ACME: Advanced Counter Mode Encryption for Secure Non-Volatile Memories

Shivam Swami and Kartik Mohanram

Department of Electrical and Computer Engineering, University of Pittsburgh, PA

shs173@pitt.edu kartik.mohanram@gmail.com

ABSTRACT

*Modern computing systems that integrate emerging non-volatile memories (NVMs) are vulnerable to classical security threats to data confidentiality (e.g., stolen DIMM and bus snooping attacks) as well as new security threats to system availability (e.g., denial of memory service (DoMS) attacks). Although counter mode encryption (CME) secures NVM-based main memories against confidentiality attacks, counter sizing is critical to balance tradeoffs between memory overhead, system performance, and re-encryption frequency (i.e., system availability). Furthermore, CME is particularly vulnerable to DoMS attacks, where a malicious application can severely impact memory availability by forcing frequent full memory re-encryption. This paper proposes **Advanced Counter Mode Encryption, i.e., ACME**, a low overhead CME-based main memory encryption solution to realize the twin security goals of confidentiality and availability in NVM-based main memories. At its core, ACME integrates counter write leveling (CWL) to reduce the frequency of full memory re-encryption while preserving the security properties of the underlying CME. Our evaluations on a phase change memory (PCM) architecture using SPEC CPU2006 benchmarks show that for a system availability of 99.999%, ACME not only requires 50% lower counter overhead, but also improves system performance by 20% in comparison to classical CME. When subject to a DoMS attack in the form of an unprivileged Linux process that sidesteps all levels of cache to constantly write to the same memory address to precipitate counter overflow, the ACME-based system provides 99.9% system availability in contrast to a classical CME-based system that is rendered non-operational.*

CCS Concepts: Security and privacy → Security in hardware; Hardware attacks and countermeasures;

Keywords: Non-volatile Memories, Counter Mode Encryption, Denial of Memory Service, Phase Change Memory.

1. Introduction

Non-volatile memory (NVM) technologies such as phase change memory (PCM) and resistive RAM (RRAM) have emerged as low power, high density, scalable alternatives for DRAM in main memories [1–3]. Whereas data persistence in NVMs is essential to tolerate power/system failures [4, 5], improve checkpointing [6], and reduce application startup time [7], data persistence also exposes NVMs to a variety of data confidentiality attacks, e.g., stolen DIMM and bus snooping attacks. These security vulnerabilities pose major challenges to be overcome prior to wide-scale adoption of NVMs into commodity/high-performance computing systems.

This research was supported by NSF Award CCF-1217738.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DAC '18, June 24–29, 2018, San Francisco, CA, USA

© 2018 ACM. ISBN 978-1-4503-5700-5/18/06...\$15.00

DOI: <https://doi.org/10.1145/3195970.3195983>

State-of-the-art NVM security solutions [8–12] integrate counter mode encryption (CME) in the secure processor to thwart data confidentiality attacks. In CME, a counter is associated with each cache line; the counter, the cache line memory address, and a secret key are used to generate a one-time pad (OTP). The OTP is bitwise XORed with the cache line data (plaintext) to generate encrypted data (ciphertext), which is written to the memory; the cache line counter is incremented on every write to ensure temporal exclusivity of the OTP. During decryption, the same OTP is XORed with the ciphertext to recover the plaintext.

To reduce on-chip (i.e., processor) memory overhead in CME, the counters are stored in main memory and cached on-chip in a counter cache to improve performance [8–10, 12]. For a fixed counter cache size, the counter size should be small to improve the hit rate of the counter cache. Whereas small counters incur low memory overhead and improve system performance, they can quickly overflow in the presence of high write traffic. On a counter overflow, CME requires the entire memory to be re-encrypted with a new secret key, causing the system to freeze for the duration of full memory re-encryption [8, 9, 13].

Counter overflow renders CME vulnerable to the DoMS attack, which threatens system availability. In a DoMS attack, a malicious application can render the memory system unavailable to other applications by forcing frequent full memory re-encryption due to counter overflow. A DoMS attack can be easily engineered using cache eviction and ordering instructions (like `clflush` and `mfence`) that can be executed in non-administrator mode to constantly write to the same cache line in main memory, forcing its counter to overflow. Split-counter mode encryption (split-CME) [13] protects DRAM-based main memories against DoMS attacks; however, directly extending split-CME to NVM-based main memories undermines the ability of NVMs to ensure data recovery in the face of power/system failures (explained in section 3.3). Past work, e.g., MECU [4] and i-NVMM [5], has motivated that preserving the data recovery property of NVMs is an important design goal for any NVM security solution. Whereas system availability can also be improved by employing large counters, this increases the memory overhead of CME and results in poor performance due to frequent counter cache misses (for a fixed counter cache size).

