ANCHOR: ARCHITECTURE FOR SECURE NON-VOLATILE MEMORIES

by

Shivam Swami

B. Tech., Guru Gobind Singh Indraprastha University, 2012

M. Tech., Indian Institute of Technology, Kharagpur, 2014

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2018

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Shivam Swami

It was defended on

June 12, 2018

and approved by

Kartik Mohanram, Ph.D., Associate Professor
Department of Electrical and Computer Engineering

Jun Yang, Ph.D., Professor
Department of Electrical and Computer Engineering

Zhi-Hong Mao, Ph.D., Professor
Department of Electrical and Computer Engineering

Natasa Miskov-Zivanov, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Rami Melhem, Ph.D., Professor
Department of Computer Science

Yiran Chen, Ph.D., Associate Professor
Department of Electrical and Computer Engineering, Duke University

Dissertation Director: Kartik Mohanram, Ph.D., Associate Professor
Department of Electrical and Computer Engineering

ANCHOR: ARCHITECTURE FOR SECURE NON-VOLATILE MEMORIES

Shivam Swami, PhD

University of Pittsburgh, 2018

Computing systems that integrate advanced non-volatile memories (NVMs) are vulnerable to several security attacks that threaten (i) data confidentiality, (ii) data availability, and (iii) data integrity. This dissertation presents ANCHOR, which integrates 4 low overhead, high performance security solutions SECRET, COVERT, ACME, and STASH to thwart these attacks on NVM systems.

SECRET is a low cost security solution for data confidentiality in multi-/triple-level cell (i.e., MLC/TLC) NVMs. SECRET synergistically combines (i) smart encryption, which prevents re-encryption of unmodified or zero-words during a write-back with (ii) XOR-based energy masking, which further optimizes NVM writes by transforming a high-energy ciphertext into a low-energy ciphertext. SECRET outperforms state-of-the-art encryption solutions, with the lowest write energy and latency, as well as the highest lifetime.

COVERT and ACME complement SECRET to improve system availability of counter mode encryption (CME). COVERT repurposes unused error correction resources to dynamically extend time to counter overflow of fast growing counters, thereby delaying frequent full memory re-encryption (system freeze). ACME performs counter write leveling (CWL) to further increase time to counter overflow, and thereby delays the time to full memory re-encryption. COVERT+ACME achieves system availability of 99.999% during normal operation and 99.9% under a denial of memory service (DoMS) attack. In contrast, conventional CME achieves system availability of only 85.71% during normal operation and is rendered non-operational under a DoMS attack.

STASH is a comprehensive end-to-end security architecture for state-of-the-art smart hybrid memories (SHMs). STASH integrates (i) CME for data confidentiality, (ii) page-level Merkle Tree authentication for data integrity, (iii) recovery-compatible MT updates to withstand power/system

failures, and (iv) page-migration friendly security meta-data management. For security guarantees equivalent to state-of-the-art, STASH reduces memory overhead by 12.7×, improves system performance by 65%, and increases NVM lifetime by 5×.

This dissertation thus addresses the core security challenges of next-generation NVM systems. Directions for future research include (i) holistic architectures for both security and reliability of smart memories, (ii) applications of ANCHOR to reduce security overhead of Internet-of-Things, and (iii) secure non-volatile processors, especially in the light of advanced attacks like Spectre and Meltdown.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

Firstly, I would like to express my sincere gratitude to my Ph.D. advisor, Prof. Kartik Mohanram, for his support, patience, and motivation throughout this journey. Kartik's immense knowledge, high research standards, and professionalism have not only shaped this dissertation, but also my personality in these past 4 years. I could not have imagined having a better advisor and mentor for my Ph.D study.

I am also grateful to my Ph.D. committee – Prof. Rami Melhem, Prof. Yiran Chen, Prof. Jun Yang, Prof. Zhi-Hong Mao, and Prof. Natasa Miskov-Zivanov – for their encouragement and valuable input on my work. Without their blessings, this dissertation could not have seen the light of day. I am equally grateful to the department of Electrical and Computer Engineering at the University of Pittsburgh and all the people therein, especially Sandy Weisberg, for the help and cooperation in these past 4 years.

No thanks can do justice to the support and help extended to me by my labmate and housemate, Joydeep, who has been an integral part of this journey from day 1. I am also thankful to my senior labmate, Poovaiah, for providing highly collegial atmosphere and getting me started at Pitt. I consider myself privileged to have been a part of this research group.

Last but not the least, I acknowledge the support and encouragement of my parents, who have always motivated me to go the extra mile and achieve excellence in my pursuits. I am also thankful to my life partner, Gunjan, and my brother, Arjun, for providing me the much needed emotional support during difficult times. And to conclude, I dedicate my doctorate to my grandparents, whose love and blessings have turned this dream into reality.

1.0 INTRODUCTION

The high power consumption and poor potential for technology scaling below 22nm of DRAM [1] has spurred research in emerging resistance-class non-volatile memories (NVMs) such as phase change memory (PCM) [2,3], resistive RAM [4,5], spin-transfer torque RAM (STT-RAM) [6], and 3D X-Point [7]. These NVMs store data by modulating the resistance (PCM/RRAM/3D X-Point) and magnetoresistance (STT-RAM) of the storage material, with data persistence on power down. Due to a large separation between the lowest and the highest resistance state, PCM, RRAM, 3D X-Point, and STT-RAM also support multi-level/triple-level cell (MLC/TLC) operation, which is the ability to store 2/3 logical bits per physical cell. Whereas these NVMs offer several advantages over DRAM, computing systems that integrate these advanced NVMs are vulnerable to several security attacks that threaten (i) data confidentiality, (ii) data availability, and (iii) data integrity. Data confidentiality attacks aim to obtain secret data stored in the system [8–13]. Data availability attacks seek to make the memory system unavailable for authorized users [14, 15]. Finally, data integrity attacks refer to any adversarial corruption or tampering of data [14, 16–18].

A broad class of memory encryption techniques have been proposed in the literature [8–12] to protect NVMs against data confidentiality attacks. Encryption algorithms demonstrate strong diffusion characteristics that ensure that a single bit change in the plaintext (i.e., unencrypted data) results in several bit changes in the ciphertext (i.e., encrypted data). Due to strong diffusion characteristics, the theoretical average cell flips per write for encrypted SLC/MLC/TLC NVM is 0.5/0.75/0.875. This renders cell flip reduction techniques like data comparison write (DCW, i.e., classical read-modify-write) [19] and flip-n-write (FNW) [20] ineffective in practice, increasing the write energy/latency and reducing the lifetime of NVMs.

Furthermore, state-of-the-art encryption techniques [8, 11, 12] are based on the principles of counter mode encryption (CME). CME associates a counter with each cache line and uses this

counter along with the memory address of the cache line and a secret key to encrypt the cache line on a memory write. To reduce on-chip memory overhead in CME, the counters are stored in main memory and cached on-chip in a counter cache to improve performance [11, 21–23]. However, CME suffers from the counter overflow problem, in which a counter overflow mandates full memory re-encryption with a new secret key, causing the system to freeze for the duration of full memory re-encryption. Counter overflow renders CME vulnerable to the denial of memory service (DoMS) attack, which threatens system availability. In a DoMS attack, a malicious application can render the memory system unavailable to other applications by forcing frequent full memory re-encryption due to counter overflow. A DoMS attack can be easily engineered using cache eviction and ordering instructions (like `clflush` and `mfence`) that can be executed in non-administrator mode to constantly write to the same cache line in main memory, forcing its counter to overflow. Whereas system availability can be improved by employing large counters, this increases the memory overhead of CME and results in poor performance due to frequent counter cache misses (for a fixed counter cache size).

Although CME ensures data confidentiality, it does not guarantee data data integrity, which refers to ability to detect any adversarial corruption or tampering of data. For data integrity, memory authentication is performed using the Merkle Tree (MT) authentication [16–18, 24–27]. In MT authentication, a logical tree – obtained by recursively computing message authentication codes (MACs, i.e., keyed hash values) over memory blocks – is maintained in the main memory [16, 21]. The integrity of the fetched data is verified by computing the corresponding chain of MACs up to the MT root, which is maintained on the processor-side memory controller. Due to its hierarchical structure, MT significantly increases memory overhead and memory accesses, negatively impacting performance.

In addition to these direct attacks on data confidentiality, availability, and integrity, memory systems are also vulnerable to side-channel attacks that exploit memory access patterns to obtain secret encryption/authentication keys [28–30]. Concealing the true memory access patters by employing techniques like Oblivious RAM [31] thwarts access-pattern based side-channel attacks; however, it increases memory traffic by 10× and deteriorates performance by 4× [28, 29, 31].

This dissertation presents ANCHOR, which integrates 4 low overhead, high performance security solutions SECRET, COVERT, ACME, and STASH to thwart these attacks on NVM sys-

tems. SECRET synergistically integrates smart encryption with energy masking to reduce write energy/latency of CME. COVERT complements SECRET to improve system availability of CME-based secure NVMs. ACME further complements COVERT to thwart DoMS attacks on NVMs. Finally, STASH is a comprehensive end-to-end security architecture for smart hybrid memories that integrates (i) CME for data confidentiality, (ii) low overhead page-level Merkle Tree (MT) authentication for data integrity, (iii) recovery-compatible MT updates to withstand power/system failures, and (iv) page-migration-friendly security meta-data management. The rest of this introduction summarizes the core contributions of SECRET, COVERT, ACME, and STASH and also outlines directions for future research.

1.1 CONTRIBUTIONS

1.1.1 SECRET

Smartly EnCRypted Energy efficienT (SECRET) NVMs synergistically integrate zero-based partial writes with XOR-based energy masking to realize low-overhead CME for MLC/TLC NVMs. Smart encryption integrates word-level re-encryption and zero-based partial writes in order to reduce memory writes. Word-level re-encryption modifies classical CME—which is used at cache line granularity—by allocating separate counters for each word, allowing data encryption at granularities smaller than a cache line. Zero-based partial writes leverages the fact that a significant fraction of the plaintext written to the memory is zero for real-life workloads [32–36]. SECRET uses a one-bit zero-flag per word to track zero-words (i.e., words with only zeros) in a cache line and maintain zero-words in their last encrypted states, saving the write overhead of re-encrypting zero-words. Following smart encryption, SECRET performs write optimization by filtering the encrypted words (i.e., the ciphertext) through *energy masks*. For an overhead of one bit (energy-flag) per word, XOR-based energy masks transform high energy states in the ciphertext into low energy states, reducing the overall write energy of the cache line. Both the zero-flag and the energy-flag are stored in encrypted state in the memory in order to ensure the security of the data.

SECRET is evaluated on MLC/TLC RRAM architectures using the NVMain [37] memory simulator on memory traces from the SPEC CPU2006 benchmark suite [38]. SECRET considers advanced encryption standard-based (AES-based) CME as the baseline. SECRET is compared to state-of-the-art encryption techniques, namely BLE [11] and DEUCE [12], which perform cache line encryption at a granularity of 128 bits and 16 bits, respectively. Simulations on MLC RRAM show that BLE, DEUCE, and SECRET reduce write energy (latency) by 40% (23%), 40% (17%), and 80% (37%) over the baseline. The lifetime evaluations show that BLE, DEUCE, and SECRET improve MLC RRAM lifetime by 35%, 36%, and 63% over the baseline. Furthermore, for TLC RRAM, BLE, DEUCE, and SECRET reduce write energy (latency) by 33% (31%), 40% (23%), and 63% (49%), respectively, over the baseline. The lifetime improvements for TLC RRAM from BLE, DEUCE, and SECRET are 18%, 24%, and 56%, respectively, over the baseline.

Thus, by integrating smart encryption with energy masking, SECRET reduces CME overheads for MLC/TLC NVMs. SECRET outperforms BLE and DEUCE, with the lowest write energy/latency, as well as the highest lifetime.

1.1.2 COVERT

Counter OVErflow ReducTion (COVERT) is an encryption solution that addresses the counter overflow problem in CME by performing on-demand memory allocation to the fast-growing counters, while also retaining the area/performance benefits of small counters. ANCHOR integrates COVERT with SECRET to provide a holistic platform to improve the system availability, performance, lifetime, and reduce the write energy/latency of CME-based secure NVMs. At its core, COVERT employs dynamic counters (DYNAMO henceforth) to reduce frequent full memory re-encryption due to small-sized counters. DYNAMO leverages the fact that a significant fraction of memory provisioned for error correction remains unutilized till very late in memory lifetime [39, 40]. DYNAMO repurposes unused error correction memory cells to the overflowing counters, thereby delaying the mandatory full memory re-encryption on a counter overflow and improving system availability. COVERT is a drop-in replacement for classical CME, and does not compromise the security of the underlying CME (i.e., COVERT preserves CME requirements of (i) encryption inside a secure processor and (ii) spatial/temporal exclusivity of the OTP).

COVERT is evaluated on a phase change random access memory (PCRAM) architecture [41] using the MARSS full-system simulator [42] on both integer and floating-point workloads from the SPEC CPU2006 benchmark suite [38]. COVERT employs 16-bit CME as the underlying encryption technique. The results show that for equivalent overhead and no impact to performance, COVERT improves system availability from 85.71% (of classical 16-bit CME [8]) to 99.3%. Thus, COVERT provides an effective solution to address the counter overflow problem in CME without compromising its security.

1.1.3 ACME

Advanced Counter Mode Encryption, i.e., ACME, complements COVERT to realize low overhead, high performance NVM security solution that is robust to both data confidentiality and system availability attacks. At its core, ACME leverages the underlying wear leveling architecture—employed to improve NVM endurance [2,43,44]—to perform counter write leveling (CWL). CWL associates counters with physical addresses (PAs) instead of logical addresses (LAs) such that when a frequently written cache line is translated from one PA to another PA for wear leveling, its associated counter is also remapped, leading to a distribution of writes across counters. Multiple counters together track the number of writes seen by a frequently written cache line, thereby delaying counter overflow; delaying counter overflow improves system availability due to delayed full memory re-encryption. ACME incurs no logic overhead, since it enables CWL by leveraging the existing wear leveling architecture that is integrated with the secure processor. ACME is a drop-in replacement for CME, and does not compromise the security of the underlying CME, i.e., ACME preserves CME requirements of (i) encryption inside a secure processor and (ii) spatial as well as temporal exclusivity of the OTP.

ACME is evaluated on a PCRAM architecture using (i) a trace-driven memory simulator that integrates the Intel Pin [45] binary instrumentation tool and (ii) the MARSS [42] full-system simulator on SPEC CPU2006 benchmarks [38]. Results show that for the system availability of 99.999%, ACME not only requires 50% lower counter memory overhead, but also improves system performance by 20% in comparison to classical CME. Further, When subject to a DoMS attack in the form of an unprivileged Linux process that sidesteps all levels of cache to constantly write to

the same memory address to precipitate counter overflow, the ACME-based system provides 99.9% system availability in contrast to a classical CME-based system that is rendered non-operational. Finally, upon integration with COVERT, ACME improves COVERTs system availability from 99.3% to 99.999% for no overhead.

1.1.4 STASH

Smart hybrid memories (SHMs) that integrate NVM, DRAM, and processor logic can provide high bandwidth, low memory latency, and high memory density to meet the needs of future high-performance computing systems. However, the unsecure DRAM, NVM, and/or memory buses in SHMs are vulnerable to data confidentiality attacks (e.g., memory scanning and bus snooping) [10, 12, 14, 26], data integrity attacks (spoofing/splicing/replay attacks) [17, 27–29], and side-channel attacks (e.g., access-pattern based attacks) [28,29] that must be addressed prior to commercialization. *SecuriTy Architecture for Smart Hybrid memories (STASH)* is the first comprehensive end-to-end security solution that makes the following three core contributions.

Low-cost page-level MT authentication (PMT) that replaces classical cache-line-level authentication to secure the SHM from data integrity attacks. PMT leverages page granularity data migration between DRAM and NVM to reduce Merkle Tree (MT) authentication overheads for thwarting data tampering attacks. Bonsai MT (BMT) [17] framework is the norm for memory authentication that is adopted by all state-of-the-art memory security techniques [26,27,29]. BMT requires a 128-bit data message authentication code (DMAC) per 512-bit cache line along with an MT constructed over encryption counters (counters henceforth). In contrast, PMT maintains a 128-bit MAC per 4kB page and constructs an MT by recursively hashing these page-level MACs (PMACs), providing equivalent security guarantees for significantly lower memory and performance overheads.

Recovery-compatible MT updates (RECOUP) is an IDR solution that performs selective meta-data (counters, MT root, etc.) updates to tolerate power/system failures in SHMs. To improve performance, it is common practice to employ an on-chip write-back counter cache to delay security meta-data updates in memory [17, 26]. This renders the meta-data residing in the SHM partially stale and unsynchronized with the ciphertext. As a result, the SHM cannot reliably decrypt and/or authenticate the ciphertext in the event of a power/system failure. Extending BMT [17] to sup-

port IDR in SHMs significantly increases NVM writes, since every cache line write also requires multiple consistent MT updates. RECOUP leverages a key observation that IDR can be supported by consistently updating only (a) the MT root in the trusted computing base (TCB) of the smart DRAM and (b) the modified MT leaf (i.e., data/counter) in the smart NVM, thereby eliminating unnecessary MT re-computations and reducing high overhead NVM writes.

Page-migration-friendly security meta-data management (PACT) supports low overhead page migration is SHMs. Since state-of-the-art security solutions [16, 17, 26, 28, 29] are designed to operate at only cache-line granularity, the meta-data overhead (>1kB per 4kB page) of these solutions on page migration from smart NVM to smart DRAM increases memory traffic and reduces effective DRAM capacity. PACT addresses this problem by (i) transferring only the required PMT meta-data (16 bytes per 4kB page) to DRAM and (ii) caching the PMT meta-data of the migrated page in a PMAC cache on the logic layer of the smart DRAM. By eliminating bulk meta-data migration to DRAM, PACT reduces memory traffic and improves DRAM utilization in SHMs.

STASH is evaluated on an SHM that integrates a 2GB HMC [46] as the DRAM cache and a 32GB smart triple-level cell (TLC) PCM [47] as the main memory. NVMain memory simulator [37] and the MARSS [42] full system simulator are used for trace-based and system-level evaluations of STASH, respectively. STASH is compared to (i) state-of-the-art ObfusMem [29] and (ii) a strawman security architecture for SHMs (SSA-SHM henceforth). Results show that in comparison to ObfusMem (SSA-SHM), STASH reduces memory overhead by 12.7× (12.7×), improves system performance by 65% (25%), and increases NVM lifetime by 5× (5×).

Hence, for security guarantees equivalent to state-of-the-art, STASH achievies the best system performance, the highest NVM lifetime and the lowest memory overhead.

1.2 FUTURE WORK

Directions for future research include (i) exploration of holistic architectures that ensure both security and reliability of smart memory systems, as well as extending ANCHOR to further reduce the overheads of instant data recovery in smart NVMs. (ii) Investigating applications of ANCHOR to reduce security overhead of Internet-of-Things (IoT). IoT security involves secure data acquisi-

tion by the front-end devices, secure data forwarding on the network, and secure data processing in the back-end cloud. (iii) Extending ANCHOR to safeguard emerging non-volatile processors (NV-processors), especially in the light of advanced attacks like Spectre and Meltdown, which can be launched from the user space without invoking any administrative privileges.

1.3 DISSERTATION ORGANIZATION

The rest of this dissertation is organized as follows. Chapter 2 covers basic concepts of NVMs, describes various security vulnerabilities of NVM-based systems, and presents related work in NVM security. Chapter 3 describes the theory and architecture of SECRET, along with evaluation and results. Chapter 4 describes the theory and architecture of COVERT, along with evaluation and results. Chapter 5 describes the theory and architecture of ACME, along with evaluation and results. Chapter 6 describes theory and architecture of STASH, along with evaluation and results. Finally, Chapter 7 presents directions for future research.

2.0 BACKGROUND

This chapter is divided into two sections. The first section covers background material related to emerging NVMs to motivate NVM integration in modern computing systems. The second section discusses various security challenges that must be addressed prior to commercialization of these advanced memory technologies.

2.1 NVM BASICS

2.1.1 PCM basics

A PCM storage element consists of a resistive heater and a phase change material (chalcogenide) placed between two metal electrodes (refer figure 1 (a)). A PCM cell is a one resistor, one transistor device comprising of a PCM storage element and an access transistor (refer figure 1 (b)). PCM stores data by altering the resistance of its phase change material (e.g., $Ge_2Sb_2Te_5$ (GST)) between the fully amorphous state with high resistance (reset state) and the fully crystalline state with low resistance (set state) [2, 48]. To reset a PCM cell, a high amplitude reset pulse is applied to melt the GST and then abruptly cutoff to quench the molten GST into the amorphous state. To set a PCM cell, a small amplitude set pulse heats the GST above its crystallization temperature for a sufficiently long duration to transform the GST into the crystalline state. The information stored in a PCM cell is read by measuring the resistance of the cell.

2.1.2 RRAM basics

An RRAM cell consists of a transition metal oxide (TMO) that acts as an insulator, sandwiched between two metal electrodes, forming a Metal-Insulator-Metal (MIM) structure. Information is stored by switching the resistance of the TMO between the high resistance state (HRS, logic '0')

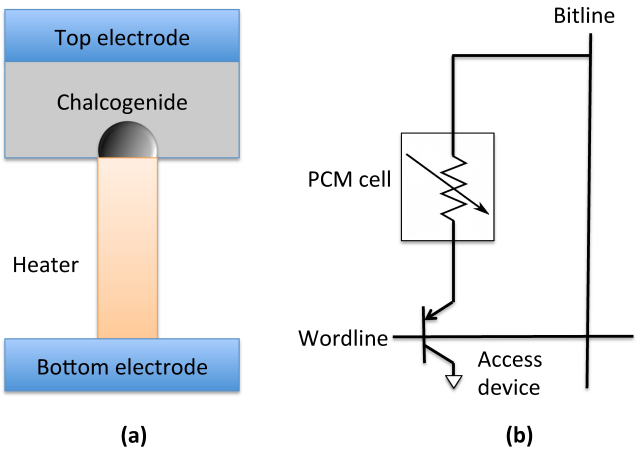


Figure 1: **(a) Physical structure of PCM (b) 1-bit PCM cell with a storage element and an access transistor [2]**

and the low resistance state (LRS, logic '1'). Applying an external voltage of specified polarity, magnitude, and duration results into the formation/destruction of one or more conductive filaments (CF) made out of oxygen vacancies [4, 5, 49]. Formation of the CFs establishes a conductive path between the top and the bottom electrode, bringing the cell into the LRS. In contrast, destruction of the CFs removes the conductive path, bringing the cell into the HRS (refer figure 2(a) and (b)). Based on the switching behavior, RRAM can be classified into two categories: bipolar and unipolar RRAM. For a unipolar RRAM cell, the magnitude and the duration of the external voltage applied across the cell alone controls the RRAM resistance switching. In contrast, for a bipolar RRAM cell, LRS to HRS switching (i.e., reset operation) and HRS to LRS switching (i.e., set operation) occur at different voltage polarities.

Two types of memory structures have been proposed in the literature for RRAM array: conventional MOSFET-accessed structure and crosspoint structure. Due to the large size of a MOSFET access device in comparison to an RRAM cell, the conventional MOSFET-accessed structure ceases the RRAM's area advantage. Therefore, architects prefer the area-efficient crosspoint RRAM array [4, 44]. In the crosspoint structure, each RRAM cell is sandwiched between a top electrode and a bottom electrode at each crosspoint of the array without an access device. In this structure, each cell only occupies an area of $4F^2$ (F is the feature size in fabrication technology), which is in theory the smallest cell area for a single-layer memory structure.

