Noname manuscript No.
(will be inserted by the editor)

Secure and Stateful Power Transitions in Embedded Systems

Archanaa S. Krishnan · Charles Suslowicz · Patrick Schaumont

Received: date / Accepted: date

Abstract Power loss occurs in devices with a transient power supply, and it leads to the loss of volatile state information of the device. To protect the state, the device stores it as a checkpoint in non-volatile memory. The checkpoints are used to restore the device to the most recent stored state upon power up. There are three facets of power transitions - cause, statefulness, and security, out of which the third facet is ignored in current embedded systems research. In this paper, we describe the intersection of two fields, stateful power transitions and secure embedded systems, which has largely been unexplored until now. We study the limitations introduced by the three facets of power transitions of embedded devices. We explore the vulnerabilities introduced by stateful power transitions and propose the Secure Intermittent Computing Protocol to overcome them. We analyze the overhead of each technology required to provide secure and stateful power transition and its effects on the duty cycle of an embedded device.

Keywords Secure checkpoints · Intermittent computing · Energy harvesters · Non-volatile memory · Embedded systems

A. S. Krishnan
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA, 24060 USA
E-mail: archanaa@vt.edu

C. Suslowicz
Army Cyber Institute
U.S. Military Academy
West Point, NY, 10666 USA
E-mail: charles.suslowicz@westpoint.edu

P. Schaumont
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA, 24060 USA
E-mail: schaum@vt.edu

1 Introduction

Computers including servers, personal computers (PCs), laptops, and embedded devices, run on electric power, which is typically supplied by the grid. Power loss, a fact of life, is a short-term or long-term shortage of power which causes computer shut downs. Upon power loss, the device transitions from ON-state to OFF-sate, losing its volatile computer state. Upon the next power-up, it transitions to ON-state and re-initializes the volatile state, thus power loss re-initializes the system on every power-up. The transition between ON, OFF, and ON-state is called power transition. The computer copes with power loss by storing checkpoints of the intermediate volatile state in non-volatile memory, illustrated in Figure 1. Non-volatile memory ensures that checkpoints remain persistent across power transitions. Upon power up, the computer is restored to the most recent checkpointed state and resumes its tasks.

In this paper, we focus on the power transitions of a secure embedded system. Energy harvesting technology converts ambient energy to electrical energy, which is sufficient to power resource constrained embedded devices. Figure 1 illustrates a device powered by a solar energy harvester. Since the availability of solar energy depends on the weather and time of the day, a solar energy harvester is a transient power source. Transient power supplies do not provide continuous power which causes power loss in embedded systems. To cope with power loss, the device is equipped with non-volatile memory. Although the device's non-volatile memory retains its data during power-off, the volatile state information is lost. Non-volatile memory by itself is insufficient to ensure forward progress of the application [28]. Intermittent computing is a stateful power transition technology, where the device stores a snapshot of the

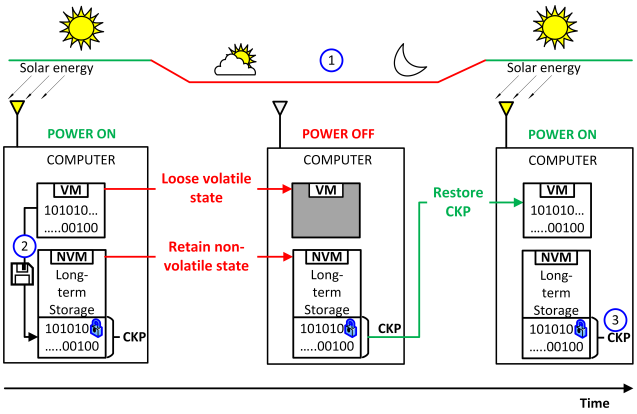


Fig. 1 The three facets of power transition: (1) Cause - lack of solar energy causes power loss, (2) Statefulness - the state of volatile memory, VM, is stored as checkpoints, CKP, in non-volatile memory, NVM, and (3) Security - checkpoints are protected in NVM.

volatile state information in non-volatile memory, as a checkpoint (CKP). The checkpoint is used to restore the device to the last known state to ensure forward progress of the application. The state-of-the-art intermittent computing techniques provide efficient checkpoint generation and restoration solutions to ensure forward progress with minimum overhead [28, 19, 29, 43]. As the checkpoints contain intermediate state of the device, they must be secured to protect power transitions.

Figure 1 illustrates the three facets of power transitions - cause, statefuleness, and security. In general purpose computers, such as servers, PCs and laptops [16, 30], all three facets of power transitions are on-going research problems. Whereas in embedded systems, only the cause [1, 39, 14] and statefulness [19] are commonly explored. Although security for embedded systems is an on-going research problem [31], the security of power transitions is widely ignored. In this paper, we highlight the need for secure power transitions in embedded systems through the following contributions:

- We study the security vulnerabilities introduced by stateful power transitions and analyze the need for secure power transitions.
- We propose the Secure Intermittent Computing Protocol (SICP) to overcome these vulnerabilities. We describe a real-life application that requires checkpoint security, which can benefit from SICP.
- We quantify power transitions in embedded systems by computing their duty cycle based on the amount of energy available from the harvester, the overhead of checkpoint generation and restoration process, and the overhead incurred to secure stateful power transitions. We demonstrate that secure and stateful power transitions are expensive but achievable in embedded systems.

The security vulnerabilities form checkpoints were first introduced in 2018 at the 8th International Conference on Security, Privacy, and Applied Cryptographic Engineering [36]. SICP was first proposed in 2019 at the Design Automation and Test in Europe [24]. In this paper, we extend and improve on the background of power transitions in embedded systems. We evaluate the need for SICP using a real-life application, and quantify its effects on the duty cycle of the application.

Organization The rest of the paper is organized as follows. Section 2 provides a brief background on the different facets of power transition and their effects on embedded systems. Section 3 discusses our attacker model, locates checkpoint vulnerabilities, and provides a set of security requirements for checkpoints. Section 4 proposes SICP to satisfy these security requirements followed by our implementation of SICP in Section 5. Section 6 evaluates SICP by introducing the need for checkpoint security to a real-life application and by studying the overhead of statefulness and security in power transition and its effect on the duty cycle of the application, followed by our conclusions in Section 7.

2 Background in Power Transitions

In this section, we define the different facets of power transitions and analyze their effects on embedded systems. The three facets of power transition are defined as follows:

1. Cause: The root cause of power loss helps identify the frequency, period of power loss, and other characteristics which help design coping mechanisms for the computer system.
2. Statefulness: A stateful power transition is aware of the intermediate state of the computer system. Through statefulness, the computer maintains its state during power loss which is used in future computations. It ensures the forward progress of the application.
3. Security: The security of a power transition is the guarantee that the state of the computer system is protected from data corruption and unauthorized access even during power loss. It preserves the security features of both the device and application across power loss.

We analyze the problems introduced by transient power supplies, describe the use of statefulness to cope with these problems and demonstrate the need for security in stateful power transitions.

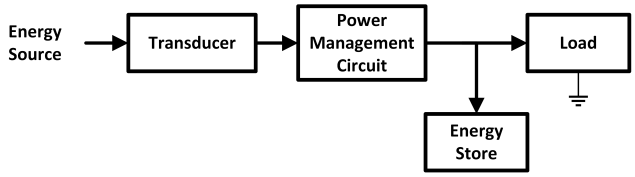


Fig. 2 Three components of an energy harvester: (1) a transducer to convert ambient energy to electrical energy, (2) a power management circuit to adjust the harvested energy based on the needs of the load, and (3) an energy storage buffer to deliver high power to the load

2.1 Cause

Energy harvesters extract energy from ambient energy sources, such as heat [39], vibration [14], and radiation [27], and convert it to electrical energy to power embedded devices. The ambient energy is processed in three steps before it is consumed by the load, illustrated in Figure 2. First, a transducer converts ambient energy to electrical energy. Second, a power management circuit efficiently manages the harvested energy based on the requirements of the load. Since the harvester typically supplies low power, third, a supercapacitor or a battery is used as an energy storage buffer to accumulate the harvested electrical energy to supply bursts of high power to the load.

