MORE^2: Morphable Encryption and Encoding for Secure NVM

Wei Zhao[1], *Dan Feng*[1], *Yu Hua*[1], *Wei Tong*[1], *Jingning Liu*[1], *Jie Xu*[1], *Chunyan Li*[1], *Gaoxiang Xu*[1] and *Yiran Chen*[2]

[1]*Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System*
[1]*School of Computer Science and Technology, Huazhong University of Science and Technology*
[1]*Ministry of Education of China*
[2]*Department of Electrical and Computer Engineering, Duke University, USA*

***Abstract*—Memory encryption can enhance the security of Non-volatile memories (NVMs), but it significantly increases the data bits written to NVMs and leads to severe lifetime and performance degradation. Current encryption techniques aim to reduce the re-encryption to many existing clean words, which unfortunately suffer from high encryption overheads (i.e. latency and energy) and many unnecessary writes. In the meantime, compression techniques can reduce the writes of encrypted NVM. However, we find that they may destroy the data patterns and increase the modified words, resulting in many encryptions in secure NVM.**

In this paper, we propose the MORphable Encryption and Encoding (MORE^2) scheme to address these problems. Our MORphable Encryption (MORE) technique aims to reduce the full-line re-encryption and avoid clean line encryption. Besides, MORE proposes a prediction-based write scheme to avoid the encryption of clean lines, and pre-encrypt the lines that are predicted as dirty. Therefore, MORE can remove the encryption from the critical path of NVM. Furthermore, MORE^2 proposes the Morphable Selective Encoding (MSE) scheme to compress the modified words while preserving clean words. MORE^2 encrypts all metadata with the line counter to guarantee high security. Experimental results show that MORE^2 reduces the bit flips of encrypted NVM by 53.5%, decreases the access latency by 27.32%, improves the IPC performance by 12.1%, and reduces the write energy by 29.1% compared with the state-of-the-art design.

***Index Terms*—NVM, Encryption, Compression**

I. Introduction

Non-volatile Memories (NVMs) such as PCM, ReRAM, and STT-MRAM have the advantages of high density, low leakage power, and fast read speed compared with traditional DRAM technology [1]–[4]. However, the non-volatility property causes security vulnerabilities. The data in the NVM-based DIMM still exist after power off. Attackers can access the sensitive data in the NVM main memory (NVMM) by streaming out the data in NVMs [5]–[7].

Encryption techniques can enhance the security of NVM-based main memory system. For the written back cache lines, the encryption engine encrypts its data with the secure key. Then, the memory controller writes the encrypted data to the NVMM. If the NVMM is stolen or snooped, the adversaries can only get secure encrypted data. Unfortunately, the security guarantee introduces performance and endurance degradation to NVM-based main memory. Firstly, the long encryption latency before a write degrades the write performance. NVMs are expected to store data as persistent memory for instantaneous failure recovery. Since achieving data consistency needs to ensure the ordering of memory writes, the writes are on the critical path of application execution. Thus a processor has to stall and wait for the encryption completed before starting the write process. In conclusion, the serial long encryption latency severely degrades the system performance. Secondly, due to the diffusion property of encryption, writing the encrypted data incurs 50% bit writes, which is $4\times$ compared with the unencrypted NVM [5]–[8]. The largely increased bit writes lead to high write energy consumption, long write latency, and significant lifetime degradation. However, even though without encryption, the write problems are severe enough [9], [10]. For example, the write endurance of PCM is about 10^{10} cycles [10], [11], which is six orders of magnitudes smaller than that of DRAM. Besides, the write energy/latency of PCM is about ten times more than DRAM. As a result, encryption presents severe performance and endurance challenges to NVM.

Existing intra-line[1] encryption techniques [5], [8], [12], [13] propose to improve the write performance of encrypted NVM by avoiding the re-encryption of many clean words existing in the new cache lines [5], [14]. Through these methods, encrypted NVMM suffers from less write pressure. BLE [12] splits a cache line into four 128-bit words and allocates a 2-bit local counter for each word. If one 128-bit word is modified, the corresponding local counter adds one. When one local counter reaches its terminal value, the entire cache line is re-encrypted. SECRET [8] is similar to BLE, and it executes encryption in 8-byte words and removes the re-encryption to zero words as well. DEUCE [5] can avoid the re-encryption to 2-byte words that are always clean in a fixed write epoch. However, once one word is modified in a write operation, then it will be re-encrypted till the end of an epoch even if it is clean. These techniques show good effectiveness, but they have two common weaknesses. Firstly, these techniques' ability to reduce word re-encryption is limited, many clean words are still encrypted. Secondly, the encryption operation and the write operation are serially executed to ensure data correctness, which enlarges the overall system stalls. In this work, we

Corresponding author: Tongwei@hust.edu.cn

[1]Some inter-line techniques [6], [7] are orthogonal to our work.

propose morphable encryption to alleviate these problems.

Data compression techniques such as Base Delta Immediate (BDI) [15], Frequent Pattern Compression (FPC) [16] can reduce the writes of encrypted NVM through compressing data before encryption. For example, BDI compresses a 512-bit all-zero cache line to a 3-bit prefix. Then, only 3-bit data are encrypted and written into NVM. CASTLE [13] integrates compression and expand coding for MLC/TLC encrypted NVMs to improve the write performance and lifetime. However, CASTLE cannot be used for SLC NVMs due to the different cell write operations. Existing compression algorithms leverage the data patterns of the new data to maximize the size reduction, but they ignore the clean words between the writes. Directly using compression techniques in encrypted main memory performs not well. There are two main reasons: (1) The data compaction operations of compression techniques make clean words disappear, leading to many increased modified words. (2) Many modified words cause frequent re-encryption operations. In this work, we design a morphable selective encoding scheme to solve the challenges of current compression schemes in encrypted NVM.

