On Securing Persistent State in Intermittent Computing

Hafiz Areeb Asad
Uppsala University
Sweden
areebasad95@gmail.com

Erik Henricus Wouters
KTH Royal Institute of Technology
Sweden
ehwo@kth.se

Naveed Anwar Bhatti
Air University
Pakistan
naveed.bhatti@mail.au.edu.pk

Luca Mottola
Uppsala University, Sweden and RISE
Sweden
luca.mottola@polimi.it

Thiemo Voigt
Uppsala University, Sweden and RISE
Sweden
thiemo.voigt.@it.uu.se

ABSTRACT

We present the experimental evaluation of different security mechanisms applied to persistent state in intermittent computing. Whenever executions become intermittent because of energy scarcity, systems employ persistent state on non-volatile memories (NVMs) to ensure forward progress of applications. Persistent state spans operating system and network stack, as well as applications. While a device is off recharging energy buffers, persistent state on NVMs may be subject to security threats such as stealing sensitive information or tampering with configuration data, which may ultimately corrupt the device state and render the system unusable. Based on modern platforms of the Cortex M* series, we experimentally investigate the impact on typical intermittent computing workloads of different means to protect persistent state, including software and hardware implementations of staple encryption algorithms and the use of ARM TrustZone protection mechanisms. Our results indicate that *i)* software implementations bear a significant overhead in energy and time, sometimes harming forward progress, but also retaining the advantage of modularity and easier updates; *ii)* hardware implementations offer much lower overhead compared to their software counterparts, but require a deeper understanding of their internals to gauge their applicability in given application scenarios; and *iii)* TrustZone shows almost negligible overhead, yet it requires a different memory management and is only effective as long as attackers cannot directly access the NVMs.

CCS CONCEPTS

• **Security and privacy → Embedded systems security**; • **Computer systems organization → Embedded software**.

KEYWORDS

intermittent computing, transiently-powered embedded system, embedded security

ACM Reference Format:
Hafiz Areeb Asad, Erik Henricus Wouters, Naveed Anwar Bhatti, Luca Mottola, and Thiemo Voigt. 2020. On Securing Persistent State in Intermittent Computing. In *The 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems (ENSsys '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3417308.3430267

1 INTRODUCTION

Energy harvesting allows embedded sensing devices to mitigate, if not to eliminate, their dependency on traditional batteries. However, because of erratic energy patterns [6], unanticipated system shutdowns are difficult to avoid. Computing then becomes *intermittent* [11]; periods of normal computation and periods of energy harvesting come to be unpredictably interleaved.

Problem. System support exists to enable intermittent computing, employing either a form of *checkpointing* or *task-based programming abstractions* [9, 12, 15, 21] to let the program cross periods of energy unavailability [4, 16]. Both approaches rely on some form of *persistent state* stored onto non-volatile memory (NVM) in anticipation of power failures. Persistent state is then retrieved back from NVM once the system resumes with sufficient energy.

Persistent state may include critical information on operating system and network stack configurations, as well as application data. Crucially, while the device is off recharging energy buffers, persistent state may be subject to a variety of security threats. For example, depending on NVM technology, an attacker may steal precious information from persistent state or corrupt the data in ways that prevent the application to operate correctly ever after.

To address these issues, we experimentally investigate the use of staple encryption mechanisms and system-level features offered by modern architectures to protect persistent state. Using microcontroller units (MCUs) of the Cortex M* series, we measure the energy and time overhead of software- and hardware-based encryption algorithms, such as the Advanced Encryption Standard (AES) and ARM TrustZone system-level protection. We assess the impact of the corresponding overhead on typical intermittent computing workloads against the provided level of protection.

The conclusions we draw are that *i)* despite the advantages due to increased modularity and easier updates, software implementations impose a significant overhead in energy and time, which may result in preventing forward progress of applications; *ii)* hardware implementations yield much lower overhead, but require a deeper understanding of their internals to gauge their applicability,

ENSsys '20, November 16–19, 2020, Virtual Event, Japan
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8129-1/20/11…$15.00
https://doi.org/10.1145/3417308.3430267

as for example when protecting against side-channel attacks; and *iii)* TrustZone shows almost negligible overhead, yet it requires a different memory management that complicates implementations and is only effective as long as the threat model does not include the attacker's ability to directly access the NVM bypassing the MCU.