The ambient energy sources depend on external factors, including, but not limited to, weather, time of day, human activity, and location of the harvester. For example, sunlight is only available during daytime and is dependent on the weather conditions; wind energy is similarly dependent on the weather; kinetic energy is dependent on machine or human motion. This dependency limits the available ambient energy and causes a harvester to supply intermittent power to its load.

$$D_{Load} = \frac{P_{EH}}{P_{Load}} \times 100 \quad (1)$$

Equation 1 states the relation between the load duty cycle, D_{load}, the average power available from the harvester, P_{EH}, and the net power required by the load P_{load}. If the load was supplied by a constant power supply, the supply and demand will match, i.e., P_{EH} would equal P_{load}, in which case the device would operate at 100% duty cycle. The harvested power typically does not match the power required by the load. For example, consider a load which performs a cryptographic signature [32] powered by a kinetic energy harvester [9]. Each signature requires 7.3 mW (P_{load}), whereas the harvester only supplies an average of 2 mW (P_{EH}). The load can only operate at a 27% duty cycle to compute signatures. Thus, the duty cycle is determined by the power budget available from the energy harvester.

Since the load may require more power than the harvester's output, it is bound to lose power during its computation unless the energy storage buffer is large enough to satisfy its requirements. The storage buffer, which is usually a battery or supercapacitor, accumulates the energy until it can deliver sufficient power to the load. Supercapacitors are well suited for energy harvesting applications, as they provide infinite charge/discharge life cycles, fast recharge rate and high power density compared to batteries. In the above example, the load requires 7.3mW in 12.5s to compute a cryptographic signature, which requires 91mJ of energy. When the load is supplied by a 3V input voltage, it requires a minimum of 0.02F supercapacitor to supply the power required to compute one signature. The number of signatures that can be computed before a power loss occurs depends on the size of the energy buffer which is typically small to reduce the size of the energy harvesting circuit and capacitor charge time. The load experiences a power loss after it exhausts its energy buffer.

Conventionally, after each power cycle the device is reinitialized and loses the progress made during the previous power on state, restarting the application every time. Stateful power transitions are needed to avoid re-initialization after every power loss.

2.2 Statefulness

The intermittent computing model, a stateful power transition technique, was introduced to guarantee forward progress of long-running applications when powered by an intermittent power supply. All the state information necessary to restore the device is stored as a checkpoint in non-volatile memory. A checkpoint consists of the system state, such as processor registers, peripheral registers, and application state, such as stack, heap, and developer defined variables that are required to resume program execution. After a power cycle, the device is restored to the last known checkpointed state.

Several intermittent computing techniques have been proposed, among which a majority optimize two criteria, energy efficiency and rollback minimization. The latter also ensures that the former is achieved by preventing re-execution of completed tasks. The state-of-the-art techniques use various techniques, such as architectural support [19], energy aware checkpoint calls [21], kernel-oriented design [5], task based programming and execution model [28], non-volatile processors [23], and, probabilistic algorithms [15] to obtain energy efficient checkpoints.

Irrespective of the checkpointing technique in use, the device transitions through two states, ON-state and OFF-state. During the ON-state, the device performs

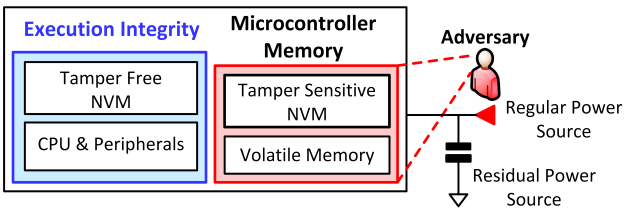


Fig. 3 The architectural assumptions and memory model for SICP illustrating the assumed attacker model with two capabilities - (1) control power supply to the device and (2) view and modify tamper sensitive non-volatile memory during power-off periods.

its regular tasks. It may employ protection features such as control flow integrity [11], attestation and isolation [31], and protection against cold-boot attacks [17]. The variables required to implement these security features must also be checkpointed to ensure the continuation of these security properties in future ON-states. During the OFF-state, the checkpoint remains in non-volatile memory. The checkpoint contains the intermediate state of the application, which may be a cryptographic algorithm, and the critical settings of the security features employed during ON-state, such as kernel privileges and memory access rights. A majority of the intermittent computing techniques store their checkpoints as plaintext in non-volatile memory. A few techniques explore security in power transitions [15, 42] but they do not provide a comprehensive security solution. The existing secure power transition solutions from general purpose computers [7, 8, 16] cannot be used in embedded systems because they were not designed for resource constrained devices.

3 Problem Description

Checkpoints, which are generated to provide stateful power transitions, introduce vulnerabilities to an embedded device which may otherwise be secure when it is powered on. In this section, we define our attacker model, describe the risks introduced by unprotected checkpoints, and list a set of minimum security requirements to protect power transitions against the assumed attacker model.

3.1 Attacker model

The attackers aims to gain useful information from the intermittent execution model. We define an attacker model with the following capabilities, illustrated in Figure 3, to study the security vulnerabilities introduced by checkpoints.

1. The attacker has complete control over the power supplied to the device. The attacker can arbitrarily stop the application on the target device, for example, the attacker can tamper with the energy harvester input to control the input to the target device. The aim of the attacker is not to completely stop the application on the target device, but to stop the target device at strategic points in the application to gain information from the checkpoints. Thus, denial of service by cutting off power supply is out of scope of this attacker model.
2. The attacker has access to the majority of the device memory when it is powered off. The attacker can read from and write to the unprotected non-volatile memory, which we call tamper-sensitive non-volatile memory. In this scenario, even though the device must be powered on to access the contents of memory, the CPU is still not powered-on, i.e, the processor is in idle state. For example, the attacker can access the memory by providing read/write commands to Direct Memory Accessc(DMA) via debug probes. Since DMA is independent of the processor, the attacker need not power-on the processor to access memory [33].

We assume that the device is equipped with a tamper-free non-volatile memory, which is secure from the assumed attacker model. This requirement can be satisfied by using an off-the-shelf microcontroller with secure non-volatile memory, such as Maxim's ZA9L1 [2]. For example, the secure memory may only be accessible from authorized code and unauthorized access may lead to zeroization of secure memory. We assume that the device is physically protected from the attacker. The attacker cannot access the device memory during ON-state, the volatile and non-volatile system states are inaccessible to the attacker when the device is powered on. We assume that the device's execution integrity and memory protection during power-on states are guaranteed by a protected embedded software execution environment [31]. The mitigation of side channel and fault injection attacks on the checkpointing system are beyond the scope of this work.

3.2 Checkpoint vulnerabilities

Non-volatility of persistent memory compromises the privacy of unsecured persistent data. The state-of-the-art non-volatile memory protections are not designed for resource constrained devices [22, 40]. The state-of-the-art intermittent computing techniques also fail to secure their checkpoints. Checkpoints consist of the volatile and non-volatile state of a device, which may

contain sensitive data. When left unsecured, they introduce the following vulnerabilities to an intermittent system.