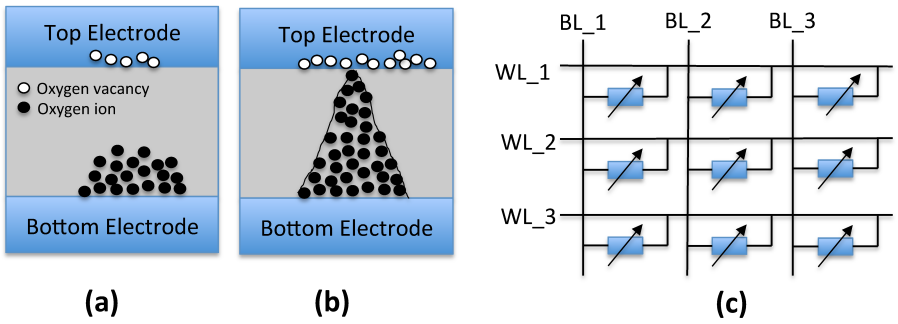


Figure 2: **Physical structure of an RRAM cell in (a) high resistance state (HRS) (b) low resistance state (LRS). (c) Schematic view of crosspoint RRAM array (WL = wordline, BL = bitline) [49]**

2.1.3 STT-RAM basics

An STT-RAM cell comprises of a magnetic tunnel junction (MTJ) connected in series with an access transistor. MTJ is basically an oxide layer sandwiched between two ferromagnetic layers [6, 50,51]. As shown in figure 3(a), the magnetic orientation of one of the ferromagnetic layers is fixed (*reference layer (RL)*), while the magnetic orientation of the other layer (*free layer (FL)*) is changed by applying an external field. Parallel magnetization of FL and RL results in a low resistance state of the MTJ and corresponds to logic '0'. In contrast, anti-parallel magnetization of FL and RL results in a high resistance state of the MTJ and corresponds to logic '1'. Figure 3(b) shows an STT-RAM bit cell. There are two approaches of reading an STT-RAM bit cell (i) parallel reading and (ii) anti-parallel reading. In parallel reading, select line (SL) is grounded and a small bias is applied to the bit line (BL). In anti-parallel reading, the voltage polarity of SL and BL are opposite and the current flows in reverse direction from SL to BL.

2.2 NVM SECURITY

This section covers the NVM threat model, the role of memory encryption and authentication in NVM security, and also discusses state-of-the-art NVM security solutions.

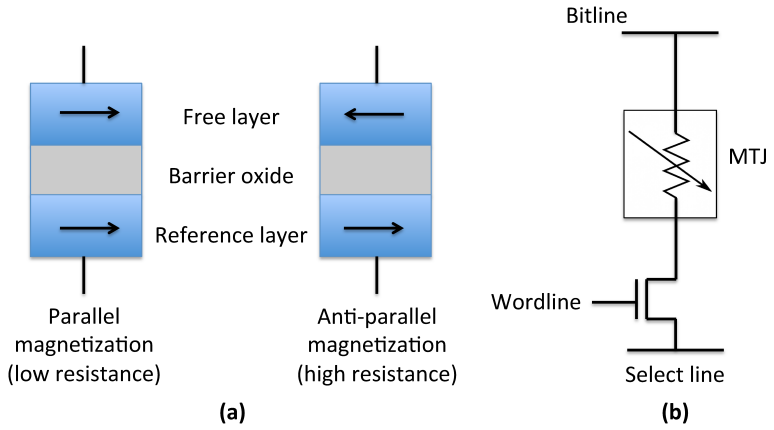


Figure 3: **(a) STT-RAM storing device (MTJ) (b) Structure of an STT-RAM cell with a storage element and an access transistor [6, 50]**

2.2.1 Threat model

The security of modern computing systems is based on the three cornerstone properties of confidentiality, availability, and integrity [14]. Since it is neither practically feasible, nor desirable, to design a system that can guarantee complete protection from all possible attacks to these security requirements, systems are designed only for a subset of attacks, with certain security assumptions to eliminate the possibility of other attacks that are not a part of the threat model [14, 21]. As in prior studies on memory security, the trusted computing base (TCB) consists of the processor and core parts of the operating system (e.g., security kernels); external memory and peripherals are assumed to be untrusted [8, 9, 11, 12]. This threat model is extended to include on-module processing logic on smart hybrid memories (discussed in detail in section 6.3).

2.2.2 Data confidentiality attacks

Past research [8–12] on NVM security has recognized the stolen DIMM and bus snooping attacks as the two most common security attacks to data confidentiality in NVMs. Unlike DRAM, where only specialized cold boot attacks [52] can potentially retrieve data after power down, data retrieval from powered-down NVMs is much easier due to data persistence. In the stolen DIMM attack, the attacker has physical access to the NVM DIMM, enabling them to stream data from the DIMM.

Data persistence of NVMs exposes data in the plaintext to attackers on power down. A similar problem has been addressed in disk storage through encryption, motivating encryption of NVMs. In the bus snooping attack, the attacker can acquire data by monitoring unsecured off-chip communications. Again, it is widely accepted that such attacks can be thwarted by implementing data encryption in the secure processor.

Data encryption for NVMs is generally achieved by applying a block cipher (e.g., AES) to the plaintext to transform it into the ciphertext using a secret key. However, direct encryption of data is vulnerable to dictionary-based attacks where an attacker can compare encrypted data to figure out which lines store the same content. Hence, recent research [12,21,22,26,53] advocates the use of counter mode encryption (CME) as a secure memory encryption technique that is robust against dictionary-based attacks.

2.2.3 Data availability attacks

With the advent of row-hammer-like attacks [54–56], NVM-based main memories have also become vulnerable to denial of memory service (DoMS) attacks that threaten system availability. In a DoMS attack, a malicious application can render the memory system unavailable to other applications by forcing frequent full memory re-encryption due to counter overflow (a well-known limitation of CME [16, 17]). DoMS attacks can be easily engineered using cache eviction and ordering instructions (like clflush and mfence) that can be executed in non-administrator mode to constantly write to the same cache line in main memory, forcing its counter to overflow. Split-counter mode encryption (split-CME) [16] protects DRAM-based main memories against DoMS attacks; however, directly extending split-CME to NVM-based main memories undermines the ability of NVMs to ensure data recovery in the face of power/system failures. Past work, e.g., MECU [8] and i-NVMM [10], has motivated that preserving the data recovery property of NVMs is an important design goal for any NVM security solution.

2.2.4 Data integrity attacks

Data integrity attacks (spoofing/splicing/replay) refer to the tampering of data in order to compromise the security of the computing system on which the data is stored [11, 14, 16, 18, 21, 21, 26].

Spoofing attacks: Arbitrary data tampering by adversary constitute spoofing attacks. These attacks can potentially disrupt the normal system operation or reveal confidential information stored in the system.

Splicing or relocation attacks: Unauthorized copying (also swapping) of data from one memory address to another constitute splicing attacks. Such an attack may be viewed as a spatial permutation of memory blocks.

Replay attacks: Replacing a memory block's data with a valid older value constitute replay attacks. A memory block located at a given address is recorded and inserted at the same address at a later point in time. Such an attack may be viewed as a temporal permutation of a memory block, for a specific memory location.

2.2.5 Counter mode encryption

In counter mode encryption (CME), a block cipher is used to encrypt a seed with a secret key (stored on the processor) to produce a one-time pad (OTP). This OTP is bitwise XORed with the plaintext to generate the ciphertext. During decryption, the same OTP is XORed with the ciphertext to obtain the plaintext. The spatial and temporal exclusivity of the OTP is critical for the security of CME and requires that (i) the OTPs should be unique for different cache lines and (ii) the OTPs for a particular cache line should be unique for every write. These unique OTPs are generated from unique seeds that have two components: (i) the cache line address (to ensure spatial exclusivity) and (ii) a counter, which is incremented on each write (to ensure temporal exclusivity).

To reduce on-chip memory overhead in CME, the counters are stored in main memory and cached in an on-chip counter cache to improve performance [12, 22, 26]. Figure 4 shows CME in the presence of a counter cache. To increase the counter cache hit rate, small counters are preferred in practice, since a fixed size counter cache can store more counters if each counter is small. However, small counters can quickly overflow in the presence of high memory write traffic. Conventionally, a counter overflow is handled by changing the secret key to prevent reuse of OTPs [16]. However, since the same secret key is shared by every cache line, a change of secret key requires the entire memory to be re-encrypted, causing the system to freeze for the duration of full memory re-encryption. Whereas large counters can be employed to delay counter

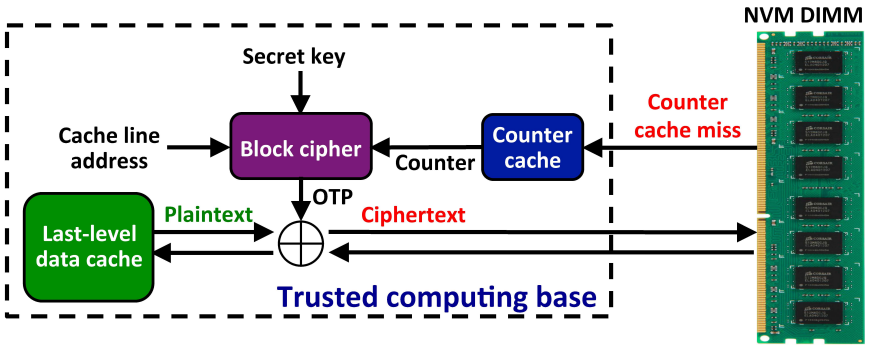


Figure 4: **This figure depicts counter mode encryption, which uses a counter, line address, and secret key to generate a one-time pad (OTP). During encryption (decryption), the OTP is XORed with the plaintext (ciphertext) to generate the ciphertext (plaintext).**

overflow, large counters increase the memory overhead of CME and result in poor performance due to frequent counter cache misses. Thus, CME imposes heavy overheads on memory, performance, and availability in practice.

To address CME overheads, the authors in [16] proposed split-counter mode encryption (split-CME). Split-CME is a 2-level counter based encryption that employs a 7-bit minor counter per 512-bit cache line and a 64-bit major counter per 4kB page (i.e., net memory overhead is 1.56% per cache line). For OTP generation, both major and minor counters are concatenated. A minor counter is incremented on every write to its cache line and a minor counter overflow is handled by: (i) resetting all the minor counters in a 4kB page to 0 (ii) increasing the pages major counter by 1 and (iii) re-encrypting the entire 4kB page using the new counter values. Whereas split-CME improves system availability by reducing the cost of a counter overflow from full memory re-encryption to a 4kB page re-encryption, it requires special support from the operating system to prevent OTP reuse for cache lines with the same virtual address (if the virtual address is used for OTP generation); if the physical address is used for OTP generation, split-CME needs bookkeeping for reliable decryption of the pages that are moved between main memory and the disk.

To eliminate the dependence of the OTP on the cache line memory address, the authors in [17] proposed address-independent seed encryption (AISE). AISE repurposes split- CME's 64-bit major counter per page to a logical page identifier (LPID) that is unique for every page. While generating the OTP for a cache line, AISE uses the LPID and the cache line page offset within a page as

the memory address. On a minor counter overflow, similar to split-CME, AISE re-encrypts the corresponding page by assigning it a new LPID and resetting all the minor counters. Thus, AISE retains the performance and availability benefits of split-CME for no hardware/software overhead to support cache line memory address (physical/virtual).

2.2.6 Merkle Tree authentication

Merkle Tree (MT) authentication is the most widely used scheme for main memory authentication [14, 16, 17, 26]. In MT authentication, a message authentication code (MAC) generated by a cryptographic hash function (e.g., NIST-approved SHA-1/2/3) using a secret key is computed for each cache line and stored along with the cache line in the main memory. For protection against spoofing (arbitrary data tampering) and splicing (replacing a cache line with another cache line) attacks, MAC generation includes the cache line data and memory address, respectively. On every read, the MAC is computed for the fetched data and compared with the stored MAC value. If the stored MAC is equal to the recomputed MAC, fetched data integrity is established; else, tampering is detected and the system raises an exception.

Although single-level MACs are robust against spoofing and splicing attacks, they are ineffective against replay attacks wherein an adversary replaces the data and counter with their older values. To protect memory from replay attacks, MT maintains a hierarchical tree structure of MACs, with the data and counter as its leaf nodes. The root of this tree is stored on the secure processor. A cache line write/read propagates MAC update/verification up the tree to the root. Since the on-chip MAC root is the cryptographic signature of the entire memory, a replay attack by an adversary is always detected.

2.2.7 Bonsai Merkle Tree authentication

Due to its hierarchical structure, MT significantly increases memory overhead and memory accesses, negatively impacting performance. To reduce memory and performance overheads of MT, the authors in [17] proposed Bonsai Merkle Tree (BMT). In BMT authentication, an MT is maintained only for counters instead of both data (cache lines) and counters; a cache line is protected by only a single-level MAC, which is computed using the ciphertext (i.e., cache line data), cache

line memory address, and its counter value. BMT significantly reduces memory/performance overhead of MT authentication without compromising security. As in CME (which caches counters in the on-chip counter cache), the BMT nodes (i.e., counter hashes) are also cached in the on-chip counter cache to further improve system performance [17, 26, 57].

2.3 RELATED WORK

Initial proposals to protect data confidentiality, e.g., execute-only memory (XOM) [58] used direct encryption that incurred high encryption/decryption latency. Early tamper-evident systems like AEGIS [59] incurred high authentication overhead due the use of hash trees. To reduce the performance overhead of memory encryption/decryption, the authors in [21, 22] proposed OTP-based CME, which removes the encryption/decryption process from the critical path of LLC miss handling (as explained in section 2.2.5). The authors in [53] proposed a counter prediction and OTP pre-computation mechanism to improve CME performance without employing a counter cache. The performance and availability of CME were further improved by state-of-the-art split-CME [16] and AISE [17] by using a 2-level counter design (explained in detail in section 2.2.5).

Memory authentication architectures have evolved from performance-intensive and memory-intensive integrity trees like Merkle Tree [24, 25], Parallelizable Authentication Tree (PAT) [60], and Tamper-Evident Counter tree (TEC-Tree) [61] to state-of-the-art Bonsai Merkle Tree [17], which significantly reduces memory/performance overhead of authentication. Furthermore, to remove the integrity verification process from the critical path of LLC miss handling, PoisonIvy [57] proposes safe speculative execution while performing integrity verification in the background.

In addition, several NVM-centric encryption solutions, e.g., incremental NVMM encryption (i-NVMM) [10], block-level encryption (BLE) [11], and dual-counter encryption (DEUCE) [12] have been proposed to account for high write energy and latency, and low endurance of NVMs. Techniques like i-NVMM perform memory-side (i.e., encryption unit is placed in memory) incremental encryption by keeping only the frequently accessed memory pages in unencrypted form in the memory and securing rest of the memory in encrypted form. However, these techniques suffer from security vulnerabilities due to partial memory encryption and unencrypted off-chip communi-

cations. Processor-side encryption techniques like BLE [11] and DEUCE [12] have been proposed to address the security vulnerabilities of i-NVMM by placing the encryption unit in the secure processor. Additionally, in order to reduce the encryption penalty (i.e., high write energy/latency and cell flip rate), these techniques encrypt only the modified words in a cache line and allow the unmodified words to remain in their last encrypted state (word-level read-modify-write). Whereas state-of-the-art security solutions have focused on reducing the encryption penalty (increased write energy/latency and reduced memory lifetime) in single-level cell (SLC) NVMs, the realization of low encryption penalty solutions for multi-/triple-level cell (MLC/TLC) secure NVMs as well as protecting NVMs against data availability and side-channel attacks remain areas of active research and development.

3.0 SECRET: SMARTLY ENCRYPTED ENERGY EFFICIENT NON-VOLATILE MEMORIES

3.1 SECRET: MOTIVATION

Encryption algorithms demonstrate strong diffusion characteristics that ensure that a single bit change in the plaintext results in several bit changes in the ciphertext. Due to strong diffusion characteristics, encryption renders cell-level write reduction techniques like FNW [20] and cell-level DCW [19] ineffective in practice, increasing the average cell flips per write operation. As shown in figure 5(a), average cell flips per write in encrypted SLC NVM is 0.5, and it increases to 0.75 (0.875) in encrypted MLC (TLC) NVM. High average cell flips per write results in increased write energy/latency and reduced memory lifetime.

Furthermore, writing to an MLC/TLC requires configuring the cell into a target resistance range by using state-of-the-art program-and-verify (P&V) [5, 36, 62, 63]. Typically, P&V requires several iterations of read-verify-write to bring an MLC/TLC to the desired state. The main drawback of P&V is that programming an MLC/TLC to intermediate states consumes more write energy and latency in comparison to programming it to the terminal states. Table 1 shows that the energy/latency required to configure an RRAM TLC to states 3 and 4 are 10× in comparison to the terminal states

Table 1: **Latency/energy for TLC RRAM program-and-verify [5, 63]**

State	0	1	2	3	4	5	6	7
Latency (ns)	15.2	46.8	98.3	143	150	101	52.7	12.1
Energy (pJ)	2.0	6.7	19.3	35.1	35.6	19.6	6.1	1.5

Figure 5(b) illustrates the impact of encryption on write energy of MLC/TLC RRAM NVMs. Simulations on SPEC CPU2006 benchmarks demonstrate that on average, write energy of encrypted MLC (TLC) NVMs is 8.6× (8.9×) in comparison to unencrypted MLC (TLC) NVMs.

In summary, memory encryption is indispensable to protect NVMs against security vulnerabilities, but it comes at the cost of increased energy/latency and reduced lifetime.

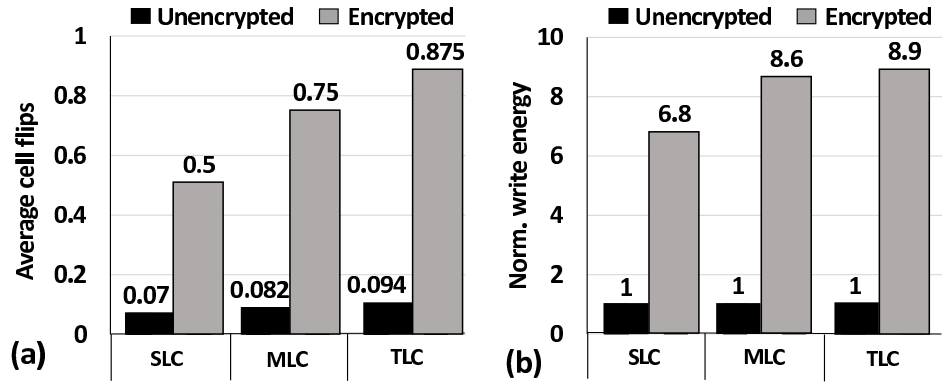


Figure 5: **This figure reports (a) verage cell flips per write and (b) geometric mean of write energy (normalized to unencrypted memory) for SPEC CPU2006 benchmarks for SLC/MLC/TLC RRAM. Average cell flips per write for encrypted SLC/MLC/TLC NVMs is 0.5/0.75/0.875. High average cell flips increase write energy/latency and reduce lifetime of NVMs. In comparison to unencrypted SLC/MLC/TLC NVM, energy of encrypted SLC/MLC/TLC NVM is 6×/8.6×/8.9×.**

3.2 SECRET: CONTRIBUTIONS

This section describes SECRET, a low hardware, low memory overhead scheme which significantly reduces the cell flips, decreases the write energy/latency, and improves the lifetime of MLC/ TLC NVMs without compromising its security.

3.2.1 Smart encryption

Without exception, counter mode encryption ensures data security during the write-back of a cache line to main memory by re-encrypting all the words in a cache line with a new OTP. However, this introduces a high encryption penalty of increased cell flips leading to increased write energy/latency and reduced lifetime of NVM. SECRET proposes smart encryption to reduce the encryption penalty of counter mode encryption as follows:

- Whereas classical counter mode encryption uses a single counter for an entire cache line, SECRET allocates separate counters for each word, allowing word-level re-encryption at granularities smaller than a cache line.
- Zero-based partial encryption, where zero-words (i.e., words with only zeros) are maintained in their last encrypted states, saving the write overhead of re-encrypting zero-words.

During the write-back operation of a cache line, the majority of the words remain unmodified [11, 12]. Smart encryption preserves these words in their previous encrypted states, without compromising the security of the data. This is referred to as word-level data comparison write (DCW) in this work. SECRET performs word-level DCW to prevent undesired cell flips by blocking the re-encryption of the unmodified words. To perform word-level DCW, SECRET allocates a separate local counter to each word in a cache line, which is updated only when that particular word is modified during a write-back. SECRET also maintains a global counter for the entire cache line. During a write-back, each modified word is re-encrypted with an OTP, which is generated using a counter value obtained from the concatenation of the global counter and the updated local counter for that word, whereas the unmodified words are left unchanged.

The local counter (LC) associated with any word overflows when the corresponding word observes higher number of writes than what the local counter can handle (LC_{max}). For e.g., a 2-bit

local counter can handle at most (2^2=) 4 writes of the corresponding word before it overflows. Whenever a local counter in a cache line overflows, it compromises security because of the pad-reuse. To avert this situation and preserve the security in the event of a local counter overflow, smart encryption takes the following measure: all the local counters are reset to 0, the global counter is incremented by 1, and the entire cache line is re-encrypted using the new global and local counters. Thus, smart encryption prevents the re-encryption of unchanged non-zero-words without compromising the security of the data in memory.

To further reduce the encryption penalty of increased cell flips, SECRET leverages the fact that a significant fraction ($\approx$ 60% of the bytes) of the plaintext written to the memory is zero [33], and prevent the re-encryption of this zero data. SECRET provisions a single flag-bit for each word in a cache line, which records the status (i.e., zero/non-zero) of the word. SECRET sets the zero-flag of a word to 1 whenever a zero is written to it and leaves the word in its previous encrypted state in main memory during write-back. Note that zero-flags are written to the memory in encrypted state, which prevents an attacker from learning if the word written was zero (discussed in depth in Sec. 3.2.2). Since a majority of data during a write-back is zero, preventing zero-word re-encryption further assists in reducing cell flips due to encryption.

