Filesystem Encryption or Direct-Access for NVM Filesystems? Let's Have Both!

Kazi Abu Zubair
Electrical and Computer Engineering
North Carolina State University
Raleigh, North Carolina
kabuzub@ncsu.edu

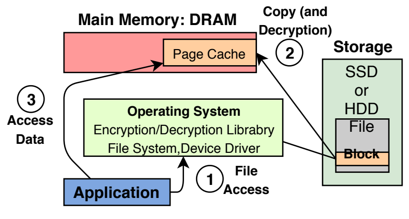
David Mohaisen
Computer Science
University of Central Florida
Orlando, Florida
mohaisen@ucf.edu

Amro Awad
Electrical and Computer Engineering
North Carolina State University
Raleigh, North Carolina
ajawad@ncsu.edu

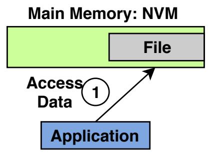
I. Introduction

Emerging Non-Volatile Memories (NVMs) [1] have low latency and data persistence capability. Therefore, they provide an opportunity to merge storage systems and main memory into a single concept. Such features allow them to be used as a byte-addressable persistent memory that can be directly accessed through Load/Store commands. In a legacy storage device (e.g., Solid State Drives of SSDs), the system software interrupts the file access, and pages are copied from the device into a software-managed page cache structure. The additional system software overhead for copying pages is negligible in this case compared to the access latency of current storage devices. However, such latency can easily dominate the access latency of emerging NVM devices. Due to their impressive speed of loading and storing data, it is desirable to remove as much system software overhead as possible from the load/store path. Additionally, the emerging NVMs can be accessed in finer granularity, and there is no need to copy pages elsewhere; they can be accessed directly using load and store operation. Modern Operating Systems support such direct access (DAX [2]) support for byte-addressable persistent memories, allowing us to utilize the full potential of such memories. Figure 1 illustrates the file I/O mechanism in the DAX-based filesystem and conventional filesystem. While DAX allows fast access to files, there is a *fundamental challenge* for using direct-access with NVM-based filesystems. Traditionally, filesystem encryption is implemented in the system software layer, which relies on the operating system to have control over file access. Such encryption method encrypts pages before evicting them from the page cache and decrypts them when copied to the page cache. Unfortunately, DAX completely removes such interactions with the page cache. Therefore, it is challenging to implement filesystem encryption while maintaining direct access to NVMs at the same time. In this work[1], we explore a low-cost solution to achieve transparent and fine-grained encryption for DAX files within the memory controller. To the best of our knowledge, this work is the first to propose any filesystem-level encryption for DAX. Our simulation results show that our scheme (FsEncr) incurs only 3.8% extra execution time overhead for several real-world benchmarks.

[1]This work has been published in the 28th IEEE International Symposium on High-Performance Computer Architecture (HPCA-28)



(a) The conventional way of accessing files.



(b) Accessing DAX-based files.

Fig. 1: Accessing files in conventional vs DAX-based filesystems.

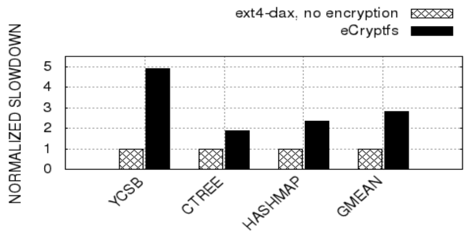


Fig. 2: Overheads of software based filesystem encryption.

II. Motivation

Current options for encrypting filesystem in NVMs are to either use software-based encryption (high overhead) or to modify applications to enable file encryption (impractical). To obtain more insights on this matter, we have simulated eCryptfs [3] over emulated persistent memory region (using `memmap`) in Gem5 [4] full-system simulation. We then observe the performance implications compared to plain ext4-dax. The overheads clearly show the slowdown caused by having a software level encryption mechanism. As shown in Figure 2, on average, the software encryption incurs **2.7x slowdown** for the benchmarks we have tested to compare Direct Access with software encryption. This leaves only two options for NVM operated in a direct access manner; no filesystem-level encryption, or no performance benefit for using DAX. Therefore, it is vital to have a method that can combine the benefit of both filesystem-level encryption and direct access.

III. Design

We allow the OS to communicate to the memory controller by writing to specific memory-mapped I/O registers on file creation and page faults. Whenever a new encrypted file is created, the kernel sends the key and associated File ID and Group ID to the memory controller. The memory controller stores the mapping inside a hardware structure named Open Tunnel Table (OTT). After opening the file, every first access to a page will trigger a page fault, and corresponding page table entry will be created. During each page fault, one specific bit (`DF-bit, or DAX-file bit`) is set in the physical address to identify encrypted file requests within the memory controller. Additionally, for DAX files, our scheme communicates to the memory controller to store file ID and Group ID within the file encryption metadata on every page fault.

