Securing Emerging Nonvolatile Main Memory With Fast and Energy-Efficient AES In-Memory Implementation

Mimi Xie, *Student Member, IEEE*, Shuangchen Li, *Student Member, IEEE*, Alvin Oliver Glova, *Student Member, IEEE*, Jingtong Hu, *Member, IEEE*, and Yuan Xie, *Fellow, IEEE*

***Abstract*— As CMOS technology approaches its scaling limit, emerging nonvolatile memory (NVM) technologies become promising alternatives to DRAM due to their low leakage power and better scalability. However, the nonvolatile main memory system suffers from a new security vulnerability. An attacker can readily access sensitive information on the memory, since the nonvolatility allows information to be retained for a long time even after the power is OFF. While real-time memory encryption during memory accesses with dedicated Advanced Encryption Standard (AES) engine is an effective solution for this vulnerability, it incurs runtime performance and energy overhead. Alternatively, in this paper, we propose a fast and efficient AES in-memory (AIM) implementation, to encrypt the whole/part of the memory only when it is necessary. Rather than adding extra processing elements to the cost-sensitive memory, we take advantage of NVM's intrinsic logic operation capability to implement the AES algorithm. We leverage the benefits (large internal bandwidth and dramatic data movement reduction) offered by the in-memory computing architecture to address the challenges of the bandwidth intensive encryption application. Embracing the massive parallelism inside the memory, AIM outperforms existing mechanisms with higher throughput yet lower energy consumption. The experimental results show that compared with state-of-the-art AES engine running at 2.1 GHz, AIM speeds up the encryption process by 80× for a 1-GB NVM.**

***Index Terms*— Advanced Encryption Standard (AES), encryption, main memory, nonvolatile.**

I. Introduction

DRAM has been employed as the main memory for computers for decades. However, as technology scales down, DRAM will suffer from prohibitively high leakage power. Consequently, researchers are actively developing promising candidates such as phase change memory (PCM) [35], resistive random access memory [30], and spin-transfer torque magnetic random access memory (MRAM) [27] to be deployed as next-generation nonvolatile main memory (NVMM). These nonvolatile memories (NVMs) have several significant advantages over traditional DRAM main memory. They provide promising features such as nonvolatility, high density, low leakage power, and high scalability. The nature of nonvolatility avoids the need of a frequent refresh for DRAM and allows the data in NVM to be retained for a long time after power is OFF. Intel's recent announcement of 3-D Xpoint [2] and the JEDEC's NVDIMM-P specification [1] are the latest efforts toward the goal of next-generation NVMM.

In spite of these advantages, NVMM suffers from a new security vulnerability. Since the information in NVMM will not lose data after the power is turned OFF, an attacker with physical access to the system can readily scan the main memory content and extract all valuable information from the main memory [7], [12]. In contrast, the security of DRAM memory relies on its short retention time which varies from 500 ms to 50 s [28]. To protect the data of the NVMM, the whole memory should be provided with a security mechanism with comparable security level to DRAM.

Real-time memory encryption with pad-based or stream cipher is an effective solution for this vulnerability, in which every cache line is encrypted or decrypted before being written to or read from the main memory [15]. The real-time memory encryption is a strong protection, and it can also prevent other attacks such as memory bus snooping [9]. Unfortunately, the strong protection is at the expense of runtime performance loss, since the decryption latency (as an overhead of read access) is on the critical path. In addition, encrypting and decrypting every memory access also result in severe energy overhead.

Strong real-time protection is, however, not always necessary. For example, when a mobile device (e.g., smart phone or laptop) is being used, the attack that requires physical access to the NVMM can rarely happen. Only when the device is shut down or put into sleep/screenlock mode, the memory encryption is required. i-NVMM [7] further proposes to

Manuscript received March 25, 2018; revised June 21, 2018; accepted July 28, 2018. Date of publication September 10, 2018; date of current version October 23, 2018. The work of M. Xie and J. Hu was supported by NSF under Grant CNS-1830891 and Grant CCF-1820537. The work of S. Li, A. O. Glova, and Y. Xie was supported in part by NSF under Grant 1730309/1719160/1500848, in part by CRISP, one of the six centers in JUMP, and in part by DARPA through the Semiconductor Research Corporation Program. *(Corresponding author: Mimi Xie.)*

M. Xie and J. Hu are with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261 USA (e-mail: mm.xie@pitt.edu; jthu@pitt.edu).

S. Li, A. O. Glova, and Y. Xie are with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106 USA (e-mail: shuangchenli@ece.ucsb.edu; aomglova@ece.ucsb.edu; yuanxie@ece.ucsb.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2865133

encrypt the main memory incrementally while maintaining an unencrypted working set which needs fast bulk encryption when necessary. Instead of the real-time encryption with performance loss and energy cost for every cache line, encrypting the working set in bulk or the whole memory when necessary is preferred in such mobile scenarios where strong protection for this part of the memory is not required all the time.

Even though the bulk memory encryption approach results in zero performance loss at runtime, reduces the encryption tasks, and hence the energy consumption, two challenges still persist: first, it should be fast in order to lower the vulnerability window when locked and provide an instant response when unlocked. This is even more critical under the development of multicore processor and increasing demand of much larger main memory. Second, it requires energy-efficient encryption considering the limited battery life.

To address these two challenges, we propose AES In-Memory (AIM), a novel AES in-memory encryption architecture for fast and energy efficient NVMM encryption. Embracing the benefit of the processing-in-memory (PIM) architecture, AIM takes advantage of large internal memory bandwidth, vast bitline-level parallelism, and low *in situ* computing latency, eliminating data movement between memory and host. Leveraging the nondestructive read in NVMs for performing efficient XOR operations, we can perform the entire Advanced Encryption Standard (AES) procedure in-place by adding lightweight logic gates to the memory peripheral circuitry. Specifically, we explore three levels (chip, bank, and subarray) of parallelism to provide different design choices under different performance and energy efficiency requirements. We also propose a new combined cipher mode for AIM in order to maintain high parallelism with the best performance and reduced area overhead.

The remainder of this paper is organized as follows. Section II describes the background on NVMM organization and AES encryption. Section III describes the motivation of this paper. Section IV presents the complete in-memory encryption architecture. Sections V and VI discuss the proposed encryption mode and key storage, respectively. Detailed experimental evaluation is provided in Section VII. Section VIII presents the related works. Finally, Section IX concludes this paper.

II. Background

A. Nonvolatile Main Memory

Main memory is logically organized as a hierarchy of channels, ranks, and banks. Channels work in parallel and share the same physical link to the processor. Each channel contains several ranks and each rank has several physical chips. A physical chip has several banks which contend for the same I/O in the same channel. Banks from different channels can be accessed completely independently of each other. A memory bank has several subarrays which share the global data line and global row buffer. Each subarray is a 2-D array of memory cells, which has a local row buffer. A subarray can be further divided into different mats, which has its private row buffers and write driver. Row and column addresses are often decoded at mat level. Peripheral circuitry, such as sense amplifiers (SAs) and write drivers are shared among several columns.

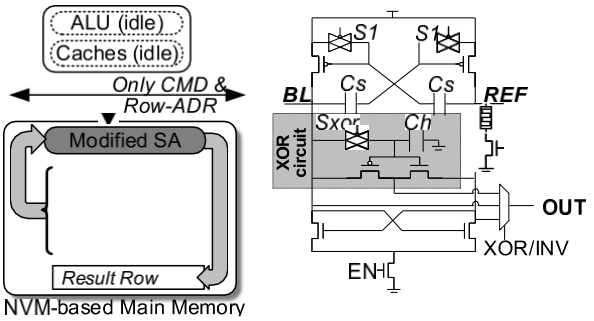


Fig. 1. Pinatubo's architecture computes vector bitwise operations inside NVMs (left). SA modification in Pinatubo to perform in-memory XOR operations [18] (right).

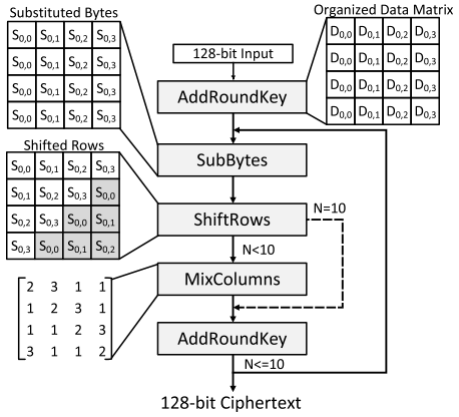


Fig. 2. AES flowchart.

