This article has been accepted for publication in IEEE Computer Architecture Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LCA.2022.3225949

Ensuring Data Confidentiality in eADR-based NVM Systems

Jianming Huang, and Yu Hua, *Senior Member, IEEE*

Abstract—Extended Asynchronous DRAM Refresh (eADR) proposed by Intel significantly improves the performance of Non-Volatile Memory (NVM) systems due to the persistence of caches. When the persistence boundary moves up to caches for high performance, existing designs however fail to provide efficient encryption schemes to ensure data confidentiality in the eADR-based NVM systems. Once the system crashes, eADR has to flush the un-encrypted data from the cache into NVM, in which security issues occur due to the persistence of caches. In order to ensure data confidentiality, in eADR-based NVM systems, we propose a cost-efficient Sepencr that is able to encrypt the cached data in eADR-based NVM systems. Our evaluation results show that compared with directly encrypting data in the cache, our design significantly reduces system overheads while efficiently ensuring data confidentiality.

Index Terms—Non-volatile memory, eADR, confidentiality

1 Introduction

Non-Volatile Memory (NVM) provides near-DRAM performance and disk-like durability. The byte-addressable feature of NVM is efficient to deliver high performance for NVM devices as Persistent Memory (PM) [1]. The NVM, as persistent memory, allows the persistence boundary to move from storages, e.g., disks and SSDs, to the memory [1]. Moreover, Intel proposes Asynchronous DRAM Refresh (ADR) [2] and extended ADR (eADR) [3] to further extend the persistence domain. Specifically, ADR guarantees that the write pending queue (WPQ) in the memory controller becomes a persistent domain by flushing the data from WPQ into NVM upon power-down via the backup battery. eADR guarantees that all on-chip data buffers, including CPU caches, are persistent domains [3] by flushing data from these buffers into NVM upon crashes. By using ADR and eADR, the persistence boundary moves to the on-chip buffers.

Since data need to be maintained in the persistent devices for reliability and durability, data confidentiality in the persistent devices is important to protect data from attacks. In existing storage architectures, different persistent devices leverage different encryption schemes. For the secondary storage, the data-at-rest encryption schemes are used to encrypt data, e.g., full disk encryption (FDE) in the disks [4]. For the persistent memory, e.g., Intel Optane PM, the standard 256-AES hardware encryption [5] is used to encrypt data in PM.

Although the on-chip caches are the security domains in the community [6], [7], [8], [9], efficient encryption is still important for protecting the cached data in the eADR-based NVM systems. The caches in the eADR domain are the *out-of-place persistent domains*, which means that these caches are volatile, and the data in caches are guaranteed to be persisted in the real non-volatile device, e.g., NVM, after crashes. The plaintext data moving out from the caches to the NVM without encryption become vulnerable to attackers. Unfortunately, we also observed that existing counter mode encryption (CME) scheme in the ADR-based memory controller cannot guarantee data confidentiality in eADR-based NVM systems. The CME works until the system crashes, while data are flushed after the system crashes via the support of eADR. Although the written data during the running time are encrypted by CME in the memory controller, the plaintext data flushed after crashes via eADR are vulnerable.

In the eADR-based NVM system, upon crashes, the plaintext data in caches are flushed into NVM to achieve the data persistence, at the cost of data confidentiality. To ensure data confidentiality, an intuitive approach is not to flush the cached data into NVM upon crashes, i.e., just discarding the cached data for security. However, this intuitive approach breaks the persistence of eADR-based caches, invalidating the eADR mechanism. Hence, the eADR-based NVM systems cannot ensure both data persistence and confidentiality at the same time.

To encrypt data in CPU caches, implementing encryption/decryption with extremely low latency is important. Compared with WPQ, PM, and disk, the caches are more latency-sensitive, due to being closer to the processor, and frequently handling data writes and reads. Existing encryption schemes, including the CME, are inefficient when applied into caches due to the high latency.