In summary, current encryption and compression techniques of NVM face several challenges: (1) The serial encryption before a write brings performance cost. (2) Many clean words are still re-encrypted. (3) Current compression techniques can reduce data size, but it increases the encrypted words. To solve the challenges above, we propose **MOR**phable **E**ncryption and **E**ncoding (MORE^2) to greatly improve encrypted NVM's performance. The detailed contributions of this paper are as follows:

- ***A Morphable Encryption mechanism without encryption latency.*** We propose morphable counter mode encryption to reduce the full-line and clean line re-encryptions. Next, we propose a prediction-based write scheme to remove the encryption latency. For cache lines that are predicted to be clean, they need no encryption. While for cache lines that are predicted to be dirty, the encryptions are pre-executed to remove latency.
- ***A Morphable Encoding Scheme to Preserve Clean Words and Zero Words.*** We propose an encoding scheme to preserve the clean words and zero words. Firstly, we compact the modified words through the dirtiness tracking bits. Next, the modified and non-zero words are compressed with our morphable encoding techniques.
- ***Efficient Synergization of Encryption and Encoding with High Security.*** We propose **MOR**phable **E**ncryption and **E**ncoding (MORE^2) by combining our proposed techniques. MORE^2 combines the advantage of encryption and encoding, which can largely improve the write performance and endurance of encrypted NVM with a high-security guarantee.
- ***System Level Implementation and Evaluation.*** We implement MORE^2 on Gem5 [17] with NVM simulator NVmain [18], and the results are comprehensively evaluated using benchmarks from SPEC CPU2006 [19] and PARSEC 2.1 [20].

II. Background

A. Attack Models and Memory Encryption

Attack Models. The non-volatility property enables NVM to work with low leakage power and persist data after power off. However, it makes NVMs more vulnerable to be eavesdropped by attackers. Like existing works [5]–[8], this paper aims to discuss two common attacks, including stolen DIMM and bus snooping attacks[2]. When the NVM DIMMs are stolen, due to the non-volatility of NVM, adversaries can stream out the sensitive data from stolen DIMMs. As for the bus snooping attack, since Non-volatile main memory (NVMM) accesses data through the memory bus, attackers can insert a bus snooper or a memory scanner in the bus to obtain the data communicated between the processor and NVMM chip.

Memory encryption can enhance the NVM security. Directly encrypting/decrypting data before a write/read operation is not a smart scheme, which leads to large encryption/decryption latency before the write/read operation. Therefore, naive encryption degrades the system performance. To ensure low encryption/decryption overhead, counter mode encryption (CME) technique [21], [22] is proposed to reduce the decryption latency in the critical path. CME uses a secure global key in the CPU, address of the cache line, and the per-line counter to generate the One Time Pad (OTP) through the Advanced Encryption Standard (AES) circuit. Then, the generated OTP is XORed with the cache line data to get the decrypted/encrypted cache line. Fig. 1(a) shows the process of counter mode encryption. In the read process, AES can generate the OTP through the on-chip cached counter at the same time. Therefore, only the XOR operation is on the critical path. The timeline of the decryption process is shown in Fig. 1(b). In this way, counter mode encryption can largely reduce the decryption latency. Since the XOR operation is simple and fast, the overhead can be negligible.

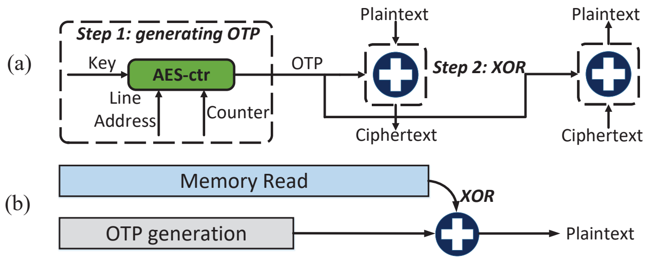


Fig. 1: (a) Counter mode encryption (b) The read operation.

B. Counter Mode Encryption techniques

To improve the write performance of encrypted NVM, several works focus on avoiding the writes to clean words in the lines. BLE [12] splits a cache line into four 128-bit words and allocates a 2-bit local counter for each word. If one 128-bit word is modified, the corresponding local counter adds one. When one local counter reaches its terminal count, the entire cache line is re-encrypted. SECRET [8] similarly reduces re-encryptions, and it can execute encryption in 64-bit words and remove the re-encryption to zero words. In contrast,

[2]We don't consider bus tampering attacks. They can be defended via Merkle Trees-based authentication techniques, which are orthogonal to our work.

DEUCE [5] splits a cache line into 32 16-bit words and maintains a leading counter (LCTR) and a trailing counter (TCTR) for the entire cache line. When a write hits the line, LCTR adds one, and TCTR is the value of `LCTR&0xFFE0`. When the delta of LCTR and TCTR exceeds the pre-determined fixed write epoch, the full-line will be re-encrypted. Fig. 2 shows the architecture of DEUCE. According to the new dirtiness tracking bits, `word2, word5, word6` need to be re-encrypted, while the rest words are clean. The modified words use line counter LCTR to get encrypted data. While clean words retain the last encrypted data, which are encrypted with the TCTR.

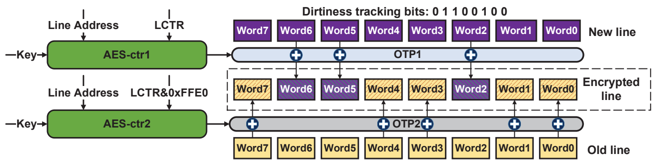


Fig. 2: The architecture overview of DEUCE.

