Secure Application Continuity in Intermittent Systems

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

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

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

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

Abstract*—Intermittent systems operate embedded devices without a source of constant reliable power, relying instead on an unreliable source such as an energy harvester. They overcome the limitation of intermittent power by retaining and restoring system state as checkpoints across periods of power loss. Previous works have addressed a multitude of problems created by the intermittent paradigm, but do not consider securing intermittent systems. In this paper, we address the security concerns created through the introduction of checkpoints to an embedded device. When the non-volatile memory that holds checkpoints can be tampered, the checkpoints can be replayed or duplicated. We propose *secure application continuity* as a defense against these attacks. Secure application continuity provides assurance that an application continues where it left off upon power loss. In our secure continuity solution, we define a protocol that adds integrity, authenticity, and freshness to checkpoints. We develop two solutions for our secure checkpointing design. The first solution uses a hardware accelerated implementation of AES, while the second one is based on a software implementation of a lightweight cryptographic algorithm, Chaskey. We analyze the feasibility and overhead of these designs in terms of energy consumption, execution time, and code size across several application configurations. Then, we compare this overhead to a non-secure checkpointing system. We conclude that securing application continuity does not come cheap and that it increases the overhead of checkpoint restoration from 3.79 μJ to 42.96 μJ with the hardware accelerated solution and 57.02 μJ with the software based solution. To our knowledge, no one has yet considered the cost to provide security guarantees for intermittent operations. Our work provides future developers with an empirical evaluation of this cost, and with a problem statement for future research in this area.

***Keywords*-secure embedded systems; intermittent systems; application continuity; energy harvesting**

I. Introduction

Energy harvesters convert ambient energy into green energy which can power up small devices. Improvements in energy harvesting have created the potential for energy harvested embedded systems that are not limited to a strict battery life or grid connection [1]. Instead, new systems are now equipped with an energy harvester, a small energy store, and intermittent capability to allow continuous operation without the need for frequent system maintenance. This creates opportunities to deploy embedded systems to locations that are difficult to support or manage, especially for sensor applications and control applications in remote locations.

While offering sustainable energy and autonomy to embedded computing systems, energy harvesters also bring a new computing paradigm to these systems. Embedded devices now operate within an intermittent computing model where they continue their operations across periods of power loss through the retention of their system state as checkpoints in non-volatile memory (NVM). This is possible through the advent of low energy NVM technologies such as ferroelectric RAM (FRAM) and phase-change memory [2], [3].

A. Security Dimension

One aspect of intermittent systems has been ignored by the current body of work, the security implications of intermittent operation. The act of storing the complete system state in NVM exposes critical pieces of system information to potential tampering by an adversary with access to the device.

Existing intermittent computing solutions operate on the assumption that if a checkpoint exists, it is valid and will properly restore the system [4], [5], [6]. This is a naive and potentially dangerous assumption. Once in non-volatile memory, the checkpoint is at risk of modification, copy, replacement, or corruption by both malignant and erroneous actions. A small number of NVM protection techniques have been proposed which incorporate encryption into every memory access [7], [8]. Unfortunately, these techniques are not energy efficient for low-power devices. Ghosdi *et al.* suggest securing intermittent system checkpoints through the encryption of each checkpoint, but their solution only provides confidentiality and fails to account for checkpoint integrity, authenticity or freshness [9].

If a checkpoint is tampered, the system is exposed to a host of problems. Secret information on the device, such as private keys, could be exposed. Replay of old checkpoints could trigger sensitive operations that enable side-channel attacks on normally secure behaviors. Traditional security features, control-flow integrity monitoring or memory protection schemes, can be bypassed through the modification of their data structures during the power-off period [10], [11]. Some schemes, such as SANCUS [12], which include custom hardware support for their security guarantees are also susceptible if the checkpointing process exposes the values stored in the protection mechanism's custom registers or shadow stack within a checkpoint [13]. The storage of this information would be necessary to restore the correct system state, but it exposes otherwise protected information to tampering by an adversary and creates a discontinuity in the security guarantees provided by these protection mechanisms.

