hardware accelerated solution and 57.02 $\mu J$ with the software based solution. To date, no one has considered the cost to provide security guarantees to intermittent operation. Our work provides future developers with an empirical evaluation of this cost, and with a problem statement for future research in this area.

## 3.2 Introduction

The introduction of intermittent computing to embedded devices has the potential to dramatically change the field. Intermittent devices are able to continue their operations across periods of powerloss through the retention of their system state in non-volatile memory (NVM). This is possible through the advent of low energy NVM technologies such as ferroelectric RAM (FRAM) and phase-change memory [18, 21].

Similarly, improvements in energy harvesting have created the potential for energy harvested embedded systems that are not limited to a strict battery life [16]. Instead, new systems can be equipped with an energy harvester, a small energy store, and intermittent capability to allow continuous operation without the need for a human to recharge or maintain the deployed system. This creates opportunities to deploy embedded systems to locations that are difficult to support or manage, especially for sensor applications and control applications in remote locations.

One aspect of these systems has been ignored by the current body of work, the security

implications of intermittent operation. The act of storing the complete system state in NVM exposes a number of critical pieces of system information to potential tampering by an adversary with access to the device.

Existing intermittent computing solutions operate on the assumption that if a checkpoint exists, it is valid and will properly restore the system [9, 11, 22]. This is a naive and potentially dangerous assumption. Once in non-volatile memory, the checkpoint is at risk of modification, copy, replacement, or corruption by both malignant and erroneous actions. If corrupted, the intermittent system risks entering an invalid state which can only be corrected by a factory or developer reset, since a normal reset operation will restore the altered checkpoint and leave the system unusable or vulnerable.

A small number of NVM protection techniques have been proposed which incorporate encryption into every memory access [3, 10]. Unfortunately, these techniques treat the encryption overhead as a constant and ignore the threat of powerloss to the system's integrity. Ghosdi et al. suggest securing intermittent system checkpoints through the encryption of each checkpoint, but their solution provides only confidentiality and fails to account for checkpoint integrity, authenticity or freshness [8] .

If a checkpoint is tampered, the system is exposed to a host of problems. Secret information on the device, such as private keys, could be exposed. Sensitive operations could be replayed to enable side-channel attacks on normally secure behaviors. Traditional security features, control-flow integrity monitoring or memory protection schemes, can be bypassed through the modification of their data structures during the power-off period [2, 14]. Some schemes, such as SANCUS [15], which include custom hardware support for their security guarantees are also susceptible if the checkpointing process exposes the values stored in the protection mechanism's custom registers or shadow stack within a checkpoint [4]. The storage of this information would be necessary to restore the correct system state, but it exposes other-

wise protected information to tampering by an adversary and creates a discontinuity in the security guarantees provided by these protection mechanisms.

For intermittent devices to benefit from existing security solutions and have application continuity, they must verify that their stored state information is correct and unmodified. This can be accomplished through the cryptographic verification of the checkpoint's integrity, authenticity, and freshness. By verifying the *integrity* of the checkpoint, it is possible for the intermittent system to ensure that no tampering has occurred since the time the checkpoint was created.

Similarly, we must verify the *authenticity* of the checkpoint to ensure that it was in fact created by the intermittent system itself. Without authenticity, it is possible for an adversary to create a valid checkpoint of a vulnerable or weakened state on a test device and later load it onto the target device to facilitate an attack.

Finally, the *freshness* of the checkpoint must be validated to prevent previously created checkpoints from being reloaded onto the device. Without this feature, it would be possible for an adversary to 'roll-back' the state of a device to an arbitrary previous checkpoint. By their nature, intermittent systems will restore their most recent checkpoint during their operation. But, allowing the restoration of *any* prior checkpoint, which exposes the device to replay attacks, would undermine application level security protections.

Naturally, security protection does not come for free, as cryptographic operations consume energy and processor cycles. Our goal is to evaluate, for a realistic scenario in intermittent computing, the overhead of secure application continuity.

### 3.2.1 Contributions

We present the following contributions through our work:

- We highlight the need for continuity in intermittent systems and provide a road-map to support the continuity of existing embedded security applications.

- We introduce a protocol for ensuring the integrity, authenticity, and freshness of an intermittent system's checkpoints across periods of powerloss.

- We empirically evaluate the cost to provide the necessary checkpoint security with both a software implementation utilizing lightweight cryptography and a hardware solution using a hardware accelerated standard cryptographic primitive.

The remaining portions of this chapter are structured as follows: Section 3.3 describes our approach to providing application continuity to a system's checkpointing process and details our protocol to secure the checkpointing process. It outlines our implementation of a basic intermittent system with integrity, authenticity, and freshness. Section 3.4 details the steps involved in integrating security to intermittent system on our test platform. Section 3.5 discusses our experimentation strategy, evaluation, and results. Finally, we close with our conclusions.

