Energy-efficient Persistently Secure Block-based Differential Checkpointing for Energy Harvesting Devices

***Abstract*—Energy harvesting devices (EHDs) are becoming inevitable because of their ultra-low-power, highly portable, self-sustainable nature. But it is challenging to use these devices as they rely on sporadic and variable sources of ambient energy and are equipped with very small memories. NVM memories such as FRAM used in EHDs suffer from a security vulnerability where the data remains even after the system is powered down. An attacker with physical access to the system can extract sensitive information from the memory; our goal is to find an efficient way to protect the memory from such vulnerabilities. We propose a secure mechanism that includes encryption for protection against attacks such as snooping, splicing, and replay. Moreover, due to the intermittent energy, there is repeated loss of device state that can cause nontermination of the programs executing on these devices. We use block-based differential checkpointing as the state retention technique that achieves termination and guarantees the program's forward progress. Also, our work includes an efficient way of calculating and comparing energy to start the encryption and checkpointing process. We show a 27.81x improvement in the progress of the program, and more than 99% reduction in latency and energy overhead compared to the baseline method.**

***Index Terms*—Non-volatile Memory; Energy Harvesting Devices; Security; Differential Checkpointing; Checkpoint vulnerabilities; Cryptographic algorithm;**

I. Introduction

Battery-powered IoT devices are less portable and have limited lifetimes; also, the batteries need to be replaced often as and when they deplete. Furthermore, disposal of as many batteries is toxic to the environment as they contain nickel, lead, and mercury [12]. Thus, to reduce environmental pollution, the use of self-sustainable capacitor-based EHDs was introduced, for which the market is already over 500 million USD and is projected to double by 2030, due to its ease of deployment in remote and hazardous locations, where the battery-powered devices cannot be used. The memory used in such devices is ferroelectric random-access memories (FRAM); compared to dynamic and static random-access memories (DRAM and SRAM), FRAM has the advantage of being non-volatile, as it does not require a power supply to maintain stored data [3]. They also offer faster write and read times, higher endurance, and lower energy consumption compared to conventional non-volatile flash and electronically erasable programmable read-only memories (EEPROM).

The goal of this study is to find a solution to the security vulnerability of lingering data in a FRAM that must satisfy at least three requirements. One requirement is that data should be retained in memory in an encrypted form (unintelligible) so that the attackers cannot retrieve any sensitive information and the data can be recovered for normal system operation when the processor boots or wakes up from hibernation. A second requirement is that the encryption ability must not depend on a particular processor platform or require specific changes to the processor architecture. The final requirement is that the solution should not incur substantial performance or energy overheads for applications running on the system.

Unlike traditional IoT devices, EHDs extract energy from ambient sources like solar, RF signals, vibrations, and wind and are unpredictable [5]; therefore, an energy storage unit like a supercapacitor is used in EHDs, but still, the device may fail due to energy depletion caused by the insufficient power supply. In order for the system to function correctly despite a power failure, a checkpoint (backup) of the current program's state is done in the non-volatile memory (NVM). The checkpoint is restored, and execution is continued when sufficient energy is harvested by the system. This is called intermittent computing in EHDs [17].

Our major contributions to this proposal are:

- Our approach uses an AES-CBC hardware module to encrypt/decrypt and HMAC the checkpoints, in order to be independent of the processor platform.
- We use differential checkpointing, by tracking the changed memory blocks to reduce the overhead of encryption and as such the intermittent computing.
- Decisions about when to encrypt and checkpoint are made at run time and are assisted by the energy

monitor program.

- A predetermined location in the NVM is used for the encrypted checkpoint as we are sure about completing the checkpoint process using the energy monitor program. This will ease the run-time development and reduce the NVM requirement.
- By using direct memory access (DMA) to access FRAM, we further reduce the overhead.