This paper proposes **Advanced Counter Mode Encryption, i.e., ACME**, an NVM security solution to realize low overhead, high performance CME that is robust to both data confidentiality and system availability attacks. At its core, ACME leverages the underlying wear leveling architecture—employed to improve NVM endurance [1, 3, 14]—to perform counter write leveling (CWL). CWL associates counters with physical addresses (PAs) instead of logical addresses (LAs) such that when a frequently written cache line is translated from one PA to another PA for wear leveling, its associated counter is also remapped, leading to a distribution of writes across counters. Multiple counters together track the number of writes seen by a frequently written cache line, thereby delaying counter overflow; delaying counter overflow improves system availability due to delayed full memory re-encryption. ACME in-

curs no logic overhead, since it enables CWL by leveraging the existing wear leveling architecture that is integrated with the secure processor. ACME is a drop-in replacement for CME, and does not compromise the security of the underlying CME, i.e., ACME preserves CME requirements of (i) encryption inside a secure processor and (ii) spatial as well as temporal exclusivity of the OTP.

We evaluate ACME on a PCM architecture using (i) a trace-driven memory simulator that integrates the Intel Pin [15] binary instrumentation tool and (ii) the MARSS [16] full-system simulator on SPEC CPU2006 benchmarks [17]. Results show that for the system availability of 99.999%, ACME not only requires 50% lower counter memory overhead, but also improves system performance by 20% in comparison to classical CME. Further, to validate the robustness of ACME to availability attacks, we launched a DoMS attack on a Linux machine with an Intel Core i3 CPU and 16GB main memory. The attack invokes `clflush` and `mfence` instructions to sidestep all levels of cache and constantly writes to the same LA until the counter associated with that LA overflows. We observed that when subject to the DoMS attack, the ACME-based system provides 99.9% availability in contrast to a classical CME-based system that is rendered non-operational.

Section 2 provides necessary background in memory security and motivates ACME. Section 3 describes the theory and architecture of ACME. Section 4 presents evaluation and results, section 5 covers related work, section 6 discusses integration of ACME with state-of-the-art CME solutions, and section 7 is a conclusion.

2. Background and Motivation

This section covers the NVM threat model and the role of counter mode encryption (CME) as the NVM security solution. CME trade-offs are also discussed in depth to motivate ACME.

2.1 Threat Model

As in prior studies on memory security, our trusted computing base (TCB) consists of the processor and core parts of the operating system (e.g., security kernels); the external memory and peripherals are assumed to be untrusted [4, 10, 11]. Along these lines, recent research [4, 5, 10, 11] on NVM security has recognized the stolen DIMM and bus snooping attacks as the two most common security attacks to data confidentiality in NVMs. More recently, with the advent of row-hammer-like attacks [18], NVM-based main memories have also become vulnerable to denial of memory service (DoMS) attacks that threaten system availability; to the best of our knowledge, there are no known solutions that explicitly address DoMS attacks in the context of NVMs.

Recent research [8, 9, 11, 19, 20] advocates the use of counter mode encryption (CME) as a secure memory encryption technique that is robust against data confidentiality attacks. Although CME ensures data confidentiality, it can be exploited by DoMS attacks to render the memory non-operational. Note that in addition to data confidentiality and system availability, data integrity is necessary for complete NVM security. Along state-of-the-art InvisiMem [21] and Intel SGX [22], we employ isolated execution of enclaves to thwart data integrity attacks (i.e., spoofing/splicing/replay attacks).

2.2 Counter Mode Encryption (CME)

In CME, a block cipher is used to encrypt a seed with a secret key (stored on the processor) to produce a one-time pad (OTP). This OTP is bitwise XORed with the plaintext to generate the ciphertext. During decryption, the same OTP is XORed with the ciphertext to obtain the plaintext. The spatial and temporal exclusivity of the OTP is critical for the security of CME and requires that (i) the OTPs should be unique for different cache lines and (ii) the OTPs for a particular cache line should be unique for every write.

