# Efficient Hardware-Assisted Crash Consistency in Encrypted Persistent Memory

Zhan Zhang*, Jianhui Yue†, Xiaofei Liao*, Hai Jin*
*National Engineering Research Center for Big Data Technology and System
Services Computing Technology and System Lab, Cluster and Grid Computing Lab
School of Computing Science and Technology, Huazhong University of Science and Technology, Wuhan, China
†Computer Science Department, Michigan Technological University, Houghton, Michigan
zhangzhan, xfliao, hjin@hust.edu.cn, jyue@mtu.edu

***Abstract*—The *persistent memory* (PM) requires maintaining the crash consistency and encrypting data, to ensure data recoverability and data confidentiality. The enforcement of these two goals does not only put more burden on programmers but also degrades performance. To address this issue, we propose a hardware-assisted encrypted persistent memory system. Specifically, logging and data encryption are assisted by hardware. Furthermore, we apply the counter-based encryption and the *cipher feedback* (CFB) mode encryption to data and log respectively, reducing the encryption overhead. Our primary experimental results show that the transaction throughput of the proposed design outperforms the baseline design by up to 34.4%.**

***Index Terms*—Persistent Memory, Computer Architecture**

## I. Introduction

Due to high storage density, fast speed, non-volatility, and byte-addressable, the *persistent memory* (PM) becomes a promising candidate to replace DRAM. The commercially available Intel Optane is a good indicator of this trend. However, data crash consistency and security are two challenges to be solved when persistent memory is widely adopted.

Crash consistency ensures that all updates within a transaction are reflected to PM in a nothing-or-all manner in case of system crashes. Logging methods are commonly adopted to achieve crash consistency. With logging, a transaction's updates are applied to in-place data after their logs are stored in PM. Logging incurs significant performance overhead due to both write amplification and ordering constraints that data update is performed after log. Research efforts address these issues by proposing software optimizations and hardware optimizations. Software logging requires a programmer to generate and write log entries to PM before performing in-place updates, increasing bounden to programmer [1]. To enforce ordering constraints, PM instructions and memory barrier instructions used by software logging incur CPU pipeline stalls, leading to inferior performance [2]. On the other hand, hardware-assisted logging not only offers better performance but also reduces a programmer's burden. This paper focuses on hardware-assisted logging approach.

This work was supported by National Key Research and Development Program of China under grant No.2017YFB1001603, National Natural Science Foundation of China under grant No. 61732010, 61825202, 61672251, and USA NSF 1745748.

Due to non-volatility, data stored in PM are accessible to attackers who steal the powered-off PM devices. To address this security issue, secure PM systems adopt the counter mode encryption to encrypt data written to PM, since it can hide data decryption latency [3]. A counter involved with encryption and decryption is updated for every updating to its data. Therefore, modified data and its counter should be persisted atomically to ensure crash consistency [3]. This requires a programmer to persist not only updated data and log entry but also their corresponding counters, complicating programming for PM.

Motivated by the aforementioned analysis, this paper proposes a secure hardware-assisted logging PM design to achieve crash consistency and data security, with minimal programming burden. The proposed design enables hardware to generate log entries, and enforces the write ordering constraint of data and its log entry to encrypted PM. If counter mode encryption is straightforwardly applied to both data and its log entry, writing one cacheline data will generate four PM write operations including an encrypted cacheline data, encrypted log entry, and their corresponding counters. This write amplification worsens the slow PM write operations, hurting system performance. In order to mitigate write amplification, we design a hybrid encryption mode to avoid storing counters for log entries without compromising system crash consistency and data security. Additionally, a new log management scheme is proposed to efficiently work for the proposed hybrid encryption method. We refer the design with these two enhancements to as EASY-PM.

The experimental results show that our proposed PM designs can collectively outperform the baseline by up to 34.4% in terms of transaction throughput.

The main contributions of this paper are as follows:

- This paper proposes a secure hardware-assisted logging PM design to achieve crash consistency and security, with the minimal programmer's burden.
- A hybrid encryption method is proposed to mitigate the write amplification issue caused by log security metadata.
- We also optimize the log management scheme to efficiently support the proposed encryption method.

The rest of this paper is organized as follows. Section II discusses hardware-assisted crash consistency and security in persistent memory. We present our designs in Section III.

