DEUCE: Write-Efficient Encryption for Non-Volatile Memories

Vinson Young Prashant J. Nair Moinuddin K. Qureshi

School of Electrical and Computer Engineering
Georgia Institute of Technology
{vyoung,pnair6,moin}@ece.gatech.edu

Abstract

Phase Change Memory (PCM) is an emerging Non Volatile Memory (NVM) technology that has the potential to provide scalable high-density memory systems. While the non-volatility of PCM is a desirable property in order to save leakage power, it also has the undesirable effect of making PCM main memories susceptible to newer modes of security vulnerabilities, for example, accessibility to sensitive data if a PCM DIMM gets stolen. PCM memories can be made secure by encrypting the data. Unfortunately, such encryption comes with a significant overhead in terms of bits written to PCM memory, causing half of the bits in the line to change on every write, even if the actual number of bits being written to memory is small. Our studies show that a typical writeback modifies, on average, only 12% of the bits in the cacheline. Thus, encryption causes almost a 4x increase in the number of bits written to PCM memories. Such extraneous bit writes cause significant increase in write power, reduction in write endurance, and reduction in write bandwidth. To provide the benefit of secure memory in a write efficient manner this paper proposes *Dual Counter Encryption (DEUCE)*. DEUCE is based on the observation that a typical writeback only changes a few words, so DEUCE re-encrypts only the words that have changed. We show that DEUCE reduces the number of modified bits per writeback for a secure memory from 50% to 24%, which improves performance by 27% and increases lifetime by 2x.

Categories and Subject Descriptors B.3.2 [*Hardware*]: Primary Memory

Keywords PCM, Encryption, Tamper-proof Design.

1. Introduction

As DRAM scaling becomes challenging, architects are looking for alternative technologies for designing future memory systems. Emerging Non-Volatile Memory (NVM) technologies, such as Phase Change Memory (PCM), Spin-Torque Transfer RAM (STT-RAM), and Resistive RAM (RRAM), have the potential for building high-density power-efficient memory systems due to enhanced scalability and non-volatility properties [1]. Without loss of generality, this paper analyzes PCM as a representative example of NVM, however the problem, insight, and solutions we develop are applicable to other NVM technologies as well.

One of the key challenges in designing a PCM system tends to be the efficient handling of write operations [2, 3]. Writes not only have a higher latency compared to reads, but they also tend to consume significant power and have limited endurance [4]. Therefore, PCM systems are optimized to reduce the write traffic, in order to improve power efficiency, write bandwidth, and endurance limited lifetime [5, 6]. Fortunately, when a write operation is performed to the main memory system, only a small number of bits get modified, and PCM systems leverage this observation to optimize the number of bits written to memory on each write operation. For example, PCM systems employ a technique called *Data Comparison Write (DCW)* [7] whereby only modified bits in the cache line get written to PCM memory. A further enhancement on this scheme, called *Flip-N-Write (FNW)* [8] reduces the bit writes to PCM even further by inverting the data if more than half of the bits get modified. FNW bounds the number of bit flips on each write to a maximum of half the number of bits in the line. With these optimizations, the number of bits written on each write operation to memory gets reduced to only 10% to 15%, on average. Thus, such optimizations are highly effective at reducing the number of bits written to PCM, and, given the reduced write performance and limited endurance of PCM memories, are vital to achieving high-performance and durability for PCM.

PCM systems are non-volatile and can store data for long after the system is powered down. While the non-volatility of PCM is a desirable property for reducing leakage power, this property can make PCM systems vulnerable to new security issues. For example, in traditional systems if a DRAM DIMM gets lost or stolen, the data is lost within a few seconds due to the low retention of DRAM. However, if a PCM DIMM gets stolen, then it is possible to access sensitive data even after several days or months. An adversary (such as a malicious repairman) with access to the DIMM can stream out the data stored in the PCM array and obtain sensitive information [9, 10]. Therefore, securing the memory against

ASPLOS '15, March 14–18, 2015, Istanbul, Turkey.

http://dx.doi.org/10.1145/2694344.2694387

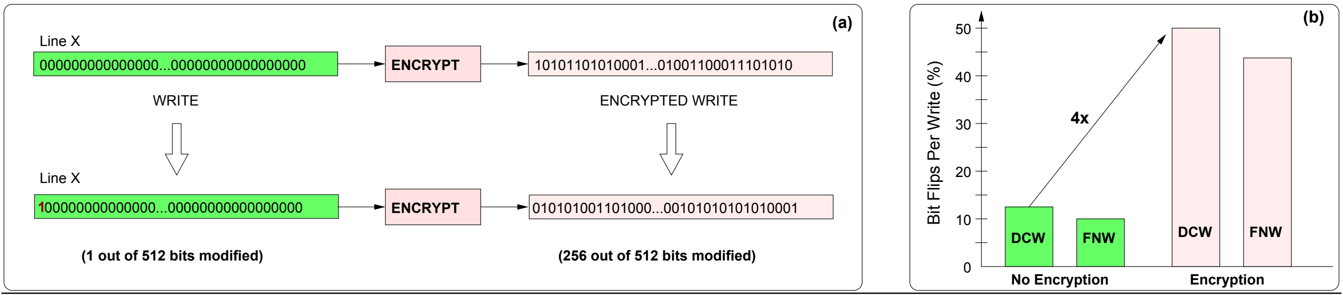

Figure 1. (a) Encryption changes half the bits in the line, even if only one bit in the line changes (b) Our evaluations show that encryption increases the number of bit flips in PCM by 4x for both Data Comparison Write (DCW) and Flip-N-Write (FNW).

such unintended access has become an even bigger concern for PCM systems compared to DRAM. Furthermore, PCM DIMMs are likely to be implemented using conventional bus-based interface standards such as LPDDR-NVM, making them susceptible to traditional attacks such as those that rely on bus snooping [11]. To prevent loss of sensitive data, it is important to make PCM memories secure against both bus snooping attacks as well as stolen DIMM attacks.

The security of PCM memories can be enhanced by encrypting the data. Unfortunately, all good encryption algorithms follow the Avalanche Effect [12], whereby a change of even a single bit in the input data causes half of the bits in the encrypted data to be changed. While this Avalanche property is essential to providing security, it causes the number of bits written on each memory write to be 50% regardless of the number of bits modified in the line. For example, consider Figure 1(a) where a write operation modifies only the MSB of a line that has been initialized to all zeros. This will cause only 1 bit to flip in the memory system without encryption. However, when encryption is employed, the difference between the two encrypted values will be half the number of bits. Thus, encryption causes the number of bits written to encrypted memory to be large, rendering optimizations such as DCW and FNW ineffective.

We analyze PCM memory system with benchmarks from the SPEC suite. Figure 1(b) shows the average number of bits modified per write for the unencrypted memory and encrypted memory when DCW and FNW are employed. We implement FNW at a granularity of two bytes, where 1 flip bit is provisioned per 16 bits. A write to memory modifies only 12.2% of the bits (as optimized by DCW) on average, and FNW reduces this to 10.5%. Unfortunately, encryption causes DCW and FNW to flip 50% and 43% of the bits per write, respectively. Thus, encryption increases the number of bits written to PCM memory by almost a factor of 4. This increase in bit writes can correspondingly increase the power consumption of writes by 4x, reduce endurance-related lifetime by 4x, and cause significant reduction in PCM write bandwidth due to power limitations. Ideally, we want to secure memory, without having to pay the power overheads, lifetime overheads, and performance overheads that happen due to extra writes from encryption. The goal of this paper is devise such a write efficient architecture for secure memory.