## 3.3 Approach

Our approach is to target the checkpoint system itself, protect it, and provide existing embedded security solutions the continuity to protect intermittent systems. This will enable the use of previously developed embedded security systems by ensuring their continuity across the periods of powerloss and checkpointing behavior unique to intermittent systems.

**Algorithm 1** `refresh` and `restore`

**Require:** $KEY$, $STATE$, $S_i$, $CNT_i$, $T_i$, where $i \in \{A, B\}$
$operation \in \{\texttt{REFRESH}, \texttt{RESTORE}\}$

1: **if** $T_B = MAC(S_A|CNT_A|T_A, KEY)$ **then**
2: **if** $operation = \texttt{RESTORE}$ **then**
3: $STATE \leftarrow S_A$
4: **end if**
5: $CNT_B \leftarrow CNT_A + 1$
6: $S_B \leftarrow STATE$
7: $T_A \leftarrow MAC(S_B|CNT_B|T_B, KEY)$
8: **else**
9: **if** $T_A = MAC(S_B|CNT_B|T_B, KEY)$ **then**
10: **if** $operation = \texttt{RESTORE}$ **then**
11: $STATE \leftarrow S_B$
12: **end if**
13: $CNT_A \leftarrow CNT_B + 1$
14: $S_A \leftarrow STATE$
15: $T_B \leftarrow MAC(S_A|CNT_A|T_A, KEY)$
16: **end if**
17: **else**
18: $abort()$
19: **end if**

### 3.3.1 A Secure Protocol

Algorithm 1 depicts the protocol we developed to achieve application continuity in our intermittent system. It details the two necessary functions: `refresh` for the creation of checkpoints and `restore` for the restoration of checkpoints. Both functions are essentially the same except for the extra step of copying the stored system state, $S_i$, into the device's current state, $STATE$, in steps 3 and 11 during checkpoint restoration. We store the system checkpoint in one of two buffers, $A$ or $B$. The buffers are updated in an alternating manner to ensure that at least one secure checkpoint remains valid in the event of powerloss during the execution of `refresh` or `restore`. This ensures that our protocol is robust against powerloss.

When stored, each secure checkpoint is a tuple of three elements: the checkpointed state ($S_i$), a 128-bit nonce ($CNT_i$), and a 128-bit authentication tag ($T_i$).

The checkpoint state, $S_i$, is a copy of $STATE$, the current state of the device. It includes three types of information. First, the necessary application specific state required to resume program execution after powerloss. Second, the microcontroller system state including the program counter, stack pointer, status register and other general purpose registers. And finally, the necessary microcontroller peripheral settings for any peripherals in use.

A nonce is required to provide freshness to each checkpoint. It is introduced to our checkpoints in the form of a counter, $CNT_i$, whenever the checkpoint is generated or restored. The authentication tag, $T_i$, is computed over the current checkpoint state and current counter value using a device unique key, $KEY$, which is at least 128-bit long. A Message Authentication Code (MAC) is used to securely generate the authentication tag. The use of a device unique key, which is kept secret, ensures that every checkpoint is tied to the device and that the checkpoint cannot be replayed on another device.

When a checkpoint is generated the entire control flow of the microcontroller, including the

program counter, stack, status register, and other system critical information, is stored as data in NVM. To preserve the control flow integrity of the program across powerlosses we introduce the concept of *tag-chaining* to authentication tags. We include the authentication tag of the previous secure checkpoint in the computation of the current secure checkpoint's authentication tag, creating a unique chain of tags which reflects the current state and all the past states of the device. The current tag can only be regenerated if the device's current state is reached after execution of exactly the same pattern of previous states. A valid secure checkpoint will now contain the current system state, the current counter value and the corresponding tag which authenticates this system state, and all the past system states, and counter value.

When the system is powered on, the most recent checkpoint, as determined by the tag chain, is used to restore the system. This execution of the `restore` function occurs before any other operations, an approach similar to the solution proposed for Quickrecall [9].

A part of the checkpoint, if not the whole, must be made inaccessible to the adversary to prevent replay attacks. To provide this protection, we have divided the NVM available on the device into tamper-free NVM and tamper-sensitive NVM. Tamper-free NVM is a small section of memory that cannot be tampered or accessed by the adversary even when the device is powered off. It is used to store the smallest elements of our secure checkpoints, the nonces and the device unique key. Since the nonce is unavailable to the adversary, they cannot obtain a copy the entire checkpoint for the purpose of a replay attack.