B. Pinatubo: PIM in NVM

In addition to the recent research that leverages 3-D-stacking DRAM such as a hybrid memory cube to support PIM architecture, Pinatubo [18] paves another way that leverages the emerging NVM to support PIM while incurring negligible area overheads. As shown in Fig. 1, Pinatubo modifies the SA circuit of the normal emerging NVMM, so that the SA not only can serve for reading but also carry out bitwise operations such as AND, OR, and XOR. Instead of activating one row and reading the data out, Pinatubo activates two rows at once which correspond to the two operand vectors. The output of the SA is then the result of the bitwise operation of these two rows (vectors). To perform an XOR operation, Pinatubo first opens the one operand row, and stores the data in the capacitor inside the modified SA. Then, it opens the second operand row, and the data from this row and the previous data in the capacitor go through the simple XOR circuit inside the modified SA, after which, the readout result of this SA is the XOR result of these two rows. The AIM design takes advantage of this fast in-memory XOR operation offered by Pinatubo [18].

C. Advanced Encryption Standard

The AES [8] is a symmetric block cipher which consists of four transformations as shown in Fig. 2. SubBytes is

a nonlinear invertible byte substitution that replaces each byte of the state matrix using a substitution table (S-box). Each byte $D_{i,\ j}$ in the state matrix is replaced with a new byte $S_{i,\ j}$ in SubBytes step. ShiftRows cyclically shifts all bytes in each row by different offsets. The first row is unchanged; each byte in the i th row is cyclically shifted left by i bytes, respectively. MixColumns combines the 4 B of each column of the state matrix using an invertible linear transformation. This transformation can be written as a matrix multiplication in the finite field of GF(2^8) where the state matrix is multiplied by a constant matrix composed of 1, 2, and 3. AddRoundKey combines the state matrix with the round key by bitwise XOR operations. Each byte in the state matrix is XORed with a byte in the same row and column of the key matrix. These round keys are generated from the key with a key schedule which expands a short key into a number of separate round keys.

The four transformations of AES are comprised of XOR, shift, and LUT operations. The intermediate results after each transformation are maintained as a state matrix of bytes. At the start of the algorithm, a round key is added to the input by a bitwise XOR operation. Then, the state array is transformed by implementing four basic transformations 10 times when the key has 128 b, while the last round does not include MixColumns.

III. Motivation and Overview

A. NVMM's Vulnerability Challenge

We take the case of smartphones as a motivating example. We assume NVMM has been adapted as the replacement of DRAM, due to its advantages of low leakage and high density. The vulnerability challenge emerges that the content in the memory is under risk if the attacker stoles the device. Even though the device is locked, the attacker can remove the memory, plug it in another machine, and read it. The threat is more severe in the case of NVMM, since the retention time of NVM cells is typically much longer (a few years [24]) compared with 500 ms to 50 s [28] in the case of DRAM. An effective solution is real-time memory encryption with Pad-based or Stream cipher encryption, however, at an expense of performance degradation and also energy overhead (4% reported by previous work [33]). Instead of the real-time encryption, a smarter approach is to encrypt the memory only when necessary. For example, when the device is being used (unlocked), the attacker can rarely take it away. Only when the device is turned OFF or put into sleep/screenlock mode should the bulk memory encryption be committed.

B. PIM: A Potential Solution

To address those challenges, we propose a PIM architecture for memory encryption. The PIM offers the benefit of high internal memory bandwidth, massive parallelism (chip, bank, and subarray level), and most importantly, it eliminates the data movement between the memory and processors. Meanwhile, we observe that in the one-time memory encryption application, the memory bandwidth is a bottleneck since a dedicated AES encryption engine (EE) provides a much larger throughput (53 Gbps [22]) than the DDR throughput.

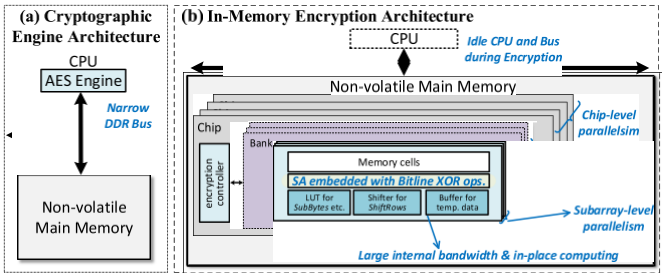


Fig. 3. Memory encryption architecture. (a) Traditional encryption approach implemented a cryptographic engine outside main memory. (b) Proposed AIM design: in-memory computing with NVM's intrinsic features.

Moreover, the energy for the fetching data from memory with the DDR bus is also dominant. It is shown that 91.6% energy is spent on fetching and writing this data from the experimental results. Considering both advantages offered by PIM and the workload characteristics of the target one-time memory encryption application, we believe the PIM can effectively address NVMM's vulnerability challenge.

C. Design Overview

Based on the above-mentioned observations, we propose AIM, an in-memory encryption mechanism for NVMM, as shown in Fig. 3. Different from the coprocessor AES engine [Fig. 3(a)], the proposed AIM avoids the narrow DDR bus and embraces the large intramemory bandwidth. It also benefits from multiple memory blocks parallel encryption by leveraging the flexible parallelism inside the memory, marked as chip-level, bank-level, and subarray-level parallelism in the figure. To perform the AES algorithm, we build all its required arithmetics (i.e., XOR, Shift, and LUT) inside each memory subarray. Instead of implementing all those operations with logic gates, we take the advantage of NVM's unique feature and implement the most time-consuming operation, XOR, within the SAs themselves, as described in Section II-B [18]. The data buffer is added to store intermediate results, reducing expansive write operations to NVM cells. In addition, an encryption controller is implemented in each chip to provide control signals to direct the encryption process. In Section IV, the details of the hardware implementation and how the AES algorithm is mapped to the proposed AIM are described.

IV. AES In-Memory Implementation

A. Data Organization

AES in-memory implementation takes advantage of different levels of parallelism in the NVMM. In this paper, in-memory encryption is performed directly on the mem- ory cells. The data in memory cells are read out with SAs, each of which is shared by several adjacent columns with a MUX as shown in Fig. 4. Since the unit data matrix to be encrypted needs to be organized in a certain fashion to facilitate the encryption process, we distribute the 8 b of each element in the data matrix into different mats and different columns in the same mat so that they can be used concurrently. In this way, the plaintext data block does not

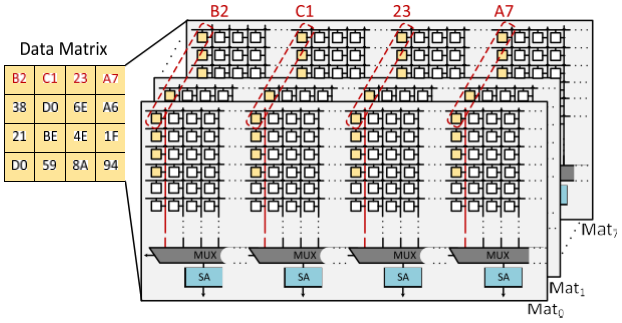


Fig. 4. Distributed data organization for AES encryption.

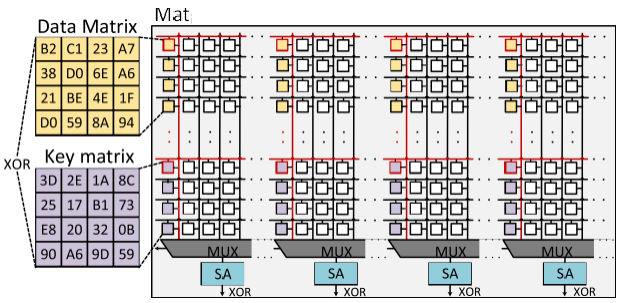


Fig. 5. Addroundkey stage with $_{XOR}$ operation.

have to be pretransformed into matrix form before encryption starts. For illustration purposes, we assume that there are M mats Mat_i $(i = 0, 1, \ldots, M)$, the size of each mat is $N * N$, and K columns share one SA.