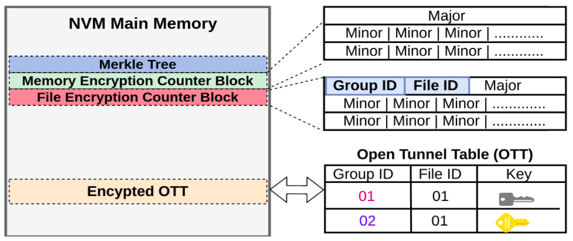

NVM Main Memory
Merkle Tree
Memory Encryption Counter Block
File Encryption Counter Block
Encypted OTT
Major
Minor | Minor | Minor | ....
Minor | Minor | Minor | ....
Group ID | File ID | Major
Minor | Minor | Minor | ....
Minor | Minor | Minor | ....
Open Tunnel Table (OTT)
Group ID | File ID | Key
01 | 01
02 | 01

Fig. 3: Encryption Metadata and OTT Table.

Figure 3 shows the encryption metadata organization and Figure 4 shows the read and write operation to enable transparent filesystem encryption. The memory controller identifies the encrypted DAX files from the request's physical address by extracting the `DF-bit`. If the request is only a typical memory request, only one level of encryption is done using the memory encryption key and Memory Encryption Counter Block (MECB). If the request is for encrypted DAX files, one additional level of encryption is done using a file key and File Encryption Counter Block (FECB). We use counter mode encryption where counters (MECB and FECB) are organized as split-counter [5] organization. FECB Block stores the File ID and Group ID along with the counters. Each MECB block in memory is followed by a FECB block. Therefore, if the request is identified to be a DAX file request, FECB is fetched, and the corresponding key from the OTT is searched using File ID and Group ID, and encryption/decryption is performed.

IV. Methodology and Results

We use Gem5 simulator [4] in its full-system mode to evaluate our scheme. We use a modified version of Linux Kernel 4.14 and a disk image based on Ubuntu 16.04 in our simulation. The kernel was initialized by configuring the 4GB starting from 12GB as a persistent region, using `memmap=4G!12G`. The persistent space is formatted with DAX-enabled ext4 filesystem, then mounted for use with persistent applications and libraries.

We use Whisper [6] and PMEMKV [7] benchmarks to measure the performance of our scheme. The PMEMKV-S benchmarks are configured to access 64B data, and PMEMKV-L benchmarks are configured to access 4096B on each query.

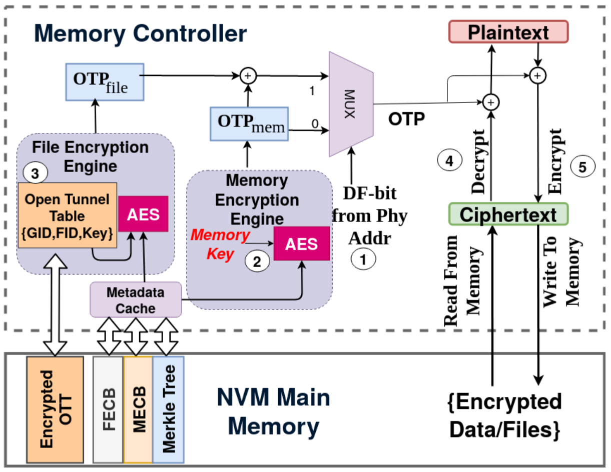


Fig. 4: FsEncr Read and Write Operation.

Figure 5 shows the slowdown normalized to the baseline ext4-dax with memory encryption only. On average, our scheme incurs only a 3.8% slowdown.

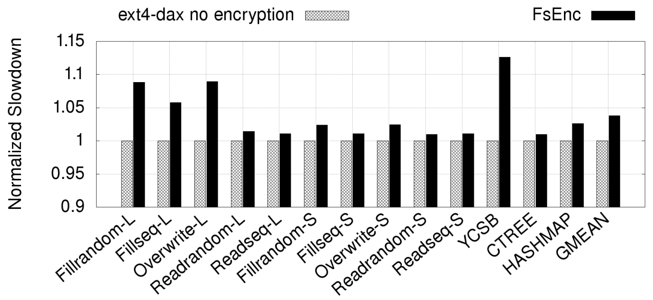


Fig. 5: Slowdown normalized to the baseline.

V. Conclusion

Direct Access (DAX) allows faster access to files for emerging NVMs. However, such feature makes it difficult to ensure filesystem security. In this work, we present a filesystem encryption scheme for emerging NVMs that can be enabled alongside direct-access with minimal changes. Our method offers the flexibility of DAX while maintaining file security with only 3.8% slowdown.

References

[1] Intel, "Intel 3DXpoint." "http://newsroom.intel.com/docs/DOC-6713", 2019. [Online; accessed 22-July-2019].
[2] Linux, "Linux Direct Access of Files (DAX)." "https://www.kernel.org/doc/Documentation/filesystems/dax.txt", 2019. [Online; accessed 22-July-2019].
[3] M. A. Halcrow, "ecryptfs v0. 1 design document," 2006.
[4] 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*, vol. 39, pp. 1–7, Aug. 2011.
[5] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2, pp. 179–190, 2006.
[6] S. Nalli, S. Haria, M. D. Hill, M. M. Swift, H. Volos, and K. Keeton, "An analysis of persistent memory use with whisper," in *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 135–148, ACM, 2017.
[7] "pmemkv." "https://github.com/pmem/pmemkv". [Online; accessed 05-February-2020].