Section IV shows the experimental setup and results. Section V summarizes related work. Finally, Section VI concludes this paper.

## II. Background

### A. Hardware-assisted Logging

Memory write requests can be cached, coalesced, and re-ordered for better performance, and these optimizations make write requests arrive at PM in the order different from the user program order. In addition, modified data in caches will lose after a system crash. The out-order executions of memory updates and the volatility of caches lead to the data inconsistency after the system crash, which is referred to as the crash consistency issue. Logging for modified data is commonly used to ensure crash consistency at the cost of performance overhead. There are some research efforts to maintain persistent memory crash consistency with software and hardware approaches. Without hardware modifications, software optimization approaches require programmers to use optimization-specific primitives and protocols to ensure crash consistency. Software approaches put more burden on programmers, degrade system performance [4], and make programs hard to debug [1]. On the other hand, the hardware approaches implement logging and enforce write ordering constraints between data and its log entry effectively, without introducing a significant burden to programmers. The hardware approaches are not only efficient but also easy for programmers.

ATOM [5] is a state-of-art hardware-assisted design for PM crash consistency. Upon a store instruction arriving at L1 cache, the L1 cache controller can produce an undo log entry for the incoming store and send it to the memory controller before allowing L1 cache to serve this store instruction. In order to maintain the ordering constraints between data and its log entry, the memory controller prevents modified cachelines from being written to PM before corresponding log entries are persisted to PM. Additionally, ATOM proposes an efficient log management design to reduce writing logs' metadata traffic, by coalescing multiple log entries' metadata into a cacheline. ATOM's log generation and ordering enforcement can ensure crash consistency and simplify programming for PM. This paper proposes a secure and hardware-assisted crash consistency PM design based on ATOM.

### B. Data Block Encryption

Data stored in PM is accessible to attackers who steal the powered-off PM devices, leading to a security issue. Therefore, data must be encrypted before being stored to PM. Note that the memory controller encrypts data blocks written to PM, and data blocks are not encrypted in the last level cache and higher-level caches. Since data read operation is on the critical execution path, decrypting data introduces additional 40ns [3] latency to memory read path, significantly degrading performance. In order to mitigate this issue, counter mode encryption was proposed to encrypt data [6] by deploying a counter cache in the memory controller. Figure 1 illustrates counter mode encryption. Before encrypting a data block, a *one-time-pad* (OTP) is generated by encryption engine with the key, the data block address, and the counter of the data block. After the OTP is ready, the data block is XORed with OTP, producing the corresponding ciphertext. When decrypting an encrypted data block, the decryption engine recovers the OTP with the same key, address, and counter as encryption, and then the encrypted data block is XORed with the OTP to produce the plaintext. When a data block is written back to memory, its corresponding counter is also incremented. If the counter is in the counter cache, the decryption latency for memory read request can be hidden by overlapping the calculating OTP and fetching encrypted data from PM.

When a data block is persisted, its counter also needs to be written to PM to ensure that the data can be decrypted correctly. In the secure PM with the counter mode, log entries are encrypted with the counter mode and their counters are persisted to PM when encrypted log entries are written to PM [3], which introduces the $counter_cache_writeback(var)$ function to flush the $var$'s counter to PM. It is clear that writing a data block is translated to writing four data blocks, including the written data, the corresponding log entry, and two counters for them, degrading performance. Writing these extra counter not only deteriorates slow PM writing but also reduces the PM life-span. This paper aims to reduce write traffic to PM without sacrificing crash consistency and data security.