To ensure data confidentiality in the eADR-based NVM systems and implement the low-latency encryption scheme, we propose the **Sep**arate **Encr**yption (Sepencr) scheme by exploring and exploiting the insights that the CME can be divided into two separate steps. CME leverages the counters and memory addresses of data to generate OTPs. The OTPs are then XORed with the plaintext data for encryption. Note that the latency of CME is mainly incurred by generating OTPs while XOR operation is very fast [10]. Our Sepencr generates OTPs for all cached data in advance, and stores the OTPs in the memory controller to encrypt all cached

• *Jianming Huang and Yu Hua are with Huazhong University of Science and Technology, Wuhan, Hubei 430074, China (e-mail: jmhuang@hust.edu.cn; csyhua@hust.edu.cn).*

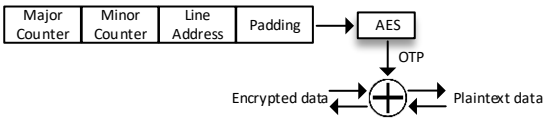


Fig. 1. The Counter Mode Encryption scheme.

data with low latency in case the crashes. The system needs to obtain the data addresses and counters for generating OTPs. To avoid cached data being flushed before the OTPs are generated, our Sepencr leverages the cache addresses instead of the memory addresses of specific data to generate the OTPs (called C-OTPs) when systems start running. The data are encrypted by CME during system running time, and the C-OTPs are only used for encryption on system crashes. Therefore, the counters of C-OTPs only increase when the system crashes and recovers. Due to the low energy and space costs of XOR gates [11], the data can be XORed with C-OTPs in the memory controller on crashes by using the backup battery in eADR.

Sepencr efficiently supports data confidentiality in eADR-based systems. To evaluate the performance of our proposed design, we use Gem5 [12] to implement Sepencr to run 5 persistent workloads. Compared with directly encrypting data in the cache, our Sepencr significantly reduces performance overheads from 403% to 4%.

2 Backgrounds

2.1 Threat Models

Like the threat models in the community [6], [7], [8], [9], we assume that only the on-chip domain, including the processor, caches and memory controller, in the computer system is safe. In our threat model, the plaintext data pass through the memory bus and are stored in non-volatile DIMM, which are potential to be attacked and cause information leakage.

2.2 Counter Mode Encryption

To ensure data confidentiality, the counter mode encryption (CME) has been widely used in existing NVM-based secure systems [6], [13], [14]. The CME is executed in the memory controller and transparent to applications. As shown in Fig. 1, the AES engine encrypts the counter, data memory address and padding via the secret key to generate the one-time padding (OTP). When writing data into NVM, the plaintext data XOR the OTPs to generate the encrypted data. When reading data from NVM, the encrypted data XOR the OTPs to generate the plaintext data. Compared with the direct AES encryption using the unchanged secret key, CME is more secure since the OTPs for encryption are *one-time*. Since counter blocks are cached in the memory controller, systems generate the OTPs in parallel with reading the encrypted data from NVM. Therefore, the decryption latency in CME is hidden by the latency of reading data.

2.3 Security issues in eADR-based NVM systems

In order to store data, hierarchical architectures generally consist of cache, memory, and storage, which have dramatically changed over the decades. The bottom-up storage hierarchy contains more and more persistence levels, which meantime introduce associated encryption schemes to protect data. Specifically, the disk-based persistent domain encrypts data via full disk encryption (FDE) [4]. A standard 256-AES hardware encryption [5] is leveraged in the non-volatile memory (NVM) to protect the data. To efficiently offer system consistency, Intel proposed ADR [2] to flush data from the WPQ in the memory controller (MC) into NVM upon system crashes. The memory controller is the persistent domain in the ADR-based NVM system. The data passing through the memory bus may be attacked in the context of our threat model. To protect the data confidentiality, the system leverages the Counter Mode Encryption (CME) in the memory controller to encrypt the data before the data enter memory bus. Recently, the eADR [3] technique is further proposed, in which the data caches are the persistent domain. In the eADR-based NVM system, the persistence boundary moves up compared with the ADR-based NVM system. However, existing encryption schemes designed for ADR-based NVM system are inefficient to encrypt the data as we describe below.

In our threat model, the cache exists in the secure on-chip domain. However, the eADR-based caches are the *out-of-place persistent domains*, i.e., the data in the cache are finally flushed to NVM for persistence after crashes, which incurs the confidentiality issue. Specifically, during system running time, the cached data are encrypted by CME in the memory controller. The encrypted data are flushed into NVM. However, after system crashes, the AES-engine in CME does not work due to lack of power. But the cached plaintext data are still flushed into NVM through the memory bus, which is unsecure in our threat models.

3 System Designs and Implementations

To ensure data confidentiality, we first discuss the direct data encryption in the cache. We further demonstrate the separate structure of counter mode encryption. We finally implement our Sepencr to encrypt data in the eADR-based NVM systems.