These unique OTPs are generated from unique seeds that have two components: (i) the cache line address (to ensure spatial exclusivity) and (ii) a cache line counter, which is incremented on each write (to ensure temporal exclusivity). Note that the cache line address used for OTP generation can be a virtual address (VA, i.e., the address generated by the CPU) or a physical address (PA, i.e., the address generated after virtual to physical address translation by the memory management unit, i.e., MMU). In [8, 13, 23] it was argued that VAs are more difficult to support because they are not directly available at the lowest-level on-chip cache, and different processes use the same VAs, which can lead to the reuse of OTPs across different processes. In contrast, PAs are easier to use, but require book-keeping for reliable decryption of the pages that are moved between main memory and the disk. Further, advanced cryptographic primitives (for DRAM-based main memories), e.g., oblivious RAM [24] also use PAs for encryption. Along these lines, this work assumes that the PA is used for OTP generation.

To reduce on-chip memory overhead in CME, the counters are stored in main memory and cached on-chip in a counter cache to improve performance [9, 12, 20]. To increase the counter cache hit rate, small counters are preferred in practice, since a fixed size counter cache can store more counters if each counter is small. However, small counters can quickly overflow in the presence of high memory write traffic. Conventionally, counter overflow is handled by changing the secret key to prevent reuse of OTPs [8, 13]. However, since the same secret key is shared by every cache line, a change of secret key requires the entire memory to be re-encrypted, causing the system to freeze for the duration of full memory re-encryption. This limitation of CME can be exploited in a DoMS attack to render a memory system non-operational by forcing frequent full memory re-encryption. A DoMS attack can be easily engineered using cache eviction and ordering instructions (like `clflush` and `mfence`) that can be executed without any administrative privileges to constantly write to the same cache line in main memory, forcing its counter to overflow. Past work [25] on NVM security has shown that counter overflow can freeze a 16GB PCM memory system for ≈ 1 minute, which is unacceptable in practice.

To reduce counter overflow and increase system availability, large counters are used in CME, since they do not overflow frequently. However, large counters increase the memory overhead of CME and result in poor system performance due to frequent counter cache misses. Thus, CME imposes heavy overheads on memory, performance, and system availability in practice.

3. Advanced Counter Mode Encryption

This section describes Advanced Counter Mode Encryption, i.e., ACME, a CME-based low overhead main memory encryption solution that performs counter write leveling (CWL) to reduce full memory re-encryption frequency, without compromising the security of the underlying CME. Note that ACME is agnostic to the underlying memory technology, i.e., ACME is applicable to single-/multi-/triple-level cell (SLC/MLC/TLC) NVMs.

3.1 ACME: Observation

Memory access locality in real-world applications causes some of the memory addresses to be written much more frequently than others. Figure 1 reports the write-back, i.e., main memory write distribution of the SPEC CPU2006 benchmarks [17]. The table reports the maximum writes seen by a memory address (in main memory) for various benchmarks; the median writes for different addresses are also reported. We observe that on average, 68% addresses are written 1-2 $\times$, whereas only 2.7% addresses are written > 64 $\times$ during multi-billion-instruction simulations of these benchmarks. *Due to this disparity in write-back distribution between ad-*

Figure 1: **Write-back (i.e., main memory write) distribution for SPEC CPU2006 benchmarks [17].** 68% of the total addresses written receive < 2 writes and only 2.7% of the total addresses written receive > 64 writes. Due to this disparity in write-back distribution between addresses, a majority of counters in CME remain underutilized and never reach their terminal count in practice.

addresses, counters belonging to only a few frequently written cache lines reach their terminal count forcing full memory re-encryption, whereas a majority of counters remain underutilized and never reach their terminal count in practice.

3.2 ACME: Design

In order to delay counter overflow (i.e., system freeze) due to the disparity in write distribution between addresses and improve counter utilization, we propose counter write leveling (CWL) by re-assigning underutilized counters to frequently written cache lines. This results in a uniform growth across counters, thereby delaying full memory re-encryption due to counter overflow and improving system availability. To perform CWL, ACME leverages the inbuilt wear leveling logic present on the secure processor. Note that in practice, the wear leveling logic is integrated with the secure processor to thwart wearout attacks [14] on NVMs.

Since NVMs suffer from low endurance, it is common to perform memory wear leveling to uniformly distribute writes across memory, which prevents the more frequently written memory cells from failing early in comparison to other cells. Region-based Start-Gap (RBSG) [14] is a state-of-the-art wear leveling technique that divides the memory into multiple fixed size regions (Start-Gap regions), and performs wear leveling for each region independently. For insignificant memory overhead (one spare cache line termed the GapLine per 2^{18} cache lines in a region), RBSG achieves $\approx 97\%$ wear leveling, i.e., 97% of total writes to main memory are uniformly spread across all memory addresses. This work assumes that RBSG is implemented in memory.