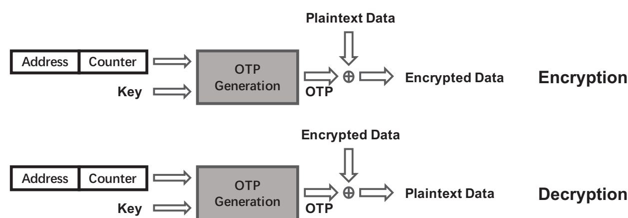


Fig. 1: Counter Mode Encryption/Decryption

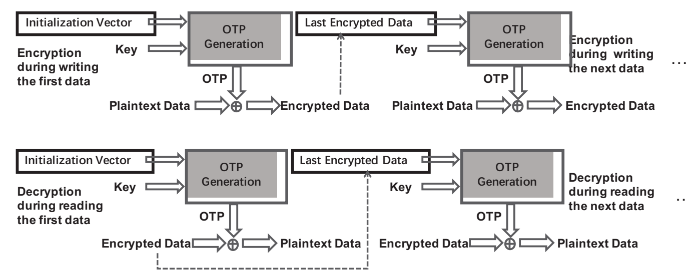


Fig. 2: CFB Mode Encryption/Decryption

Different from the counter mode encryption, the *Counter Feedback* (CFB) mode encryption [7] does not require a counter for each data block. The CFB feeds the encryption engine *Initial Vector* (IV) and key to generate OTP for the first data block and produces the ciphertext by XORing the OTP with the first data block. To encrypt the following data blocks, the IV is replaced with the most recent ciphertext

to produce the next ciphertext, shown in Figure 2. The CFB provides IV and the key to the decryption engine to recover the corresponding OTP and decrypts the ciphertext by XORing it with the OTP. To decrypt the other data blocks, CFB repeats the previous steps by replacing IV with the previous ciphertext, shown in Figure 2. Compared with counter mode encryption, CFB does not require a counter for each data block but a randomly generated IV for a group of data blocks. Note that CFB mode encryption and counter mode encryption can share the encryption/decryption engine [8]. In this paper, we propose a hybrid encryption scheme to reduce logging traffic, by exploiting the advantages of CFB mode encryption.

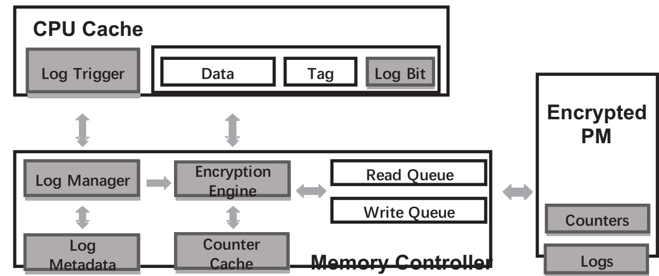


Fig. 3: Baseline Architecture

## III. EASY-PM DESIGN

After introducing the programming model, we show the hardware assisted-logging and secure PM design, as the baseline design. To address issues of the baseline design, we propose the hybrid encryption policy and the log management strategy for CFB encryption. These two enhancements over the baseline design are collectively referred to as EASY-PM.

### A. Programming Model

This paper presents a programming model to reduce a programmer's burden to maintain crash consistency and data security for persistent memory. This model achieves the crash consistency with hardware logging by using the primitives $tx_begin()$ and $tx_end()$ to specify a transaction. Then the modified data in a transaction are specified by the $flush_data(var)$ and are persisted to PM at the end of the transaction. In addition to flush these data, $flush_data(var)$ persists their counters in the counter cache to PM. Different from Ref. [3], the proposed programming model relieves programmers from log generations and log space management, to reduce programming burden. In a transaction, programmers mainly focus on processing logic, and the proposed hardware helps to maintain crash consistency and data security.

### B. Baseline Architecture

The baseline design aims to persist and encrypt data stored in PM in hardware way to relieve the burden of programmers. It is not trivial to implement these features in hardware as it coordinates the memory write operations, logging, and data encryption/decryption. Before going to detail, we briefly introduce the two major modules of the design: the log management module and the memory encryption module. The log management module generates and persists undo log to PM, and ensures log entries be persisted to PM before the corresponding data to achieve crash consistency. The memory encryption module encrypts/decrypts memory data in the counter-based encryption mode, including data and its log entry. The baseline design's hardware modifications introduce new components to the on-chip cache and memory controller shown in the gray in Fig. 3.

After detecting store requests to L1 cache, the *Log Trigger* in L1 cache controller creates undo log entries, and sends them to the *Log Manager* before store requests written to the L1 cache. Each cache line is augmented with a *Log Bit* to indicate whether its log has been done to avoid unnecessary logs caused by repeated writes to a cache line in one transaction. Similar to ATOM [5], the *Log Metadata* maintains the metadata to manage log space and log entries, and these tasks are performed by Log Manager module. The *Encryption Engine* encrypts either data from the *Last Level Cache* (LLC) or log entries from Log Manager module. After that, an encrypted data block is sent to the memory controller's ADR protected write queue, whose entries survive power failure and can be safely written to PM [3]. Encryption Engine also decrypts encrypted data and log entries read from PM. In the case of a counter cache miss, the requested counter is fetched from PM, failing to overlap the OTP generation with the ciphertext fetching.