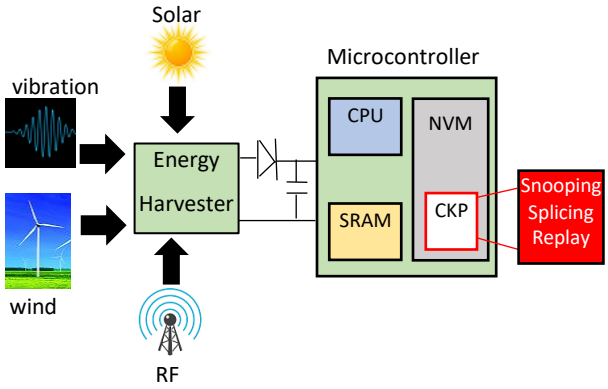

Fig. 1. Energy harvesting Checkpoint

II. Background and Motivation

A. Checkpoint Vulnerabilities

The security threat introduced by the checkpoint of an intermittent system could be because the attacker has physical access to the device and could employ techniques like on-chip probing [19]. Additionally, the attackers can also read/write to the memory through access ports or the JTAG interface. They can not only read sensitive information but also modify the data in non-volatile memory without physically damaging the device. In the following subsections, we'll elaborate on the different types of checkpoint attacks in Fig 1.

1) Checkpoint Snooping: Sensitive data like secret keys and application variables are available to the attacker when he/she has direct access to the checkpoints. If Compute Through Power Loss (CTPL) [16] is used, the attacker can study the utility or detect the pattern and extract sensitive information by identifying the location of the checkpoint in the memory.

2) Checkpoint Spoofing: Usually, while restoring the checkpoints after the power is back on, it is not verified if the checkpoint was not modified. The attacker can locate the sensitive variables and change their values without resetting the CTPL valid flag. Therefore, the device restores the tampered checkpoint at the next power-up and continues execution in an attacker-controlled sequence. The restored checkpoint will not correspond to a valid system or application state, and it might result in an unstable state, leading to a system crash.

3) Checkpoint Replay: An attacker can collect a few or more checkpoints by snooping and overwriting the current checkpoint in the NVM with an older checkpoint. This paves a way for the attacker to jump to any point in the software program since the CTPL does not check if it is the latest checkpoint before restoration after the power is back up.

B. Motivation

While working with EHDs like MSP430, using CTPL is inevitable, and it proves to be disastrous as far as security is concerned. The checkpoints created need to be protected, and the main aim of our work is to do the same. Some of the previous works, as seen in section 3, have tried to secure the data, but with higher energy overhead. If they have tried to reduce the overhead, they end up compromising on data protection, or they end up with tailor-made solutions based on the application, which can't be used on platforms where there is a lot of switching between applications. Some of the state-of-the-art have also proved that using hardware-based encryption has less overhead than using software, which is also used in our work.

The most important components of intermittent computing are recovering the program's pre-checkpoint state and ensuring the recovered program's state is consistent. There are many approaches to checkpointing [13]; one is backing up all volatile memory to NVM or a differential checkpointing scheme based on the concept of tracking changed memory blocks. Next is creating a checkpoint at a predetermined location every time in the NVM, which eases run-time development and reduces NVM requirements, but the checkpoint operation must be guaranteed to complete; otherwise, part of a previous checkpoint may be overwritten with the new checkpoint or make sure that there is a valid checkpoint to restore any time by having two checkpoints, old and the current one, which is called double buffering [20].

More efficient methods have been used to lower the overhead. These include differential checkpointing [6] (where the time and energy needed for a checkpoint depend on the total volatile memory size), an energy monitor program to make sure that the encryption and checkpointing process starts at the right time to make sure that every checkpoint is complete and not damaged, and direct memory access (DMA) [11] to access the FRAM, which lowers the overhead because it is faster to get to the internal FRAM memory. Moreover, our checkpoints are kept in the predetermined location as

we make sure our checkpoints are complete from the previous step.

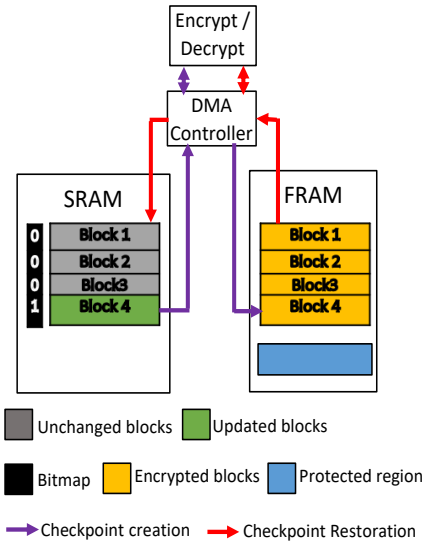

Fig. 2. System Overview

III. Related Work

At this day and age, using an energy harvesting system is inevitable, due to the reduction in the size of the devices. There has been a lot of research going on for keeping the data secure during the operation and storage in the device.