3.1 Move the Encryption Up

To ensure the confidentiality of data in eADR-based NVM systems, an intuitive idea is to move the encryption engine to cache to directly encrypt data in the eADR-based system. The OTP is generated in the cache, and all data in cache are encrypted by XORing the corresponding OTPs. After system crashes, the encrypted data in cache are directly flushed into NVM. However, directly encrypting the data in cache is inefficient for protecting data confidentiality due to the high latency of encryption/decryption in the latency-sensitive cache. When a processor reads/writes data from caches, the data need to be decrypted/encrypted, i.e., generating the OTPs and XORing the data with the OTPs. The data decryption/encryption in the latency-sensitive caches exists on the critical path of reading/writing data, and significantly decreases the system performance. Moreover, the cache domain is designed to store data, which is typically constructed via SRAM. Moving the AES engine into cache will increase the complexity of the manufacturing process.

3.2 The Separate Structure of CME

As shown in Fig. 2, CME is partitioned into two stages: OTP generation and XOR encryption. In the OTP generation stage, the counter-based seeds (i.e., counters, addresses and

This article has been accepted for publication in IEEE Computer Architecture Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LCA.2022.3225949

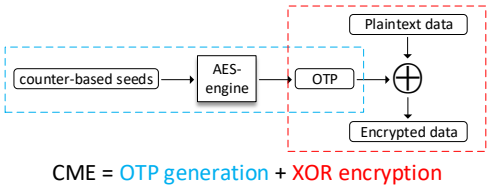


Fig. 2. The separate structure of counter mode encryption.

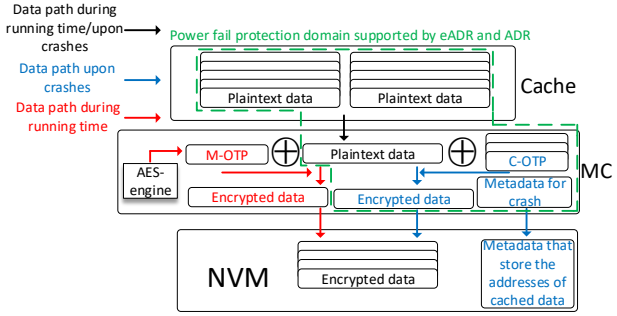


Fig. 3. The design of Sepencr.

paddings) are encrypted by the secret key via the AES-engine to generate the OTPs. In the XOR encryption stage, the OTPs are XORed with plaintext data to perform encryption. In the CME, the OTP generation dominates the latency, and the operation of XORing OTPs with data becomes fast, i.e., less than 1 cycle [10]. The data are finally encrypted in the XOR encryption stage.

From Fig. 2, we observe that the OTP generation and XOR encryption are in different stages in CME, and can be separately performed. Based on this observation, we propose **Sep**arate **Encr**yption (Sepencr) to encrypt data in the eADR-based NVM systems. The idea of Sepencr is to decouple the OTP generation and data encryption, i.e., we generate the OTPs in advance similar to [15], and XOR the OTPs with cached data upon crashes.

3.3 Sepencr Design

Our Sepencr proposes two types of OTPs: M-OTPs and C-OTPs. Like Fig. 1, M-OTPs are generated via the memory addresses and counters of data. Instead, C-OTPs are generated via the cache addresses and counters. We define the cache address of a cache line as a combination of the *cache level*, the *set number* in the cache, and the *way number* in the set of the cache line. According to the line's location in the cache, each cache line corresponds to a unique and fixed cache address.

As shown in Fig. 3, the M-OTPs are used during running time like the traditional CME. The C-OTPs are only used upon system crashes to encrypt data. Since the cache addresses of cache lines are unique and fixed, we generate the C-OTPs of all cache lines at system startup, and store the C-OTPs in the memory controller. In our Sepencr, in addition to the data buffers on-chip, i.e., data caches, counter cache, and metadata buffer, the XOR gates also reside in the power-failure protection domain by slightly extending the eADR domain as shown in Fig. 3. During running time, when flushing data from cache into NVM, the M-OTPs are generated in the memory controller on demands, and the plaintext data are encrypted by M-OTPs. Upon system crashes, the plaintext cached data are flushed into the memory controller. These plaintext data are further XORed with the C-OTPs (M-OTPs cannot be generated after system crashes) by the support of the backup battery in eADR. Finally, the encrypted data are flushed into the NVM. Moreover, Sepencr flushes the metadata storing the memory addresses of cached data into NVM via the support of eADR for recovery. On recovery, the system reads the cached data via the memory addresses stored in metadata, and generates the old C-OTPs again via the cache addresses and counters of cache lines to decrypt the cached data.