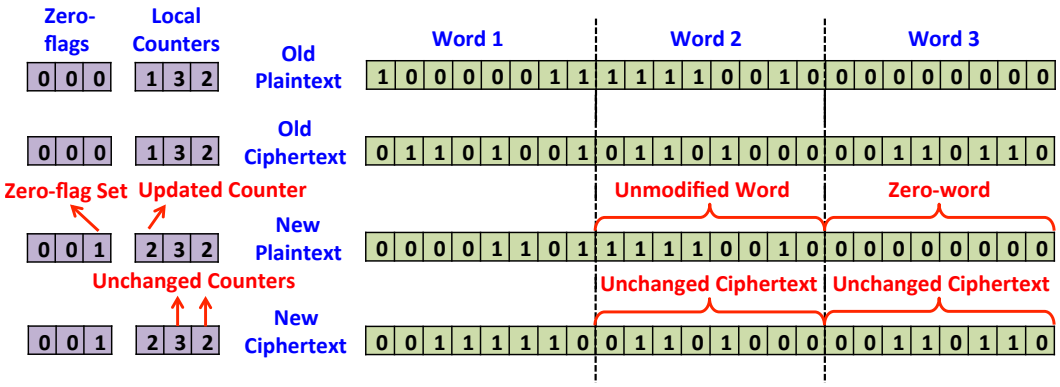


Figure 6: **Smart encryption, which prevents the re-encryption of unmodified words (word 2) and zero-words (word 3). Each word is allocated a local counter that facilitates smart encryption, and is updated only when the word is modified. The zero-flags track zero-words in a cache line; zero-flag = 1 for word 3 in the figure.**

Figure 6 illustrates smart encryption. Without loss of generality, SECRET assumes a 24-bit cache line, with a word size of 8 bits and a local counter size of 2 bits. The first two lines show the previous plaintext, and the corresponding ciphertext that was written-back to main memory. The initial states of the local counters for words 1–3 (left to right) are 1, 3, and 2 respectively. The zero-flags of all the words are set to 0 initially. When the new data (plaintext) arrives, it is observed that word 2 has the same data, whereas word 3 has all zeros. Hence, the words are left in their respective previous encrypted state (unchanged ciphertext) in the main memory. The local counter for words 2 and 3 are unchanged and the zero-flag for word 3 is set to 1. Only word 1 is modified to a new non-zero value; hence its local counter is updated, and the new ciphertext corresponding to the updated counter is written to main memory. Thus, smart encryption successfully prevents re-encryption of unmodified/zero-words, thereby reducing cell flips and resulting in lower write energy/latency as well as improved lifetime.

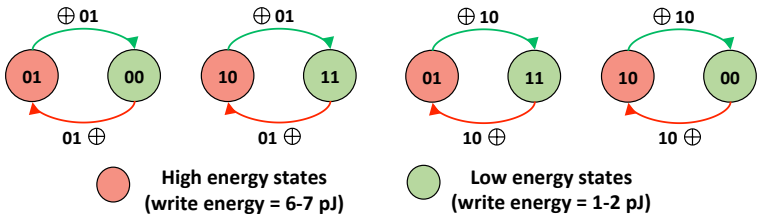


Figure 7: **Transition from high to low and low to high energy states using XOR operation. Energy values are from [5, 63].**

3.2.2 Energy masks

In order to ensure the security of the ciphertext, encryption algorithms introduce a high degree of confusion in the ciphertext [64]. Confusion refers to making the relationship between the secret key and the ciphertext as complex and as involved as possible. Thus, instead of long runs of zeros or ones (which are often seen in the plaintext), a large number of 01/10 pairs is seen in the ciphertext. Since programming an MLC into 01/10 states consumes 3-4$\times$ more energy in comparison to 00/11 states [5, 63], writing the ciphertext to memory is highly energy intensive in practice.

Whereas smart encryption is effective in reducing cell flips and therefore the energy of the unchanged and the zero-words, words which change are re-encrypted and written to the memory. To reduce the write energy of these re-encrypted words, SECRET incorporates post-encryption write optimization by filtering the words through energy masks. For every n-bit word (ciphertext), SECRET uses an n-bit full energy mask with $n/2$ pairs of 01 (high energy state), which are XORed with the ciphertext to transform the high energy states in the ciphertext into low energy states (i.e., states 00 or 11). Figure 7 depicts the XOR-based state transition of high energy states to low energy states and vice versa. Note that either state 01 or 10 can be used in the energy mask; this work uses 01 for the analysis. Additionally, SECRET uses an n-bit half energy mask with the upper $n/2$ bits as 00 and the lower $n/2$ bits as $n/4$ pairs of 01. The half energy mask covers the cases where high energy states are concentrated in the lower half of the word and not uniformly distributed.

Each word in encrypted cache line is XORed in parallel with a full and a half energy mask. Thereafter, the write energy of the full-mask word, half-mask word, and unmasked word are compared to identify the best mask for each word, as shown in figure 8. Continuing the example from figure 6, figure 9 illustrates the concept of energy masking. Word 1 is already in a low energy state,

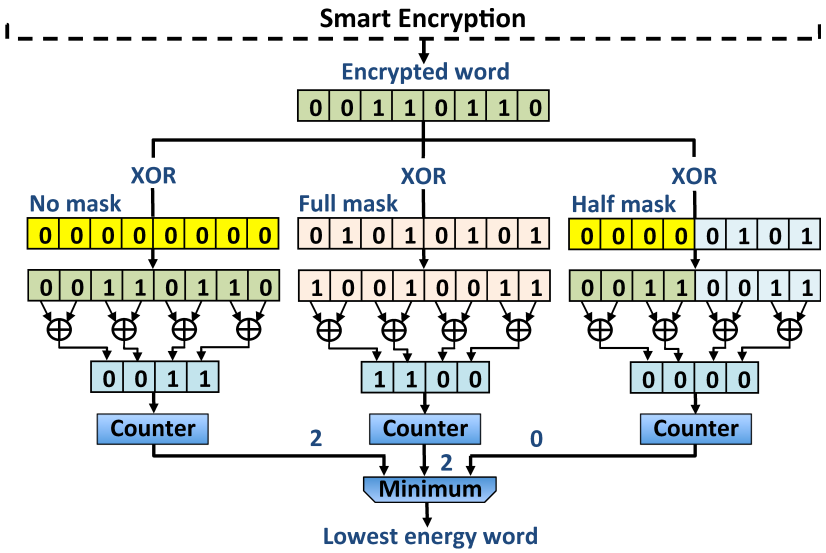


Figure 8: **Energy masking module to select the lowest energy mask for a word. The mask which gives the minimum number of high energy states is selected. Note that for parallel operation, a separate energy masking module is present for each word.**

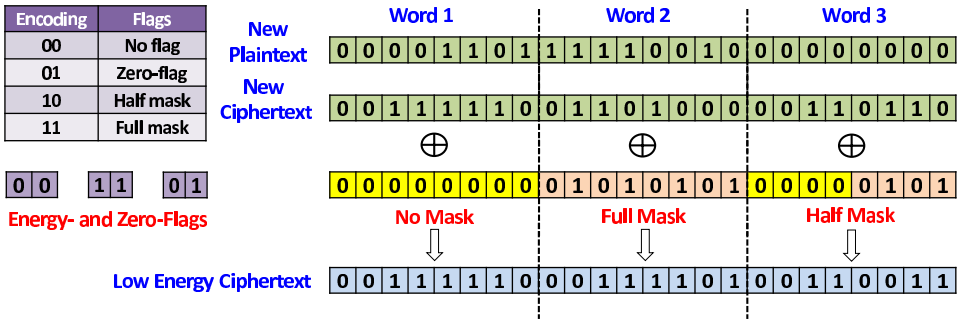


Figure 9: **This figure depicts the energy masks for 3 different words from figure 6. Each word is XORed with an energy mask such that the overall write energy of the word is minimized. Note that if the zero-flag for a word is set, the energy mask for that word is not recorded in the flag-bits. The low energy ciphertext (LEC) obtained from application of the energy mask gets written on the NVM.**

hence it is left unmodified. Word 2 uses a full mask and word 3 uses a half mask to reduce the ciphertext into a low energy ciphertext (LEC), which is finally written to the NVM. The information about energy mask and zero-word are encoded in a 2-bit flag for each word as follows: 00 – no flag, 01 – zero word, 10 – half energy mask, and 11 – full energy mask. Note that if the zero-flag is set for a word, then the energy mask information for that word is not required for decryption. The combined memory overhead for smart encryption and energy masking is 6.25% per cache line (refer Sec. 3.2.4). Hence, we need only 2 bits to uniquely detect 4 different states for a word.

3.2.3 Flag-bit encryption

This work considers a 512-bit cache line consisting of eight 64-bit words. Thus, size of the meta-data (i.e., zero-flags and energy-flags) is 16 bits (2 bits/word). For ensuring security of the data, the meta-data is also encrypted and stored along with the cache line in the memory. Since 16-bit encryption is susceptible to attacks, the size of the meta-data is extended to 66 bits by including 5 error-correcting pointers (ECP) [39], each of 10 bits (for a 512-bit cache line). An ECP uses a pointer to point to a failed cell and also stores the correct value for that cell. For example, on a row with 256 MLCs (i.e., 512-bit cache line), 8 logical bits are required to uniquely point to any failed cell ($\log_2 256$) and 2 bits logical bits are required to store the data of the failed cell (1

MLC = 2 logical bits), i.e., a total of 10 logical bits. The standard ECP implementation reserves six ECPs for every 512-bit (single-level cell) memory block, enabling each 512-SLC block to recover from up to 6 hard errors. Since MLCs are more susceptible to errors, SECRET reserves 5 ECPs per 256 MLCs. Note that the use of ECPs for endurance enhancement of NVMs is common practice [39,63]. Hence, SECRET ensures the security of the meta-data for no overhead beyond the flag-bits, by incorporating already provisioned ECP bits along with the meta-data to form a secure word. The meta-data encryption is achieved by XORing the meta-data with an OTP generated from the line address and the concatenation of global and all the local counters.

3.2.4 SECRET: Architectural design

SECRET requires modifications to the write and read data-paths to realize smart encryption and energy masking. This section describes each in detail, and finally evaluate the hardware overhead of a practical implementation.

3.2.4.1 Write operation

Figure 10 depicts the modified write data-path circuitry for SECRET. Writes result in (i) smart encryption followed by (ii) energy masking. Smart encryption performs a read-modify-write scheme at the word level, identifying the unmodified words, which are subsequently not written back to the NVM module (main memory). The modified words, referred to as the plaintext in figure 10, are then examined to identify the zero-words, which are also not written back to the NVM module. This is achieved by the zero detection module, which eliminates zero-word writes. The non-zero-words are then encrypted by XORing with an OTP. Parallel AES units generate all the possible combinations of one-time pads (OTPs) corresponding to all possible states of the local counter ($2^2 = 4$ in this example), to avoid the latency of serial generation of OTPs. The line address (LA) and the sum of the global counter (GC) and individual local counters (LC) are provided as inputs to the AES units, which generate the corresponding OTPs. The appropriate OTP for a word is selected from the combinations based on the updated local counter bits of the modified word. The ciphertext is then passed through the energy masking module (EMM). The EMM transforms the ciphertext to a low energy ciphertext that is written back to the NVM module.

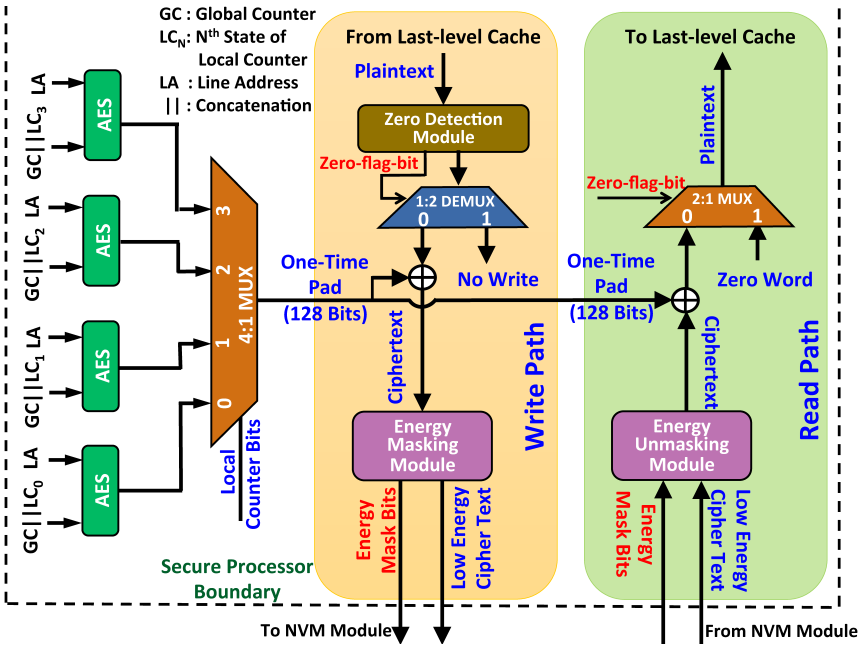


Figure 10: **This figure depicts the architecture for write (yellow) and read (green) data-paths of SECRET. In the write path, the zero detection module and implements the zero-based partial write, whereas the energy masking module implements energy masking. In the read path, the energy unmasking module generates the true ciphertext from the low energy ciphertext, and the zero-flag-bits determine the zero-words during decryption.**

3.2.4.2 Read operation

The modified read data-path to implement SECRET is also illustrated in figure 10. During a read, SECRET reverses the transformations that it applies to the data on a write operation. After the data is fetched from the NVM module, the meta-data (zero-flags, energy-flags, ECP) is first decrypted (not shown in the figure). The data, which is in the low energy ciphertext form, is transformed to the true ciphertext by passing it through the energy unmasking module (EUM). The EUM generates the true ciphertext by XORing the low energy ciphertext with the appropriate energy mask (no, half, full), which is selected on the basis of the energy-flag for that word. The true ciphertext is then decrypted by XORing it with the OTP corresponding to its local counter. However, if the zero-flag-bit of a word is set, a zero-word is forwarded as the plaintext to the cache.

3.2.5 Hardware overhead

SECRET has memory overhead associated with storage of the (i) local counters, (ii) zero-flag-bits, and (iii) energy-flags. For a 512-bit cache line with 64-bit words, each word has a 2-bit local counter and 2 bits of combined zero-flag and energy-flag, i.e., an overhead of 6.25%. SECRET requires 4 additional AES units over conventional AES-based counter mode encryption. A highly optimized and energy-efficient hardware implementation of AES for the 22nm node is reported in [65]. The AES unit overhead is 8360 gates with an area of $\approx 0.02\text{mm}^2$. The standard Intel i7 (22nm) die size is 160mm^2 [66], therefore, the AES unit area overhead (0.0125%) of SECRET is negligible in practice.

3.3 SECRET: EVALUATION AND RESULTS

Trace-based simulations on an MLC/TLC RRAM architecture was performed using NVMain [37] on both integer and floating-point workloads from the SPEC CPU2006 [38] benchmark suite. NVMain is a cycle accurate main memory simulator designed to simulate emerging non-volatile memories at the architectural level. NVMain was configured with 8GB main memory, organized as a single channel, with 1 rank, and eight x8 devices/rank. The memory controller performs first-ready first-come-first-serve scheduling, with open page policy. NVMain was modeled with energy/latency parameters provided in [5, 63]. Further, memory lifetime evaluation was performed using an in-house simulator that operates at the page level with a page size of 4KB. Along the lines of [39, 63], simulations assumed perfect wear leveling and a mean cell lifetime of 10^8 writes until failure.

3.3.1 Evaluated techniques

BLE [11] and DEUCE [12] are state-of-the-art counter mode encryption techniques that perform word-level DCW to reduce the number of re-encrypted words in a cache line. BLE splits a cache line into four 128-bit words and assigns a 2-bit local counter to each word. When a local counter reaches its terminal count, the entire cache line is re-encrypted. In contrast, DEUCE splits a cache

line into 32 16-bit words and maintains a leading and a trailing counter for the entire cache line. After a pre-determined number of writes (epoch), the entire cache line is re-encrypted. While the leading counter is incremented on every write, the trailing counter is fixed and incremented to the leading counter's value after every 32 writes (epoch interval). Due to its design, DEUCE requires that the words that are modified once in a given epoch interval be re-encrypted on every write during that epoch. In contrast, SECRET only encrypts non-zero modifies words, and incorporates XOR-based energy masking to reduce memory writes of encrypted data. Results compare write energy, latency, and memory lifetime of SECRET with BLE and DEUCE. AES-based counter mode encryption, which re-encrypts the entire cache line on every write, is the baseline.

3.3.2 Summary

Table 7 summarizes the energy, latency, and lifetime results for BLE, DEUCE, and SECRET, normalized to the baseline. Whereas BLE and DEUCE rely exclusively on read-modify-write to reduce memory writes, SECRET uses zero-based partial writes along with word-level DCW to reduce memory writes, and energy masks to lower the energy overhead of write operations.

Table 2: **Summary of energy, latency, lifetime improvements, and memory overhead of BLE, DEUCE, and SECRET over AES-based counter-mode encryption.**

NVM	Encryption technique	Energy reduction	Latency reduction	Lifetime improvement	Memory overhead
MLC RRAM	BLE	40%	23%	35%	1.56%
	DEUCE	40%	17%	36%	6.25%
	SECRET	80%	37%	63%	6.25%
TLC RRAM	BLE	33%	31%	18%	1.56%
	DEUCE	40%	23%	24%	6.25%
	SECRET	63%	49%	56%	7.84%

3.3.3 MLC RRAM NVM

First, the results for energy, latency, and lifetime, for an MLC RRAM architecture are presented. Note that an MLC stores 2 logical bits per cell; a 64-bit word is stored in 32 physical MLCs.

3.3.3.1 Energy and latency

As shown in the figure 11, BLE, DEUCE, and SECRET reduce write energy by 40%, 40%, and 80%, respectively, over the baseline. While energy reductions for BLE (DEUCE) result from 128-bit (16-bit) word-level DCW only, energy reductions for SECRET result from 3 levels of optimization – 64-bit word-level DCW, zero-based partial writes, and energy masks. Latency reductions for BLE, DEUCE, and SECRET over the baseline are 23%, 17%, and 37%, respectively. It is interesting to note that for benchmarks such as perlbench and bzip2, the latency reductions for SECRET are lower than DEUCE. This occurs when a particular word in a cache line changes on every write, resulting in a rollover of its local counter. A counter rollover precipitates the re-encryption of the entire cache line, increasing memory latency.

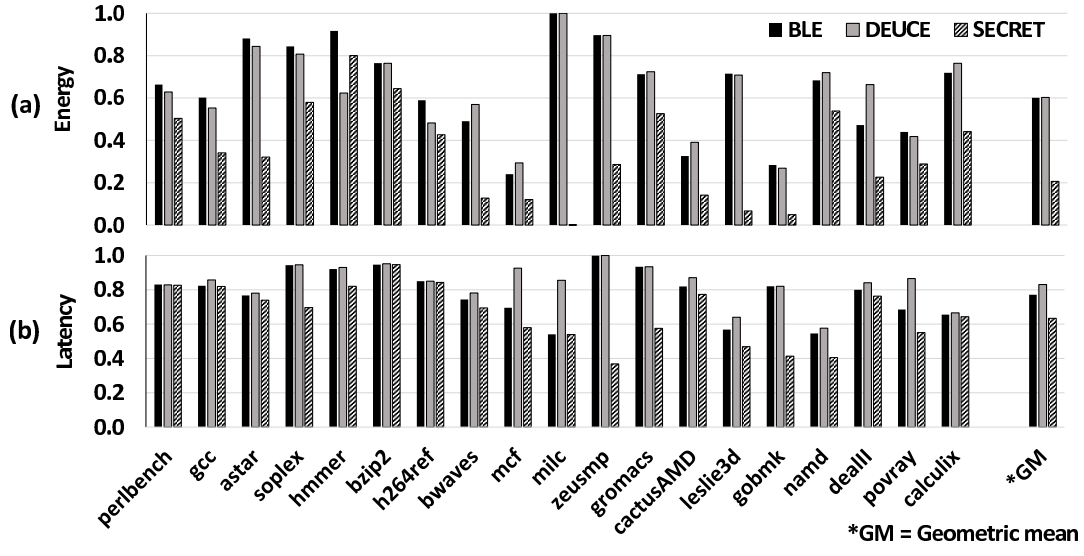


Figure 11: **(a) Write energy and (b) latency of BLE, DEUCE, and SECRET (normalized to baseline) evaluated using SPEC CPU2006 benchmarks [38] on NVMain [37]. For MLC RRAM, BLE, DEUCE, and SECRET reduce write energy (latency) by 40% (23%), 40% (17%), and 80% (37%), respectively, over baseline AES-based counter mode encryption.**

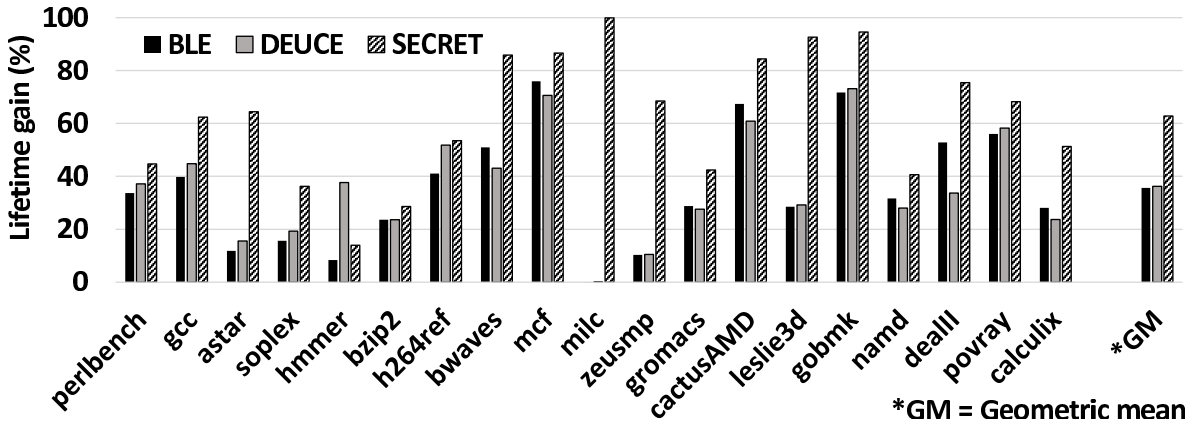


Figure 12: **Memory lifetime (normalized to AES-based counter mode encrytion) for MLC RRAM using BLE, DEUCE, and SECRET encryption architectures. Lifetime evaluated using an in-house simulator, using average cell lifetime of 10^8 writes until failure [63].**

3.3.3.2 Memory lifetime

Figure 12 reports the improvements in memory lifetime of MLC RRAM NVM using BLE, DEUCE, and SECRET, normalized to the baseline. As seen from the figure, there is no improvement in lifetime from BLE and DEUCE for the milc benchmark because the milc benchmark consists of a large number of zeros in the plaintext. Since SECRET eliminates zero encryption in the plaintext, only SECRET increases lifetime for the milc benchmark. In contrast, BLE and DEUCE re-encrypt zero words (that are significant in the milc benchmark), resulting in high cell flip rate and consequently low NVM lifetime. On average, BLE, DEUCE, and SECRET improve memory lifetime by 35%, 36%, and 63%, respectively. SECRET improves memory lifetime by reducing the number of programmed cells using both word-level DCW and zero-based partial encryption, thereby achieving the maximum reduction in the wear rate of memory.

3.3.4 TLC RRAM NVM

Energy, latency, and lifetime of SECRET for TLC RRAM NVM are also evaluated. A 512-bit cache line is partitioned into nine 51-bit words and one 53-bit word for smart encryption and energy masking. A 51-bit (53-bit) word is stored in 17 (18) TLCs (3 logical bits per physical TLC). Although a 51/53-bit word is used for implementing SECRET, SECRET does not modify

the instruction set architecture (ISA) of the underlying system. Similar to MLC NVM, SECRET uses two energy masks (a full mask and a half mask) with repeated state 3 (011), since it is one of the highest energy/latency consuming states in a TLC (refer Table 1). SECRET does not employ more than 2 energy masks to keep the memory overhead below (8%).

3.3.4.1 Energy and latency

Figure 13 compares the write energy and latency of TLC RRAM NVM using BLE, DEUCE, and SECRET. SECRET reduces energy (latency) by 63% (49%) over the baseline. BLE and DEUCE reduce write energy (latency) by 33% (31%) and 40% (23%), respectively, over the baseline.