We implement memory encryption in the baseline using counter mode encryption [13, 14] scheme which uses One Time Pad (OTP) to implement low latency encryption. Each line has a counter associated with it which gets incremented on each write to the line. The line address and the counter is provided to the AES engine to generate the OTP, and this OTP is ex-ored with the data for encryption, or for decrypting the encrypted data. We leverage the observation that a typical writeback only modifies a few words, so it is not necessary to re-encrypt all the words with the new OTP. If we can keep track of the words that have been modified then we can continue to use the older OTP for the unmodified words. With this insight, we propose *Dual Counter Encryption (DEUCE)*. DEUCE maintains bits to track which words in the line have been modified. Words that have been modified use the current value of the line counter for encryption and decryption, whereas the unmodified words use an older value (obtained from masking few LSB of the current counter) of line counter for decryption. Periodically, the entire line is re-encrypted which resets the modified bits.

Our evaluations show that DEUCE reduces the number of bit flips for a write to encrypted memory by more than 2x, lowering it from 50% to 24%. The write reduction of DEUCE improves the performance of encrypted memory by 27% and EDP by 43%, while incurring a storage overhead of only 32 bits per line for a system with encrypted memory.

Reducing the bit writes by 2x with DEUCE does not result in a equivalent increase of 2x in lifetime, because certain bit locations continue to receive more writes than other locations. We propose *Horizontal Wear Leveling (HWL)* based on algebraic functions that provides near perfect intra line wear leveling, while avoiding the per line storage for tracking the rotation amount. We show that DEUCE with HWL improves the lifetime of encrypted memory by 2x.

While we evaluate DEUCE for a system that performs encryption at the line level, DEUCE is applicable to Block Level Encryption (BLE) schemes that perform re-encryption at the granularity of the AES block (16 bytes). For example, DEUCE lowers the average number of bit flips for a memory encrypted with BLE from 33% to 19.9%.

2. Background and Motivation

2.1 Attack Models

In this paper we specifically seek to protect PCM memories from two types of attacks. First, the traditional attack of bus snooping, where an adversary has access to the bus between the processor and memory and can obtain memory data by simply doing hardware probes of this memory bus [9]. Second, the stolen DIMM attack, whereby an adversary (or a malicious machine repairman) can get access to the PCM DIMM. The attacker can then simply stream the data out of the PCM memory to obtain all the data contained in the PCM DIMM. As PCM DIMM is non-volatile, the attack of the second type becomes much more severe in PCM compared to DRAM DIMM. We seek an efficient solution that will protect PCM DIMM from both types of attacks.

2.2 Security via Data Encryption with Line Counters

PCM memories can be protected against security threats by encrypting the memory contents. The straightforward way to encrypt data at runtime is to employ a one-way function such as AES, and use a global key, as shown in Figure 2(a). However, if the same key is used for all lines, one can simply compare encrypted lines to figure out which lines store the same content, and employ a dictionary-based attack.

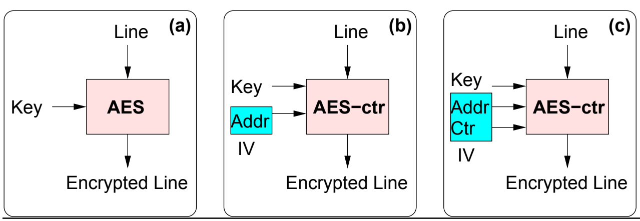

Figure 2. Enhancing security of (a) encrypted memory by adding (b) line address and (c) line counter to encryption.

A more secure approach is to use the line address along with the key to perform encryption, as shown in Figure 2(b). This ensures that each line gets encrypted with its own key, thwarting stolen DIMM attacks. However, this scheme is still vulnerable to bus snooping attacks that can monitor consecutive writes to the same line. An even stronger approach to data security is to employ a per-line counter in conjunction with the line address and key to perform encryption, as shown in Figure 2(c). The per-line counter is incremented on each write, ensuring that each write of each line gets encrypted with a unique key, and thus protects the system from both bus snooping attack and stolen DIMM attack.

2.3 Low Latency Encryption via One Time Pad (OTP)

Memory encryption not only incurs the storage overhead of a per-line counter, but it also incurs the latency overheads of encryption and decryption. Since memory read operations are in the critical path, the decryption latency is of particular importance as it gets added to the overall memory latency, as shown in Figure 3. While this latency may be acceptable when the latency of accessing data is quite high (for example, access from hard drive, which takes a few milli seconds), this latency becomes impractically expensive when the latency for both the data access and decryption are similar, such as for main memory (both take a few tens of nanoseconds).

Prior work [13, 14] on counter mode encryption avoids the serialization of memory access and decryption by employing the One Time Pad (OTP) technique. The key idea is to generate the OTP in parallel with the memory access, and simply XOR the OTP with the memory contents, as shown in Figure 3. As XOR incurs low latency, this scheme removes the decryption latency from the critical path of memory access, while maintaining provable security guarantees because of the uniqueness of the OTP.

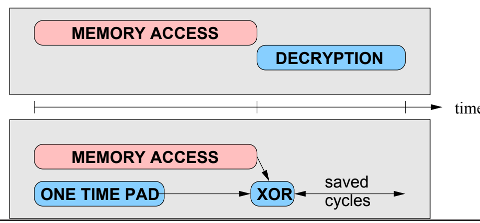

Figure 3. Counter mode encryption uses One Time Pad (OTP) to reduce the impact of decryption latency.

2.4 Operation of Counter Mode Encryption

Counter mode encryption uses a block cipher (such as AES), not to encrypt the data directly but to generate the OTP. To maintain security, the OTP for each line must be generated only once during the lifetime. To ensure this, counter mode encryption uses the line address and the per-line counter as the input data for the AES circuit to generate a unique output, which is used as a PAD. On a write to memory, the cache line is encrypted by XORing the content of the cache line with the OTP, as shown in Figure 4(a). To read the memory contents, the per-line counter and line address is used to regenerate the PAD, and this PAD is XORed with the encrypted contents, as shown in Figure 4(b).

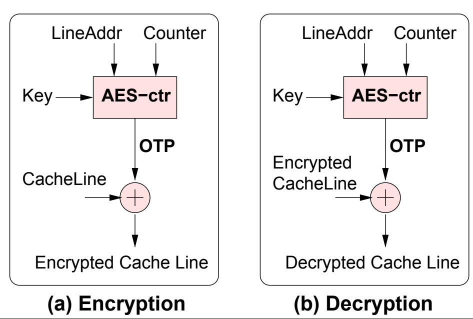

Figure 4. Counter mode generates OTP using line address and counter for both (a) Encryption and (b) Decryption.

The counter used for counter mode encryption need not itself be protected with encryption, as having the knowledge of the value of the counter does not enable the attacker to

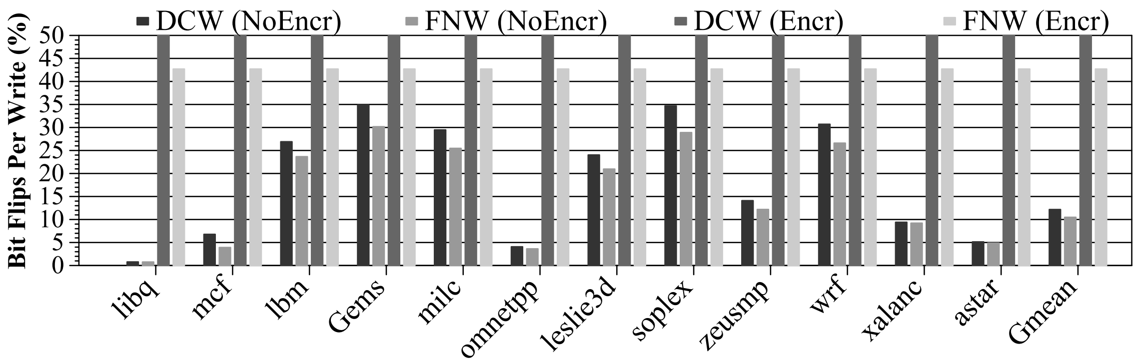

Figure 5. Average number of modified bits per write for unencrypted memory (NoEncr) and encrypted memory (Encr) under Data Comparison Write (DCW) and Flip-N-Write (FNW). Encryption increase the number of bit writes to PCM by almost 4x.