A. Secure with High Energy Overhead

In order to reduce the response latency and the energy requirement in a harvester-supported system, the structure is divided into online and offline portions. The online portion has a real-time dependency on the availability of data, and the offline portion is previously computed and stored in the memory to support the online portion. The cryptographic algorithms are partitioned into two for securing the online and offline portions; this is a way of optimizing cryptography in energy harvesting applications. [15].

Another way of securing the intermittent computing systems is by protecting the checkpoints stored in the non-volatile memory from tampering by external adversary sources. Energy harvester devices can be defended against checkpoint replay attacks by providing assurance that an application continues where it left off upon power loss. Suslowicz et al. [14] developed both hardware- and software-based solutions to solve this problem that provide data integrity, authenticity, and freshness to checkpoints, but neither of them solved the high energy overhead.

Krishnan et al. [9] proposed a protocol that associates every checkpoint with a unique power-on state to checkpoint replay, and every checkpoint is cryptographically connected to its predecessor. This helps in carrying run-time security properties when power interruption is used as an attack vector, which was a problem in the previous proposals. MSP430 was used to investigate the overhead of several cryptographic kernels, and they seem to cost an overhead of tens of seconds and hundreds of mJ of energy, which is a high overhead for any energy harvesting device.

B. Non-secure with Low Energy Overhead

In order to reduce the overhead, they introduced secure intermittent architecture (SIA), with both power-on and off states, self and remote attestation, and secure communication [2]. SIA did not have memory protection when the data was sent over system buses or stored in external memory from physical attacks, passive (bus snooping) and active (fault injection) attacks. SIA was also missing sealing where confidential code or data is wrapped in such a way that it can only be unwrapped under a certain configuration of device and/or software module. Next, the authors come up with an energy-harvester subsystem interface to optimize the run-time activity of the intermittent system, such that the wasted energy is eliminated and that run-time performance is improved [7]. They also studied the effects on the duty cycle of an embedded device by providing secure and stateful power transitions [10] and proposed a configurable checkpoint security protocol that is application-specific, using a suite of embedded benchmark applications [8].

C. Hardware Implementation with low overhead

Intermittent computing is also practiced in the Cortex M* series. Asad et al. [1] investigated different means to protect persistent state and concluded that software implementation bears a significant overhead in energy and time, sometimes harming forward progress, but also retaining the advantage of modularity and easier updates, and hardware implementations offer lower overhead but require a deeper understanding of their internals to gauge their applicability in given application scenarios. On the other hand, using ARM TrustZone shows almost negligible overhead, yet it requires different memory management and is only effective as long as attackers cannot directly access the NVMs.