III. Motivation

(1) ***Two weaknesses of current encryption techniques.***

1. Encryption latency leads to performance degradation. Crash-consistent applications for NVMs exhibit a unique property—writes to memory are on the critical path of program execution. For conventional DRAM memory, only reads are on the critical path, while writes can be buffered, coalesced, and reordered on the way to memory for better performance. However, for persistent NVM, the order of writes is severely constrained to ensure data recoverability across failures [7], [23], [24]. Therefore, the serial encryption latency can prolong the write latency and lead to performance degradation. Fig. 3 shows the write process of current encryption techniques (e.g. DEUCE, SECRET). For an incoming write, the memory controller reads and decrypts the encrypted data to get the clean words. While the read process can hide the decryption latency. Then, the encryption logic encrypts the modified words. Finally, the encrypted data and metadata are written into memory chips. The serial encryption can prolong the write latency. Typically, the optimized AES-128 encryption latency is about 40-100 *ns*, while the PCM timing parameter `tWR` is 300 *ns* [7], [24], [25]. In conclusion, encryption can result in large performance degradation.

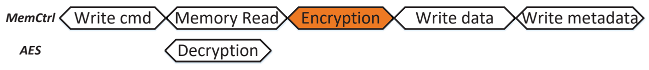


Fig. 3: Serial encryption prolongs the write latency.

2. Many clean words are still encrypted. BLE [12], DEUCE [5], SECRET [8] are effective to avoid the re-encryption to clean words. But there are still many clean words that are encrypted. As for DEUCE, due to the fixed write epoch, it suffers from frequent full-line re-encryption while all the words are modified, even the new data is clean. As for BLE and SECRET, the word sizes are 16B and 8B, and they cannot reduce the encryption of fine-grained clean words. Besides, the local counters (2-bit for a word) are small, leading to frequent overflows.

(2) ***Current compression techniques lead to many modified words.*** Compression techniques can reduce the size of encrypted data. However, they destroy the original data patterns. Fig. 4 shows the normalized compressed line size and the modified words. By using BDI compression, the average line size reduces to 57.4%. As for modified words, the average modified word ratio of no compression is 38.1%, while the ratio of BDI compression is 61.5%. Although compression can significantly reduce the compressed line size, it changes the data layout leading to many increased modified words. Due to this phenomenon, encrypting compressed data may lose effectiveness due to rare existing clean words. Therefore, traditional compression schemes cannot be used for current encryption algorithms.

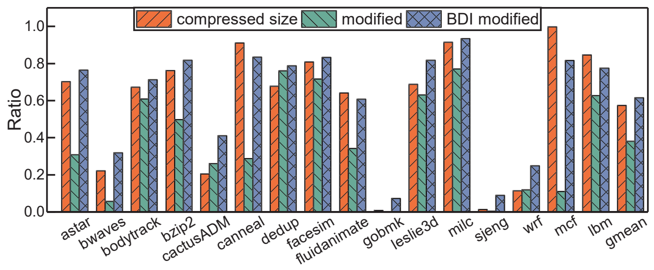


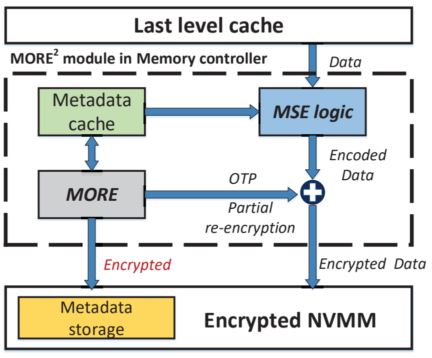
Fig. 4: Compression leads to many modified words.

IV. Design

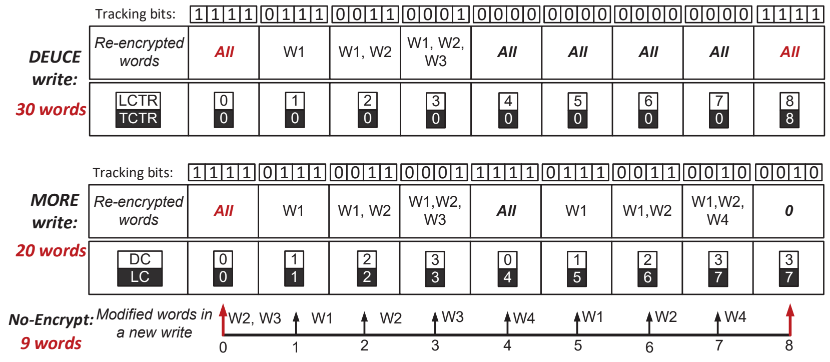
To solve the problems of current encryption techniques, we propose a light-weight morphable encryption (MORE) architecture. MORE can reduce the full-line encryption and avoid the encryption to clean lines. Besides, MORE uses a light-weight prediction-based pipeline to reduce encryption latency and energy consumption. Next, we propose a morphable selective encoding scheme (MSE) to reduce the writes to modified words. We combine MORE and MSE to form MORE^2. The overall architecture of MORE^2 is shown in Fig. 5. When a cache line is written back to NVMM, MSE encoding logic firstly encodes the coming data. Next, the encoded data are encrypted by the morphable encryption engine with low encryption overhead.