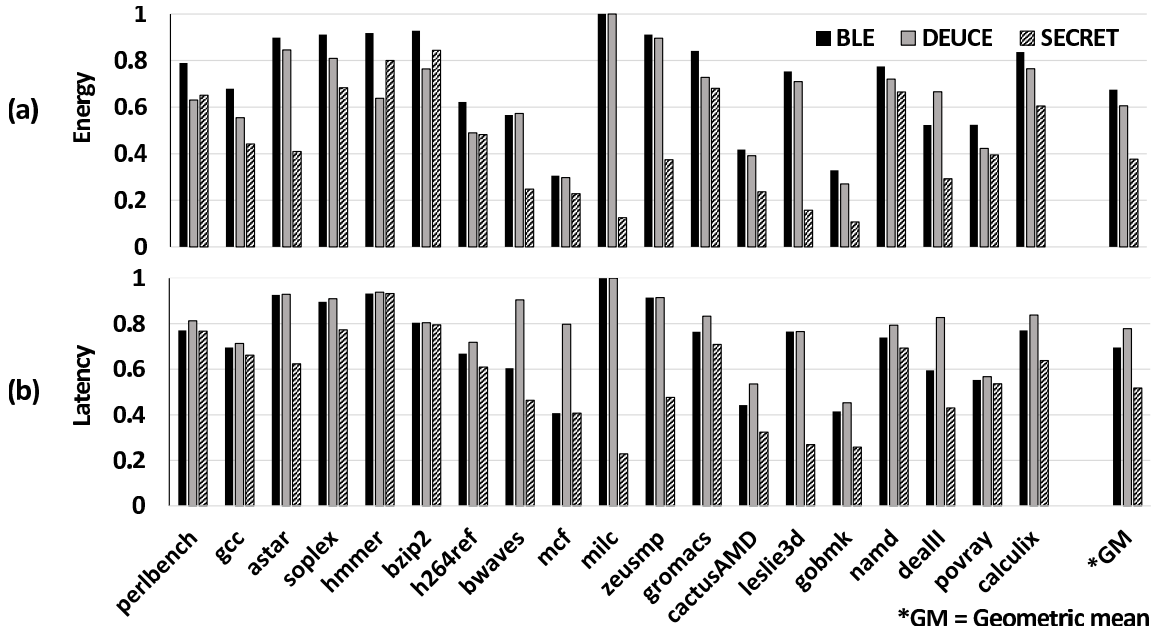


Figure 13: **(a) Write energy and (b) latency of BLE, DEUCE, and SECRET (normalized to AES-based counter mode encryption) evaluated on SPEC CPU2006 benchmarks [38] using NVMain [37]. For TLC RRAM, BLE, DEUCE, and SECRET reduce energy (latency) by 33% (31%), 40% (23%), and 63% (49%), respectively.**

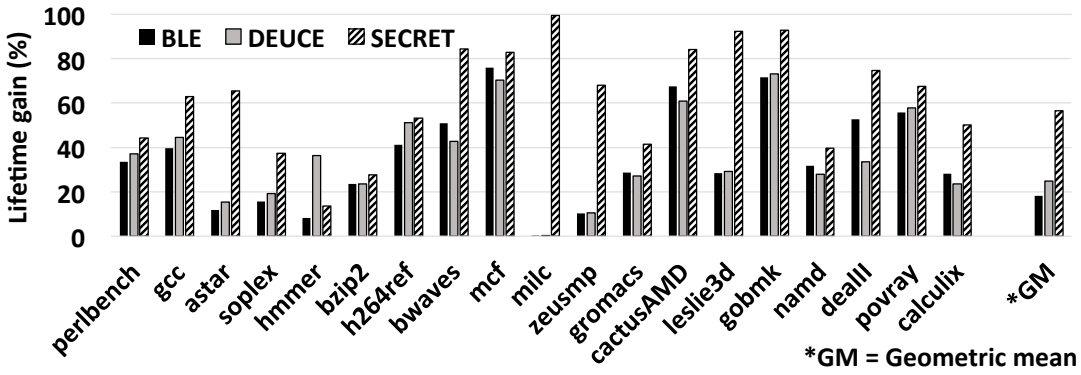


Figure 14: **Memory lifetime (normalized to AES-based counter mode encryption) for TLC RRAM using BLE, DEUCE, and SECRET. Lifetime evaluated using an in-house simulator, assuming average cell lifetime of 10^8 writes until failure [63].**

3.3.4.2 Memory lifetime

Figure 14 reports memory lifetime of TLC RRAM using BLE, DEUCE, and SECRET, normalized to the baseline. As seen from the figure, lifetime improvements are maximum for SECRET (56%), followed by DEUCE (24%) and BLE (18%).

3.4 SECRET: CONCLUSIONS

Whereas encryption ensures data confidentiality in NVMs, it increases NVM write energy/latency and degrades NVM lifetime. This problem is exacerbated in MLC/TLC NVMs, due to the prevalence of high energy (latency) states. SECRET is the first work to target encryption penalty reduction for secure MLC/TLC NVMs. SECRET integrates smart encryption with XOR-based energy masking to realize secure MLC/TLC NVMs with low write energy/latency and improved lifetime. Smart encryption prevents re-encryption of unmodified or zero-words during a write-back, thereby preventing unnecessary cell flips and reducing write energy. For modified non-zero-words, energy masking further optimizes the write operation by applying XOR-based masks to transform a high energy ciphertext into a low energy ciphertext. SECRET outperforms state-of-the-art NVM encryption solutions, with the lowest write energy and latency, as well as the highest lifetime.

4.0 COVERT: COUNTER OVERFLOW REDUCTION FOR EFFICIENT ENCRYPTION OF NON-VOLATILE MEMORIES

4.1 COVERT: MOTIVATION

To reduce on-chip memory overhead in CME, the counters are stored in main memory and cached in an on-chip counter cache to improve performance [16,21,22,53]. Previous work [22] has shown that the counter values can be stored in the plaintext because the attacker still needs the secret key to regenerate the pad. On a read, if the counter for the requested cache line is available in the counter cache, CME overlaps the OTP generation with the off-chip data fetch, thereby hiding the decryption latency of the cache line. However, if the counter is not available in the counter cache, the OTP generation is delayed until the counter is fetched to the processor-side memory controller, increasing the decryption latency. These scenarios are depicted in figure 15. Similarly, during a write, a counter cache hit reduces the encryption latency by eliminating the off-chip counter fetch.

To increase the counter cache hit rate, small counters are preferred in practice, since a fixed size counter cache can store more counters if each counter is small. However, small counters can quickly overflow in the presence of high memory write traffic. Conventionally, a counter overflow is handled by changing the secret key to prevent reuse of OTPs [16,21,22,53]. However, since the same secret key is shared by every cache line, a change of secret key requires the entire memory to be re-encrypted, causing the system to freeze for the duration of full memory re-encryption. Re-encryption involves reading, decrypting, encrypting, and writing back every cache line in the memory. For the latest PCM prototype (read = 75ns, write = 150ns) [41], re-encryption of 16GB memory (16GB = 256 million cache lines of 64 bytes each) takes $\approx$ 1 minute (256 million cache lines $\times$ 225ns re-encryption per cache line). To reduce counter overflow and increase system availability, large counters are used for encryption, since they do not overflow frequently. However,

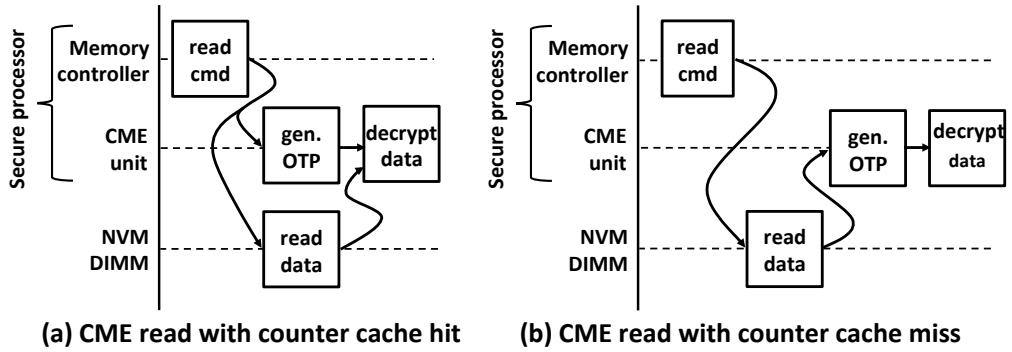


Figure 15: **Illustration of CME read on a (a) counter cache hit and (b) counter cache miss. On a read, a counter cache hit overlaps the OTP generation with the data (ciphertext) fetch from the memory, whereas a counter cache miss precipitates a read request to fetch the counter and the ciphertext, which delays the OTP generation by 1 read cycle.**

large counters increase the memory overhead of CME and result in poor system performance due to frequent counter cache misses. CME thus imposes heavy overheads on memory, system performance, and system availability in practice

The goal of this work is to develop low-overhead secure NVMs, without compromising system performance/availability. Although PCM is considered as the representative NVM in this work, the solutions developed here are applicable to other NVM technologies such as RRAM, spin-transfer torque RAM (STT-RAM), and 3D X-Point.

4.2 COVERT: CONTRIBUTIONS

This section describes COVERT, a CME-based memory encryption solution that performs on-demand memory allocation to reduce the memory encryption frequency of fast growing counters, while also retaining the area and performance benefits of small-sized counters. The next section discusses the contributions of COVERT in detail.

4.2.1 COVERT: Dynamic counter (DYNAMO)

DYNAMO is a technique to reduce the frequency of full memory re-encryption resulting from counter overflow in conventional CME. To improve system availability without incurring area and erformance penalty of large counters, DYNAMO performs on-demand memory allocation to the overflowing counters. By providing extra memory to an overflowing counter, DYNAMO allows continued operation based on the overflowed counter, thereby avoiding counter reset, and hence delaying full memory re-encryption.

Observation: Emerging NVMs suffer from low write endurance, resulting in early cell failures (PCM cells typically fail after 10^{8-10} writes [2, 3, 5, 41, 63]) that result in hard errors. To improve NVM reliability, architects employ error-correcting pointers (ECP) [39, 40], which uses a pointer to point to a failed cell and also stores the correct value for that cell. The standard ECP implementation reserves six ECPs in memory for every 512-bit memory block (i.e., per cache line), enabling each cache line to recover from up to 6 hard errors (ECP-6 henceforth). In terms of the pages maintained by the operating system, this translates to 384 ECPs per 4kB page (6 ECPs per 512-bit cache line $\times$ 64 cache lines). However, uniform ECP allocation across the memory leads to under-utilization of ECPs, since different cache lines in a page have different error correction requirements. For example, in [40], it is reported that very few cache lines ($<$1%) in a page use all the 6 available ECPs at the time of page failure[1]. **Thus, a significant portion of the memory provisioned for ECPs remains unutilized on a failed page and can be repurposed till such time they are necessary for error correction.** Without loss of generality, COVERT considers ECP-6 as the underlying error correction technique; under-utilization of error correction resources is observed in other error correction techniques also.

4.2.1.1 DYNAMO design

DYNAMO leverages unused ECPs available in memory to extend the overflowing counters. When DYNAMO detects a counter overflow on a write to memory, it determines if unused ECPs are available for the memory block where the cache line is to be written. Each 512-bit cache line maps

[1]Techniques like PAYG [40] and Zombie memory [67] improve ECP utilization at the cost of system performance, and still result in more than 80% of cache lines possessing unused ECPs on a failed page. Since encryption is itself latency intensive, it is not expected that latency-intensive error correction techniques like PAYG/Zombie memory will be integrated with NVM encryption in immediate future.

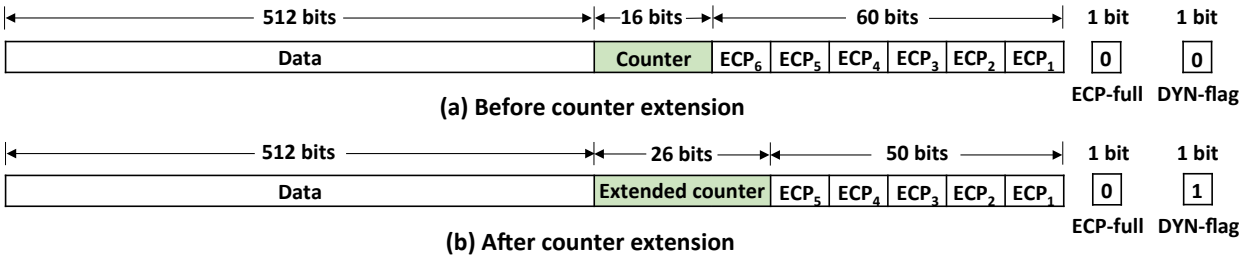


Figure 16: **Illustration of counter extension using DYNAMO. An overflowing counter is allocated an unused ECP and a DYN-flag is set to indicate the counter extension. This reduces the total available ECPs for that memory address to 5.**

to 512 single-level cells (SLCs) in the memory. To detect unused ECPs, DYNAMO leverages the ECP-full flag, which is already provisioned in the standard ECP-6 design to indicate if a memory block has consumed all 6 ECPs. A reset/set ECP-full flag indicates available/exhausted ECPs. If unused ECPs are available, DYNAMO allocates one unused ECP to the overflowing counter and sets a DYNAMO flag (DYN-flag) to indicate counter extension for the cache line. This is shown in figure 16, where the last ECP, i.e., ECP_6 is allocated to the overflowing counter and the DYN-flag is set to 1. DYN-flag indicates number of ECPs available per 512-bit memory block; DYN-flag = 1 implies 5 ECPs, DYN-flag = 0 implies 6 ECPs. Assigning one unused ECP increases the size of an overflowing counter by 10 bits (COVERT uses only 8 bits and reserve 2 bits for future use). Hence, the effective counter size after extension is 24 bits and it delays counter overflow by 2^{24} - 2^{16} writes.

Leveraging wear leveling for DYNAMO: It is common to perform wear leveling to achieve a uniform distribution of writes across NVMs, which prevents the more frequently written memory cells from failing early in comparison to other cells. Thus, wear leveling becomes necessary for NVMs that suffer from low endurance. Start-Gap [43] is a state-of-the-art wear leveling technique that incurs insignificant memory overhead (one spare cache line termed the GapLine) to achieve $\approx$ 97% wear leveling. This work assumes that Start-Gap wear leveling is implemented in memory.

The frequently written cache lines, i.e., the cache lines with fast-growing counters experience more cell failures and are quite likely to exhaust their available ECPs, rendering DYNAMO ineffective. However, in the presence of Start-Gap wear leveling, a cache line gets mapped to multiple

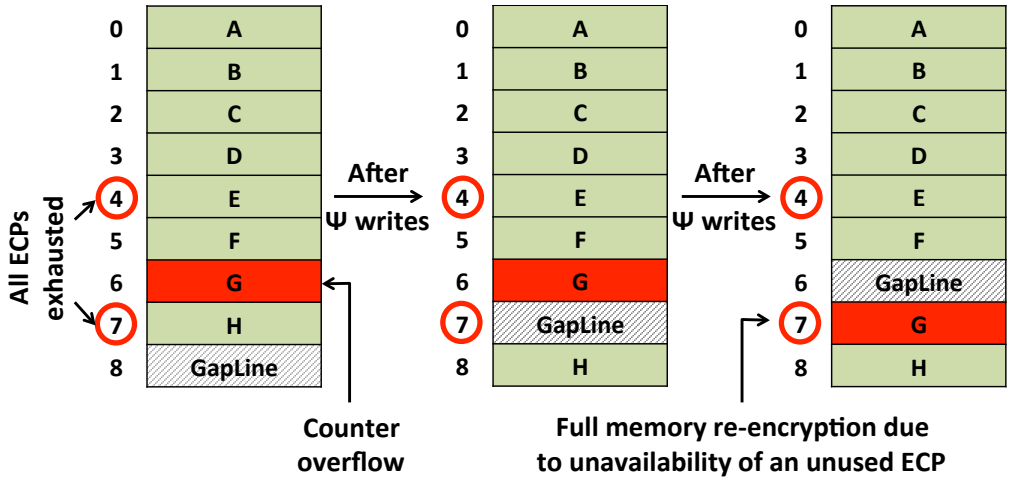


Figure 17: **A snapshot of memory in the presence of Start-Gap wear leveling [43] and DYNAMO. Start-Gap wear leveling maps a cache line to different physical addresses to achieve uniform wear across memory. Note that full memory re-encryption is required only when a cache line with an overflowing counter gets mapped to a physical address with no unused ECP. Following [43], $\psi = 100$ is assumed.**

memory locations, which increases the probability of finding unused ECPs for frequently written cache lines. The full memory re-encryption occurs only when a cache line with an overflowing counter gets mapped to a memory address with no unused ECPs. As shown in figure 17, memory addresses 4 and 7 do not possess unused ECPs and cache line 'G' has an overflowing counter. When 'G' gets mapped to address 7, due to the unavailability of an unused ECP, full memory re-encryption is required. Hence, the probability of full memory re-encryption due to unavailability of ECPs is pq, where p is the probability of a memory address exhausting all 6 ECPs and q is the probability of a counter overflow. The value of p is $\approx 1\%$ [40] and q is $\approx 30\%$ (based on the simulations on SPEC CPU2006 benchmarks [38]). This implies that the probability of finding unused ECPs for counter extension is greater than 99% ($1 - pq$) in practice.

4.2.2 Memory operations

Covert integrates DYNAMO inside the NVM DIMM to realize high-performance, low-overhead CME. On a memory write, if a counter overflows, DYNAMO is invoked to perform counter extension and the DYN-flag for that memory address is set to 1. However, if counter extension is

not feasible, full memory re-encryption is performed using a new secret key and all the counters are reset to 0. Full memory re-encryption involves reading, decrypting, encrypting, and writing back every cache line in the memory, causing the system to freeze for the duration of full memory re-encryption.

4.3 COVERT: EVALUATION AND RESULTS

COVERT is evaluated on a phase change random access memory (PCRAM) architecture using the MARSS full-system simulator [42] on both integer and floating point workloads from the SPEC CPU2006 benchmark suite [38]. COVERT employs 16-bit CME (i.e., CME with 16-bit counter per line) as the underlying encryption technique in COVERT.

4.3.1 Simulation framework

MARSS is configured to simulate a standard 4-core out-of-order system running at 3GHz. Each core is assigned a private L1 instruction/data cache of 32kB (latency = 2ns) and a private L2 cache of 128kB (latency = 5ns). Finally, L3 is a single, shared write-back cache of 8MB (latency = 20ns). The main memory is modeled as a 16GB, 8 banks, single channel PCRAM with read latency = 75ns and write latency = 150ns [12, 41]. Furthermore, COVERT implements CME using a fully pipelined 128-bit advanced encryption standard (AES) crypto-engine (OTP generation latency = 72ns [53]). A 512kB 32-way set-associative counter cache [53] is integrated inside the memory controller for all the techniques evaluated in this work. COVERT statically splits the counter cache for COVERT into two parts: (i) 384kB 16-bit counter cache for storing a majority of non-overflowing 16-bit counters and (ii) 128kB 24-bit counter cache for storing a small number of overflowing counters that are extended from 16 bits to 24 bits using DYNAMO. Although it is possible to realize a split counter cache architecture that dynamically varies the counter cache block size in response to dynamic counter extension [68, 69], the implementation details and results for that architecture will be covered in future work.

Table 3: **Summary of IPC, re-encryption rate, and memory overhead of COVERT and other CME techniques. COVERT provides the optimal trade-off between performance (IPC), system availability, and memory overhead.**

Technique	Counter size	IPC	Re-encryption rate	Memory overhead
8-bit CME	8 bits	0.84	1.39 (sec)	1.56%
64-bit CME	64 bits	0.77	millennia	12.5%
MECU	16 bits	0.82	6.0 (min)	3.12%
DEUCE	32 bits	0.79	274 (days)	6.25%
COVERT	**16 bits**	**0.82**	**25.7 (hr)**	**3.32%**

4.3.2 Summary of results

Table 3 summarizes the system performance (measured in instructions per cycle (IPC)) and re-encryption frequency of COVERT and other CME implementations. IPC is normalized to ideal CME, i.e., CME with zero latency overhead for OTP generation. COVERT is compared with state-of-the-art MECU [8] and DEUCE [12]. MECU (DEUCE) is designed with a 16-bit (32-bit) counter per 512-bit cache line, incurring a memory overhead of 3.12% (6.25%). To establish bounds on encryption rate and system performance of CME, 8-bit and 64-bit CME are also evaluated. The results show that COVERT requires a full system re-encryption only once in 25.7 hours (i.e., 250$\times$ improvement over 16-bit CME like MECU) with no penalty on system performance (i.e., COVERT and MECU have the same IPC).

4.3.3 Re-encryption rate

In this section evaluates the re-encryption rate of COVERT for applications from the SPEC CPU20-06 benchmark suite [38]. Following [16], the growth rate of the fastest growing counter for each application was tracked and these growth rates were used to estimate the re-encryption frequency in long-running applications. Furthermore, a constant write-back rate of 40MB/s was assumed. Note that although the data transfer rate of DDR3 is 6400MB/s, the PCRAM prototype available has a maximum write throughput of only 40MB/s [41].

Table 4 reports the re-encryption rates for different CME techniques. Whereas the re-encryption rate reduces with an increase in the counter size (1.39 seconds for 8-bit CME versus 1000 millennia for 64-bit CME), it comes at the cost of high memory overhead (1.5% for 8-bit CME versus 12.5% for 64-bit CME) and poor system performance (IPC of 64-bit CME is 10% lower than 8-bit CME). By employing DYNAMO for counter extension, COVERT reduces the re-encryption frequency of a 16-bit CME (e.g., MECU) from every 6 minutes to every 25 hours (i.e., 250× improvement), for an additional memory overhead of only 0.19% (1 bit DYN-flag per 512-bit cache line). Note that scheduling a minute-long full memory re-encryption once in 25 hours (≈ 99.93% system availability) is feasible for a majority of systems in practice. However, for systems that are expected to provide higher availability (e.g., web/database servers), COVERT can be implemented with 24-bit CME to achieve 99.999% system availability (using DYNAMO) with minimal impact to system performance and only 4.8% total memory overhead. In contrast, DEUCE [12] achieves equivalent system availability for 20% increase in counter memory (800MB for COVERT versus 1GB for DEUCE) and 1-2% reduction in system performance over COVERT.

4.3.4 Lifetime improvements

COVERT improves memory lifetime by reducing the memory wear attributed to full memory re-encryption. To compute the reduction in memory wear, computed the total number of writes (due to write-back and full memory re-encryption) experienced by the memory in one re-encryption interval. When the re-encryption interval is large, writes due to full memory re-encryption are reduced, which in turn improves endurance. In these computations, a constant write-back rate of 40MB/s was assumed and total writes on re-encryption as 256 million (since 16GB memory = 256 million cache lines). Results show that COVERT improves memory lifetime by 1.9× over 16-bit CME (e.g., MECU).

4.4 COVERT: CONCLUSIONS

Memory encryption is required to address security vulnerabilities in NVM-based main memories. Counter mode encryption (CME), a state-of-the-art main memory encryption technique, imposes

overheads on memory, system performance, and system availability. COVERT is the first paper to target these problems of CME for NVMs. COVERT employs DYNAMO, which utilizes unused error correction resources to reduce memory re-encryption frequency for insignificant memory overhead and no impact on system performance of the underlying CME architecture.

