A Wear-Leveling-Aware Counter Mode for Data Encryption in Non-Volatile Memories

Fangting Huang, Dan Feng*, Yu Hua, Wen Zhou
Wuhan National Laboratory for Optoelectronics
School of Computer, Huazhong University of Science and Technology, Wuhan, China
*Corresponding Author: dfeng@hust.edu.cn

***Abstract*—Counter-mode encryption has been widely used to resist NVMs from malicious attacks, due to its proved security and high performance. However, this scheme suffers from the counter size versus re-encryption problem, where per-line counters must be relatively large to avoid counter overflow, or re-encryption of the entire memory is required to ensure security. In order to address this problem, we propose a novel wear-leveling-aware counter mode for data encryption, called Resetting Counter via Remapping (RCR). The basic idea behind RCR is to leverage wear-leveling remappings to reset the line counter. With carefully designed procedure, RCR avoids counter overflow with much smaller counter size. The salient features of RCR include low storage overhead of counters, high counter cache hit ratio, and no extra re-encryption overhead. Compared with state-of-the-art works, RCR obtains significant performance improvements, e.g., up to a 57% reduction in the IPC degradation, under the evaluation of 8 memory-intensive benchmarks from SPEC 2006.**

I. Introduction

Due to higher scalability and non-volatility, NVMs hold great promises to replace DRAM to build the future main memory systems [1]. Despite of salient features, security is a serious challenge for using NVMs as the main memory.

Counter mode is a well-know encryption technique and adopted by a broad class of NVMs encryption schemes [2][3]. However, the counter-mode encryption suffers from the *counter size versus re-encryption problem* [4]. Small per-line counters overflow frequently, which requires to re-encrypt the entire memory and causes significant performance overheads. While using large per-counter size to avoid counter overflow results in large storage overheads either in on-chip cache or memory. We note that, compared with DRAM, the counter size versus re-encryption problem for NVMs is more severe for the following reasons: 1) the capacity of NVMs increases significantly which means that the storage overhead of counters is much heavier; 2) the re-encryption incurred by counter overflow has much greater impact on NVMs, since writing to NVM cells is much slower and requires large power consumption; 3) normal write traffic can be reduced by up to half via plenty of techniques while re-encryption writes is not likely be optimized [2].

On the other hand, due to the limited write endurance of NVMs, wear leveling is an inevitable component to make the non-uniform write traffic distribute evenly by remapping the heavily-written lines to the less ones periodically [5]. Typically, wear-leveling schemes divide the memory space into multiple regions and remap the memory lines one by one for every certain number of writes within its region [5][6][7].

To the best of our knowledge, existing counter-mode encryption techniques are all independent of wear-leveling techniques, i.e., not aware of the wear leveling. In this paper, we propose a novel wear-leveling-aware counter mode for data encryption in NVMs, called Resetting Counters via Remappings (RCR), which leverages the remapping operations of wear leveling to reset line counters periodically and thus reduce the line counter size required to avoid counter overflow. Specifically, we use a 64-bit region counter for per region to record the number of completed remapping rounds and increase the region counter by 1 when the region completes a remapping round. The per-line counters are used to store the number of writes to the lines since their last remappings, which are reset when the lines are remapped. Therefore, RCR limits the line counter size to be the maximum number of writes to a line before the line is remapped. It is worth noting that, by leveraging the writes incurred by remapping, the re-encryption caused by resetting line counter of RCR does not introduce extra read and write overhead. The experiment results show that RCR reduces the IPC degradation by 39%, 57% and 41%, compared with 32-bit monolithic counter mode, 64-bit monolithic counter mode and split counter mode, respectively.

The rest of this paper is organized as follows. In section II, we present the background. In section III, we present our method, i.e., Resetting Counter via Remappings (RCR). We provide the evaluation results in Section IV and conclude this paper in Section V.

II. Background

In this section, we introduce the counter-mode encryption which is widely used to enhance the security of NVMs and present a brief introduction to wear-leveling schemes.