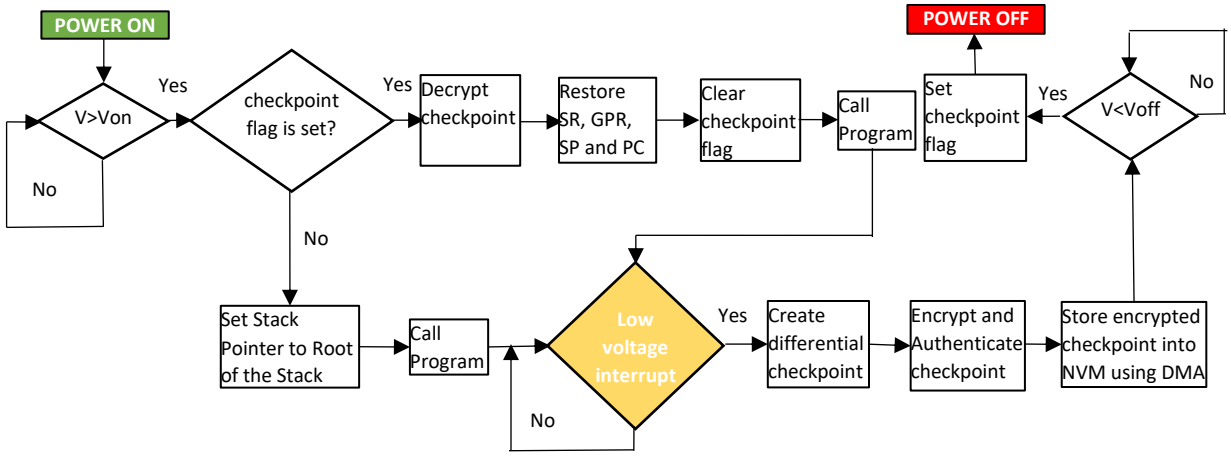

Fig. 3. Operational Flow

IV. Design

A. System Overview

Fig. 2 shows the overall system overview, consisting of four blocks: SRAM (volatile memory), which loses its contents when the device is powered off; FRAM (non-volatile memory), which retains its data even after the device is switched off; AES, which is a hardware module for encryption/decryption of the changed blocks; and the DMA controller, through which SRAM, FRAM, and AES communicate with each other. The data in the SRAM is divided into equal parts called blocks, and the blocks that change (green) during normal system operation are kept track of using the bitmap (black). The purple line and arrows show the checkpoint direction, where the changed block is encrypted and saved in the FRAM. In addition, the FRAM also contains the protected region, which contains some of the OS and program code, which does not change during the normal checkpointing process. When the power is back on, the system restores all the blocks in the checkpoint back to the SRAM, following the red line and the arrows to continue normal operation.

B. Operational Flow

The operational flow of the system is explained in Fig 3; the device switches on when the current voltage is higher than V_{on}. As soon as it is powered on, the device continues with the normal operation after the stack pointer is set at the root of the stack if the checkpoint flag is not set. Otherwise, the checkpoint is decrypted, the registers, stack pointer, and program counter are restored, and the checkpoint flag is cleared before continuing with the execution of the program. A low voltage interrupt is caused when there is a sudden drop in voltage or when there is not enough energy to continue normal operation, which causes a creation of a checkpoint. The checkpoint created is a differential checkpoint; the blocks that have changed since the last checkpoint are updated to the NVM as seen in Fig. 2. Moreover, we also encrypt the changed block before writing to the FRAM, which gives protection from the checkpoint vulnerabilities as discussed in section 2. In addition, we also use the DMA to access the FRAM, which will increase the compute speed in comparison to using standard I/O peripherals. Whenever the current voltage goes below V_{Off}, the checkpoint flag is set, and the device gets powered down.

Algorithm 1 Energy Calculation Algorithm

Input: C-capacitance, V_{max}-max operating voltage of the device, V_{min}-min operating voltage of the device, x-voltage interval for energy calculation

Output: Set with Voltage, Energy pair [V,E] at x interval, y-number of [V,E] pairs

```
y = 0
for V_i = V_max to V_min do
    E_i = 1/2*CV_i^2
    Save the voltage and corresponding energy in the set [V_i,E_i]
    y = y + 1
    V_i=V_i-x
end for
return [V,E], y =0
```

C. Energy Monitor

1) Energy Pre-Calculation: As discussed in the system overview, when there is only enough energy to

encrypt and checkpoint the changed blocks, the system creates a checkpoint before powering off. In order to compare the energy level at any given time, we have pre-calculated the energy at multiple voltages and save it in a set of voltage, energy [V,E] pair. The concept is to calculate energy for current V_i from V_{max} to V_{min} at an interval of x volts as seen in the algorithm 1. The values of V_{max}, V_{min}, and x are decided based on the device, the network connection, and the usage. In addition to the above, the value of the storage capacitance C is also given as input to the algorithm. The output will be the set of voltage and energy [V,E] pairs and the number of [V,E] pairs in the set as y.

2) Start Checkpoint Process: Even though we have pre-calculated the energy for different voltages, we also need to start the process of checkpointing at a particular current voltage, which is explained in algorithm 2. The inputs to the algorithm are the maximum and minimum voltage required for the device in order to encrypt and checkpoint all the blocks and at least one block, respectively. We use the calculated set of [V,E] pairs from the algorithm 1 to compare the energy calculated with the current energy the device is left with, after which a decision can be made if there is enough energy to encrypt and checkpoint the changed blocks. We keep track of the changed blocks by maintaining a bitmap M. We also input the total number of blocks and the number of [V,E] pairs so we know the upper limit to the number of changed blocks and can search the right energy number from the set of [V,E] pairs, respectively.