Tamper-free memory is not currently a standard component of most microcontroller platforms, but recent work in attestation and isolation for microcontrollers demonstrates the feasibility of tamper-free NVM in future devices and Texas Instruments provides limited memory protection capabilities in existing platforms [7, 19]. Tamper-sensitive NVM, the vast majority of our device's NVM, does not posses tamper resistance, and is used to store

the rest of the secure checkpoint including the checkpoint states and the authentication tags.

### 3.3.2 Implementation

To validate our protocol we extended and modified an existing intermittent solution, the *Compute-Thru-Power-Loss* (CTPL) library from Texas Instruments [18]. First, we modified the library to be compatible with the `msp430-elf-gcc` compiler to work with our other existing MSP430 code base and tools. Since the original CTPL code only supported transitions to low power modes and the creation of system checkpoints before shutdown, next, we extended the CTPL feature set to include on-demand checkpointing, a user identified secure data section, alternating checkpoint storage, and integration of the security primitives required by our protocol within the checkpoint creation and restoration process. Finally, we selected cryptographic algorithms based on the available hardware modules and previously demonstrated performance of existing lightweight MACs.

To integrate the necessary security features into CTPL, it was necessary to rewrite the low-level assembly functions of the library that relied on functionality unique to TI's `cl430` compiler. This included changes to dependency resolution, macros, and section declarations while maintaining the overall functionality of each low level function. The addition of security functions to the checkpoint process required significant modification to the control flow of the checkpointing process. Support was added to identify if the requested checkpoint required the use of a security function and to execute the identified function after a checkpoint was assembled. Finally, the original `ctpl_state` flag used by the library was re-purposed to prevent endless checkpoint loops during the wakeup process rather than identify if a checkpoint existed for the system.

### Data Identification and Storage

The first requirement was met through the definition of a memory section, `.secure`, within the device's linker description file. The `.secure` section was defined in a portion of the device's NVM and provided developers with the ability to clearly declare variables that should be included within the device's checkpoints. In fact, the section serves as a live snapshot of the device's current state and includes the allocated non-volatile memory to store the device's stack, register information, and peripheral information during the checkpoint creation process. The use of a specified memory section simplified the creation of checkpoints by collating all of the necessary system data in one contiguous memory region. The collection of the processor state and peripheral information was completed through the normal CTPL process. To create a checkpoint, the device's peripherals were polled and their appropriate register states were stored in the `.secure` memory section. The register information was then pushed onto the system stack and a copy of the stack was stored in the `.secure` section. Once all of the relevant data had been collected, the secure checkpoint creation function `refresh` was invoked.

### Secure Checkpointing Functions

The `refresh` function implements the protocol we developed to determine which checkpoint storage slot is available, then appropriately store the new checkpoint. This is done through the verification of each stored checkpoint's MAC as described by Algorithm 1. The write of the newly computed MAC serves as the transition to the new checkpoint as the valid stored state. Any powerloss prior to the completion of this write will have the system restore the previous checkpoint; once the write is complete, the new checkpoint will pass validity checks and the old checkpoint will fail.

The restoration of checkpoints is implemented in the `restore` function. This is the same process as the one used by `refresh` except for the addition of the copy of the state information stored in a checkpoint into the `.secure` memory section. Once the copy is complete, `restore` also updates the protocol counters and recomputes the MAC of the stored checkpoint. This re-computation is critical for any application level security that needs to be aware of the restoration of checkpoints.

To securely start a new system, we implement an `initialize` function that checks NVM for the device reset memory pattern. This is the memory pattern written into NVM by the device's factory reset function, `0xffff` in the case of our test device. If the pattern is found, it is overwritten to prevent multiple initializations, and an initial checkpoint of the starting system is created. This bootstraps our chain of checkpoints, and allows the `refresh` and `restore` functions to be used throughout the rest of the device's operation.

### Cryptographic Primitives

For our evaluation we chose to employ two different cryptographic primitives, a software based and a hardware based primitive, for MAC operations. This evaluation shows the viability of both approaches, but primarily highlights the computational cost to properly validate a system's checkpoints. Throughout the evaluation process they are referenced as `HW-SIC`, `SW-SIC`, and `NON-SIC` based on the employed cryptographic operations.

`HW-SIC` used a hardware accelerated AES module to compute checkpoint MACs using CMAC. CMAC is a block cipher mode of operation recommended by NIST for use in determining the integrity of data against malicious modification [6]. Our implementation was developed using the Cifra cryptographic library [1] and required very few modifications to utilize the AES co-processor for its block cipher operations instead of the Cifra AES implementation. This

structure provided an excellent baseline for a hardware accelerated standard cryptographic primitive supporting 64-bit collision resistance for MAC operations and was representative of a solution for systems that either contain cryptographic co-processors or need to employ a NIST standard cryptographic primitive for compliance.