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

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

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

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

B. Contributions

We present the following contributions through our work:

- We highlight the need for protection against checkpoint replay in intermittent systems and propose securing application continuity with an anti-replay solution.
- We introduce a protocol that provides integrity, authenticity, and freshness to an intermittent system's checkpoints and continuity to existing embedded security applications across periods of power loss.
- We provide a proof-of-concept implementation of the protocol with both a software implementation utilizing lightweight cryptography and a hardware solution using a hardware accelerated standard cryptographic primitive on an MSP430 device. We then empirically evaluate the cost for securing application continuity across different device configurations.

The remaining portions of this paper are structured as follows: Section II describes the details of our protocol to provide secure application continuity. It outlines our implementation of a basic intermittent system with integrity, authenticity, freshness, and application continuity. Section III details the the procedure to integrate security to the intermittent system on our test platform. Section IV discusses our experimentation strategy, evaluation, and results. Finally, we close with our conclusions.

Algorithm 1 `refresh` and `restore`

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

II. Approach

Our approach is to target the checkpoint system itself, protect it, and thereby provide existing embedded solutions the continuity necessary to securely operate on an intermittent device. Our protocol was designed based on the assumption that the embedded system employs existing embedded solutions to protects its execution integrity and memory when it is powered on. This will enable the use of previously developed embedded systems by ensuring their secure application continuity across the periods of power loss and checkpointing behavior unique to intermittent systems.

The design (Section II-A) and implementation (Section II-B) of a secure protocol for application continuity in intermittent systems powered by harvested energy is challenging for two reasons. First, these systems have to judiciously use every fraction of the available energy. Second, adding security to a checkpointing scheme requires a careful manipulation of the system's state in order to avoid inconsistencies or security breaches as result of an uncontrollable power loss. In this section we address these challenges, propose a secure protocol for application continuity in intermittent systems and a proof-of-concept implementation. Furthermore, our experimental results (Section IV) provide a unique perspective on the cost of secure application continuity for devices powered by harvested energy.

A. Secure Protocol

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

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

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

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

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

When the system is powered on, the most recent checkpoint, as determined by the tag chain, is used to restore the system if it passes the authentication tag check. This execution of the `restore` function occurs before any other operation, an approach similar to the solution proposed for Quickrecall [5].

A part of the checkpoint, if not the whole, must be made inaccessible to the adversary to prevent replay attacks. Apart from the regular tamper-sensitive NVM, Algorithm 1 requires a tamper-free NVM. It is a section of memory that cannot be tampered with or accessed by the adversary even when the device is powered off. Since tamper-free memory is small in size, as demonstrated in the Zatara ZA9L1 [14], and checkpoint size varies depending on the application, it is not feasible to store an entire checkpoint in tamper-free memory. Instead, we use it to store the smallest elements of our secure checkpoints, the nonces and the device unique key. Since the nonce is unavailable to the adversary, they cannot obtain a copy of the entire checkpoint for the purpose of a replay attack.

Tamper-free memory is not currently a standard component in most microcontroller platforms, but recent work in attestation and isolation for microcontrollers demonstrates the feasibility of tamper-free NVM in future devices and Texas Instruments provides limited memory protection capabilities in existing platforms [15], [16]. The vast majority of our device's NVM does not possess tamper resistance and is used to store the rest of the secure checkpoint including the checkpoint states and the authentication tags.

B. Implementation

We extended and modified an existing intermittent solution, the *Compute-Thru-Power-Loss* (CTPL) library from Texas Instruments [3], to validate our protocol using a software solution. First, we modified the library to be compatible with the `msp430-elf-gcc` compiler to work with our other existing MSP430 code base and tools. Since the original CTPL code only supported transitions to low power modes and the creation of system checkpoints before shutdown, we extended the CTPL feature set to include on-demand checkpointing, a user identified secure data section, alternating checkpoint storage, and integration of the security primitives required by our protocol within the checkpoint creation and restoration process.

Finally, we selected cryptographic algorithms based on the available hardware modules and previously demonstrated performance of existing lightweight MACs.