The remainder of the paper unfolds as follows. After a brief account of the state of the art in intermittent computing and related security solutions, reported in Sec. 2, we describe the experimental setup in Sec. 3. Based on this, Sec. 4 discusses the results we gather and the conclusions we can draw. We end the paper in Sec. 5 with an outlook on follow-up directions and brief concluding remarks.

2 BACKGROUND AND RELATED WORK

In this section, we provide necessary background information and a brief account of related work.

Intermittent computing. Frequent unanticipated power failures hinder continuous operation as the device loses the progress achieved, restarting from scratch when energy is back.

Several solutions exist to ensure forward progress efficiently. Many of these are based on some form of checkpoint [3, 4, 7, 13, 16, 20] and apply techniques such as periodic or dynamic checkpointing [3, 4, 13, 16], differential checkpointing [2], and compiler-based analysis [7, 20] In contrast to checkpoint mechanisms that apply to unmodified sources, task-based programming abstractions [9, 12, 15, 21] require programmers to split the application in separate tasks executing with transactional semantics. The underlying runtime takes care of ensuring the results of a completed task are made persistent before transitioning to the next task.

In available solutions, the system state is persisted on NVM fully or partially. The confidentiality and integrity of the persisted data is questionable and to date, only a few solutions address these issues.

Security persistent state. Persistent state on NVM includes system configurations and application data present in main memory at run-time, and is thus sensitive.

A foundation to reason on the related security problems is offered by S. Krishnan et al. [17], who present an attack model for unsecured and cryptographically secured checkpoints for both knowledgeable and blind attackers with hardware access, that is, the ability to read and write NVM. They show how persistent state is vulnerable to sniffing, spoofing, or replay attacks [17]. An attacker may simply sniff persistent state by reading the contents of the NVM using debug ports. Sniffing may be prevented by encrypting data on NVM, thus ensuring confidentiality, but does not prevent an attacker from spoofing, that is, tampering encrypted data, which threats the checkpoint integrity. By collecting several checkpoints, attackers may execute the previous checkpoints to repeat the execution order.

A few solutions are available to secure persistent state. Ghodsi et al. [10] use lightweight algorithms [8] for securing checkpoints, ensuring confidentiality. Valea et al. [19] propose a SECure Context Saving (SECCS) hardware module inside the MCU, using Trivium stream cipher to encrypt data and SHA-256 as MAC module to provide data integrity, thus providing both confidentiality and integrity, at the cost of hardware modifications. Khrishnan et al. [14] present a generic secure protocol and apply Authenticated Encryption with Associated Data (AEAD) to protect checkpointing data. AEAD provides both confidentiality and integrity simultaneously. To secure checkpoints, they use EAX [5] by accelerating the algorithm using the on-chip AES module on TI MSP 430 MCUs.

Complementing these early works, our goal is to provide a quantitative assessment and comparison of the overheads imposed by software- or hardware-based implementations of AES and using ARM TrustZone, as available on recent Cortex M* MCUs.

3 EXPERIMENTAL SETUP

We implement benchmarks that run directly on bare-metal without operating system support. They emulate the occurrence of checkpoint and restore operations, while measuring the overhead of given security mechanisms.

We describe next the security mechanisms we test, the corresponding threat models, and the target hardware. Note that the threat models we assume for software- or hardware-based encryption as opposed to TrustZone are inherently different, thus the quantitative data we provide in Sec. 4 are not directly comparable.

3.1 Security Mechanisms

We experimentally investigate the use of AES and ARM TrustZone to protect persistent state in intermittent computing.

AES. The AES symmetric cipher represents an industry standard. We assume that the attacker has physical access to the device and may snoop (read) or spoof (tamper) persistent data on NVM. We also assume that keys and tags are stored in a secure key area instead of plain NVM. An attacker may also execute replay attacks by collecting sequences of encrypted checkpoints. We use three block cipher variants of AES: Electronic Cipher Mode (ECB), Chain Block Cipher (CBC) and Galois Counter Mode (GCM).