There are two methods the algorithm is triggered: one is when there is a changed block, and the other is when there is a low voltage interrupt. In the first method, after every block has changed, algorithm 2 checks if there is enough energy to encrypt and checkpoint the blocks that have changed so far. When the energy required matches the energy available, the encryption and checkpointing process is started after the bitmap M is reset. In case there are a lot of changes in one specific block and the energy is draining more than expected and we can't wait till it moves on to the next block, we use the second method of triggering a low voltage interrupt after a certain time to start the encryption and checkpointing process.

Algorithm 2 Start Checkpoint Algorithm

Input: V_{max}-max operating voltage of the device, V_{min}-min operating voltage of the device, V_t-current voltage, set [V,E], E_0-energy needed for encrypting and checkpoint one block, M-bitmap to record the changed blocks, m-number of changed blocks, n-total number of blocks, y-number of [V,E] pairs

```
1:  m = 0
2:  i = 0
3:  if block_i has changed then
4:      set i_th bit in M
5:      m = m + 1
6:      if V_t is between V_max and V_min then
7:          for j = 1 to y do
8:              if V_t is less than or equal to V_j then
9:                  E = E_j
10:             else
11:                 E = E_j+1
12:             end if
13:         end for
14:     end if
15:     if E_t is between E_0*(m+1) and E_0*(m+2) then
16:         Reset bitmap M
17:         Start Checkpoint
18:         Break
19:     end if
20: end if
21: if i is between 0 and n then
22:     i = i + 1
23:     go to 3
24: end if
25: if low voltage interrupt then
26:     if V_t is between V_min and V_max then
27:         if E_t is between E_0*(m+1) and E_0*(m+2) then
28:             Reset bitmap M
29:             Start Checkpoint
30:             Break
31:         end if
32:     end if
33: end if=0
```

D. AES-CBC Encryption

Advanced Encryption Standard - Cipher Block Chaining (AES-CBC) has been the most commonly used mode of encryption [18], where each block of plaintext is XORed with the previous ciphertext block before being encrypted. Therefore, each ciphertext block depends on all plaintext blocks processed up to that point, and to make each message unique, an initialization vector (IV) must be used in the first block. So, one does not need to decrypt the previous block before using it as the IV for the decryption of the current one; hence, decryption can be parallelizable, even though the encryption has to be sequential, which will in turn reduce the latency for decryption. Another advantage of using CBC is that even

if a random cipher block is corrupted, it can be safely discarded, and the rest of the decryption can be done with no change in the original plaintext.

E. HMAC

In cryptography, an HMAC (hash-based message authentication code) involves a cryptographic hash function and a secret cryptographic key, which may be used to simultaneously verify both the data integrity and authenticity of a message [4]. HMAC uses two passes of hash computation. The first pass of the hash algorithm produces an internal hash derived from the message and the inner key, and the second pass produces the final HMAC code derived from the inner hash result and the outer key. Both the outer and inner keys are derived from the secret cryptographic key. Therefore, the algorithm provides better immunity against length extension attacks caused by using one key and one pass. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and the size and quality of the key. The size of the output of HMAC is the same as that of the underlying hash function (e.g., 256 and 512 bits in the case of SHA-256 and SHA3-512, respectively), although it can be truncated if desired.

F. Block-based Differential Checkpointing

It is difficult to run programs continuously for devices with intermittent power because the program keeps important state in volatile memory (SRAM), which is irretrievably erased when a device loses power, and execution starts over from the beginning again when the power is restored. One solution is to save the SRAM state in the NVM (non-volatile memory) with checkpoints, which persists even after a power failure. When the device has enough power again, the system retrieves the state saved in the NVM and continues execution from that state. Although several checkpointing strategies have been proposed, none have given an energy-efficient and time-saving solution for the EHDs. Since the cost of a checkpoint is directly proportional to the amount of memory the checkpoint must save, we have taken measures to reduce it with a differential checkpoint that saves the state of a program at specific intervals, recording only the changes made since the last checkpoint. The intervals could be a specific time interval, but our proposed technique is designed to perform the differential checkpointing, being aware of the harvested energy level at run-time to avoid incomplete checkpoints caused by sudden power losses.