Table 4: **Full memory re-encryption frequency for different CME techniques. The system remains unavailable for the duration of re-encryption (≈1 minute for 16GB memory). Large counters reduce the full memory re-encryption rate at the cost of increased memory overhead and poor system performance.**

Benchmark	8-bit CME (seconds)	64-bit CME (millennia)	MECU (minutes)	DEUCE (days)	COVERT (hours)
astar	4.79	> 1000	20.4	931	87.4
bzip2	1.76	> 1000	7.52	342	32.1
gcc	3.33	> 1000	14.2	646	60.6
namd	0.29	> 1000	1.27	57.8	5.42
omnetpp	0.38	> 1000	1.65	75.3	7.06
xalancbmk	0.12	> 1000	0.54	24.8	2.32
soplex	4.55	> 1000	19.4	886	83.1
perlbench	0.33	> 1000	1.42	65	6.08
GemsFDTD	35.3	> 1000	150	6858	643
Geometric mean	**1.39**	**> 1000**	**6.0**	**274**	**25.7**

5.0 ACME: ADVANCED COUNTER MODE ENCRYPTION FOR SECURE NON-VOLATILE MEMORIES

5.1 ACME: MOTIVATION

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

These unique OTPs are generated from unique seeds that have two components: (i) the cache line address (to ensure spatial exclusivity) and (ii) a cache line counter, which is incremented on each write (to ensure temporal exclusivity). Note that the cache line address used for OTP generation can be a virtual address (VA, i.e., the address generated by the CPU) or a physical address (PA, i.e., the address generated after virtual to physical address translation by the memory management unit, i.e., MMU). In [16, 17, 21] it was argued that VAs are more difficult to support because they are not directly available at the lowest-level on-chip cache, and different processes use the same VAs, which can lead to the reuse of OTPs across different processes. In contrast, PAs are easier to use, but require book-keeping for reliable decryption of the pages that are moved between main memory and the disk. Further, advanced cryptographic primitives (for DRAM-based main memories), e.g., oblivious RAM [30, 31] also use PAs for encryption. Along these lines, this work assumes that the PA is used for OTP generation.

To reduce on-chip memory overhead in CME, the counters are stored in main memory and cached on-chip in a counter cache to improve performance [22, 23, 26]. To increase the counter

cache hit rate, small counters are preferred in practice, since a fixed size counter cache can store more counters if each counter is small. However, small counters can quickly overflow in the presence of high memory write traffic. Conventionally, counter overflow is handled by changing the secret key to prevent reuse of OTPs [16, 21]. However, since the same secret key is shared by every cache line, a change of secret key requires the entire memory to be re-encrypted, causing the system to freeze for the duration of full memory re-encryption. This limitation of CME can be exploited in a DoMS attack to render a memory system non-operational by forcing frequent full memory re-encryption. A DoMS attack can be easily engineered using cache eviction and ordering instructions (like `clflush` and `mfence`) that can be executed without any administrative privileges to constantly write to the same cache line in main memory, forcing its counter to overflow. Past work [70] on NVM security has shown that counter overflow can freeze a 16GB PCM memory system for $\approx$ 1 minute, which is unacceptable in practice.

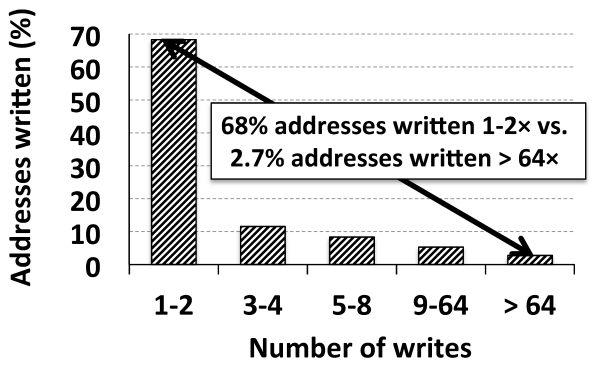
To reduce counter overflow and increase system availability, large counters are used in CME, since they do not overflow frequently. However, large counters increase the memory overhead of CME and result in poor system performance due to frequent counter cache misses. Thus, CME imposes heavy overheads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

5.2 ACME: CONTRIBUTIONS

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

5.2.1 ACME: Observation

Memory access locality in real-world applications causes some of the memory addresses to be written much more frequently than others. Figure 18 reports the write-back, i.e., main memory



Benchmark	Max writes	Median writes
gcc	194	2
astar	236	2
hmmer	507	24
cactusADM	94	8
h264ref	370	1
dealII	760	2
tonto	826	4
namd	2009	1

Figure 18: **Write-back (i.e., main memory write) distribution for SPEC CPU2006 benchmarks [38]. 68% of the total addresses written receive $<$ 2 writes and only 2.7% of the total addresses written receive $>$ 64 writes. Due to this disparity in write-back distribution between addresses, a majority of counters in CME remain underutilized and never reach their terminal count in practice.**

write distribution of the SPEC CPU2006 benchmarks [38]. The table reports the maximum writes seen by a memory address (in main memory) for various benchmarks; the median writes for different addresses are also reported. It is seen that on average, 68% addresses are written 1-2×, whereas only 2.7% addresses are written $>$ 64× during multi-billion-instruction simulations of these benchmarks. *Due to this disparity in write-back distribution between addresses, counters belonging to only a few frequently written cache lines reach their terminal count forcing full memory re-encryption, whereas a majority of counters remain underutilized and never reach their terminal count in practice.*

5.2.2 ACME: Design

In order to delay counter overflow (i.e., system freeze) due to the disparity in write distribution between addresses and improve counter utilization, ACME proposes counter write leveling (CWL) by re-assigning underutilized counters to frequently written cache lines. This results in a uniform growth across counters, thereby delaying full memory re-encryption due to counter overflow and improving system availability. To perform CWL, ACME leverages the inbuilt wear leveling logic present on the secure processor.

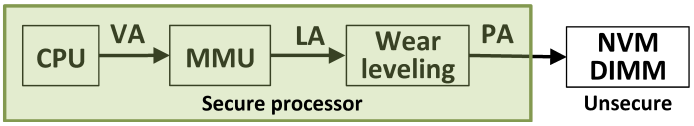


Figure 19: **Address translation due to wear leveling in NVMs.**

Since NVMs suffer from low endurance, it is common to perform memory wear leveling to uniformly distribute writes across memory, which prevents the more frequently written memory cells from failing early in comparison to other cells. Region-based Start-Gap (RBSG) [43] is a state-of-the-art wear leveling technique that divides the memory into multiple fixed size regions (Start-Gap regions), and performs wear leveling for each region independently. For insignificant memory overhead (one spare cache line termed the GapLine per 2^{18} cache lines in a region), RBSG achieves $\approx$ 97% wear leveling, i.e., 97% of total writes to main memory are uniformly spread across all memory addresses. This work assumes that RBSG is implemented in memory.

Wear leveling introduces an additional level of indirection in the address translation process. A VA generated by the CPU is first translated to an intermediate or logical address (LA) by the MMU. In DRAM-based main memory, the LA is the final PA where the cache line is stored in the memory. However, due to wear leveling in NVMs, the LA of a cache line is further translated by the wear leveling unit to a new PA where the cache line is actually stored in the memory. Figure 19 illustrates address translation in NVMs.

Since wear leveling maps an LA to different PAs over time, employing the PA for OTP generation can lead to the OTP reuse problem. For example, OTP reuse will occur if an LA with a certain counter value (conventionally, a counter is associated with each cache line, i.e., LA) is translated to a new PA that has already seen that counter value with some other LA. Figure 20 illustrates the OTP reuse problem if the PAs are used for OTP generation when counters are associated with the LAs. At time t_1, cache line with LA 'D' is stored at PA 40. This cache line is encrypted using an OTP generated from PA 40 and counter 120. At time t_2, cache line 'D' has been mapped to PA 50 and the GapLine occupies PA 40. Additionally, counter values have been updated to reflect the total writes seen by each cache line. At time t_3, cache line 'C' is mapped to PA 40. Since the current counter value of cache line 'C' is 120, the OTP used in encrypting cache line 'C' is

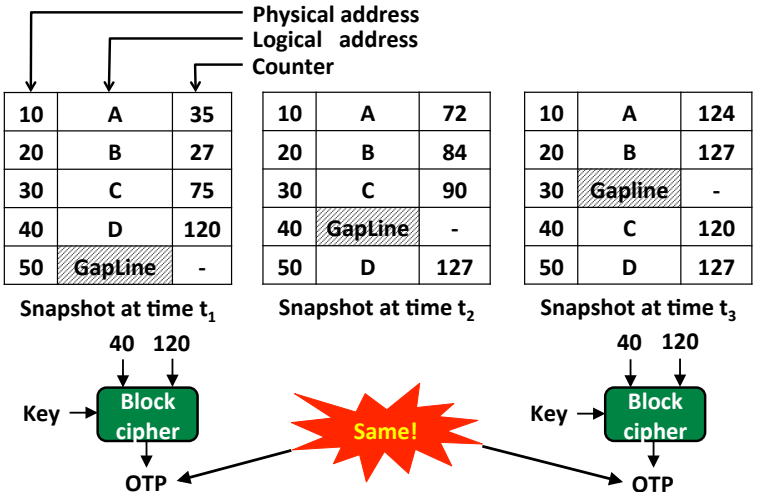


Figure 20: **Illustration of OTP reuse, when the PA is used for OTP generation and the counters are associated with LAs.**

the same OTP that was used for encrypting cache line 'D' at time t_1. This is a security breach that can be exploited to execute known-plaintext attacks (KPA) [16, 21, 22]. To avoid such scenarios, state-of-the-art NVM encryption solutions [12, 23, 70] decouple CME from wear leveling by generating the OTP using the LA instead of the PA.

In contrast, ACME employs the PA for OTP generation and overcomes the OTP reuse problem by employing physical counters that do not translate with the LA, i.e., ACME assigns a counter per 512-bit physical memory location; for every write to a PA, its counter is incremented by 1. Thus, for a given PA, a counter value is never repeated, thwarting KPAs. Additionally, when a frequently written cache line is translated from one PA to another PA due to wear leveling, its associated counter is also changed, leading to a distribution of writes across counters, i.e., counter write leveling (CWL). Figure 21 contrasts CME and ACME. When cache line 'D' is translated from PA 40 to PA 50 using CME (figure 21(a)), its associated counter, i.e., CTR_D remains unchanged and continues to grow with writes to cache line 'D', resulting in early overflow. In contrast, when cache line 'D' is translated from PA 40 to PA 50 using ACME (figure 21(b)), its associated counter is changed from CTR_{40} to CTR_{50}. This results in sharing of writes to cache line 'D' between CTR_{40} and CTR_{50}. CWL thus increases the time to counter overflow, thereby delaying the time to full memory re-encryption.

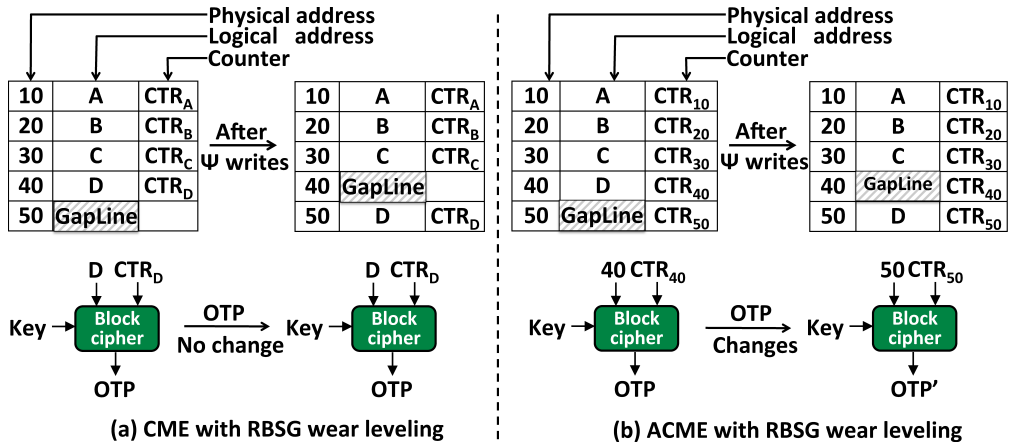


Figure 21: **This figure contrasts OTP generation of CME and ACME, in the presence of Region-Based Start-Gap (RBSG) wear leveling. (a) OTP generation for CME remains unaffected after cache line 'D' is remapped. (b) OTP generation for ACME requires the new PA and its corresponding counter value after cache line 'D' is remapped. In this work, the GapLine moves after every ψ writes ($\psi = 64$).**

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

5.2.3 ACME: Memory organization

Conventionally, CME stores the ciphertext and counters separately in main memory such that counters belonging to consecutive LAs are stored in consecutive physical locations [16, 22, 26]. This memory organization facilitates prefetching of multiple counters that are consecutive in the logical address space. Since real-world applications exhibit high spatial locality, prefetching consecutive LA counters results in a high counter cache hit rate. For a 512-bit counter cache block and n-bit counter, each counter cache miss fetches $\lfloor 512/n \rfloor$ counters to the counter cache.

Furthermore, since the counter cache is implemented as a write-back cache, a counter is updated in the main memory only when its respective counter cache block is evicted from the counter

cache. This delay in updating the counters renders the ciphertext and counters partially unsynchronized in the main memory, and a native CME memory system cannot support data recovery in the face of power/system failures. Since CME was originally proposed for DRAM-based main memories, which are volatile and not expected to recover data in the face of power/system failures, it allowed the ciphertext and counters to be left unsynchronized in the main memory until the ciphertext is swapped out to the disk. However, past work [8, 10] has motivated that preserving data recovery is an important design goal for any NVM security solution. Whereas employing a write-through counter cache preserves the data recovery property of NVMs, it doubles the number of memory writes, since each data write requires an additional write to update the counter. Since NVM writes are 2-3$\times$ slower than NVM reads, a 2$\times$ increase in memory writes significantly reduces system performance.

To ensure consistent counter updates and support data recovery, ACME stores counters alongside their respective ciphertext as a single memory block (as shown in figure 22). Whereas CME requires separate memory accesses to fetch/update the ciphertext and counter, ACME enables the ciphertext and its counter to be simultaneously fetched or updated in a single memory access. Since ACME employs a 24-bit counter (refer section 5.3.1) per 512-bit cache line, and the width of a typical data bus is 64 bits, 9 data beats are required to transfer data and a 24-bit counter between the NVM DIMM and the processor-side memory controller (ignoring the transfer of error correction meta-data [44] for simplicity). For a DDR3-1600 data bus connected to a PCM-based memory, an increase of 1 data beat per transfer increases effective memory access latency by $< 2\%$ [2]. In contrast, a separate memory access (as seen in native CME memory systems) increases the effective memory access latency by 100%. Hence, storing the ciphertext and counters alongside preserves data recovery without impacting performance. Note that the changes required for ACME – 1 extra data beat during data transfer and colocated counter and ciphertext – are minor in comparison to the performance and data recovery benefits afforded by ACME. These requirements can easily be fulfilled using advanced memory systems such as Samsung high-bandwidth memory (HBM), Micron hybrid memory cube (HMC), Intel Optane, etc.

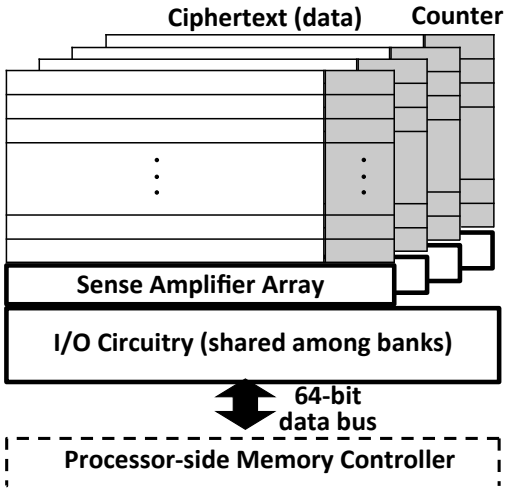


Figure 22: **ACME memory organization: Counter and ciphertext are stored alongside for concurrent access.**

5.2.4 ACME: Memory operations

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

Write: On a write, CME increases the counter value of the LA of a cache line; ACME increases the counter value of the PA. Thus, when a cache line is translated to a new PA due to wear leveling, it involves: (i) reading the cache line to the processor-side memory controller, (ii) re-encrypting the cache line with the new OTP, and (iii) writing the cache line to the new memory location. However, it is important to note that steps (i) and (iii) are also performed in conventional CME due to wear leveling. Furthermore, step (ii) does not add any latency because generation of the old and new OTPs can be overlapped with step (i) by caching the counters (in the already provisioned counter cache) corresponding to the old/new PAs (already known to the wear leveling architecture [43]).

5.2.5 ACME: Security

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

The robustness of ACME to availability attacks (e.g., DoMS) can be studied by analyzing how soon malicious code can render the memory system unavailable to other applications by forcing full memory re-encryption. For example, a simple DoMS attack can repeatedly write to the same LA by employing the `clflush` instruction to sidestep the last-level cache (LLC) and `mfence` instruction to clear the processor's fill-buffer[1]. The following C++ code was used in our experiments to execute the DoMS attack. This code was verified by monitoring the store operations using hardware performance counters for Linux [72]. Note that this code does not require any administrative privileges to execute any of its instructions.

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

Repeatedly writing to the same cache line (i.e., LA) causes the counter for that cache line to overflow in classical CME, forcing full memory re-encryption. However, since ACME distributes the writes to a cache line across multiple counters, it is able to sustain memory availability even when the memory is subject to this DoMS attack. Availability analysis of various CME- and ACME-based systems to the above DoMS attack is provided in section 5.3.3.

[1]The researchers at Google Project Zero [71] documented equivalent row-hammer attacks on DRAM.

5.3 ACME: EVALUATION AND RESULTS

This section evaluates and compares ACME to CME on a single-level cell (SLC) PCM architecture using both integer and floating point workloads from the SPEC CPU2006 benchmark suite [38]. Our evaluations comprise (i) trace-based simulations to evaluate system availability and (ii) system-level simulations using the MARSS full-system simulator [42] to evaluate system performance.

Trace-driven simulation: An in-house trace-based simulator (which models RBSG wear leveling) is used to track the growth rates of the fastest growing counter for each benchmark. The counter growth rates are used to estimate the fastest counter overflow rate, i.e., full system re-encryption frequency in long-running applications. Multi-billion instruction memory traces were generated using the Intel Pin [45] binary instrumentation tool.

Full-system simulation: MARSS [42] is configured to simulate a standard 4-core out-of-order system running at 3GHz. Further, each core is assigned a private L1 instruction and data cache of 32kB each and a private L2 cache of 128kB; the L3 cache (LLC in this case) is a single, shared write-back cache of 8MB. Finally, the main memory is modeled as a 16GB, 1 rank, 8-bank, single DDR3-1600 channel SLC PCM with a read latency of 60ns and a write latency of 150ns [2]. MARSS is also modified to integrate a 512kB 32-way set-associative counter cache [70] with the processor-side memory controller for all the techniques evaluated in this work. The OTP generation latency of a 128-bit AES crypto-engine is 72ns [53, 70].

Workloads: To evaluate system performance in real-world scenarios, these evaluations use 5 composite workloads with each workload containing 4 SPEC CPU2006 [38] benchmarks. Each core runs one benchmark, forming multi-programming workloads operating in different virtual address spaces. Table 6 lists these workloads along with their corresponding LLC misses per kilo-instruction (MPKI) values as reported by MARSS [42]. For fairness, these evaluations employ a variety of workloads with different (high/medium/low) MPKI values.

Table 5: **SPEC CPU2006 [38] composite workloads.**

Workload	Benchmarks	MPKI
WD_1	perlbench, soplex, bwaves, lbm	21.05
WD_2	perlbench, soplex, milc, povray	14.20
WD_3	h264ref, astar, milc, lbm	20.40
WD_4	h264ref, sjeng, bwaves, povray	28.22
WD_5	perlbench, leslie3d, GemsFDTD, lbm	12.71

5.3.1 System availability

In order to determine an appropriate counter size for ACME, the maximum memory writes per address for each SPEC CPU2006 benchmark is computed. Assuming a constant counter growth rate, simulation results are extrapolated to a system that continuously writes back to the memory via a DDR3 data bus until the fastest growing counter overflows. It is observed that 24 bits is the smallest counter size that can ensure 99.999% system availability for ACME. Note that a similar approach was followed by the authors in [16] to determine the counter overflow rate.

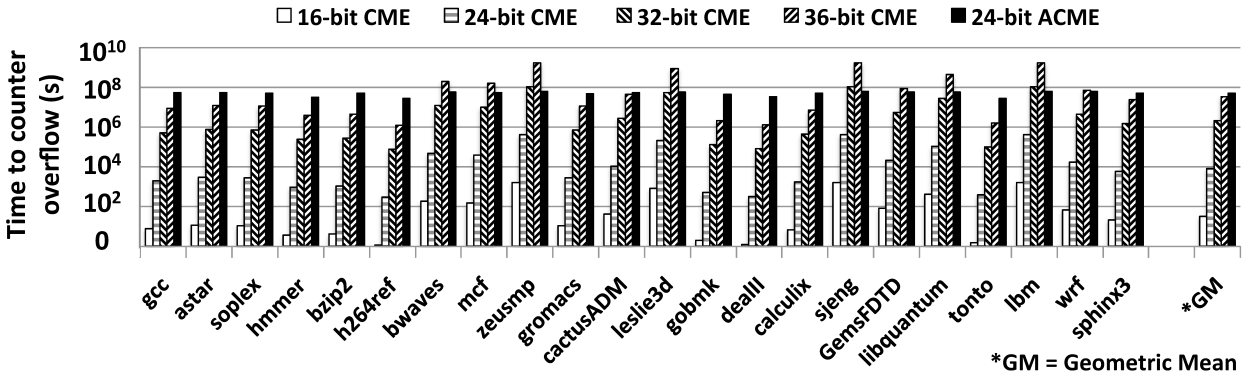


Figure 23: **Time to counter overflow of 24-bit ACME and various CME schemes for the SPEC CPU2006 benchmarks. Results show that despite its smaller counter size, 24-bit ACME increases the time to counter overflow in comparison to 36-bit CME.**

Figure 23 contrasts time to counter overflow of ACME with different CME schemes. Our results show that for a majority of applications, CWL extends the time to counter overflow (i.e., reduces counter overflow frequency) of ACME. However, for applications like bwaves, mcf, zeusmp, leslie3d, and sjeng, where the disparity in counter growth is not significant, the large counters overflow later. Hence, for these applications, the time to counter overflow for 36-bit CME is longer in comparison to ACME. On average, the time to counter overflow of ACME is 4.87×10^7 seconds, i.e., ACME sees one full memory re-encryption every 564 days, which is the highest in comparison to other CME schemes. Since full memory re-encryption freezes the system for $\approx$ 1 minute (refer section 5.1), ACME achieves a system availability of 99.999%. In contrast, CME with equivalent memory overhead, i.e., 24-bit CME results in only 99.3% system availability, which may be unacceptable for web/database servers in practice. Whereas 36-bit CME achieves 99.999% system availability, it requires 50% additional counter memory in comparison to ACME (24 bits/line for ACME versus 36 bits/line for CME).

