# CASTLE: Compression Architecture for Secure Low Latency, Low Energy, High Endurance NVMs

Poovaiah M. Palangappa
University of Pittsburgh
poovaiah.palangappa@pitt.edu

Kartik Mohanram
University of Pittsburgh
kartik.mohanram@gmail.com

## Abstract

CASTLE is a Compression-based main memory Architecture realizing a read-decrypt-free (i.e., write-only) Secure solution for low laTency, Low Energy, high endurance non-volatile memories (NVMs). CASTLE integrates pattern-based data compression and incomplete data mapping (i.e., expansion coding) to improve NVM energy, latency, and lifetime without impacting the security guarantees of the underlying security architecture. System-level simulations of a TLC RRAM architecture show that CASTLE reduces memory energy by 19% and write latency by 38.7%, and improves lifetime by 1.8× over state-of-the-art solutions for NVM security.

**CCS Concepts: Security and privacy → Security in hardware**

## 1 Introduction

Resistance-class non-volatile memory (NVM) technologies such as phase-change memory (PCM) [1] and resistive RAM (RRAM) [2] are under active research and development as potential replacement candidates for DRAM due to their advantages in non-volatility (low refresh energy), improved scalability, and high data density [3]. However, the non-volatility exposes data to security threats [4–6]. Whereas data encryption [4–7] can guarantee NVM security, it increases entropy, resulting in significant cell-update, instructions-per-cycle (IPC), and bandwidth overheads. These limitations are further aggravated by the move from single-level cell (SLC) to denser multi-/triple-level cell (MLC/TLC) NVM technologies, with average cell updates to 50%, 75%, and 87.5% of the total cells for SLC, MLC, and TLC NVMs, respectively[1]. Whereas standard NVM bit-write reduction techniques like data-comparison write (DCW) [8] and Flip-N-Write (FNW) [9] can reduce bit-writes, their effectiveness is constrained by the increased entropy of the encrypted data.

Block-level encryption (BLE) [10] and dual counter encryption (DEUCE) [11] are state-of-the-art counter-mode encryption (CME) solutions for NVM security that achieve cell update reduction by using a read-decrypt-modify-write sequence, where the ciphertext (encrypted data) stored in main memory is read to the processor-side memory controller, decrypted, and compared with the incoming plaintext data in chunks of 128 bits and 16 bits, respectively. Both BLE and DEUCE ensure that only the ciphertext corresponding to the modified data chunks are updated in the memory. Whereas BLE and DEUCE achieve bit-write reductions over DCW and FNW, they are limited by the mandatory read-decrypt steps integral to every write; our system-level simulations show that the read-decrypt-modify-write sequence results in an IPC loss of 7.2% on average (up to 50%), which is prohibitive in practice.

CASTLE is a compression-based block-level read-decrypt-free, i.e., write-only CME main memory security solution to realize low latency, low energy, high endurance NVMs. Whereas a write-only approach potentially increases cell updates, impacting energy and latency, CASTLE integrates pattern-based compression and incomplete data mapping (IDM) [12] (IDM is an instance of expansion coding [13]) to realize energy reductions and lifetime improvements over CME solutions such as DEUCE. CASTLE is agnostic to the choice of pattern-based compression, and is evaluated using both frequent pattern compression (FPC) [14] and base-delta-immediate (BΔI) compression [15]. On all writes, data undergoes compression prior to CME and transmission to the NVM DIMM. An IDM codec in the NVM DIMM controller selectively applies IDM to compressed encrypted data; uncompressed encrypted data is encoded using classical binary coding. CASTLE is a drop-in security preserving NVM CME solution, and is also compatible with soft/hard error detection and correction support (ECC, ECP, etc.) integral to NVMs.

Our system-level simulations using MARSS [16] and DRAMSim2 [17], and workloads from the SPEC CPU2006 [18] benchmark suite considers a 64-bit word, 64-byte cache line system that integrates TLC RRAM [12]. Our results show that CASTLE improves IPC by 1.6× and 1.3× and memory bandwidth by 2.1× and 1.3× in comparison to classical binary coding and state-of-the-art DEUCE [11], respectively (without exception, all evaluations integrate DCW [8] to update only the modified cells in the NVM array). For a deep, multi-billion-instruction evaluation of CASTLE, NVMain [19] is used to process the memory traces of SPEC CPU2006 benchmarks generated using the Intel Pin toolset [20]. Simulations of these traces show that FPC-CASTLE/BΔI-CASTLE that integrates IDM(8,4), reduces total write energy by 33.5%/25.4% and 23.3%/14% over classical binary coding and DEUCE, respectively. Simultaneously, FPC-CASTLE/BΔI-CASTLE reduces write latency by 43%/48% and 36%/42% on average in comparison to classical binary coding and DEUCE, respectively. Finally, we show that CASTLE improves TLC RRAM lifetime by 1.8× over both classical binary coding and DEUCE.