We analyze the overhead of the checkpoint procedure and present another optimization to decrease the checkpoint time by splitting the SRAM into blocks and checkpointing only the changed blocks instead of the entire SRAM. Our implementation further reduces the checkpoint time by maintaining the same location in the FRAM for each block at all times, preserving the program state, and ensuring data consistency during recovery after power is restored. Altogether, all our efforts to mitigate the energy overhead and latency of checkpointing by understanding the power-failure characteristics of the harvested energy source have maximized the forward progress of the program more effectively than the state-of-the-art techniques.

V. Evaluation

A. Experimental Set-up

The platform used for the experiments in this project is the Texas Instruments MSP430FR5994 16MHz microcontroller. The MSP430FR5994 features a 16-bit RISC architecture and is equipped with 256KB of FRAM (Ferroelectric Random Access Memory) and 8KB of SRAM. We evaluate the performance of the block-based differential checkpointing by focusing on the latency and energy consumption of encryption, decryption, hashing, and checkpointing the contents of SRAM in case of a voltage drop of the EHDs capacitor. After the measurements of the above parameters, a simulator is used to run the tests for 5 minutes on each application, and the progress made is calculated. The energy harvesting is also included during the simulation for the capacitor of 60 mF to get charged before progressing further on the benchmarks in order to avoid power failure. The experimental results show that the proposed method can perform with minimal energy and time overhead for a block size of 128 bytes when compared to the previous work in the field.

B. Results

The two methods used for AES-128 encryption are the accelerator in the MSP430FR5994 and the software program that is run on the microcontroller's CPU to

TABLE I
Comparison Methods

Method Name	Checkpoint Contents	Checkpoint Option	Encryption Option
CPU SRAM	Entire SRAM	CPU	SW
CPU SW	Changed Blocks	CPU	SW
DMA SW	Changed Blocks	DMA	SW
CPU Acc	Changed Blocks	CPU	Accelerator
DMA Acc	Changed Blocks	DMA	Accelerator

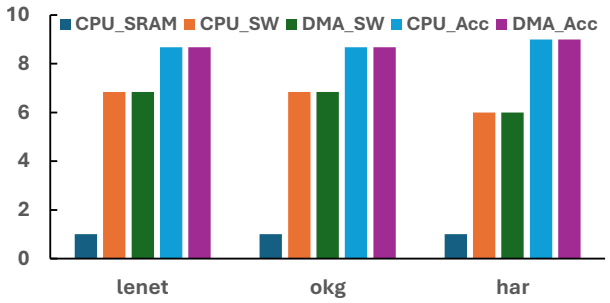

Fig. 4. Progress made by the applications (Normalized to CPU SRAM) for simulation run of 5 minutes with capacitor 60mF

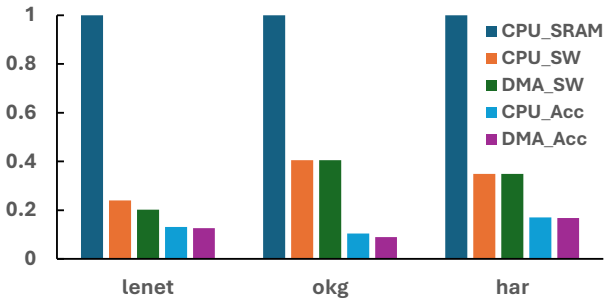

Fig. 5. Energy Overhead (Normalized to CPU SRAM) for simulation run of 5 minutes with 60mF capacitor

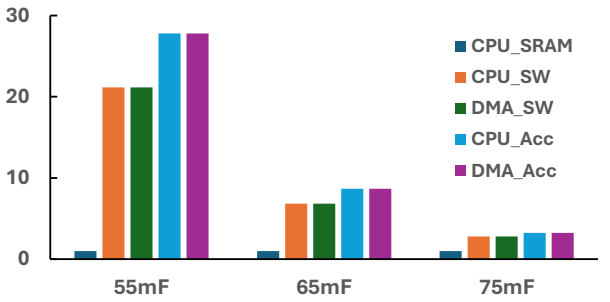

Fig. 6. Progress made by lenet with different capacitors (Normalized to CPU SRAM) for simulation run of 5 minutes

ensure data confidentiality. After encryption, each block is hashed using the HMAC-SHA256 algorithm to provide data integrity. Moreover, we have implemented two ways of moving the data from SRAM to FRAM during checkpointing, through DMA and the CPU. To show the massive improvement of using the DMA, different methods are shown in TABLE I. The machine learning applications like LeNet, OKG, and HAR are selected to stress our model to prove the best of the considerations.