- *Checkpoint snooping*: The attacker can read the non-volatile memory, and in turn read the checkpoints to extract sensitive information stored in them as the checkpoints are stored as plaintexts. Non-volatile data, which is otherwise private during power-on, is now open to attackers in checkpoints. The attacker can study the checkpoints to identify the location of sensitive information [36]. While checkpoint encryption may provide protection against snooping [15], it does not protect against the other vulnerabilities.
- *Checkpoint spoofing*: The state-of-the-art intermittent computing techniques simply restore a checkpoint, if one exists, without checking its integrity. With the knowledge of the location of sensitive variables, the attacker can spoof checkpoints by modifying them in non-volatile memory. Unknowingly, the device restores itself with a modified checkpoint from where it resumes execution in an attacker controlled sequence. Encrypted checkpoints [15] are also vulnerable to spoofing as they do not guarantee integrity. The attacker can modify an encrypted checkpoint, which may not correspond to a valid checkpointed state upon decryption. When the device is powered up, it is restored with the decrypted modified encrypted checkpoint, which may lead to a system crash.
- *Checkpoint replay*: The attacker can combine snooping and spoofing to replay checkpoints. The attacker can store a copy of all the checkpoints of an intermittent system, where each checkpoint corresponds to a state of the application, to create a pool of checkpoints. The state-of-the-art intermittent computing techniques do not check if the checkpoint to be restored is indeed the latest checkpoint, which enables checkpoint replay. The attacker can overwrite the current checkpoint with any checkpoint from their pool; upon power-up, the device is restored to a stale state. A checkpoint security solution which only protects checkpoint confidentiality and integrity, such as SECCS [42], will not detect checkpoint replay.

3.3 Exploiting unsecured checkpoints

The attacker can exploit these vulnerabilities to gain access to sensitive information about the application on the device. If a device is programmed with a cryptographic algorithm, such as Advanced Encryption Standard(AES) [10], the application variables must be included in its checkpoint to ensure forward progress of the algorithm in the event of power loss. The attacker can identify the sensitive variables in a checkpoint [36], such as the intermediate state and round counter of AES. The ability to spoof checkpoints enables the attacker to replace sensitive variables of AES with attacker controlled variables and extract the secret key using cryptanalysis.

Checkpoint security is essential to ensure that the security properties of ON-states are maintained across power transitions, without any compromise. The continuous execution paradigm is shifting to an intermittent execution paradigm, which makes checkpoints an integral part of the execution environment. The existing secure software execution environments are only designed for ON-states based on the assumption that the power supply is continuous [31, 13, 11]. They propose to restart their system, including the security modules and features, when they encounter a power failure. These assumptions do not apply to a system powered by a transient power supply. Secure software execution must consider the security of both its ON-state and OFF-state, which includes checkpoint security.

3.4 Checkpoint Security Requirements

Although the security requirements may vary depending on the application and device, we must consider the following as a set of minimum requirements to overcome the vulnerabilities discussed above.

- *Information security*: The checkpoint's confidentiality, integrity, authenticity, and freshness must be ensured to protect against checkpoint snooping, spoofing, and replay.
- *Atomicity*: The checkpoint generation and restoration process must be atomic. This guarantees that the checkpoints will not be corrupted even if a power loss occurs during the checkpointing process.
- *Continuity*: Secure application continuity maintains the order of checkpoints, to provide assurance that the device is at the current state because it executed the previous states without any attacker intervention.

3.5 Architectural Assumptions

Secure and stateful power transitions require certain architectural features and protection guarantees, illustrated in Figure 3. The device must have three types of memory. First, volatile memory to store the runtime program state, which is erased upon power loss. Second,

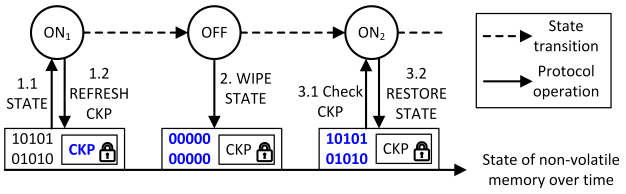


Fig. 4 A protocol scenario for secure power transitions, depicting a sequence of ON-states, OFF-states, and the corresponding state of non-volatile memory of the device. The protocol provides rules for (1) creating secure checkpoints, *CKP*, during ON_1 and for (3) restoring an unmodified *CKP*. It also ensures the protection of plaintext state by (2) overwriting it with zeros upon power loss.

tamper sensitive non-volatile memory, which does not possess any tamper resistance. Third, tamper-free non-volatile memory, which is secure against the assumed attacker model. The size of tamper free memory must be minimized to reduce hardware cost and complexity. We only place necessary variables in tamper-free memory, including the secret key and nonce, instead of placing the entire secure checkpoint in it. The rest of the secure checkpoint is placed in the tamper sensitive non-volatile memory, which is unprotected.

Apart from the different types of memory, the device must have a residual power source to provide a small, finite source of energy. For example, an on-chip or on-board capacitor may act as a residual source to power the device for a small period even after the main power supply is powered-off. Since power loss is considered a threat, sensitive variables must be wiped as soon as the device encounters a power loss. We assume that the residual power source is sufficient to wipe sensitive variables and to finish writing a 128-bit value in non-volatile memory. Since the device is physically protected from the attacker, the assumed physical protection also extends to the residual source.

4 Secure Intermittent Computing Protocol

Checkpoint security is essential, without which the security features from ON-state are lost during OFF-state. Intermittent computing techniques only ensure the forward progress of the application, the continuity of the security properties require a set of rules to detect and prevent tampering. This introduces a need for a protocol or a frame of reference to describe and achieve the security requirements discussed in Section 3.4.

We define the Secure Intermittent Computing Protocol (SICP) to protect the checkpoint vulnerabilities introduced in Section 3.2 and to ensure forward progress of the application and continuity of security properties. SICP defines a set of rules among the different

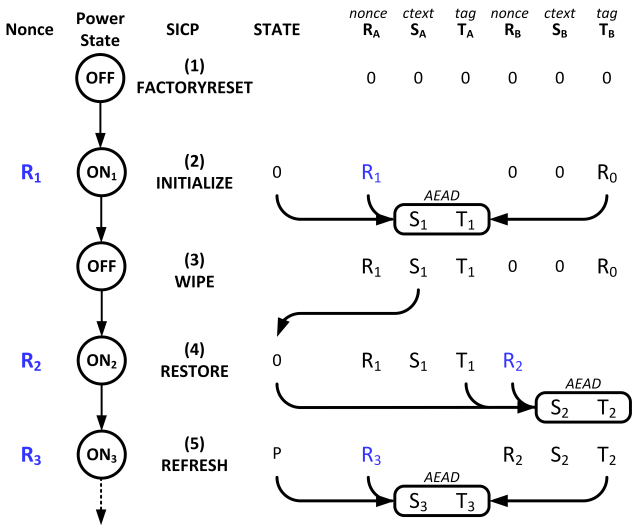


Fig. 5 An example SICP scenario. (1) The system is cleared by the *factory_reset()* operation. (2) A fresh nonce, R_i is associated with each power-on state. The first valid state save packet, SS_1, is created by `INITIALIZE`. On power loss, (3) `WIPE` clears the volatile *STATE* and upon subsequent power up, (4) `RESTORE` validates the latest state save packet, SS_1, restores the program state, and generates a new state save packet SS_2. (5) During program execution, `REFRESH` is called to create a new checkpoint SS_3, overwriting the oldest state save packet, SS_1.

states of the device, illustrated in Figure 4. The non-volatile memory, which holds the checkpoints, is the prover and the device verifies the validity of these checkpoints. During power-on, the device creates a secure checkpoint and stores it in non-volatile memory (Step 1). After a power cycle, the device verifies if the checkpoint to be restored is indeed the latest and unmodified checkpoint (Step 3). With SICP, the device can differentiate between a malicious and valid checkpoint in memory. It detects malicious checkpoints and prevents restoring the device to an attacker controlled state.

4.1 Satisfying the security requirements

We start with a device that has gone through *factory_reset()* which restores the device to manufacturer settings and programs the tamper-free non-volatile memory with a secure key, K. With the unsecured checkpoint, *STATE*, which contains the application and microcontroller data, we create a secure checkpoint in several steps, illustrated in Figure 5.