5.3.2 System performance

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

Figure 24 reports the system performance in instructions per cycle (IPC), normalized to ideal CME, which does not incur OTP generation latency. It is observed that 24-bit ACME improves IPC by 20% and 3.8% in comparison to 36-bit conventional CME and 36-bit CME+CCU, respectively. The high IPC of 24-bit ACME results from (i) the reduced write overhead of concurrent counter updates (refer section 5.2.3) and, (ii) the use of smaller counters (24 bits vs. 36 bits), which improves the counter cache hit rate (note that the counter cache size is constant for fairness of comparison), thereby lowering OTP generation latency. Furthermore, the IPC gains from the high

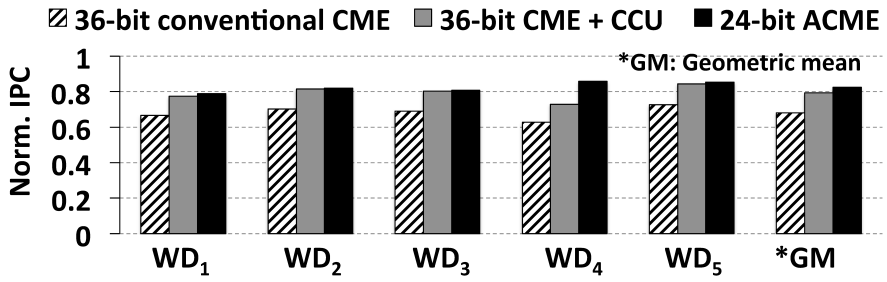


Figure 24: **Comparison of IPC (normalized to ideal CME) of 24-bit ACME with 36-bit conventional CME, and 36-bit CME with concurrent counter updates (CCU).**

counter cache hit rate are prominent for the workloads with high MPKI values (e.g., WD_4), since these workloads frequently access the counter cache for read/write operations.

5.3.3 Denial of memory service (DoMS) attack

To validate the robustness of ACME to availability attacks, a DoMS attack is launched on a Linux machine with an Intel Core i3 CPU and 16GB main memory. The attack invokes `clflush` and `mfence` instructions (using the code in section 5.2.5) to sidestep all levels of cache and constantly writes back to the same memory address until the counter associated with that address overflows.

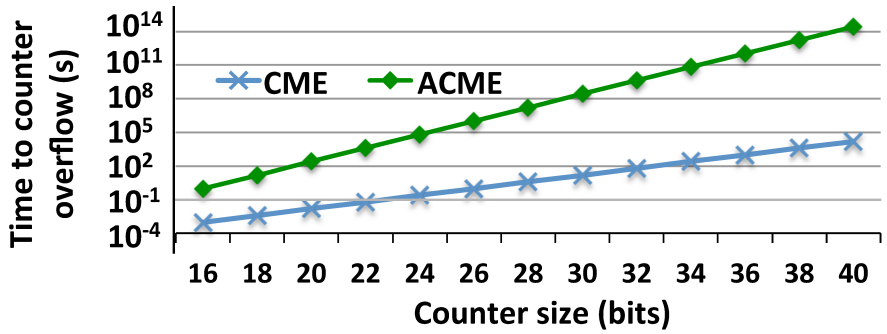


Figure 25: **Time to counter overflow versus counter size for ACME and CME in the presence of DoMS attack.**

For a system under a DoMS attack, figure 25 compares the time to counter overflow of ACME and CME for different counter sizes. It is observed observe that the time to counter overflow for 24-bit ACME is 62911 seconds or 17.47 hours for a system availability of 99.9%. In contrast, the time to counter overflow for 24-bit CME is less than a second, which corresponds to a largely non-operational system. Due to CWL, ACME effectively increases the width of a counter by $\log_2 n$, where n is the number of counters in a Start-Gap region. Since ACME allocates 2^{18} cache lines (and hence 2^{18} counters) per Start-Gap region, the effective width of a 24-bit counter in ACME is $\log_2(2^{24} \times 2^{18})$, i.e., 42 bits. Thus, 24-bit ACME is equivalent to 42-bit CME in terms of system availability, and the results in figure 25 demonstrate excellent agreement between theory and practice.

5.4 RELATED WORK

Early solutions for NVM security, e.g., i-NVMM [10], performed memory-side encryption, and hence were ineffective against the bus snooping attack. To address this vulnerability, state-of-the-art NVM security solutions, e.g., DEUCE [12], SECRET [23], and COVERT [70] perform encryption inside the secure processor-side memory controller. Whereas these solutions are secure against data confidentiality attacks (e.g., stolen DIMM and bus snooping), due to the underlying CME, these techniques are still vulnerable to the system freeze problem due to counter overflow.

Split-CME [16] secures DRAM-based memories against data confidentiality attacks while ensuring high system availability. However, split-CME employs CME memory organization, where the ciphertext and counter are stored in separate memory regions. Hence, split-CME incurs high performance overhead to support consistent counter updates in NVMs. Note that split-CME cannot store the counters alongside the ciphertext due its complicated dual counter design, where a global counter is shared by all the lines in a page; hence, the evaluation of split-CME with ACME is beyond the scope of this work.

5.5 ACME + STATE-OF-THE-ART IN NVM SECURITY

ACME is a drop-in replacement for CME in state-of-the-art in NVM security like DEUCE [12], SECRET [23], and COVERT [70]. ACME was integrated and evaluated with these techniques, results were as follows. Results show that DEUCE+ACME (i) reduces memory overhead by 14.2% (28-bit counter per cache line for DEUCE vs. 24-bit counter per cache line for DEUCE+ACME), (ii) improves system availability by 300× (because ACME replaces DEUCE's logical counters with physical counters and performs CWL to delay counter overflow), and (iii) improves IPC by 1% in comparison to DEUCE (due to improved caching of the small counters of DEUCE+ACME in the counter cache). Note that since SECRET extends DEUCE to only address write energy and latency of the ciphertext without modifying the underlying encryption technique (i.e., conventional CME), SECRET+ACME registers similar improvements to DEUCE+ACME. Finally, COVERT employs idle error correcting pointers (ECPs) to extend an overflowing counter. However, since the availability of idle ECPs reduces as the memory ages, the ability of COVERT to delay counter overflow decreases with time. Integrating ACME with COVERT improves COVERT's system availability (during the initial years of operation) from 99.3% to 99.999% for no overhead. The system availability evaluations of ACME+COVERT for different phases of memory lifetime are excluded for brevity.

5.6 ACME: CONCLUSIONS

ACME is a low overhead CME-based encryption solution to realize the twin security goals of confidentiality and availability in NVMs. ACME employs counter write leveling to reduce full memory re-encryption frequency due to counter overflow, without compromising the security of the underlying CME. ACME is the first work to study and propose a solution for the DoMS attack in the CME-based NVMs. In comparison to CME, ACME incurs 50% lower counter memory overhead and improves system performance by 20%. Further, when subject to a DoMS attack, the ACME-based system provides 99.9% system availability in contrast to a classical CME-based system that is rendered non-operational.

6.0 STASH: SECURITY ARCHITECTURE FOR SMART HYBRID MEMORIES

6.1 STASH: MOTIVATION

Whereas novel resistance-class non-volatile memories (NVMs) such as phase change memory (PCM), resistive RAM (RRAM), and 3D XPoint memory [2, 5, 7] provide low power, dense, scalable alternatives to DRAM, the high latency and low endurance of these NVMs are impediments to the rapid commercialization of NVM-only memory systems [3, 44, 73].

Smart hybrid memories (SHMs) is the convergence of the best of hybrid NVM-DRAM memories [3, 73] with smart memories such as the hybrid memory cube (HMC) [46]. In hybrid NVM-DRAM memories, the DRAM caches a majority of accesses to the NVM, effectively concealing the high latency of the NVM and also improving NVM lifetime. In smart memories like the HMC, the integration of on-module processor logic can support high-bandwidth packetized Serializer/Deserializer (SerDes) processor-memory transfers. Thus, SHMs that integrate NVM, DRAM (as the NVM cache), and processor logic can provide high bandwidth, low memory latency, and high memory density to meet the needs of future high-performance computing systems. However, the unsecure DRAM, NVM, and/or memory buses in SHMs are vulnerable to data confidentiality attacks (e.g., memory scanning and bus snooping attacks) [10, 12, 14, 23, 26–29] and data integrity attacks (e.g., spoofing, splicing, and replay attacks) [17, 27–29] that must be addressed prior to commercialization.

SHMs pose unique challenges to the direct adoption of state-of-the-art security solutions like ObfusMem [29] and InvisiMem [28], which address data confidentiality, data integrity, and memory side-channel attacks in NVM-/DRAM-only smart memories. These include the (i) presence of both volatile and non-volatile memory modules, which render SHMs vulnerable to security attacks in both operational and powered down states, (ii) need for consistent NVM updates to tol-

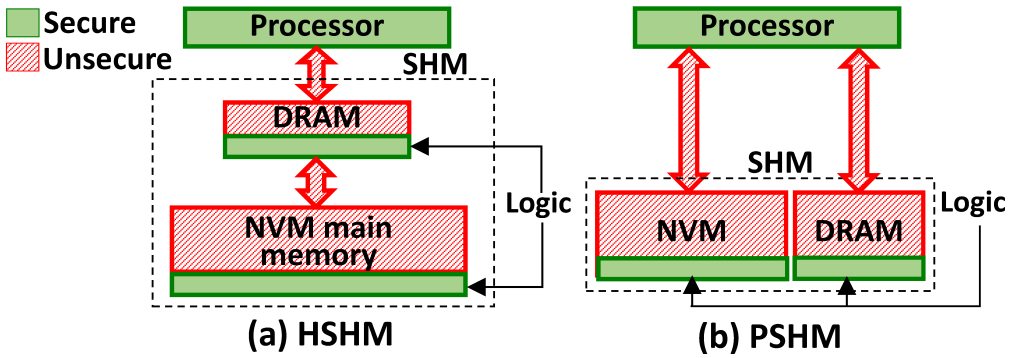


Figure 26: **SHM based on (a) HSHM and (b) PSHM architectures. Whereas HSHM uses a small, OS-transparent smart DRAM as a cache with smart NVM, PSHM uses a large, OS-managed smart DRAM in parallel with smart NVM.**

erate power/system failures, i.e., instant data recovery (IDR), and (iii) management and migration of page-granularity data and security meta-data from NVM to DRAM, which increases memory traffic and reduces DRAM utilization. Whereas a strawman solution that addresses these considerations can be derived by extending ObfusMem [29] and InvisiMem [28], this imposes heavy overheads on memory, performance, and NVM lifetime.

6.2 SMART HYBRID MEMORIES (SHMS)

SHMs integrate smart DRAM [46] with smart NVM [74] to realize high bandwidth, low latency, and high density memory systems. Similar to hybrid DRAM-NVM systems [3, 73, 75], there are two classes of SHMs: (i) hierarchical SHMs (HSHM) with smart DRAM as the cache and a larger smart NVM [3, 75] and (ii) parallel SHMs (PSHM) wherein smart DRAM and smart NVM present a unified interface [73]. Since PSHM requires a large DRAM and changes to the operating system (OS) for memory management, PSHM increases both system cost and complexity. In contrast, HSHM uses DRAM as a small hardware-managed cache without any OS modifications. Figure 26 illustrates both architectures. This work assumes HSHM as the underlying SHM; however the solutions developed here are equally applicable to PSHMs.

6.3 STASH: THREAT MODEL

It is common practice to design a secure computing system assuming a trusted computing base (TCB) and a set of valid threats, which constitute the threat model of the secure system [12, 14, 23, 26–29]. Along ObfusMem [29] and InvisiMem [28], our TCB consists of the processor chip, the processing logic of the smart DRAM, and the processing logic of the smart NVM, as illustrated in figure 26. Core parts of the OS (e.g., security kernels) are also trusted. Since the TCB is not susceptible to physical attacks, the processor and the processing logic on the smart DRAM/NVM are capable of performing cryptographic operations for SHM security. The memory layers in the smart DRAM and the smart NVM as well as the processor-DRAM and DRAM-NVM memory buses are untrusted. Our threat model encompasses threats to data confidentiality and integrity in both the operational and powered-down system states.

6.4 STRAWMAN SECURITY ARCHITECTURE (SSA)

The strawman security architecture for SHMs (SHM-SSA) is a comprehensive end-to-end security framework that also supports instant data recovery in the face of power/system failures.

6.4.1 Security primitives of SHM-SSA

For the TCB and threat model summarized in section 6.3, SHMs require protection against: (i) data confidentiality attacks, (ii) data integrity attacks, (iii) side-channel access-pattern-based attacks, and (iv) memory request integrity attacks. The proposed strawman architecture employs:

- **Data encryption** for data confidentiality in memory modules and buses: The National Institute of Standards and Technology (NIST) [76] approves that counter mode encryption (CME) based on the advanced encryption standard (AES) block cipher algorithm can provide cryptographic protection for data confidentiality. Along [16, 26, 29], SHM-SSA employs split-counter mode encryption (split-CME) in the processor to encrypt the data stored in main memory. Figure 27(a) shows the encryption/decryption unit (128-bit AES) for data in the core processor.

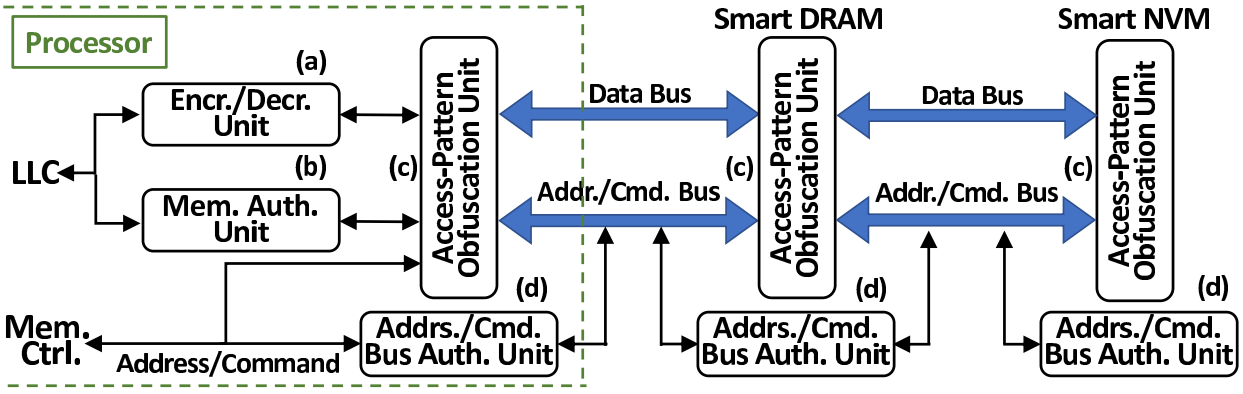


Figure 27: **This figure illustrates the end-to-end security architecture of SHM-SSA. Only the secure processing logic of smart DRAM and smart NVM are shown in the figure.**

- **Memory authentication** for data integrity verification: Without exception, state-of-the-art memory security solutions [26, 27, 29] use Bonsai Merkle Tree (BMT) [17] for memory authentication. BMT stores a 128-bit data MAC (DMAC) per 512-bit cache line to detect spoofing and splicing attacks, and a hierarchical tree structure (Merkle Tree, i.e., MT) of hashes with the counters as its leaf nodes to detect replay attacks. In SHM-SSA, the DMACs and MT nodes are stored in the smart NVM module and verified on the processor, as shown in figure 27(b).

- **Access pattern obfuscation** for data confidentiality against side-channel attacks: In SHM-SSA, memory access encryption is used by the sender (processor/memory) and the receiver (memory/processor) for access pattern obfuscation [28, 29]. On the processor, SHM-SSA encrypts the memory request, address, and write data before transmission. On the memory, SHM-SSA encrypts the requested read data before transmission. This is shown in figure 27(c). This two-way encryption is possible because the secure logic layers on both smart DRAM and smart NVM are equipped with crypto-engines that employ 128-bit AES CME. These crypto engines maintain separate counters, but the counters are incremented in lockstep for consistency. This enables OTP pre-computation for encryption/decryption of the next memory access [28, 29]. The ciphertext (from processor/memory) is re-encrypted before transmission to obfuscate temporal re-use of data [29]. Further, to ensure reads and writes are indistinguishable from each other, dummy ciphertext is transmitted by the processor/memory on a read/write.

- **Memory bus authentication**, i.e., authenticated encryption for memory request integrity verification: Whereas BMT authentication can detect data tampering in memory module/buses, BMT is ineffective against memory request (command/address) tampering. To detect memory request tampering, memory bus authentication is performed using Galois/Counter Mode (GCM) authenticated encryption [28, 77]. In authenticated encryption, the sender (processor/memory) generates an authentication tag (keyed hash value) for the encrypted message (memory access). The tag is also transmitted to the receiver (memory/processor). The receiver regenerates the tag using the received message and verifies it with the received tag to ascertain message integrity. SHM-SSA integrates authenticated encryption logic on the processor as well as memory modules (figure 27(d)).

6.5 SHM-SSA OVERHEADS

The SHM-SSA incurs memory, performance, and endurance overheads over an unsecure SHM , and these are exacerbated by (i) the IDR requirements of SHMs, and (ii) the increased memory traffic and reduced DRAM utilization due to page migration of security meta-data in SHMs.

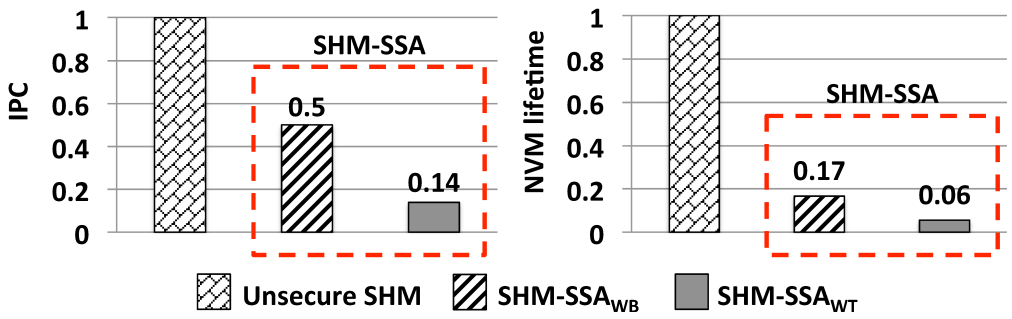


Figure 28: **Comparison of system performance (IPC) and NVM lifetime of unsecure SHM to SHM-SSA$_{WB}$ and SHM-SSA$_{WT}$.**

6.5.1 Security

Figure 28 compares the system performance (measured in instructions per cycle, i.e., IPC) and NVM lifetime of unsecure SHM to SHM-SSA. The SHM integrates a 2GB HMC [46] (as DRAM cache) with a 32GB smart TLC PCM (refer section 6.7 for framework details). Let us consider 2 SHM-SSA variants: (i) SHM-SSA$_{\texttt{WT}}$, which operates smart DRAM and on-chip counter cache in the write-through mode, and (ii) SHM-SSA$_{\texttt{WB}}$, which operates smart DRAM and on-chip counter cache in conventional write-back mode. Whereas SHM-SSA$_{\texttt{WB}}$ only ensures security, SHM-SSA$_{\texttt{WT}}$ ensures both security and IDR (discussed in section 6.5.2). It is observed that in comparison to unsecure SHM, SHM-SSA reduces IPC by at least 2× and reduces NVM lifetime by at least 6×. The key bottlenecks for the low IPC and NVM lifetime of SHM-SSA are (i) latency-intensive BMT authentication, (ii) poor DRAM utilization due to high overhead of the security meta-data (discussed in section 6.5.3), and (iii) high cell flip rate of the hashes/encrypted data.

6.5.2 Instant data recovery

SHMs integrate smart NVMs to enable IDR, which is required for tolerating power/system failures [8, 10], efficient checkpointing [78], and reduced application startup time [79]. Since SHMs use DRAM as a write-back cache to reduce NVM writes, the data residing in the NVM module is stale for the pages that are cached in the DRAM module. In addition, the counters and BMT nodes cached on the processor make reliable data decryption and authentication infeasible during power/system failures, thereby compromising IDR. Whereas SHM-SSA$_{\texttt{WT}}$ achieves IDR by consistently updating ciphertext and meta-data, it significantly increases NVM write traffic, deteriorating system performance and NVM lifetime.

Figure 28 compares the IPC and NVM lifetime of SHM-SSA$_{\texttt{WT}}$ with SHM-SSA$_{\texttt{WB}}$. Results show that SHM-SSA$_{\texttt{WT}}$ reduces IPC by 3.6× and reduces NVM lifetime by 3× in comparison to SHM-SSA$_{\texttt{WB}}$. Hence, achieving IDR with a write-through smart DRAM and on-chip counter cache is infeasible in practice.

6.5.3 Page migration

When SHM-SSA migrates a page from the smart NVM to the smart DRAM, it mandates the migration of its corresponding meta-data to the smart DRAM for data decryption and/or authentication. Since the smart DRAM in SHMs is organized at a 4kB page-level granularity, meta-data migration to the smart DRAM requires migrating multiple 4kB meta-data pages (counter page, DMAC page, etc.), which contain the meta-data corresponding to multiple data pages. This bulk meta-data migration to the smart DRAM is wasteful since the extra meta-data occupies space in the smart DRAM and is not used until the data pages corresponding to this extra meta-data are also migrated to the smart DRAM. Since SHM-SSA uses BMT authentication, which incurs more than 1kB meta-data overhead per 4kB page, the effective DRAM capacity for SHM-SSA is reduced by over 25%.

6.6 STASH: CONTRIBUTIONS

STASH is a refinement of SHM-SSA that integrates PMT, RECOUP, and PACT to overcome security, IDR, and page migration overheads, respectively, of SHM-SSA.

6.6.1 STASH: PMT

Page-level Merkle Tree (PMT) is a low overhead authentication solution for SHMs to reduce the memory, performance, and endurance overheads of BMT [17]. In SHMs, data is migrated from the smart NVM to the smart DRAM at 4kB page granularity (i.e., 64 512-bit cache lines), which mandates transferring of 64 DMACs (one for each cache line) for integrity verification. *However, if the 64 DMACs per page are replaced by a single 128-bit page-level MAC (PMAC), the memory and performance overhead of authentication can be significantly reduced without compromising security.*

PMT stores a single 128-bit PMAC per 4kB page to detect spoofing and splicing attacks. Similar to the DMAC, a PMAC is generated by a cryptographic hash function (e.g., NIST-approved SHA-1/2/3) using a secret key, page content, page address, and the corresponding counter. Fur-

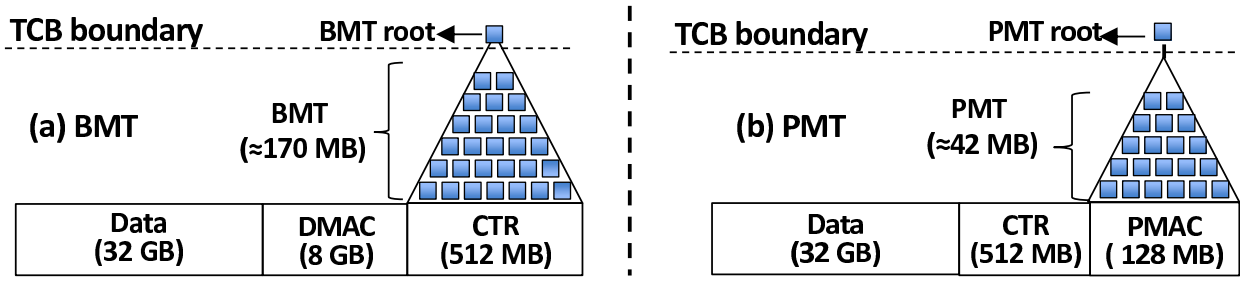


Figure 29: **Data authentication overhead of (a) Bonsai MT (BMT) [17] and (b) page-level MT (PMT). BMT maintains a 128-bit DMAC per 512-bit cache line to detect spoofing and splicing attacks, and a recursively hashed MT over counters to detect replay attacks. In contrast, PMT maintains a 128-bit PMAC per 4kB page to detect spoofing and splicing attacks, and a recursively hashed MT over PMACs to detect replay attacks.**

thermore, to detect replay attacks, STASH constructs an MT by recursively hashing these PMACs, and stores the root of this page-level MT (PMT) on the trusted logic of the smart DRAM. Since PMT is robust against replay attacks, STASH does not maintain a separate MT over counters.