Encryption. The basic AES mode is ECB, which supports both ways parallel encryption and decryption. The attacker may, however, rearrange the ciphertext blocks in an arbitrary order, and repeat or omit blocks to construct a different valid ciphertext, which makes it insecure [18]. The CBC mode circumvents this by using an initialization vector as an input in addition to the plaintext, ensuring that distinct ciphertexts are produced even when the same plaintext is encrypted multiple times with the same key. Both the ECB and CBC variants ensure confidentiality. GCM is one of the AEAD-based schemes that ensure confidentiality and integrity. Ensuring integrity is required to avoid spoofing, that is, to prevent that the attacker tampers with the encrypted checkpoint and drives the system into an unstable state.

The most primitive attacks to break encryption algorithms are brute force attacks. AES is generally considered secure against these. However, more fierce attacks against encryption algorithms are side-channel attacks, where the adversary exploits the physical characteristics of a running system, such as electromagnetic radiations and power usage during encryption operations, to gain information useful to retrieve the keys. In intermittent computing, such an attack may become more viable, as the adversary can re-execute the same encrypted checkpoint multiple times to acquire information for side-channel attacks. It is therefore necessary to employ countermeasures to prevent information leakage.

AES may be implemented in software or with dedicated hardware support. We experiment with both. To check software-based AES implementations, we select two existing well-known cryptographic libraries: the mbedTLS [1] and WolfCrypt[2] cryptographic libraries. In contrast, special-purpose hardware modules implementing AES exist to reduce the load on the main computing unit. Both hardware platforms we consider, described next, are equipped with such hardware support, albeit with different features and performance.

TrustZone. ARM describes a collection of hardware security extensions to the Cortex family of 32-bit processors in the TrustZone specifications. Similar to Intel's Software Guard Extensions, it provides a Trusted Execution Environment (TEE) intended to be more secure than the user-facing operating system. Accordingly, the TrustZone architecture exposes a secure world processor context besides the normal world context. The separation between the secure and normal world is achieved by extending the memory management to split the physical memory into secure and normal regions. We use the secure region for checkpoint data.

In the case of TrustZone, we assume that the adversary has the ability to run malicious code on the device, either by installing it remotely or through physical access to the device. We also assume that attackers may read the normal world memory. Data sections are not executable and code sections are not writable, but they are readable. Note that checkpoints are not encrypted when using TrustZone, which only protects the persistent data from malicious software accesses. We therefore assume that the debug port is not vulnerable, or attackers may read and tamper checkpoint data.

We place the subroutine to perform a checkpoint in the secure world. The checkpoint is triggered from the non-secure world and copies the contents of the main memory, program counter, and register file to a secure region in the non-volatile memory directly. The checkpoint is restored directly from the secure region of the flash to the non-secure memory. We configure the Secure Attribution Unit (SAU) of our target MCU, described next, to protect the region of the non-volatile memory where we store persistent state from reads and writes originating in the normal world. We implement a test application to confirm that reading or writing to this address range does trigger a specific exception.

3.2 Hardware

We perform our experiments on ultra-low power ARM MCUs from Microchip, namely the the ATSAML21J18B Cortex-M0+[3] and ATSAML11E16A Cortex-M23[4]. Due to their energy figures, both MCUs are credible targets for energy harvesting [20].

We consider the ATSAML21J18B Cortex-M0+ since it supports several AES and performance modes. The Cortex-M0+ core is based on the ARMv6-M instruction set and can operate at up-to 48 MHz within the same voltage range as the Cortex-M23, with a current consumption below $35\mu A/MHz$ in active mode. The MCU has an internal flash memory of 256 Kbytes and 40 Kbytes of SRAM, split in 32 KBytes of main memory and 8 Kbytes of low-power memory. It includes a separate AES module that incorporates advanced AES modes such as CBC, CTR, and GCM, along with basic AES-ECB mode. The MCU offers two performance levels: the default performance level 0 (P0) and performance level 2 (P2). P0 enables maximum energy efficiency, while P2 allows the MCU to run at maximum operating frequency.

We select the ATSAML11E16A specifically as it features TrustZone technology. At the time of writing, it represents one of the very few MCUs equipped with such technology. It operates at a low voltage range of 1.62V to 3.63V with a current consumption below $25\mu A/MHz$ in active mode. The Cortex-M23 core is based on the ARMv8-M instruction set and contains 64 Kbytes of flash memory, 16 Kbytes of Static Random-Access Memory (SRAM), secure-key storage, True Random Number Generator (TRNG), and a basic crypto accelerator. The crypto accelerator only supports AES-ECB mode and two hashing algorithms.