Fig. 4 illustrates the memory distribution for one data matrix. In the AES algorithm, the basic processing unit is 1 B of the data matrix; therefore, the data matrix is distributed into eight mats so that each mat has a 1-b level of the data matrix. In order to encrypt each row of the data matrix in parallel, four columns of the data matrix are distributed to different columns of memory array connecting four adjacent SAs separately. These four columns of the memory array are of the same local column address. In this way, when one row of the subarray is activated and a local column address is selected for each MUX, every four adjacent SAs will sense out a row of the data matrix. In total, every four rows of the subarray contain $MN/4K$ data blocks of 128 b.

To enable further processing of the data matrix, the intermediate results, which are the state matrices, need to be buffered. To avoid extra hardware overhead and simplify the circuity, we write the intermediate results back to the data matrix. In the proposed encryption mechanism, AES encryption generates less than 60 writes during 10 rounds of encryption to each cell in the encrypted memory block. This amount of writes has a negligible impact on the endurance of memory. NVMM encryption is performed before the system is powered down. Therefore, we assume that the main memory is encrypted 20 times every day. We also conservatively assume that the deployed NVM has an endurance of 10^9 cycles. In 5 years, AES encryption will generate $60 * 20 * 356 * 5 = 1.1 * 10^5$ writes which is less than 0.2% of its total life cycles.

B. AddRoundKey

In this stage, the data matrix is combined with the key matrix. Each byte of the data matrix is combined with the corresponding subkey of the key matrix using bitwise XOR operation. AddRoundKey is implemented with the modified SA design of Pinatubo [18], which realizes bitwise XOR operation with two microsteps inside SA.

Fig. 5 shows the process of AddRoundKey transformation for one row of the data. First, the first row of the data in the data matrix is read into the added capacitor in each SA by activating the first wordline in red color and selecting a column with MUX. Second, the first row of the data in the key matrix is read into the latch in each SA by activating the second wordline in red color and selecting a column with MUX. After these two steps, the bitwise XOR result of the first row is latched in each SA. Suppose it takes t_{XOR} to complete XOR operation for one row of data matrix, it takes $4t_{XOR}$ to complete AddRoundKey transformation for a data matrix since there are four rows in each data matrix. This AddRoundKey transformation is parallelized because of multiple SAs. Since there are M mats and N/K SAs in each mat, $(N/4K)(M/8)$ data matrices are transformed simultaneously. In our design, after AddRoundKey transformation for one row of data, SubBytes is performed immediately for this row of the data instead of continuing performing AddRoundKey for all the four rows.

The initial AddRoundKey stage is performed with the initial key. The other 10 rounds of AddRoundKey are performed with the corresponding round key. As shown in Fig. 5, the encryption key is maintained in the NVM array and round keys overwrite the encryption key after finishing each round of encryption.

C. SubBytes

In this step, each byte of the data matrix is replaced with a new byte by doing the nonlinear transformation. This transformation is realized with S-box which is used to obscure the relationship between the key and the ciphertext. The S-box can be realized with LUT by implementing combinational logic which has 8-b input and 8-b output or ROM which has 16 rows and 16 columns while each entry is a byte. In this paper, S-box is realized with combinational logic since it incurs lower overhead.

After the AddRoundKey stage of one row of state matrix, the intermediate results are latched in the SAs. For SubBytes transformation, each byte of the data matrix is decoded from eight mats and input to the S-box as shown in Fig. 6. In this figure, the *Add Roundkey* results of the second row of the data matrix are latched in the SAs. SubBytes is performing on the second byte $C7$. The output of S-box is the substituted byte $C6$. In this figure, there is one S-box combinational logic which has 8-b input and 8-b output. Since we can only input 1 B each time to the S-box, the SubBytes transformation can only be done sequentially which takes a long time. To accelerate the SubBytes transformation, we can add more S-box combinational logics to enable parallel SubBytes performing. At the same time, we need to consider the hardware overhead

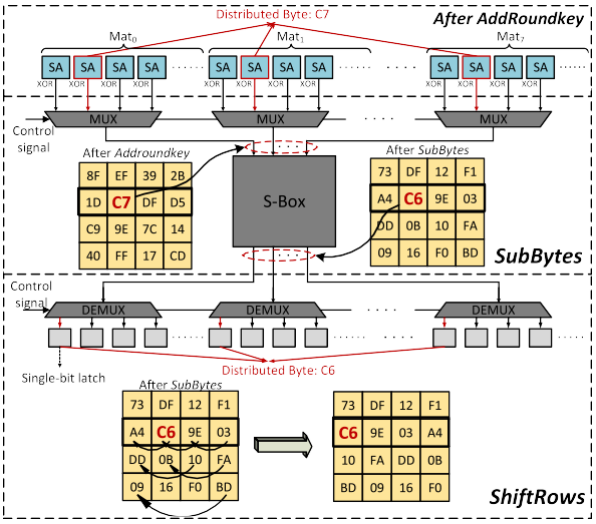


Fig. 6. SubBytes transformation with LUT and ShiftRows transformation with addressing logic.

introduced by multiple S-boxes. We have different designs in terms of S-box considering both encryption speed and overhead. The experimental section will show the performance comparison of different designs.

After we obtain the 8-b output of S-box, it will not be immediately written back. Instead, the next stage, ShifRow, will be performed on the output.

D. ShiftRows

In this step, the bytes in each row of the data matrix are cyclically shifted by a certain offset. Specifically, while the top row remains unchanged, each bit in the second row of the bit-level data matrix is cyclically shifted left by 1 b, each bit in the third row of the bit-level data matrix is cyclically shifted left by 2 b, and each bit in the third row of the bit-level data matrix is cyclically shifted left by 3 b (right by 1 b).

The ShiftRows transformation is realized with control signal and address decoding, as shown in Fig. 6 (bottom). Originally, the 8-b output of S-box needs to be written back where each input bit is located. This process needs address decoding to write to the right position. The ShiftRows transformation can leverage this address decoding process to do shifting by address decoding. By combining an offset with the column address, the output of S-box is shifted to another address according to the ShiftRows algorithm. In Fig. 6, the second byte *C*6 in the second row of state matrix after SubBytes needs to be shifted to the left by 1 B. This means each bit needs to shifted left by 1 b according to the data matrix distribution in the memory. This shifting process is down by selecting the first column with the control signal.

After ShiftRows transformation, each bit will be buffered in the single-bit latch until SubBytes and ShiftRows are performed on all the data in the SAs. Then, the values in the row buffer are transmitted to the write driver and written back to the memory array. This row buffer gathers the intermediate results of one row and avoids writing to the NVM row multiple times.

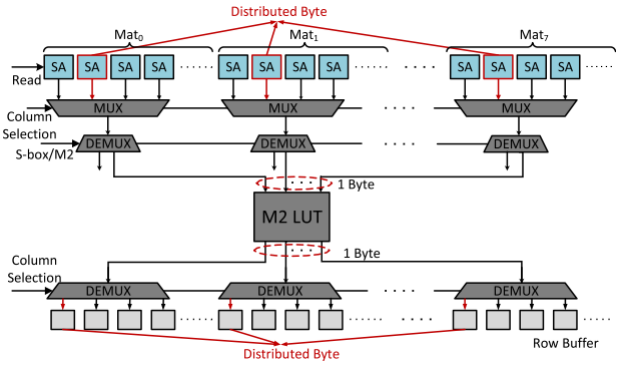


Fig. 7. MixColumn substep: M-2 LUT.

E. MixColumns

In MixColumns stage, the 4 B of each column of the data matrix are combined together using an invertible linear transformation to provide diffusion in the cipher. The MixColumns transformation multiplies the data matrix by a known matrix as shown in Fig. 2.

