Optimizing Cryptography in Energy Harvesting Applications

Charles Suslowicz
Virginia Tech
Blacksburg, Virginia
cesuslow@vt.edu

Archanaa S. Krishnan
Virginia Tech
Blacksburg, Virginia
archanaa@vt.edu

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

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 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.

KEYWORDS

energy harvesting, precomputation, random number generation, internet of things

1 INTRODUCTION

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

The domain of intermittent computing contains a significant amount of work to address power loss during 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.

ASHES'17, November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-5397-7/17/11…$15.00
http://dx.doi.org/10.1145/3139324.3139329

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.

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

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

2 BACKGROUND

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

Table 1: Data and Energy Retention Time

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

2.1 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 partial completed operation, and subsequent operations will fail due to these faulty or unexpected inputs [18] [9].

The volatile nature of power for energy harvested systems highlights the stable nature of data stored in non-volatile memory compared to the retention of energy stored in a battery or supercapacitor. Energy within a supercapacitor will discharge based on the leakage current and surrounding circuitry at a relatively quick rate. A rechargeable battery will retain the same energy for a longer period of time, but will also eventually discharge even if the device has not executed any operations but no additional energy is provided [27] [19].

When that energy is converted to a *coupon* and stored in non-volatile memory, it can be maintained in FRAM for 100 years at room temperature (20° C) and 10 years in extreme conditions (85° C). The stability of this data is illustrated on Table 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

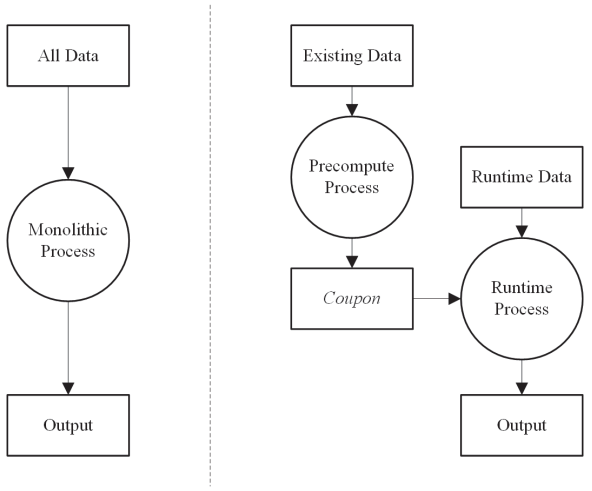

Figure 1: The process for a single operation, shown on left, and the precomputed operation, shown on right. When combined, the *coupon* and runtime data, available only immediately before execution, allow the generation of an output identical to the single, monolithic, process.

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.

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

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

Our proposal for precomputation with energy harvested devices provides a solution to the scaling problem facing new IoT devices without the drawbacks demonstrated by recent hardware proposals and with full interoperability with existing intermittent computing paradigms.

2.4 Threat Model

A multitude of threats exist for energy harvested systems, especially those deployed in remote and unsecured areas. In recognition

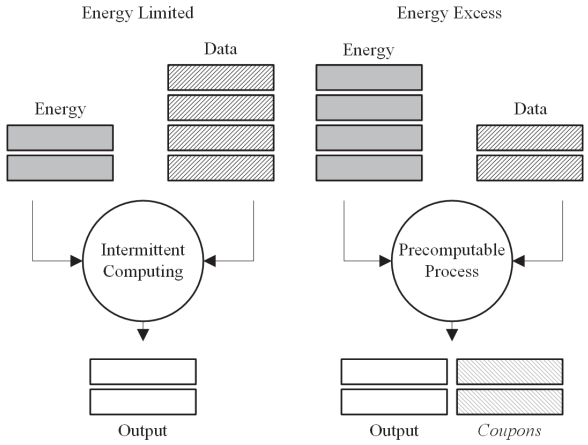

Figure 2: Intermittent computing ensures that as much output as possible is created during a scarcity of energy. Our work ensures that excess energy is utilized to improve the efficiency of future operations with *coupons*.

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.

3 PRECOMPUTATION, ENERGY HARVESTED DEVICES, AND CRYPTOGRAPHY

How can the latency and efficiency of cryptographic operations on these platforms be improved? First, we evaluate the limiting factors of energy harvested platforms for cryptographic operations, Second, we evaluate the partitioning of cryptographic algorithms, the use of intermediate value *coupons* and their effect on process execution. Finally, we consider a framework of metrics for evaluating the benefit to a particular implementation in terms of energy efficiency.

