2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-7652-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/HPCA56546.2023.10071082

SecPB: Architectures for Secure Non-Volatile Memory with Battery-Backed Persist Buffers

Alexander Freij
North Carolina State University
North Carolina, USA
atfreij@ncsu.edu

Huiyang Zhou
North Carolina State University
North Carolina, USA
hzhou@ncsu.edu

Yan Solihin
University of Central Florida
Florida, USA
yan.solihin@ucf.edu

***Abstract*—The durability of data stored in persistent memory (PM) exposes data to potentially data leakage attacks. Recent research has identified the requirements for crash recoverable secure PM, but do not consider recent trends of the persistency domain extending on-chip to include cache hierarchies. In this paper, we explore this design space and identify performance and energy optimization opportunities.**

We propose *secure persistent buffers (SecPB)*, a battery-backed persistent structure that moves the point of secure data persistency from the memory controller closer to the core. We revisit the fundamentals of how data in PM is secured and show how various subsets of security metadata can be generated lazily while still guaranteeing crash recoverability and integrity verification. We analyze the metadata dependency chain required in securing PM and expose optimization opportunities that allow for SecPB to reduce performance overheads by up to 32.8×, with average performance overheads as low as 1.3% observed for reasonable battery capacities.

I. Introduction

Persistent memory (PM) or non-volatile memory (NVM) is increasingly integrated into modern systems, with technologies such as Intel Pmem[1] [25] being widely available. With its high density and byte addressability, PM is being adopted to augment or substitute DRAM as main memory. PM's non-volatility allows data to remain across system boots and power cycles, exposing it to potential data leakage or unauthorized modifications. Recognizing these threats, proposals for memory encryption and integrity protection for persistent data in PM have been proposed [5]–[7], [18], [19], [23], [68], [74].

A key requirement for hosting persistent data on secure memory is the crash consistency between data and its metadata, which determines whether post-crash recovery yields correct data plaintext and/or whether integrity verification succeeds [18]. It has been pointed out that the *memory tuple*, consisting of data, counter, message authentication code (MAC), and Merkle Tree root update, must be updated and persisted atomically in the order specified by the persistency model. This previous work, along with others [5]–[7], [19], [23], [68], [74], assume that the point of persistency is at the memory controller.

Recently, there have been significant interests in expanding the persistency domain to the entire memory hierarchy. We refer to it as the *persistent hierarchy* approach. With persistent hierarchy, strict persistency is achieved automatically, cache line flushes are no longer needed for achieving persistence, and fences are only needed for persistency models that do not guarantee total store ordering. The elimination of flushes and fences not only improves persistency performance substantially, but it also simplifies programming and reduces chances of bugs. Persistent hierarchy can be accomplished using *non-volatile caches* [30], [44], [62], Intel *eADR* [51], or *battery-backed buffer* (*BBB*) [4]. Non-volatile caches rely on using technologies such as STT-RAM [31], PCM [42], or ReRAM [29], for caches. eADR relies on battery or supercapacitors to back the SRAM caches such that on power loss, cache content is flushed to the PM. BBB introduced a battery-backed SRAM *persist buffer* (PB) at the same level as the L1D cache, which is drained to PM on power loss. Due to PB's small size relative to caches, BBB requires a much smaller battery than eADR [4].

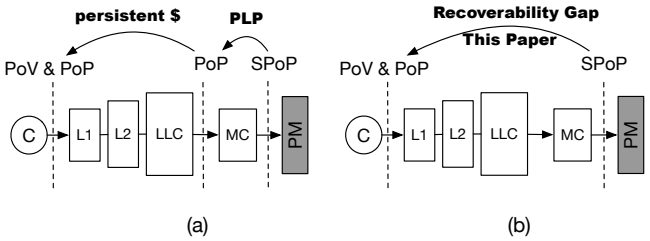

Fig. 1. Illustrating the context of prior works (a) and the goal of this paper (b), which covers a new *recoverability gap* problem.

Figure 1(a) shows the context of the works discussed above. It shows a core, three levels of caches, the memory controller with a write pending queue (WPQ), and PM. *Point of Visibility* (PoV) is the point in which a store becomes visible to other cores. *Point of Persistency* (PoP) is the point in which a store becomes persistent. *Security Point of Persistency* (SPoP) is the point in which changes to security metadata become persistent. The figure illustrates a traditional system, with PoV, PoP, and SPoP, differing from one another. Authors of PLP [18] pointed out that the memory tuple has to be atomically updated, hence the SPoP is moved up to match the PoP. On the other hand, persistent hierarchy moves up PoP to match the PoV to close the visibility and persistency gap.

[1]Renamed from Optane DC persistent memory.

978-1-6654-7652-2/23/$31.00 ©2023 IEEE

While improving performance and programmability, persistent hierarchy creates a *new recoverability gap* as illustrated in Figure 1(b). The figure assumes the state-of-the-art BBB approach that only adds PB to the persistency domain. As PoP is moved up to near the core, a new gap between PoP and SPoP is created. This gap affects crash recoverability of data since data and its security metadata no longer get atomically persisted. With the gap, post-crash recovery may yield wrong plaintext or fail integrity verification. Hence, the goal of this paper is to remove the gap by moving the SPoP up to match the PoP.

Closing PoP/SPoP gap should not be done naively as it will substantially slow down persistence. For example, if every time a store enters the PB, security metadata is updated, i.e. counter incremented, MAC calculated, BMT root updated from leaf to root, then it becomes the new performance bottleneck. Not only the latency of metadata persistence slows down data persistence, but the volume of metadata persistence increases substantially due to the lack of filtering by caches.

In this paper, we propose secure persistent buffers (SecPB) and explore designs and optimizations that remove the performance bottleneck. The optimizations are enabled by a key observation: the crash recovery observer does not need to see the cache/memory state at the time of a crash, only at the time of recovery. Thus, there exists a time gap between a crash and recovery where data and metadata can be made consistent. This observation creates a performance/battery cost trade off spectrum, where we can use an early strategy (update all metadata as soon as data is persisted by a store), to a late strategy (update all metadata only after a crash occurs). We explore different design points along this spectrum and report our findings and insights.

To summarize, the contributions of this work are:

- We propose to close the PoP/SPoP gap by using persistent on-chip *secure persistent buffers* (SecPB) approach and specify the architectural requirements to secure PM with on-chip persistent hierarchy.
- We explore a design spectrum consisting of six secure persistency schemes that expose the performance/battery cost trade-offs for securing persistent hierarchies.
- We evaluate the performance of various designs and the battery capacity required for each design, and discuss our key findings.

The rest of the paper is organized as follows: Section II presents the background and related work. Section III details the security challenges and opportunities exposed with secure persistent hierarchy implementations. Section IV discusses the different secure hierarchy schemes and details the validity of the schemes and the hardware architecture design. Section V presents our evaluation methodology. Section VI evaluates our proposed schemes, and Section VII concludes our work.

II. Background and Related Work

A. Threat Model

Similar to prior work, we assume that adversaries have physical access to the system [12], [43], [52], [57], [73] to perform attacks on the memory system, such as snooping [60], tampering with data stored in PM [32], [55], or replay attacks [9], [66], [74], [75]. Non-volatility of data is a much stronger concern in PM than in DRAM-based systems as well [21], [22], [38], as data can remain in memory for long periods of time without power. Similar to prior work [7], [33], [35], [74], we assume that attackers cannot directly access on-chip resources such as caches and registers, therefore establishing the physical boundary of the CPU as the trusted computing base (TCB). Specifically, we assume that data moved off chip needs to be protected, while data on chip can be read and updated in plaintext.

B. Memory Encryption & Integrity Verification

Memory encryption provides data confidentiality for values written in off-chip memory [8], [26], [28], [56], [69] or sent to other processing units within the system [70]. Various implementations, such as XTS [27] and counter mode encryption [67] have been proposed and utilized in commercial products. Counter mode encryption involves generating a one-time pad (OTP) by encrypting a nonce counter and XOR'ing the OTP with the plaintext value to create the ciphertext [13], [15], [68], [76]. Since encrypted values may be susceptible to tampering, integrity verification is required to ensure splicing or spoofing attacks are detectable [59]. Furthermore, data, counter, and MAC can be rolled back to their older versions, hence the freshness of at least one of them has to be ensured, typically by relying on an integrity tree. Data fetched from off-chip memory must have its integrity verified when brought onto the TCB [47], [48]. Integrity verification is commonly provided by message authentication codes (MACs) and integrity trees, such as Bonsai Merkle Trees (BMT) [46], TEC trees [17], [61], Merkle Trees [20], or SGX Counter Trees [5], [15]. In this work, we use split counter-mode encryption [65] and BMT & MAC for integrity verification.