Matrix multiplication is done in finite field GF(2^8). $S_{i,j}$ and $S'_{i,j}$ are used to indicate the byte in row i, column j of the state matrix and the transformed state matrix, respectively. The MixColumns transformation can be decomposed to modular multiplication and XOR operations as follows:

$$\begin{aligned} S'_{0,j} &= 2 \cdot S_{0,j} \oplus 3 \cdot S_{1,j} \oplus S_{2,j} \oplus S_{3,j} \\ S'_{1,j} &= S_{0,j} \oplus 2 \cdot S_{1,j} \oplus 3 \cdot S_{2,j} \oplus S_{3,j} \\ S'_{2,j} &= S_{0,j} \oplus S_{1,j} \oplus 2 \cdot S_{2,j} \oplus 3 \cdot S_{3,j} \\ S'_{3,j} &= 3 \cdot S_{0,j} \oplus S_{1,j} \oplus S_{2,j} \oplus 2 \cdot S_{3,j}. \end{aligned} \tag{1}$$

Multiplication-by-2 (M-2) in the finite field can be realized by shifting each bit of the operand left by 1 b, followed by a XOR operation with 0×1 B if the most significant bit is 1. A more efficient way is leveraging LUT. M-3 in the finite field GF(2^8) of MixColumn can be realized with M-2 and XOR logic. This is because

$$3 \cdot S_{i,j} = 2 \cdot S_{i,j} \oplus S_{i,j}. \tag{2}$$

Therefore, MixColumns stage is decomposed into M-2 LUT and XOR operations. MixColumns needs several sub steps and generates several intermediate values. To both accelerate this transformation and maintain a low hardware overhead, we leverage the vacant NVM rows as buffer rows for intermediate results. The MixColumns stage is realized with LUT and XORs as follows:

$$\begin{aligned} S'_{0,j} &= T_j \oplus 2 \cdot S_{0,j} \oplus 2 \cdot S_{1,j} \oplus S_{0,j} \\ S'_{1,j} &= T_j \oplus 2 \cdot S_{1,j} \oplus 2 \cdot S_{2,j} \oplus S_{1,j} \\ S'_{2,j} &= T_j \oplus 2 \cdot S_{2,j} \oplus 2 \cdot S_{3,j} \oplus S_{2,j} \\ S'_{3,j} &= T_j \oplus 2 \cdot S_{0,j} \oplus 2 \cdot S_{3,j} \oplus S_{3,j} \end{aligned} \tag{3}$$

where

$$T_j = S_{0,j} \oplus S_{1,j} \oplus S_{2,j} \oplus S_{3,j}. \tag{4}$$

The first step of MixColumns is M-2 transformation with LUT. This process shares the same address decoding logic of S-box with a MUX as shown in Fig. 7. Since we can

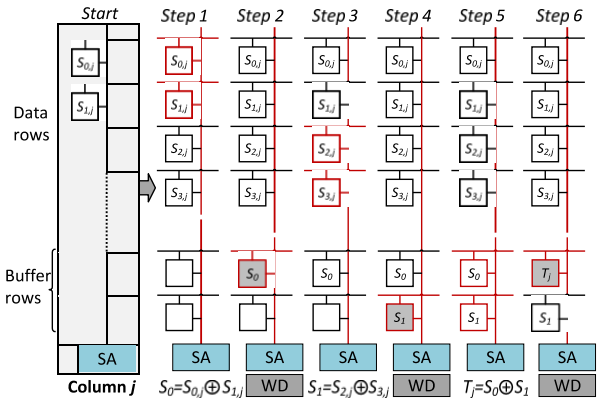


Fig. 8. Example of MixColumns substep: calculate T_j for each column (4).

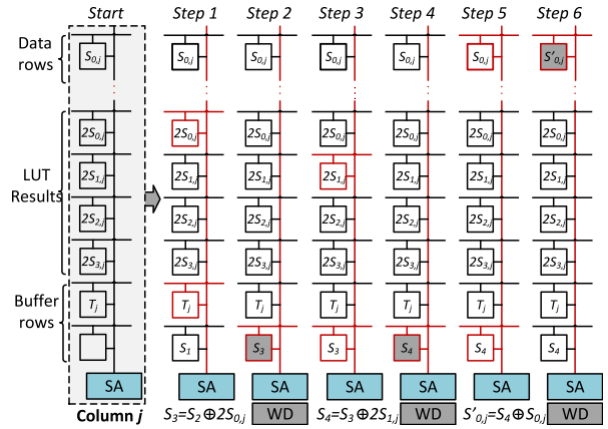


Fig. 9. Example of MixColumns substep: calculate $S'_{0,j}$.

only input 1 B each time to the LUT, this transformation can only be done sequentially, which takes a long time. To accelerate this transformation, we add multiple M-2 LUT combinational logics to enable parallel performing. Like S-box design, we have different multiplication-by-2 LUT designs considering both encryption speed and overhead. After M-2 transformation, outputs are latched in a row buffer until all bytes of the activated row finishes M-2 transformation. Then, data in this row buffer is written to a vacant memory row. In total, four empty NVM rows are used for storing LUT results.

The next step of MixColumns is calculating T_j following (4). Fig. 8 shows the detailed process of calculating T_j for a specific column. Every time two rows are activated to get the XOR result of two memory cells, then next step of this result is written to an empty buffer row. From this figure, this step costs three XOR operations and three writes. In this step, since all SAs are working simultaneously, T_j for each column is calculated in full parallel.

The final step of MixColumns is calculating the result of MixColumns transformation following (3). In this step, with M-2 LUT results stored in four rows and T_j values, we can finish the MixColumns transformation for one row of selected columns in six steps as shown in Fig. 9. This figure shows an example of how to calculate $S'_{0,j}$ in row 0. After three times of activating two rows, four operands are XORed together to get the final result of $S'_{0,j}$, and then this result is written back by replacing $S_{0,j}$.

During MixColumns, several writes are generated. In M-2 LUT step, M-2 results are written to four rows, therefore a column of four memory cells takes four writes. In the second step of calculating T_j, three writes are generated as shown in the colored memory cells of Fig. 8. In the final step, transforming one value takes three writes. Thus, 12 writes are generated for transforming four values. In total, 19 writes are generated for each column of four memory cells which means five writes on average are generated for each cell in the MixColumns transformation.

F. Discussion

The AES encryption process leverages innate parallelism of main memory to accelerate encryption. To support in-memory AES encryption, multiple S-box LUTs, M-2 LUT, MUX, and DEMUX are implemented inside the memory system. For decryption, inverse S-box LUT needs to be added except for the available resources for encryption. When the memory system receives an encryption signal, the original key shared among all memory chips is transferred to different memory chips. When each round of encryption finishes, the initial key is expended to get the round key. The encryption controller in each chip takes care of the detailed encryption and the decryption process.

V. Cipher Modes

A. Exploration of Different Cipher Modes

Encrypting two identical plaintext blocks with the same key will generate two identical ciphertext blocks. An attacker would be able to achieve useful information and discover the original plaintext by analyzing the identical blocks of the ciphertext. To allow block ciphers to work with a large number of data blocks, different block cipher modes of operations are devised to blur the ciphertext so that the ciphertext blocks of two identical plaintexts are different. Common modes of block cipher include electronic codebook (ECB), cipher block chaining (CBC), cipher feedback (CFB), output feedback (OFB), and counter (CTR) [5].

1) Electronic Codebook: ECB is the simplest mode that encrypts each data block of the input plaintext separately. Since there is no dependence in encrypting different data blocks, this cipher mode allows different blocks to be encrypted simultaneously and supports high parallelism. However, if there are identical plaintext blocks in the NVMM, encrypting bulk NVMM with the same key is vulnerable.

2) Cipher Block Chaining: In the CBC mode, the next plaintext is always XORed with previously produced ciphertext block before it is encrypted. Since there is no previous ciphertext block, the first plaintext block is XORed with a random initialization vector (IV) which has the same size as a plaintext block. As a result, every subsequent ciphertext block depends on the previous one. Since the CBC mode

encrypts the plaintext sequentially, it will lead to high latency in the AIM encryption process. Different from encryption, decrypting different cipher blocks can be done simultaneously.

3) Cipher Feedback: In the CFB mode, the previous ciphertext block is encrypted and then XORed with the next plaintext block to generate the next ciphertext block. Since there is no previous ciphertext block before the first plaintext block, a random IV is encrypted and then XORed with the first plaintext. Similar to the CBC mode, encryption in CFB mode is performed sequentially while decryption can be performed simultaneously. Therefore, it suffers a similar drawback to the CBC mode. Compared with CBC mode, the CFB mode only uses the encryption of the block cipher. Therefore, the CFB mode gets rid of the required resource for implementing decryption.