3.3 Metrics and Workloads

Our primary metrics are time and energy overhead due to the use of either security mechanism described earlier. We compare this figure to executions where checkpoints and restore operations occur with no security feature. We use Microchip's power measurement module XAM, available on both development boards we use, to measure the current drawn by the MCU and by its peripherals. We correlate code execution and energy consumption by manipulating the GPIO pins at the boundaries of different code sections.

The workloads we consider allow us to measure the impact of either metrics on the energy patterns of typical intermittently-computing applications. Each workload is iterated 50 times and we compute the average of the results to factor out inaccuracies in the measurement setup. Similar to existing work, we consider *i)* a simple Bit Counting procedure performed over 2048 random bits, *ii)* an implementation of the Dijkstra's Shortest Path algorithm fed with a graph of 128 nodes and 512 edges, *iii)* an implementation of the Fast Fourier Transform over 128 complex numbers with two decimal points, and *iv)* an implementation of the Quicksort algorithm operating on an array of 256 elements.

To decide when to take a checkpoint, we analytically calculate the latest point in time when it is necessary and possible to do so, that is, whenever the remaining energy is barely sufficient to dump the system state on the MCU's internal flash memory. This is determined based on a given energy budget, corresponding to the charge of a given capacitor.

4 EVALUATION

We discuss first the results characterizing the intermittent execution of the workloads we consider. Next, we dissect the system operation and study where and how energy is specifically spent by the different security mechanisms. We mainly focus on checkpoint operations, as restore operations show similar trends. The complete code of our experiments is available online [1].

Workload characterization. Fig. 1 reports the energy measurements obtained by running 50 iterations of every benchmark application, using either software-based AES encryption, the onboard crypto accelerator, TrustZone, or no security mechanism.

With a small capacitor of 60μF, as shown in Fig. 1a and Fig. 1b, the Bit Counting procedure fails most often regardless of the security mechanism and sometimes even in the case of no security. This is because the capacitor is too small to accumulate sufficient energy to perform some application processing, execute security mechanisms

[1] https://tls.mbed.org/
[2] https://www.wolfssl.com/products/wolfcrypt-2/
[3] https://developer.arm.com/ip-products/processors/cortex-m/cortex-m0-plus
[4] https://developer.arm.com/ip-products/processors/cortex-m/cortex-m23

if any, and write on flash in a single power cycle. Note that applying any security mechanism provides no progress for the application, as it merely represents overhead. The workload can only complete using hardware-based encryption or TrustZone on the Cortex M23, with both showing a performance almost identical to no security.

In all other cases of Fig. 1a and Fig. 1b, only the internal crypto accelerator and TrustZone make it possible to apply some security mechanism within the given energy budget and also complete the workload. Using the crypto accelerator, the overhead imposed compared to the case of no security is roughly comparable to the cost of application processing, as shown in Fig. 1a. In the case of TrustZone, the overhead is marginal, which makes the energy figure similar to the case of no security, as shown in Fig. 1b. The software implementation of AES, on the other hand, makes it again impossible to complete the workload. Although a software implementation offers better modularity and the possibility of future updates, here it becomes a major hampering factor due to the extreme scarcity of energy, resulting in no forward progress.