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

3.1 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.

3.2 *Coupons* and the Precomputation of Algorithms

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

The generation of a *coupon* will be unique to each cryptographic operation, but in all cases it represents a function that accepts some input data not dependent on runtime parameters and some amount of energy to produce an intermediate data block in the operation. This process converts energy that would normally be stored in an energy storage medium, a supercapacitor or rechargeable battery, into data that can be stored on silicon. By executing this conversion, the energy harvesting system converts energy into data for a future operation thereby reducing the energy required to produce the final output at runtime as illustrated by the equations in (1).

$$\begin{aligned} E_{original} &\leq E_{precomputation} + E_{runtime} \\ E_{runtime} &< E_{original} \end{aligned} \tag{1}$$

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.

3.3 Metrics for Comparison

To properly evaluate the effectiveness of *coupon* precomputation we considered the energy required to complete an operation, the operation's cycle count, an operation's delay, and the 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 r. In general it is expected

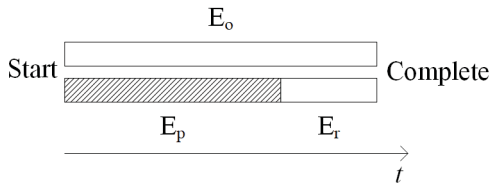

Figure 3: Illustration of the energy required for a monolithic computation versus seperation into a precomputation and runtime operation.

that the following relationships are true:

$$\begin{aligned} E_o &\leq E_p + E_r \\ C_o &\leq C_p + C_r \\ D_o &\leq D_p + D_r \end{aligned} \tag{2}$$

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:

$$\begin{aligned} \text{Speedup} : C_i &= \frac{C_o}{C_r} \\ \text{Energy Improvement} : E_i &= \frac{E_o}{E_r} \\ \text{Latency Improvement} : D_i &= \frac{D_o}{D_r} \\ \text{EDP Improvement} : EDP_i &= \frac{EDP_o}{EDP_r} \end{aligned} \tag{3}$$

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

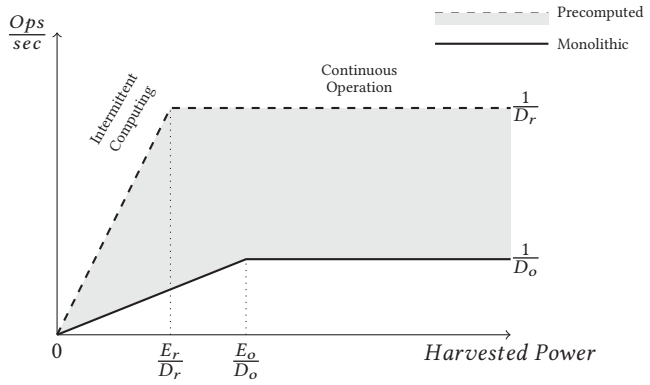

Figure 4: Operations per second as a function of Energy influx into the system. When *coupons* are available the device is able to execute more operations within a given time period until limited by the latency of the minimum runtime computation (D_r).

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 4.

3.4 Conversion of Energy to Data via Precomputation

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

The transformation of energy to *coupons* for future use allows us to exploit the improved data storage capacity of modern energy harvesting systems and improve the runtime performance of our cryptographic operations. This is illustrated by Figure 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 4 as $\frac{E_o}{D_o}$ for the original operation and $\frac{E_r}{D_r}$ for the runtime operation with *coupons*.

3.5 Effect of Precomputation on Security

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

3.5.1 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.

3.5.2 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.

3.5.3 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.

3.5.4 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.

4 CASE STUDIES

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

Table 2: Key features of MSP430FR5994 and MSP432P401R

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

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

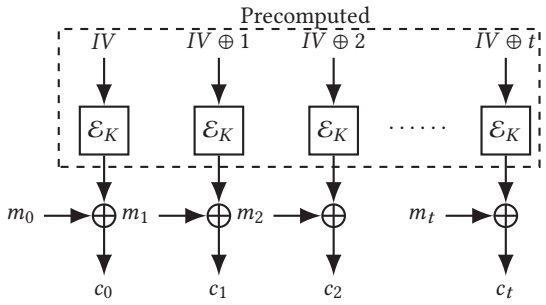

Figure 5: Block diagram of counter mode operation [15] with precomputable portion highlighted.