4) Output Feedback: The OFB mode creates keystream blocks with the original key and a random IV, which are then XORed with the plaintext blocks to get the ciphertext blocks. Because of the continuous creation of keystream bits, both encryption and decryption are done sequentially. Therefore, this mode has poor parallelism. In addition, the usage of only the encryption of the block cipher gets rid of the required resource for implementing decryption.

5) Counter: The CTR mode creates keystream blocks by encrypting a nonce value added by an increasing counter. The plaintext blocks can be encrypted simultaneously with different counters allowing high-level parallelism. However, CTR mode becomes vulnerable if counters repeat. This mode also gets rid of the required resource for implementing decryption.

In addition, the five cipher modes, Galois/counter mode (GCM) [23] is also a very interesting and powerful cipher mode. However, the implementation of GCM requires adding more circuitry to the memory architecture than other modes because of its authenticity and confidentiality ability. Meanwhile, GCM requires more steps to generate a tag for authenticity and, thus, has much longer encryption time and energy consumption compared with the other cipher modes. Because of the much larger area overhead and the lower performance and energy efficiency, GCM mode has inferior performance to the other discussed cipher modes. Therefore, GCM mode is not considered in this paper.

B. CTR-CFB Encryption

The cipher algorithm requirement and parallelism of encryption direction and decryption direction are summarized in Table I. Among them, CTR mode has the best parallelism based on counters. CFB, OFB, and CTR only need encryption direction implementation which saves hardware resource for implementing decryption. Compared with OFB, CFB has a better parallelism in decryption.

A direct solution to enhancing the security is deploying CTR mode which allows high parallelism. However, CTR mode fails catastrophically when a counter value is reused, because it is a pure XOR stream cipher: XORing two ciphertext blocks that were generated with the same key and counter values cancels out the encryption. Therefore, in this paper, we propose to combine CTR and CFB modes to enhance the AES security while maintaining the parallelism level.

TABLE I
COMPARISON OF DIFFERENT CIPHER MODES

Modes	Cipher requirement		Parallelism	
	Encryption	Decryption	Encryption	Decryption
ECB	✓	✓		✓
CBC	✓	✓		✓
CFB	✓			✓
OFB	✓			
CTR	✓		✓	✓

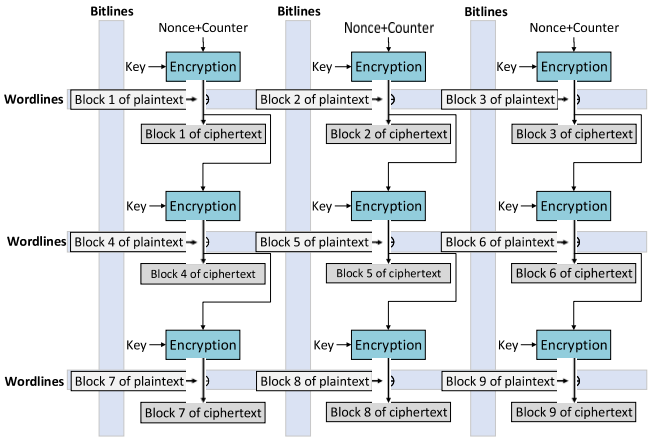


Fig. 10. Encryption of CTR+CFB cipher mode.

The implementation is shown in Fig. 10. In the vertical direction, the CFB mode can be implemented, since each word row needs to be activated for encryption one by one in sequential. In the horizontal direction, CTR mode can be implemented to allow parallelism since several columns can now be encrypted simultaneously. The introduction of the CFB mode to CTR mode avoids the counters for the vertical encryption direction.

The challenges in the implementation lie in generating nonce which is a random number, and the design of counters (different value for different blocks). For generating a nonce, there are two ways: first, the key generator can generate a second key as the nonce and second, the original key can be used to generate the nonce by hashing the original key. To have different counters for different blocks, the bank id, subarray id, mat id, and column id are concatenated together to generate different counters for the first row of plaintext blocks in NVMM as follows:

$$\text{Counter} = \text{Bank}_{\text{id}}||\text{Sub}_{\text{id}}||\text{Mat}_{\text{id}}||\text{Column}_{\text{id}}||\text{Mux}_{\text{id}}||\text{Counter}_{+1} \quad (5)$$

where Counter_{+1} is the increment-by-one counter, which works by incrementing by one after finishing one time of bulk encryption. Therefore, the encryption function for the first row of data blocks is as follows:

$$C = P \oplus \text{En}_{\text{key}}(\text{Nonce} + \text{Counter}). \quad (6)$$

This design of Counter guarantees a unique sequence for each plaintext so that different plaintext blocks are encrypted with different key blocks. Meanwhile, the same counter

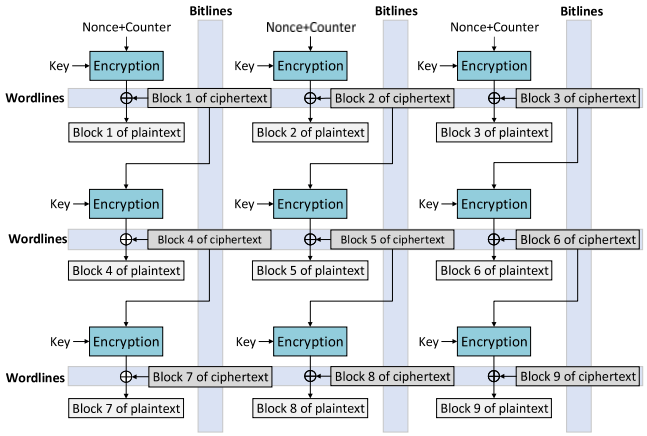


Fig. 11. Decryption of CTR+CFB cipher mode.

sequence will not repeat for a long time so that the same plaintext block will not be encrypted with the same keystream twice for a long time, ensuring the security of the proposed cipher mode.

C. CTR-CFB Decryption

The decryption process of CTR+CFB cipher mode is shown in Fig. 11. From this figure, the decryption of different columns of data blocks is decrypted simultaneously while the data blocks for the same columns are decrypted sequentially. This cipher mode only uses the encryption algorithm of the block cipher as shown in the blue box, thus avoiding the required resource for implementing decryption algorithm especially the inverse S-box.

VI. Key Generation and Storage

The method of key generation, key storage, and key handling significantly influences the security of the crypto-systems. In this section, we will first describe three possible master key generation schemes. Then, we will describe the round key generation in AIM.

A. Master Key Generation and Storage

For the AIM encryption mechanism, there are three possible ways of generating the master key: user input, randomizer, and physical unclonable function (PUF). The user input method allows the user of a device to input a key that the user can remember or a biometric-based key before shutting down or putting the device into sleep/screenlock mode. After inputting a key, this key is then transferred to the NVMM to start the AES in-memory encryption. After finishing bulk encryption, this key is cleared. When the user wants to use the device again, the same key is input to decrypt the NVMM. In this way, there is no key storage overhead or key leakage risk. The second way of creating the original key, randomizer, is to use a pseudorandom number generator to create a one-time random number. Since the device might be powered down, this key should be stored in a protected NVM for decryption. Therefore, this key should be placed far away from the NVMM, such as in the processor, to keep the generated key away from the attackers.

Compared with a randomizer, PUF avoids the need of key storage in NVM. The process of extracting a key from the physical intrinsic properties due to different materials and physical variations from the fabrication process of hardware is described in [21]. PUF-based key generator avoids the need for a pseudorandom number generator by harvesting the hardware unique randomness and processes it into a cryptographic key. Since the randomness is already intrinsically present in the device, there is no need for a protected NVM. Since the randomness is static throughout the lifetime of the device, it can be harvested again to regenerate the same key for decryption. This PUF-based key cannot be found by an attacker who opens up the device because the key is not permanently stored and not present when the device is not active. This way of deriving a key has great security advantages compared to randomizer which needs the key storage in NVM.