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

1) Data Identification and Storage: Defining a separate memory section, the `.secure` section, within the device's linker description file, provides a clear user defined secure data section. The `.secure` section is located in a portion of the device's NVM and provides developers with the ability to clearly declare variables that should be included within the device's checkpoints. It serves as a live snapshot of the device's current state and includes the allocated NVM to store the device's stack, register information, and peripheral information during the checkpoint creation process. Using a specified memory section simplifies the creation of checkpoints by collating all of the necessary system data in one contiguous memory region. The collection of the processor state and peripheral information is completed through the normal CTPL processes. To create a checkpoint, the device's peripherals are polled and their appropriate register states are stored in their allocated portion of the `.secure` memory section. The register information is then pushed onto the system stack and a copy of the stack is stored in the `.secure` section. Once all of the relevant data has been collected, the secure checkpoint creation function, `refresh`, is invoked to create the checkpoint.

2) Secure Checkpointing Functions: The `refresh` function implements the protocol we developed to store the new checkpoint in the appropriate checkpoint storage slot. As described in Section II, two secure checkpoint storage slots, A and B, are defined in the implementation. The latest checkpoint is identified through the verification of each stored checkpoint's MAC as described by Algorithm 1 and the new checkpoint is stored in the alternate slot. The write of the newly computed MAC serves as the transition to the new checkpoint as the valid stored state. Any power loss prior to the completion of this write will have the system restore the previous checkpoint; once the write is complete, the new checkpoint will pass validity checks and the old checkpoint will fail.

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

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

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

`HW-SIC`, hardware supported secure intermittent computing, uses a hardware AES accelerator to compute checkpoint MACs using CMAC. CMAC is a block cipher mode of operation recommended by NIST to determine the integrity of data against malicious modification [17]. Our implementation was developed using the Cifra cryptographic library [18] and required very few modifications to utilize the AES accelerator for its block cipher operations instead of the Cifra AES implementation. This structure provides an excellent baseline for a hardware accelerated standard cryptographic primitive supporting 64-bit collision resistance for MAC operations and is representative of a solution for systems that either contain cryptographic co-processors or need to employ a NIST standard cryptographic primitive for compliance.

`SW-SIC`, software supported secure intermittent computing, implements a software based MAC using Chaskey [19] to measure the overhead that would be experienced by devices that lack an AES accelerator. Chaskey is an efficient lightweight MAC algorithm for use with 128-bit keys. It specifically targets platforms that are not robust enough to employ a standard hash-based authentication code (HMAC) but still require effective security. We chose Chaskey as a representative MAC for this category for two reasons: its MSP430 performance in the FELICS test suite [20], where it outperformed many other similar lightweight ciphers [21], and its current consideration for standardization by ISO [22]. Combined, these qualities make it a reasonable representative of lightweight cryptographic primitives that may be considered for use in energy harvested intermittent systems.

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

III. Measurement Platform and Test Structure

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

A. Platform

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

The FRAM present on the MSP430FR5994 is an excellent example of the new non-volatile memory technologies

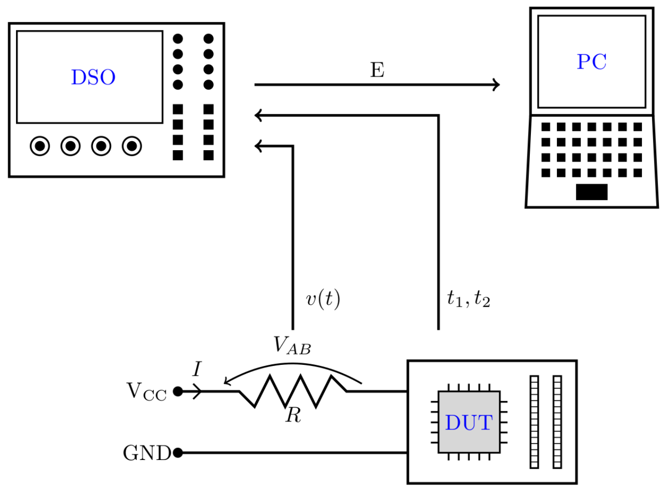

Fig. 1: The measurement circuit constructed to observe the energy required for different device operations. The device under test (DUT) was powered by an external power supply to execute a continuous loop of operations including oscilloscope triggers via GPIO.

enabling intermittent computing systems. It is efficient, with a delay of 1 μs and consuming only 225 μA of current for each FRAM read or write at 1 MHz operation [23]. Additionally, the manufacturer guarantees byte-level write atomicity regardless of the current power condition for the chip. This allows our system to assume that if a write to FRAM is executed, the data is written correctly up to a single byte and does not require additional error checking to detect partial writes at the byte level [24]. The FRAM is capable of normal operation up to 8 MHz, beyond which it is supported by processor wait-states, which utilizes a 2-way associative 256-bit cache for frequencies up to 16 MHz. The effect of processor wait states and cache on the device's energy consumption at 16 MHz is observable in our evaluation and analyzed in Section IV-B.