A log entry consists of unmodified cacheline data (64B) and its address (8B). Similar to ATOM [5], a record has seven unmodified data blocks (log data blocks) and a 64B header. The header stores seven data block addresses and the amount of valid log entries in the record. A header is encrypted and written to PM at the head of the record, after the corresponding log entries are encrypted and persisted. A group of headers and records forms a bucket, which is a memory allocation/deallocation unit for the log region. Multiple log data blocks stored in a record are encrypted and persisted to PM, prior to the corresponding header encrypted and written to PM [5]. While log space allocation is triggered by writing log entries during the execution of a transaction, log entries of a transaction are freed after it is committed at the end of $tx_end()$.

### C. Hybrid Encryption Policy

Achieving both security and crash consistency for PM require encrypting and logging for the modified data block. Writing logging entries introduces performance overhead, slowing down performance. Encrypting a data block and its log entry involves writing counters of both the data block and the log entry. Persisting these extra counters further worsens system performance.

In order to mitigate performance overhead caused by writing counters, we propose the hybrid memory encryption policy that eliminates writing a counter for each log entry without sacrificing data security. The main idea is that we apply the CFB encryption to encrypt log entries without involving extra

counters. CFB generates the OTP using a secret key and the most recent ciphertext, and produces the ciphertext by XORing the OTP with current plaintext. Then this ciphertext is sent to PM and it is also stored in the memory controller's register, referred to as *Recent Ciphertext* (RC) register. When a log encryption request arrives, we encrypt the log entry with the RC register and the key, and then update the RC register with the newly generated ciphertext. In this way, no counter is written to PM for a log entry.

It is not beneficial to adopt CFB to encrypt program data compared with the counter encryption mode. To decrypt a data block, CFB needs to know the recent ciphertext for the data block and the recent ciphertex is referred to as the tracking metadata of the data block. Maintaining the tracking metadata for each data block not only updates it after encrypting the data block, but also introduces additional latency to access it before decrypting the data block. However, CFB works well for log entries without the aforementioned overhead. This is because the log entries sequential layout and append-only access pattern in PM enables CFB to easily locate an encrypted log entry's recent ciphertext, which is stored before the log entry. Therefore, we still apply the counter mode encryption to program data.

### D. *Log Management for the CFB Encryption*

It is not trivial to manage log space with CFB encryption. The hybrid encryption policy adopts undo logging to maintain crash consistency and encrypts logs with CFB. A log record consists of seven log entries, with seven log data blocks and a header block. When a modified data block is evicted from LLC, its log entry and the log record should be written to PM. If the log record has less than seven log entries, writing this log record leaves some invalid data blocks in it. Each data block in this log record is required to encrypted by CFB and encrypting invalid data blocks wastes precious encryption engine cycles. To avoid this resource waste, only valid log data blocks are encrypted and persisted before their header, which contains the number of valid log data blocks. Note that the encryption of the header depends on the ciphertext of the last valid log data block of this log record, due to the encryption nature of CFB. In the recovery process, the ciphertext of a header is needed to be decrypted before its corresponding encrypted log data blocks so that the memory controller knows the exact number of log data blocks to be decrypted. However, the ciphertext of a header can not be decrypted until its last valid log data block is encrypted. This makes the crash consistency recovery hard.

In order to address this issue, we design a new log management method. The basic idea is that we treat headers and log data blocks as two separate chains. A bucket has a fixed number of log records and it consists of a header chain, a log data chain, and metadata including *Valid Header Number* (VHN), *Header IV* (H-IV), and *Log IV* (L-IV), shown in Fig. 4. A header and log data blocks of a log record are stored in the header chain and log data chain respectively, which means a header and its log data blocks are not sequentially stored in PM, different from ATOM. A header also keeps the number of valid log entries in the corresponding log record. The Header IV and Log IV store IVs for the header chain and the log chain respectively. The VHN indicates the active header block. Using CFB, we encrypt the active blocks in the header chain and log data chain with the Header IV and Log IV respectively, and save the ciphertext in the corresponding RC registers to generate OTP for following blocks.