B. Round Key Generation—Rijndael Key Schedule

AES requires a separate round key for each round of encryption to achieve a high level of confusion. Expanding the original key into several rounds of keys in AES is known as the Rijndael key schedule. AES key expansion consists of RotWord, SubWord, XOR operations, all of which can be realized with the previously introduced implementations of AES encryption. Therefore, the round keys can also be generated within NVMM instead of using a dedicated key generator. In the proposed CTR+CFB cipher mode implementation, the key for each round is generated only once and stored in the NVMM for each time of memory bulk encryption. After completing the encryption, these round keys are cleared to avoid key information leakage.

VII. Experimental Evaluation

A. Experiment Setup

AIM is evaluated on both MRAM-based and PCM-based main memory with a DDR3 interface and 65-nm technology. The MRAM-based main memory has a 512-b page size. We conservatively assume the MRAM has 256 Mb per chip with a $34F^2$ cell size. The PCM-based main memory's page size is 1024 b, and the capacity is 1 Gb per chip with the cell size of $9F^2$. We modified NVSim [10] and Cacti-3DD [6] to achieve the parameters for the NVM-based main memory. Table II lists the parameters of PCM and MRAM at bit level for main memory implementation [25]. To evaluate the circuitry, we added to support AIM, we synthesize these circuits with Design Compiler with FreePDK.

We compare AIM with three different dedicated memory EEs as follows.

1) EE-1 [13] designs an AES encryption hardware core suited for devices with low power consumption. It has a maximum frequency of 290 MHz, and takes 9.9 nJ and 160 cycles to encrypt a data block.
2) EE-2 [22] implements an AES CMOS application specified integrated circuit encryption core which has a frequency of 2.13 GHz with a total power consumption

TABLE II
PCM AND MRAM PARAMETERS AT BIT LEVEL

Features	PCM	MRAM
Read Latency (ns)	27.17	31.97
Write Latency (ns)	146.39	41.52
Read Energy (pJ)	0.04	0.03
Write Energy (pJ)	0.12	0.06

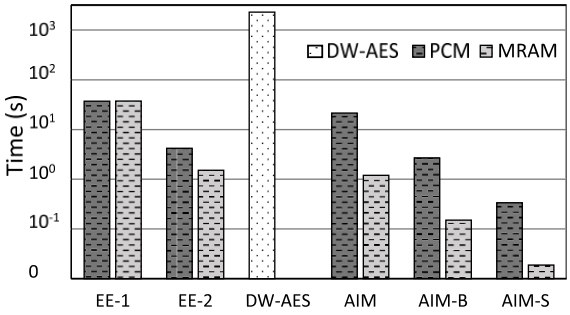


Fig. 12. Comparison of latency between different baselines and different AIM designs.

at 125 mW, and takes five cycles to encrypt four data blocks with an area of 4400 um^2.

3) DW-AES [29] implements an AES encryption core with domain-wall nanowires which has a frequency of 30 MHz and takes 1022 cycles and 2.4 nJ to encrypt one data block.

For the proposed design, we evaluate different configurations described as follows.

4) AIM is the basic configuration, where only one bank works on encryption at one time.
5) AIM-B has the encryption add-on circuit for each bank. To perform a whole memory encryption, all banks in a chip can work in parallel.
6) AIM-S has the add-on circuit for each subarray. By leveraging the subarray-level parallelism [17], multiple subarrays in the same bank work on the encryption/decryption task simultaneously.

B. Performance and Energy Evaluation

1) Latency: Fig. 12 shows the encryption latency of 1-GB memory. We have three observations. First, the encryption latency becomes quite large when the size of NVM is large. For a low-frequency EE DW-AES, encrypting the whole memory can take as long as many hours or days. Second, for an EE of very high frequency, the encryption latency is very small. If the writing latency is larger than the encryption latency, the encryption latency will be countervailed by the writing latency. EE-2 has very high encryption speed; the time it takes to encrypt the whole memory turns to the time of reading and writing to all memory blocks sequentially. Therefore, the encryption time of EE-2 is different for PCM and MRAM as shown in the corresponding two columns of Fig. 12. On the contrary, EE-1 has very low encryption speed and its encryption latency is much larger than the data movement which is, thus, overlapped by the encryption time

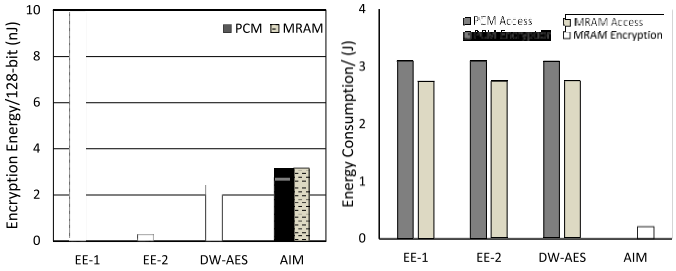


Fig. 13. Energy for encrypting 128-b block (left). Energy for accessing and encrypting 1-GB main memory (right).

that dominates the overall latency. Therefore, the encryption time of EE-1 is the same for both PCM and MRAM as shown in the corresponding two columns of EE-1. Fourth, multiple levels of parallelism in NVM accelerate the encryption process of AIM mechanism. When only one bank works in a chip, AIM can reach the encryption speed of 21 s and 1.2 s if the memory is implemented with PCM and MRAM, respectively. When we enable bank-level parallelism and let banks encrypt independently, AIM-B is able to encrypt the whole memory in 2.66 s and 0.15 s correspondingly. When the subarray level parallelism is enabled, AIM-S is able to encrypt the whole memory in 0.33 s and 0.018 s for PCM and MRAM, respectively.

From Fig. 12, we can see that EE-2 has the fastest encryption speed. When the main memory is implemented with PCM, the AIM-B design has similar encryption performance for 1-GB main memory and AIM-S can encrypt 1 GB much faster than EE-2. When the main memory is implemented with MRAM, all three designs AIM, AIM-B, and AIM-S work faster than EE-2. In addition, when the size of NVM scales up, the latency of EE-2 will scale up accordingly. However, for AIM-B, as long as main memory power budget allows, it can continue to leverage the parallelism and maintain a short encryption latency.

2) Power: All three designs AIM, AIM-B, and AIM-S work within the power budget [14] of main memory. Among the three designs, AIM has the smallest power which is around 1 and 13 mW for encrypting one chip of PCM-based main memory and MRAM-based main memory, respectively. The power of AIM-B is around 8 and 108 mW for encrypting one chip of PCM-based main memory and MRAM-based main memory, respectively. AIM-S has the largest power consumption since it has the best performance among the three designs and the power is 70 mW for encrypting each chip of PCM-based main memory. When the main memory is implemented with MRAM, the power of AIM-S exceeds the budget since the parallelism of AIM-S is the highest. Therefore, this design is not recommended if the power budget is small. However, since AIM-S has the best performance, if the system has the need for fast encryption and a large power budget, this design can still be employed. In conclusion, when we implement the AIM encryption schematic inside the NVMM, both power budget and encryption latency should be considered together to choose the most suitable design.

3) Energy Efficiency: Fig. 13 compares the energy efficiency for encrypting a 128-b block and for encrypting 1-GB

Fig. 14. Different AIM designs area overhead.

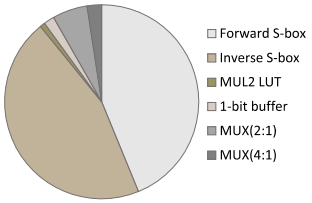


Fig. 15. Breakdown of encryption overhead.

main memory sequentially. From Fig. 13(a), EE-2 incurs the smallest energy, 0.265 nJ to encrypt 128-b block while AIM ranks the third and costs 2.78 and 3.17 nJ for 128-b PCM and MRAM blocks, respectively. Fig. 13(b) shows the energy consumption for encrypting 1-GB nonvolatile PCM and MRAM. In this figure, the lower parts of the first six columns show the energy spent on accessing main memory and the upper parts show the energy spent on encrypting process with the EEs. From this figure, we have two observations. First, EE-1, EE2, and DW-AES cost significant amount of energy on memory access. This is because, for an encryption operation outside of main memory like those of EE-1 and EE-2, the encryption processor needs to read a memory block from the main memory and then write this memory block back to its original position after encryption is completed. During the reading and writing periods, complex address decoding and bus transfer costs a lot of energy which is much more than the energy spent for encrypting this block. For DW-AES, the large energy comes from a large number of shifting operations required for write to perform the AES with domain-wall memory (DWM). Second, AIM costs the lowest energy compared with the three specific EEs. Since AIM encrypts each memory block inside the main memory, it avoids a large part of reading and writing energy consumption from outside the main memory.