C. Store Ordering for Persistent Hierarchies

Memory persistency models have been defined in prior research to provide a store ordering framework to reason the order in which data written to PM becomes durable with respect to other stores [3], [11], [14], [16], [40]. These models allow for programmers to reason about the correctness of the persistent memory state with respect to a crash recovery observer. The most conservative, intuitive, but slow is *strict persistency* (SP) which requires persistency order to follow the program order. Other more relaxed persistency models, such as epoch persistency or buffered epoch persistency [36], [49], [53], [54], [71] allow for a more relaxed persistency model where stores within an epoch may persist in any order, but are ordered across epochs. SP is often considered as too performance restrictive. However, persistent hierarchy is a game changer for SP as a store persists instantly as it enters the L1D cache or persist buffer. Therefore, in this work, we focus on optimizing for a SP model.

D. Related Work

Prior works in PM either do not consider persistent hierarchy or do not consider security. In the secure PM category, many works investigated how data should be updated in a crash consistent manner. Prior research pointed out that data and its counter [9], [76] or data and MAC [50], [58] need to be persisted atomically. Freij et al. [18] formally discussed invariants required for correct crash recoverability of data. The invariants require that a memory tuple, consisting of data, counter, MAC, and BMT root, to be updated atomically in the order that the persistency model specifies for data. The work identified the BMT root update as a key performance bottleneck. Han et. al [23] proposed methods to reduce the performance overheads by securing data in the write pending queue (WPQ) that resides in the MC, while other research [19] utilized small, on-chip non-volatile caches to reduce the height of the integrity tree. None of the works discussed above consider a system with persistent hierarchy, which is the focus of this paper.

In the persistent hierarchy category, prior works discussed expanding the on-chip persistency domain to the entire cache hierarchy, which reduces programming complexity and performance overheads of persistency models, especially SP. Intel eADR [51] adds all caches into the persistent domain, thus allowing all stores to persist as soon as they access the caches. Alshboul et. al [4] proposed battery-backed persistent buffers (PB) to provide a low-cost method for achieving persistent hierarchy with a much smaller battery. When applied to a system with memory encryption and integrity verification, persistent hierarchy creates a new recoverability gap. Closing this new gap efficiently is the focus of this paper.

Finally, some recent research has analyzed adding security to on-chip PM [21]. Rathi et. al [45] proposed architectures to obfuscate or erase data stored in on-chip non-volatile cache. These works differ from ours in their threat model and goal, and did not address persistency and crash recovery.

III. SecPB Design Space

In this section, we will discuss the design rationale and design space of SecPB, our proposed architecture that ensures secure persistent memory where SPoP is aligned with PoP, allowing nearly-negligible strict persistency overheads while guaranteeing crash consistency of persistent system state.

Before commencing on the discussion, we will limit the scope of the discussion. In the persistent hierarchy approach, the PoP and PoV are aligned right after the core, at the L1D level. Because non-volatile caches are not mainstream yet, and eADR requires a large battery/supercapacitor, we limit our discussion to the battery-backed buffer (BBB) technique [4]. However, our approach is adaptable to other techniques as well.

Figure 2(a) illustrates the BBB system, with non-volatile or battery-backed components shown in grey. BBB adds a persist buffer (PB) to each core. Each PB entry stores a block of data. When a core performs a store, the store is exposed to the L1D cache and PB simultaneously. If the block is found in the PB, the store updates the appropriate byte/word. Otherwise, the block is fetched and allocated in the PB and the store updates the affected byte/word in the block. Multiple stores to the same word or different words in the same block are coalesced at the PB. PB is drained when it reaches a high watermark, until sufficient entries have been drained to reach a low watermark. If a crash is detected, all PB entries are drained to PM. PB is considered memory-side, hence as soon as a block is updated by a store in PB, it can be considered to have updated the PM and persisted. Cache replacement and coherence protocol are modified to ensure coherence between multiple PBs in multiple cores [4].

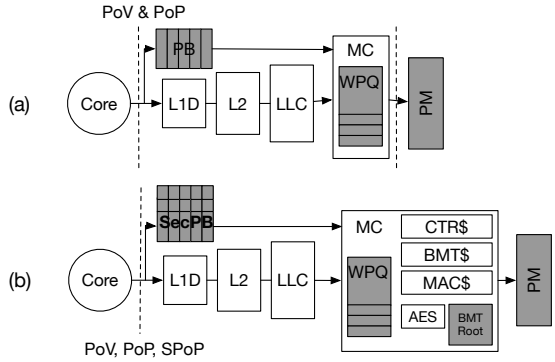

Fig. 2. Redesigning BBB (a) to achieve secure persistent hierarchy with our secure persist buffer or SecPB (b).

When SPoP is not aligned with PoP, i.e., it is at the memory controller, the result is a *recovery gap*. Recovery gap prevents crash consistency between data and security metadata because they may be persisted at different times and may be reordered with respect to other data persists. Therefore, our goal is to align SPoP with PoP. Figure 2(b) illustrates our system, which aligns SPoP with PoP. With the alignment, as soon as a store enters the persist buffer, which makes it persistent, its security metadata is updated and persisted. The system has metadata caches in the memory controller (MC), e.g. counter cache, BMT cache, and MAC cache, which may be physically separate or unified. The BMT root is kept in a non-volatile register in MC. The SecPB is modified from PB by adding multiple new fields (more details later) in each entry. SecPB's controller co-ordinates the persistence of security metadata with respect to data, to ensure crash recovery correctness.

A. Crash Recovery Correctness Requirements

The primary obstacle to aligning SPoP with PoP is satisfying the two correctness invariants required by crash recovery of secure persistent memory, specified in PLP [18]. We will briefly summarize them to provide context for our subsequent discussion. The PLP paper defines a *memory tuple* as consisting of data and all security metadata. For split counter and BMT schemes, the tuple consists of (C, γ, M, R) where C representing data ciphertext, γ the counter for the block, M the MAC, and R BMT root. Two invariants are required for correct crash recoverability of a secure persistent memory.

The first invariant requires that the entire tuple updated and persisted in order to consider data to be persisted. That is, counter must have been incremented (and persisted), MAC must have been recomputed (and persisted), and BMT root updated (and persisted), for data to be considered to have persisted. If any of the tuple is not updated and persisted, crash recovery may fail, either the correct plaintext is not recovered and/or integrity verification fails during recovery.

The second invariant (*persist order invariant*) deals with the persist ordering as defined by the persistency model. If the persistency model specifies that two stores α_1 and α_2 are ordered with respect to persistency, i.e. $\alpha_1 \rightarrow \alpha_2$, then all their memory tuple components must be persisted in the same order, i.e. $(C_1, \gamma_1, M_1, R_1) \rightarrow (C_2, \gamma_2, M_2, R_2)$. If the persist ordering of any component is not followed, crash recovery may not be able to recover to a consistent state between the two stores.

These two invariants make it very challenging for aligning SPoP with PoP. Consider two stores α_1 and α_2 that go out from the core's store buffer to the persist buffer. Without considering SPoP, they can persist instantly as they enter the persist buffer. However, with SPoP aligning with PoP, all security metadata relevant to the two stores must be updated and persisted as well, in the same order as the persisting of the two stores. To follow the two crash recovery invariants, this means that as α_1 is persisted, we must update and persist its counter, MAC, and BMT root, before we can persist α_2. This means that high-latency operations, especially BMT root update which may take hundreds of clock cycles, have now become a part of the critical path of data persist. Therefore, a naive SecPB design incurs a very formidable new performance bottleneck.

B. Opportunity for Optimizations

Building on BBB, our naive strategy would be to straightforwardly align SPoP with PoP, as illustrated in Figure 3a. The figure shows six stores sorted from oldest (St A) to youngest (St F). At this point, St A through St E have been allocated in the persist buffer (PB), hence they have reached PoV and PoP. Furthermore, all memory tuples, including counter, MAC, and BMT root, have been updated, hence SPoP has also been reached. The PB is about to drain block A to the PM. We refer to the gap between stores that have been drained and PoV/PoP as the *draining gap*. It is this gap that has to be covered by the battery or supercap. Suppose now there is a crash, and the execution stops. The battery drains the PB and as a result, draining catches up to the PoV/PoP, as illustrated in Figure 3b. The state seen by the crash observer includes the persistent memory values that result from St A through St E. We refer to this strategy as the early strategy, where SPoP is achieved instantly without any delay as each store enters the PB. We have discussed that the strategy may be prohibitively expensive, but draining is as simple as with a regular PB.