1) Progress: LeNet, OKG, and HAR tests were run in an energy harvesting environment continuously for five minutes, and it was observed that the most progress was made by the option DMA Acc, as seen in Fig. 4. On average for all three benchmarks, the run time for DMA SW is 6.56x and DMA Acc is 8.79x of the baseline CPU SRAM, showing that DMA Acc is the best method to consider.

2) Energy Overhead: When the CPU SRAM method is used as the baseline, the energy overhead to encrypt, hash, and checkpoint for the Accelerator methods is as low as 0.14x and 0.13x of the CPU SRAM for the CPU Acc and DMA Acc methods, respectively, on average for all three benchmarks of LeNet, OKG, and HAR. Whereas the CPU and DMA SW have, respectively, an energy overhead of 0.33x and 0.32x of CPU SRAM, as seen in Fig. 5.

3) Different Capacitors: We decided to investigate our proposed technique by using different capacitors; the results are shown in Fig. 6 for lenet. It is clearly seen that as the capacitor value increases, the difference between the baseline and the proposed technique seems to reduce. The highest performer is the CPU Acc at 55 mF with 27.81x of progress made when compared to CPU SRAM, and the lowest performer is the CPU SW at 75 mF with 2.78x. Moreover, the latency shows an opposite trend of 0.06x of the baseline for CPU Acc at 55mF and 0.39x for CPU SW at 75mF as seen in Fig 7. Hence, it is proved that our proposed technique works best with lower capacitance, which means less time to charge and, in turn, to harvest energy.

4) Different Power Sources: Even though the 60mF capacitor is completely charged before we start to run the simulation, there is not enough time to recharge with the Wifi option; recharge it once with the solar and multiple times with the thermal. The progress made with Thermal is between 8 and 9 times, with an average of 8.72x, and the progress made with Solar on average is 1.9x across all four techniques, including the baseline as seen in Fig 8. The simulation was run on OKG for 20 minutes in order to give enough time for energy harvesting through different sources. Results show that our proposed technique will work well with different energy sources just like the baseline method. However, having unintercepted and higher power signals will definitely help with faster progress.

VI. Conclusion

This paper proposes a secure mechanism for checkpointing in energy harvesting devices, right before the power failure. The mechanism protects the data from confidentiality and integrity attacks with low overheads. Considering the evaluation on progress, latency, and

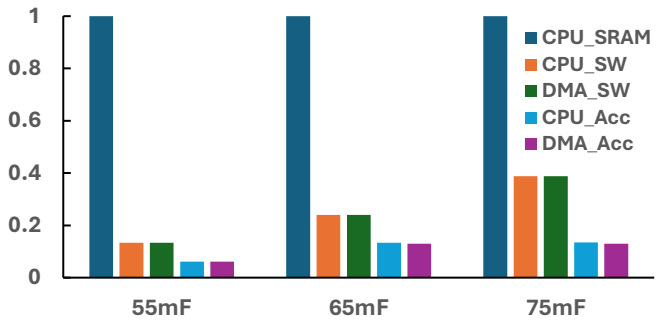

Fig. 7. Latency for lenet with different capacitors (Normalized to CPU SRAM) for simulation run of 5 minutes

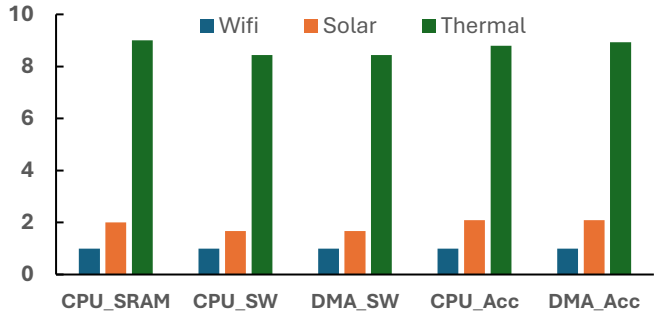

Fig. 8. Progress made by okg (normalized to CPU SRAM) with different Power sources for a simulation run of 20 minutes

energy overhead, DMA Acc is an efficient mechanism combined with our energy monitor program and our block-based differential checkpointing method when compared to the existing state-of-the-art. Our results also prove that the proposed technique will work on lower charging capacitors and with different energy sources in an effective manner.