First, the freshness requirement is satisfied by associating each *STATE* with a nonce, R_i, which is stored in tamper-free non-volatile memory. *nonce()* generates a unique and fresh R_i. Second, the confidentiality, integrity and authenticity requirements are satisfied by encrypting *STATE* and R_i using Authenti-

Algorithm 1 INITIALIZE

Require: K
1: $Q \leftarrow nonce()$
2: $T_B \leftarrow nonce()$
3: $STATE \leftarrow 0$
4: $R_A \leftarrow Q$
5: $S_A \leftarrow AEAD_{encr}(STATE, T_B, R_A, K)$
6: $T_A \leftarrow AEAD_{auth}(S_A, T_B, R_A, K)$

cated Encryption with Associated Data (AEAD) [35]. $AEAD_{encr}()$ takes the plaintest $STATE$, R_i, and the non-confidential associated data as input to generate the encrypted checkpoint, S_i. $AEAD_{auth}()$ generates an authentication tag, T_i, over the newly encrypted checkpoint along with the nonce and associated data[1]. After a power cycle, if a valid authentication tag exists, it decrypts S_i using $AEAD_{decr}()$. If the authentication tag check fails, *abort()* is called to raise a violation of the protocol. At a minimum, *abort()* must either halt the device or clear the device memory and restart it. A secure checkpoint is a tuple of S_i, R_i, and T_i, which is called a state save packet, SS_i.

Third, the atomicity requirement is satisfied by storing the state save packets in a two-state buffer, SS_A and SS_B. They are updated in an alternating manner to ensure one packet is kept valid at all times. At a given point of time, the non-volatile memory will contain the latest packet, SS_i, and the previous packet, SS_{i-1}, illustrated in Figure 5. Fourth, the continuity requirement is satisfied by tag-chaining, which is the process of cryptographically chaining the authentication tags of the checkpoints in chronological order. It is achieved by using the authentication tag from the previous packet, T_{i-1} as associated data to generate the latest packet, SS_i. For example, in Figure 5, T_1 is used to compute SS_2, from which T_2 is used to compute SS_3. The authentication tags protect the integrity and authenticity of checkpoints as well as its chronological order.

4.2 Protocol

We define SICP as a collection of four algorithms described below.

INITIALIZE : The device is initialized with the first packet, SS_1, with Algorithm 1. Upon power-up, INITIALIZE is called if the device has gone through a *factory_reset()*, which is identified by a unique reset memory pattern. INITIALIZE is called only once to create SS_1, which is stored in buffer SS_A. Since the first packet has no previous authentication tag to be used as associated data, a nonce is used as associated data, T_B. This ensures a unique chain of tags are generated after every *factory_reset()*. Next, $STATE$, where the plaintext checkpoint is collected, is zeroized to overwrite the reset memory pattern to prevent future calls to INITIALIZE. A valid state save packet, SS_A, is created by encrypting and authenticating $STATE$, R_A, and T_B using AEAD to generate S_A and T_A. A state save packet, SS_i, is valid if it satisfies two conditions. First, its nonce, R_i, must match the nonce used in AEAD operations. Second, its associated data in the AEAD operations must match the authentication tag of the previous state save packet. It ensures only one packet is valid between the two buffers, SS_A and SS_B.

[1] The encryption and tag calculation in AEAD operations are separated here to provide clarity in protocol operations

Algorithm 2 REFRESH and RESTORE

Require: $K, STATE, S_i, R_i, T_i$, where $i \in \{A, B\}$
$operation \in \{\texttt{REFRESH}, \texttt{RESTORE}\}$
1: $Q \leftarrow nonce()$
2: **if** $T_A = AEAD_{auth}(S_A, T_A, R_A, K)$ **then**
3: **if** $operation = \texttt{RESTORE}$ **then**
4: $STATE \leftarrow AEAD_{decr}(S_A, T_A, T_B, R_A, K)$
5: **end if**
6: $R_B \leftarrow Q$
7: $S_B \leftarrow AEAD_{encr}(STATE, T_A, R_B, K)$
8: $T_B \leftarrow AEAD_{auth}(S_B, T_A, R_B, K)$
9: **else**
10: **if** $T_B = AEAD_{auth}(S_B, T_A, R_B, K)$ **then**
11: **if** $operation = \texttt{RESTORE}$ **then**
12: $STATE \leftarrow AEAD_{decr}(S_B, T_B, T_A, R_B, K)$
13: **end if**
14: $R_A \leftarrow Q$
15: $S_A \leftarrow AEAD_{encr}(STATE, T_B, R_A, K)$
16: $T_A \leftarrow AEAD_{auth}(S_A, T_B, R_A, K)$
17: **end if**
18: **else**
19: $abort()$
20: **end if**

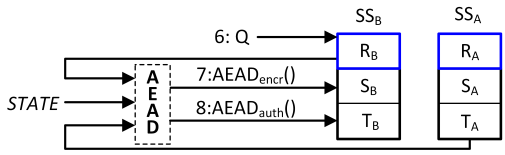


Fig. 6 An example REFRESH of state save packet, SS_B, based on Algorithm 2. (1) Update R_B with the latest nonce, Q (line 6), (2) encrypt the checkpoint with the nonce and authentication tag from previous packet, T_A, and update S_B (line 7), and (3) last, update the authentication tag, T_B, which invalidates SS_A and validates SS_B as the most recent valid packet (line 8).

REFRESH : Algorithm 2 defines both the secure checkpoint generation and restoration process, as they involve similar cryptographic operations with the difference listed on line 4 and 12. During power-on, REFRESH is called to generate the latest state save packet.

It determines which is the valid buffer, between SS_A and SS_B to update the alternate buffer. For example, when REFRESH is called, if SS_A is valid, line 2 in Algorithm 2 is true. Correspondingly, SS_B is updated with the latest checkpointed state by first updating R_B and then S_B, as illustrated in Figure 6. SS_A remains valid until T_B is updated. As soon as T_B is updated with the latest authentication tag, in line 8, SS_A is invalidated and SS_B is the latest valid packet. This update to the authentication tag, T_B in line 8 and T_A in line 16, makes REFRESH atomic. SICP makes an explicit assumption that this tag update is an atomic operation. This assumption is satisfied using the residual power source, explained further in Section 5.4.

RESTORE : RESTORE is called upon every power-up, except immediately after a *factory_reset()*, to decrypt and restore the most recent valid *STATE* of the device. The authentication tags of both the buffers are checked to identify the valid packet. If both authentication tag checks fail, *abort()* is called to indicate checkpoint tampering, which prevents restoring the device with a malicious state.

If the authentication tag check is passed on either line 2 or 10 in Algorithm 2, a valid state save packet exists which is decrypted and used to restore the device *STATE*. RESTORE documents each power-on event in the sequence of checkpoints by generating a new state save packet upon every power-up. For example, if SS_A is valid, S_A is decrypted and restored in *STATE*. SS_B is updated with this *STATE*, new nonce, Q and T_A. Now, SS_B is made valid, invalidating SS_A. SICP ensures that every power cycle is documented in the series of checkpoints.

WIPE Power loss is an adversarial event, based on our attacker model. WIPE must be called as soon as the device detects a power loss to clear sensitive information. It wipes all transient information, such as program variables, stored as plaintext using the residual power source in two steps. First, *STATE* is overwritten with zeros to clear persistent plaintext information. Second, volatile memory is also wiped to prevent cold boot style attacks [18]. The residual power source must have sufficient power to completely wipe transient information and maintain the confidentiality of checkpointed data.

5 Implementation

In this section, we describe our choice of target device, stateful power transition technique, and several design choices and device specific features used in implementing SICP.