We make an observation that in this architecture, the crash observer is only allowed to see the persistent state as defined by the PoV/PoP, and not as defined by the execution status of stores or the draining status of stores from PB. In Figure 3a, stores that have reached the PB are considered to have persisted and their effect is reflected in the persistent state, while stores that have not reached the PB are not considered to have persisted hence their effect on the persistent state should not be visible to the crash observer. With this observation, if the battery is enlarged, we may be able to let SPoP run behind the PoV/PoP, and even behind draining. We refer to this as *late* strategy, which is the foundation of our approach. The late strategy is illustrated in Figure 3c. In addition to the draining gap, the late strategy has another gap that we refer to as security synchronization (*sec-sync*) gap, which is the gap between draining and security metadata update and persistence. In the figure, blocks D and E have not been drained, but blocks A, B, and C have been drained. As blocks A, B, and C are drained, they reach the memory controller (MC), at which time the memory tuple is updated, e.g. the blocks are encrypted, their counters incremented, MACs generated, and BMT updated from their leaves to the root. Therefore, SPoP is lagging behind draining, creating the sec-sync gap. When a crash occurs, now the battery needs to cover both the draining gap as well as the sec-sync gap. It now takes more time until draining of PB is completed, and any blocks drained to have their memory tuples updated and persisted. After sufficient time, though, the resulting observable state is the same as before (Figure 3b).

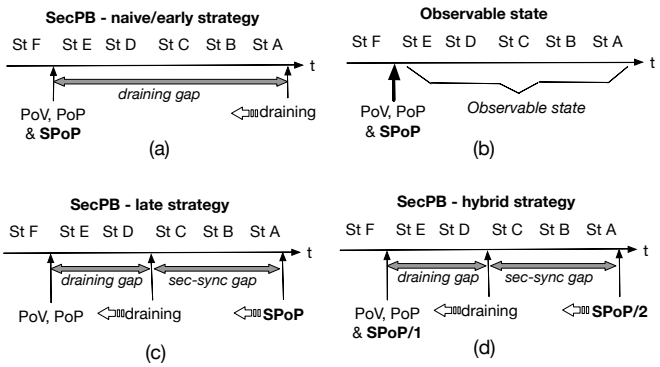

Fig. 3. Contrasting the early, late, and hybrid security metadata persistence strategies.

While the gaps are being closed, one question is what the crash observer should be allowed to see. Since the persistent memory state is not consistent until the gaps are closed, the system must either employ a *blocking policy* (it prevents the crash observer to see the system state), or a *warning policy* (it warns the crash observer to wait until the persistent system state reaches crash consistency).

Both the blocking and warning policies assume that a crash can be detected, which applies to loss of power, hardware failures, or some system software failures. A natural question that arises is how an application crash should be handled. First, an application crash can also be detected if it incurs exceptions such as segmentation fault, divide by zero, single-stepped debugging, etc. Once detected, we can handle the

crash in two ways. The first choice is *drain-process* policy, where the PB only drains and sec-sync PB entries for data belonging to the application process. If another process runs in a different thread context in the same core, it may have entries in the same PB. Its PB entries are not drained. The drain-process policy requires the PB to be tagged with address space identifier (ASID) for each process, which increases the PB size and complexity slightly. A second choice is *drain-all* policy, where the PB drains and sec-sync all entries regardless of which process the entries belong to. This policy may unnecessarily drain and reduce coalescing opportunities for other processes, but removes the need to tag PB entries with ASID. Furthermore, the lack of behavior isolation may open up a side channel opportunity that the attacker can exploit. However, addressing side channel is beyond the scope of this work. Furthermore, a secure application should not run in the same core as a non-secure application to begin with, otherwise many other side channels exist beyond the PB. Considering that application crashes are rare events, we choose the drain-all policy. As with system crashes, to ensure that the crash observer only sees consistent state, we can handle the application crash in the same way, i.e. blocking or warning. Delaying observation is feasible in this case considering that PB is expected to have a small number of entries, e.g. 64 entries, and can be high- and low-watermark thresholds can be selected to provide an upperbound of the number of entries to drain.

One final observation that we make is that Figure 3a and Figure 3c show two opposite ends of the spectrum of early and late security metadata persistence. A question arises of what can be designed in between them. In other words, some security metadata update and persistence can be done early while others late. This *hybrid* strategy is illustrated in Figure 3d, where SPoP is split into the early part done prior to crash (SPoP/1) and the late part done post crash (SPoP/2). The implication for the hybrid design is that it can occupy the space between early/late in terms of critical path delay to data persistence and battery size, allowing us to fine tune the design in the trade off space. The more work done prior to crash, the higher the performance overhead during normal execution but the smaller the battery needs to be.

IV. SecPB Architecture Design

In Section III, we discussed the eager, late, and hybrid strategies for updating and persisting security metadata, which define the design space that we can explore. In this section, we identify the design points that exist in the space, and analyze each point's implications on the performance and energy requirements.

Figure 4 shows the steps for security metadata update for a given store, showing both event trigger and data dependence relations. After a store is committed, it is released into the L1D cache and PB in the order specified by the memory consistency model. This event triggers a counter to be incremented (and persisted in the SecPB). Afterward, the counter is used to generate a one-time pad (OTP), and updates the BMT from

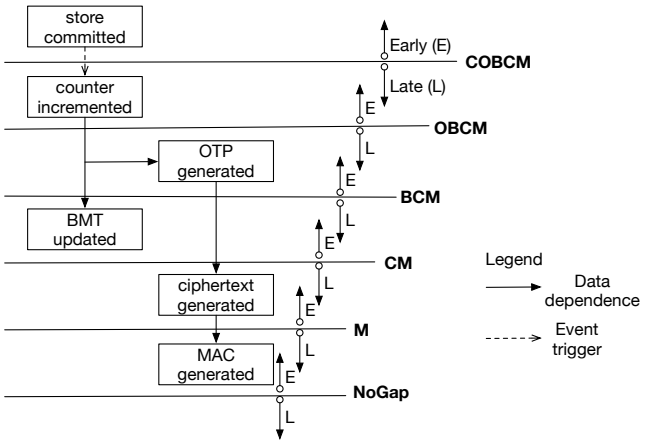

Fig. 4. Security metadata dependency graph within a secure environment which demonstrates the design space for SecPB showing early (NoGap), late (COBCM), and hybrid strategies (all others).

leaf to root. The OTP is XOR'ed to generate the ciphertext, which is used to generate MAC. Any combination of these steps can be performed early (at store persist time) or late (post crash).

The most eager design is *NoGap*, which eliminates the sec-sync gap entirely by updating all security metadata and persisting them early. NoGap adheres with the persist order invariant constraint strictly. Other schemes are represented by letters indicating which components are performed at post-crash time, so the longer the name, the more it performs late. The next one up is *M* design, which performs everything early at store persist time, except for computing the MAC. This improves over NoGap in two ways. First, it reduces the time to persist a store by the latency of MAC generation. However, the reduction may not be much if the latency is already hidden. Second, it reduces work (and hence power consumption) by generating a MAC once for a dirty block. For example, if a word is written multiple times or multiple words in a block are written by different stores, while the block resides in PB, the MAC is only generated once when the block is drained. In contrast, with NoGap, each store requires the MAC to be generated while the block is in SecPB.

Design *CM* performs the actual encryption and MAC generation at post crash time. Note that in counter mode encryption, ciphertext generation is performed through bitwise XORing the plaintext and the OTP, hence it only takes one clock cycle. Thus, we expect CM to perform similarly to M in terms of performance. However, it still benefits from less work due to avoiding ciphertext generation until the block is drained from PB.

The next model is *BCM*. It moves BMT root update to post crash. Updating the BMT root from leaf incurs high latency commensurate with the number of tree levels, even if accessing each level in the tree hits in the metadata cache. Moving it to the post-crash time would substantially remove the bottleneck for data persist, while the benefit of coalescing reduces the total number of BMT root updates, and avoids

collisions between two stores updating common ancestors in the BMT. We expect BCM to substantially outperform CM.

The next model is *OBCM*, which adds OTP generation into the post-crash time. Due to its latency lower than BMT update and that it can be performed in parallel with BMT update, we do not expect much performance benefit from it, but we expect a reduction in the number of times OTP is generated for each block.

The final model is *COBCM*, which delays everything, even counter increment, to the post crash time. Over OBCM, we expect COBCM to be superior in eliminating all runtime metadata performance overheads from the critical path.

All the designs above incur trade offs in performance overheads, coalescing opportunities for security metadata update, energy, and battery cost requirements. This requires detailed evaluation that we will present in Section VI.

A. Optimization for NoGap, M, and CM

We note that NoGap, M, and CM, all recalculate security metadata as soon as a store enters the PB, hence they perform many security metadata generations for a single block due to multiple stores to it. The higher the degree of temporal and within-block spatial locality, the worse NoGap, M, and CM perform with respect to lazier schemes. Here, we identify one optimization based on the different ways various security metadata types are computed.

For split counter and BMT scheme, security metadata can be divided into *data value dependent* ones (data ciphertext and MAC) that are computed based on the value of data plaintext, versus *data value independent* ones (counters, OTPs, and BMT) that are computed without relying on the value of data plaintext. Data value dependent metadata must reflect each change to the plaintext by a store, but this is not the case for data value independent metadata.