During the system recovery process, we decrypt log ciphertext in the buckets. Initially, we read the header IV from the bucket and store it to the header RC register. With the RC register, we can decrypt a header along the header chain to obtain valid log entry number and undo log entries' source address, and save the current header's ciphertext to header RC register. This process is repeated until the end of the header chain is reached, which is indicated by Valid Header Number. After the plaintext of a header is available, the header's valid log entry number determines encrypted log data blocks accompanied by this header. We decrypt these log data blocks with the log RC register along the log data chain. Each log data's ciphertext is stored in the log RC register to prepare for decrypting the next log data. This process is repeated until all valid log data in the current record are encrypted before proceeding to the next record.

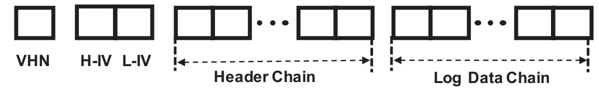


Fig. 4: Log Bucket Structure for CFB

The overhead of the undo log component is comparable to ATOM [5], which does not support the encryption. We have the same storage overhead of log metadata registers in the memory controller as ATOM, which is 3.15KB for 32 concurrent transactions. Due to CFB encryption, we introduce the recent ciphertext registers in the memory controller, consuming 2KB for 32 concurrent transactions.

## IV. Evaluation

### A. *Experiment Setup*

TABLE I: System Parameters

| Cores | 4 OoO core @2GHz,<br>192 ROB entries, 48 store queue entries |
|---|---|
| L1 I/D Cache | private, 32KB, 2 cycles, 8 way |
| L2D Cache | private, 256KB, 8 cycles, 8 way |
| LLC | shared, 8MB, 64B cacheline, 25 cycles, 16-way |
| Memory Controller | 32 write queue entries, 32 read queue entries |
| Persistent Memory | 1 channel, 1 rank, 8 banks, 8GB,<br>300(48) ns write(read) latency [3], [9] |
| En/decryption | 40ns latency [3] |
| Counter cache | 512KB, 12 cycle |

The proposed designs are implemented and evaluated by using ChampSim [10] with DRAMSim2 [11]. ChampSim is an Intel PIN [12] based simulator that models the out-of-order micro-architecture at cycle level with detailed memory access behaviors including LSQ memory dependence, TLB, and cache miss status holding registers, which are not modeled in the McSimA+ [13] used in previous NVM crash consistency researches [4], [14]–[16]. To accurately model PM accesses, the cycle-level memory simulator DRAMSim2 is incorporated to ChampSim. We enhance ChampSim to support *tx_begin*, *tx_end*, and *flush_data*. The configurations of the processor and memory system used in our experiments are listed in Table I. The workloads used for evaluation include Array, Hash Table, Chain Queue, B+ Tree, and RB_Tree, similar to Ref. [3].

We evaluate the following designs:

- The *Baseline Design*: Logging and memory encryption are performed by hardware without a programmer intervention, except to follow the programming model proposed in this paper. It enforces the ordering constraints of logging and in-place update, and adopts the counter mode encryption for both log entries and data. Since log entries are generated by hardware, a programmer cannot control the writing process of log counters like SCA [3]. Therefore, we write log entries to PM along with their counters. This design acts as the first baseline.
- *Asynchronous-Log-Counter* (ALC): Encrypted data, data counters, and encrypted log entries are flushed to PM at the end of a transaction, while log counters are asynchronously written to PM due to the counter cache evictions. It does not ensure the consistency between log entries and their counters but removes the flushing of log counters. We use ALC as the second baseline since it has no overhead of synchronously flushing counters compared with the counter mode encryption.
- *EASY-PM*: It applies the counter mode encryption and CFB encryption to data and log entries respectively, different from the baseline designs.

### B. Results

Fig. 5 demonstrates EASY-PM can improve transaction throughput by 22.04% on average and up to 34.4% compared with the first baseline design. The improvement mainly comes from the EASY-PM's elimination of flushing log counters to PM. The reduced write traffic can speed up a transaction execution. In addition, EASY-PM outperforms ALC by on average 3.68% in terms of transaction throughput, due to no log counters written to PM. Note that for the B+ Tree, EASY-PM is slightly worse than ALC because of writing extra IVs for log entries.