B. Testbed

Measurements were taken with a Tektronix DPO3034 oscilloscope operating at 50 kS/s across a shunt resistor of 1 $k\Omega$ as depicted in Figure 1. We computed the energy consumed by the target device during a function's execution using Equation 1.

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

The three critical functions for secure intermittent operations, `HW-SIC`, `SW-SIC`, and `NON-SIC`, were measured for both energy consumption and execution time. Each execution was identified within the testbench via a GPIO trigger on the test board. Additional energy measurements were taken with a second active GPIO trigger and subtracted from the single trigger measurement to calculate the energy consumed by a single GPIO trigger. This overhead was again subtracted from the single trigger measurement to obtain accurate energy measurements for every function.

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

Because `refresh` and `restore` execute an additional MAC computation if the first test is invalid, we forcibly tested the functions an equal number of times for each condition. These values were averaged to produce the data in Figure 2 and Figure 3. To verify the stability of our test bench, we computed the standard error for our energy measurements. Since these values were less than 10^{-8} J, we omitted them for clarity.

All the measurements were taken on a device powered by an external power supply. We expect the same behaviors to persist when the device is powered by an energy harvester.

IV. Evaluation and Results

With an experimental setup established, we tested our implementation and gathered measurements across a range of feasible system configurations. We recorded the effect of different system configurations on the overhead incurred by the software and hardware secure intermittent computing solutions in order to identify patterns that may be useful when choosing operating conditions based on energy and cycle count measurements.

All the performance figures reported are for the three main functions required to provide security to an intermittent system, namely `initialize`, `refresh`, and `restore`. The number and placement of checkpoints in a particular application are not considered since they are determined by application-specific constraints and optimization goals. However, the energy consumption and execution time scale linearly with the number of checkpoint operations executed.

A. Experimental Results

From our experiments, we were able to successfully observe the energy consumption of the intermittent system across all scenarios. The energy required for securing application continuity highlights an interesting characteristic of our chosen platform, a reduced energy consumption at the middle clock frequencies in Figure 2, that we explore in more detail in our analysis. The results from changes to the size of the system state, depicted in Figure 3, were as expected when the energy consumption increased linearly with increasing state size. Finally, the code size also followed our expectations with the software supported operations requiring more space than the hardware supported primitive.

B. Analysis

The energy required to secure a 512 B state is 42.96 μJ (`HW-SIC`) or 57.02 μJ (`SW-SIC`), an increase by a factor of 11.33 or 15.04 over the `NON-SIC` solution at 8 MHz.

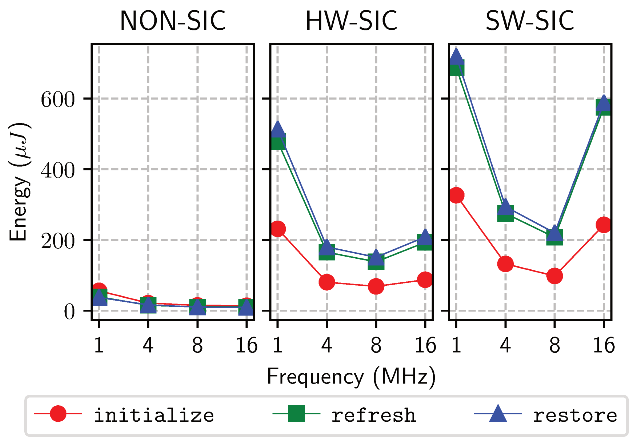

Fig. 2: Energy consumption as a function of the operational frequency shows the effect of our device's larger static power consumption, leading to a non-linear relationship between 1, 4, and 8 *MHz* operation. The dramatic spike in energy costs for 16 *MHz* is tied to the increased minimum supply voltage required for the testbed to operate and the introduction of FRAM wait-states at frequencies above 8 *MHz*.