5.1 Target device

The embedded device used with energy harvesters plays an important role in utilizing the harvested energy and is selected based on several criteria. First, on-chip non-volatile memory is required to store checkpoints. The use of off-chip non-volatile storage in the absence of on-chip non-volatile storage is not a secure solution, as the communication to off-chip memory and the memory itself is vulnerable to attackers as it can be easily monitored/removed. Second, the device must consume low power to judiciously use the available resources. The choice of device determines the overhead incurred by secure and stateful power transition.

We implement SICP on Texas Instruments'(TI) MSP430FR5994 Launch Pad Development Kit to demonstrate the feasibility of and to evaluate secure and stateful power transitions. We chose TI's MSP430FR5994 for several reasons. First, it is a low power device, only consuming 120μA/MHz of active current [38]. Second, it is equipped with 256kB of ferroelectric random access memory(FRAM), which is known for its ultra-low power consumption, high endurance, and fast read/write speeds. Third, it operates in a unified memory model, where SRAM, FRAM, and all the peripherals are mapped in a single global memory, which provides a common interface for all the data that must be secured and checkpointed. Fourth, it contains an on-chip AES accelerator, which can be used to speed-up the cryptographic primitives in SICP.

5.2 Three facets

5.2.1 Cause

In our proof-of-concept implementation, the microcontroller is powered by a constant DC power supply. We use a switch to power cycle the microcontroller at arbitrary time intervals to cause power loss.

5.2.2 Statefulness

During the ON-state, the microcontroller stores its general purpose registers, such as program counters (PC), in SRAM and application variables in FRAM. The application variables, found in `.data` and `.bss` sections, are placed in FRAM using the linker description file. After power loss, only FRAM data remains persistent, whereas the SRAM data is lost, leading to memory inconsistency between the volatile and non-volatile program state. We implement a modified version of TI's Computer Through Power Loss(CTPL) utility [41] to maintain a consistent checkpoint across all types of

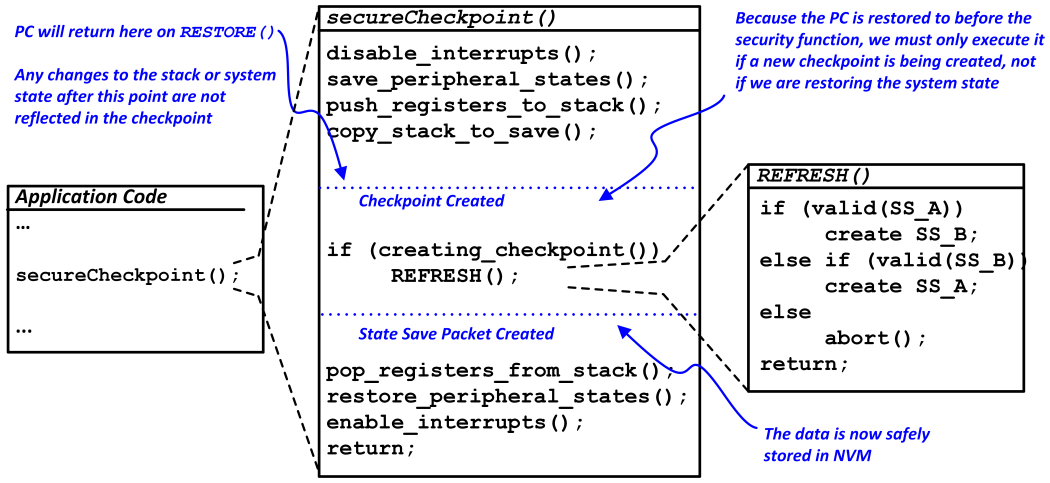


Fig. 7 The control and data flow for the creation of a checkpoint and subsequent state save packet. `REFRESH()` is only called when a checkpoint is created.

memory. The CTPL utility is designed for TI's `cl430` compiler. It was ported to compile on `msp430-elf-gcc` with changes to preprocessing references and to compile specific assembly code. It was further modified to support user declarable checkpoint functions, to invoke SICP functions within checkpoint calls and to incorporate SICP functions at system startup.

Checkpoint Location and Contents: The memory section containing *STATE* is separately declared as `.checkpoint` section in the device linker file, enabling easy identification of the data to be checkpointed and forcing its location within tamper sensitive memory. It provides a single known location for the `WIPE()` operation to target, discussed in detail later in this section. A guaranteed memory location also allows a straightforward check on the existence of a *factory_reset()* operation. It provides the application developer a simpler declaration interface, enabling the use of GCC's variable attributes, marked with the `__attribute__` keyword, instead of a complex variable registration interface and tracking data structure. We define `secureCheckpoint()` to generate a checkpoint in this dedicated location and create a state save packet using the SICP algorithms. Figure 7 illustrates the control flow involved in creating a checkpoint and subsequent state save packet. First, the volatile peripherals in use are saved on the stack, such as a timer and a comparator. Second, the general purpose registers are pushed on the stack. Since the first two steps mangle the stack and peripheral states, they must be restored to their original state after checkpoint generation. Third, the stack is saved in the `.checkpoint` section. Fourth, the non-volatile data which is to be secured is also stored in the checkpoint along with the volatile state. The checkpoint is ready to be secured by SICP. To create a state save packet, `REFRESH()` is called to wrap the segment up in a valid state save packet.

5.2.3 Security

The four algorithms of the protocol are defined as functions to create and restore the state save packet. We define `REFRESH()` to generate the latest state save packet, `RESTORE()` to restore the latest unmodified state save packet, `INITIALIZE()` to create the first state save packet, and `WIPE()` to wipe sensitive data using the residual power source. `INITIALIZE()` and `RESTORE()` are called automatically during system startup, as shown in Figure 8. `WIPE()` is also automatically triggered upon power loss.

5.3 System Integration

The modified checkpointing system is wrapped with the SICP function calls to enable secure and stateful power transitions. The device specific implementation of system start-up, cryptographic primitives, and `WIPE()` are as follows:

Startup: Figure 8 illustrates the startup sequence for a system employing SICP. A portion of the non-volatile memory region containing *STATE* is first checked for the factory reset bit pattern. This is used to determine if a *factory_reset()* has occurred and `INITIALIZE()` must be invoked, or a normal boot sequence with `RESTORE()` must occur. In either case, the appropriate SICP function is executed overwriting *STATE* with either zeros, for `INITIALIZE()`, or the authenticated and decrypted

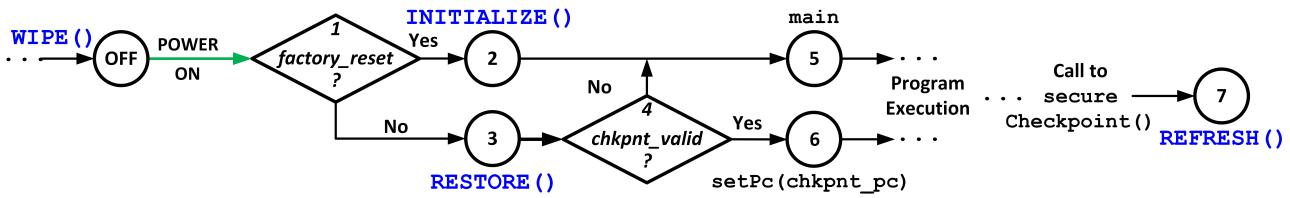


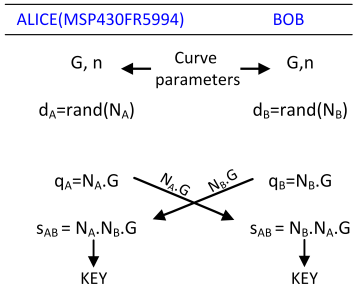
Fig. 8 Startup sequence for MSP430FR5994 with SICP. (1) Checks for *factory_reset()* and calls (2) `INITIALIZE()` or (3) `RESTORE()` to populate *STATE* in non-volatile memory. (4) the device inspects *STATE* for a valid checkpoint, restoring the checkpoint (6) if one is found or invoking `main()` (5) if one does not exist. Program execution will then continue normally until power is lost or another checkpoint is created.

system state, for `RESTORE()`. If the checkpointing system determines that no valid checkpoint exists, such as on the first boot after a *factory_reset()*, it will invoke `main()` as would be expected in a standard system startup.

nonce(): A majority of the nonces used in this protocol are provided via a 128-bit counter that is initialized to zero during `INITIALIZE()` and incremented each time a new nonce is requested. The exception is for the nonce for T_B used in `INITIALIZE()`, which is generated randomly. This nonce is generated randomly to ensure that no two different uses of a device create the same pattern of tags, even if the exact same code is executed following a *factory_reset()* [20, 34].