generated the pad [15]. Even if the attacker had access to the counter value, the attacker would still need the key and line address to generate the pad, and it is assumed that the key is well protected. Therefore, we store the line counter in plain text. The security of counter mode encryption has been approved by NIST. In our studies we will assume that memory is encrypted with counter based encryption.[1]

2.5 The Problem: Write Overhead of Encryption

The per-line counter in Counter Mode Encryption is incremented on every write, which allows every cacheline to have a unique PAD for each write. As the PAD is guaranteed to not be reused such systems provide strong security. In PCM systems, this security, however, comes at a cost to write endurance and bandwidth as 50% of the bits will be expected to flip on every writeback, due to the random bits of the PAD, regardless of the input data of the cacheline. Figure 5 shows the average number of bits modified per write for under both no encryption and encryption, when DCW and FNW are applied. For the unencrypted system, the average bits written per write are quite small, 12.4% for DCW and 10.5% for FNW. However, encryption increases this by almost 4x, making it 50% under DCW and 43% under FNW.

2.6 Goal: Write-Efficient Memory Encryption

Encryption causes the number of bit writes to increase by almost 4x, thereby significantly increasing write power, reducing performance, and lowering the lifetime of PCM memory system. Ideally, we would like to have memory security without the significant write overheads of memory encryption. A prior proposal, i-NVMM [17], proposes to employ partial encryption of memory whereby only cold pages are encrypted, and hot pages are encrypted only on power down. This scheme does not protect against bus snooping attacks (as the writeback from processor to memory is kept unencrypted for hot pages). We can also reduce the write overhead by using Block-Level Encryption (BLE) [18], which performs re-encryption at the granularity of AES block (16 bytes) using per block counters. Unfortunately, BLE still incurs high write overhead because even if a byte changes in the 16-byte block, the entire block must get re-encrypted. Our goal is to significantly reduce the write overhead of encryption, while retaining the security of memory contents. To this end, we propose *Dual Counter Encryption (DEUCE)*. We explain our methodology before explaining our design.

[1] Note that if the attacker has the ability to tamper with the memory values or the bus directly, then a new form of attack can be formed, whereby the attacker always resets the counter value to zero. This would make the system reuse the pad, and then the attacker can employ "pad reuse" based attacks to potentially gain access to the encrypted data. While, such *Bus Tampering Attacks* are much harder than *Bus Snooping Attacks*, and are beyond the scope of the attack models we consider, one can use Merkle Trees [14, 16] based authentication techniques to ensure the integrity of memory contents and protect against such attacks.

3. Methodology

3.1 Configuration

The parameters for the baseline system are shown in Table 1. The L4 cache is 64MB, and the L4 cache capacity is partitioned equally between all the cores. The linesize of all cache units is fixed at 64 bytes. Write requests arrive at the PCM memory only on eviction from the L4 cache. Both read and write requests are at the granularity of cache line, and are serviced by one of the PCM banks, based on the address of the line. We assume that relevant pages have already been brought into memory and been initially encrypted as they are placed into the memory via the memory-controller. For counter mode encryption, we assume each line is provisioned with a 28 bit line counter. We implement FNW at a two-byte boundary, so FNW incurs an overhead of 32 bits.

Processors	
Number of cores	8 cores, each 4-wide 4GHz
L1/L2/L3 (private)	32KB/256KB/1MB (8-way each)
L4 cache	64MB (8MB per core), 50 cycles latency
Phase Change Memory	
Capacity	4 ranks, each 8GB = 32 GB total
Read Latency	75 ns
Write Latency	150ns for each slot of 128 bits [19]

Table 1. Baseline Configuration

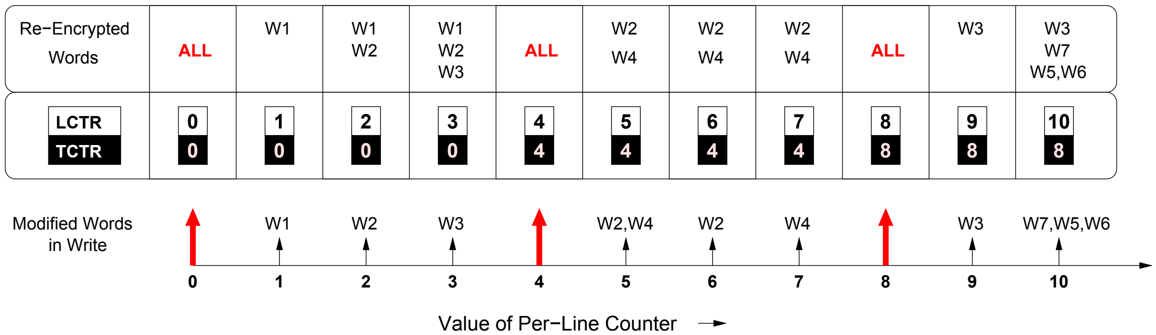

Figure 6. Operation of DEUCE for a system with Epoch Interval of 4 and 8 words per line (W0-W7). At start of each Epoch (red arrow), all words are re-encrypted and TCTR=LCTR. In between Epochs, all words that have been modified at least once since the start of the Epoch get re-encrypted. Such words use LCTR, whereas the unmodified words continue to use TCTR.

3.2 Workloads

We use a representative slice of 4 billion instructions for each benchmark from the SPEC2006 suite. We perform evaluations by executing the benchmark in rate mode, where all the eight cores execute the same benchmark. We perform our studies using all the 12 benchmarks that have at least 1 Writeback per One Thousand Instructions (WBPKI). Table 2 shows the read/write characteristics of our workloads.

Workload (8-copies)	L4 Read Miss (MPKI)	L4 WriteBack (WBPKI)
libq	22.9	9.78
mcf	16.2	8.78
lbm	14.6	7.25
Gems	14.4	7.14
milc	19.6	6.80
omnetpp	10.8	4.71
leslie3d	12.8	4.38
soplex	25.5	3.97
zeusmp	4.65	1.97
wrf	3.85	1.67
xalanc	1.85	1.61
astar	1.84	1.29

Table 2. Benchmark Characteristics.

3.3 Figure of Merit

We measure the average number of bits modified on each writeback as the figure of merit for different systems. While reporting the modified bits per cacheline, we also include the bit flips that happen in metadata (e.g. the Flip bit in FNW).

4. Dual Counter Encryption (DEUCE)

DEUCE leverages the observation that most words in any cacheline are written back unmodified. Fortunately, these unmodified words need not be re-encrypted and only the modified words can be re-encrypted. For example, if only one word in the line is modified, instead of rewriting the whole line, the design could re-encrypt only the modified word with an updated counter and write only that word back. On decryption, the design could decrypt the modified word with the updated counter, decrypt the rest of the cacheline with the previous counter, and combine them to form the newest cacheline. A straightforward way to implement this would be to have a separate counter per word (to avoid PAD reuse). However, this would not only incur high storage overhead but would also require an encryption algorithm that operates at a granularity of a word, making AES an unsuitable choice given that the minimum block length for AES is 16 bytes. To implement re-encryption at word granularity efficiently, we propose DEUCE, a simplified design that relies on generating two PADs to reduce the number of bit writes.