Section 2 is a background on data encryption for security in MLC/TLC NVMs. Section 3 describes CASTLE with examples. Section 4 presents evaluation results. Section 5 presents conclusions.

## 2 Background and motivation

Data security concerns of confidentiality, integrity, and availability in modern computing systems are further exacerbated by data

[1]The increased cell updates due to data encryption was not a serious limitation in DRAM due to its low write latency/energy and high endurance over NVMs.

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 Association for Computing Machinery.

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196007

persistence in NVMs. In practice, a secure computing system is designed assuming a trusted computing base (TCB), which is the section of a computing system considered to be secure, and a set of valid threats, termed as the threat model of the secure system [21].

This work uses the secure processor paradigm for its TCB: the processor, on-chip cache system, and on-chip memory controller are considered secure, while the memory bus and the main memory that are off-chip are considered vulnerable to attacks [11]. Along the lines of earlier research, this work focuses on a threat model comprising of two important data confidentiality attacks in NVMs: the stolen-DIMM and the bus-snooping attacks [5, 7, 11, 22]. In the stolen-DIMM attack, the adversary gains physical access to the DIMM [11, 22]; this is of particular concern in NVM-based systems, where data persists in the memory after power down. In the bus-snooping attack an adversary can obtain memory traces—address, data, access times, etc.—by simply probing the bus between the main memory and the processor-side memory controller [5, 7].

**Counter mode encryption (CME)** [2]**:** Memory security solutions based on the advanced encryption standard (AES) rely on a one-time pad (OTP) generated using a secret global key stored in the processor-side memory controller (TCB) to encrypt memory for security. Since an OTP generated using only a global key cannot successfully secure the data from dictionary attacks, it is customary to use the cache line address along with the global key for OTP generation; however, this still leaves the system vulnerable to bus-snooping attacks, where consecutive accesses to the same location can be monitored to decrypt data using simple XORs [11].

Modern CME solutions thwart both dictionary and bus-snooping attacks by using a counter along with the global key and cache line address for OTP generation, as shown in Fig. 1 [11]. Depending on the underlying counter architecture, CME solutions can be classified into global counter CME (GC-CME) and cache line counter CME (LC-CME). GC-CME uses a single large global counter located in the processor's memory controller (TCB), and increments it on each memory write for OTP generation; the counter value used for encryption is stored with the encrypted cache line for decryption on reads. In contrast, LC-CME maintains one counter per cache line in the main memory; on a write to a cache line, the counter is fetched to the processor, incremented, used for encryption, and written back along with the ciphertext.

**GC-CME vs. LC-CME:** Since GC-CME schemes store the global counter on the processor, OTP generation can begin as soon a write is initiated in the memory controller. In contrast, LC-CME schemes need to first initiate a memory read to fetch the counter value associated with the cache line, increment it, and only then begin the OTP generation process. Therefore, LC-CME incurs the overhead of a read before every write, which can severely limit the performance of LC-CME schemes in practice, especially in NVMs.

In contrast, since GC-CME schemes increment the global counter on every write, the global counter is larger in size in comparison to LC-CME cache line counters. GC-CME counter sizing is determined by practical memory overhead limits, but this has to be carefully balanced against making the GC-CME system prone to frequent counter overflows. GC-CME counter overflows are a serious security vulnerability, and the solution is to re-encrypt the entire memory using a new global key, freezing the memory system for several seconds. In Sec. 4.1, we discuss these tradeoffs and determine practical counter sizes for all the LC-CME and GC-CME architectures considered in this work.

**Counter cache CME (CC-CME):** CC-CME solutions improve the performance of the read-before-write-limited LC-CME schemes by caching counters in the processor's memory controller [5, 7]. On a CC hit, as in GC-CME, OTP generation can begin immediately on write initiation in the memory controller; on a CC miss, as in LC-CME, the cache line counter is first read from memory. CC-CME thus approaches the performance of GC-CME for overheads comparable to LC-CME.

**Dual counter encryption (DEUCE) [11]:** CME solutions mostly evolved in the context of DRAM; however, as data persistence in NVMs brought security issues in NVMs to the forefront, these CME solutions were applied directly to SLC NVMs at high cost to energy, latency, and endurance. DEUCE [11] is the state-of-the-art LC-CME solution for NVM security that explicitly addresses these energy and latency overheads. DEUCE uses a block-level read-decrypt-modify-write sequence, where the cache line and cache line counter are read to the memory controller (TCB) to recover the original plaintext, which is compared to the new plaintext so that only those parts of the ciphertext that correspond to modified data blocks are written to the memory. Note that although DEUCE can leverage the advantages of a CC (from CC-CME schemes), the only benefit is parallelization of OTP generation with data (i.e., ciphertext) access on CC hits; while this is advantageous on reads, writes are still limited by the mandatory data (i.e., ciphertext) access integral to the read-decrypt steps regardless of a CC hit/miss.