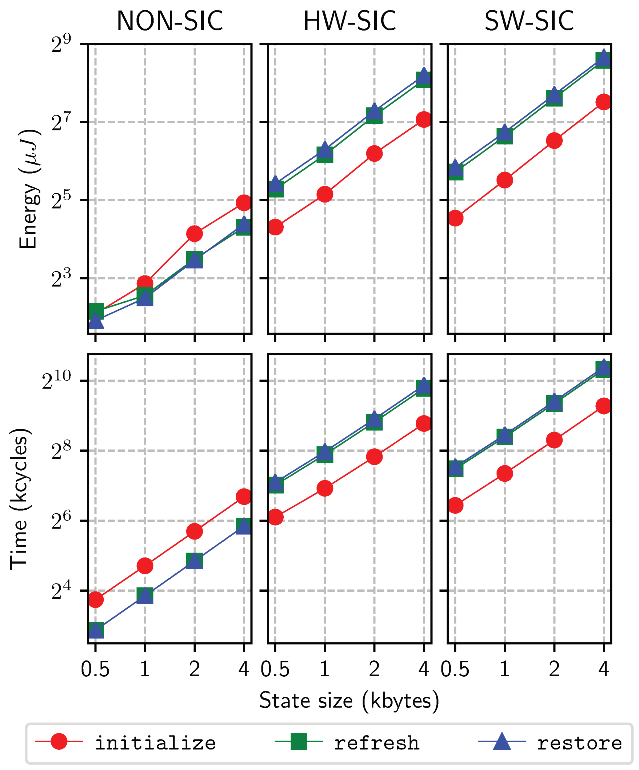

Fig. 3: The effect of system state size on the execution time and energy consumption of the three security functions operating at 8 *MHz* is reasonable. `SW-SIC` requires slightly more resources than `HW-SIC` and `initialize` is the cheapest and simplest of the three operations.

We recognize the high cost of securing a system's intermittent operation through checkpoint integrity. Algorithm 1 was designed with security against replay attacks as its primary and performance as its secondary objective, hence we focused on evaluating a functionally secure protocol rather than an optimized implementation. Further reducing this overhead through more novel applications of established cryptographic primitives or re-engineering of the overall system to better support the high computational requirements for the verification operations is a worthwhile future endeavor to obtain efficient secure application continuity.

An interesting behavior we observed in Figure 2 was the dramatic reduction in energy required for the secure intermittent computing solutions at 4 and 8 *MHz*. Typically, when a device's static power consumption is significantly less than its dynamic power consumption, the overall energy consumed per function at different frequencies is determined by the dynamic power consumption. Hence, the overall energy consumed at different frequencies grows as the frequency increases, a relationship depicted in Equation 2. Instead, we determined that the device under test had a significantly higher static power consumption than dynamic power consumption, allowing higher frequency operations to save more energy. For example, according to Equation 2, we found α to be 152.44 and β to be 567.11 for the MSP430FR5994 in our `SW-SIC` experiments at frequencies less than or equal to 8 *MHz*. Our calculations for α and β in our `HW-SIC` tests were less consistent, an effect we attribute to the operation of the AES co-processor during only portions of the function's execution.

$$E = T \cdot (P_{dyn} + P_{static}) \qquad \begin{aligned} P_{dyn} &= \alpha \cdot f \\ P_{static} &= \beta \end{aligned} \tag{2}$$

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

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

The code size overhead of our solution is shown in Table I. The `SW-SIC` code required the largest space as `HW-SIC` was

TABLE I: Effect on Code Size, `.text` (B)

Optimization	NON-SIC	HW-SIC	SW-SIC
-O0[1]	11,936	18,516	35,728
-O3[1]	6,932	14,556	21,432
-Os[1]	6,916	10,716	19,808

[1] Compiler optimization flags,-O0: no optimizations, -O3: highest level of optimization in terms of compile time and memory usage, -Os: optimization for code size

able to omit the software AES functions implemented in Cifra library using the AES accelerator available on-chip. Similarly, the reduction in code size between -O3 and -Os optimizations is larger for HW-SIC than SW-SIC because the more complex structure of CMAC favors more implementation trade-offs.

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

C. Discussion

Although adding security to an intermittent system results in a significant overhead, securing application continuity is feasible, even for very constrained devices powered solely from harvested energy.