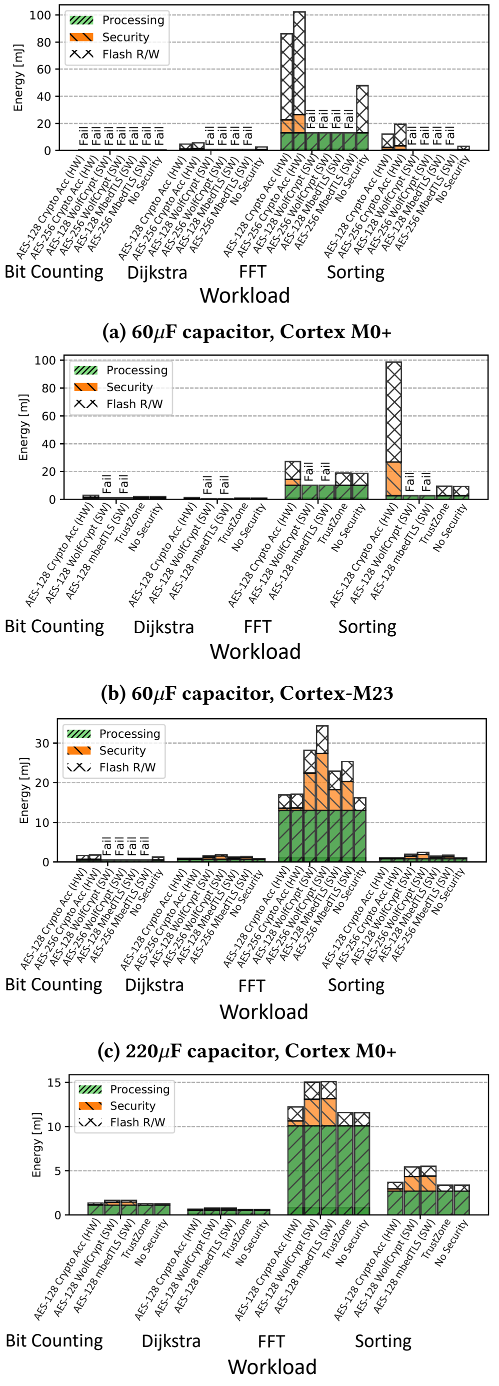
The same experiments executed with a 220μF capacitor, as shown in Fig. 1c and Fig. 1d, show the Bit Counting procedure again failing for all software implementations of AES using the Cortex M0+, but all other benchmarks completing successfully. The energy overhead for relying on the onboard crypto accelerato is now appreciable, but limited, whereas TrustZone imposes essentially no additional cost. The performance penalty due to running AES in software compared to the overall energy expenditures of an intermittent execution is, however, significant. We find this to be often higher than the cost of operating on the internal flash. As this operation repeats at every power cycle, the impact on the overall energy efficiency is notable and indicates a clear opportunity for optimisation.

Energy analysis. Fig. 2 reports the analisys of energy and time spent in performing a checkpoint operation for a given amount of data, by possibly applying a given security mechanism.

As for energy, the hardware implementation of AES on the onboard crypto accelerator bears half the overhead of equivalent software implementation(s) for the Cortex M0+, as shown in Fig. 2a, and only a fraction of that on the Cortex M23, as shown in Fig. 2b. The performance of a software implementation compared to using the crypto accelerator may be intuitive at first, yet we found no indication of the quantitative gap between the two in existing literature and in the context of intermittent computing. We also recognize the use of TrustZone in intermittent computing to be largely unexplored. Our measurements indicate that, whenever available and provided the treat model is respected, TrustZone is to be preferred to protect persistent state in terms of energy consumption, as the overhead is almost negligible.

Fig. 2a prompts two additional observations. First, using the crypto accelerator, the energy cost of running AES-128 or AES-256 is basically the same. Should energy be the main concern, there would be no reason to weaken the level of protection. Second, a marked gap also exists between the two software implementations we test, with mbedTLS outperforming wolfCrypt on the Cortex M0+. If hardware support is not available, the choice of an efficient implementation becomes crucial depending on the platform.

The time measurements are reported in Fig. 2c and Fig. 2d. Using the Cortex M0+, the conclusions remain largely the same as in Fig. 2a, but the gaps are sometimes amplified. For example, a small gap emerges between 128- or 256-bit keys with the crypto accelerator. We also experiment with GCM mode, and obtain similar results. This is despite the additional tags GCM uses that would, in principle, increase NVM operations and yet, because of the granularity of flash reads/writes, rarely become a factor. In relative terms, the