Wear leveling introduces an additional level of indirection in the address translation process. A VA generated by the CPU is first translated to an intermediate or logical address (LA) by the MMU. In DRAM-based main memory, the LA is the final PA where the cache line is stored in the memory. However, due to wear leveling in NVMs, the LA of a cache line is further translated by the wear leveling unit to a new PA where the cache line is actually stored in the memory. Figure 2 illustrates address translation in NVMs.

Since wear leveling maps an LA to different PAs over time, employing the PA for OTP generation can lead to the OTP reuse problem. For example, OTP reuse will occur if an LA with a certain counter value (conventionally, a counter is associated with each cache line, i.e., LA) is translated to a new PA that has already been seen that counter value with some other LA. Figure 3 illustrates the OTP reuse problem if the PAs are used for OTP generation when counters are associated with the LAs. At time t_1 , cache line with LA ‘D’ is stored at PA 40. This cache line is encrypted using an OTP generated from PA 40 and counter 120. At time t_2 , cache line ‘D’ has been mapped to PA 50 and the GapLine occupies PA 40.

Figure 2: **Address translation due to wear leveling in NVMs.**

Figure 3: **Illustration of OTP reuse, when the PA is used for OTP generation and the counters are associated with LAs.**

Additionally, counter values have been updated to reflect the total writes seen by each cache line. At time t_3 , cache line ‘C’ is mapped to PA 40. Since the current counter value of cache line ‘C’ is 120, the OTP used in encrypting cache line ‘C’ is the same OTP that was used for encrypting cache line ‘D’ at time t_1 . This is a security breach that can be exploited to execute known-plaintext attacks (KPA) [8, 9, 13]. To avoid such scenarios, state-of-the-art NVM encryption solutions [11, 12, 25] decouple CME from wear leveling by generating the OTP using the LA instead of the PA.

In contrast, ACME employs the PA for OTP generation and overcomes the OTP reuse problem by employing physical counters that do not translate with the LA, i.e., ACME assigns a counter per 512-bit physical memory location; for every write to a PA, its counter is incremented by 1. Thus, for a given PA, a counter value is never repeated, thwarting KPAs. Additionally, when a frequently written cache line is translated from one PA to another PA due to wear leveling, its associated counter is also changed, leading to a distribution of writes across counters, i.e., counter write leveling (CWL). Figure 4 contrasts CME and ACME. When cache line ‘D’ is translated from PA 40 to PA 50 using CME (figure 4(a)), its associated counter, i.e., CTR_D remains unchanged and continues to grow with writes to cache line ‘D’, resulting in early overflow. In contrast, when cache line ‘D’ is translated from PA 40 to PA 50 using ACME (figure 4(b)), its associated counter is changed from CTR_{40} to CTR_{50} . This results in sharing of writes to cache line ‘D’ between CTR_{40} and CTR_{50} . CWL thus increases the time to counter overflow, thereby delaying the time to full memory re-encryption.

Figure 4: **This figure contrasts OTP generation of CME and ACME, in the presence of Region-Based Start-Gap (RBSG) wear leveling. (a) OTP generation for CME remains unaffected after cache line ‘D’ is remapped. (b) OTP generation for ACME requires the new PA and its corresponding counter value after cache line ‘D’ is remapped. In this work, the GapLine moves after every ψ writes ($\psi = 64$).**

Note that ACME does not require any changes to the wear leveling unit in the host memory controller. While ACME uses the PA generated by the existing wear leveling unit for encryption/decryption, ACME does not interfere with the existing wear leveling architecture, i.e., wear leveling proceeds as-is in both the baseline (CME) and ACME scenarios. ACME thus ensures that wear leveling remains transparent to the host memory controller.

3.3 ACME: Memory Organization

Conventionally, CME stores the ciphertext and counters separately in main memory such that counters belonging to consecutive LAs are stored in consecutive physical locations [9, 13, 20]. This memory organization facilitates prefetching of multiple counters that are consecutive in the logical address space. Since real-world applications exhibit high spatial locality, prefetching consecutive LA counters results in a high counter cache hit rate. For a 512-bit counter cache block and n -bit counter, each counter cache miss fetches $\lceil 512/n \rceil$ counters to the counter cache.

Furthermore, since the counter cache is implemented as a write-back cache, a counter is updated in the main memory only when its respective counter cache block is evicted from the counter cache. This delay in updating the counters renders the ciphertext and counters partially unsynchronized in the main memory, and a native CME memory system cannot support data recovery in the face of power/system failures¹. Past work [4, 5] has motivated that preserving data recovery is an important design goal for any NVM security solution. Whereas employing a write-through counter cache preserves the data recovery property of NVMs, it doubles the number of memory writes, since each data write requires an additional write to update the counter. Since NVM writes are $2\text{--}3\times$ slower than NVM reads, a $2\times$ increase in memory writes significantly reduces system performance.