DEUCE uses two OTPs for encrypting a cache line: the first OTP is for encryption of the first write in an epoch (32 writes) whereas the second OTP is for encryption of the data chunks on subsequent writes that differ from the first write. As a result, a cache line stored using DEUCE can be encrypted using two OTPs, which requires disambiguation during decryption; this ambiguity is resolved by recording the locations of the modified data blocks using between 64 to 8 'tracking' bits per cache line for word sizes ranging from 8 to 64 bits, respectively.

[2] **Abbreviations:** Global counter CME (GC-CME), cache line counter CME (LC-CME), and counter cache CME (CC-CME).

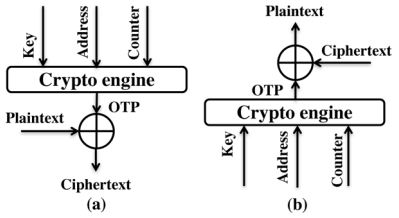


Figure 1: **AES counter mode (a) encryption and (b) decryption using a global key, line address, and line counter.**

## 3 Contributions

This work identifies and addresses two important open problems. First, NVM security solutions such as DEUCE are fundamentally limited by the read-decrypt-modify-write sequence, which is core to its energy and latency reduction. This motivates NVM encryption architectures that eliminate the read-decrypt steps in favor of

a simple write-only approach, which can reduce the energy and latency of CME schemes such as DEUCE. Second, in denser MLC/TLC NVMs, encryption increases cell updates from 50% in SLCs to 75%, and 87.5% of the total cells, respectively, with significant implications for memory energy, latency, and endurance.

CASTLE is a compression-based architecture that provides a block-level read-decrypt-free, i.e., write-only security solution for low latency, low energy, high endurance NVMs. CASTLE is compatible with both CC-CME and GC-CME architectures and leverages pattern-based compression and IDM to simultaneously reduce energy and latency, and improve IPC, bandwidth, and lifetime for no memory overhead over CC-CME/GC-CME.

## 3.1 'Read-decrypt'-free write

Consider Fig. 2, which simultaneously illustrates write scheduling for CASTLE, state-of-the-art DEUCE [11], CC-CME, and GC-CME NVM security architectures.

Figure 2(a) describes the write sequencing for state-of-the-art DEUCE [11]. During writes, the DEUCE memory controller first issues a read command to fetch both the cache line counter and the ciphertext. The ciphertext received from the NVM module is decrypted and compared with the new plaintext to identify the differing data blocks. The differing data blocks are then encrypted using a new OTP and written to the main memory. As discussed earlier, even if DEUCE uses a CC to parallelize OTP generation with the memory access, it still needs to read the encrypted data from memory to realize its energy/latency advantages.

Figure 2(b) describes the write sequencing for CC-CME, which improves latency over LC-CME. During writes, CC-CME performs a CC lookup of the required counter. On a CC hit, CC-CME uses a write-only approach; on a CC miss, CC-CME uses a counter read-before-write approach. CC-CASTLE is a CC-CME architecture that first compresses the write data using pattern-based compression (described in detail in Section 3.5) before following the timing sequence of CC-CME. Thus, CC-CME and CC-CASTLE enable 'read-decrypt-free' writes for a majority of write accesses.

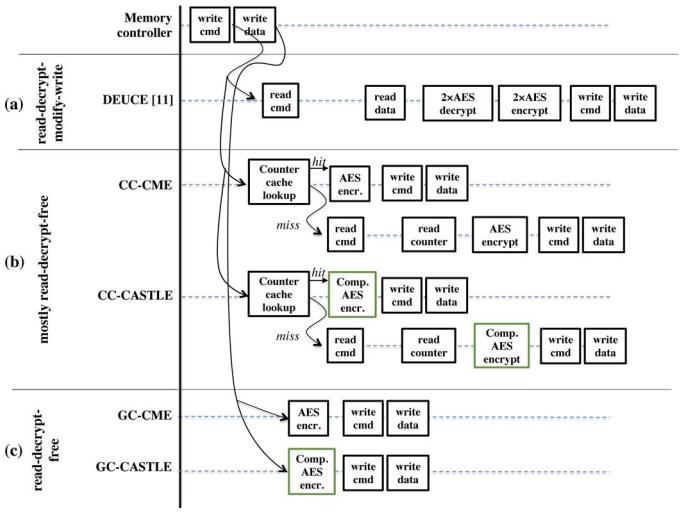


Figure 2: **Write sequencing in CASTLE in comparison to DEUCE [11], CC-CME, and GC-CME. CASTLE reduces write latency due to its ability to provide read-decrypt-free writes for both CC-CME and GC-CME architectures.**

Figure 2(c) describes the write sequencing for GC-CME and GC-CASTLE, both of which realize 'read-decrypt-free' writes. On writes, both schemes use the global counter in the memory controller to start OTP generation and data encryption immediately; this is followed by writes of the encrypted data to memory. Note that GC-CASTLE compresses the plaintext prior to the XOR with the OTP, which adds a negligible compression latency (evaluated in Sec. 3.3).