Consider that for counter-mode encryption, a counter only needs to be incremented when a data block is written back to memory to provide counter freshness. In our earlier description, suppose two stores α_1 and α_2 affect the same block and require persistency ordering, i.e. $\alpha_1 \rightarrow \alpha_2$, then all their memory tuple components must be persisted in the same order, i.e. $(C_1, \gamma_1, M_1, R_1) \rightarrow (C_2, \gamma_2, M_2, R_2)$. However, this is assuming that the crash recovery observer is allowed to see the persistent state between the two stores. If, however, both stores are accepted by the SecPB, the crash recovery observer is only permitted to see the state after the second store, and not between the two stores. Hence, work related to updating and persisting security metadata can be combined between the two stores. Note that this is also the case when there are writes to other blocks between these two stores, e.g., St A (counter & BMT update) $\rightarrow$ St B (counter & BMT update) $\rightarrow$ St A (no update), as the observer only sees the state after SecPB is drained.

In particular, the counter for the block can be incremented once (instead of once per store) when the block first becomes dirty. That is, on the first store (α_1), we could increment its counter in the SecPB, but not increment it further in subsequent stores (including the second store α_2). Furthermore, since OTP and BMT root are updated only when the counter is incremented, this observation can extend to them as well, hence the OTP can be generated once and BMT root updated once. This optimization is especially beneficial for the high-latency BMT root update operation. Furthermore, this optimization avoids incrementing the counter frequently for a single dirty block frequently, delaying counter overflow which requires page re-encryption [46].

The optimization above can be applied to all the models except COBCM, which already performs all metadata updates late. But we expect it to be especially impactful for NoGap/M/CM, which without the optimization, would update BMT root often.

B. SecPB Architecture Design

SecPB architecture depends on which design we choose from the design space spectrum. They differ in the amount of information that need to be tracked; designs with late strategies require tracking less information, resulting in a simpler design.

Our baseline architecture assumes separate metadata caches (e.g. counter cache, BMT cache, and MAC cache), and Asynchronous DRAM Refresh (ADR) write pending queue (WPQ) [34]. This architecture enables baseline strict persistency (SP) model discussed in [18]. Our architecture assumes that SecPB as the sole battery-backed structure in the memory hierarchy.

Figure 5 shows the SecPB architecture. SecPB connects directly to the core, accepting stores released from the core store buffer, and to the memory controller, which contains the cryptographic engine that performs encryption, MAC computation, integrity verification, and caches security metadata. The SecPB has a controller that relies on a finite state machine (FSM) to track the status of security metadata persistence, and a table where data and security metadata are tracked for persistence purpose. Each entry contains the plaintext of data block (Dp, 64B). Many things need to be tracked, hence each entry further contains the pre-computed OTP (O, 64B), data ciphertext (Dc, 64B), counter (C, 8 bits), BMT root update acknowledgement (B, 1 bit), and MAC (M, 512 bits). Each of the fields has a valid bit to indicate whether the field contains valid updated value or not, except for the BMT field which only indicates whether BMT root (kept in the MC) has been updated. Different designs need to track different information, hence not all the fields are needed for all designs. The table in the figure shows which fields are kept for different designs.

The mechanism works as follows. After a store retires, it accesses the L1D cache ❶. In parallel, the store also accesses the SecPB ❷. If both hit, the store updates the value in both the SecPB and L1D cache. If both miss, the block is fetched and allocated in both the PB and L1D cache. If one hits and the other misses, the block is copied to the other, and then the case is handled in the same way as both hit. If the block is found in the L1D without sufficient state to write (e.g. S in MESI), then the store stalls until invalidation is posted and acknowledgments from other cores are received.

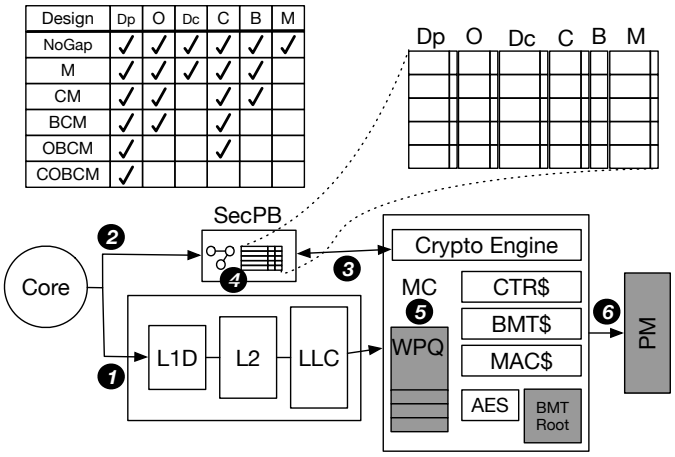

Fig. 5. Architectural design with per-core SecPB. The top left segment shows the SecPB fields which are populated eagerly for each store based on the scheme, and top right shows a detailed view of the SecPB. The bottom segment shows the dataflow of a persisted datum.

If the security metadata is not found, it is fetched from metadata caches in the MC ❸. Then, depending on the design, the block counter may be incremented, OTP, ciphertext, MAC, may be computed, and BMT root may be updated ❹. Valid bits are used to track which metadata has been updated in the PB. When all valid bits are set, then the persist of the security metadata is considered complete, after which the entry is drainable. For the NoGap design, at the completion of persist, the PB issues an unblocking signal to the store buffer to indicate that it can then accept a new store. For COBCM, as soon as a store updates a plaintext block in the PB, the PB can accept a new store. For other designs, the unblocking signal is raised after any "early" memory tuples have been updated. If an entry is drained from the PB, either due to eviction or due to a crash, the MC completes the update of all memory tuples in the metadata caches ❺. Then, they are flushed to the PM ❻.

C. Cache Coherence and Inclusion

a) Data and Metadata Cache Inclusion: Since dirty blocks are guaranteed to be drained from SecPB to reach the PM, they no longer need to be written back when evicted from the last level cache (LLC). Thus, a dirty block from the persistent memory region is allocated in the cache with a special dirty state; its eviction from the LLC is silently discarded similar to a clean cache block. Similarly, some designs allow security metadata to reside in two locations (SecPB and metadata caches). We handle this case similarly with data, a special state or flag indicates a dirty block in the metadata cache should be silently discarded if evicted. Furthermore, if a security metadata is drained from the SecPB, the block containing the metadata in the metadata cache is updated and or invalidated for coherence, ensuring that a future metadata cache miss will fetch the updated metadata value.

b) Memory Consistency: If strict persistency is enforced for relaxed memory consistency models, the core store buffer also needs to be battery backed, as encountered in [4]. Such a case may arise if the visibility order of the stores to other threads is not important, but the persist order of stores is important for crash consistency. In this case, stores may reach the SecPB out of the program order due to relaxed memory consistency models. The program order of stores is only kept in the store buffer. The implication is unique to our SecPB in that security metadata needs to be updated in the SecPB in program order. A simpler solution, however, is to use a lazier approach, such as the COBCM design. In that case, security metadata update can be performed out of the program order of stores, removing the need to coordinate with the store buffer.

c) Cache Coherence Issues: Metadata caches are usually not involved in coherence because they are beyond the caches and exist on the memory side, where no replication across cores is allowed. However, with some of our early schemes, metadata may also be kept in the SecPB, and each core has its own SecPB. Thus, care must be taken not to allow the replication of metadata across cores, otherwise coherence issues arise. To avoid that, the metadata caches are tagged with a directory that indicate which SecPB the metadata may also reside in. If there is a SecPB miss on another core, the entry is migrated from the current SecPB to the missing SecPB and the directory is updated.

Since SecPB writes occur in parallel with L1 data cache writes, data is located in both the data cache and the SecPB. Therefore, there are two situations to consider for data coherence. First, if a requesting core issues a read request on an address that is located in another core's SecPB, the datum is sent from the owner's cache and set to a shared state. The SecPB entry would then be flushed to PM and the request is serviced in parallel, persisting both the most recent versions of the data and metadata while sending the latest data to the requesting core. Second, if the requesting core issues a write request, the SecPB entry would be migrated to the requesting core. The overheads of the write rely on the metadata generation scheme assumed. Eager schemes (M, NoGap) would incur overheads to generate the ciphertext and MAC but the migration incurs minimal additional security overheads. The reason is that the requesting core would not require a counter, OTP, or BMT root update as the data-independent security metadata has already been updated. When the migration is executed, the directory is updated to indicate where the entries reside. Therefore, migration avoids replication and avoids having to keep coherence state for blocks in SecPB.