REFERENCES

[1] H. A. Asad, E. H. Wouters, N. A. Bhatti, L. Mottola, and T. Voigt. On securing persistent state in intermittent computing. In *Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, pages 8–14, 2020.

[2] D. Dinu, A. S. Khrishnan, and P. Schaumont. Sia: Secure intermittent architecture for off-the-shelf resource-constrained microcontrollers. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 208–217, 2019.

[3] T. Eshita, T. Tamura, and Y. Arimoto. Ferroelectric random access memory (fram) devices. In *Advances in non-volatile memory and storage technology*, pages 434–454. Elsevier, 2014.

[4] S. Jiang, X. Zhu, and L. Wang. An efficient anonymous batch authentication scheme based on hmac for vanets. *IEEE Transactions on Intelligent Transportation Systems*, 17(8):2193–2204, 2016.

[5] O. Kanoun. *Energy Harvesting for Wireless Sensor Networks: Technology, Components and System Design*. Walter de Gruyter GmbH & Co KG, 2018.

[6] K. Keller and L. Bautista-Gomez. Application-level differential checkpointing for hpc applications with dynamic datasets. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 52–61. IEEE, 2019.

[7] A. Krishnan and P. Schaumont. Hardware support for secure intermittent architectures. In *Workshop on Energy-Secure System Architectures (ESSA)*, 2019.

[8] A. S. Krishnan and P. Schaumont. Benchmarking and configuring security levels in intermittent computing. *ACM Transactions on Embedded Computing Systems (TECS)*, 2022.

[9] A. S. Krishnan, C. Suslowicz, D. Dinu, and P. Schaumont. Secure intermittent computing protocol: Protecting state across power loss. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 734–739. IEEE, 2019.

[10] A. S. Krishnan, C. Suslowicz, and P. Schaumont. Secure and stateful power transitions in embedded systems. *Journal of Hardware and Systems Security*, 4(4):263–276, 2020.

[11] S. Ma, L. Huang, Y. Lei, Y. Guo, and Z. Wang. An efficient direct memory access (dma) controller for scientific computing accelerators. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

[12] S. Schismenos, M. Chalaris, and G. Stevens. Battery hazards and safety: A scoping review for lead acid and silver-zinc batteries. *Safety science*, 140:105290, 2021.

[13] P. Singla and S. R. Sarangi. A survey and experimental analysis of checkpointing techniques for energy harvesting devices. *Journal of Systems Architecture*, 126:102464, 2022.

[14] C. Suslowicz, A. S. Krishnan, D. Dinu, and P. Schaumont. Secure application continuity in intermittent systems. In *2018 Ninth International Green and Sustainable Computing Conference (IGSC)*, pages 1–8. IEEE, 2018.

[15] C. Suslowicz, A. S. Krishnan, and P. Schaumont. Optimizing cryptography in energy harvesting applications. In *Proceedings of the 2017 Workshop on Attacks and Solutions in Hardware Security*, pages 17–26, 2017.

[16] TI. TI CTPL. https://www.ti.com/tool/TIDM-FRAM-CTPL/, 2015. [Online; https://www.ti.com/tool/TIDM-FRAM-CTPL5].

[17] S. Umesh and S. Mittal. A survey of techniques for intermittent computing. *Journal of Systems Architecture*, 112:101859, 2021.

[18] M. Vaidehi and B. J. Rabi. Design and analysis of aes-cbc mode for high security applications. In *Second International Conference on Current Trends In Engineering and Technology-ICCTET 2014*, pages 499–502. IEEE, 2014.

[19] H. Wang, D. Forte, M. M. Tehranipoor, and Q. Shi. Probing attacks on integrated circuits: Challenges and research opportunities. *IEEE Design & Test*, 34(5):63–71, 2017.

[20] S. Wu, F. Zhou, X. Gao, H. Jin, and J. Ren. Dual-page checkpointing: An architectural approach to efficient data persistence for in-memory applications. *ACM Transactions on Architecture and Code Optimization (TACO)*, 15(4):1–27, 2019.