In summary, CASTLE completely sidesteps the read-decrypt-modify-write sequence of state-of-the-art LC-CME solutions such as DEUCE [11] to provide a low latency read-decrypt-free (mostly read-decrypt-free) solution for GC-CME/LC-CME (CC-CME).

## 3.2 CASTLE: Low latency/energy encoding

Although moving away from the read-decrypt-modify-write sequence enables CASTLE to improve IPC and bandwidth, it increases memory energy and latency due to the increased cell updates; CASTLE integrates pattern-based compression [14, 15, 23] and expansion coding [13], which generalizes incomplete data mapping (IDM) [12], to address these challenges.

**Data compression:** CASTLE leverages the compressibility of program data to realize its energy/latency wins. Program data exhibits regularities (redundancies) due to which we can compress cache lines to a smaller size. However, this regularity is lost when data is encrypted, since encryption scrambles the data to remove all regularity. Therefore, CASTLE first attempts to compress the incoming data using a cache line compression scheme (frequent-pattern compression (FPC) [14] and base-delta-immediate (BΔI) compression [15] in this work); if compression is successful, then the compressed data is encrypted by XORing it with the OTP, which is truncated to the size of the compressed data—similar to DEUCE. However, if the data is not compressible, then the entire cache line is encrypted as-is.

Note that CASTLE's effectiveness depends only on the compressibility of the incoming write data, i.e., it is independent of the previous data that is existing in the memory location, enabling it to be a read-decrypt-free scheme. Further, once the data is in the NVM DIMM, it is selectively encoded using IDM depending on its compression status (IDM of compressed data is discussed following the illustrative example below). Note that CASTLE requires 1 tag bit to indicate if compression is successful/unsuccessful; we show later in this section that this binary tag bit can be absorbed into the last TLC without incurring any overhead.

**Example:** Without loss of generality, consider a 64-bit example shown in Fig. 3 illustrating the 3 data encoding schemes considered in this paper—DCW [8], DEUCE [11], and CASTLE for TLC RRAM (TLC energy and latency parameters are from [12]).

Fig. 3(a) corresponds to the existing data in memory—plaintext data, OTP-1 (the OTP used to encrypt the existing data in the memory), the ciphertext (existing data in the memory), and the TLC map (classical binary coding of 3 logical bits per physical TLC). The data set shown in Fig. 3(b) corresponds to the new data—plaintext, the new OTP (OTP-2 ≠ OTP-1), and the ciphertext.

Figure 3(c) illustrates the encoding using DCW [8]. Due to data scrambling, which results in high cell updates, all 22 TLCs require cell updates using DCW (updated TLCs are in red). The high cell updates translates to an energy of 337.6 pJ and latency of 150 ns.

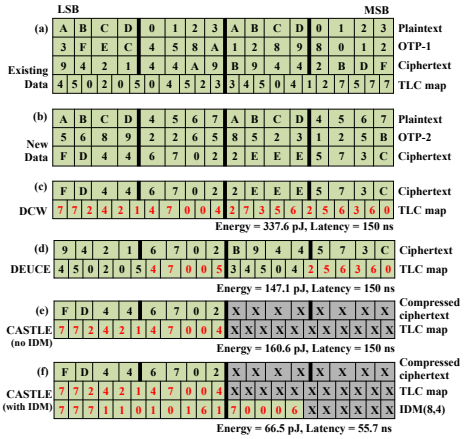


Figure 3: **Illustration of cell updates, energy, and latency for DCW, DEUCE, and CASTLE for 64-bit input data.**

In contrast, Fig. 3(d) illustrates DEUCE [11] operating on 16-bit boundaries. DEUCE identifies that only two 16-bit blocks have been modified in the new plaintext data. Therefore, only the TLCs corresponding to the differing blocks are updated (shown in red). Hence, the modify-write step of DEUCE reduces the energy to only 147.1 pJ, while having no impact on latency.

Finally, Fig. 3(e) illustrates data encoding with CASTLE. CASTLE uses FPC to compress the data to half its original size, which requires updates to only 11 of the 22 TLCs; the energy expense is 160.6pJ and there is no impact on latency. Although this example shows that the energy and latency reduction of CASTLE is comparable to state-of-the-art DEUCE, CASTLE leverages the observation that a large fraction of the TLCs remain unused after compression (see Fig. 3(f)). CASTLE uses IDM (described below) to realize simultaneous energy and latency improvements over DEUCE.

**Incomplete data mapping (IDM) [12] (an instance of expansion coding [13]):** The iterative program-&-verify (P&V) procedure used for programming MLC/TLC NVMs results in the central MLC/TLC states requiring more energy and latency in comparison to the terminal states. The energy/latency numbers for TLC RRAM show that the energy and latency to program the central states can be about 24× and 12× in comparison to the terminal states, respectively. Such large disparity in the energy and latency of states motivates data encoding using only the low energy states, while avoiding the high energy states altogether to reduce energy and latency. IDM allows the use of only the desired MLC/TLC states with no modifications to the write circuitry.