To ensure consistent counter updates and support data recovery, ACME stores counters alongside their respective ciphertext as a single memory block. Whereas CME requires separate memory accesses to fetch/update the ciphertext and counter, ACME enables the ciphertext and its counter to be simultaneously fetched or updated in a single memory access. Since ACME employs a 24-bit counter (refer section 4.1) per 512-bit cache line, and the width of a typical data bus is 64 bits, 9 data beats are required to transfer data and a 24-bit counter between the NVM DIMM and the processor-side memory controller (ignoring the transfer of error correction meta-data [3] for simplicity). For a DDR3-1600 data bus connected to a PCM-based memory, an increase of 1 data beat per transfer increases effective memory access latency by $< 2\%$ [1]. In contrast, a separate memory access (as seen in native CME memory systems) increases the effective memory access latency by 100%. Hence, storing the ciphertext and counters alongside preserves data recovery without impacting performance. Note that the changes required for ACME – 1 extra data beat during data transfer and colocated counter and ciphertext – are minor in comparison to the performance and data recovery benefits afforded by ACME. These requirements can easily be fulfilled using advanced memory systems such as Samsung high-bandwidth memory (HBM), Micron hybrid memory cube (HMC), Intel Optane, etc.

3.4 ACME: Memory Operations

Read: On a read, both CME and ACME perform counter cache lookup to retrieve the counter value of the desired cache line. On a hit, the OTP generation and data fetch from memory proceed in parallel; on a miss, the OTP generation is delayed until the required counter is fetched on-chip. Whereas CME requires a separate read access to fetch the required counter from the memory, ACME fetches the required counter along with the data using only 1 extra data beat. As a result, the OTP generation for ACME starts immediately after the 9th data beat. In contrast, the OTP generation for CME is delayed by 1 extra main memory read cycle.

¹ CME was originally proposed for DRAM-based main memories, which are volatile and not expected to recover data in the face of power/system failures. This allows the ciphertext and counters to be left unsynchronized in the main memory until the ciphertext is swapped out to the disk.

Write: On a write, CME increases the counter value of the LA of a cache line; ACME increases the counter value of the PA. Thus, when a cache line is translated to a new PA due to wear leveling, it involves: (i) reading the cache line to the processor-side memory controller, (ii) re-encrypting the cache line with the new OTP, and (iii) writing the cache line to the new memory location. However, it is important to note that steps (i) and (iii) are also performed in conventional CME due to wear leveling. Furthermore, step (ii) does not add any latency because generation of the old and new OTPs can be overlapped with step (i) by caching the counters (in the already provisioned counter cache) corresponding to the old/new PAs (already known to the wear leveling architecture [14]). Therefore, ACME does not introduce any extra latency in comparison to CME.

3.5 ACME: Security

The security of CME against confidentiality attacks such as bus snooping and stolen DIMM requires that (i) the encryption is performed inside a secure processor and (ii) a given OTP is never reused. The modifications introduced by ACME do not compromise these requirements, ensuring that ACME is as secure as CME.

The robustness of ACME to availability attacks (e.g., DoMS) can be studied by analyzing how soon malicious code can render the memory system unavailable to other applications by forcing full memory re-encryption. For example, a simple DoMS attack can repeatedly write to the same LA by employing the `clflush` instruction to sidestep the last-level cache (LLC) and `mfence` instruction to clear the processor's fill-buffer². The following C++ code was used in our experiments to execute the DoMS attack. We verified this code by monitoring the store operations using hardware performance counters for Linux [26]. Note that this code does not require any administrative privileges to execute any of its instructions.

```
1  size_t x = 0;
2  while(1) {
4    x = x + 1;
4    asm volatile ("clflush (%0)" :: "r"(&x));
5    asm volatile ("mfence");
6  }
```

Repeatedly writing to the same cache line (i.e., LA) causes the counter for that cache line to overflow in classical CME, forcing full memory re-encryption. However, since ACME distributes the writes to a cache line across multiple counters, it is able to sustain memory availability even when the memory is subject to this DoMS attack. Availability analysis of various CME- and ACME-based systems to the above DoMS attack is provided in section 4.3.