A. Lightweight Morphable Encryption

Morphable Encryption. Current encryption techniques cannot fully avoid the re-encryption of clean data. As for DEUCE, there are many full-line re-encryptions when all the dirtiness tracking bits become dirty. While for BLE and SECRET, they can only avoid the encryption of course-grained clean words (16B and 8B, respectively), and they may suffer from frequent full-line re-encryption due to small local counters. Fig. 5(b) shows a write epoch of DEUCE. The write epoch of DEUCE is 8, and each line has 4 words. For the 4-th write of DEUCE, all words in the write epoch are modified. Therefore, DEUCE needs to encrypt the whole line till the end of this epoch. To solve the problem, we re-organize the form of counters to build morphable encryption (MORE). We set a line counter (LC) to record the counter value of the memory line. While a 5-bit delta counter (DC) records the counter difference between the memory line and clean



(a) The architecture of MORE^2.



(b) The write example of DEUCE and our morphable encryption.

Fig. 5: The introduction of overall architecture and morphable encryption.

words. For a coming write, if there are no clean words, the DC resets to '0'. Otherwise, DC pluses one. As for the 4-th write in Fig. 5(b), MORE encounters a full-line re-encryption, then the DC resets to '0' and tracking bits reset to 'clean'. Therefore, MORE starts a new write epoch from 4-th write. When the DC reaches its maximum value, the line needs to be re-encrypted. In summary, MORE can achieve morphable write epochs, removing many full-line re-encryptions.

Besides the full-line re-encryptions, we find that many lines in the main memory are clean. Fig. 6 shows the clean line ratio of several benchmarks. On average, 42.4% of memory lines are clean. As for these lines, we can bypass the encryption and write operations. In comparison, current encryption techniques do not consider this feature, and they directly encrypt all the written back lines even if some evicted lines are clean. For the example in Fig. 5(b), the 8-th write for DEUCE is at the epoch boundary, and DEUCE needs to re-encrypt the line, even though it is clean. While MORE just bypass it. As a result, DEUCE has to write 30 words, while MORE writes 20. To ensure data security, the dirtiness tracking bits and DC are encrypted using the line counter.

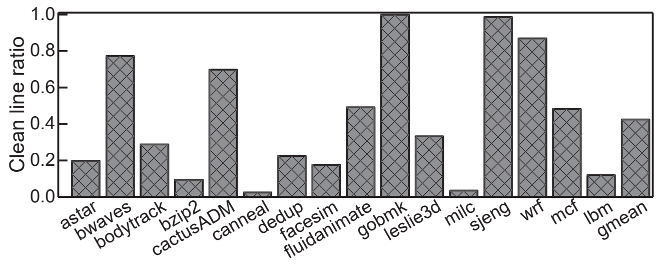


Fig. 6: The clean line ratio.

Reducing Encryption overhead with a light-weight predictor. Encryption leads to extra latency and energy overhead, and these overheads can degrade system performance and increase the overall energy. Previous work Janus [24] proposes to pre-execute encryption operation with the help of software interfaces and compiler. But it is not friendly for users due to the complex modifications to application code. We propose a hardware predictor to optimize the encryption overhead with close performance to Janus. First, we analyze the encrypt-write processes shown in Fig. 7. As for the serial way, when the memory controller receives a write command, the read-decrypt process is used to acquire the clean words information. While decryption can be executed with the memory read in parallel. Next, the encryption logic will encrypt the line if it is not clean. The serial way can avoid encryption to clean lines, but it may suffer from severe encryption latency if many lines are dirty. To reduce the encryption latency, the parallel way pre-executes the encryption together with the last-line decryption. In this way, it needs another AES logic to finish the encryption operation, and the encryption can be removed from the critical path. However, the parallel way encrypts all lines even the lines are clean, leading to high encryption energy. In summary, the serial way has low encryption energy, but high encryption latency, and the parallel way is the opposite of the serial way.

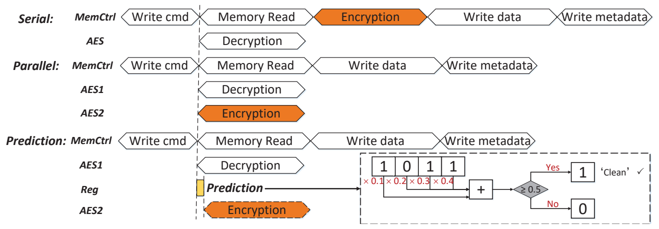
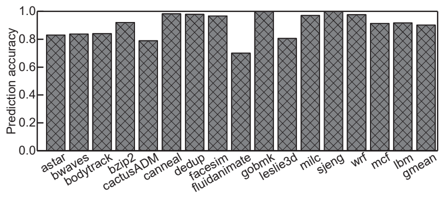
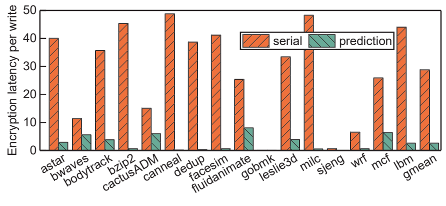


Fig. 7: Line-level prediction-based encryption.

To leverage the strengths of the two ways above, we propose a prediction-based way to reduce the encryption overhead. We use a locality-based predictor to predict whether the next line is clean or not, and we don't encrypt the clean lines. As for a coming write, if the last several writes are clean/dirty, the incoming memory line is probably clean/dirty. Based on the locality, we use the last four writes to predict the incoming write, and each state multiplies its corresponding weight. Fig. 7 shows an example of prediction process. The last four lines are 'clean', 'dirty', 'clean', 'clean', respectively, and then the four weighted values are accumulated. Finally, we get the predicted clean state through the comparison with the threshold. In this example, the predicted state is clean. The overhead of the predictor includes a 4-bit register and simple hardware logic. Therefore, the overhead can be negligible. Fig. 7 also displays the pipeline of our prediction way. When the prediction state is correct, no encryption latency is needed. The read process can validate the predicted value. If a dirty


(a) Prediction accuracy.


(b) Encryption latency of each write (ns).