4) Overhead Evaluation: Fig. 14 shows the area overhead results. As shown in this figure, AIM and AIM-B both incur insignificant area overhead of only 0.06% and 0.45% area overhead for PCM-based main memory, and 0.08% and 0.63% for MRAM-based main memory. Compared with AIM and AIM-B, AIM-S incurs a relatively larger area overhead of 3.59% and 5.05%.

Fig. 15 shows the distribution of hardware overhead. Among all added circuitry, forward S-box and inverse S-box have the largest area overhead. Since we can only look up byte by byte each time, more S-box LUTs mean more parallelism. Therefore, this overhead is unavoidable.

In addition, the area overhead for added circuitry, buffer rows are required for storing intermediate results generated in

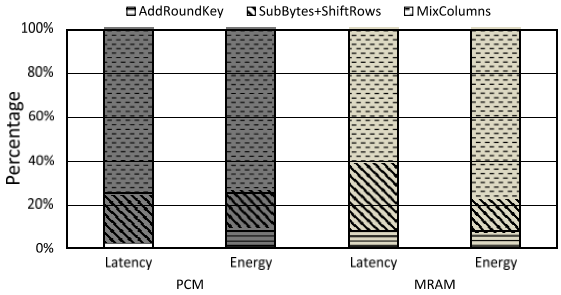


Fig. 16. Breakdown of latency and energy consumption.

the encryption process. As shown in Fig. 8, six buffer rows are required at most in the MixColumns step. For a normal memory bank that has 512 rows, six buffer rows are only 1.2% of the whole memory size. During the normal working time of the main memory, these six buffer rows can also be used for storing the working data.

5) Further Improvement: The breakdown of latency and energy consumption of AIM implementations for different encryption stages is shown in Fig. 16. Since SubBytes and ShiftRows are combined together in the AIM design, we analyze the two stages together. From Fig. 16, AddRoundKey consumes the minority of both total encryption latency and energy. This is because AddRoundKey stage only consists of parallel XOR operations based on Pinatubo design which is fast and costs a little energy. MixColumns consumes the medium latency and energy. This stage involves LUT operations of S-box. The latency of this stage varies with the number of S-box. MixColumns consumes the majority of both total latency and energy. This is because MixColumns generates several intermediate encryption state matrices from substeps. These intermediate encryption state matrices are buffered in the NVM cells in AIM design. This buffering process costs considerable energy and latency, since write operations in NVMM are usually expensive in terms of both energy and latency.

Writing pulsewidth to NVM determines the retention time of the written states. In AIM, the encryption latency and energy can be further reduced by supporting short-latency light writes, since the intermediate encryption states only need to stay for a short while. In this way, buffering intermediate encryption states with light writes will cost less energy and latency.

C. Evaluation of Different Cipher Modes

In this section, we evaluate the proposed CTR+CFB cipher modes from three aspects including encryption latency, energy efficiency, and area overhead. Among the evaluated cipher modes, CBC mode and CTR mode are used as the baselines. The other kinds of cipher modes are not evaluated since they do not support parallel encryption.

1) Latency: Fig. 17 shows the encryption latency of 1-GB memory with three different cipher modes under AIM design. Among the three cipher modes, the CBC mode has the shortest encryption latency. Compared with the CBC mode, the CTR mode increases the encryption latency by 2.17%

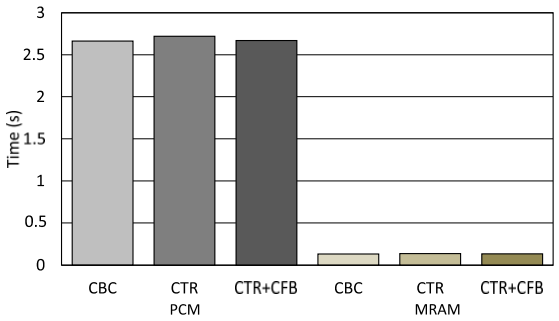


Fig. 17. Comparison of latency among different cipher modes.

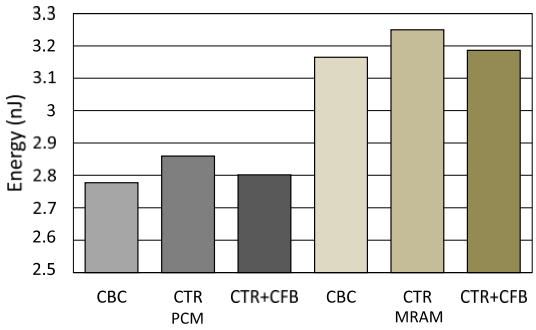


Fig. 18. Comparison of energy for encrypting 128-b block among different cipher modes.

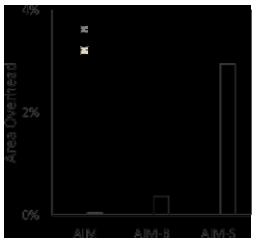


Fig. 19. Different AIM design area overhead with the CTR + CFB mode.

and 2.97% when the main memory is implemented with PCM and MRAM, respectively. Compared with the CTR mode, the proposed CTR+CFB cipher only slightly increases the encryption latency by 0.25% and 1.00% when the main memory is implemented with PCM and MRAM, respectively.

2) Energy Efficiency: Fig. 18 shows the encryption latency of encrypting one memory block with three different cipher modes under AIM design. Among the three cipher modes, the CBC mode has the highest energy efficiency because it generates the lowest energy overhead. Compared with the CBC mode, the CTR mode increases the energy consumption by 2.96% and 2.68% when the main memory is implemented with PCM and MRAM, respectively. Compared with the CTR mode, the proposed CTR+CFB cipher only slightly increases the energy consumption by 0.88% and 0.68% when the main memory is implemented with PCM and MRAM, respectively.

3) Overhead: Fig. 19 shows the area overhead results of implementing the proposed CTR+CFB cipher mode. As shown in this figure, AIM and AIM-B both incur insignificant area overhead of only 0.03% and 0.26% for

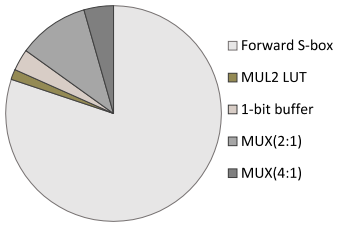


Fig. 20. Breakdown of encryption overhead with the CTR + CFB mode.

PCM-based main memory, and 0.05% and 0.37% area overhead for MRAM-based main memory. Compared with AIM and AIM-B, AIM-S incurs a relatively larger area overhead of 2.10% and 2.95% for PCM-based main memory and MRAM-based main memory. Compared with the CBC mode as shown in Fig. 14, the proposed CTR+CFB mode further reduces the area overhead by 40% on average by removing the inverse S-box.

Fig. 20 shows the distribution of area overhead. Compared with Fig. 15, the overhead from inverse S-box is removed. Among all added circuitry, forward S-box has the largest area overhead.

4) Discussion: The proposed CTR+CFB cipher avoids the complex management of a large amount of counter values and the security problem of the repeated counter in a short time. Meanwhile, the whole decryption process is based on the AES encryption algorithm, saving the hardware resource for implementing AES decryption algorithm. The above-mentioned experimental results show that the proposed CTR+CFB cipher mode achieves comparable or even lower latency and energy consumption with small hardware overhead.

The high encryption performance of AIM is owing to the high parallelism inside main memory architecture. As long as the memory cells are resistance-based, the corresponding memory technology will work with the proposed architecture. Furthermore, from our experimental evaluation, the activation latency of a memory row is much larger than the read/write latency of a memory cell and thus dominates the overall memory access latency. This activation latency mainly depends on the length of a memory row and the circuit design instead of the memory type. From our research, most NVMs have the properties varying between MRAM and PCM. Therefore, the proposed techniques are good for other NVM as well.

Although AIM requires memory architecture modification for encryption, the modification is small, simple, and easy to implement. The three levels of encryption parallelism AIM supports with in-memory design bring benefits of significantly improved encryption throughput and lowered energy overhead.