We now analyze the feasibility of extending the eADR domain to support the XOR operations. Unlike the complex AES-engine [16], the XOR gate is one of the basic circuits with the simplest operation [17]. Moreover, the silicon area and energy overheads of the XOR gate are very small, i.e., about 16.56 x 12.81 um^2 with 11 transistors, and 100 fJ [11]. Therefore, it is practical to implement Sepencr in the eADR-based NVM system.

4 Performance Evaluation

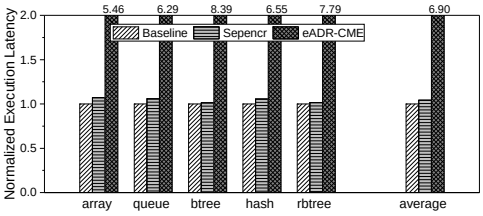
To evaluate the performance of our Sepencr, we model the cycle-accurate systems in the Gem5 [12]. The data cache is 4MB while the counter cache in the memory controller is 512KB for storing counter blocks. We model the 16-GB NVM via PCM, and use 5 typical persistent workloads, i.e., array, queue, btree, hash, and rbtree to evaluate the systems. Like existing eADR-based designs [1], [3], we also remove the CLFLUSH, CLFLUSHOPT, and CLWB instructions from source codes to build the workloads.

To comprehensively examine the performance of our proposed designs, we evaluate and compare the following schemes.

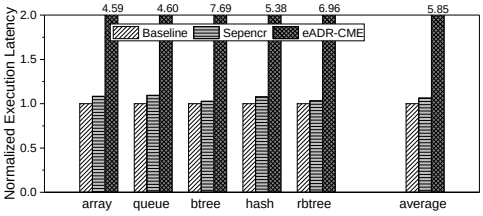
- Unsecure eADR-based NVM system as Baseline. The Baseline system contains the eADR mechanism without any data encryption.
- eADR system with CME in cache (eADR-CME). eADR-CME moves the AES engine from the memory controller to the cache as described in Section 3.1 to encrypt data in the cache. The encryption and decryption operations exist on the critical path of writing/reading data into/from caches.
- Our proposed Sepencr (Sepencr). In Sepencr, the XOR gates reside in the power failure protection domain. Upon crashes, the data are XORed with C-OTPs in the memory controller for encryption and flushed into NVM.

4.1 Result Analysis

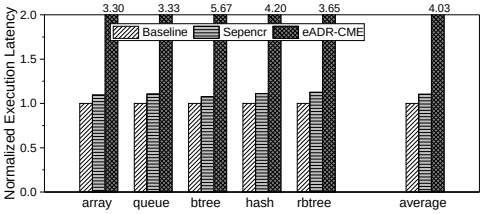
We run the workloads in different transaction sizes, e.g., from 64B to 1,024B. As shown in Fig. 4, the performance overheads of eADR-CME are very high. On average, the execution latency of eADR-CME is 4.03x – 6.90x longer than that of the Baseline scheme. In eADR-CME, when writing data from the processor into the cache, the AES engine generates the OTPs to encrypt the plaintext data. When reading data from the cache to the processor, the AES engine generates the OTPs to decrypt the encrypted data. The encryption/decryption significantly decreases the performance of eADR-CME. Unlike eADR-CME, Sepencr prepares the OTPs at system startup, i.e., the C-OTPs. Sepencr stores C-OTPs in the memory controller and leverages C-OTPs to encrypt the cached data upon system crashes. In Sepencr, the data are encrypted/decrypted via the traditional CME



(a) 64B transaction size



(b) 256B transaction size



(c) 1,024B transaction size

Fig. 4. The execution latencies of workloads in different transaction sizes (normalized to Baseline).

method during running time, and the C-OTPs are only XORed with data upon crashes in the memory controller. Since the decryption latency in CME is masked by that of reading data from NVM [6], [13], [14], the execution latency of Sepencr is small, i.e., 1.04x – 1.10x longer than that of Baseline on average.

4.2 Space and Energy Costs

The extra space cost in Sepencr over the Baseline scheme is incurred by storing C-OTPs and metadata in the memory controller (MC). The size of one C-OTP is the same as that of one data line (64B). To encrypt all cached data, the space cost of C-OTPs is the same as the size of the data caches, e.g., when we use 4MB cache to store user data, we need 4MB space in MC to store C-OTPs. For metadata buffer, since the memory address of the 64B data block is 64b, the size of the metadata storing memory addresses of cached data is 1/8 that of the cached data, i.e., 512KB (4MB/8=512KB). Moreover, since the atomic operation granularity in the processor is 8b, we place 8 XOR gates in the memory controller in Sepencr. The silicon area costs of 8 XOR gates is 0.001697 mm^2 with 88 transistors [10].