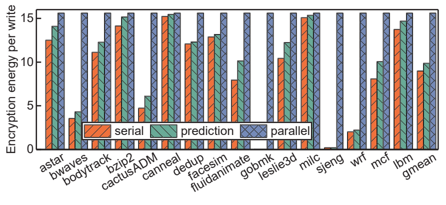

(c) Encryption energy of each write (nJ).

Fig. 8: The performance of prediction-based encryption.

cache line is incorrectly predicted to be clean, the encryption executes after the read. Fig. 8(a) shows the prediction accuracy, in average, 90.2% predictions are right. Fig. 8(b),(c) shows the encryption overhead of the three ways. As for the latency overhead, the values of three ways are 28.8 *ns*, 2.65 *ns*, 0 *ns*, respectively. While for energy overhead per write, the values are 8.99 *nJ*, 9.85 *nJ*, 15.6 *nJ*, respectively. In summary, our prediction-based encryption method can largely reduce encryption overhead with low hardware cost.

B. Morphable Selective Encoding

Traditional compression techniques directly compact the compressed data, increasing the number of modified words. While many modified words may aggravate the write pressure of encrypted NVM. In this section, we propose a Morphable Selective Encoding (MSE) scheme to reduce data writes while preserving the clean words (2-byte). Fig. 9 shows an example of our selective encoding (SE) scheme. Firstly, we exclude the clean words through the new tracking bits of encryption, and we compact the modified words. Next, we find 57.5% of words are zero. Therefore, we remove the zero words in the modified data. We use a simple 1-bit zero tag to record a zero word rather than compress it. In this example, 2nd and 4th modified words are zero, and the corresponding zero tags are "0101". Then, we compress the rest modified words with 32-bit data patterns in TABLE I, while the data patterns are insensitive to 2-byte zeros. Finally, we write the compressed data into the area of modified data, avoiding to destroy the data layout of clean words.

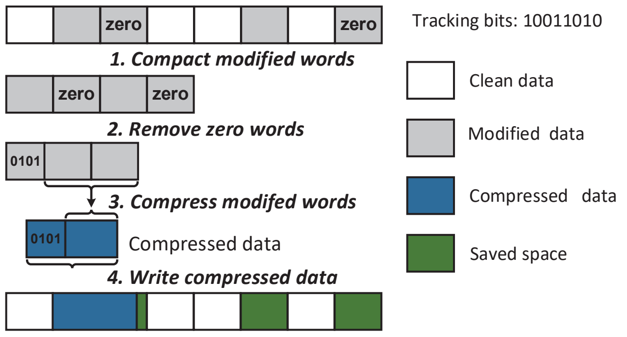


Fig. 9: The example of our selective encoding scheme.

In the compression process, we only compress the rest 2-byte modified words to avoid destroying the data patterns. Besides, we use the matched patterns to compress data rather than other compression algorithms like BDI, because the compacted modified data leads to fragmented data layout. While the fragmented data can destroy the word-level similarity that BDI compression relies on, leading to poor compression effectiveness. As for SE, if there are few zero words, it may perform poorly. Therefore, we use the normal FPC [16] as an alternative compression method. Fig. 10 displays the data flow of our morphable selective encoding (MSE). Compared with SE, FPC method doesn't exclude the zero words in modified words. MSE chooses the compression method that has the minimum compressed size. We assign a 2-bit tag for a line to indicate the compression type and whether this line is compressible or not.

The decoding process of MSE is similar to FPC decoding. While decompressing a memory line, the decompressor can get the compression state and compression type via the 2-bit tag. If the line is compressed by SE, the zero tags can be known through the dirtiness tracking bits. Then, the decompression logic can get original modified words through the zero tag and compressed patterns. While for the FPC way, the decompression process is same with normal FPC.

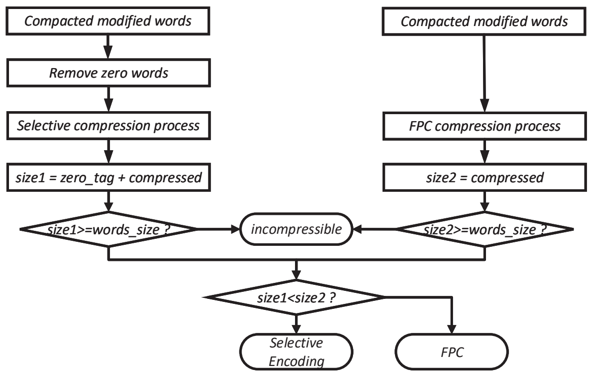


Fig. 10: The data flow of morphable selective encoding.

C. The Implementation of MORE^2

In this section, we combine the morphable encryption and our morphable selective encoding to implement MORE^2.

Write Operation. When NVMM receives a write request, the memory controller executes a read-decrypt-decoding process with the cached metadata to obtain the clean words. At the same time, the predictor uses the state register to predict the clean state of this coming line, and the AES engine pre-executes encryption (OTP generation) through the prediction result. After the read operation, we can know the clean words. Besides, the new tracking bits are updated through the XNOR operation between the old tracking bits and clean tag bits. Then, the encoding logic can encode the modified words

TABLE I: 32-bit zero-insensitive compression patterns of selective encoding [16], [26].

Prefix	Data Pattern	Example	Compressed Example	Encoded Size
000	2-bit sign-extended in each byte	`0x01FE0101`	`0x065`	11 bits
001	4-bit sign-extended in each byte	`0x03F905FE`	`0x1395E`	19 bits
010	1-byte sign-extended	`0xFFFFFFB6`	`0x2B6`	11 bits
011	2-byte sign-extended	`0xFFFFE432`	`0x3E432`	19 bits
100	4-bit padded with a zero 4-bit in each byte	`0x10203040`	`0x41234`	19 bits
101	two halfwords, each a byte sign extended	`0xFFB60036`	`0x5B636`	19 bits
110	4 repeated bytes	`0x20202020`	`0x620`	11 bits

according to the new tracking bits. Next, the encoded modified data and metadata are encrypted through XNOR operation with the new OTP. Finally, they are written into NVMM while preserving the clean data.