An IDM($m$,$n$) code encodes data using only $n$ low-energy states in an $m$-ary NVM cell. For example, IDM(8,4) operates on an 8-level TLC NVM by encoding data using only 4 low-energy states out of the total 8 states [12]; this corresponds to the $(3,2)_8$ expansion code [13]. Since IDM(8,4) excludes 4 out of 8 states, we need more TLCs to encode the data. Furthermore, since $8^2 = 4^3$, 6 logical bits that originally required 2 TLCs require 3 TLCs using IDM(8,4).

For the example in Fig. 3(e), the integration of IDM(8,4) to the compressed data leverages the unused TLCs for energy and latency gains. Figure 3(f) illustrates the integration of IDM(8,4), which results in an energy expense of only 66.5 pJ and latency of only 55.7 ns. This translates to a 5× (2.7×) and 2.2× (2.7×) reduction in energy (latency) over DCW and DEUCE, respectively.

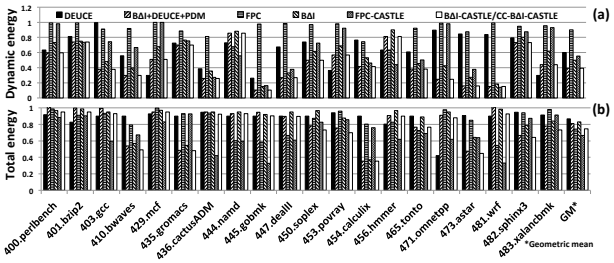


Figure 4: **(a) Dynamic energy and (b) total energy (using NVMain [19]) for DEUCE [11], FPC, BΔI, FPC-CASTLE, and BΔI-CASTLE/CC-BΔI-CASTLE[8], normalized to classical binary coding. CASTLE reduces write energy by 53% (21%) and total energy by 29% (19%) over binary coding (DEUCE).**

**No tag overhead:** Although CASTLE requires a single tag bit in the data field (512 bits) to record the outcome of compression, it is concatenated with the data (513 bits) and absorbed into the last TLC at no cost $\lceil 513/3 \rceil = \lceil 512/3 \rceil = 171$ TLCs. Since $2^n$ ($n > 1$) can never be a multiple of 3, the tag bit can always be encoded within the existing TLCs without any overhead for word sizes that are integer powers of 2. Further, to prevent the tag bit from becoming a latency bottleneck, it is encoded using the lowest latency states of the TLC—'0' and '7'—when the data is in IDM(8,4) form.

**CASTLE security:** Both GC-CASTLE and CC-CASTLE preserve the security of the underlying CME architecture. Similar to DEUCE [11], CASTLE uses a minimum data size of 64 bits—achieved through zero padding—for all encryption before data transmission from the processor to the memory module. This ensures that there is no leak of the pattern used for compression even in the case where the compressed plaintext data is less than 64 bits.

**CASTLE reliability:** CASTLE is compatible with both ECC [24] and ECP [25], which are integral to soft/hard error detection and correction (EDAC) in MLC/TLC NVMs. NVMs with EDAC usually use separable coding, i.e., codes such that the data field is stored separate from the EDAC field. Since a write may alter both the data and the EDAC fields, the benefits of encoding the data field using IDM(8,4) may be nullified by high latency writes in the EDAC field. To address this potential shortcoming, we propose that whenever the data field is compressible, the EDAC field be written in IDM(8,4) form. The additional cells required for encoding the EDAC field using IDM(8,4) can be obtained from the residual cells after compression of the data field. BΔI-CASTLE leaves 24 TLCs (there are 30 free TLCs, but 6 are used to store the counter value in IDM(8,4) form) unused in the worst case, which can be provisioned to provide EDAC in IDM(8,4) to preserve/extend the latency/energy benefits of CASTLE. However, if the data field is incompressible, both the data and EDAC fields are written using classical binary coding.

# 4 Evaluation and results

## 4.1 Simulation configurations

As motivated in Sec. 2, GC-CME, LC-CME, and CC-CME are sensitive to counter size. In DEUCE [11], a 32-bit counter with 8 'tracking' bits (40 bits in total) per cache line was derived as an appropriate choice for LC-CME. The same configuration is used for DEUCE and the LC-CME CASTLE architectures in this paper. For fairness in comparison, we use a 40-bit counter for the GC-CME

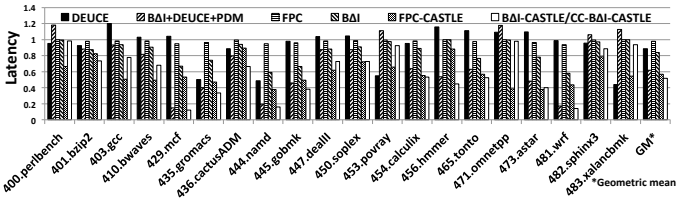


Figure 5: **Normalized latency for DEUCE [11], FPC, BΔI, FPC-CASTLE, and BΔI-CASTLE/CC-BΔI-CASTLE[8]. CASTLE reduces latency by 45.7% (38.7%) over the baseline (DEUCE).**