Fig. 6 indicates EASY-PM reduces not only write PM traffic but also read PM traffic by on average 55.61% and 5.26% respectively, compared with the first baseline design. While REQ_RD and REQ_WT are overall PM read and write operations, CNT_RD and CNT_WT are PM read and write operations caused by the counter accesses shown in Fig. 6. Due to CFB for log encryption, EASY-PM avoids log counters read and write to PM. In addition, the elimination of counters for log entries can reduce counter cache contention and result in less counter cache evictions, which translates to the counter request traffic reduction.

Fig. 7 shows the L1 cache access latency reduction in EASY-PM compared with the first baseline. As shown in Fig. 7, EASY-PM can reduce L1 cache access latency on average 18.63% and up to 25.89%. This is because EASY-PM can reduce memory traffic to PM and hence decrease waiting time for programmer's memory read requests.

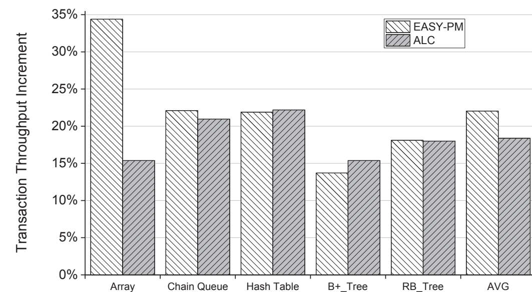


Fig. 5: Transaction Throughput

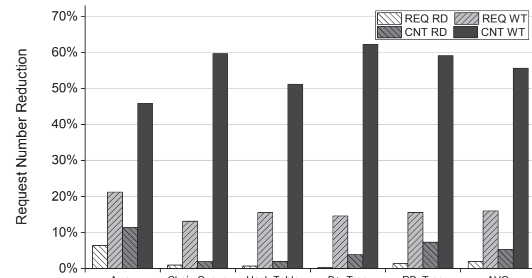


Fig. 6: PM Traffic Reduction

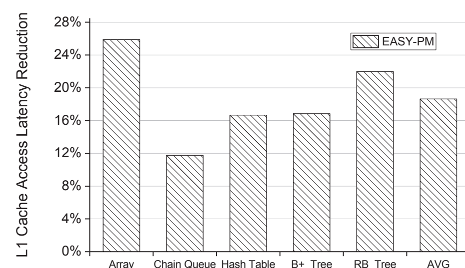


Fig. 7: L1 Cache Access Latency

## V. Related Work

**Secure PM.** Data stored in PM are needed to be encrypted since it is prone to physically stolen after power down. There are research efforts to address performance overhead for secure PM [3], [9], [17], [18]. Ref. [3] identifies the counter atomicity overhead for the counter mode encryption and proposes the selective counter atomicity, which only enforces

counter atomicity for memory updates affecting recoverability. DeWrite [19] parallelizes memory encryption and memory deduplication for the main memory backed by PM and designs space-efficient metadata layout by co-locating their metadata. Janus [9] optimizes the performance of PM *Backend Memory Operations* (BMO), including encryption, verification, and others, by parallelizing and pre-executing sub-operations of these BMO operations. Osiris [17] effectively reduces the PM write overhead by using ECC to sanity-check counter validity and reduces write traffic by eliminating premature counter evictions from the counter cache. Triad-NVM [18] treats persistent data regions and non-persistent data regions in NVM differently to improve its encryption and verification. cc-NVM [20] targets encryption and authentication on NVM. By introducing an epoch-based Bonsai Merkle Tree, cc-NVM caches the security metadata in CPU cache as much as possible, and reduces the computational overhead of data authentication by deferred spreading, which improves system performance and write efficiency. Different from previous studies, our paper presents a hardware-assisted design to support data logging and encryption for PM, and proposes a hybrid encryption scheme to encrypt data and log entries differently to reduce secure logging overhead.