4.1 DEUCE: Overview

DEUCE tracks which words have been modified, and uses the most recent counter value for the modified words, and an older value of the counter for the unmodified words. We provide the overview of DEUCE with an example. Consider that the line has two counters, a *Leading Counter (LCTR)* and a *Trailing Counter (TCTR)*. The LCTR is identical to the line counter. The TCTR is obtained by masking off a few Least Significant Bits (LSBs) of the LCTR. For our example, let us consider that we mask the 2 LSB of LCTR. Thus, LCTR will be equal to TCTR once every 4 writes, a duration which we call *Epoch Interval*. In between the Epoch, the value of TCTR remains unchanged, whereas the LCTR keeps getting incremented. Each word is appended with a bit to indicate if it has been modified since the start of the Epoch. When LCTR equals TCTR, the modified bits associated with all words in the line are reset. On each subsequent write, the modified words are encrypted using LCTR, whereas the unmodified words keep using TCTR. Note that both LCTR and TCTR are virtual counters, whose values get determined by the line counter. Thus, except for the existing line counter, DEUCE does not require separate counters for implementation.

Figure 6 shows the operation of DEUCE for a design when the Epoch Interval is four writes and each line contains 8 words (W0 to W7). The counter is incremented on each write, and, when the two least significant bit of the line counter is 00 (at 0, 4, 8, etc.), all the words in the line are re-encrypted, and TCTR is advanced up to LCTR. Otherwise,

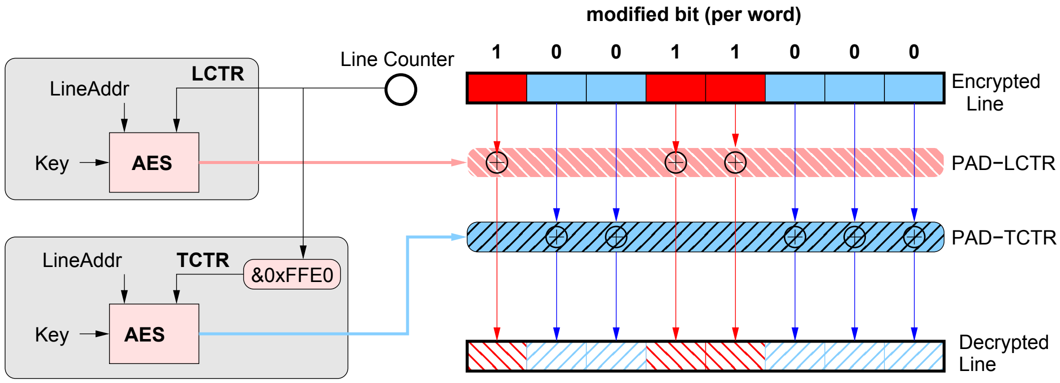

Figure 7. Operation of DEUCE. Two decryptions are performed, one with the PAD generated from the leading counter (PAD-LCTR), and the second with the PAD generated with the trailing counter (PAD-TCTR). The modified bit of the word decides which decrypted value to use. Note: both LCTR and TCTR are virtual counters, their values get determined by the line counter.

on every write LCTR is incremented, while TCTR is not. On every write, all the words that have changed since the last Epoch get re-encrypted. For example, at line counter value 1, W1 gets written, so only W1 is re-encrypted. On next write only W2 gets modified, but now both W1 and W2 get re-encrypted. Similarly, on the next write after that W3 gets written, so W1,W2, and W3 get re-encrypted. The next write triggers the start of the new Epoch so all words get re-encrypted, and TCTR is incremented from 0 to 4. And, so on. Thus, for each write during the Epoch, only a subset of words (that have been modified at least once since the last Epoch) are re-encrypted. This way, the modified words use LCTR, and other unmodified words continue to use TCTR.

4.2 DEUCE Parameters: Epoch and Word Size

The two design parameters for DEUCE are *Epoch Interval* and *Word Size*. The Epoch Interval impacts the effectiveness of DEUCE in terms of reducing bit flips, as there is a full line re-encryption on every Epoch Interval. A larger interval will reduce the full re-encryption overhead. However, a longer Epoch Interval may also cause DEUCE to continuously re-encrypt many words that are modified shortly after the Epoch but have stopped being modified since.

The word size determines the storage overhead and fineness of tracking modified parts of the cacheline. As we need one modified bit per word, we would need 64 bits per line if tracking is done at a byte granularity, and only 8 bits per line if tracking is done at 8 bytes. A finer granularity improves the effectiveness of DEUCE, but incurs higher overhead. So, the word size of DEUCE must be carefully selected.

4.3 Design of DEUCE

4.3.1 Structure

To implement DEUCE on top of a secure memory system, we simply need 1 bit per word in the line to indicate if the word has changed since last Epoch. For example, if tracking is done at a 2 byte granularity then we would need 32 modified bits for each line in memory.

4.3.2 Write Operation

At the start of each Epoch, all words in the line are re-encrypted and all the modified bits are reset. On subsequent writes, a read is performed to identify the words that are modified by the given write. The modified bits corresponding to these words are set. Only the words which have been modified at least once since the start of the Epoch are re-encrypted with the LCTR. The unmodified words (identified by the modified bit equal to zero) are encrypted with the TCTR, and thus continue to remain unchanged in memory.

4.3.3 Read Operation

On a read, decryption is performed as shown in Figure 7. Two PADs are obtained, one with the leading counter (PAD-LCTR), and second with the trailing counter (PAD-TCTR). Decryption is performed based on both PADs; however, the modified bit indicates which of the two values should be used for the given word. If the modified bit for a given word is set then the value decrypted using PAD-LCTR is used, otherwise the value decrypted using PAD-TCTR is used.

4.3.4 Overheads

The storage overhead of DEUCE is simply 1 bit per tracking granularity, so for our default configuration that tracks at a granularity of two bytes, we need 32 bits per line. The latency overhead of DEUCE is negligible, as we generate both PADs in parallel and use a multiplexer to choose between the two decrypted values. Thus, we would need two AES circuits, or we can simply time divison multiplex the AES circuits to generate the two PADs one after the other. As long as both PADs are obtained before the data from the memory arrives, we would avoid the latency overheads.

4.3.5 Security

The security of the counter based scheme relies on the premise that a given OTP is not reused, therefore it is critical to ensure that DEUCE does not use the same PAD multiple times for re-encryption of data. DEUCE always uses a different PAD for re-encrypting the word if the word gets modified. DEUCE simply avoids re-encryption for words that remain unmodified. Given the uniqueness of the PAD for re-encryption of the modified words, the security level of DEUCE is similar to the baseline OTP scheme. The attacker can only retrieve information that a particular word has been modified since the last Epoch, which is similar to knowing which line is modified in a page, based on the line address.

4.4 RESULTS: Sensitivity to DEUCE Word Size

Figure 8 shows the number of modified bits per write, on average, as the tracking granularity of DEUCE is varied from 1 byte (64 bit overhead per line) to 8 bytes (8 bit overhead per line) for an Epoch interval of 32. A smaller granularity is more efficient at reducing write overhead. As the tracking granularity is increased from 1 byte to 2 byte to 4 byte to 8 bytes, the average number of modified bits increases from 21.4% to 23.7% to 26.8% to 32.2%. Our default configuration uses a granularity of 2 bytes, which has a storage overhead of 32 bits per line.

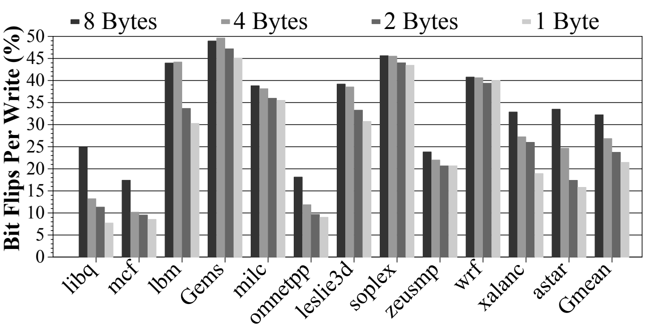

Figure 8. Number of modified bits per write with DEUCE as the tracking granularity is varied. DEUCE with word size of 2 bytes reduces modified bits from 50% to 23.7%.