CASTLE architectures; this has a system availability of 0.9997 and a re-encryption time of 4.4s for 16GB (derivations omitted for brevity). Furthermore, we use a 32-way 512KB set-associative CC (90%+ hit rate). The following schemes are evaluated in this work:

- Classical binary coding (baseline) uses 40-bit GC-CME (≡ CASTLE without compression and IDM).
- Read-decrypt-modify-write uses 32-bit LC-CME to demonstrate the impact of this sequence on performance.
- State-of-the-art DEUCE [11] uses 32-bit LC-CME with 8 'tracking' bits per 64-bit word in a 512-bit cache line.
- BΔI+DEUCE+IDM, which illustrates how CASTLE enhances DEUCE, uses BΔI to compress a 512-bit cache line before DEUCE-based encryption; the compressed encrypted data is then encoded using IDM(8,4).
- FPC/BΔI schemes integrate pattern-based data compression with 40-bit GC-CME (≡ CASTLE without IDM(8,4)).
- FPC-CASTLE/BΔI-CASTLE: CASTLE architectures integrating 40-bit GC-CME and IDM(8,4). Note that as appropriate and for brevity, we use CASTLE to refer to the average results of FPC-CASTLE and BΔI-CASTLE.
- CC-BΔI-CASTLE integrates BΔI-CASTLE with CC-CME (32-bit counter and 512KB CC).

Note that without exception, all the schemes integrate DCW [8] to update only the modified cells in the NVM array.

## 4.2 Memory-level energy/latency

**Summary:** Table 1 summarizes and compares the total module energy, array-level dynamic energy, array-level dynamic latency, and memory overhead for the architectures considered in this paper. In summary, CASTLE reduces the memory array write energy by 53% and 21% and write latency by 45.7% and 26.4% in comparison to classical binary coding and state-of-the-art DEUCE, respectively.

**Energy:** Our framework tracks all cell writes from the start of program execution to compute the cumulative energy. Note that the static energy of the peripheral circuits and the memory array are indirectly influenced by the reduction in the latency of each write operation. A lower write latency translates to a lower energy to keep the peripheral circuits active, which is evaluated using NVMain [19]. Figures 4(a) and 4(b) show the memory array dynamic energy and the total memory module energy for all the architectures, normalized to classical binary coding. The last entry is the geometric mean (GM) of the energy reduction across all the benchmarks; this is equivalent to simulating all these benchmarks for the same execution time.

Our simulations show that BΔI+DEUCE+IDM reduces the memory array dynamic (total) energy by 60.7% (19%) and 34.5% (6.6%) over classical binary coding and DEUCE, respectively. Second, FPC-CASTLE (BΔI-CASTLE/CC-BΔI-CASTLE) reduces the memory array dynamic energy by 44.7% (60.7%) and 7.7% (34.5%) over classical binary coding and DEUCE, respectively. Finally, FPC-CASTLE (BΔI-CASTLE/CC-BΔI-CASTLE) reduces total energy by 33.4% (25.4%) and 23.2% (14%) over classical binary coding and DEUCE, respectively. Note that since BΔI-CASTLE and CC-BΔI-CASTLE have identical memory array operations, they have identical memory array energy; further, BΔI-CASTLE reduces total energy by 8% over BΔI+DEUCE+PDM due to the static energy reduction during the read-decrypt stage of writes.

**Latency:** Due to the iterative P&V procedure, the NVM states with lower energy also require lower write latencies (Sec. 2), which enables CASTLE (using IDM(8,4)) to simultaneously reduce both energy and latency. Our simulations determine the latency for each access by tracking the maximum latency cell write for the access (since the latency for writing a word depends on the cell that requires the longest latency). The overall latency for program execution is then cumulatively computed using the latency of individual accesses. Figure 5 first shows that BΔI+DEUCE+IDM reduces the write latency by 38.7% and 29.9% over classical binary coding and DEUCE, respectively. Second, FPC-CASTLE (BΔI-CASTLE) is able to reduce the write latency by 43.1% (48.3%) and 35.6% (41.6%) over classical binary coding and DEUCE, respectively. Note that FPC-CASTLE and BΔI-CASTLE outperform DEUCE and BΔI+DEUCE+-IDM due to the additional read-decrypt latency incurred by DEUCE and BΔI+DEUCE+IDM during writes.

## 4.3 System-level IPC, bandwidth, and lifetime

CASTLE is evaluated using (i) instructions-per-cycle (IPC), which reflects system-level performance, and (ii) main memory bandwidth, which reflects main memory performance.

**Summary:** Table 1 summarizes and compares IPC and bandwidth for all the encryption schemes considered in this paper. CASTLE improves IPC by 68.5% and 26.7% in comparison to classical binary coding and DEUCE, respectively.