`SW-SIC` implemented a software based MAC using Chaskey [13] to measure the overhead that would be experienced by devices that lack an AES hardware accelerator. Chaskey is an efficient lightweight MAC algorithm for use with 128-bit keys. It specifically targets platforms that are not robust enough to employ a standard hash-based authentication code (HMAC) but still require effective security. We chose Chaskey as a representative MAC for this category for two reasons: its MSP430 performance in the FELICS test suite, where it outperformed many other similar lightweight ciphers [5], and its current consideration for standardization by ISO [12]. Combined, we felt these qualities made it a reasonable representative of lightweight cryptographic primitives that may be considered for use in intermittent systems.

Finally, we implemented our protocol without MAC support in order to provide a baseline for a non-secure checkpointing system. `NON-SIC` used the same functions to create and restore checkpoints, but omits the verification steps and instead blindly copies or updates the system state whenever it is requested. To be clear, this implementation *does not* support any of our expected security guarantees and is strictly for use as an example of an unsecured checkpointing system.

## 3.4 Measurement Platform and Test Structure

With these elements identified and created, we were able to integrate them into a fully operational secure intermittent system for use in our evaluation.

### 3.4.1 Platform

To exercise our protocol and determine the overhead incurred by our approach we looked for a reasonable low-cost development device which was representative of systems that might be employed for sensitive operations while supported by an energy harvester. For our platform we chose a MSP430FR5994 Launchpad Development board. This provided the following benefits: we were able to extensively modify TI's *Compute-Thru-Power-Loss* (CTPL) library to support our security protocol while retaining its original board compatibility, the MSP430FR5994 supports 256 kB of ferro-electric RAM (FRAM) providing a high speed NVM to store checkpoints, and is an ultra-low power platform reasonable for an energy harvested application.

The FRAM present on the MSP430FR5994 is an excellent example of the new non-volatile technologies enabling intermittent computing systems. It is efficient, consuming only 225 $\mu A$ of current for each FRAM read or write at 8 $MHz$ operation [20]. Additionally, the manufacturer guarantees write atomicity regardless of the current power condition for the chip. This allows our system to assume that if a write to FRAM is executed, the data is written correctly and does not require additional error checking to detect partial writes at the byte level [17]. The FRAM is capable of normal operation up to 8 $MHz$ and is supported by processor wait-states, which utilizes a 2-way associative 256-bit cache for frequencies up to 16 $MHz$. We were able to see the effect of the processor wait-states and cache on the device's behavior and energy consumption during our evaluation of the device at 16 $MHz$.

### 3.4.2 Testbed

Measurements were taken with a Tektronic DPO3034 oscilloscope across a shunt resistor of 1 $k\Omega$ as depicted in Figure 3.1. We computed the energy consumed by the target device

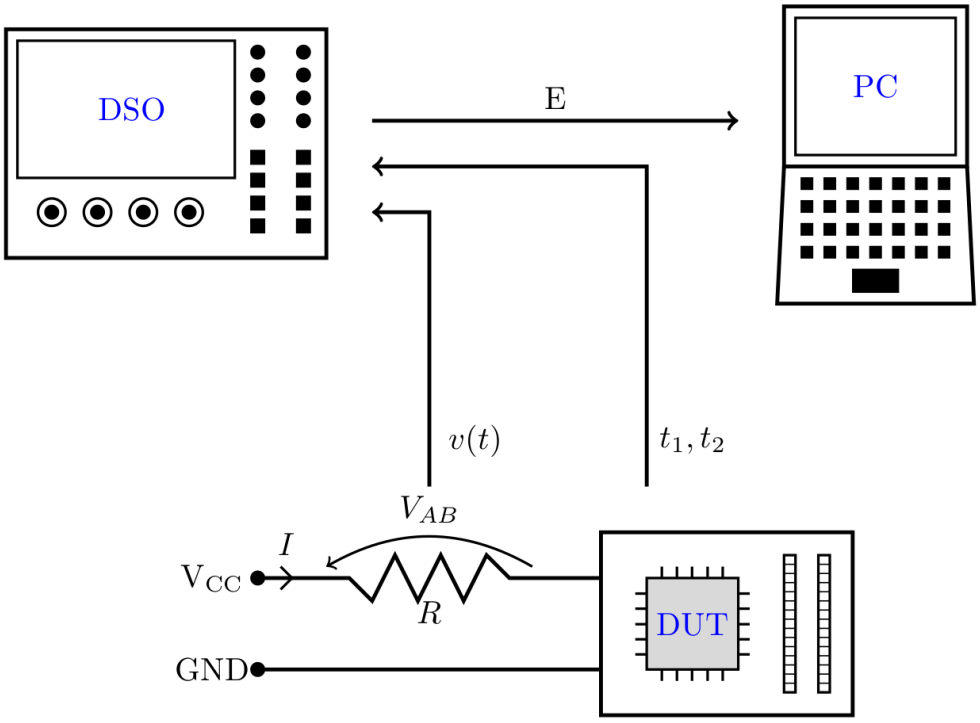


Figure 3.1: The measurement circuit constructed to observe the energy required for different device operations. The device under test (DUT) was powered by an external power supply to execute a continuous loop of operations including oscilloscope triggers via GPIO.