V. Evaluation Methodology

A. Simulation Configuration

We evaluate our secure persistency schemes and the SecPB using a cycle-accurate simulation model with Gem5 [10] with the system parameters shown in Table I. The baseline and schemes evaluated are listed in Table II. We assume battery-backed SRAM for SecPB and the WPQ and that they are the only battery-backed persistent structures in the persistent hierarchy. The BMT root is securely stored and persisted in a special-purpose on-chip register and never leaves the TCB.

We assume speculative integrity verification in all our models, similar to [33]. In our simulation, we assume separate volatile metadata caches for counter, BMT nodes, and MACs.

We study the performance overheads and battery requirements of the design spectrum discussed in Section IV. To better understand the source of overheads, we conduct a study that varies the SecPB capacity to identify the impact on performance and the energy source. To explore the impact of the full BMT height on the SecPB schemes, we compare & augment the SecPB implementation with the state-of-the-art BMF [19] and analyze the source of performance overheads.

To evaluate the performance of our proposed schemes and SecPB, we use 18 representative benchmarks from SPEC2006 [24]. All models are fast-forwarded to representative regions and the next 250 million instructions are simulated.

TABLE I
SIMULATION CONFIGURATION

Processor	
CPU	1 core, OOO, x86_64, 4.00GHz
L1 Cache	64KB, 8-way, 64B block, access: 2 cycles
L2 Cache	512KB, 16-way, 64B block, access: 20 cycles
L3 Cache	4MB, 32-way, 64B block, access: 30 cycles
WPQ	32 entries
Volatile Metadata Caches	
Ctr Cache	128KB, 8-way, 64B block, access: 2 cycles
MAC Cache	128KB, 8-way, 64B block, access: 2 cycles
BMT Cache	128KB, 8-way, 64B block, access: 2 cycles
SecPB	
Size	{8,16,32,64,128,512} entries (default 32)
Entry size	260B
Access latency	2 cycles
Drain threshold:	75%
Security Mechanisms	
BMT	8 levels
MAC Latency	40 processor cycles [18], [33], [55]
NVM	
Memory	8 GB PCM, 1200MHz Write queue: 128 entries, Read queue: 64 entries Read: 55ns, Write: 150ns [18], [19], [34]

B. Methodology for Battery Capacity Estimation

To assess the battery capacity cost to support SecPB, we apply the values from Table III which were derived from [4]. We first estimate the draining cost of our various strategies and then estimate the battery capacity required to support all the strategies discussed thus far. To understand the energy overheads of our schemes, we compare our architecture to BBB and secure eADR (s_eADR), which we assume is an eADR-enabled system with data encryption and BMT integrity protection.

There needs to be enough energy provisioned to securely persist all dirty cacheblocks and SecPB entries in s_eADR and SecPB, respectively. Therefore, we conduct our analysis

TABLE II
EVALUATED SCHEMES

Name	Scheme
bbb (baseline)	Battery-backed buffer from [4] with no security mechanisms implemented
SP	SP scheme from [18] with SPoP in MC
COBCM	Only data write to SecPB
OBCM	Update counter
BCM	Update counter, OTP
CM	Update counter, OTP, BMT root
M	Update counter, OTP, BMT root, ciphertext
NoGap	Eagerly update all metadata

assuming that all cachelines in the hierarchy are dirty and a full SecPB and the remaining metadata needs to be generated and persisted in PM. This is important because missing any memory tuple update to even one dirty cacheline/entry may results in integrity verification failures during recovery. To ensure that all memory tuples are updated correctly, we assume the worst-case situation for each datum that needs to be persisted. For s_eADR and COBCM, the following assumptions are made:

1) All persisted data blocks are considered dirty and need their security metadata updated.
2) No two data blocks share a counter encryption page (EP) and all counter cache accesses miss.
3) There are no overlaps in BMT update paths between updates and all BMT cache accesses miss. Every update must fetch missing nodes from PM and compute the hash.
4) MACs are updated in the MAC cache during runtime. If a crash occurs, the MAC does not need to be fetched from PM but must be computed.
5) OTPs for ciphertext must be generated.
6) The ciphertext XOR and counter increment are single-cycle logical operations and the energy required is negligible.

We chose the smallest battery capacity that is capable of providing the energy required to follow all the assumptions listed. Assumptions (1) and (2) are very conservative, as the metadata caches (MDC) would contain a subset of the counters and BMT nodes. Data access patterns complicate estimating the probability of overlapping BMT updates, especially as the probability increases closer to the root. However, this represents an extreme scenario that needs to be considered. For both SecPb and s_eADR, we estimate that the energy required to move data from MC to PM is similar to moving data from PM to MC. This is We look into two battery technologies, super capacitors (SuperCap) [72], and lithium thin-film batteries (Li-Thin) [39], which have energy densities of 10^{-4}Wh and 10^{-2}Wh respectively.

[1], [2]

SecPB Energy Draining Estimation The draining cost of securely persisting SecPB entries depends on the strategy implemented. We compute the draining cost of SecPB entries and security mechanisms from Table III. The estimated cost

TABLE III
ESTIMATED ENERGY COSTS OF DATA MOVEMENT AND SECURITY METADATA GENERATION.

Data Movement Operation	Energy Cost / byte
Accessing data from SRAM	1pJ [37]
Moving data from SecPB to PM	11.839nJ
Moving data from L1D to PM	11.839nJ
Moving data from L2 to PM	11.228nJ
Moving data from L3 to PM	11.228nJ
Moving data from MC to PM	11.228nJ
Metadata Generation Operation	**Energy Cost / byte**
SHA-512 for BMT node computation	79.29nJ [63]
AES-192 for data encryption	30nJ [41]
SHA-512 for MAC computation	79.29nJ [63]

of accessing and draining data from the SecPB to the MC is assumed to be equal to the energy required to move data from the L1 cache to the MC. We also assume the energy to access cachelines in the metadata caches (MDC) to be equal to accessing data in the L3 cache. This is a conservative estimate since the MDC are smaller than the L3 cache, however, this assumption affects both the analysis for SecPB and s_eADR and so does not significantly impact our final analysis. For the NoGap, M, and CM schemes, we do not pipeline MAC or BMT root updates to more accurately study the overheads imposed by SecPB and the various schemes.

As more metadata is eagerly fetched and updated for each scheme, the draining energy cost is reduced. The energy required to support COBCM must be sufficient to move all SecPB entries (64B) to the WPQ, fetch and increment counters, generate the OTP, fetch and update BMT nodes, and generate the ciphertext and MAC. As metadata is eagerly generated, the draining cost is reduced. For example, for CM, we remove the energy costs incurred by the counter fetch, OTP generation, and BMT updates. The battery must be large enough to not only drain entries from the SecPB to the MC but also to complete the current SecPB write and metadata generation in the event a crash occurs during a pending update.

Secure eADR Energy Draining Estimation The draining energy required for s_eADR depends on the level of cache that a block needs to be drained from. As stated in assumption (1), all persisted data blocks must be considered dirty and have their security metadata updated. This means that the memory tuple for all cachelines in s_eADR's persistent hierarchy must have it's metadata generated according to the assumptions stated. This is a conservative estimate, as cache clusivity and store coalescing may reduce the total number of cachelines that need to be updated. Furthermore, we do not assume any watermarking scheme or optimized security metadata generation schemes for s_eADR, which could reduce the battery capacity as well.

VI. EVALUATION & RESULTS

A. Performance Summary

The overall performance results are shown in Table IV (more detailed results will be discussed later), which shows the slowdown ratios of our different models compared to the insecure BBB baseline. The performance overheads of each model coincided with the eagerness of security metadata updates for each write. The best performing model, COBCM, removes all security metadata updates from the critical write path and incurs an average overhead of nearly-negligible 1.3% compared to BBB without any security mechanisms. This is an impressive result as it means that crash consistency, memory encryption, and integrity verification, can all be achieved simultaneously with very small overheads. COBCM incurs the occasional backflow that stalls the core when SecPB is full. OBCM demonstrates a slightly higher overhead of 1.5%. It needs to fetch counters early and hence suffers from the occasional counter cache misses. BCM shows much higher overheads averaging 14.8%, as it adds OTP generation latency to each new SecPB entry allocation.

TABLE IV
PERFORMANCE OVERHEADS FOR ALL MODELS WITH 32-ENTRY SECPB.

Model	Slowdown(%)
COBCM	1.3%
OBCM	1.5%
BCM	14.8%
CM	71.3%
M	73.8%
NoGap	118.4%

The most significant performance difference is going from BCM to CM (71.3%), where BMT root update latency is exposed to the critical path of allocating a new SecPB entry for a new store (subsequent stores do not incur root updates). The next model *M* performs slightly worse, with average overheads of 73.8%. The NoGap model suffers the highest performance degradation with 118.4% average slowdown.

B. SecPB Performance

To understand the source of the performance overheads of our models, we measured the number of SecPB *persists per thousand instructions* (PPTI) and the average *number of writes per SecPB entry* (NWPE). We will now discuss them, followed by analyzing the impact of the data value independent coalescing optimizations on BMT updates compared to prior art.