AEAD Integration: he development of the SICP API is agnostic of the underlying AEAD scheme used to enforce the protocol's security guarantees. We use a hybrid implementation of EAX [4], provided by the Cifra [6] cryptographic library. EAX is a well established two-pass AEAD scheme which avoids unnecessary decryption operations when a tag fails authentication in `REFRESH()` or `RESTORE()`. Tag failure occurs on half of the calls to these two functions since the state save packet authentication is used to determine which packet is valid and which is to be overwritten/restored. The block-cipher based nature of EAX enabled hardware acceleration by modifying the code to employ the MSP430FR5994's AES accelerator.

Tamper free memory: The secure memory is emulated using the Intellectual Property Encapsulation segment (IPE) available in MSP430FR5994 [12]. IPE is used to program a section of FRAM as secure memory, `.secure`, by setting the memory boundaries in the IPE registers. `.secure` section of the memory is programmed with read, write, and execute access. It is used to store the nonce used in SICP and the functions used to read and update the nonce. The variables stored in `.secure` section, and in turn the nonce, can only be read and updated by executing code stored in




```
ECDH PSEUDO CODE:
receive(q_B);
ec_curve_gen_(G);
ec_curve_get_ord(n);
bn_rand_mod(d_A,n);
secureCheckpoint();
ec_mul(q_A,G,d_A);
secureCheckpoint();
send(q_A);
ec_mul(s_AB,q_B,d_A);
secureCheckpoint();
kdf(k_AB, s_AB);
```

Fig. 9 ECDH key-exchange between Alice(MSP430FR5994) and Bob: a flow chart and pseudo-code.

`.secure` of the memory. The code in `.secure` section can be executed by branching into the IPE segment or by calling a function stored in IPE segment. A read access to `.secure` section from outside the IPE segment will at least return 0x3FFF.

5.4 Residual energy use

A residual energy source, which was an architectural requirement, is required to ensure atomicity and to wipe unprotected data after a power loss is detected.

Atomicity Support: The atomicity of `secureCheckpoint()` is ensured by using two state save packet buffers. All changes in non-volatile memory are made to the alternate buffer, such that the most recent packet remains unmodified. Once the new tag computation is complete and stored in a temporary buffer, the `sic_copyTag()` function is called to overwrite the previous tag and set the newly created checkpoint as the only valid checkpoint in an atomic operation. This copy function is made atomic by disabling all interrupts for the copy duration of 48 cycles and relying on the residual energy of the device and the FRAM's atomic byte write capability to ensure that even if power is lost, the copy operation will complete before the system stops operating. `secureCheckpoint()` either has no effect on the

Table 1 Breakdown of the contents of the checkpoints of our ECDH implementation

Variable	Size (B)	Comments
System data	763	Device specific data
Generator(G)	96	Memory required to
Shared secret(s_{AB})	96	store an elliptic
Bob's public key (q_B)	96	curve point as a
Alice's public key (q_A)	96	Jacobian coordinate
Alice' private key (d_A)	32	256-bit integer
Order of generator (n)	32	
Total	1211	

system, if power is lost before the tag update, or completes the checkpoint creation without incident.

`WIPE()`: The implementation of the *WIPE* operation requires detection of power loss by monitoring the device's V_{cc}. MSP430FR5994's ADC12_B analog-to-digital converter is used to measure V_{cc} against the system's V_{ref} as described in TI's FRAM Utilities [41]. The MSP430FR5994 development board's unmodified implementation, including one 10μF capacitor and three 100nF capacitors, has sufficient residual energy to consistently overwrite up to 16kB of memory using direct-memory-access (DMA) following the trigger for power loss. When V_{cc} falls below V_{ref} ADC12_B triggers overwrite of *STATE* and SRAM via DMA using the residual energy.

6 EVALUATION OF SICP

In this section we evaluate SICP based on a real-life application. We introduce our target application, our implementation of the application, its checkpoints and the need for checkpoint security, and the effect of each facet on the application.

6.1 Target application

We chose Elliptic Curve Diffie Hellman key exchange (ECDH) as a representative of an application which needs secure checkpoints. ECDH can be used to exchange keys between two entities, Alice and Bob, via an unsecured channel to secure the communication channel in several steps, illustrated in Figure 9. First, they agree upon an elliptic curve, E; a base point, G, in E whose order is n. The order n is the smallest integer such that $n.G = 0$. Second, they each chose an integer, N_i, less than n as their corresponding private key, d_i. Third, they compute the product $N_i.G$ as their public key, q_i. Last, each entity generates the shared secret by multiplying its secret with the other entity's public key, which is then used to derive the secret key, *KEY*. We implement ECDH on MSP430FR5994 using the NIST curve P-256, which provides 128-bits of security, with the help of the RELIC cryptographic library [3]. We consider Alice to be the target microcontroller. ECDH involves long running arithmetic operations on the elliptic curve, such as generating an integer and point multiplications. When the microcontroller is powered by an energy harvester, ECDH may operate at a reduced duty cycle to finish its computations, as explained in Section 2.1. We place `secureCheckpoint()` calls after long running arithmetic operations to ensure the availability of the intermediate results in the event of power loss, as illustrated in Figure 9.

Checkpoint Location and Contents:

The application specific variables that are required to ensure forward progress of ECDH are listed in Table 1. All the variables listed in Table 1 are placed in the `.checkpoint` section of tamper-sensitive non-volatile memory using the `__attribute__` keyword, as described in Section 5.2. The generator, G, of the curve is a point on the 256-bit elliptic curve, consisting of three Jacobian coordinates, X, Y, and Z, where each coordinate is 256-bit long. Thus, each point on the elliptic curve, such as the shared secret, s_{AB}, and public keys, q_A and q_B are 96 B. The shared secret, s_{AB}, and Alice's private, d_A, are also checkpointed to maintain secure sessions across power loses. Since d_A and n are 256-bit integers, they only occupy 32 B each. Of the 1211 B of checkpointed data, only G, n, and the public keys, q_A, and q_B, are global public elements, the rest of the variables must be protected from the attacker to maintain the security of the communication channel. Thus, checkpoints of ECDH require SICP to maintain the security properties of its application across stateful power transitions.

6.2 Effect of facets on duty cycle

We study the overhead introduced by each facet and in turn study its effects on the duty cycle of ECDH. The device under test was operated at 1 MHz and was powered by an external power supply. The energy and time measurements reported in this section were measured across a 1 $k\Omega$ shunt resistor using a Tektronix DPO3034 oscilloscope operating at 50 kS/s. Table 2 lists the energy and time overhead of ECDH, CTPL and SICP. It computes the net load power after introducing each facet based on the assumption that the initialize operations occur only once and are ignored

Table 2 Energy and time overhead of different technology and corresponding duty cycle, D_{load}, when input power is 2 mW

Technology	Operation	Time (ms)	Energy (μJ)	P_{load} (mW)	D_{load} (%)
Application	ECDH	7800	48300	6.2	33
CTPL	Initialize	0.02	0.03	8.0	25
	Refresh	13.8	12.1		
	Restore	13.5	12.3		
SICP	`INITIALIZE()`	0.06	0.04	9.5	21
	`REFRESH()`	216.2	160.2		
	`RESTORE()`	277.1	202.3		

and only one checkpoint generation operation is performed during each ON-state. We analyze the effects of this overhead on the duty cycle of the device, using equation 1.