during a function's execution as shown in Equation 3.1.

$$\begin{aligned} E &= V \cdot I \cdot \Delta t = (V_{CC} - V_{AB}) \cdot I \cdot \Delta t \\ &= \frac{1}{R} \cdot \int_{t_1}^{t_2} (V_{CC} - v(t)) \cdot v(t) \cdot dt \end{aligned} \tag{3.1}$$

The three critical functions for secure intermittent operations were measured for both energy consumption and execution time. Each execution was identified within the testbench via a GPIO trigger on the test board. To account for the additional energy cost of the GPIO activation, additional samples were taken with a second active GPIO to compute the energy overhead of a GPIO activation.

Cycle counts were generated for each test case through the use of the on-board timer and an interrupt to catch rollover events. Since this introduced slight variations in each function's execution time, 100 executions were measured to produce an average cycle count. Portions of the checkpointing process normally disable interrupts for real world applications, these were modified to leave interrupts enabled allowing accurate measurement of the cycle count.

Because `refresh` and `restore` execute an additional MAC computation if the first test is invalid, we forcibly tested the functions an equal number of times for each condition. These values were averaged to produce the data in Figures 3.2 and 3.3. To verify the stability of our test bench, we computed the standard error for our energy measurements. Since these values were less than $10^{-8}$ $J$, we omitted them for clarity.

## 3.5 Evaluation and Results

With an experimental setup established, we tested our implementation and gathered measurements across a range of feasible system configurations. We recorded the effect that different system configurations had on the overhead incurred by the software and hardware integrity verifications in order to identify patterns that may be useful when choosing operating conditions for future intermittent devices used in sensitive operations.

### 3.5.1 Experimental Results

From our experiments we were able to successfully observe the energy consumption of the intermittent system across all scenarios. The energy required for securing the checkpointing process highlighted an interesting characteristic of our chosen platform, a reduced energy consumption at the middle clock frequencies, that we explore in more detail in our analysis. The results from changes to the size of the system state, depicted in Figure 3.3, were as expected, increasing the energy required as the state size increased. Finally, the code size also followed our expectations with the software supported operations required more space than the hardware supported primitive.

### 3.5.2 Analysis

From our experiments we observe a few odd behaviors and recognize the very high cost of securing a system's intermittent operation. The overhead for protecting the integrity of an intermittent systems checkpoints is very high. The energy required to secure a 512 $B$ state is 42.96 $\mu J$ (`HW-SIC`) or 57.02 $\mu J$ (`SW-SIC`), an increase by a factor of 11.33 or 15.04 over the non-secure solution at 8 $MHz$. Further reducing this overhead through more novel

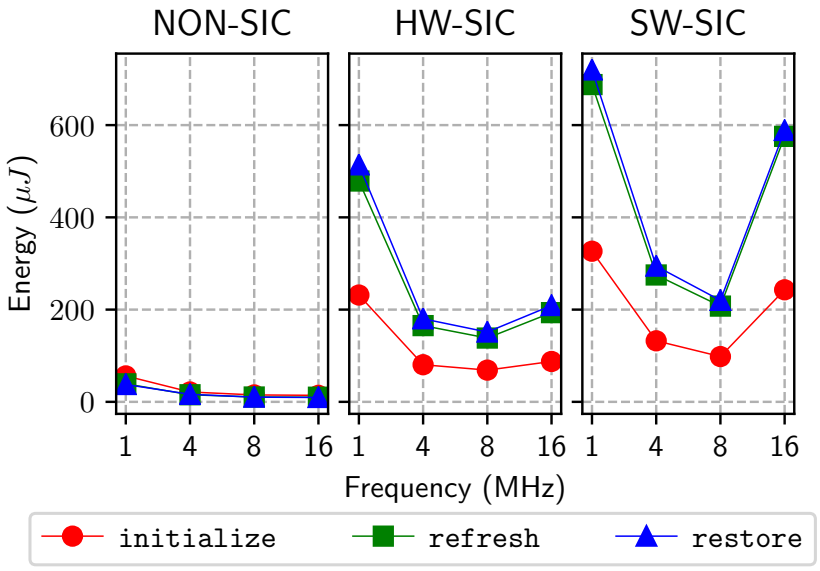


Figure 3.2: Energy consumption as a function of the operational frequency shows the effect of our device's larger static power consumption, leading to a non-linear relationship between 1, 4, and 8 $MHz$ operation. The dramatic spike in energy costs for 16 $MHz$ is tied to the increased minimum supply voltage required for the testbed to operate and the introduction of FRAM wait-states at frequencies above 8 $MHz$.

applications of established cryptographic primitives or re-engineering of the overall system to better support the high computational requirements for the verification operations is a worthwhile future endeavor.