Figure 29 contrasts memory authentication using BMT and PMT. Without loss of generality, this dissertation assumes 32GB main memory data is encrypted using the split-counter mode encryption (split-CME [16]), which requires 7-bit minor counter per cache line and 64-bit major counter per 4kB page, i.e., total counter overhead for 32GB data is 512MB. To protect data against spoofing and splicing attacks, BMT requires 128-bit DMAC per 512-bit cache line, i.e., 8GB DMAC for 32GB data. In addition, BMT constructs an MT by recursively hashing the counters (128-bit hash per 512-bit input) and stores the root of the MT on the TCB. The MT overhead for 512MB counters is ≈170MB. In contrast, PMT stores only a single 128-bit PMAC per 4kB page, and constructs an MT over PMACs. The memory overhead of PMACs for 32GB data is 128MB, and the MT overhead is ≈ 42MB. *Thus, PMT reduces the memory overhead of authentication by ≈ 12.7× in comparison to BMT.*

The PMT is stored in the smart NVM and cached in a secure PMAC cache located on the smart DRAM (discussed in section 6.6.3). The integrity verification of the pages fetched from the smart NVM into the smart DRAM is performed on the processing logic of the smart DRAM. As discussed in [28, 29, 80], a smart-DRAM-like HMC [46] has sufficient area and thermal power budget (55W) to support a low-power processor like the Intel i7-3770T [28, 81]. Hence, PMT

caching and processing for integrity verification can be easily performed on an HMC-like smart DRAM [1].

Since PMT performs memory-side data authentication (on the smart DRAM), the data transferred between the smart DRAM and the processor-side memory controller remains vulnerable to integrity attacks. To counter these attacks, STASH also enforces authenticated encryption between the processor and the smart DRAM. However, when a page is evicted from the smart DRAM, it is stored in the smart NVM without any integrity verification. This does not compromise SHM security because the data residing in the smart NVM is always verified by the smart DRAM before it is forwarded to the processor. Hence, even if data tampering is not detected on an NVM write, it is detected by the smart DRAM on the subsequent read of that data. This is consistent with past work [17, 26, 27] on memory authentication, which advocates integrity verification only on read operations.

6.6.2 STASH: RECOUP

RECOUP addresses the reduced system performance and NVM lifetime limitations of ensuring IDR in SHM-SSA. *RECOUP leverages the key observation that enabling IDR in STASH-based SHMs requires consistent updates to only (a) data (i.e., ciphertext) and its corresponding counter in the NVM module and (b) the PMT root on the TCB of the smart DRAM.* Since STASH generates a PMAC using the page address, data, counter, and a secret key, PMACs can be regenerated after a power/system failure as long as the pages and counters residing in the NVM module are consistently updated. Consistent counter updates are also necessary to reliably decrypt the ciphertext after a power/system failure.

Since the PMT is constructed by recursively hashing the PMACs, RECOUP reconstructs the PMT from the regenerated PMACs after a power/system failure. However, to ascertain the authenticity of the recovered data, the root of the regenerated PMT must match the PMT root stored on the TCB of the smart DRAM. Therefore, the PMT root on the TCB of the smart DRAM must be consistently updated on every write. Any non-volatile on-chip storage (e.g., [83, 84]) can be used

[1] Sharing a single DMAC across multiple cache lines was also proposed in [82]; however, since [82] assumes conventional DRAM with processor-side authentication, shared DMAC verification incurs the overhead of fetching the cache lines that share the DMAC to the processor for authentication. In contrast, STASH performs memory-side authentication of the entire page using PMAC without increasing memory traffic.

to store the PMT root on the TCB of the smart DRAM, thereby ensuring that the PMT root survives reset and/or power/system failures. Consistent PMT root updates do not increase smart NVM writes or impact performance because the PMT root is stored and updated on the smart DRAM TCB.

To consistently update the ciphertext and counters in the smart NVM, STASH uses line-level writes (LLW) [3], which is integral to HSHMs. LLW tracks the writes (at cache-line-level) to the pages residing in the smart DRAM so that only the modified cache lines are written to the smart NVM on page eviction from the smart DRAM. However, instead of waiting for a page eviction, STASH instantly updates a modified cache line and its counter in the NVM module. Further, STASH does not store counters in DRAM (explained in section 6.6.3), but employs a write-through on-chip counter cache for consistent counter updates.

RECOUP can be supplemented with ciphertext and MAC update reduction techniques like SECRET [23] and ASSURE [27], respectively, to decrease NVM writes. SECRET re-encrypts only the modified non-zero words to reduce NVM writes and employs energy masks to reduce the write energy of the ciphertext. ASSURE reduces redundant MAC computations corresponding to the unmodified data and leverages the spatial locality of memory accesses to maintain multi-root MTs that reduce authentication overhead. On integration with PMT, ASSURE maintains a smaller PMT that spans only the hot pages in DRAM and a larger PMT for the rest of the memory. This results in significant latency reduction for the PMT root updates corresponding to the hot pages. Integrating SECRET and ASSURE with STASH incurs memory overhead of 6.25% and 1.6%, respectively [23, 27]. Additionally, ASSURE requires a hot/cold PMAC predictor that incurs a logic overhead of 2k 2-input NAND gates [27].

Techniques like DEUCE [12], SECRET [23], and ASSURE [27] are susceptible to side-channel attacks threatening data confidentiality. *However, since STASH employs access-pattern obfuscation and authenticated encryption, the attacker cannot observe address and/or timing of the unmodified data. Thus, integrating SECRET and ASSURE with STASH does not compromise SHM security.*

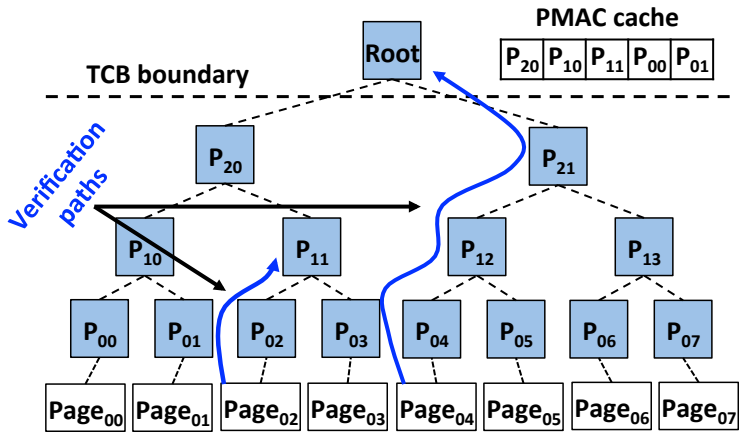


Figure 30: **Illustration of PMT authentication using PMAC cache. The frequently accessed PMT nodes are cached in the PMAC cache and used as local roots to authenticate a fetched page.**

6.6.3 STASH: PACT

PACT is a meta-data (PMACs and counters) management solution to reduce the page migration overheads of SHM-SSA. *PACT is motivated by the observation that PMACs and counters exhibit high spatial and temporal locality, i.e., employing cache-based PMAC and counter management strategies reduce memory traffic and improve DRAM utilization.*

PMAC management: PACT reduces PMAC migration and storage overhead by caching frequently accessed PMACs on the processing logic of the smart DRAM. In BMT-based authentication, frequently accessed MT nodes are cached in the on-chip counter cache, requiring a large counter cache to store both counters and MT nodes. Instead of increasing the capacity of the on-chip counter cache, STASH integrates a separate PMAC cache on the TCB of the smart DRAM. The impact of counter and PMAC cache sizes on performance is discussed in section 6.7.3.

Once a PMT node (including the leaf-level PMACs) is authenticated and stored in the PMAC cache, it is trusted and treated as a local root. As a result, the authentication (on a page fetch) is terminated as soon as a cached PMT node is encountered in the PMAC cache, as illustrated in figure 30. Without loss of generality consider a scenario where smart DRAM requests 2 pages ($Page_{02}$ and $Page_{04}$) from smart NVM. When a page is fetched from the smart NVM, all the PMT

nodes lying on the fetched page's branch are also fetched and verified to detect spoofing, splicing, and replay attacks. However, since the parent hash (P_{11}) of P_{02} is already present in the secure PMAC cache, the authentication process for $Page_{02}$ terminates with the integrity verification of P_{11}. In contrast, since none of the nodes lying on the branch of $Page_{04}$ are present in the PMAC cache, the authentication process for $Page_{04}$ terminates only at the PMT root, which never leaves the TCB. Hence, caching the PMT nodes on the TCB of the smart DRAM not only improves DRAM utilization, but also reduces the authentication overhead (additional hash fetch and verify).

Counter management: Migrating a counter page from the smart NVM to smart DRAM is wasteful, since it transfers extra counters for 63 other pages; these extra counters are not used until their corresponding pages are also migrated to the smart DRAM. PACT eliminates this waste by not storing any counters on the smart DRAM, and instead forwards the required page counters directly to the processor-side write-through counter cache via a pass-thru I/O link [46]. By eliminating bulk counter migration to the smart DRAM, PACT reduces memory traffic and improves DRAM utilization. *To detect counter tampering in smart NVM and data bus, PACT employs PMAC-based counter integrity verification and authenticated encryption, respectively.*

6.6.4 STASH: Security

The enhancements (PMT, RECOUP, and PACT) to the straw-man design (section 6.4) proposed by STASH do not compromise SHM security. As explained in section 6.6.1, PMT is as secure against spoofing, splicing, and replay attacks as BMT. Furthermore, by employing RECOUP, STASH enhances SHM security against power/system failure attacks that can potentially cause corruption or loss of data. Finally, PACT ensures counter security by employing authenticated encryption for counter transmission and PMAC based integrity verification on the DRAM module.

6.7 STASH: EVALUATION AND RESULTS

STASH is evaluated on an SHM system that integrates a 2GB HMC [46] (DRAM cache) with 32GB smart TLC PCM [47]. Both the HMC and the smart TLC PCM are configured to support

high-bandwidth SerDes links (120GB/s [28, 46]). Our evaluations use (i) trace-based simulations to study the impact of SHM security on NVM lifetime and write energy, and (ii) system-level simulations to study the impact of SHM security on system performance.

Evaluated techniques: This section evaluates (i) state-of-the-art ObfusMem [29] (**baseline**), (ii) SHM-SSA (proposed in section 6.4), and (iii) STASH. All the evaluated techniques support IDR by ensuring consistent ciphertext and meta-data (i.e., counters, hashes, etc.) updates in NVMs. For fairness of comparison, ObfusMem and SHM-SSA are extended to integrate write reduction techniques for secure NVMs, namely SECRET [23] and ASSURE [27]; these are integral to STASH. These low-power ObfusMem and low-power SHM-SSA are referred as $\text{ObfusMem}_{\text{LP}}$ and $\text{SHM-SSA}_{\text{LP}}$, respectively, in this dissertation.

Cryptographic primitives: The latency of a 45nm fully-pipelined AES core was obtained from [28, 29]. STASH integrates 2 AES cores on the processor and 2 (1) AES cores on the logic layer of the smart DRAM (NVM) module. Whereas ObfusMem [29] provisions a 256kB on-chip counter cache, STASH employs a 128kB on-chip counter cache and a 128kB PMAC cache on the smart DRAM module.

Trace-driven simulation: For deep, multi-billion instruction evaluation, trace-based simulations are performed using NVMain [37], which is a cycle-accurate main memory simulator. NVMain is a cycle accurate main memory simulator designed to simulate emerging non-volatile memories at the architectural level. NVMain is modified to reflect the SHM system described above.

Full-system simulation: For system-level simulations, the MARSS full-system simulator [42] is used. MARSS is configured to simulate a standard 4-core out-of-order system running at 3GHz. Each core is assigned a private L1 instruction and data cache of 32kB each and a private L2 cache of 128kB; the L3 cache (LLC here) is a single, shared write-back cache of 8MB. Finally, the main memory is configured to reflect the SHM system described above.

Workloads: To evaluate system performance in real-world scenarios, 9 composite workloads (shown in figure 31) are used; each workload containing 4 SPEC CPU2006 [38] benchmarks.

Table 6: **Composite workloads comprising of SPEC CPU2006 benchmarks [38] for full system evaluations.**

Workload	Benchmarks
WD_1	perlbench, soplex, bwaves, lbm
WD_2	perlbench, soplex, milc, povray
WD_3	h264ref, astar, milc, lbm
WD_4	h264ref, sjeng, bwaves, povray
WD_5	perlbench, leslie3d, GemsFDTD, lbm
WD_6	bzip2, gcc, sphinx3, xalancbmk
WD_7	bzip2, gcc, mcf, omnetpp
WD_8	bzip2, mcf, namd, omnetpp
WD_9	GemsFDTD, gcc, omnetpp, xalancbmk

6.7.1 Summary

Table 7 summarizes the security meta-data overhead, system performance (measured in instructions per cycle (IPC)), NVM write energy, and NVM lifetime results for $\text{ObfusMem}_{\text{LP}}$, $\text{SHM-SSA}_{\text{LP}}$, and STASH. Results (except meta-data overhead) are normalized to the **baseline** (i.e., ObfusMem without SECRET+ASSURE). It is observed that STASH outperforms both $\text{ObfusMem}_{\text{LP}}$ and $\text{SHM-SSA}_{\text{LP}}$, with the lowest meta-data overhead and NVM write energy, as well as the highest IPC and NVM lifetime.

Table 7: **Summary of IPC, NVM write energy, and NVM lifetime results normalized to the baseline.**

Architecture	Memory	Meta-data overhead	IPC	NVM write energy	NVM lifetime
$\text{ObfusMem}_{\text{LP}}$	Smart NVM	27%	1×	0.52×	2×
$\text{SHM-SSA}_{\text{LP}}$	SHM	27%	1.32×	0.52×	2×
STASH	SHM	2.1%	1.65×	0.29×	5×

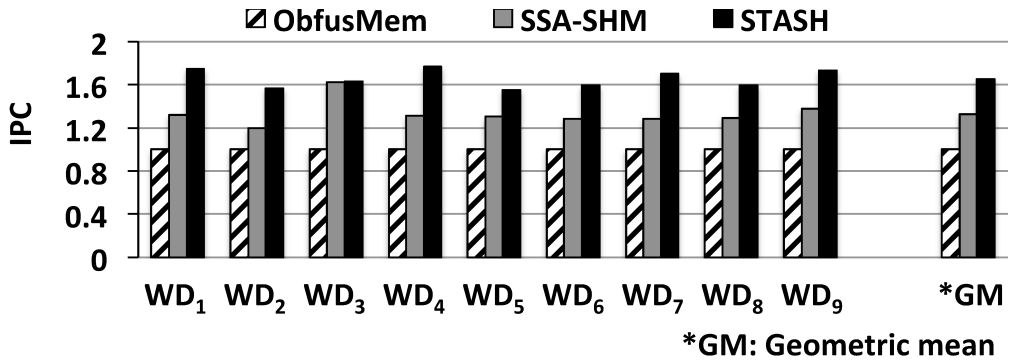


Figure 31: **STASH improves IPC by 65% and 25% in comparison to the ObfusMem$_{LP}$ and SHM-SSA$_{LP}$, respectively, on workloads comprised of SPEC CPU2006 benchmarks [38].**

6.7.2 System performance

Figure 31 illustrates the impact of end-to-end security on system performance (measured in IPC and normalized to the baseline). Along the lines of [17,26], STASH employs speculative execution wherein the requested data is forwarded to the processor while integrity verification is performed in the background. Note that on an integrity verification failure a hardware exception is eventually raised by the processor-side memory controller to halt the system and initiate remedial action by the administrator.

Results show that STASH improves IPC by 65% and 25% in comparison to ObfusMem$_{LP}$ and SHM-SSA$_{LP}$, respectively. Note that these evaluations do not assume a power-constrained NVM system, integration of SECRET+ASSURE does not impact IPC [27]. Hence, the IPC of ObfusMem$_{LP}$ (SHM-SSA$_{LP}$) is equivalent to the IPC of ObfusMem (SHM-SSA). Both STASH and SHM-SSA$_{LP}$ register large IPC gains over ObfusMem due to the use of the smart DRAM cache, which reduces the effective read latency of STASH and SHM-SSA$_{LP}$ by $\approx 10\times$ in comparison to PCM-only ObfusMem (42ns for HMC [28,46] versus 420ns for TLC PCM [47]). Further, STASH also registers high IPC gains over SHM-SSA$_{LP}$ because (i) STASH uses PMACs in place of DMACs and caches frequently accessed PMACs on the TCB of the smart DRAM, resulting in significantly lower page verification and PMT update latency, (ii) RECOUP obviates full MT branch update (in the smart NVM) on every write, reducing the effective write latency, and (iii) PACT improves smart DRAM utilization, resulting in fewer page faults.

6.7.3 Counter cache and PMAC cache sizing

STASH uses 2 types of cache: (a) counter cache and (b) PMAC cache (section 6.6.1). For fairness of comparison, the combined capacity of the counter and PMAC caches in STASH is equal to that of a single counter cache in ObfusMem/SHM-SSA. This section empirically determines counter cache and PMAC cache sizes that achieve the highest system performance (IPC) for STASH.

Figure 32 reports IPC for 5 different cache sizing schemes. In all cases, the combined capacity of the counter and PMAC cache in STASH is kept equal to the total counter cache capacity of ObfusMem/SHM-SSA. As shown in the figure, IPC increases steadily until counter cache and PMAC cache are both 128kB in size. When the counter (PMAC) cache is increased (decreased) above (below) 128kB, IPC starts decreasing sharply. This is because the miss rate of the PMAC cache becomes significant below 128kB. Whereas a counter cache miss incurs only a counter fetch and verification penalty, a PMAC cache miss incurs penalties due to: (i) PMAC fetch, (ii) PMAC child verification (which is delayed if the PMAC is not present in the PMAC cache), and (iii) PMAC verification, which propagates all the way to the PMT root if none of the PMT nodes on the fetched PMAC's branch are cached. Hence, equal-size counter and PMAC caches offer the best performance for STASH.

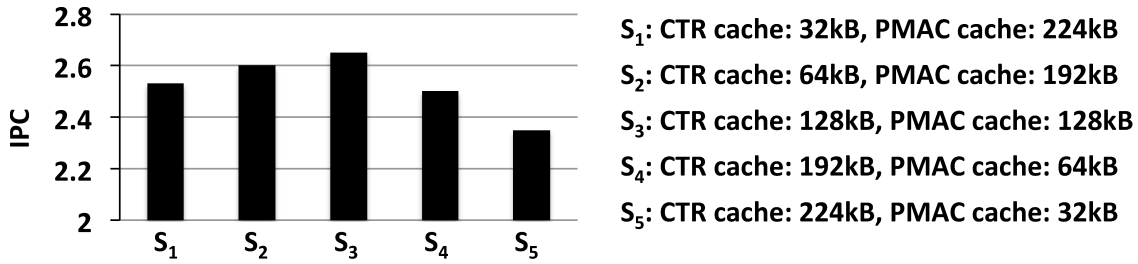


Figure 32: **IPC results for various sizes of counter cache (abbreviated as CTR cache) and PMAC cache. In all cases, the combined capacity of the counter and PMAC caches in STASH is equal to the total counter cache capacity of ObfusMem/SHM-SSA.**

6.7.4 NVM write energy and lifetime

ObfusMem (by extension) and SHM-SSA/STASH (by design) support IDR, increasing NVM writes (section 6.6.2) and negatively impacting NVM write energy and lifetime.

Write energy: In Figure 33(a), it can be observed that the write energy of ObfusMem$_{LP}$ and SHM-SSA$_{LP}$ are equal because both schemes have been extended to perform consistent ciphertext and meta-data updates (e.g., counters, hashes, etc.) for IDR, resulting in the same NVM write traffic for both schemes. In contrast, STASH employs RECOUP for IDR, which only updates the modified cache line in the NVM module and the PMT root on the secure logic of the smart DRAM, to reduce write traffic and write energy in the smart NVM by 45% in comparison to both ObfusMem$_{LP}$ and SHM-SSA$_{LP}$.

Furthermore, SECRET+ASSURE reduces write energy of ObfusMem$_{LP}$, SHM-SSA$_{LP}$, and STASH by 48%, 48%, and 71% over baseline ObfusMem that does not integrate SECRET and ASSURE. The impact of SECRET+ASSURE is most evident for benchmarks that consist of a large number of unmodified and/or zero words, e.g., mcf and gobmk.

NVM lifetime: Figure 33(b) shows that STASH improves NVM lifetime by 2.5× in comparison to both ObfusMem$_{LP}$ and SHM-SSA$_{LP}$. The high NVM lifetime of STASH results from reduced cell writes due to RECOUP. Note that integration of SECRET+ASSURE increases the NVM lifetimes of ObfusMem$_{LP}$, SHM-SSA$_{LP}$, and STASH by 2×, 2×, and 5×, respectively, over baseline ObfusMem.