**Instructions-per-cycle (IPC):** To evaluate the impact of CASTLE on IPC, we use a system-level simulator based on MARSS [16] and DRAMSim2 [17]. We simulate nine composite workloads, each with 4 benchmarks from the SPEC CPU2006 [18]. Figure 6 shows the IPC for each benchmark in each workload. The last set of bars in figure 6 represents the mean IPC for all the workloads.

| | Trace-based simulation | | | | System-level simulation | |
|---|---|---|---|---|---|---|
| | Total Energy | Dyn. Energy | Dyn. Latency | Ovhd. | IPC | B/w. |
| Classical binary coding | 100% | 100% | 100% | 7.8% | 100% | 1× |
| read-decrypt-modify-write | 100% | 60.0% | 100% | 6.25% | 92.8% | 1.1× |
| DEUCE [11] | 86.7% | 60% | 88.5% | 7.8% | 133% | 1.7× |
| BΔI+DEUCE+IDM | 81% | 39.2% | 62% | 7.8% | 146% | 1.8× |
| 64-bit FPC compression | 74.4% | 90.1% | 97.9% | 7.8% | 141% | 1.9× |
| BΔI compression | 82.9% | 89.7% | 84.1% | 7.8% | 129% | 1.8× |
| FPC-CASTLE | 66.5% | 55.3% | 56.9% | 7.8% | 168% | 2.0× |
| BΔI-CASTLE | 74.6% | 39.2% | 51.6% | 7.8% | 169% | 2.3× |
| CC-BΔI-CASTLE† | | | | 6.25% | 152% | 1.9× |

†CC-BΔI-CASTLE and BΔI-CASTLE differ in system-level operations, but they have identical array-level operations.

Table 1: **Traced-based and system-level simulation results normalized to baseline. Note that all the cases integrate DCW [8] to update only the modified cells in the NVM array.**

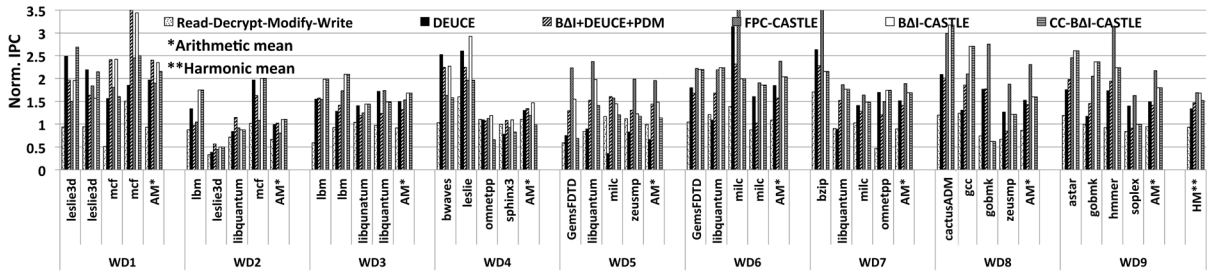


Figure 6: **System-level IPC results (MARSS+DRAMSim2) normalized to classical binary coding. The last bar in each workload represents the arithmetic mean of the IPCs of individual benchmarks. The last set of bars represent the harmonic mean of IPCs for each workload. CASTLE shows an IPC improvement of 68.5% (26.7%) over binary coding with DCW (DEUCE). Furthermore, CC-BΔI-CASTLE improves the IPC by 52% (14%) over binary coding with DCW (DEUCE).**

To study the impact of the read-decrypt-modify-write sequence on IPC, we simulate the cache-line-level read-decrypt-modify-write encryption architecture using classical binary coding. Our simulations show that the read-decrypt-modify-write architecture can incur up to 50% IPC overhead ($WD_2$) in comparison to the write-only classical binary coding architecture. CASTLE improves IPC by 68.5%, 26.7%, and 15.4% in comparison to classical binary coding, DEUCE, and BΔI+DEUCE+IDM, respectively. Note that although BΔI-CASTLE and BΔI+DEUCE+IDM have similar dynamic write latency, the read-decrypt-free write procedure of BΔI-CASTLE enables it to improve the IPC by 15.4% over BΔI+DEUCE+IDM. Furthermore, our simulations show that CC-BΔI-CASTLE improves IPC by 52%, 14%, and 4% over classical binary coding, DEUCE, and BΔI+DEUCE+IDM, respectively; the IPC improvement of CC-BΔI-CASTLE is about 90% in comparison to BΔI-CASTLE.

**Memory bandwidth:** Figure 7 shows the main memory bandwidth across nine composite workloads normalized to the baseline (binary encoding with DCW). Our simulations show that CASTLE improves the average memory bandwidth by 2.2×, 1.3×, and 1.1× over classical binary coding, DEUCE, and BΔI+DEUCE+IDM, respectively. Furthermore, CC-BΔI-CASTLE shows bandwidth improvement of 1.9×, 1.1×, and 1.05× over classical binary coding, DEUCE, and BΔI+DEUCE+IDM, respectively.