4. Evaluation and Results

We evaluate and compare ACME to CME on a single-level cell (SLC) PCM architecture using both integer and floating point workloads from the SPEC CPU2006 benchmark suite [17]. Our evaluations comprise (i) trace-based simulations to evaluate system availability and (ii) system-level simulations using the MARSS full-system simulator [16] to evaluate system performance.

Trace-driven simulation: We used an in-house trace-based simulator (which models RBSG wear leveling) to track the growth rates of the fastest growing counter for each benchmark. The counter growth rates are used to estimate the fastest counter overflow rate, i.e., full system re-encryption frequency in long-running applications. Multi-billion instruction memory traces were generated using the Intel Pin [15] binary instrumentation tool.

² The researchers at Google Project Zero [18] documented equivalent row-hammer attacks on DRAM.

Figure 5: Time to counter overflow of 24-bit ACME and various CME schemes for the SPEC CPU2006 benchmarks. Results show that despite its smaller counter size, 24-bit ACME increases the time to counter overflow in comparison to 36-bit CME.

Full-system simulation: MARSS [16] is configured to simulate a standard 4-core out-of-order system running at 3GHz. Further, each core is assigned a private L1 instruction and data cache of 32kB each and a private L2 cache of 128kB; the L3 cache (LLC in this case) is a single, shared write-back cache of 8MB. Finally, the main memory is modeled as a 16GB, 1 rank, 8-bank, single DDR3-1600 channel SLC PCM with a read latency of 60ns and a write latency of 150ns [1]. We also integrate a 512kB 32-way set-associative counter cache [25] with the processor-side memory controller for all the techniques evaluated in this work. The OTP generation latency of a 128-bit AES crypto-engine is 72ns [19,25].

Workloads: To evaluate system performance in real-world scenarios, we use 5 composite workloads with each workload containing 4 SPEC CPU2006 [17] benchmarks. Each core runs one benchmark, forming multi-programming workloads operating in different virtual address spaces. Table 1 lists these workloads along with their corresponding LLC misses per kilo-instruction (MPKI) values as reported by MARSS [16]. For fairness, we evaluate a variety of workloads with different (high/medium/low) MPKI values.

Workload	Benchmarks	MPKI
WD ₁	perlbench, soplex, bwaves, lbm	21.05
WD ₂	perlbench, soplex, milc, povray	14.20
WD ₃	h264ref, astar, milc, lbm	20.40
WD ₄	h264ref, sjeng, bwaves, povray	28.22
WD ₅	perlbench, leslie3d, GemsFDTD, lbm	12.71

Table 1: SPEC CPU2006 [17] composite workloads.

4.1 System Availability

In order to determine an appropriate counter size for ACME, we first computed the maximum memory writes per address for each SPEC CPU2006 benchmark. Assuming a constant counter growth rate, we extrapolated our simulation results to a system that continuously writes back to the memory via a DDR3 data bus until the fastest growing counter overflows. We observed that 24 bits is the smallest counter size that can ensure 99.999% system availability for ACME. Note that a similar approach was followed by the authors in [13] to determine the counter overflow rate.

Figure 5 contrasts time to counter overflow of ACME with different CME schemes. Our results show that for a majority of applications, CWL extends the time to counter overflow (i.e., reduces counter overflow frequency) of ACME. However, for applications like bwaves, mcf, zeusmp, leslie3d, and sjeng, where the disparity in counter growth is not significant, the large counters overflow later. Hence, for these applications, the time to counter overflow for 36-bit CME is longer in comparison to ACME. On average, the time to counter overflow of ACME is 4.87×10^7 seconds, i.e., ACME sees one full memory re-encryption every 564 days, which is the highest in comparison to other CME schemes. Since full memory re-encryption freezes the system for ≈ 1 minute (refer section 2.2), ACME achieves a system availability of 99.999%. In

contrast, CME with equivalent memory overhead, i.e., 24-bit CME results in only 99.3% system availability, which may be unacceptable for web/database servers in practice. Whereas 36-bit CME achieves 99.999% system availability, it requires 50% additional counter memory in comparison to ACME (24 bits/line for ACME versus 36 bits/line for CME).

4.2 System Performance

We compare the system performance of 24-bit ACME with 36-bit CME, since both schemes ensure 99.999% system availability. We evaluate two 36-bit CME architectures: (i) conventional CME, wherein the ciphertext and counters are stored in separate memory locations, and (ii) modified CME, wherein counters are stored alongside the ciphertext (similar to ACME) to enable concurrent counter updates (CCU). Note that all three techniques maintain updated ciphertext and counters for post-crash data recovery. Also, all three techniques employ RBSG wear leveling, and hence equivalent LA to PA translation overhead.