Read Operation. When NVM receives a read request, the decryption logic can decrypt data through the tracking bits and the counters. Especially, the modified words are decrypted by the line counter, while clean words use the counter value of `LC-DC`. Next, the MSE decoder decodes the decrypted modified words. Finally, the original data can be sent to CPU.

D. Overhead

Encryption. Compared with the encrypted NVM baseline, we assign 32-bit tracking bits for each line to record clean words, and a 5-bit delta counter is used to remove many full-line re-encryption. Next, the predictor needs a 4-bit register stored in the memory controller, which is negligible. Besides, the prediction-based encryption needs another four AES engines. The area of AES can be neglected compared with CPU [27].

Encoding. As for encoding, we assign a 2-bit tag for each line to record compression metadata. The encoding/decoding latency is similar with FPC, which is about 1.5 *ns* [16]. The encoding/decoding energy are several *pJs* for each line [26], which can be ignored compared with writing a memory line.

Overall, the memory overhead is 7.62% (39/512) for a line, and the hardware logic overheads are so small compared with CPU that can be negligible [8], [26].

E. Security Analysis

In general, the metadata of the NVM must be safe to ensure the security of data. In MORE^2, the metadata is the delta counter, compression tags, and tracking bits. We encrypt all metadata with the line counter to ensure security. When attackers obtain the encrypted data through the ways of our attack models, attackers cannot get the raw data due to the fully encrypted metadata.

V. Evaluation

To verify the effectiveness of MORE^2, we compare it with six different schemes. We use Gem5 [17] with NVMain [18] to evaluate several schemes. Gem5 is an accurate full system simulator, and NVMain is a main memory simulator for NVM, which can accurately simulate the timing and energy information of NVM systems. We have implemented the MORE^2 prototype, including morphable encryption and encoding modules in NVMain. TABLE II lists the system configurations. We set the AES-128 logic encryption energy/latency as 50 *ns*/3.9 *nJ* [25], [27]. Besides, we assume the metadata cache is big enough (e.g. 512 KB) for nearly 100% hit rate [7]. As for the benchmarks, we choose 11 benchmarks from SPEC CPU2006 suite [19]. We run eight copies of each benchmark in the CPU, and the applications are warmed up with small instructions then run overall 4 billion instructions. We also run 5 multi-threaded benchmarks from PARSEC 2.1 [20] with the *medium* input set, while the thread count is 8. All the schemes use Data-Comparison-Write (DCW) [28] to eliminate the redundant bit flips. The several comparison schemes are listed as follows:

- DCW: Memory line data are encrypted first, then encrypted data are updated by DCW.
- DEUCE: DEUCE can reduce the writes to words that are always clean in a write epoch. In this paper, DEUCE equips 32-bit tracking bits for each memory line.
- SECRET: Each 8-byte word has a 3-bit local counter and 1-bit zero word flag.
- DEUCE + BDI: Incoming data are compressed by BDI, then compressed data are encrypted with DEUCE.
- SECRET + BDI: Incoming data are compressed by BDI, then compressed data are encrypted with SECRET.
- MORE^2: This is our scheme with all optimizations.
- No-Encryption: Memory line data are updated by DCW without encryption.

TABLE II: System Configurations.

Cores	8-Core, 3.2GHz
L1 I/D cache	private, 32KB, 2-way, 2-cycle , LRU
L2 Cache	private, 256KB, 8-way, 20-cycle , LRU
L3 Cache	shared, 8MB, 16-way, 50-cycle , LRU
Encryption logic	50 *ns*, 3.9 *nJ* for AES-128 logic
Memory Organization	8GB PCM, 64B memory line, 400 MHz tRCD/tCL/tCWD/tFAW/tWTR/tWR = 60/15/13/50/7.5/300 *ns* Ereset, Eset = 0.054, 0.102 *nJ*

A. Bit flips

Fig. 11 shows the bit flips ratio. The bit flips of encrypted NVM include the memory line data and its metadata. As for the DCW scheme, the bit flips ratio is 50% due to the diffusion property of encryption. DEUCE can reduce the encryption to clean words, leading to a 23.6% bit flips ratio. Compared with DEUCE, SECRET can reduce the encryption to zero words and clean lines, reducing the bit flips ratio to 19.7%. As for `DEUCE+BDI`, `SECRET+BDI`, which combines compression and encryption techniques, the bit flips reduction is not significant compared with DEUCE, SECRET, respectively. This reason is that compression leads to many modified words, diminishing the role of encryption techniques. As a result, they bring to 22.5%, 18.8% bit flips ratio, respectively. For our scheme MORE^2, it reduces the bit flips from 50% to 8.8%. MORE^2 works well mainly due to the morphable encryption and encoding techniques: (1) Morphable encryption reduces the full-line encryptions and removes the encryptions to clean lines; (2) The compression scheme doesn't destroy the clean words, and it largely reduces the size of modified words as well. In particular, the result of MORE^2 is even better than NoEncryption, while NoEncryption reduces the bit flips ratio to 14.8%. Compared with SECRET+BDI, MORE^2 reduces bit flips by 53.5%. The reduced bit flips can improve the lifetime of encrypted NVM [5].