A. Counter-Mode Encryption

In the counter-mode encryption, each memory line is associated with a counter, which is increased by 1 on each write to the line. The line address and counter are provided to the block cipher to generate a One Time Pad (OTP), which is then XORed with the plaintext for encryption, or with ciphertext for decryption. If the counter hits in the on-chip cache, OTP

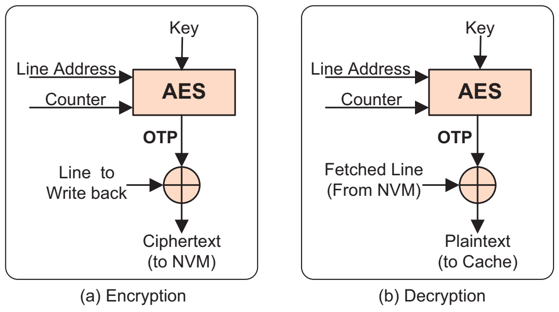

Fig. 1: Counter-mode encryption and decryption.

generation can be performed in parallel with the memory access of ciphertext, thus removing the decryption latency from the critical read path. As shown in Fig.1 (a), for a write back to NVMs, the line counter and the line address are used to generate the OTP, and the cache line is encrypted by being XORed with the OTP. To read data from NVMs, as shown in Fig.1 (b), the same OTP is used to decrypt the line. We refer this traditional counter-mode encryption as *monolithic counter-mode encryption*.

Unfortunately, the counter-mode encryption suffers from the counter size versus re-encryption problem [4]. To ensure security, the same pad will not be used more than once. Therefore, the line counter must be large enough to avoid counter overflow which requires to re-encrypt the entire memory. However, large counter size could not only incur large storage overheads for memory but also decrease the hit ratio of the counter cache. Split counter-mode encryption [8] divides memory lines into pages (each of which consist of several lines) and uses large per-page counters and small per-line counters: if any of the line counters overflows, the page counter is increased by 1, all the lines within the same page reset their line counters, and finally the whole page is re-encrypted. Although split counter-mode encryption reduces the storage overhead of counters and the granularity of the re-encryption, it could not eliminate the re-encryption overhead incurred by counter overflows yet.

B. Wear Leveling

Due to the limited write endurance, wear leveling is essential for NVMs, which extends the device lifetime by distributing writes evenly throughout the entire memory space. Typically, wear-leveling schemes divide the memory into multiple fixed-size regions, and perform wear leveling for each region independently. Every certain number of writes to the region induces a remapping movement to a line, for which its content is read from the old physical location and written to its remapped physical location. For example, RBSG moves a line to its neighborhood line every 100 normal writes to the region [5]. The lines of a region are remapped one by one. Once all the lines of the region have performed a remapping, one remapping round accomplishes. Such remapping round continues. The number of normal writes before triggering a remapping is called remapping interval.

III. Resetting Counters via Remapping

A. The Overview of RCR

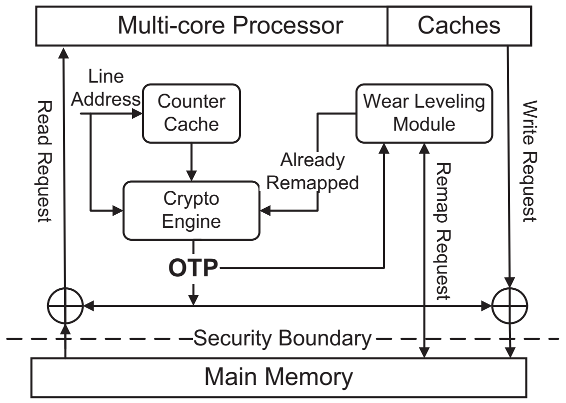

Fig. 2: Overview of secure processors with RCR.

The overview of secure processors with RCR is shown in Fig. 2. RCR is based on the observation that all memory lines within a region are remapped one by one by wear leveling periodically. Specifically, RCR uses small per-line counters to record the number of writes to each line since its last remapping. A line counter is increased by one for each normal write to the line and is reset when the line is remapped in the process of wear leveling. Moreover, to avoid the re-use of OTP, RCR employs a 64-bit per-region counter that records the number of the completed remapping rounds for a region, which also indicates the number of remappings that each line has performed. Since a region consists of thousands of lines, the overhead of region counter is negligible (e.g. several kilobytes per bank). We denote the value of the region counter as C_{region} and the value of the line counter as C_{line}. Both C_{region} and C_{line} are used together with the line address to generate OTP.