Energy harvesters: In the continuous execution paradigm, the microcontroller consumes 48.3 mJ of energy in 7.8 s to arrive at the shared secret of an ECDH operation, which requires 6.2 mW of power. Energy harvesters do not always provide the peak power required by the microcontroller, they typically provide a few μW to mW of power [25]. We assume that the microcontroller is powered by a kinetic energy harvester [9], which provides an average power of 2 mW. In the intermittent execution paradigm, the microcontroller still requires 6.2 mW of power to arrive at the shared secret but it operates at only 32% duty cycle as P_{load} is greater than P_{EH}. The microcontroller repeatedly experiences power loss for every 2 mW of power it consumes.

Statefulness: CTPL stores the volatile state information in non-volatile memory as a checkpoint and retrieves it to restore the microcontroller after a power cycle, which introduces overhead. Table 1 lists the checkpoint size of our target application as 1211 B, which is calculated by studying the memory section containing *STATE* [36]. The checkpoint generation and restoration operations combined introduce an overhead of 1.8 mW, listed in Table 2. In addition to the power requirements of ECDH operations, the checkpointing overhead increases the net load power, P_{load}, to 8 mW. The microcontroller duty cycle is reduced to 25% to arrive at the shared secret, s_{AB}, assuming that the device is still powered by the same kinetic energy harvester.

Security: Table 2 lists the additional overhead SICP introduces to stateful power transitions. It presents the amount of energy and time required to secure the generation and restoration of 1211 B of checkpointed data. The overhead measurements correspond to `INITIALIZE`, `REFRESH`, and `RESTORE` operations of the protocol. The microcontroller requires an additional 1.4 mW to secure 1211 B of checkpoint. This increases the net load power, P_{load}, to 9.5 mW. The microcontroller must operate at 21% duty cycle to finish its ECDH operations while ensuring that its security properties and that of ECDH are maintained across power transitions.

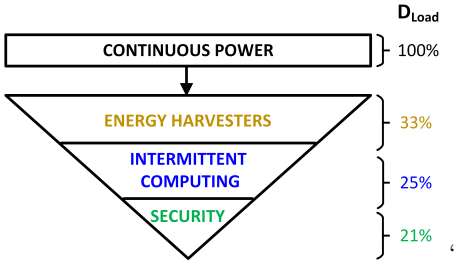


Fig. 10 Facets of power transition and corresponding effect on duty cycle, D_{load}

6.3 Analysis

In our implementation of secure and stateful power transition, we observe that each facet, including the kinetic energy harvester, CTPL, and SICP, introduces a limiting factor which progressively reduces the duty cycle of the device, illustrated in Figure 10. We studied the effects of energy influx, type of non-volatile memory, and checkpoint size on the duty cycle of the load. Figure 11 illustrates the change in duty cycle based on the energy influx for different checkpoint sizes and non-volatile memories. The data points on each line graph correspond to the duty cycle of the device based on the energy influx from three types of sources, including kinetic, vibration, and thermal harvesters, which provide 2 mW, 4.5mW, and 5.2 mW, respectively [9, 26]. The energy influx varies, between a few μW to a few mW, depending on the choice of the harvester [25]. We chose two types of non-volatile memory commonly available in off-the-shelf devices. First, we studied the flash memory available in TI's MSP432P401R [37]. Second, we studied FRAM available in MSP430FR5994 [38]. Since

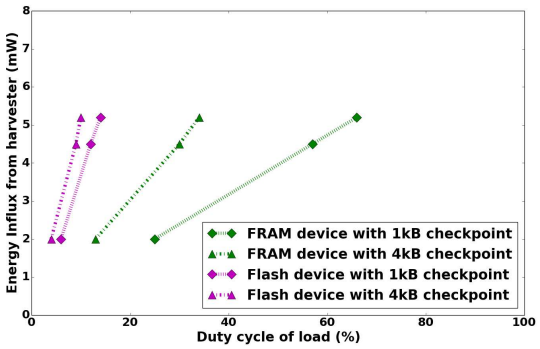


Fig. 11 Changes to the duty cycle of load, D_{load}, based on the energy influx [9, 26], size of checkpoint, and type of non-volatile memory. The duty cycle measurements were calculated based on the ratings of flash memory and FRAM available in the device datasheet [37, 38]

FRAM consumes low power when compared to flash, the duty cycle of FRAM devices is higher than that of flash devices. We also considered two checkpoint sizes, 1kB and 4kB, to account for applications whose checkpoints maybe larger than ECDH's 1.2kB checkpoint. Figure 11 illustrates that larger checkpoints reduce the duty cycle of the load, irrespective of the energy influx and type of non-volatile memory in use. The duty cycle reported in Figure 11 will reduce further when secure checkpoints are employed. Thus, we must consider the various technologies involved in an energy harvested node, including, but not limited to the energy influx, type of non-volatile memory, application, frequency of checkpoints, and type of device, to achieve the required duty cycle of the target device.

7 Conclusion

We presented the Secure Intermittent Computing Protocol to bridge the gap between stateful power transitions and secure embedded systems. It is the first secure intermittent solution to provide comprehensive security to the power transitions of an embedded system. It is a fail-safe and generic protocol that can be used with existing stateful power transition solutions to enhance their security. We provide a proof-of-concept implementation of secure and stateful power transitions on an MSP430FR5994 to demonstrate the feasibility of secure checkpoints. The introduction of each facet and its corresponding technology affects the duty cycle of the target application. Our evaluation demonstrates the need for careful design choices, including but not limited to non-volatile memory, low power device, cryptographic hardware, and secure memory, to improve the duty cycle of the application. Several low power microcontrollers are equipped with cryptographic hardware, mostly for encryption. In the future, we must consider including hardware accelerated authenticated encryption engine, low power non-volatile memory, and secure storage capabilities to microcontrollers to improve the duty cycle of the application to facilitate secure and stateful power transitions.

Acknowledgements This work was supported in part by NSF grant 1704176 and SRC GRC Task 2712.019.

References

1.
2. Zatara High-Performance, Secure, 32-Bit ARM Microcontroller. Tech. rep., Maxim (2009)
3. Aranha, D.F., Gouva, C.P.L.: RELIC is an Efficient LIbrary for Cryptography (2010)
4. Bellare, M., Rogaway, P., Wagner, D.: The EAX Mode of Operation, pp. 389–407. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). DOI 10.1007/978-3-540-25937-4_25
5. Berthou, G., Delizy, T., Marquet, K., Risset, T., Salagnac, G.: Sytare: A lightweight kernel for NVRAM-based transiently-powered systems. IEEE Trans. Computers **68**(9), 1390–1403 (2019)
6. Birr-Pixton, J.: Cifra: Cryptographic Primitive Collection. https://github.com/ctz/cifra (2017)
7. Biswas, S., Neogy, S.: Secure checkpointing using public key cryptography in mobile computing. In: 2011 Fifth IEEE International Conference on Advanced Telecommunication Systems and Networks (ANTS), pp. 1–3 (2011). DOI 10.1109/ANTS.2011.6163669
8. Bronevetsky, G., Marques, D., Pingali, K., McKee, S., Rugina, R.: Compiler-enhanced incremental checkpointing for openmp applications. In: 2009 IEEE International Symposium on Parallel Distributed Processing, pp. 1–12 (2009). DOI 10.1109/IPDPS.2009.5160999
9. Da, Y., Khaligh, A.: Hybrid offshore wind and tidal turbine energy harvesting system with independently controlled rectifiers. In: 2009 35th Annual Conference of IEEE Industrial Electronics, pp. 4577–4582 (2009). DOI 10.1109/IECON.2009.5414866
10. Daemen, J., Rijmen, V.: Rijndael for AES. In: AES Candidate Conference, pp. 343–348 (2000)
11. Davi, L., Hanreich, M., Paul, D., Sadeghi, A., Koeberl, P., Sullivan, D., Arias, O., Jin, Y.: HAFIX: hardware-assisted flow integrity extension. In: Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015, pp. 74:1–74:6 (2015)
12. Dinu, D., Krishnan, A.S., Schaumont, P.: SIA: secure intermittent architecture for off-the-shelf resource-constrained microcontrollers. In: IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019, McLean, VA, USA, May 5-10, 2019, pp. 208–217 (2019)
13. Eldefrawy, K., Francillon, A., Perito, D., Tsudik, G.: 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 (2012)