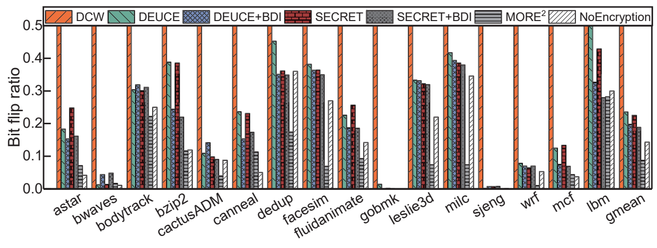


Fig. 11: The normalized bit flips ratio of different schemes.

B. Access Latency

Fig. 12 illustrates the average access latency of several schemes. DEUCE, SECRET, DEUCE+BDI, SECRET+BDI, MORE^2 can reduce the written data size, leading to shorter access latency. For some benchmarks, e.g. *bwaves*, *sjeng*, *gobmk*, these techniques can reduce largely existing clean words, resulting in significant latency optimization. Compared with SECRET+BDI, MORE^2 can reduce the access latency by 27.3%. The good effectiveness are contributed from two aspects: (1) Prediction-based encryption mechanism can reduce the encryption latency from 50 *ns* to only 2.6 *ns*; (2) Morphable encryption techniques can reduce the writes to clean words, and morphable selective encoding scheme can reduce the writes to modified words. These two aspects of write reduction cooperates to reduce the access latency. In particular, MORE^2 can gain more latency reduction than NoEncryption. TABLE III lists the detailed write and read latency of several scheme. MORE^2 shortens the write latency by decreasing data size and removing encryption latency. While read latency is decreased by reducing the request waiting time in the memory queue. Compared with SECRET+BDI, MORE^2 reduces the read/write latency by 32.9%/36.7%.

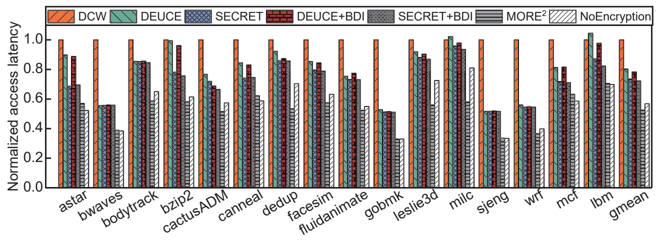


Fig. 12: The normalized access latency of different schemes.

C. Performance

The performance comparisons of several schemes are shown in Fig. 13. Due to the shortened access latency, the system Instruction Per Cycle (IPC) and memory bandwidth can be improved. TABLE III lists the bandwidth comparisons. Compared with SECRET+BDI, MORE^2 can improve memory bandwidth by 22.8%. Fig. 13 shows the normalized IPC. In some benchmarks, e.g. *bwaves, gobmk, sjeng*, the IPC improvements are significant due to large reduced access latency. Besides, some benchmarks e.g. *bodytrack, bzip2, cactusADM, fluidanimate* are not sensitive to access latency, the IPC results are not obvious. Compared with the baseline, DEUCE, SECRET, DEUCE+BDI, SECRET+BDI, MORE^2, NoEncryption can improve IPC by 14.3%, 17.7%, 14.7%, 18.3%, 32.6%, 31.3%, respectively. While MORE^2 can obtain better performance than NoEncryption due to smaller access latency. Compared with SECRET+BDI, MORE^2 can improve IPC by 12.1%.

TABLE III: Normalized Read/Write Latency and Bandwidth.

	Read Latency	Write Latency	Bandwidth
DCW	1.0	1.0	1.0
DEUCE	0.77	0.81	1.19
SECRET	0.72	0.74	1.25
DEUCE+BDI	0.76	0.79	1.20
SECRET+BDI	0.70	0.73	1.27
MORE^2	**0.51**	**0.50**	**1.56**
NoEncryption	0.55	0.59	1.49

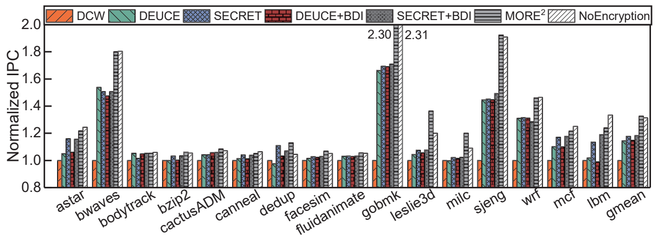


Fig. 13: The normalized IPC of several schemes.

D. Write Energy

The write energy includes encryption energy and NVM write energy. The comparison results are shown in Fig. 14. Compared with the baseline, DEUCE, SECRET, DEUCE+BDI, SECRET+BDI, MORE^2, NoEncryption can reduce write energy by 37.2%, 45.1%, 38.5%, 47.5%, 62.8%, 81.7%. MORE^2 can largely reduce the bit flips, therefore, the write energy can be greatly decreased. While the write energy of NoEncryption is much lower than MORE^2, this is because MORE^2 has extra encryption energy. For DEUCE and SECRET, they are encrypted by the serial way with several AES-128 logics, leading to lower encryption energy than MORE^2, therefore, in some benchmarks e.g. *astar*, *canneal*, *mcf*, the results of MORE^2 are higher than SECRET+BDI. Compared with SECRET+BDI, MORE^2 reduces the overall write energy by 29.1%.

VI. Conclusion

Encrypted secure NVM suffers from severe performance and endurance degradation. Current encryption techniques aim to reduce the re-encryption of clean words, which however suffers from high performance overheads and many unnecessary encryptions. In the meantime, compression techniques can reduce the writes of encrypted NVM. But, we find that

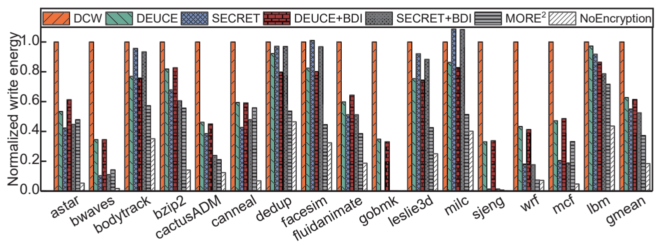


Fig. 14: The normalized write energy of different schemes.

they may destroy the data patterns and increase the modified words, resulting in many encryptions in secure NVM.