For each normal write-back to a specific line, the corresponding C_{line} is increased by 1. If the line has been remapped in the current remapping round, we use C_{region} plus 1 associated with C_{line} as the overall counter. Otherwise, we use C_{region} associated with C_{line}. Note that the status of whether a line has been remapped in the current remapping round can be obtained from the wear leveling module. The overall counter is concatenated with the line address and the key for encryption. In a region, if any of C_{line} overflows, C_{region} is increased by 1, all the lines within the same region reset their C_{line} and finally the lines are re-encrypted with the new overall counters. In the similar way, for a line fetched from NVMs, we obtain its overall counter according to the state whether the line has been remapped in the current remapping round or not. The overall counter and line address are used to decrypt the ciphertext which is read from NVMs.

B. The Remapping Process of RCR

To show how RCR works, an example of RCR remapping process is illustrated in Fig. 3. For a certain number of writes

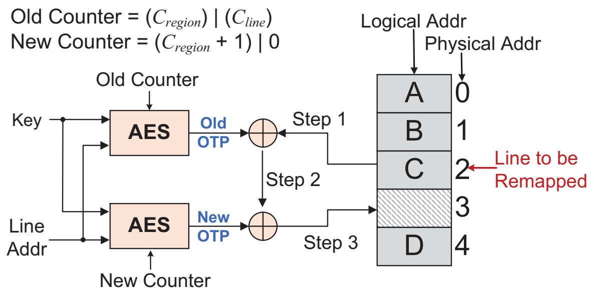

Fig. 3: An example of RCR remapping process.

to the region, a remapping is triggered and here the line to be remapped is line C. First, the ciphertext of line C is read out from its old location. Second, the old OTP is generated by concatenating C_{region}, C_{line} and the line address of line C as the input to AES. In the meantime, the new OTP is generated via the same operation, except that the old counter is replaced by the new counter that composed of its corresponding C_{region} plus one, and the reset C_{line} (i.e., zero). After that, the ciphertext is XORed with the old OTP to get the plaintext and the plaintext is XORed with the new OTP to get the new ciphertext. Finally, we write the new ciphertext to the new physical location of line C (i.e., line 3). The line to be remapped is re-encrypted with new counters in the process of wear leveling, which does not incur extra memory reads and writes. In addition, if the line to be remapped is the last un-remapped line in the current remapping round, the corresponding C_{region} is increased by 1. Therefore, the number of completed remapping rounds can be well recorded by C_{region}. We observe that, there is no re-used OTP in this procedure, thus providing security assurance.

We can see that, C_{line} is reset by RCR when performing remapping. Therefore, the line counter size needs only to be large enough to record the number of writes to the most heavily written line within a remapping round. In the worst case, all the writes of the region occur in the same line. The line counter size is equal to the number of write to a region within a remapping round, since all lines are remapped once within a remapping round. For the recommended configuration that the number of lines in a region is 2^{13} and the remapping interval is 64 [6], all the lines of a region are remapped within $2^{13} \times 64$ writes, thus 21-bit line counter is able to prevent counter overflow in the worst case. Fortunately, the write traffic of real-world applications is much more even and the size of line counter required is much smaller than the worst case. Therefore, we prefer to use a smaller line counter to avoid overflow for general applications and re-encrypt the region with the incremented C_{region} if any of C_{line} overflows. The appropriate size of line counter for RCR will be empirically studied in Section IV.

IV. Performance Evaluation

The experiments are conducted with *Gem5* [9], a full system cycle-accurate simulator. In the experiment platform, the system consists of an 8-core processor (4 GHz) with a private 32KB L1 cache, a shared 512 KB L2 cache and 1MB L3 cache, and an 32 MB L4 DRAM cache, similar with [2].We assume that: the total capacity is 16GB; the NVM read time and write time is 125 ns and 450 ns, respectively; the line size of all cache units is fixed as 64 bytes; the 256-bit AES latency is 96ns [10]. We also assume that the NVM main memory leverages a state-of-the-art wear-leveling scheme (i.e., RBSG [5]). We use 8 memory-intensive benchmarks (*gems, lbm, leslie3d, libq, mcf, milc, xalanc* and *zeusmp*) from SPEC2006 suite to comprehensively examine the system performance, in which all 8 CPU cores execute the same benchmark in parallel.