14. Gaglione, A., Rodenas-Herraiz, D., Jia, Y., Mascolo Sarfraz Nawaz, E.A.C., Soga, K., Seshia, A.A.: Energy Neutral Operation of Vibration Energy-harvesting Sensor Networks for Bridge Applications. In: Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks, EWSN 8217;18, pp. 1–12. Junction Publishing, USA (2018). URL http://dl.acm.org/citation.cfm?id=3234847.3234849
15. Ghodsi, Z., Garg, S., Karri, R.: Optimal checkpointing for secure intermittently-powered IoT devices. pp. 376–383 (2017). DOI 10.1109/ICCAD.2017.8203802
16. Gofman, M.I., Luo, R., Yang, P., Gopalan, K.: SPARC: A Security and Privacy Aware Virtual Machinecheckpointing Mechanism. In: Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society, WPES '11, pp. 115–124 (2011)
17. Guan, L., Lin, J., Ma, Z., Luo, B., Xia, L., Jing, J.: Copker: A Cryptographic Engine Against Cold-Boot Attacks. IEEE Trans. Dependable Sec. Comput. **15**(5), 742–754 (2018)
18. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: cold-boot attacks on encryption keys. Commun. ACM **52**(5), 91–98 (2009)
19. Hicks, M.: Clank: Architectural Support for Intermittent Computation. In: Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017, pp. 228–240 (2017)
20. Holcomb, D.E., Burleson, W.P., Fu, K.: Power-up SRAM state as an identifying fingerprint and source of true random numbers. IEEE Trans. Computers **58**(9), 1198–1210 (2009)
21. Jayakumar, H., Raha, A., Lee, W.S., Raghunathan, V.: Quickrecall: A HW/SW approach for computing across power cycles in transiently powered computers. JETC **12**(1), 8:1–8:19 (2015)
22. Kannan, S., Karimi, N., Sinanoglu, O., Karri, R.: Security Vulnerabilities of Emerging Nonvolatile Main Memories and Countermeasures **34**(1), 2–15 (2015). DOI 10.1109/TCAD.2014.2369741
23. Khanna, S., Bartling, S., Clinton, M., Summerfelt, S.R., Rodriguez, J.A., McAdams, H.P.: An FRAM-Based Nonvolatile Logic MCU SoC Exhibiting 100% Digital State Retention at $VDD = 0$ V Achieving Zero Leakage With $<$ 400-ns Wakeup Time for ULP Applications. J. Solid-State Circuits **49**(1), 95–106 (2014)
24. Krishnan, A.S., Suslowicz, C., Dinu, D., Schaumont, P.: Secure intermittent computing protocol: Protecting state across power loss. In: Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019, pp. 734–739 (2019)
25. Ku, M., Li, W., Chen, Y., Ray Liu, K.J.: Advances in Energy Harvesting Communications: Past, Present, and Future Challenges. IEEE Communications Surveys Tutorials **18**(2), 1384–1412 (2016). DOI 10.1109/COMST.2015.2497324
26. Li, J., HoonHyun, J., SamHa, D.: A multi-source energy harvesting system to power microcontrollers for cryptography. In: IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, Washington, DC, USA, October 21-23, 2018, pp. 901–906 (2018)
27. Lu, X., Wang, P., Niyato, D., Kim, D.I., Han, Z.: Wireless Networks With RF Energy Harvesting: A Contemporary Survey. IEEE Communications Surveys Tutorials **17**(2), 757–789 (2015). DOI 10.1109/COMST.2014.2368999
28. Lucia, B., Ransford, B.: A simpler, safer programming and execution model for intermittent systems. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015, pp. 575–585 (2015)
29. Maeng, K., Colin, A., Lucia, B.: Alpaca: Intermittent Execution Without Checkpoints. Proc. ACM Program. Lang. **1**(OOPSLA), 96:1–96:30 (2017)
30. Nam, H., Kim, J., Hong, S.J., Lee, S.: A secure checkpointing system. In: 8th Pacific Rim International Symposium on Dependable Computing (PRDC 2001), 17-19 December 2001, Seoul, Korea, pp. 49–56 (2001)
31. Noorman, J., Bulck, J.V., Mühlberg, J.T., Piessens, F., Maene, P., Preneel, B., Verbauwhede, I., Götzfried, J., Müller, T., Freiling, F.: Sancus 2.0: A Low-Cost Security Architecture for IoT Devices. ACM Trans. Priv. Secur. **20**(3), 7:1–7:33 (2017)
32. Pabbuleti, K., Mane, D., Schaumont, P.: Energy Budget Analysis for Signature Protocols on a Self-powered Wireless Sensor Node. In: N. Saxena, A.R. Sadeghi (eds.) Radio Frequency Identification: Security and Privacy Issues, pp. 123–136. Springer International Publishing, Cham (2014)
33. Piegdon, D.R.: Hacking in physically addressable memory. In: Seminar of Advanced Exploitation Techniques, WS 2006/2007 (2006)
34. Rahmati, A., Salajegheh, M., Holcomb, D.E., Sorber, J., Burleson, W.P., Fu, K.: TARDIS: Time and Remanence Decay in SRAM to Implement Secure Protocols on Embedded Devices without Clocks. In: Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012, pp. 221–236 (2012)
35. Rogaway, P.: Authenticated-encryption with associated-data. pp. 98–107 (2002). DOI 10.1145/586110.586125
36. Santhana Krishnan, A., Schaumont, P.: Exploiting security vulnerabilities in intermittent computing: 8th international conference, space 2018, kanpur, india, december 15-19, 2018, proceedings. pp. 104–124 (2018). DOI 10.1007/978-3-030-05072-6_7
37. MSP432P401R,MSP432P401 MSimpleLink Mixed-SignalMicrocontrollers. Tech. rep., Texas Instruments (2015)
38. MSP430FR599x, MSP430FR596x Mixed-Signal Microcontrollers. Tech. rep., Texas Instruments (2016). Revised August 2018. Available at http://www.ti.com/lit/ds/slase54c/slase54c.pdf
39. Stark, I.: Integrating Thermoelectric Technology into Clothing for Generating Usable Energy to Power Wireless Devices. In: Proceedings of the Conference on Wireless Health, WH '12, pp. 17:1–17:2 (2012)
40. Swami, S., Mohanram, K.: ACME: Advanced Counter Mode Encryption for Secure Non-volatile Memories. In: Proceedings of the 55th Annual Design Automation Conference, DAC '18, pp. 86:1–86:6 (2018)
41. Texas Instruments: MSP MCU FRAM Utilities (2017)
42. Valea, E., Silva, M.D., Natale, G.D., Flottes, M., Dupuis, S., Rouzeyre, B.: SECCS: secure context saving for iot devices. CoRR **abs/1903.04314** (2019)
43. Van Der Woude, J., Hicks, M.: Intermittent Computation Without Hardware Support or Programmer Intervention. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI'16, pp. 17–32 (2016)