4.5 RESULTS: Sensitivity to Epoch Interval

Figure 9 shows the number of modified bits per write, on average, with DEUCE as the Epoch interval is varied from 8 to 32. As the Epoch interval increases, the full encryption of the line happens less frequently, reducing the number of modified bits. However, at a much larger Epoch Interval, some words that were modified earlier in the Epoch but are no longer being modified continue to get re-encrypted, resulting in higher bit flips. For example, for wrf and milc there is an increase in bit flips when the Epoch Interval is increased from 8 to 16, and 16 to 32, respectively. The average number of bits modified for Epoch Interval of 8 is 24.8%, for Epoch Interval of 16 is 24.0%, and for Epoch Interval of 32 is 23.7%. Overall, increasing the Epoch Interval has a relatively small effect (less than 1%) on bit writes, as most writebacks tend to be to similar locations. We use a default Epoch Interval of 32 in our studies.

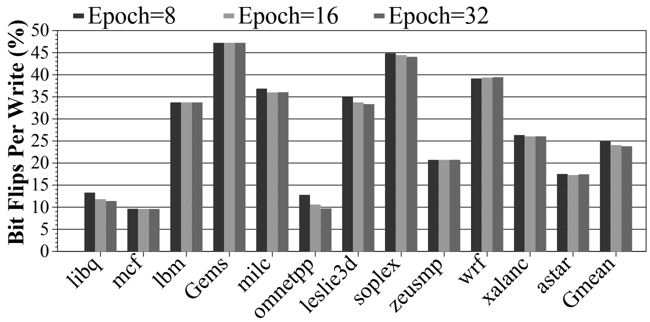

Figure 9. Number of modified bits per write with DEUCE as the Epoch Interval is varied. DEUCE with Epoch Interval of 32 reduces modified bits from 50% to 23.7%.

4.6 DynDEUCE: Morphing from DEUCE to FNW

DEUCE is effective at reducing the average number of bit flips for an encrypted memory by more than 2x (from 50% to 24%). DEUCE performs well when programs write to cachelines in sparse and regular patterns, which tends to be the common case for most general general purpose applications. However, if the applications modifies all the words of the line on every write, then DEUCE will not be able to reduce the number of bit writes to memory, as it will deem all the words as modified and will re-encrypt all the words of the line on each write, thus causing 50% of the bits to flip on each write. In such a scenario, it would have been better to use Flip-N-Write (FNW) and get the 14% fewer bit flips (from 50% to 43%) that FNW can provide. For example, consider the number of modified bits per write shown in Figure 9. Both Gems and soplex have an average of more than 43% bit flips with DEUCE, and would have benefited from FNW instead of DEUCE. Both FNW and DEUCE incur similar storage overheads (32 bits per line). As both FNW and DEUCE are orthogonal, one can potentially implement encrypted memory with both (a configuration we call FNW+DEUCE). However, this configuration will incur double the storage overhead, of 64 bits per line. Ideally, we want benefit of FNW, if FNW is better than DEUCE, without paying for the extra storage overhead. We propose such a configuration that can dynamically switch from DEUCE to FNW, in between the Epoch if FNW provides more benefit than DEUCE. We call this configuration as *DynDEUCE*.

To maintain a constant storage overhead, DynDEUCE reuses the bits allocated for tracking the modified words in DEUCE as the FlipBits in FNW when it is deemed that is FNW is beneficial. Thus, the storage overhead of DynDEUCE is a single bit (ModeBit) per line. If the ModeBit is zero, DynDEUCE operates as DEUCE. When the ModeBit is 1, then DEUCE gets disabled, all words are re-encrypted, and the modified bits allocated for DEUCE are repurposed as FlipBits of FNW. Determining which of the two policies, DEUCE or FNW, is better is done dynamically by measur-

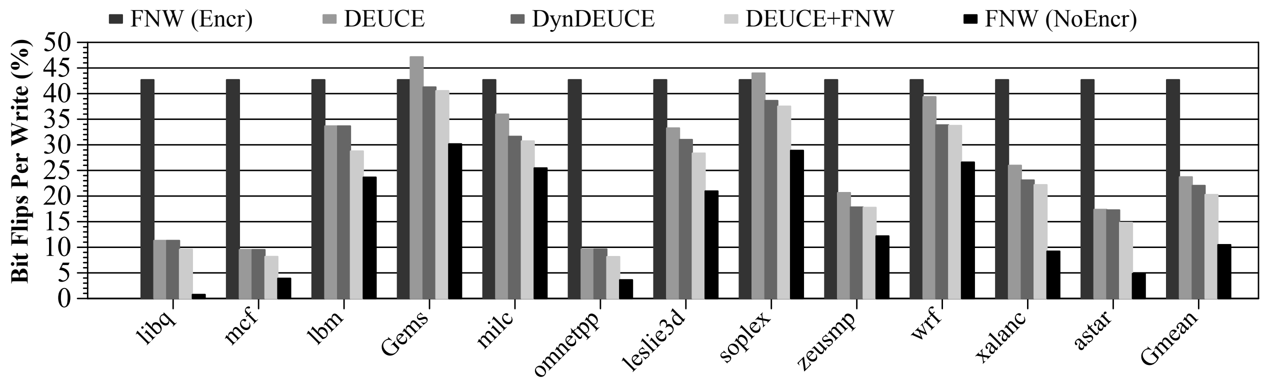

Figure 10. Number of bit flips per write with various schemes. DEUCE and DynDEUCE reduce the bit flips to 24% and 22% respectively. These schemes bridge two thirds of the gap between FNW (with encryption) and FNW (without encryption).

ing the number of bit flips expected with the two policies. If FNW has fewer bit flips then the ModeBit is set to use FNW.

Morphing the mode from DEUCE to FNW is straightforward; however, going from FNW to DEUCE is impractical, as the state at the start of the Epoch is lost when the mode is changed from DEUCE to FNW. DynDEUCE solves this problem by ensuring that the mode returns to DEUCE at the start of every Epoch. Then, at each write during the Epoch, if the number of bit flips with FNW is smaller then the mode is shifted from DEUCE to FNW. Figure 11 shows an example of DynDEUCE. At the start of the Epoch the ModeBit is set to use DEUCE. At the next write (and all subsequent writes till the next Epoch) the number of bit flips with DEUCE and FNW are estimated. For the example, encrypting the line with DEUCE gives 67 bit flips and encrypting the line with FNW gives 55 bit flips. So, DynDEUCE decides to change the mode to FNW, and for all the writes till the next Epoch, the mode continues to be FNW. Thus, DynDEUCE can be implemented with a storage overhead of only 33 bits per line (1 bit for mode selection and 32 bits, which are used either as modified bits for DEUCE or as Flip bits for FNW).

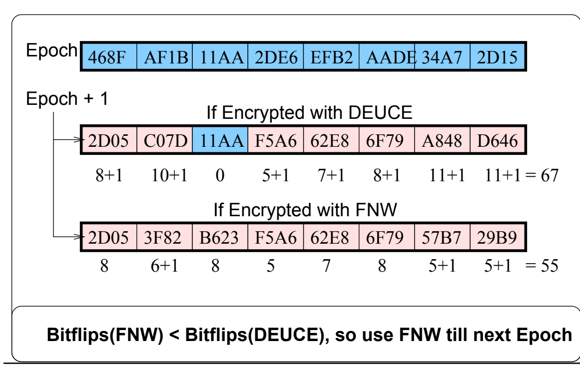

Figure 11. Operation of DynDEUCE. For each write after the Epoch, DynDEUCE calculates the expected number of bit flips with FNW and DEUCE. If FNW has fewer bit flips than DEUCE, the mode is set to FNW till the next Epoch.