An interesting behavior we observed in Figure 3.2 was the dramatic reduction in energy required for the secure intermittent computing solutions at 4 and 8 $MHz$. Normally, we would have expected a flat energy consumption relationship as the time required to execute the operations decreased while the frequency, and energy consumption rate of the processor, increased. Instead, we determined that the device under test had a significantly higher static power consumption than dynamic power consumption allowing the reduced operation time for the higher frequency operations to save more power than the increased frequency consumed. For example, we found $\alpha$ to be 152.44 and $\beta$ to be 567.11 for the MSP430FR5994 in our `SW-SIC` experiments at 8 $MHz$ according to Equation 3.2. Our calculations for $\alpha$ and

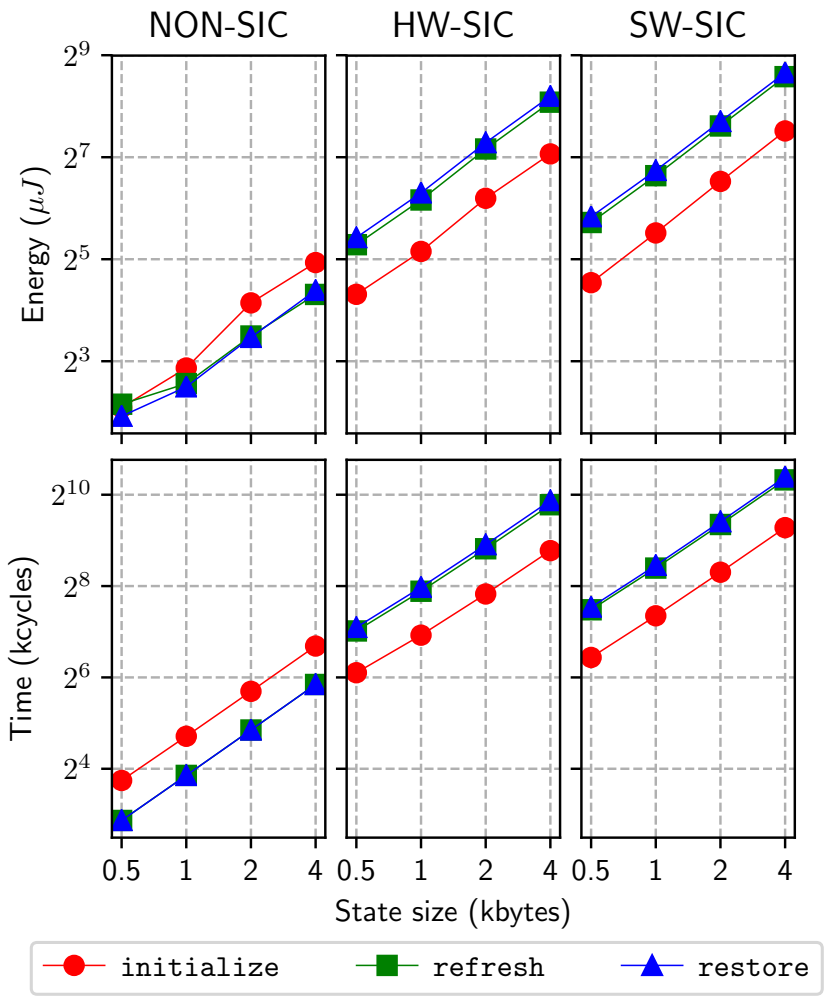


Figure 3.3: The effect of system state size on the execution time and energy consumption of the three security functions operating at 8 $MHz$ is reasonable. `SW-SIC` requires slightly more resources than `HW-SIC` and `initialize` is the cheapest and simplest of the three operations.