4.1 Experimental setup

We have used the Texas Instruments(TI) MSP430FR5994 and the TI SimpleLink MSP432P401R launchpad development kits in our case studies. Different styles of TRNG were implemented on the MSP430FR5994 and AES-CTR mode was implemented on both the devices. Table 2 lists some important features that makes the selected devices ideal to be used as an 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.

4.2 AES counter mode

AES as a block cipher can be used in different modes of operation to encrypt messages that are longer than one block of data. In 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 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 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.

```
char *aes_ctr_monolithic(char *key, char *ctr,
    char *PT) {
  while(blocks < 8) {
    aes_encrypt(char *ctr,char *key);
    increment_counter(char *ctr);
    xor_mask(char *PT, char *OTP, char *CT);
  }
  return CT;
}
```

Figure 6: Pseudo-code for Monolithic AES-CTR

Table 3: Cost of Monolithic AES-CTR encryption

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

```
char *aes_ctr_online(char *PT,
    char *precomp-coupons) {
  while(blocks < 8) {
     xor_mask(char *PT, char *precomp-coupons,
         char *CT);
  }
  return CT;
}
```

Figure 7: Pseudo-code for precomputed AES-CTR

4.2.1 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 6 to encrypt a message.

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

When the whole encryption is done in one online stage, we measured a delay of 6055 μs to finish encrypting a 128 byte block using T-box implementation of software AES in MSP432P401R (Table 3). This delay is proportional to the latency of algorithm at runtime.

4.2.2 AES-CTR with precomputation. The above program in Figure 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 7, which greatly reduces the runtime energy requirement.

Table 4 gives a clear picture of the cost of XOR masking in both MCUs. Since AES block cipher operations are precomputed in the

Table 4: Runtime Cost of AES-CTR with precomputed OTP

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

Table 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

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 MSP432P401R is 13.8 μJ.

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

It can be seen that there is a vast improvement in runtime latency, energy requirement and security in AES-CTR mode when OTPs are precomputed. The EDP improvement for the AES-CTR implementations using hardware co-processors is lower than other implementations listed in Table 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 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 co-processor
`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

4.3 Hardware Random Number Generator

This case study analyzes a true random number generator as a possible best case situation for the precomputation of *coupons*. 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).

4.3.1 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 6 identifies the specific TRNGs used within the case study and the label associated with that TRNG's results throughout our collected data.

4.3.2 True Random Number Generation Without Precomputation. In normal operation, when a program requests a random value execution is handed off to a 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

Table 7: TRNG Measurements and Precomputation

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

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

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.

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

4.3.4 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.

5 FUTURE WORK

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

ensure future devices are always executing in the most efficient manner.

6 CONCLUSION

This paper 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.

ACKNOWLEDGMENTS

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

REFERENCES

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

[2] Giuseppe Ateniese, Giuseppe Bianchi, Angelo T. Capossele, Chiara Petrioli, and Dora Spenza. 2017. Low-Cost Standard Signatures for Energy-Harvesting Wireless Sensor Networks. *ACM Trans. Embed. Comput. Syst.* 16, 3, Article 64 (April 2017), 23 pages. https://doi.org/10.1145/2994603

[3] Aydin Aysu and Patrick Schaumont. 2015. 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. 2015. Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems. *IEEE Embedded Systems Letters* 7, 1 (March 2015), 15–18. https://doi.org/10.1109/LES.2014.2371494

[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. 2010. *SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.* Technical Report. Gaithersburg, MD, United States.

[6] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David B. Wilson. 1993. *Fast Exponentiation with Precomputation.* Springer Berlin Heidelberg, Berlin, Heidelberg, 200–207. https://doi.org/10.1007/3-540-47555-9_18

[7] Michael Buettner, Richa Prasad, Alanson Sample, Daniel Yeager, Ben Greenstein, Joshua R. Smith, and David Wetherall. 2008. RFID Sensor Networks with the Intel WISP. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys '08)*. ACM, New York, NY, USA, 393–394. https://doi.org/10.1145/1460412.1460468

[8] Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. 2017. Bespoke Processors for Applications with Ultra-low Area and Power Constraints. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 41–54. https://doi.org/10.1145/3079856.3080247

[9] Alexei Colin and Brandon Lucia. 2016. Chain: Tasks and Channels for Reliable Intermittent Programs. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2016)*. ACM, New York, NY, USA, 514–530. https://doi.org/10.1145/2983990.2983995