4.7 RESULTS: Comparison of Bit Flip Reduction

Figure 10 shows the number of modified bits per write, for various bit flip reductions schemes, such as FNW, DEUCE, DynDEUCE, and DEUCE+FNW (has dedicated bits for both DEUCE and FNW). We compare these schemes to a FNW system with no encryption. The average number of bit flips with FNW with encrypted memory remains constant at 43% regardless of the workload. DEUCE is highly effective at reducing the number of bit flips, especially for workloads such as libq, mcf, and omnetpp. These workloads modify only a small number of words per write, and the footprint of which word gets modified does not change significantly. On average, DEUCE reduces the number of bit flips to 23.7%.

For Gems and Soplex, DEUCE has higher bit flips than FNW. For both these workloads DynDEUCE is more effective than either FNW or DEUCE standalone. On average, DynDEUCE has 22.0% bit flips. DynDEUCE bridges half the gap between DEUCE and FNW+DEUCE, a policy that dedicates dedicated storage to both FNW as well as DEUCE (total of 64 bits per line). Overall, DEUCE and DynDEUCE remove two-thirds of the extra bit flips due to encryption.

4.8 Comparison of Storage Overhead

Table 3 shows the storage overhead of different schemes for an encrypted memory system (already equipped with per line counter). Both FNW and DEUCE are implemented with identical storage overhead; DynDEUCE needs 1 extra mode bit, and DEUCE+FNW needs twice the storage.

Scheme	Overhead	Avg. Bit Flips Per Write
FNW	32 bits/line	42.7%
DEUCE	32 bits/line	23.7%
DynDEUCE	33 bits/line	22.0%
DEUCE+FNW	64 bits/line	20.3%

Table 3. Storage Overhead and Effectiveness

5. Horizontal Wear Leveling for Durability

PCM is an endurance limited memory technology, and reduction in write traffic can enhance the system lifetime significantly. DEUCE reduces the bit writes to PCM by a factor of 2x, so ideally we would expect the lifetime to also increase by 2x. However, we found that the average lifetime with DEUCE increases by only 11%. This happens because some portions of the line continue to get written much more frequently than others causing them to wear out sooner. This section analyzes the problem of non-uniform bit writes within a line, and develops a low cost and highly effective wear leveling algorithm to make bit writes uniform.

5.1 Problem of Non-Uniformity in Bit Writes

Figure 12 show the number of writes to each bit position of a cache line, normalized to the average writes to a bit, for *mcf* and *libquantum*. The bit writes are highly non-uniform, and the most frequently written bit receives as much as 6x (for mcf) to 27x (for libquantum) the writes compared to the average. Such non-uniformity in bit writes not only reduce the lifetime of unencrypted memory but also that of DEUCE, as the same set of words get re-encrypted frequently.

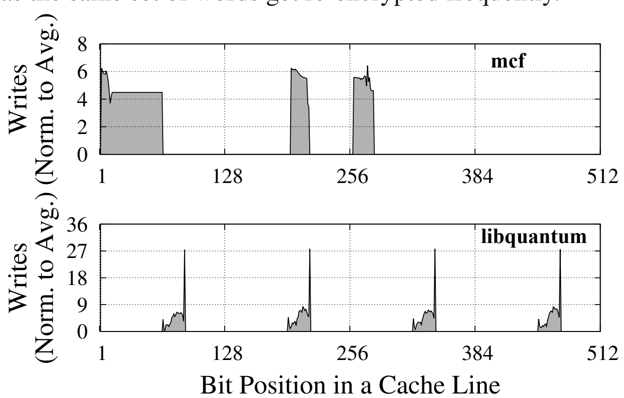

Figure 12. Variation in writes per bit position of a line. The most frequently written bit receives as much as 6x (for mcf) to 27x (for libquantum) more writes than average.

5.2 Background on Vertical Wear Leveling

The problem of non-uniformity in writes can be addressed by wear leveling. We call wear leveling algorithms that operate at a line granularity, such as Start-Gap [20] and Security Refresh [21], as *Vertical Wear Leveling (VWL)* algorithms. Unfortunately, Vertical Wear Leveling is not effective at reducing the bit write variation within a line. Bit writes in a line can be made uniform by rotating the line periodically and keeping track of the rotation amount per line [7]. Ideally, we want uniform bit writes without relying on dedicated storage on a per line or per page basis. We extend vertical wear leveling algorithms, such as Start-Gap and Security Refresh, to provide a storage overhead free intra-line wear leveling, which we call *Horizontal Wear Leveling (HWL)*.

We explain Start-Gap as a low-cost Vertical Wear Leveling algorithm. Figure 13(a) shows the structures for Start-Gap for a memory with 8 lines (A-H). It consists of two pointers, Start and Gap, and one dummy line called the GapLine to aid movement. Every so often (say 100 writes), the Gap line moves by one, by copying the content of the previous line to the next line, as shown in Figure 13(b), After the Gap has gone through all the lines, the Start register is incremented by 1. Thus, Start keeps track of the total number of global rotations, and Gap keeps track of the lines moved in current rotation. Using Start and Gap, the remapped location can be obtained easily.

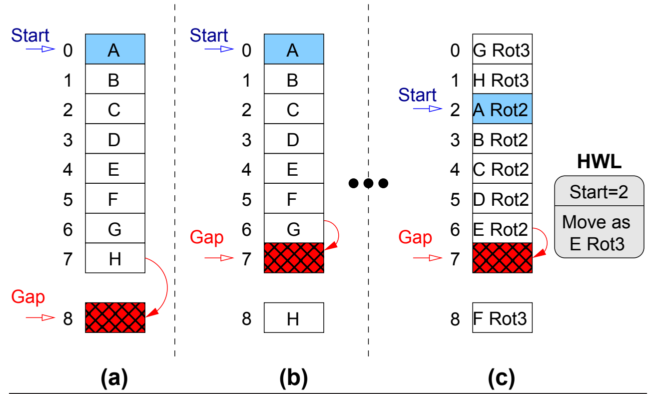

Figure 13. Extending Start-Gap to perform HWL (a) Start-Gap (b) Operation of Start Gap (c) Implementing HWL by rotating a line by (Start+1) on Gap moves.

5.3 Horizontal Wear Leveling via Algebraic Functions

Our insight to develop low cost Horizontal Wear Leveling (HWL) algorithms is to make the rotation amount an algebraic function of the global structures used by Vertical Wear Leveling. For example, we can get the rotation amount as *Rotation Amount = [Start' % BitsInLine]*, where Start' equals (Start+1) if the Gap has already crossed the line, otherwise Start' is equal to Start. This way, rotation amount can increase as Start increases and wrap around when the line has been rotated through all the bits (including any metadata bits associated with the line). Note that, if the Gap has already crossed through the line, then the rotation amount is determined as *Rotation Amount = [(Start+1) % BitsInLine]*. This ensures that when Start increments, all the lines for which the Gap has already passed through have already been rotated by the amount equal to the new value of the Start. Figure 13(c) shows the extension of Start-Gap that supports HWL. For typical applications, the Start register gets incremented by several hundred thousand, so all the bits within each line get rotated several hundred times, ensuring close to uniform bit writes. We found that the lifetime with HWL is within 0.5% of the lifetime of perfect wear leveling.[2]

[2] Given the deterministic nature of HWL, an adversary (or a badly written) program can purposely shift the write pattern at the granularity of the change of Start. The system can be protected against such access patterns easily by calculating a unique rotation amount for each line, as a function of the line address and Start. For example, calculate rotation amount using *Rotation Amount = [Hash(Start',LineAddress) % BitsInLine]*.

To support HWL, the memory is equipped with shifters, for rotating the line on a write and reverse rotating the line on read. Note that our proposed HWL does not incur any storage overhead. It avoids extra writes for line rotation by leveraging the line movement that is incurred by vertical wear leveling.