6.8 STASH: CONCLUSIONS

Smart hybrid memories (SHMs) are an efficient means to bridge the latency and endurance gaps between NVM-only and DRAM-only memory systems. However, SHMs are vulnerable to data confidentiality and integrity attacks that can be executed on the unsecure NVM, DRAM, and/or memory buses. STASH is the first comprehensive end-to-end SecuriTy Architecture for SHMs that integrates (i) counter mode encryption for data confidentiality, (ii) low overhead page-level Merkle Tree (MT) authentication for data integrity, (iii) recovery-compatible MT updates to toler-

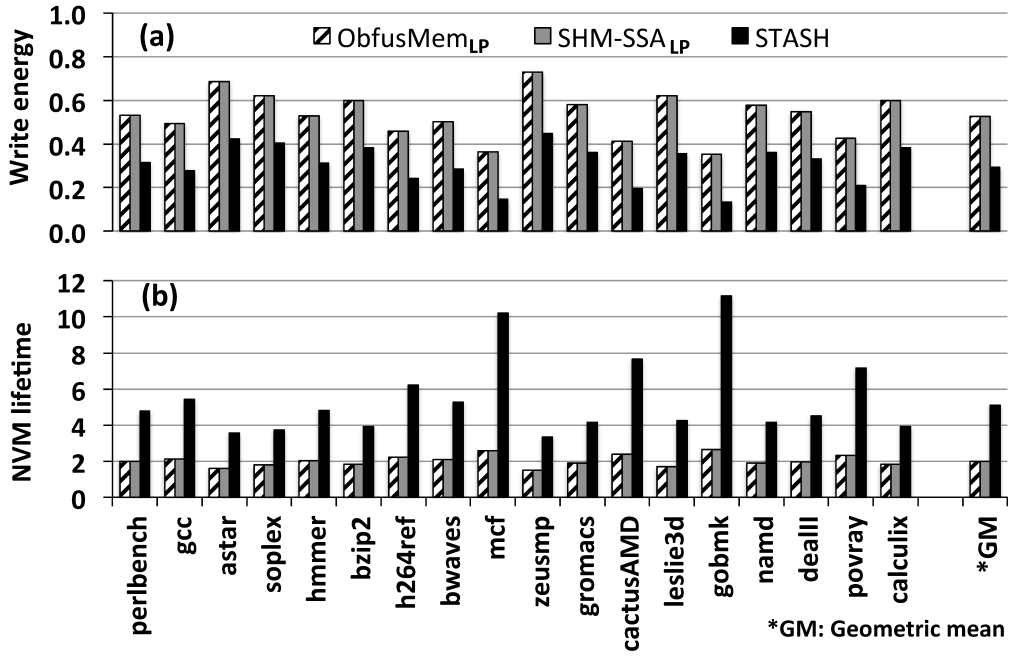


Figure 33: **(a) NVM write energy and (b) NVM lifetime of ObfusMem$_{LP}$, SHM-SSA$_{LP}$, and STASH, normalized to the baseline (ObfusMem without SECRET+ASSURE). The write energy of STASH is 45% lower than both ObfusMem$_{LP}$ and SHM-SSA$_{LP}$. Furthermore, NVM lifetime of STASH is 2.5× higher than both ObfusMem$_{LP}$ and SHM-SSA$_{LP}$. By integrating SECRET+ASSURE, ObfusMem$_{LP}$, SHM-SSA$_{LP}$, and STASH register significant improvements in write energy and NVM lifetime in comparison to the baseline.**

ate power/system failures, and (iv) page-migration-friendly security meta-data management. For security guarantees equivalent to state-of-the-art, STASH reduces memory overhead by 12.7×, improves system performance by 65%, and increases NVM lifetime by 5×.

7.0 FUTURE WORK

Directions for future research will include (i) exploration of holistic architectures that ensure both security and reliability of smart memory systems, (ii) investigating applications of ANCHOR to reduce security overhead of Internet-of-Things, and (iii) extending ANCHOR to safeguard emerging non-volatile processors, especially in the light of advanced attacks like Spectre and Meltdown. The rest of this chapter elaborates these future directions of research.

7.1 SECURITY AND RELIABILITY OF SMART MEMORY SYSTEMS

Next-generation smart NVMs (like Intel Optane [85]) offer high bandwidth and possess on-module processing logic, making them ideal for supercomputing, machine learning, bioinformatics, edge computing, networking applications, etc. [86, 87].

Challenge: Employing smart NVMs for memory-intensive applications poses several challenges on reliability and security fronts. Furthermore, in order to preserve the instant data recovery (IDR) property of smart NVMs, memory transactions must be atomic, consistent, isolated, and durable (i.e, ACID). Maintaining ACID guarantees with security meta-data (encryption counters, hashes, MACs) becomes even more challenging [88].

Potential solutions: Past work [32, 34, 36, 89] has shown that data from real-world applications exhibit significant amount of redundancy. Data compression schemes exploit these redundancies to store data in a compact form, thereby saving memory. Using pattern-based data compression, cache lines can be opportunistically compressed to recover free space, which can be used for storing security meta-data (counters, hashes, etc.). Assuming 64-bit counter and 64-bit MAC, the

minimum compression ratio (i.e., size of original / compressed data) required for this scheme is 1.33 (512/377). By opportunistically storing the ciphertext (i.e., encrypted cache line) alongside its meta-data (as a single 512-bit memory block), this scheme can ensure IDR while also reducing memory read/write traffic, memory writes (wear), power consumption, and improving overall system performance.

Additionally, the 64-bit MAC can also be used to detect different hard/soft errors in NVMs. Since MACs are cryptographic signature of the data, any modification in the data (deliberate or due to memory errors) can be detected during MAC verification. As a result, accessing the error correction data can be avoided if MAC verification does not reveal data modification, saving memory bandwidth and power.

7.2 SECURITY OF INTERNET-OF-THINGS

The International Telecommunication Union defines Internet-of-Things (IoT) as "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies." Through the exploitation of identification, data capture, processing, and communication capabilities, the IoT makes full use of 'things' to offer services to all kinds of applications, while ensuring that security and privacy requirements are fulfilled [90].

Challenge: With billions of devices connected via the Internet, IoT has become an integral part of our lives. Unfortunately, increasing the number of interconnected devices also increases the attack surface, rendering IoT vulnerable to myriad security attacks [91–93]. In the past, attackers have targeted all three cornerstones of IoT: (i) the front-end devices, (ii) the network fabric, and (iii) the back-end cloud. These security attacks can be thwarted by implementing end-to-end security that integrates data acquisition (by front-end devices), forwarding (via the network), and processing (in the back-end cloud). However, integrating security (e.g., encryption and authentication) with IoT is difficult because it (a) increases power consumption of the devices (which are often battery backed), (b) increases compute latency of the devices (unacceptable for mission-critical sensors), (c) reduces the network bandwidth (due to security meta-data overhead), and (d) relies on third-party cloud service providers to ensure data isolation for different users.

Potential solutions: Security solutions proposed in this dissertation like SECRET and STASH can be deployed in IoT devices to reduce power and computation overhead of encryption. Also, Edge/Fog Computing [91], which performs on-device processing, can be leveraged to reduce network traffic (both useful data and security meta-data). This can be coupled with data encoding and compression schemes to save the network bandwidth. Finally, to ensure back-end cloud security, customized security solutions have to be developed for different cloud computing service models. For example, software as a service (SaaS) model (e.g., Google Docs, Facebook, Microsoft Office 365, etc.) only requires application-level security; in contrast, infrastructure as a service (IaaS) model (e.g., Amazon Web Services, Azure, Google Cloud Platform, etc.) requires secure hypervisors, storage, and networking.

7.3 SECURITY OF NON-VOLATILE PROCESSORS

To reduce data movement between the front-end IoT devices (sensors) and the back-end cloud, IoT devices are being equipped with processing capabilities. However, since a majority of IoT devices rely on ambient energy sources such as solar energy, Wi-Fi, Radio Frequency (RF) energy from mobile base stations, these devices cannot perform power-intensive computation/processing that requires uninterrupted power supply. In order to perform complex state-dependent processing on these devices while also meeting the quality-of-service requirements, non-volatile processors (NV-processors) can be used. NV-processors employ non-volatile flip-flops for certain CPU registers and hybrid volatile/non-volatile SRAM for cache. These NV-processors offer the following advantages: (i) nearly zero leakage power, (ii) efficient backup and restore, and (iii) resilience to power failures [94, 95].

Challenge: Spectre [96] and Meltdown [97] are recently discovered vulnerabilities that allow reading of process memory by other unauthorized processes by exploiting the speculative execution property of modern processors. Speculative execution involves predicting the future execution paths and prematurely executing the instructions in them. Speculative execution is highly effective in saving hundreds of processor cycles when future execution paths depend upon some uncached value that has to be fetched from the main memory. If the processor mispredicts some future

execution path, the speculative computation is simply discarded and the correct execution path is taken. However, the discarded values of a mispredicted execution path are not erased from the system and can be accessed by other processes using side-channel attacks. These vulnerabilities are exacerbated in NV-processors, which follow a copy-on-write (CoW) approach when writing to non-volatile parts such as NV-registers and NV-SRAM.

Furthermore, NV-processors perform batch checkpointing, where as many NV-registers are backed up as allowed by the power budget. If an attacker interrupts the power supply in between the batches, it causes the backup to be inconsistent, which may potentially lead to control and/or data faults in the processor. This is unacceptable for mission critical devices [98].

Potential solutions: Securing NV-processors requires redefining the conventional threat model that has hitherto considered the processor chip to be secure and tamper-proof. In the light of advanced attacks like Spectre and Meltdown, which can be launched from the user space, secure processor assumptions have to be augmented, especially for NV-processors. This requires encrypting and authenticating the data stored in the on-chip NV-registers and NV-SRAM. These technique incur power and performance overhead and will also require significant architectural changes in NV-processors. Additionally, checkpointing should be secured to preclude unauthorized data tampering between batch backups.

BIBLIOGRAPHY

[1] *International Technology Roadmap for Semiconductors*, 2011.

[2] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. Intl. Symposium on Computer Architecture*, 2009.

[3] M. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. Intl. Symposium on Computer Architecture*, 2009.

[4] H. Akinaga and H. Shima, "Resistive random access memory (ReRAM) based on metal oxides," *Proceedings of the IEEE*, 2010.

[5] C. Xu, D. Niu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Understanding the trade-offs in multi-level cell ReRAM memory design," in *Proc. Design Automation Conference*, 2013.

[6] E. Kltrsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *Proc. Intl. Symposium on Performance Analysis of Systems and Software*, 2013.

[7] https://www.micron.com/about/our-innovation/3d-xpoint-technology.

[8] W. Enck, K. Butler, T. Richardson, P. McDaniel, and A. Smith, "Defending against attacks on main memory persistence," in *Proc. Annual Computer Security Applications Conference*, 2008.

[9] W. Enck, K. Butler, T. Richardson, and P. McDaniel, "Securing non-volatile main memory," *Technical Report, Pennsylvania State University*, 2008.

[10] S. Chhabra and Y. Solihin, "i-NVMM: A secure non-volatile main memory system with incremental encryption," in *Proc. Intl. Symposium on Computer Architecture*, 2011.

[11] J. Kong and H. Zhou, "Improving privacy and lifetime of PCM-based main memory," in *Proc. Intl. Conference on Dependable Systems and Networks*, 2010.

[12] V. Young, P. J. Nair, and M. K. Qureshi, "DEUCE: Write-efficient encryption for non-volatile memories," in *Proc. Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.

[13] A. Awad, P. Manadhata, S. Haber, Y. Solihin, and W. Horne, "Silent Shredder: Zero-cost shredding for secure non-volatile main memory controllers," in *Proc. Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, 2016.

[14] R. B. Lee, "Security basics for computer architects," *Synthesis Lectures on Computer Architecture*, 2013.

[15] T. Moscibroda and O. Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," in *Proc. USENIX Security Symposium*, 2007.

[16] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," in *Proc. Intl. Symposium on Computer Architecture*, 2006.

[17] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using Address Independent Seed Encryption and Bonsai Merkle Trees to make secure processors OS- and performance-friendly," in *Proc. Intl. Symposium on Microarchitecture*, 2007.

[18] R. Elbaz, D. Champagne, C. Gebotys, R. Lee, N. Potlapally, and L. Torres, "Hardware mechanisms for memory authentication: A survey of existing techniques and engines," *Transactions on Computational Science IV*, 2009.

[19] B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu, "A low power phase change random access memory using a data-comparison write scheme," in *Proc. Intl. Symposium on Circuits and Systems*, 2007.

[20] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proc. Intl. Symposium on Microarchitecture*, 2009.

[21] G. E. Suh, D. Clarke, B. Gassend, M. v. Dijk, and S. Devadas, "Efficient memory integrity verification and encryption for secure processors," in *Proc. Intl. Symposium on Microarchitecture*, 2003.

[22] J. Yang, L. Gao, and Y. Zhang, "Improving memory encryption performance in secure processors," *IEEE Transactions on Computers*, 2005.

[23] S. Swami, J. Rakshit, and K. Mohanram, "SECRET: Smartly encrypted energy efficient non-volatile memories," in *Proc. Design Automation Conference*, 2016.

[24] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. Symposium on Security and Privacy*, 1980.

[25] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor, "Checking the correctness of memories," *Algorithmica*, 1994.

[26] J. Lee, T. Kim, and J. Huh, "Reducing the memory bandwidth overheads of hardware security support for multi-core processors," *IEEE Transactions on Computers*, 2016.

[27] J. Rakshit and K. Mohanram, "ASSURE: Authentication scheme for secure energy efficient non-volatile memories," in *Proc. Design Automation Conference*, 2017.

[28] S. Aga and S. Narayanasamy, "InvisiMem: Smart memory defenses for memory bus side channel," in *Proc. Intl. Symposium on Computer Architecture*, 2017.

[29] A. Awad, Y. Wang, D. Shands, and Y. Solihin, "Obfusmem: A low-overhead access obfuscation for trusted memories," in *Proc. Intl. Symposium on Computer Architecture*, 2017.

[30] J. Rakshit and K. Mohanram, "LEO: Low overhead encryption ORAM for non-volatile memories," *IEEE Computer Architecture Letters*, 2018.

[31] C. Liu, A. Harris, M. Maas, M. Hicks, M. Tiwari, and E. Shi, "Ghostrider: A hardware-software system for memory trace oblivious computation," *Proc. Intl. Conference on Architectural Support for Programming Languages and Operating System*, 2015.

[32] A. R. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for L2 caches," *Technical Report, Dept. Computer Science, University of Wisconsin-Madison*, 2004.

[33] M. Ekman and P. Stenström, "A robust main-memory compression scheme," in *Intl. Symposium on Computer Architecture*, 2005.

[34] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proc. Intl. Conference on Parallel Architectures and Compilation Techniques*, 2012.

[35] S. Sardashti, A. Arelakis, P. Stenstrom, and D. A. Wood, *A Primer on compression in the memory hierarchy*. Morgan & Claypool Publishers, 2015.

[36] P. M. Palangappa and K. Mohanram, "CompEx++: Compression-expansion coding for energy, latency, and lifetime improvements in MLC/TLC NVMs," *ACM Transactions on Architecture and Code Optimization*, 2017.

[37] M. Poremba and Y. Xie, "NVMain: An architectural-level main memory simulator for emerging non-volatile memories," in *Proc. Computer Society Annual Symposium on VLSI*, 2012.

[38] J. L. Henning, "SPEC CPU2006 benchmark descriptions," in *ACM SIGARCH Computer Architecture News*, 2006.

[39] S. Schechter, G. H. Loh, K. Straus, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," in *Proc. Intl. Symposium Computer Architecture*, 2010.

[40] M. Qureshi, "Pay-As-You-Go: Low-overhead hard error correction for phase change memories," in *Proc. Intl. Symposium Microarchitecture*, 2011.

[41] Y. Choi *et al.*, "A 20nm 1.8 V 8Gb PRAM with 40MB/s program bandwidth," in *Proc. Intl. Solid-State Circuits Conference*, 2012.

[42] A. Patel, F. Afram, and K. Ghose, "Bedeschi, Ferdinando and Fackenthal, Rich and Resta, Claudio and Donze, Enzo Michele and Jagasivamani, Meenatchi and Buda, Egidio and Pellizzer, Fabio and Chow, David and Cabrini, Alessandro and Calvi, Giacomo Matteo Angelo and othersMarss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors," in *Proc. Design Automation Conference*, 2011.

[43] M. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proc. Intl. Symposium on Microarchitecture*, 2009.

[44] S. Swami and K. Mohanram, "Reliable non-volatile memories: Techniques and measures," *IEEE Design & Test*, 2017.

[45] C.-K. Luk *et al.*, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proc. Conference on Programming Language Design and Implementation*, 2005.

[46] J. T. Pawlowski, "Hybrid memory cube (hmc)," in *IEEE Hot Chips Symposium*, 2011.

[47] M. Stanisavljevic, H. Pozidis, A. Athmanathan, N. Papandreou, T. Mittelholzer, and E. Eleftheriou, "Demonstration of reliable triple-level-cell (TLC) phase change memory," in *IEEE Intl. Memory Workshop*, 2016.

[48] S. Raoux, G. Burr, M. Breitwisch, C. Rettner, Y. Chen, R. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H. Lung, and C. Lam, "Phase-change random access memory: A scalable technology," *IBM Journal of Research and Development*, 2008.

[49] S.-S. Sheu, K.-H. Cheng, M.-F. Chang, P.-C. Chiang, W.-P. Lin, H.-Y. Lee, P.-S. Chen, Y.-S. Chen, T.-Y. Wu, F. T. Chen, *et al.*, "Fast-write resistive RAM (RRAM) for embedded applications," *Design Test of Computers, IEEE*, 2011.

[50] Z. Pajouhi, X. Fong, and K. Roy, "Device/Circuit/Architecture co-design of reliable STT-MRAM," in *Proc. Intl. Design, Automation & Test in Europe Conference & Exhibition*, 2015.

[51] W. Wen, Y. Zhang, M. Mao, and Y. Chen, "State-restrict MLC STT-RAM designs for high-reliable high-performance memory system," in *Proc. Design Automation Conference*, 2014.

[52] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: Cold-boot attacks on encryption keys," *Communications of the ACM*, 2009.

[53] W. Shi, H.-H. S. Lee, M. Ghosh, C. Lu, and A. Boldyreva, "High efficiency counter mode security architecture via prediction and precomputation," in *Proc. Intl. Symposium on Computer Architecture*, 2005.

[54] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proc. Intl. Symposium on Computer Architecture*, 2014.

[55] M. Seaborn and T. Dullien, "Exploiting the DRAM row-hammer bug to gain kernel privileges," *Black Hat*, 2015.

[56] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One bit flips, one cloud flops: Cross-VM row-hammer attacks and privilege escalation," in *USENIX Security Symposium*, 2016.

[57] T. S. Lehman, A. D. Hilton, and B. C. Lee, "PoisonIvy: Safe speculation for secure memory," in *Proc. Intl. Symposium on Microarchitecture*, 2016.

[58] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems*.

[59] G. E. Suh, D. Clarke, B. Gassend, M. Van Dijk, and S. Devadas, "AEGIS: Architecture for tamper-evident and tamper-resistant processing," in *Proc. Intl. Conference on Supercomputing*, 2003.

[60] W. E. Hall and C. S. Jutla, "Parallelizable authentication trees," in *Selected Areas in Cryptography*, Springer, 2005.

[61] R. Elbaz, D. Champagne, R. Lee, L. Torres, G. Sassatelli, and P. Guillemin, "TEC-Tree: A low-cost, parallelizable tree for efficient defense against memory replay attacks," *Cryptographic Hardware and Embedded Systems*, 2007.

[62] F. Bedeschi, R. Fackenthal, C. Resta, E. M. Donze, M. Jagasivamani, E. Buda, F. Pellizzer, D. Chow, A. Cabrini, G. M. A. Calvi, *et al.*, "A multi-level cell bipolar-selected phase change memory," in *Proc. Intl. Solid-State Circuits Conference*, 2008.

[63] D. Niu, Q. Zou, C. Xu, and Y. Xie, "Low power multi-level cell resistive memory design with incomplete data mapping," in *Proc. Intl. Conference on Computer Design*, 2013.

[64] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[65] S. Mathew, S. Satpathy, V. Suresh, M. Anders, H. Kaul, A. Agarwal, S. Hsu, G. Chen, and R. Krishnamurthy, "340mV–1.1V, 289Gbps/W, 2090-gate nano-AES hardware accelerator with area-optimized encrypt/decrypt GF $(2^4)^2$ polynomials in 22nm tri-Gate CMOS," *Solid-State Circuits*, 2015.

[66] http://ark.intel.com/products.

[67] R. Azevedo, J. D. Davis, K. Strauss, P. Gopalan, M. Manasse, and S. Yekhanin, "Zombie Memory: Extending memory lifetime by reviving dead blocks," in *Proc. Intl. Symposium on Computer Architecture*, 2013.

[68] C. Dubnicki and T. J. LeBlanc, "Adjustable block size coherent caches," in *Proc. Intl. Symposium on Computer Architecture*, 1992.

[69] K. Inoue, K. Kai, and K. Murakami, "Dynamically variable line-size cache exploiting high on-chip memory bandwidth of merged DRAM/logic LSIs," in *Proc. Intl. Symposium on High-Performance Computer Architecture*, 1999.

[70] S. Swami and K. Mohanram, "COVERT: Counter overflow reduction for efficient encryption of non-volatile memories," in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, 2017.

[71] https://googleprojectzero.blogspot.com.

[72] https://perf.wiki.kernel.org/index.php/Main-Page.

[73] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *Proc. DAC*, 2009.

[74] R. K. Ramanujan, G. J. Hinton, and D. J. Zimmerman, "Dynamic partial power down of memory-side cache in a 2-level memory hierarchy." https://www.google.com/patents/US20140304475, 2014.

[75] Y. Zhou, R. Alagappan, A. Memaripour, and A. B. D. Wentzlaff, "HNVM: Hybrid NVM enabled datacenter design and optimization," *Microsoft Research, Tech. Rep.*, 2017.

[76] http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf.

[77] https://csrc.nist.gov/projects/block-cipher-techniques/bcm.

[78] X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, and Y. Xie, "Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exascale systems," in *Proc. Conference on High Performance Computing Networking, Storage and Analysis*, 2009.

[79] H. Kim, H. Lim, D. Manatunga, H. Kim, and G. H. Park, "Accelerating application start-up with non-volatile memory in android systems," *IEEE Micro*, 2015.

[80] Y. Eckert, N. Jayasena, and G. H. Loh, "Thermal feasibility of die-stacked processing in memory," in *Proc. Workshop on Near-Data Processing*, 2014.

[81] https://ark.intel.com/products/65525/Intel-Core-i7-3770T-Processor-8M-Cache-up-to-3.70-GHz.

[82] B. Gassend, G. E. Suh, D. Clarke, M. Van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *Proc. Intl. Symposium on High-Performance Computer Architecture*, 2003.

[83] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *Proc. ISCA*, 2009.

[84] J. Zhao, S. Li, D. H. Yoon, Y. Xie, and N. P. Jouppi, "Kiln: Closing the performance gap between systems with and without persistence support," in *Proc. MICRO*, 2013.

[85] https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html.

[86] J. S. Vetter and S. Mittal, "Opportunities for non-volatile memory systems in extreme-scale high-performance computing," *Computing in Science Engineering*, 2015.

[87] T. Coughlin, "Crossing the chasm to new solid-state storage architectures," *IEEE Consumer Electronics Magazine*, 2016.

[88] S. Liu, A. Kolli, J. Ren, and S. Khan, "Crash consistency in encrypted non-volatile main memory systems," in *Proc. Intl. Symposium on High Performance Computer Architecture*, 2018.

[89] L. Jiang, B. Zhao, Y. Zhang, J. Yang, and B. R. Childers, "Improving write operations in MLC phase change memory," in *Proc. Intl. Symposium on High Performance Computer Architecture*, 2012.

[90] https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060.

[91] A. Alrawais, A. Alhothaily, C. Hu, and X. Cheng, "Fog computing for the internet of things: Security and privacy issues," *IEEE Internet Computing*, 2017.

[92] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, 2017.

[93] M. Conti, A. Dehghantanha, K. Franke, and S. Watson, "Internet of Things security and forensics: Challenges and opportunities," *Elsevier*, 2018.

[94] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Architecture exploration for ambient energy harvesting nonvolatile processors," in *Proc. Intl. Symposium on High Performance Computer Architecture*, 2015.

[95] Y. Liu, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M.-F. Chang, S. John, Y. Xie, *et al.*, "Ambient energy harvesting nonvolatile processors: from circuit to system," in *Proc. Design Automation Conference*, 2015.

[96] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *arXiv preprint arXiv:1801.01203*, 2018.

[97] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *arXiv preprint arXiv:1801.01207*, 2018.

[98] P. Cronin, C. Yang, D. Zhou, K. Qiu, X. Shi, and Y. Liu, "'The danger of sleeping', an exploration of security in non-volatile processors," in *Proc. Asian Hardware Oriented Security and Trust Symposium*, 2017.