**Hardware Assisted PM Crash Consistency.** Maintaining crash consistency is important to enable PM widely adopted. Prior studies [2], [5], [15] have proposed different hardware-assisted methods to ensure crash consistency. Ref. [15] presents a hardware-assisted approach to redo log for transactions with a controlled-evictions victim buffer. While ATOM [5] proposes a hardware approach and optimizations to undo logs. ReDU [2] flushes modified data from CPU caches to a DRAM cache with redo logging and directly performs in-place updates using modified data stored in the DRAM cache. These hardware-assisted approaches do not consider encrypting data. However, our paper presents a hardware-assisted design to take both crash consistency and data security into account.

## VI. Conclusion

Maintaining both crash consistency and security for PM is challenging not only for the system design but also for programmers. To address this issue, we propose a hardware-assisted data persistent and secure PM system using counter mode memory encryption, requiring minimal programming efforts. In this baseline design, the write amplification caused by logging and encryption metadata not only deteriorates the slow PM writing issue but also reduces the lifespan of PM. To mitigate the aforementioned write amplification issue, we propose to apply counter-based and CFB encryption methods for data and log entries respectively. Our primary experimental results show that the proposed hybrid encryption scheme outperforms baseline design by up to 34.4% in terms of transaction throughput.

## References


[1] T. Nguyen and D. Wentzlaff, "Picl: A software-transparent, persistent cache log for nonvolatile main memory," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Fukuoka, Japan, 2018.

[2] J. Jeong, C. H. Park, J. Huh, and S. Maeng, "Efficient hardware-assisted logging with asynchronous and direct-update for persistent memory," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Fukuoka, Japan, 2018.

[3] S. Liu, A. Kolli, J. Ren, and S. Khan, "Crash consistency in encrypted non-volatile main memory systems," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, Austria, 2018.

[4] M. A. Ogleari, E. L. Miller, and J. Zhao, "Steal but no force: Efficient hardware undo+redo logging for persistent memory systems," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, Austria, 2018.

[5] A. Joshi, V. Nagarajan, S. Viglas, and M. Cintra, "ATOM: atomic durability in non-volatile memory through hardware logging," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Austin, USA, 2017.

[6] 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 International Symposium on Computer Architecture (ISCA)*, New York, USA, 2006.

[7] M. Dworkin. Recommendation for block cipher modes of operation-methods and techniques. https://csrc.nist.gov/publications/detail/sp/800-38a/final.

[8] M. Goodrich and R. Tamassia. Introduction to computer security. Pearson, 2011.

[9] 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 International Symposium on Computer Architecture (ISCA)*, New York, USA, 2019.

[10] Champsim. https://github.com/ChampSim/.

[11] Dramsim. https://github.com/umd-memsys/DRAMSim2.

[12] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proceedings of the ACM SIGPLAN Conference on Programming language design and implementation (PLDI)*, New York, USA, 2005.

[13] J. H. Ahn, S. Li, S. O, and N. P. Jouppi, "Mcsima+: a manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Austin, USA, 2013.

[14] J. Zhao, S. Li, D. H. Yoon, Y. Xie, and N. P. Jouppi, "Kiln: Closing the performance gap between systems with and without persistence support," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Davis, USA, 2013.

[15] K. Doshi, E. Giles, and P. J. Varman, "Atomic persistence for scm with a non-intrusive backend controller," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Barcelona, Spain, 2016.

[16] X. Hu, M. Ogleari, J. Zhao, S. Li, A. Basak, and Y. Xie, "Persistence parallelism optimization: A holistic approach from memory bus to rdma network," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Fukuoka, Japan, 2018.

[17] M. Ye, C. Hughes, and A. Awad, "Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Fukuoka, Japan, 2018.

[18] A. Awad, M. Ye, Y. Solihin, L. Njilla, and K. A. Zubair, "Triad-nvm: Persistency for integrity-protected and encrypted non-volatile memories," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, New York, USA, 2019.

[19] P. Zuo, Y. Hua, M. Zhao, W. Zhou, and Y. Guo, "Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Fukuoka, Japan, 2018.

[20] F. Yang, Y. Lu, Y. Chen, H. Mao, and J. Shu, "No compromises: Secure nvm with crash consistency, write-efficiency and high-performance," in *Proceedings of the Design Automation Conference (DAC)*, Las Vegas, USA, 2019.