We estimate the energy costs of Sepencr following BBB [3], i.e., assuming energy costs of flushing data from the cache/memory controller into NVM are $11.228nJ/Byte$. Moreover, the energy cost of XORing two bytes to obtain one byte is $800fJ$ [10]. The Baseline requires 47.0936mJ to flush 4MB cached data upon crashes. In Sepencr, in addition to 47.0936mJ for flushing cached data, we require 11.7734mJ to drain 512KB counter cache and 512KB metadata buffer from the memory controller into NVM, and 0.0034mJ to support XOR gates upon crashes. Therefore, the overall energy cost of Sepencr is 58.8704mJ that is 25% higher than Baseline.

Although our Sepencr incurs the space overheads for ensuring data confidentiality, we can leverage some optimizations to reduce this overhead. Specifically, only the dirty cached data need to be flushed into NVM upon crashes to update the stale data, while the clean cached data are discarded upon crashes without any data loss. We can generate C-OTPs only for the dirty cached data to reduce space overhead of C-OTPs. Moreover, the deduplication technique can be leveraged in the memory controller to reduce the duplicated dirty cached data upon crashes, which further reduces the number of C-OTPs. We leave the optimization of Sepencr in terms of space overhead as our future work.

5 Conclusion

This paper proposes Sepencr to ensure data confidentiality in the eADR-based NVM systems. Sepencr leverages the unchanged cache addresses of cache lines to generate the C-OTPs on system-start. These C-OTPs are stored in the memory controller and only used to encrypt cached data upon system crashes. During system running time, the data in the system are encrypted by using CME. We slightly modify the eADR to support basic XOR operation in Sepencr upon crashes. Experimental results show that our Sepencr significantly reduces performance overheads compared with directly encrypting data in caches.

References

[1] J. Yi, M. Dong, F. Wu, and H. Chen, "Htmfs: Strong consistency comes for free with hardware transactional memory in persistent memory file systems," in *FAST*, 2022.
[2] "A. m. rudo. 2016. deprecating the pcommit instruction." https://software.intel.com/en-us/blogs/2016/09/12/deprecate-pcommit-instruction.
[3] M. Alshboul, P. Ramrakhyani, W. Wang, J. Tuck, and Y. Solihin, "BBB: Simplifying persistent programming using battery-backed buffers," in *HPCA*, 2021.
[4] L. Khati, N. Mouha, and D. Vergnaud, "Full disk encryption: bridging theory and practice," in *CT-RSA*, 2017.
[5] "Brief: Intel optane persistent memory." https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-dc-persistent-memory-brief.html.
[6] A. Awad, P. Manadhata, S. Haber, Y. Solihin, and W. Horne, "Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers," in *ASPLOS*, 2016.
[7] S. Liu, A. Kolli, J. Ren, and S. Khan, "Crash consistency in encrypted non-volatile main memory systems," in *HPCA*, 2018.
[8] K. A. Zubair and A. Awad, "Anubis: ultra-low overhead and recovery time for secure non-volatile memories," in *ISCA*, 2019.
[9] A. Awad, M. Ye, Y. Solihin, L. Njilla, and K. A. Zubair, "Triad-nvm: Persistency for integrity-protected and encrypted non-volatile memories," in *ISCA*, 2019.
[10] L. Wang and G. Xie, "A novel xor/xnor structure for modular design of qca circuits," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 12, pp. 3327–3331, 2020.
[11] P. Saravanan and P. Kalpana, "An energy efficient xor gate implementation resistant to power analysis attacks," *J. Eng. Sci. Technol*, vol. 10, no. 1, pp. 1275–1292, 2015.
[12] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, and S. Bharadwaj, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.
[13] S. Swami, J. Rakshit, and K. Mohanram, "Secret: Smartly encrypted energy efficient non-volatile memories," in *DAC*, 2016.
[14] 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 *MICRO*, 2018.
[15] X. Han, J. Tuck, and A. Awad, "Dolos: Improving the performance of persistent applications in adr-supported secure memory," in *MICRO*, 2021.
[16] C. Giraud, "Dfa on aes," in *AES*, 2004.
[17] "Introduction to digital logic design." https://trashworldnews.com/files/digital_logic_design.pdf.