Figure 6 shows per-benchmark performance overheads for all our models with a 32-entry SecPB, normalized to the insecure BBB baseline.

With NoGap, we observed a high correlation between the PPTI and SecPB write locality of workloads and the performance overheads. For example, *gamess* has a PPTI of 47.4 and NWPE of 2.1, which indicates that for every 2.1 SecPB writes, the 8-level BMT is updated from leaf to root ($8 \times 40 = 320$ cycles.) A MAC for each SecPB write would be generated, also consuming 40 compute cycles. The estimated IPC is $\frac{1000}{320 \times (47.4/2.1) + 40 \times 47.4} = 0.11$, which is very close to the actual IPC of 0.13, confirming our results. The actual IPC is higher because of the generation of several MACs is overlapped with BMT updates for new SecPB entries.

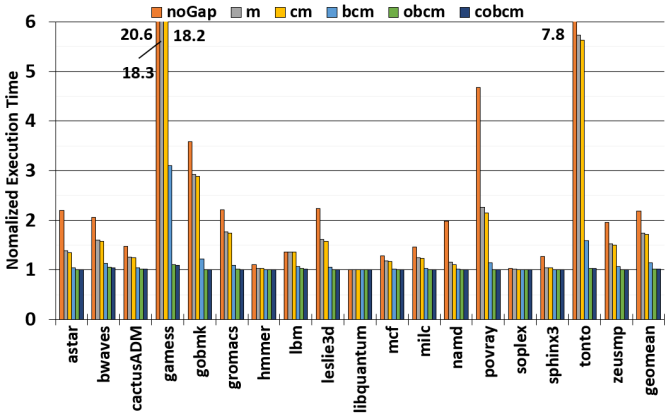

Fig. 6. Execution time of 32-entry SecPB normalized to BBB model.

By delaying MAC generation until a SecPB drain, the M model significantly improves the performance of memory-intensive workloads over *NoGap*: 20.6% overall reduction in execution time, but up to 51.6% for *povray* and 37.2% for *astar*. *povray* has very high PPTI of 38.8 and NWPE of 17.6, hence MAC generations are substantially cut down with the M model.

The CM model performs slightly better than the *M* model, dropping the average execution time by 2.5%. Delaying the ciphertext generation benefits more write-intensive workloads, such as *gamess* (12.2% reduction), due to all data value-dependent metadata being removed from the critical write path. With CM, the overheads observed stem from constraining the system to one in-flight BMT update. The data value independent metadata coalescing reduced the overheads and will be explored further later in this section.

The BCM model is the first model in the design spectrum that removes the BMT root update from the critical write path. We observed a massive performance improvement, with average execution time reduced by 56.5%. This is especially pronounced in *gamess*, where the overheads are reduced from $18.2\times$ to $3.1\times$. Similar to CM, NWPE now takes precedence in this model to reduce the total number of OTPs generated. The rate of reduction follows the BMT root update reduction seen in Figure 8.

OBCM removes the last cycle-intensive security metadata mechanism, OTP generation, from the critical path. Overall, the performance overhead observed is only 1.5% over the BBB baseline. The source of the overhead stems from the SecPB access latency being incurred twice for new SecPB entries before unblocking the L1D: one to write the data block and the second to check the counter valid bit.

The final model, COBCM, is the highest performing with an overall overhead of 1.3%. Overall, the performance of workloads tested nearly match the BBB performance. This is attributed to the high watermark mechanism draining SecPB entries to prevent stalling the L1D to drain entries. Some write-intensive benchmarks such as *gamess* suffer higher overheads, e.g., 9.6%. The write frequency and low spatial locality reduced the efficiency of the high watermark scheme.

To showcase the energy source requirements of SecPB and s_eADR, the last two columns in Table V report the ratio w.r.t. the footprint area of a client-class core [1], [2] (a $5.37mm^2$ footprint). We assume a cubic battery shape and deduce the footprint area from the volume. The areas need for s_eADR are substantial: assuming SuperCap technology, the battery required would be 4459% the size of a core, while a Li-Thin source would be 206.9% of the core area. In contrast, the battery source for *COBCM* requires only 53.6% of the core area for SuperCap and 2.5% for Li-Thin.

TABLE V
ESTIMATES OF THE SIZE OF THE ENERGY SOURCE NEEDED TO IMPLEMENT ALL MODELS WITH A 32-ENTRY SECPB COMPARED TO SECURE EADR. VALUES SHOWN ARE PER-CORE.

System	Size/Volume (mm^3)		Ratio to Core Area(%)	
	SuperCap	Li-Thin	SuperCap	Li-Thin
COBCM	4.89	0.049	53.6%	2.5%
OBCM	4.82	0.048	53.1%	2.5%
BCM	4.72	0.047	52.4%	2.4%
CM	0.73	0.007	15.1%	0.7%
M	0.67	0.006	14.2%	0.6%
NoGap	0.28	0.003	7.9%	0.4%
s_eADR	3,706.00	37.060	4459.6%	206.9%
BBB	0.07	0.001	3.16%	0.2%
eADR	149.32	1.490	524.1%	24.3%

C. Draining Cost Comparison

To determine the draining costs of our SecPB and secure eADR, we estimate two energy source types, SuperCap [72] and Li-Thin [64], and apply the analysis as discussed in Section V. Table V shows the estimates needed for the active energy source required to support SecPB across all the models discussed. We include energy results for BBB [4] and eADR without any security support to compare the additional battery capacity required to support SecPB. eADR requires a capacity of $149.32mm^3$ for SuperCap and $1.490mm^3$ for Li-Thin, $2500\times$ larger than the energy source required for BBB. We observe a similar trend when comparing secure eADR (s_eADR) to SecPB. s_eADR requires a battery source of $3{,}706mm^3$ for SuperCap and $37.06mm^3$ for Li-Thin. The model that requires the largest energy source, COBCM, with SecPB requires $4.92mm^3$ for SuperCap and $0.049mm^3$ for Li-Thin. This represents a $753\times$ decrease in the required battery capacity to support SecPB compared to s_eADR.

As more security metadata is eagerly generated, the maximum battery capacity required is reduced. We observe a significant drop in the battery required between the BCM and CM model by $6.5\times$ for SuperCap and $6.7\times$ for Li-Thin. Since the BMT root update is no longer delayed until after a crash, the intermediate BMT nodes do not need to be fetched or computed, therefore reducing the total energy required. The NoGap model requires the smallest battery amongst the models proposed, as enough energy needs to be provisioned for one memory tuple update and draining SecPB entries to

the WPQ, with only $0.36mm^3$ for SuperCap and $0.003mm^3$ for Li-Thin.

Overall, the best solution in the performance-battery size trade off space depends on the cost and form factor limitations for the supercap/battery. COBCM is clearly superior performance wise, and the cost of supercap/battery is still substantially smaller than eADR, but much larger than BBB. The budget-conscious solution may be CM, with much smaller cost for supercap/battery but also much larger slowdown (71.3% on average).

TABLE VI
ESTIMATED SUPERCAPACITOR OR BATTERY CAPACITY FOR VARYING SECPB SIZES FOR COBCM AND NOGAP MODELS.

SecPB Size	COBCM		NoGap	
	SuperCap	Li-Thin	SuperCap	Li-Thin
8	1.33	0.013	0.08	0.001
16	2.52	0.025	0.14	0.001
32	4.89	0.049	0.28	0.003
64	9.63	0.096	0.55	0.006
128	19.12	0.191	1.10	0.011
256	38.11	0.381	2.18	0.022
512	76.10	0.761	4.35	0.044

D. SecPB Size Impact

To better understand the impact of SecPB design on our models, we vary the SecPB capacity and measure the performance overheads and battery capacity. We choose CM because the observations apply to NoGap, M, and BCM as well. We vary the SecPB size from 8 to 512 entries, and show the results in Figure 8. As expected, the larger the SecPB, the coalescing opportunity of BMT root updates increases since SecPB entries are drained less frequently. A 8-entry SecPB reduces BMT updates to 12.7%, which demonstrates that even with a small capacity the BMT coalescing optimization is effective. A 512-entry SecPB capacity reduces the number of BMT updates further to 1.8%.

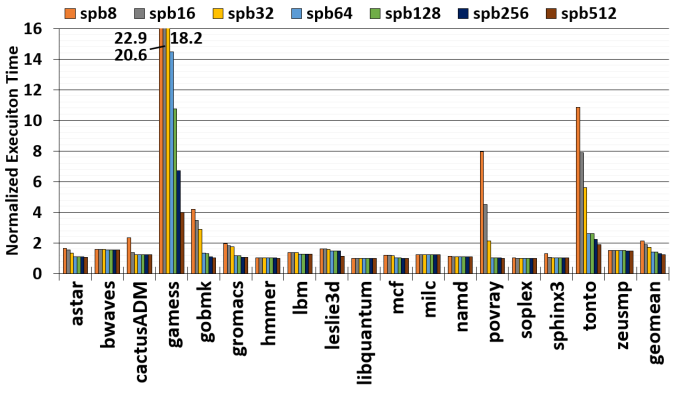

Fig. 7. Execution time of various SecPB sizes assuming a CM model.