$\beta$ in our `HW-SIC` tests were less consistent, an effect we attribute to the operation of the AES co-processor during only portions of the function's execution.

$$E = T \cdot (P_{dyn} + P_{static}) \qquad \begin{aligned} P_{dyn} &= \alpha \cdot f \\ P_{static} &= \beta \end{aligned} \tag{3.2}$$

This relationship did not continue to hold for the 16 $MHz$ cases. Instead, we saw a dramatic increase in the energy required for our security operations. This increase is primarily tied to the increased supply voltage required for the board to operate at 16 $MHz$. At lower frequencies we were able to successfully power the board with a supply voltage of 3.5 $V$; for successful operation at 16 $MHz$ it was necessary to increase our supply voltage to 4 $V$ dramatically increasing the power consumption of the testbed.

In addition to the increased power consumption from a higher supply voltage, operation at 16 $MHz$ introduced the use of FRAM wait-states, limiting the benefit of the increased operational frequency as cycles were wasted waiting for FRAM operations to complete. Combined, these two effects conspired to increase the power consumption of the testbed at 16 $MHz$ and highlight a potential design consideration for future intermittent systems that have similar energy consumption profiles. If the static power consumption of the device is sufficiently larger than the dynamic power consumption, increased operational frequency may yield reduced overall power consumption as long as the supply voltage can be kept in line with lower frequency operation.

The code size overhead of our solution is shown in Table 3.1 and is less interesting than the energy consumption measurements. The `SW-SIC` code required the largest space as `HW-SIC` was able to omit the software AES functions implemented within the on-board AES module. Similarly, the reduction in code size between `-O3` and `-Os` optimizations is larger for `HW-SIC`

Table 3.1: Effect on Code Size, `.text` (B)

| **Optimization** | NON-SIC | HW-SIC | SW-SIC |
|---|---|---|---|
| -O0 | 11,936 | 18,516 | 35,728 |
| -O3 | 6,932 | 14,556 | 21,432 |
| -Os | 6,916 | 10,716 | 19,808 |

as the Chaskey implementation functions did not significantly change in size only reducing from 4694 $B$ to 4488 $B$.

Finally, we are able to show that these techniques are feasible on devices that lack a hardware accelerator if an appropriate lightweight cryptographic primitive is chosen. Chaskey is able to provide an equivalent level of security guarantee compared to the hardware accelerated AES based CMAC computation with only 32.73% in additional energy required. This may prove very useful on previously deployed platforms that may lack an on-board AES module but still benefit from operations with secure checkpoints.

## 3.6 Conclusion

This chapter demonstrated a simple protocol to verify the integrity, authenticity, and freshness of an intermittent system's checkpoints. Our solution was extensively measured and compared with a non-secure version of the underlying checkpointing system across a variety of system configurations to examine their effects on energy consumption, execution time, and code size. Ultimately, we show that verification of checkpoints is both necessary and expensive. It is impossible to ignore the security requirements of the real world, but previous works avoid this complication of the intermittent computing space. We show that proper protections for application continuity can be implemented, but the current cost is high and opens the door to future work in improving the energy and performance overhead

of checkpoint verification.

# Bibliography

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

[2] Nathan Burow, Scott A. Carr, Joseph Nash, Per Larsen, Michael Franz, Stefan Brunthaler, and Mathias Payer. Control-flow integrity: Precision, security, and performance. *ACM Comput. Surv.*, 50(1):16:1–16:33, April 2017.

[3] 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.

[4] L. Davi, M. Hanreich, D. Paul, A. R. Sadeghi, P. Koeberl, D. Sullivan, O. Arias, and Y. Jin. Hafix: Hardware-assisted flow integrity extension. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[5] Daniel Dinu, Alex Biryukov, Johann Großschädl, Dmitry Khovratovich, Yann Le Corre, and Léo Perrin. Felics–fair evaluation of lightweight cryptographic systems. In *NIST Workshop on Lightweight Cryptography*, 2015.

[6] M J Dworkin. Recommendation for block cipher modes of operation: The CMAC mode for authentication. Technical report, 2016.

[7] Karim Eldefrawy, Aurélien Francillon, Daniele Perito, and Gene Tsudik. SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust. In *NDSS 2012, 19th Annual Network and Distributed System Security Symposium, February 5-8, San Diego, USA*, San Diego, UNITED STATES, 02 2012.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] Nicky Mouha. Chaskey: a mac algorithm for microcontrollers – status update and proposal of chaskey-12 –. Cryptology ePrint Archive, Report 2015/1182, 2015. https://eprint.iacr.org/2015/1182.

[13] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey: An efficient mac algorithm for 32-bit microcontrollers. Cryptology ePrint Archive, Report 2014/386, 2014. https://eprint.iacr.org/2014/386.

[14] Santosh Nagarakatte, Jianzhou Zhao, Milo M.K. Martin, and Steve Zdancewic. Softbound: Highly compatible and complete spatial memory safety for c. In *Proceedings*

*of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 245–258, New York, NY, USA, 2009. ACM.