A. The Counter Size of RCR

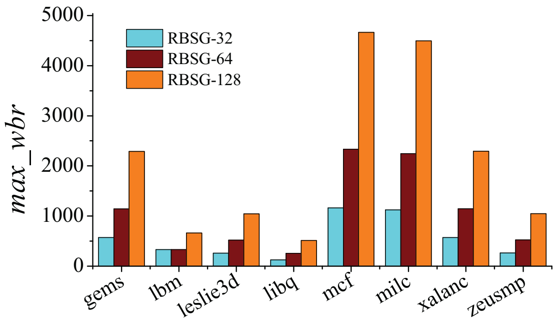

Fig. 4: The maximum of writes to a line before being remapped.

As discussed in Section III, the counter size of RCR is determined by the trade-off between the overflow re-encryption risk and the storage overhead. Ideally, we would like that the counter size of RCR can cover the maximum number of writes to any line before the line gets remapped (which is denoted as max_wbr), under general real-world applications and wear-leveling mechanism. In the experiments, we use the 8 memory-intensive benchmarks as the representative of real-world applications to evaluate the appropriate counter size.

We fast-forward 5 billion instructions and run the benchmarks repeatedly. During the procedure, for each line, the number of writes within every remapping round is recorded. At the end (note that all the regions have finished at least one remapping round), we take the maximum number of writes over all the lines and all the remapping rounds as max_mbr. We vary the number of regions and find that it has little impact on the results. We hence use 512 regions per bank by default. The results of varying the remapping interval (32, 64 and 128) are shown in Fig. 4. As we can see, max_wbr increases as the remapping interval increases, since the larger the remapping interval is, the more writes are needed for a region to trigger a remapping. Moreover, the *mcf* benchmark shows the highest max_wbr because of its uneven write traffic, which is 2,340 for the recommended configuration (the remapping interval as 64). As a result, 12-bit line counter are sufficient to avoid overflow. We also use 12 as the number of bits for the line counter in the following experiments.

B. Comparisons with Other Counter-Mode Encryption Schemes

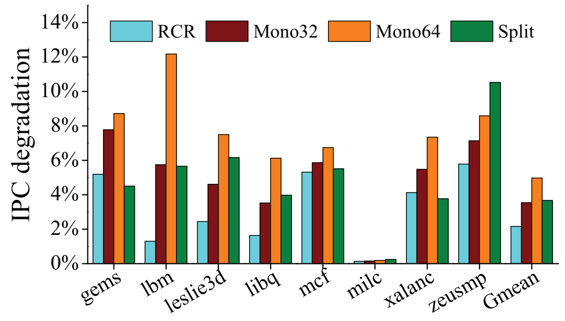

Fig. 5: Comparison of normalized IPC degradation.

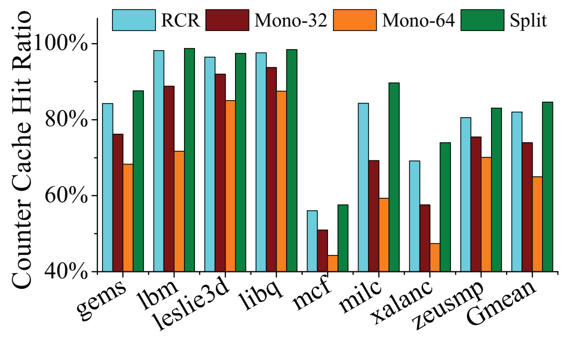

Fig. 6: Comparison of counter cache hit ratio.