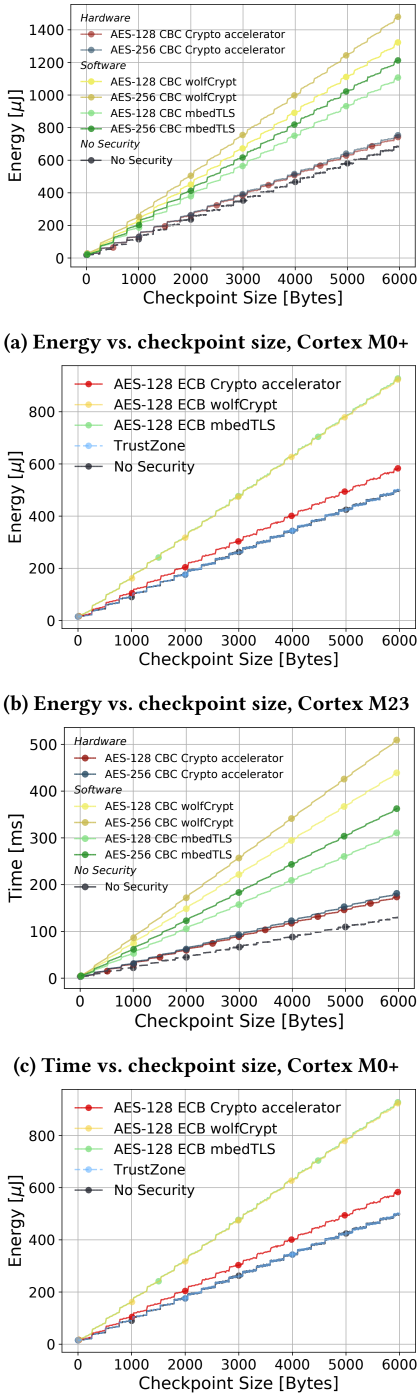
(a) 60μF capacitor, Cortex M0+

(b) 60μF capacitor, Cortex-M23

(c) 220μF capacitor, Cortex M0+

(d) 220μF capacitor, Cortex M-23

Figure 1: Breakdown of energy consumption.



(a) **Energy vs. checkpoint size, Cortex M0+**

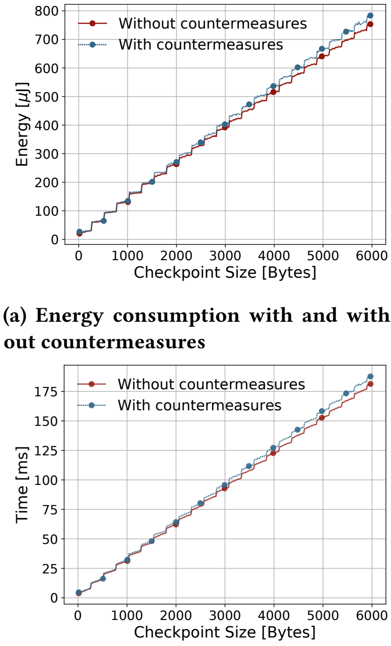
(b) **Energy vs. checkpoint size, Cortex M23**

(c) **Time vs. checkpoint size, Cortex M0+**

(d) **Time vs. checkpoint size, Cortex M23**

Figure 2: Energy consumption and execution time.

gains of a crypto accelerator are now even larger than for energy consumption, compared to software implementations. In contrast, using the Cortex M23, the software implementations perform differently in time, despite the comparable performance in energy, as seen in Fig. 2b. TrustZone, on the other hand, shows limited overhead in time as well, and becomes the best performing option in this case too, whenever provided by the underlying platform and within the constraints of the corresponding threat model.



(a) **Energy consumption with and without countermeasures**

(b) **Execution time with and without countermeasures**

Figure 3: Energy and time with countermeasures against side-channel attacks, using the crypto accelerator onboard the Cortex M0+ and AES-256 CBC mode.

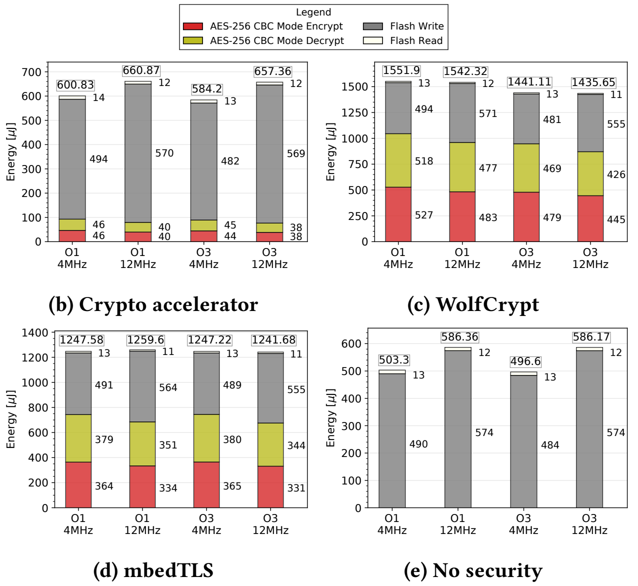
Contributing factors. A closer look reveals that the analysis of performance and trends in Fig. 2, and thus the results of Fig. 1, must carefully consider aspects such as compiler configuration, MCU settings, and level of protection provided.