An 8-entry SecPB incurs overheads of 112.3% while the 512-entry SecPB incurs overheads of 24%. In general, this observation was true for workloads that were sensitive to the SecPB size. Some workloads, such as *bwaves*, does not observe a reduction in BMT root updates as the capacity increased. This is due to minimal changes in NWPE as the SecPB capacity varies. Write-intensive workloads such as *gobmk* observes continued reduction of performance overheads as the SecPB capacity and NWPE increases. The increased SecPB capacity reduces the possibility of thrashing, where cache blocks drain due to the high watermark being reached are written to the SecPB again and need to update the BMT.

However, the performance benefit of the increasing SecPB capacity reaches a diminishing return with 32 or 64 entries. When also considering the cost of a larger battery, the return is not worth it. Table VI shows the impact of increasing the SecPB size for the COBCM and NoGap models, representing the largest and smallest capacity required to support SecPB. A larger SecPB increases the battery capacity even further if the BCM, OBCM or COBCM models are utilized, as the energy required increases in order to fetch and update BMT nodes for each SecPB entry. Therefore, the BMT root update has not only been exposed as a performance bottleneck but as an energy bottleneck as well. Therefore, we use 32 entries as our default SecPB size. Next, we will study how to reduce the performance impact of the BMT root update.

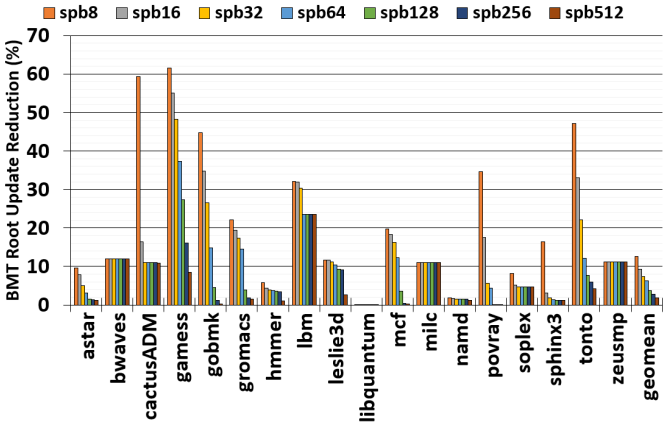

Fig. 8. Total BMT root updates in our proposed models. Normalized to the total number of BMT root updates in sec_wt.

E. BMT Height Study

To further study the impact of the BMT root update on the performance overheads, we modeled the state-of-the-art BMT height reduction mechanisms proposed in [19]. Figure 9 shows a 32-entry SecPB and CM model with DBMF and SBMF implemented and compared to the DBMF (*sp_dbmf*) and SBMF (*sp_sbmf*) mechanisms with a 4KB root cache. For the SecPB implementations, we reduced the BMT height from 8 levels to 2 for the DBMF scheme (*cm_dbmf*) and to 5 levels for the SBMF (*cm_sbmf*).

The BMT height reduction with SecPB outperformed the state-of-the-art DBMF for both the *cm_dbmf* and *cm_sbmf* models. *sp_dbmf* showed a performance overhead of 88.9%, while *cm_dbmf* improved the overhead by 55.6%, observing

an overhead of just 33.3%. *sp_sbmf* observed a slowdown of 3.43× that was reduced to just 56.6% in *cm_sbmf*. The *cm_sbmf* even outperformed the *sp_dbmf* scheme by 32.3%. This demonstrates the effectiveness of the optimizations proposed, even with a full height BMT, and re-exposes the BMT root update as the key performance bottleneck.

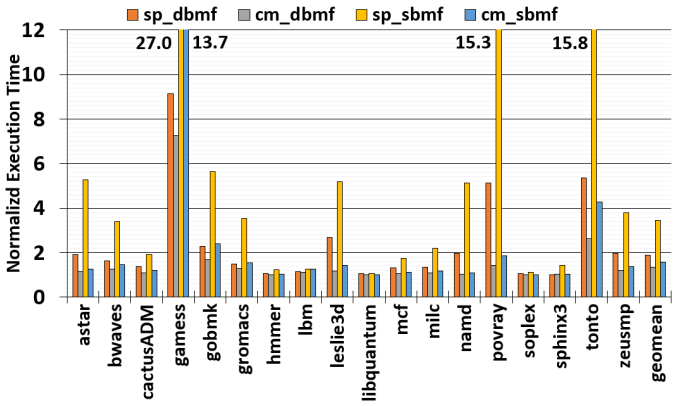

Fig. 9. Execution time of SecPB with the CM model and DBMF and SBMF implemented. Normalized to BBB baseline.

Overall, BMT height reduction techniques such as BMF are useful to pair with our SecPB if we have a limited supercap/battery budget such that we can only afford more eager schemes (e.g. CM rather than COBCM) but want to minimize the performance overheads.

VII. Conclusion

In this work, we propose *secure persistent buffers* to close the gap between a datum's point of persistency (PoP) and the secure point of persistency (SPoP) in on-chip persistent hierarchies. We proposed six secure persistency schemes that eagerly persist various security metadata elements to provide a design spectrum with performance and battery capacity considerations. Our proposed design significantly reduced performance overheads compared to strict persistency models assumed in prior work, with overheads as low as 1.3% for securing persistent memory.

Acknowledgements

We would like to thank the reviewers for their insightful feedback and comments. The NCSU team is funded in part by NSF grant 1908406, while the UCF author is funded in part by ONR grant N00014-20-1-2750.

References

[1] "Die walkthrough: Alder Lake-S/p and a touch of zen 3," 2022. [Online]. Available: https://locuza.substack.com/p/die-walkthrough-alder-lake-sp-and

[2] "Intel® Core™ i9-12900KS Processor," 2022. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/225916/intel-core-i912900ks-processor-30m-cache-up-to-5-50-ghz.html

[3] M. Alshboul, J. Tuck, and Y. Solihin, "Lazy persistency: A high-performing and write-efficient software persistency technique," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018.

[4] M. Alshboul, P. Ramrakhyani, W. Wang, J. Tuck, and Y. Solihin, "BBB: Simplifying Persistent Programming using Battery-Backed Buffers," in *The 27th IEEE International Symposium on High-Performance Computer Architecture (HPCA-27)*, 2021.

[5] M. Alwadi, K. Zubair, D. Mohaisen, and A. Awad, "Phoenix: Towards ultra-low overhead, recoverable, and persistently secure nvm," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[6] M. Alwadi, V. R. Kommareddy, C. Hughes, S. D. Hammond, and A. Awad, "Steal-persist: Architectural support for persistent applications in hybrid memory systems," in *he 27th IEEE International Symposium on High-Performance Computer Architecture (HPCA-27)*, 2021.

[7] M. Alwadi, A. Mohaisen, and A. Awad, "ProMT: optimizing integrity tree updates for write-intensive pages in secure NVMs," in *Proceedings of the ACM International Conference on Supercomputing*, 2021.

[8] AMD, "AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More," 2020.

[9] A. Awad, L. Njilla, and M. Ye, "Triad-nvm: Persistent-security for integrity-protected and encrypted non-volatile memories (nvms)," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019.

[10] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, 2011.

[11] D. R. Chakrabarti, H.-J. Boehm, and K. Bhandari, "Atlas: Leveraging locks for non-volatile memory consistency," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, 2014.

[12] S. Chhabra and Y. Solihin, "i-nvmm: A secure non-volatile main memory system with incremental encryption," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011.

[13] S. Chhabra, B. Rogers, and Y. Solihin, "Shieldstrap: Making secure processors truly secure," in *Proceedings of the 2009 IEEE International Conference on Computer Design*, 2009.

[14] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better i/o through byte-addressable, persistent memory," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, 2009.

[15] V. Costan and S. Devadas, "Intel sgx explained," Cryptology ePrint Archive, Report 2016/086, 2016.

[16] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *Proceedings of the Ninth European Conference on Computer Systems*, 2014.

[17] R. Elbaz, D. Champagne, R. B. Lee, L. Torres, G. Sassatelli, and P. Guillemin, "Tec-tree: A low-cost, parallelizable tree for efficient defense against memory replay attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2007.

[18] A. Freij, S. Yuan, H. Zhou, and Y. Solihin, "Persist Level Parallelism: Streamlining Integrity Tree Updates for Secure Persistent Non-Volatile Memory," in *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020.