The MSP430FR5994 Launchpad Development board we used for our experiments has a supercapacitor of 0.22 F which gives a total energy storage of 1.35 J at 3.5 V. With only 10% of this energy allocated to secure application continuity, an intermittent system can compute up to 3,136 or 2,363 secure checkpoints when using HW-SIC and SW-SIC, respectively. Therefore, secure application continuity can be supported from harvested energy. Moreover, careful application-specific optimizations can improve our experimental results further.

Although our study is limited to a single evaluation board, it shows that security should and can be added to application continuity in intermittent systems. Energy harvesters present in current intermittent systems can support the overhead of secure application continuity.

Finally, we stress that the proposed protocol is generic and can be applied to secure application continuity of any intermittent system. However, this work focused on the most challenging case, namely intermittent systems powered by harvested energy.

V. Conclusion

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

Acknowledgements

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

References

[1] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design considerations for solar energy harvesting wireless embedded systems," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, ser. IPSN '05. Piscataway, NJ, USA: IEEE Press, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1147685.1147764

[2] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec 2010.

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

[4] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '15. New York, NY, USA: ACM, 2015, pp. 575–585. [Online]. Available: http://doi.acm.org/10.1145/2737924.2737978

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

[6] J. V. D. Woude and M. Hicks, "Intermittent computation without hardware support or programmer intervention," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. GA: USENIX Association, 2016, pp. 17–32. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/vanderwoude

[7] S. Chhabra and Y. Solihin, "i-nvmm: A secure non-volatile main memory system with incremental encryption," in *38th International Symposium on Computer Architecture (ISCA 2011), June 4-8, 2011, San Jose, CA, USA*, 2011, pp. 177–188.

[8] S. Kannan, N. Karimi, O. Sinanoglu, and R. Karri, "Security vulnerabilities of emerging nonvolatile main memories and countermeasures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 1, pp. 2–15, Jan 2015.

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

[10] N. Burow, S. A. Carr, J. Nash, P. Larsen, M. Franz, S. Brunthaler, and M. Payer, "Control-flow integrity: Precision, security, and performance," *ACM Comput. Surv.*, vol. 50, no. 1, pp. 16:1–16:33, Apr. 2017.

[11] S. Nagarakatte, J. Zhao, M. M. Martin, and S. Zdancewic, "Softbound: Highly compatible and complete spatial memory safety for c," in *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '09. New York, NY, USA: ACM, 2009, pp. 245–258. [Online]. Available: http://doi.acm.org/10.1145/1542476.1542504

[12] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens, "Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base," in *Proceedings of the 22Nd USENIX Conference on Security*, ser. SEC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 479–494.

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

[14] *Zatara High-Performance, Secure, 32-Bit ARM Microcontroller*, Maxim Integrated. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/ZA9L1.pdf

[15] K. Eldefrawy, A. Francillon, D. Perito, and G. Tsudik, "SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust," in *NDSS 2012, 19th Annual Network and Distributed System Security Symposium, February 5-8, San Diego, USA*, 02 2012.

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

[17] M. J. Dworkin, "Recommendation for block cipher modes of operation: The CMAC mode for authentication," Tech. Rep., 2016. [Online]. Available: https://doi.org/10.6028/nist.sp.800-38b

[18] J. Birr-Pixton, "Cifra: Cryptographic primitive collection," https://github.com/ctz/cifra, 2017.

[19] N. Mouha, B. Mennink, A. V. Herrewege, D. Watanabe, B. Preneel, and I. Verbauwhede, "Chaskey: An efficient mac algorithm for 32-bit microcontrollers," Cryptology ePrint Archive, Report 2014/386, 2014, https://eprint.iacr.org/2014/386.

[20] D. Dinu, A. Biryukov, J. Großschädl, D. Khovratovich, Y. Le Corre, and L. Perrin, "FELICS – Fair evaluation of lightweight cryptographic systems," in *NIST Workshop on Lightweight Cryptography*, 2015.

[21] D. Dinu, Y. Le Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov, "Triathlon of lightweight block ciphers for the Internet of Things," *Journal of Cryptographic Engineering*, pp. 1–20, 2015.

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

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

[24] *MSP430 FRAM Quality and Reliability*, Texas Instruments, Mar. 2012, revised May 2014.