5.4 RESULTS: Impact on Lifetime

Figure 14 shows the lifetime of FNW, DEUCE, and DEUCE with HWL (DEUCE-HWL), normalized to the lifetime of the encrypted memory. Note that for encrypted memory, all the bits in the line get written with 50% probability, so it achieves an almost uniform bit writes within the line and HWL is not required. FNW reduces bit flips by 14% (50% to 43%) and the writes are spread uniformly across all bits, so it provides a 14% improvement in lifetime. Even though DEUCE reduces bit writes by 2x, the average lifetime improvement is only 11%. This is because DEUCE continues to write to the frequently written parts of the line. However, DEUCE when combined with HWL provides an average lifetime of 2x, which is in proportion to the reduction in bit writes. Thus, HWL achieves lifetime close to uniform bit writes, while incurring no storage overhead, other that what is required for vertical wear leveling.

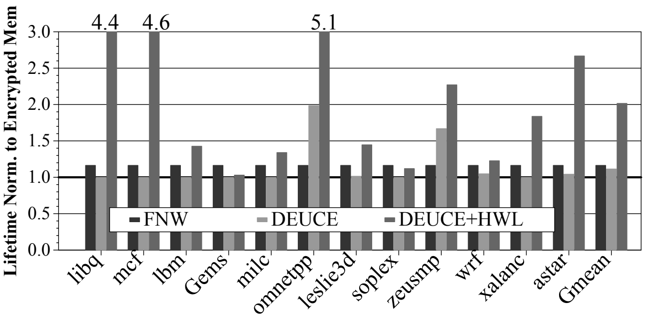

Figure 14. Lifetime normalized to encrypted memory. HWL increases the lifetime of DEUCE from 1.1x to 2x.

6. Impact on Performance and Power

6.1 Impact on Write Throughput

Due to power limitations, PCM memory systems tend to have limited write throughput. Typically, there is not enough current to write all the bits of a 64 byte cache line within one shot. For example, the recent 8Gb PCM prototype [19] shows a write width of 128 bits, and the write of a cache line can take up-to 4 write slots. For our system, we assume a 128 bit width for write (provisioned for handling a maximum of 64 bit flips using internal Flip-N-Write [22]). We also assume that multiple writes can be scheduled concurrently, provided the total number of bit flips does not exceed the current capacity [22].

Bit flip reduction can potentially cause a write to consume fewer write slots, as multiple portions of a cacheline can be serviced concurrently. However, the reduction in bit flips does not always directly translate into fewer write slots because of fragmentation. For example, if the given write causes 70 flips, and one slot can only handle 64 flips, then this write will take two slots. Figure 15 shows the average number of write slots consumed per write. FNW on an encrypted PCM reduces bit flips by 14% (50 to 43), but this reduction is not enough to allow the PCM system to service the request in three slots. DEUCE, on the other hand, reduces the number of write slots from 4.0 to 2.64, which reduces the effective write latency by 34%, thus improving the write throughput by 52%. The unencrypted memory consumes 1.92 write slots on average. Thus, DEUCE bridges two-thirds of the gap for consumption of write slots between encrypted memory and unencrypted memory.

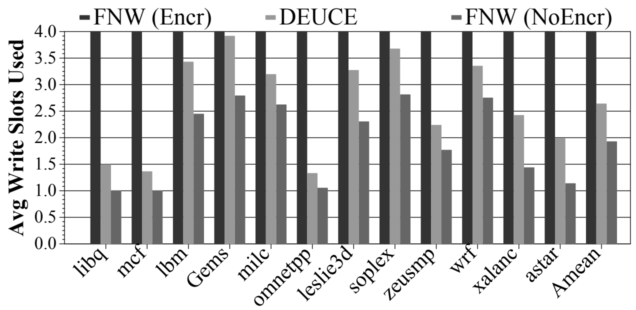

Figure 15. Average number of write slots used per write request. On average, DEUCE consumes 2.64 slots whereas unencrypted memory takes 1.92 slots out of the 4 write slots.

6.2 Impact on System Performance

Servicing the writes quickly can reduce the memory contention for reads and thus improve system performance. Figure 16 shows the system speedup of FNW (with and without encryption) and DEUCE compared to the encrypted memory system. Applying FNW on encrypted memory has negligible impact on performance because of the fragmentation of write slots, which means even with FNW the system tends to have write latency similar to that of the encrypted memory without FNW. DEUCE, however, is able to increase system performance by 27% on average, due to its increased write throughput. Disabling encryption and using FNW would improve the system performance by 40%, on average. Thus, DEUCE bridges two-thirds of the performance gap between encrypted memory and unencrypted memory.

6.3 Impact on Power and Energy

As writes in PCM consume significant power, the reduction in bit writes also has a significant impact on the energy and power of the system. We measure energy and power for the PCM memory system, taking into account the power consumed for each bit written. Figure 17 compares the speedup, energy consumption, power, and Energy Delay Product (EDP), normalized to an encrypted memory system.

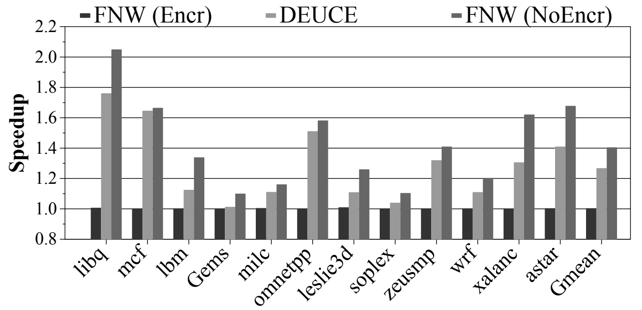

Figure 16. Speedup compared to encrypted memory system. DEUCE bridges two-thirds of the performance gap between encrypted memory and unencrypted memory.

FNW has only a small impact on the energy and power consumption of PCM memory, reducing it by approximately 11%. DEUCE reduces memory energy consumption by 43% and power by 28%. The reduction in power is less than the reduction in energy because of the shorter execution time.

Energy Delay Product (EDP) is often used as a metric combining power and performance. Compared to the baseline encrypted memory, FNW provides a reduction of 4% and DEUCE of 43%. Disabling the encryption and employing FNW would provide an EDP improvement of 56%. Thus, DEUCE bridges three-fourths of the EDP gap between encrypted and unencrypted memory.

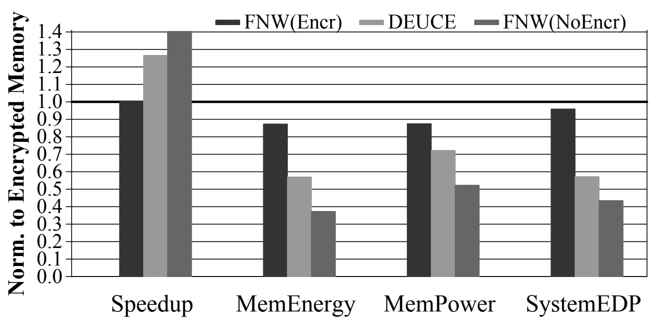

Figure 17. Speedup, memory energy consumption, memory power consumption, and system Energy Delay Product.

7. Related Work

Our study addresses the problem of reducing write traffic for encrypted memory systems. We discuss some of the closely related works that have looked at secure PCM memories.

7.1 Block Level Encryption at AES Granularity

In our studies, we assume counter mode encryption that operates at a line granularity [13, 14]. Each line has one counter, which is incremented on each write, thus causing the entire line to be modified. Block-Level Encryption (BLE) [18] reduces write overhead by performing re-encryption at the granularity of an AES block (16 bytes). As the cacheline is 64 bytes, BLE provisions the line with four separate counters, one per each block. On a write, BLE re-encrypts only the modified blocks in the cacheline and increments the counter of the modified blocks. Unfortunately, BLE still incurs high write overhead as it rewrites 16 bytes, even if one bit in the block gets modified. DEUCE, on the other hand, operates at a much smaller granularity of 2 bytes, and performs fine grained re-encryption. Thus, DEUCE decouples the granularity of re-encryption from the granularity of AES. Nonetheless, DEUCE is orthogonal to BLE and can be combined for greater benefit.