[19] A. Freij, H. Zhou, and Y. Solihin, "Bonsai merkle forests: Efficiently achieving crash consistency in secure persistent memory," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021.

[20] B. Gassend, G. E. Suh, D. Clarke, M. van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, 2003.

[21] S. Ghosh, M. N. I. Khan, A. De, and J.-W. Jang, "Security and privacy threats to on-chip non-volatile memories and countermeasures," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016.

[22] J. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Appelbaum, and E. Felten, "Lest we remember: Cold boot attacks on encryption keys," in *USENIX Security Symposium*, 2008.

[23] X. Han, J. Tuck, and A. Awad, "Dolos: Improving the performance of persistent applications in adr-supported secure memory," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021.

[24] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, 2006.

[25] Intel, "Intel and micron produce breakthrough memory technology," 2015.

[26] Intel, "Intel Architecture Memory Encryption Technologies Specification," 2019.

[27] Intel, "Intel® architecture memory encryption technologies," 2021.

[28] D. Kaplan, J. Powell, and T. Woller, "AMD Memory Encryption," 2016.

[29] Y.-B. Kim, S. R. Lee, D. Lee, C. B. Lee, M. Chang, J. H. Hur, M.-J. Lee, G.-S. Park, C. J. Kim, U.-I. Chung, I.-K. Yoo, and K. Kim, "Bi-layered rram with unlimited endurance and extremely uniform switching," in *2011 Symposium on VLSI Technology - Digest of Technical Papers*, 2011.

[30] A. Kokolis, N. Mantri, S. Ganapathy, J. Torrellas, and J. Kalamatianos, "Cloak: Tolerating non-volatile cache read latency," in *Proceedings of the 36th ACM International Conference on Supercomputing*, 2022.

[31] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating stt-ram as an energy-efficient main memory alternative," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013.

[32] R. B. Lee, P. C. S. Kwan, J. P. McGregor, J. Dwoskin, and Zhenghong Wang, "Architecture for protecting critical secrets in microprocessors," in *32nd International Symposium on Computer Architecture (ISCA'05)*, 2005.

[33] T. S. Lehman, A. D. Hilton, and B. C. Lee, "PoisonIvy: Safe Speculation for Secure Memory," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, 2016.

[34] S. Liu, A. Kolli, J. Ren, and S. M. Khan, "Crash Consistency in Encrypted Non-volatile Main Memory Systems," *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[35] S. Liu, K. Seemakhupt, G. Pekhimenko, A. Kolli, and S. Khan, "Janus: Optimizing memory and storage support for non-volatile memory systems," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19, 2019.

[36] S. Nalli, S. Haria, M. D. Hill, M. M. Swift, H. Volos, and K. Keeton, "An analysis of persistent memory use with whisper," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017.

[37] D. Nayak, D. P. Acharya, and K. Mahapatra, "An improved energy efficient sram cell for access over a wide frequency range," *Solid-State Electronics*, 2016.

[38] X. Pan, A. Bacha, S. Rudolph, L. Zhou, Y. Zhang, and R. Teodorescu, "Nvcool: When non-volatile caches meet cold boot attacks," *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018.

[39] D. Pech, M. Brunet, H. Durou, P. Huang, V. Mochalin, Y. Gogotsi, P.-L. Taberna, and P. Simon, "Ultrahigh-power micrometre-sized supercapacitors based on onion-like carbon," *Nature Nanotechnology*, 2010.

[40] S. Pelley, P. Chen, and T. Wenisch, "Memory Persistency," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*, 2014.

[41] P. Prasithsangaree and P. Krishnamurthy, "Analysis of energy consumption of rc4 and aes algorithms in wireless lans," in *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, 2003.

[42] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.

[43] J. Rakshit and K. Mohanram, "Assure: Authentication scheme for secure energy efficient non-volatile memories," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017.

[44] N. Rathi, S. Ghosh, A. Iyengar, and H. Naeimi, "Data privacy in non-volatile cache: Challenges, attack models and solutions," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 348–353.

[45] N. Rathi, S. Ghosh, A. Iyengar, and H. Naeimi, "Data privacy in non-volatile cache: Challenges, attack models and solutions," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016.

[46] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using address independent seed encryption and bonsai merkle trees to make secure processors os- and performance-friendly," 2007.

[47] B. Rogers, M. Prvulovic, and Y. Solihin, "Efficient data protection for distributed shared memory multiprocessors," in *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, 2006.

[48] B. Rogers, C. Yan, S. Chhabra, M. Prvulovic, and Y. Solihin, "Single-level integrity and confidentiality protection for distributed shared memory multiprocessors," in *in Proceedings of the 14th International Symposium on High Performance Computer Architecture (HPCA-14*, 2008.

[49] A. Rudoff, "Deprecating the pcommit instruction," 2016.

[50] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, and M. K. Qureshi, "Synergy: Rethinking secure-memory design for error-correcting memories," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[51] S. Scargall, "Programming persistent memory: A comprehensive guide for developers," 2020.

[52] D. Sepranos and M. Wolf, "Challenges and opportunities in vlsi iot devices and systems," *IEEE Design & Test*, 2019.

[53] S. Shin, S. K. Tirukkovalluri, J. Tuck, and Y. Solihin, "Proteus: A flexible and fast software supported hardware logging approach for nvm," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017.

[54] S. Shin, J. Tuck, and Y. Solihin, "Hiding the long latency of persist barriers using speculative execution," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017.

[55] G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "Efficient memory integrity verification and encryption for secure processors," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, 2003.

[56] G. E. Suh, C. W. O'Donnell, and S. Devadas, "AEGIS: A Single-Chip Secure Processor," *IEEE Design Test of Computers*, 2007.

[57] S. Swami and K. Mohanram, "Acme: Advanced counter mode encryption for secure non-volatile memories," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018.

[58] S. Swami and K. Mohanram, "ARSENAL: architecture for secure non-volatile memories," *Computer Architecture Letters*, 2018.

[59] J. Szefer, "Memory protections," in *Principles of Secure Processor Architecture Design*, 2019.

[60] D. L. C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[61] S. Vig, R. Juneja, G. Jiang, S.-K. Lam, and C. Ou, "Framework for fast memory authentication using dynamically skewed integrity tree," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

[62] J. Wang, X. Dong, Y. Xie, and N. P. Jouppi, "i2wap: Improving non-volatile cache lifetime by reducing inter- and intra-set write variations," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[63] B. Westermann, D. Gligoroski, and S. Knapskog, "Comparison of the power consumption of the 2nd round sha-3 candidates," in *International Conference on ICT Innovations*. Springer, 2010, pp. 102–113.

[64] Z.-S. Wu, K. Parvez, X. Feng, and K. Müllen, "Graphene-based in-plane micro-supercapacitors with high power and energy densities," *Nature Communications*, 2013.

[65] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," in *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA)*, 2006.

[66] F. Yang, Y. Lu, Y. Chen, H. Mao, and J. Shu, "No compromises: Secure nvm with crash consistency, write-efficiency and high-performance," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019.

[67] J. Yang, Y. Zhang, and L. Gao, "Fast secure processor for inhibiting software piracy and tampering," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, 2003.

[68] M. Ye, C. Huges, and A. Awad, "Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories," in *51st Annual IEEE/ACM International Symposium on Microarchitecture*, 2018.

[69] V. Young, P. J. Nair, and M. K. Qureshi, "Deuce: Write-efficient encryption for non-volatile memories," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.

[70] S. Yuan, Y. Solihin, and H. Zhou, "Pssm: achieving secure memory for gpus with partitioned and sectored security metadata," in *Proceedings of the ACM International Conference on Supercomputing*, 2021.

[71] A. W. B. Yudha, K. Kimura, H. Zhou, and Y. Solihin, "Scalable and fast lazy persistency on gpus," in *2020 IEEE International Symposium on Workload Characterization (IISWC)*, 2020.
[72] Y. Zhu, S. Murali, M. D. Stoller, K. J. Ganesh, W. Cai, P. J. Ferreira, A. Pirkle, R. M. Wallace, K. A. Cychosz, M. Thommes *et al.*, "Carbon-based supercapacitors produced by activation of graphene," *Science*, 2011.
[73] Y. Zou, A. Awad, and M. Lin, "Hermes: Hardware-efficient speculative dataflow architecture for bonsai merkle tree-based memory authentication," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 203–213.
[74] K. A. Zubair and A. Awad, "Anubis: Ultra-low overhead and recovery time for secure non-volatile memories," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019.
[75] P. Zuo and Y. Hua, "Secpm: a secure and persistent memory system for non-volatile memory," in *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, 2018.
[76] P. Zuo, Y. Hua, and Y. Xie, "Supermem: Enabling application-transparent secure persistent memory with low overheads," in *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.