**Lifetime:** We theoretically evaluate the lifetime gains of CASTLE using TLC RRAM as an example—along the lines of [26–28]—showing that CASTLE increases memory lifetime by 1.8× over both classical binary coding and DEUCE.

## 5 Conclusions

CASTLE is a compression-based read-decrypt-free, i.e., write-only secure main memory architecture for low latency, low energy, high endurance NVMs. CASTLE adopts a block-level write-only sequence to eliminate the read latency of the read-decrypt steps in state-of-the-art NVM security solutions. Whereas a write-only approach increases cell updates, energy, and latency, CASTLE integrates pattern-based compression and incomplete data mapping (i.e., expansion coding) to improve energy, latency, and lifetime over state-of-the-art NVM security solutions.

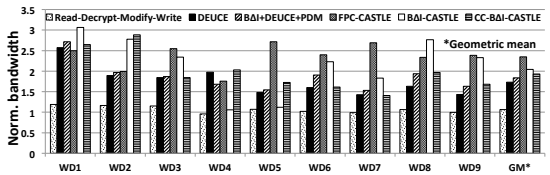


Figure 7: **Normalized memory bandwidth: CASTLE improves bandwidth by 1.2× (2.2×) over the baseline (DEUCE).**

## References


[1] B. Lee *et al.*, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. ISCA*, 2009.
[2] I. G. Baek *et al.*, "Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses," in *Proc. IEDM*, 2004.
[3] "International technology roadmap for semiconductors," 2011.
[4] G. E. Suh *et al.*, "Efficient memory integrity verification and encryption for secure processors," in *Proc. MICRO*, 2003.
[5] J. Yang *et al.*, "Improving memory encryption performance in secure processors," *IEEE Trans. Computers*, 2005.
[6] G. Duc and R. Keryell, "CryptoPage: An efficient secure architecture with memory encryption, integrity and information leakage protection," in *Annual Computer Security Applications Conference*, 2006.
[7] C. Yan *et al.*, "Improving cost, performance, and security of memory encryption and authentication," in *ACM SIGARCH Computer Architecture News*, 2006.
[8] B. Yang *et al.*, "A low power phase change random access memory using a data-comparison write scheme," in *Proc. ISCAS*, 2010.
[9] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proc. MICRO*, 2009.
[10] J. Kong and H. Zhou, "Improving privacy and lifetime of PCM-based main memory," in *Proc. DSN*, 2010.
[11] V. Young *et al.*, "DEUCE: Write-efficient encryption for non-volatile memories," in *Proc. ASPLOS*, 2015.
[12] D. Niu *et al.*, "Low power multi-level-cell resistive memory design with incomplete data mapping," in *Proc. ICCD*, 2013.
[13] P. M. Palangappa and K. Mohanram, "CompEx: Compression-expansion coding for energy, latency, and lifetime improvements in MLC/TLC NVM," in *Proc. HPCA*, 2016.
[14] A. Alameldeen and D. Wood, "Frequent pattern compression: A significance-based compression scheme for L2 caches," 2004. Univ. Wisconsin-Madison TR1500.
[15] G. Pekhimenko *et al.*, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proc. PACT*, 2012.
[16] A. Patel *et al.*, "MARSS: A full system simulator for multicore x86 cpus," in *Proc. DAC*, 2011.
[17] P. Rosenfeld *et al.*, "DRAMSim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, 2011.
[18] "SPEC CPU2006," 2006.
[19] M. Poremba and Y. Xie, "NVMain: An architectural-level main memory simulator for emerging non-volatile memories," in *Proc. Annual Symposium VLSI*, 2012.
[20] C.-K. Luk *et al.*, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proc. PLDI*, 2005.
[21] R. B. Lee, "Security basics for computer architects," *Synthesis Lectures on Computer Architecture*, 2013.
[22] S. Chhabra and Y. Solihin, "i-NVMM: A secure non-volatile main memory system with incremental encryption," in *Proc. ISCA*, 2011.
[23] D. Dgien *et al.*, "Compression architecture for bit-write reduction in non-volatile memory technologies," in *Proc. NANOARCH*, 2014.
[24] M. Awasthi *et al.*, "Efficient scrub mechanisms for error-prone emerging memories," in *Proc. HPCA*, 2012.
[25] S. Schechter *et al.*, "Use ECP, not ECC, for Hard Failures in Resistive Memories," in *Proc. ISCA*, 2010.
[26] M. K. Qureshi *et al.*, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *ACM SIGARCH Computer Architecture News*, 2010.
[27] C. Xu *et al.*, "Understanding the trade-offs in multi-level cell ReRAM memory design," in *Proc. DAC*, 2013.
[28] L. Jiang, Y. Zhang, and J. Yang, "ER: Elastic RESET for low power and long endurance MLC," in *Proc. ISLPED*, 2012.