Figure 6: Comparison of IPC (normalized to ideal CME) of 24-bit ACME with 36-bit conventional CME, and 36-bit CME with concurrent counter updates (CCU).

Figure 6 reports the system performance in instructions per cycle (IPC), normalized to ideal CME, which does not incur OTP generation latency. We observe that 24-bit ACME improves IPC by 20% and 3.8% in comparison to 36-bit conventional CME and 36-bit CME+CCU, respectively. The high IPC of 24-bit ACME results from (i) the reduced write overhead of concurrent counter updates (refer section 3.3) and, (ii) the use of smaller counters (24 bits vs. 36 bits), which improves the counter cache hit rate (note that the counter cache size is constant for fairness of comparison), thereby lowering OTP generation latency. The IPC gains from the high counter cache hit rate are prominent for the high MPKI workloads (e.g., WD₄) that frequently access the counter cache.

4.3 Denial of Memory Service (DoMS) Attack

To validate the robustness of ACME to availability attacks, we launched a DoMS attack on a Linux machine with an Intel Core i3 CPU and 16GB main memory. The attack invokes `clflush` and `mfence` instructions (using the code in section 3.5) to sidestep all levels of cache and constantly writes back to the same memory address until the counter associated with that address overflows.

For a system under a DoMS attack, figure 7 compares the time to counter overflow of ACME and CME for different counter sizes.

We observe that the time to counter overflow for 24-bit ACME is 62911 seconds or 17.47 hours for a system availability of 99.9%. In contrast, the time to counter overflow for 24-bit CME is less than a second, which corresponds to a largely non-operational system. Due to CWL, ACME effectively increases the width of a counter by $\log_2 n$, where n is the number of counters in a Start-Gap region. Since we allocate 2^{18} cache lines (and hence 2^{18} counters) per Start-Gap region, the effective width of a 24-bit counter in ACME is $\log_2(2^{24} \times 2^{18})$, i.e., 42 bits. Thus, 24-bit ACME is equivalent to 42-bit CME in terms of system availability, and the results in Fig. 7 demonstrate excellent agreement between theory and practice.

5. Related work

Early solutions for NVM security, e.g., i-NVMM [5], performed memory-side encryption, and hence were ineffective against the bus snooping attack. To address this vulnerability, state-of-the-art NVM security solutions, e.g., DEUCE [11], SECRET [12], and COVERT [25] perform encryption inside the secure processor-side memory controller. Whereas these solutions are secure against data confidentiality attacks (e.g., stolen DIMM and bus snooping), due to the underlying CME, these techniques are still vulnerable to the system freeze problem due to counter overflow.

Split-CME [13] secures DRAM-based memories against data confidentiality attacks while ensuring high system availability. However, split-CME employs CME memory organization, where the ciphertext and counter are stored in separate memory regions. Hence, split-CME incurs high performance overhead to support consistent counter updates in NVMs (as explained in section 3.3). Note that split-CME cannot store the counters alongside the ciphertext due to its complicated dual counter design, where a global counter is shared by all the lines in a page; hence, the evaluation of split-CME with ACME is beyond the scope of this work.

6. ACME + state-of-the-art in NVM security

ACME is a drop-in replacement for CME in state-of-the-art NVM security solutions like DEUCE [11], SECRET [12], and COVERT [25]. We integrated and evaluated ACME with these techniques and summarize the results here for brevity. Results show that DEUCE+ACME (i) reduces memory overhead by 14.2% (28-bit counter per cache line for DEUCE vs. 24-bit counter per cache line for DEUCE+ACME), (ii) improves system availability by 300× (because ACME replaces DEUCE’s logical counters with physical counters and performs CWL to delay counter overflow), and (iii) improves IPC by 1% in comparison to DEUCE (due to improved caching of the small counters of DEUCE+ACME in the counter cache). Note that since SECRET extends DEUCE to only address write energy/latency of the ciphertext without modifying the underlying encryption technique (i.e., conventional CME), SECRET+ACME registers similar improvements to DEUCE+ACME. Finally, COVERT employs idle error correcting pointers (ECPs) to extend an overflowing counter. However, since the availability of idle ECPs reduces as the memory ages, the ability of COVERT to delay counter overflow decreases with time. Integrating ACME with COVERT improves COVERT’s system availability (during the initial years of operation) from 99.3% to 99.999% for no overhead. The system availability evaluations of ACME+COVERT for different phases of memory lifetime are excluded for brevity.