To show the efficiency of RCR, we compare RCR with the monolithic counter-mode encryption that uses 32-bit or 64-bit line counters (denoted as $Mono32$ and $Mono64$, respectively), and the split counter-mode encryption (denoted as $Split$). The normalized instructions-per-cycle (IPC) degradation of the four schemes is shown in Fig. 5, where the baseline is a system without any memory encryption. The lower the IPC degradation is, the better the performance is. The geometric mean (Gmean) of the IPC degradation of RCR, $Mono32$ and $Mono64$ are 2.1%, 3.5% and 4.9%, respectively. That is, RCR reduces 39% and 57% of the IPC degradation compared with $Mono32$ and $Mono64$.

The improvements is owed to the much smaller counter size for RCR. We evaluate the counter cache hit ratio for the four schemes with 64KB counter cache and show the results in Fig. 6. The geometric mean of the cache hit ratio of RCR, $Mono32$, $Mono64$ and $Split$ are 82.1%, 73.9%, 64.9% and 84.5%, respectively. As we can observe, the cache hit ratio increases as the counter size decreases. With higher cache hit ratio, RCR gets higher IPC than $Mono32$ and $Mono64$, since more read requests can overlap their decryption with memory access. The $milc$ benchmark is somewhat different, for which plenty of memory requests are sent to memory in a short time, incurring very high read and write latency. Although $Split$ gets a slightly higher cache hit ratio than RCR, the Gmean of the IPC degradation of $Split$ is 3.7% and higher than that of RCR. This is because that $Split$ incurs extra memory operations due to the re-encryption of counter overflow, while RCR is able to avoid the write overhead of counter overflow. Under the 8 memory-intensive benchmarks, the re-encryption write overhead of $Split$ has a Gmean about 1.4% and is as high as 2.7% for $milc$.

V. Conclusion

Both encryption and wear leveling are key components of NVM systems. In order to address the counter size versus re-encryption problem of the widely-used counter-mode encryption, we proposed a wear-leveling-aware counter mode for data encryption, which synergizes encryption schemes and wear-leveling schemes. Unlike existing encryption schemes, the proposed RCR scheme, leverages the wear-leveling remappings to reset the line counter and avoids counter overflow with much smaller line counter size. We have demonstrated the efficiency of the proposed RCR and shown significant improvements upon the comparative schemes under eight memory-intensive benchmarks.

Acknowledgment

This work was supported by the National Key Research and Development Program of China under Grant 2016YFB1000202; the National High Technology Research and Development Program (863 Program) No.2015AA015301, No.2015AA016701; NSFC No.61303046, No.61472153, No.61502190, No.61232004; State Key Laboratory of Computer Architecture, No.CARCH201505; Wuhan Applied Basic Research Project (No.2015010101010004); This work was also supported by Engineering Research Center of data storage systems and Technology, Ministry of Education, China.

References

[1] S. Mittal and J. S. Vetter, "A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, pp. 1537–1550, 2016.

[2] V. Young, P. J. Nair, and M. K. Qureshi, "DEUCE: Write-Efficient Encryption for Non-Volatile Memories," in *ASPLOS*, 2015.

[3] S. Swami, J. Rakshit, and K. Mohanram, "SECRET: smartly EnCRypted energy efficient non-volatile memories," in *Design Automation Conference (DAC)*, 2016.

[4] M. Henson and S. Taylor, "Memory Encryption: A Survey of Existing Techniques," *ACM Computing Surveys (CSUR)*, pp. 53:1–53:26, 2014.

[5] 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 *MICRO*, 2009.

[6] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security Refresh: Prevent Malicious Wear-out and Increase Durability for Phase-change Memory with Dynamically Randomized Address Mapping," in *ISCA*, 2010.

[7] F. Huang, D. Feng, W. Xia, W. Zhou, Y. Zhang, M. Fu, C. Jiang, and Y. Zhou, "Security RBSG: Protecting Phase Change Memory with Security-Level Adjustable Dynamic Mapping," in *IPDPS*, 2016.

[8] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving Cost, Performance, and Security of Memory Encryption and Authentication," in *ISCA*, 2006.

[9] 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," *ACM SIGARCH Computer Architecture News*, 2011.

[10] W. Shi, H.-H. S. Lee, M. Ghosh, C. Lu, and A. Boldyreva, "High Efficiency Counter Mode Security Architecture via Prediction and Precomputation," in *ISCA*, 2005.