[10] H. Diewald, G. Zipperer, P. Weber, and A. Brauchle. [n. d.]. Electronic Device and Methods for Tracking Energy Consumption. ([n. d.]). https://www.google.com/patents/US20160077138,year=

[11] Viktor Fischer and Miloš Drutarovský. 2001. *Two Methods of Rijndael Implementation in Reconfigurable Hardware.* Springer Berlin Heidelberg, Berlin, Heidelberg, 77–92. https://doi.org/10.1007/3-540-44709-1_8

[12] Matthew Hicks. 2017. Clank: Architectural Support for Intermittent Computation. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 228–240. https://doi.org/10.1145/3079856.3080238

[13] J. Hsu, S. Zahedi, A. Kansal, M. Srivastava, and V. Raghunathan. 2006. Adaptive Duty Cycling for Energy Harvesting Systems. In *ISLPED'06 Proceedings of the 2006 International Symposium on Low Power Electronics and Design*. 180–185. https://doi.org/10.1145/1165573.1165616

[14] H. Jayakumar, A. Raha, and V. Raghunathan. 2014. 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*. 330–335. https://doi.org/10.1109/VLSID.2014.63

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

[16] Aman Kansal, Jason Hsu, Mani Srivastava, and Vijay Raghunathan. 2006. Harvesting Aware Power Management for Sensor Networks. In *Proceedings of the 43rd Annual Design Automation Conference (DAC '06)*. ACM, New York, NY, USA, 651–656. https://doi.org/10.1145/1146909.1147075

[17] Bin Liu and Bevan M. Baas. 2013. Parallel AES Encryption Engines for Many-Core Processor Arrays. *IEEE Trans. Computers* 62 (2013), 536–547.

[18] Brandon Lucia and Benjamin Ransford. 2015. 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)*. ACM, New York, NY, USA, 575–585. https://doi.org/10.1145/2737924.2737978 DINO.

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

[20] Philippe Oechslin. 2003. *Making a Faster Cryptanalytic Time-Memory Trade-Off.* Springer Berlin Heidelberg, Berlin, Heidelberg, 617–630. https://doi.org/10.1007/978-3-540-45146-4_36

[21] A. Orda and A. Sprintson. 2003. Precomputation schemes for QoS routing. *IEEE/ACM Transactions on Networking* 11, 4 (Aug 2003), 578–591. https://doi.org/10.1109/TNET.2003.815299

[22] S. Pelissier, T. V. Prabhakar, H. S. Jamadagni, R. VenkateshaPrasad, and I. Niemegeers. 2011. Providing security in energy harvesting sensor networks. In *2011 IEEE Consumer Communications and Networking Conference (CCNC)*. 452–456. https://doi.org/10.1109/CCNC.2011.5766511

[23] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2011. Mementos: System Support for Long-running Computation on RFID-scale Devices. *SIGARCH Comput. Archit. News* 39, 1 (March 2011), 159–170. https://doi.org/10.1145/1961295.1950386

[24] Mastooreh Salajegheh, Shane S Clark, Benjamin Ransford, Kevin Fu, and Ari Juels. 2009. CCCP: Secure Remote Storage for Computational RFIDs.. In *USENIX Security Symposium*. 215–230. https://spqr.eecs.umich.edu/papers/salajegheh-CCCP-usenix09.pdf

[25] Weidong Shi, H. S. Lee, M. Ghosh, Chenghuai Lu, and A. Boldyreva. 2005. High efficiency counter mode security architecture via prediction and precomputation. In *32nd International Symposium on Computer Architecture (ISCA'05)*. 14–24. https://doi.org/10.1109/ISCA.2005.30

[26] Farhan Simjee and Pai H. Chou. 2006. Everlast: Long-life, Supercapacitor-operated Wireless Sensor Node. In *Proceedings of the 2006 International Symposium on Low Power Electronics and Design (ISLPED '06)*. ACM, New York, NY, USA, 197–202. https://doi.org/10.1145/1165573.1165619

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

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

[29] Patrick P. Tsang and Sean W. Smith. 2008. *Secure Cryptographic Precomputation with Insecure Memory.* Springer Berlin Heidelberg, Berlin, Heidelberg, 146–160. https://doi.org/10.1007/978-3-540-79104-1_11

[30] Lane Westlund. 2006. Random Number Generation Using the MSP430. (2006).