7. Conclusions

ACME is a low overhead CME-based encryption solution to realize the twin security goals of confidentiality and availability in NVMs. ACME employs counter write leveling to reduce full memory re-encryption frequency due to counter overflow, without compromising the security of the underlying CME. ACME is the first

Figure 7: Time to counter overflow versus counter size for ACME and CME in the presence of DoMS attack.

work to study and propose a solution for the DoMS attack in the CME-based NVMs. In comparison to CME, ACME incurs 50% lower counter memory overhead and improves system performance by 20%. Further, when subject to a DoMS attack, the ACME-based system provides 99.9% system availability in contrast to a classical CME-based system that is rendered non-operational.

8. References

- [1] B. C. Lee *et al.*, “Architecting phase change memory as a scalable DRAM alternative,” in *Proc. Intl. Symposium on Computer Architecture*, 2009.
- [2] H. Akinaga and H. Shima, “Resistive random access memory (ReRAM) based on metal oxides,” *Proceedings of the IEEE*, 2010.
- [3] S. Swami and K. Mohanram, “Reliable non-volatile memories: Techniques and measures,” *IEEE Design & Test*, 2017.
- [4] W. Enck *et al.*, “Defending against attacks on main memory persistence,” in *Proc. Annual Computer Security Applications Conference*, 2008.
- [5] S. Chhabra and Y. Solihin, “i-NVMM: A secure non-volatile main memory system with incremental encryption,” in *Proc. Intl. Symposium on Computer Architecture*, 2011.
- [6] X. Dong *et al.*, “Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exascale systems,” in *Proc. Conference on High Performance Computing Networking, Storage and Analysis*, 2009.
- [7] H. Kim *et al.*, “Accelerating application start-up with non-volatile memory in android systems,” *IEEE Micro*, 2015.
- [8] G. E. Suh *et al.*, “Efficient memory integrity verification and encryption for secure processors,” in *Proc. Intl. Symposium on Microarchitecture*, 2003.
- [9] J. Yang *et al.*, “Improving memory encryption performance in secure processors,” *IEEE Transactions on Computers*, 2005.
- [10] J. Kong and H. Zhou, “Improving privacy and lifetime of PCM-based main memory,” in *Proc. Intl. Conference on Dependable Systems and Networks*, 2010.
- [11] V. Young *et al.*, “DEUCE: Write-efficient encryption for non-volatile memories,” in *Proc. Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.
- [12] S. Swami, J. Rakshit, and K. Mohanram, “SECRET: Smartly encrypted energy efficient non-volatile memories,” in *Proc. Design Automation Conference*, 2016.
- [13] C. Yan *et al.*, “Improving cost, performance, and security of memory encryption and authentication,” in *Proc. Intl. Symposium on Computer Architecture*, 2006.
- [14] M. Qureshi *et al.*, “Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling,” in *Proc. Intl. Symposium on Microarchitecture*, 2009.
- [15] C.-K. Luk *et al.*, “Pin: Building customized program analysis tools with dynamic instrumentation,” in *Proc. Conference on Programming Language Design and Implementation*, 2005.
- [16] A. Patel *et al.*, “Marss-x86: A Qemu-based micro-architectural and systems simulator for x86 multicore processors,” in *Proc. Design Automation Conference*, 2011.
- [17] J. L. Henning, “SPEC CPU2006 benchmark descriptions,” in *ACM SIGARCH Computer Architecture News*, 2006.
- [18] <https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html>.
- [19] W. Shi *et al.*, “High efficiency counter mode security architecture via prediction and precomputation,” in *Proc. Intl. Symposium on Computer Architecture*, 2005.
- [20] J. Lee *et al.*, “Reducing the memory bandwidth overheads of hardware security support for multi-core processors,” *IEEE Transactions on Computers*, 2016.
- [21] S. Aga and S. Narayanasamy, “InvisiMem: Smart memory defenses for memory bus side channel,” in *Proc. Intl. Symposium on Computer Architecture*, 2017.
- [22] <https://software.intel.com/en-us/sgx>.
- [23] B. Rogers *et al.*, “Using address independent seed encryption and Bonsai Merkle Trees to make secure processors OS- and performance-friendly,” in *Proc. Intl. Symposium on Microarchitecture*, 2007.
- [24] E. Stefanov *et al.*, “Path ORAM: An extremely simple oblivious RAM protocol,” in *Proc. Intl. Conference on Computer & Communications Security*, 2013.
- [25] S. Swami and K. Mohanram, “COVERT: Counter overflow reduction for efficient encryption of non-volatile memories,” in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, 2017.
- [26] <https://perf.wiki.kernel.org/index.php/Main-Page>.