For example, we have no evidence that mbedTLS uses specific techniques against time-based side-channel attacks, whereas WolfCrypt explicitly employs as close to constant time code as possible. Such a technique necessarily imposes an overhead, which might justify the worse performance in energy and time of WolfCrypt compared to mbedTLS for the Cortex M0+.

The crypto accelerator onboard the Cortex M0+ may also work with specific countermeasures against side-channel attacks, which are disabled in the experiments of Fig. 2. To investigate this aspect, Fig. 3 exemplifies the impact of enabling those features on energy and time overhead. The plot shows how performance is only marginally affected. Therefore, the observations we draw from Fig. 2 remain largely valid.

Fig. 4 and 5 show how varying compiler configuration and MCU settings possibly impact energy consumption and execution time on the Cortex M0+, using either of the software implementation of AES, the onboard crypto accelerator, or no security mechanism.

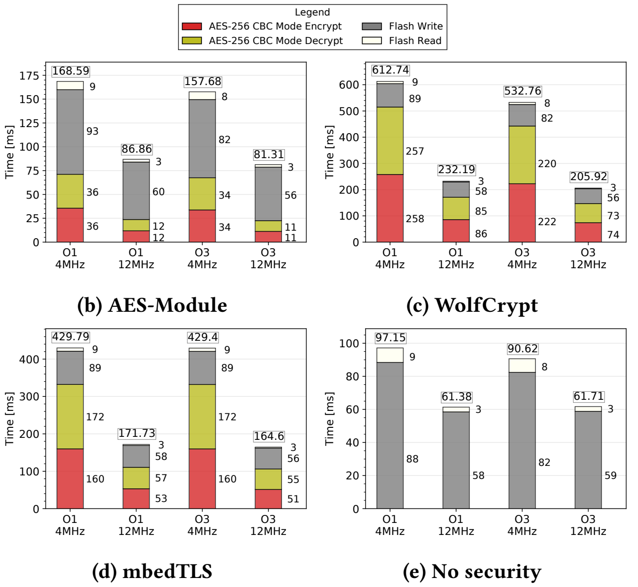
Fig. 4 shows that using the crypto accelerator, the energy cost is dominated by flash operations, as it is also the case when using



(b) Crypto accelerator **(c) WolfCrypt**

(d) mbedTLS **(e) No security**

Figure 4: Energy consumption using different compiler configurations and at different clock speeds; checkpoint size is 4 Kbytes secured with AES-256 CBC mode.



(b) AES-Module **(c) WolfCrypt**

(d) mbedTLS **(e) No security**

Figure 5: Execution time using different compiler configurations and at different clock speeds; checkpoint size is 4 Kbytes secured with AES-256 CBC mode.

no security mechanism. This observation applies regardless of the compiler configuration and MCU setting. In contrast, when using software implementations of AES, the overall energy consumption is evenly split between running the encryption algorithm and flash operations, and again remains largely the same regardless of compiler configuration and MCU setting.

Fig. 5 offers a complementary view, where the impact of running the MCU at higher clock speeds becomes apparent, and yet causes no additional energy consumption as discussed earlier. The performance is almost unaltered depending on compiler configuration, which we ultimately conclude to bear no significant impact.

5 CONCLUSION AND OUTLOOK

We presented experimental results on applying different security mechanisms to protect persistent state in intermittent computing, including software- and hardware-based implementations of AES encryption and ARM TrustZone. Using modern MCUs of the Cortex M* series and staple intermittent computing benchmarks, we found that the modularity and ease of update of software implementations comes at the cost of a marked energy and time overhead, which possibly prevents forward progress. The hardware-based implementations abate energy and time overhead, but require care in understanding their internals and their potential influence on performance. TrustZone, which is right now available on only a few platforms, shows almost negligible overhead, at the cost of a different memory management to implement in application code.