[15] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 479–494, Berkeley, CA, USA, 2013. USENIX Association.

[16] Vijay Raghunathan, Aman Kansal, Jason Hsu, Jonathan Friedman, and Mani Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN '05, Piscataway, NJ, USA, 2005. IEEE Press.

[17] Texas Instruments. *MSP430 FRAM Quality and Reliability*, 2012. Revised May 2014.

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

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

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

[21] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson. Phase change memory. *Proceedings of the IEEE*, 98(12):2201–2227, Dec 2010.

[22] Joel Van Der Woude and Matthew Hicks. Intermittent computation without hardware support or programmer intervention. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 17–32, GA, 2016. USENIX Association.

# Chapter 4

# Conclusions

This thesis explored the security of intermittent devices in two different conditions: an abundance of harvested energy and periods of insufficient energy. Both conditions present unique opportunities and challenges for enabling secure operations.

Through the use of excess energy to precompute cryptographic operations this thesis has shown it is possible to dramatically reduce the runtime cost of cryptographic operations on intermittent devices. The employment of a *coupon* system may enable the use of much stronger security measures, more powerful cryptographic primitives, than would otherwise be available on an intermittent device. Metrics were proposed for evaluating the cryptographic operations considered for precomputation and a case study was performed to demonstrate the effectiveness of precomputation in reducing the runtime energy cost of both true random number generation and AES-CTR one-time pad generation. Taken together this may serve as a base for future intermittent devices capable of stronger cryptographic operations.

When insufficient energy exists to continuously operate an embedded device, intermittent operation via checkpoints may serve as a solution. This thesis explored the energy cost of ensuring the authenticity and integrity of such system checkpoints across a variety of system configurations. No previous works have considered the security implications of system checkpoints for intermittent devices nor the potential energy overhead incurred when preventing an adversary from tampering with the stored system state. A simple protocol was implemented to ensure that stored checkpoints are robust against tamper by an adversary

with access to the intermittent device's NVM and the energy overhead was measured for both hardware supported and software only cryptographic primitives. These overhead values were then presented with the baseline, non-secure, energy costs for intermittent operation to demonstrate the large cost incurred when securing intermittent device checkpoints.

Future work may successfully explore methods in reducing the energy cost for securing intermittent operations, perhaps through the use of precomputation during periods of excess energy, and ultimately enable a more secure paradigm for energy harvested devices.

# 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] 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.

[3] 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. ISSN 1943-0663. doi: 10.1109/LES.2014.2371494.

[4] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini. Hibernus++: A self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design*

*of Integrated Circuits and Systems*, 35(12):1968–1980, 2016. ISSN 0278-0070. doi: 10.1109/TCAD.2016.2547919.

[5] Victor Boyko, Marcus Peinado, and Ramarathnam Venkatesan. *Speeding up discrete log and factoring based schemes via precomputations*, pages 221–235. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-69795-4. doi: 10.1007/BFb0054129. URL https://doi.org/10.1007/BFb0054129.

[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. ISBN 978-3-540-47555-2. doi: 10.1007/3-540-47555-9_18. URL https://doi.org/10.1007/3-540-47555-9_18.

[7] 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.

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

[9] 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. ISBN 978-1-4503-3468-6. doi: 10.1145/2737924.2737978. URL http://doi.acm.org/10.1145/2737924.2737978. DINO.

[10] Benjamin Ransford and Brandon Lucia. Nonvolatile memory is a broken time machine. In *Proceedings of the Workshop on Memory Systems Performance and Correct-*

*ness*, MSPC '14, pages 5:1–5:3, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2917-0. doi: 10.1145/2618128.2618136. URL http://doi.acm.org/10.1145/2618128.2618136.

[11] Benjamin Ransford, Shane Clark, Mastooreh Salajegheh, and Kevin Fu. Getting things done on computational rfids with energy-aware checkpointing and voltage-aware scheduling. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, pages 5–5, Berkeley, CA, USA, 2008. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1855610.1855615.

[12] 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.

[13] 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.

[14] Charles Suslowicz, Archanaa S. Krishnan, and Patrick Schaumont. Optimizing cryptography in energy harvesting applications. In *Proceedings of the 2017 Workshop on Attacks and Solutions in Hardware Security - ASHES 2017*. ACM Press, 2017. doi: 10.1145/3139324.3139329. URL https://doi.org/10.1145/3139324.3139329.

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

[16] Joel Van Der Woude and Matthew Hicks. Intermittent computation without hardware support or programmer intervention. In *12th USENIX Symposium on Operating Sys-*

*tems Design and Implementation (OSDI 16)*, pages 17–32, GA, 2016. USENIX Association. ISBN 978-1-931971-33-1. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/vanderwoude.