In this paper, we propose the MORphable Encryption and Encoding (MORE^2) scheme to address these problems. Our MORphable Encryption (MORE) technique aims to reduce the full-line re-encryption and avoid clean line encryption. Besides, MORE proposes a prediction-based write scheme to avoid the encryption of clean lines, and pre-encrypt the dirty lines. Therefore, MORE can remove the encryption from the critical path of NVM. Furthermore, MORE^2 proposes the Morphable Selective Encoding (MSE) scheme to compress the dirty words while preserving clean words. MORE^2 encrypts all metadata with the line counter to guarantee high security. Experimental results show that MORE^2 displays great bit flips, energy reduction, and large performance improvement compared with the state-of-the-art design.

Acknowledgment

This work was sponsored by the National Natural Science Foundation of China under Grant 61832007, Grant 61821003, Grant 61772222, and Grant U1705261, in part by the Fundamental Research Funds for the Central Universities No.2019kfyXMBZ037, and National Science and Technology Major Project No. 2017ZX01032-101, and Zhejiang Lab (NO.2020AA3AB07).

References

[1] C. Wang, D. Feng, W. Tong, J. Liu, Z. Li, J. Chang, Y. Zhang, B. Wu, J. Xu, W. Zhao *et al.*, "Cross-point resistive memory: Nonideal properties and solutions," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 4, pp. 1–37, 2019.

[2] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ACM SIGARCH Computer Architecture News*, 2009.

[3] W. Zhao, W. Tong, D. Feng, J. Liu, J. Xu, X. Wei, B. Wu, C. Wang, W. Zhu, and B. Liu, "Oswrite: Improving the lifetime of mlc stt-ram with one-step write," in *Proceedings of the 36th International Conference on Massive Storage Systems and Technology (MSST)*, 2020.

[4] W. Zhao, W. Tong, D. Feng, J. Liu, Z. Chen, J. Xu, B. Wu, C. Wang, and B. Liu, "Improving the energy efficiency of stt-mram based approximate cache," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1104–1109.

[5] V. Young, P. J. Nair, and M. K. Qureshi, "Deuce: Write-efficient encryption for non-volatile memories," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.

[6] A. Awad, P. Manadhata, S. Haber, Y. Solihin, and W. Horne, "Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016.

[7] P. Zuo, Y. Hua, M. Zhao, W. Zhou, and Y. Guo, "Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes," in *Proceedings of the 51st IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018.

[8] S. Swami, J. Rakshit, and K. Mohanram, "Secret: Smartly encrypted energy efficient non-volatile memories," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016.

[9] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009.

[10] J. Yue and Y. Zhu, "Accelerating write by exploiting pcm asymmetries," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 282–293.

[11] J. Boukhobza, S. Rubini, R. Chen, and Z. Shao, "Emerging nvm: A survey on architectural integration and research challenges," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 2, pp. 1–32, 2017.

[12] J. Kong and H. Zhou, "Improving privacy and lifetime of pcm-based main memory," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, 2010.

[13] P. M. Palangappa and K. Mohanram, "Castle: Compression architecture for secure low latency, low energy, high endurance nvms," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018.

[14] M. Arjomand, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das, "Boosting access parallelism to pcm-based main memory," in *ACM SIGARCH Computer Architecture News*, 2016.

[15] 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 *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, 2012.

[16] A. R. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," *Dept. Comp. Scie., Univ. Wisconsin-Madison, Tech. Rep*, vol. 1500, 2004.

[17] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[18] M. Poremba, T. Zhang, and Y. Xie, "Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, 2015.

[19] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[20] C. Bienia and K. Li, *Benchmarking modern multiprocessors*. Princeton University Princeton, NJ, 2011.

[21] G. E. Suh, D. Clarke, B. Gassend, M. v. Dijk, and S. Devadas, "Efficient memory integrity verification and encryption for secure processors," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, 2003.

[22] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," in *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, 2006.

[23] A. Kolli, J. Rosen, S. Diestelhorst, A. Saidi, S. Pelley, S. Liu, P. M. Chen, and T. F. Wenisch, "Delegated persist ordering," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[24] S. Liu, K. Seemakhupt, G. Pekhimenko, A. Kolli, and S. Khan, "Janus: Optimizing memory and storage support for non-volatile memory systems," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 143–156.

[25] R. Ueno, S. Morioka, N. Homma, and T. Aoki, "A high throughput/gate aes hardware architecture by compressing encryption and decryption datapaths," in *International conference on cryptographic hardware and embedded systems*. Springer, 2016, pp. 538–558.

[26] X. Wei, D. Feng, W. Tong, J. Liu, and L. Ye, "Morlog: Morphable hardware logging for atomic persistence in non-volatile main memory," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 610–623.

[27] S. Mathew, S. Satpathy, V. Suresh, M. Anders, H. Kaul, A. Agarwal, S. Hsu, G. Chen, and R. Krishnamurthy, "340 mv–1.1 v, 289 gbps/w, 2090-gate nanoaes hardware accelerator with area-optimized encrypt/decrypt gf (2 4) 2 polynomials in 22 nm tri-gate cmos," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 4, pp. 1048–1058, 2015.

[28] 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 *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, 2007.