Our work lays the basis to design more efficient means to protect persistent state, especially for platforms that do not support TrustZone. Using software-based security mechanisms, for example, the encryption mode and hence the related overhead may be tuned depending on energy patterns in ways to retain forward progress whenever possible. Segmenting the checkpoint and only encrypting the most sensitive parts may also help reduce the overhead, at the cost of increased complexity in checkpoint and restore operations.

6 ACKNOWLEDGMENTS

This work was supported by the Swedish Foundation for Strategic Research (SSF) through the aSSIsT project.

REFERENCES

[1] 2020. Benchmarking code for Securing Persistent State in Intermittent Computing. https://github.com/areebasad/Benchmarking-for-Securing-Persistent-State-in-Intermittent-Computing. (Accessed on 10/18/2020).

[2] Saad Ahmed, Naveed Anwar Bhatti, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2019. Efficient intermittent computing with differential checkpointing. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. 70–81.

[3] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini. 2016. Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2016).

[4] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini. 2015. Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems. *IEEE Embedded Systems Letters* (2015).

[5] Mihir Bellare, Phillip Rogaway, and David A Wagner. 2003. EAX: A Conventional Authenticated-Encryption Mode. *IACR Cryptol. ePrint Arch.* 2003 (2003), 69.

[6] N. A. Bhatti, M. H. Alizai, A. A. Syed, and L. Mottola. 2016. Energy Harvesting and Wireless Transfer in Sensor Network Applications: Concepts and Experiences. *ACM Transactions on Sensor Networks* (2016).

[7] N. A. Bhatti and L. Mottola. 2017. HarvOS: Efficient Code Instrumentation for Transiently-powered Embedded Sensing. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN).*

[8] Julia Borghoff et al. 2012. PRINCE–a low-latency block cipher for pervasive computing applications. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer.

[9] A. Colin and B. Lucia. 2016. Chain: Tasks and Channels for Reliable Intermittent Programs. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA).*

[10] Zahra Ghodsi, Siddharth Garg, and Ramesh Karri. 2017. Optimal checkpointing for secure intermittently-powered IoT devices. In *2017 IEEE/ACM International*

Conference on Computer-Aided Design (ICCAD). IEEE, 376–383.

[11] J. Hester and J. Sorber. 2017. The Future of Sensing is Batteryless, Intermittent, and Awesome. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SENSYS)*.

[12] J. Hester, K. Storer, and J. Sorber. 2017. Timely Execution on Intermittently Powered Batteryless Sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SENSYS)*.

[13] H. Jayakumar, A. Raha, W. S. Lee, and V. Raghunathan. 2015. QuickRecall: A HW/SW Approach for Computing Across Power Cycles in Transiently Powered Computers. *ACM Journal on Emerging Technologies in Computing Systems* (2015).

[14] Archanaa S Krishnan, Charles Suslowicz, Daniel Dinu, and Patrick Schaumont. 2019. Secure intermittent computing protocol: Protecting state across power loss. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[15] K. Maeng, A. Colin, and B. Lucia. 2017. Alpaca: Intermittent Execution Without Checkpoints. *Proceedings of the ACM Programming Languages* (2017).

[16] B. Ransford, J. Sorber, and K. Fu. 2011. Mementos: System Support for Long-running Computation on RFID-scale Devices. *ACM SIGARCH Computer Architecture News* (2011).

[17] Archanaa Santhana Krishnan and Patrick Schaumont. 2018. *Exploiting Security Vulnerabilities in Intermittent Computing: 8th SPACE International Conference*. 104–124.

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

[19] Emanuele Valea, Mathieu Da Silva, Giorgio Di Natale, Marie-Lise Flottes, Sophie Dupuis, and Bruno Rouzeyre. 2018. SI ECCS: SECure context saving for IoT devices. In *2018 13th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS)*. IEEE, 1–2.

[20] J. Van Der Woude and M. Hicks. 2016. Intermittent Computation Without Hardware Support or Programmer Intervention. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*.

[21] K. S. Yildirim, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester. 2018. InK: Reactive Kernel for Tiny Batteryless Sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems (SENSYS)*.