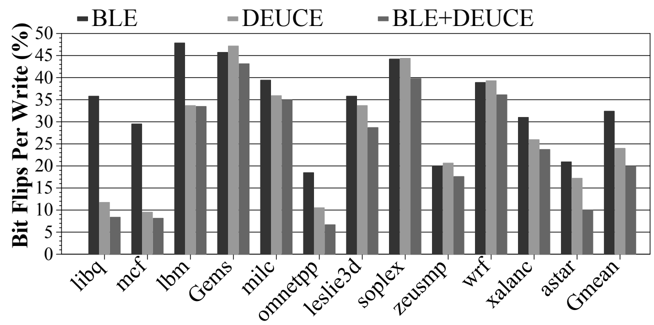

Figure 18. Bit flips with DEUCE and Block Level Encryption at granularity of AES. DEUCE (24%) is orthogonal to BLE (33%) and can be combined for greater benefits (20%).

Figure 18 shows the average number of modified bits per line with BLE, DEUCE, and the combination of BLE+DEUCE. BLE reduces the bit flips to 33%, whereas DEUCE reduces it to 24%. The combination of BLE+DEUCE cause 19.9% bit flips, outperforming either scheme standalone.

7.2 Partial Working-Set Encryption with i-NVMM

i-NVMM [17] tries to reduce the latency overheads of encryption by keeping the hot data in unencrypted form, and encrypting it only on power down. While i-NVMM may protect the system against the stolen DIMM attack, it still leaves the system vulnerable to bus snooping attacks, as writes to hot pages are sent unencrypted on the bus. We seek to protect the memory against both stolen DIMM attack as well as the bus snooping attacks. If one wishes to protect the system only against stolen DIMM attack, then it can be done with low latency (and low bit flips) by eliminating the counter from counter mode encryption, and simply using the line address to generate a unique PAD per line. When the DIMM gets stolen, the adversary will not be able to regenerate the PADs per line without the secret key. This would also ensure that the entire memory content always remains encrypted, which is not the case for i-NVMM.

7.3 Protecting PCM from Endurance Related Attacks

Given that PCM has limited endurance, PCM systems are vulnerable to endurance related attacks that write to a small portion of memory repeatedly [20, 21, 23]. PCM systems can be protected from such lifetime reducing attacks by means of an attack detector [23]. DEUCE is an encryption scheme that focuses on information security as opposed to lifetime security. Techniques that mitigate endurance related attacks could also be applied for systems that use DEUCE.

8. Conclusions

Secure PCM systems rely on data encryption, which increase the bit flips per write by almost 4x. The extra bit flips degrade write throughput, performance, lifetime, and power efficiency. Ideally, we want the security of encryption without significantly increasing the bit writes to PCM. We observe that each writeback writes to only a few words, and we can reduce bit flips by re-encrypting only modified words. With this insight, this paper makes the following contributions:

1. We propose *Dual Counter Encryption (DEUCE)*, a design that selectively re-encrypts cachelines at a fine granularities (two bytes). DEUCE reduces the number of bit flips for an encrypted memory system from 50% to 24%, while incurring a storage overhead of 32 bits per line.
2. We propose a simple *Horizontal Wear Leveling (HWL)* algorithm based on an algebraic function of Start-Gap, to make the bit writes within a line uniform. Our proposed HWL is storage overhead free, and increases the lifetime improvement provided by DEUCE from 1.1x to 2x.
3. We show that DEUCE is orthogonal to, and can be combined with, existing schemes such as Flip-N-Write (FNW) and Block Level Encryption (BLE). For example, DEUCE combined with BLE reduces the bit flips of BLE from 33% to 19.9%. We also develop *DynDEUCE*, which dynamically selects between FNW and DEUCE, depending on which scheme has fewer bit writes.

The reduction in bit flips with DEUCE improves write throughput by 52%, performance by 27%, lifetime by 2x, and EDP by 43% with respect to an encrypted memory system. DEUCE bridges two-thirds of the gap in performance, power, and lifetime between an encrypted memory system and an unencrypted memory system.

With an increase in hardware attacks, secure memory systems are becoming essential. We show how architecture support can mitigate the overheads of encryption, and make security practical. While we analyze PCM as an example of NVM, the scheme, insights, and evaluations should also be useful for other NVM technologies, such as STT-RAM and RRAM, in which write operations are expensive and therefore it is desirable to have write-efficient memory encryption to obtain both security and high performance and durability.

Acknowledgements

We thank the anonymous reviewers, Andrew Chien, Swamit Tannu, and Daniel Wong for their comments and feedback on earlier draft of this paper. We also thank Simha Sethumadhavan and Milos Prvulovic for discussions and providing useful insights on working of counter mode encryption and security analysis of encrypted memoryx. This work was supported by the Center for Future Architectures Research (C-FAR), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

References

[1] M. K. Qureshi, S. Gurumurthi, and B. Rajendran, "Phase change memory: From devices to systems," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 4, 2011.

[2] J. Yue and Y. Zhu, "Accelerating write by exploiting pcm asymmetries," in *HPCA-2013*.

[3] L. Jiang *et al.*, "A low power and reliable charge pump design for phase change memories," in *ISCA-2014*.

[4] S. Schechter *et al.*, "Use ecp, not ecc, for hard failures in resistive memories," in *ISCA-2010*.

[5] B. C. Lee, E. Ipek *et al.*, "Architecting phase change memory as a scalable dram alternative," in *ISCA-2009*.

[6] M. Qureshi, M. Franceschini, and L. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing," in *HPCA-16*, 2010.

[7] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ISCA*, 2009.

[8] S. Cho and H. Lee, "Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance," in *MICRO-2009*.

[9] A. Huang, "The trusted pc: skin-deep security," *Computer*, vol. 35, no. 10, 2002.

[10] A. B. Huang, *Hacking the Xbox: An Introduction to Reverse Engineering*. No Starch Press, 2003.

[11] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan, "Side-channel vulnerability factor: A metric for measuring information leakage," in *ISCA-2012*.

[12] A. Webster and S. Tavares, "On the design of s-boxes," in *Advances in Cryptology CRYPTO 85 Proceedings*, ser. Lecture Notes in Computer Science, 1986, vol. 218.

[13] Suh *et al.*, "Efficient memory integrity verification and encryption for secure processors," in *MICRO-2003*.

[14] C. Yan, B. Rogers, D. Englender, D. Solihin, and M. Prvulovic, "Improving cost, performance, and security of memory encryption and authentication," in *ISCA-2006*.

[15] H. Lipmaa, P. Rogaway *et al.*, "Ctr-mode encryption," 2000.

[16] B. Rogers, S. Chhabra *et al.*, "Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly," in *MICRO-2007*.

[17] S. Chhabra and Y. Solihin, "i-nvmm: A secure non-volatile main memory system with incremental encryption," in *ISCA-2011*.

[18] J. Kong and H. Zhou, "Improving privacy and lifetime of pcm-based main memory," in *DSN-2010*.

[19] Y. Choi *et al.*, "A 20nm 1.8v 8gb pram with 40mb/s program bandwidth," in *ISSCC*, 2012.

[20] M. K. Qureshi, J. Karidis, M. Franceschini *et al.*, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *MICRO-42*, 2009.

[21] 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*.

[22] A. Hay, K. Strauss, T. Sherwood *et al.*, "Preventing pcm banks from seizing too much power," in *MICRO-2011*.

[23] M. K. Qureshi, A. Seznec, L. A. Lastras, and M. M. Franceschini, "Practical and secure pcm systems by online detection of malicious write streams," in *HPCA-2011*.