VIII. Related Work

A. Memory Encryption

Encryption has been widely suggested as a solution to secure both DRAM [15] and NVM-based main memory [7], [31]. These implementations perform encryption/decryption when writing/reading a cache line to/from main memory. Though encryption techniques base on Pad-based or Stream cipher encryption where memory access could be

overlapped with the Pad or Keystream generation reduces the decryption overhead, the system still suffers, since that overhead is on the critical path (memory read access). Different from them, Colp *et al.* [9] proposes to perform one-time encryption for smartphones and tablets only when the device is screen locked. AIM performs a one-time encryption to the main memory system before there is a possible attack (e.g., before power OFF). Other than that it runs as a normal main memory without any latency overhead. Furthermore, existing encryption methods rely on a dedicated EE on the processor or in the main memory. AIM takes advantage of in-memory computing, hence achieves a better throughput with less energy consumption.

B. NVM Encryption

There are several work that are particularly optimized for encryption on NVM [7], [16], [19], [26], [31]. I-NVMM [7] proposes to encrypt main memory incrementally. However, our method taking advantage of the PIM architecture outperforms i-NVMM, because i-NVMM relies on the dedicated AES engine on the processor side and limited by its small bandwidth and parallelism. DEUCE [31] and SECRET [26] propose techniques to reduce the bit flips during data encryption, which helps NVM reliability since encryption involves a significant amount of expensive writes. Silent Shredder [4] proposes techniques to obviate the writing of zeros to memory pages. Their techniques are orthogonal to AIM, and their method can be applied to AIM to further reduce the encryption energy.

C. In-Memory Encryption

In-memory encryption is a promising solution for NVM encryption which has limited research. Reference [11] explores different spintronic devices-based memory that could be leveraged to implement logic functions with the AES algorithm as a case study. Angizi *et al.* [3] demonstrates the efficiency of AES algorithm on a proposed in-memory processing platform with novel spin Hall effect-driven domain-wall motion devices that support both NVM cell and in-memory logic design. A recent work, Recryptor [34], proposes a reconfigurable cryptographic processor using in-memory computing. By replacing a standard SRAM bank with a custom bank with in-memory and near-memory computing, Recryptor provides an IoT platform that accelerates primitive cryptographic operations. DWM is also utilized to perform in-memory encryption [20], [29], [32], where inherent DWM device functions were used to perform the operations required for encryption.

IX. Conclusion

In this paper, we propose a fast and energy-efficient AES in-memory implementation, AIM, by taking advantage of NVM's resistive nature and utilizing existing memory peripheral circuits. With AIM, the memory blocks are encrypted simultaneously within each memory bank and the entire encryption process can be completed within the main memory without exposing the results to the memory bus. Compared with state-of-the-art AES engine running at 2.1 GHz, AIM can speed up the encryption process by 80X when encrypting 1-GB MRAM with 3% area overhead.

References

[1] *JEDEC DDR5 & NVDIMM-P Standards Under Development*. Accessed: 2017. [Online]. Available: https://www.jedec.org/news/pressreleases/jedec-ddr5-nvdimm-p-standards-under-development

[2] (2015). *Intel: First 3D XPoint SSDs Will Feature up to 6GB/s of Bandwidth*. [Online]. Available: http://www:kitguru:net/components/memory/anton-shilov/intelfirst-3d-xpoint-ssds-will-feature- up-to-6gbs-of-bandwidth

[3] S. Angizi, Z. He, N. Bagherzadeh, and D. Fan, "Design and evaluation of a spintronic in-memory processing platform for non-volatile data encryption," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/8113549/

[4] A. Awad, P. Manadhata, S. Haber, Y. Solihin, and W. Horne, "Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers," *ACM SIGPLAN Notices*, vol. 15, no. 4, pp. 263–276, 2016.

[5] E. B. Barker *et al.*, "Guideline for using cryptographic standards in the federal government: Directives, mandates and policies," Tech. Rep. Special Publication (NIST SP)-800-175A, Aug. 2016. [Online]. Available: https://www.nist.gov/publications/guideline-using-cryptographic-standards-federal-government-directives-mandates-and

[6] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 33–38.

[7] S. Chhabra and Y. Solihin, "i-NVMM: A secure non-volatile main memory system with incremental encryption," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, pp. 177–188, 2011.

[8] P. Chown, *Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)*, document RFC 3268, 2002. [Online]. Available: http://www.rfc-editor.org/info/rfc3268

[9] P. Colp *et al.*, "Protecting data on smartphones and tablets from memory attacks," in *Proc. 20th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2015, pp. 177–189.

[10] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 7, no. 31, pp. 994–1007, Jul. 2012.

[11] D. Fan, S. Angizi, and Z. He, "In-memory computing with spintronic devices," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2017, pp. 683–688.

[12] J. A. Halderman *et al.*, "Lest we remember: Cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, no. 5, pp. 91–98, 2009, pp. 45–60.

[13] P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen, "Design and implementation of low-area and low-power aes encryption hardware core," in *Proc. 9th EUROMICRO Conf. Digit. Syst. Design*, Aug./Sep. 2006, pp. 577–583.

[14] A. Hay, K. Strauss, T. Sherwood, G. H. Loh, and D. Burger, "Preventing PCM banks from seizing too much power," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2011, pp. 186–195.

[15] M. Henson and S. Taylor, "Memory encryption: A survey of existing techniques," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 1–26, Mar. 2014.

[16] F. Huang, D. Feng, Y. Hua, and W. Zhou, "A wear-leveling-aware counter mode for data encryption in non-volatile memories," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Mar. 2017, pp. 910–913.

[17] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (SALP) in DRAM," in *Proc. 39th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2012, pp. 368–379.

[18] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proc. ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[19] D. Liu, X. Luo, Y. Li, Z. Shao, and Y. Guan, "An energy-efficient encryption mechanism for NVM-based main memory in mobile systems," *J. Syst. Archit.*, vol. 76, pp. 47–57, May 2017.

[20] T. Luo, W. Zhang, B. He, and D. Maskell, "A racetrack memory based in-memory booth multiplier for cryptography application," in *Proc. ASP-DAC*, Jan. 2016, pp. 286–291.

[21] R. Maes, A. Van Herrewege, and I. Verbauwhede, "PUFKY: A fully functional PUF-based cryptographic key generator," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, 2012, pp. 302–319.

[22] S. Mathew *et al.*, "53Gbps native GF$(2^4)^2$composite-field AES-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors," in *Proc. IEEE Symp. VLSI Circuits (VLSIC)*, Jun. 2010, pp. 169–170.

[23] D. McGrew and J. Viega, "The Galois/counter mode of operation (GCM)," *NIST Modes Oper. Process*, vol. 20, 2004. [Online]. Available: https://scholar.google.com/scholar?hl=en&as_sdt=0%2C39&q=The+Galois%2FCounter+Mode+of+Operation+%28GCM%29+&btnG=

[24] U. Russo, D. Ielmini, and A. L. Lacaita, "Analytical modeling of chalcogenide crystallization for PCM data-retention extrapolation," *IEEE Trans. Electron Devices*, vol. 54, no. 10, pp. 2769–2777, Oct. 2007.

[25] K. Suzuki *et al.*, "The non-volatile memory technology database (NVMDB)," Dept. Comput. Sci. Eng., Univ. California, San Diego, San Diego, CA, USA, 2015.

[26] S. Swami, J. Rakshit, and K. Mohanram, "SECRET: Smartly EnCRypted Energy efficienT non-volatile memories," in *Proc. 53nd ACM/EDAC/IEEE Design Automat. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[27] K. Tsuchida *et al.*, "A 64Mb MRAM with clamped-reference and adequate-reference schemes," in *Proc. ISSCC*, Feb. 2010, pp. 258–259.

[28] R. K. Venkatesan, S. Herr, and E. Rotenberg, "Retention-aware placement in DRAM (RAPID): Software methods for quasi-non-volatile DRAM," in *Proc. 12th Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2006, pp. 155–165.

[29] Y. Wang, L. Ni, C.-H. Chang, and H. Yu, "DW-AES: A domain-wall nanowire-based AES for high throughput and energy-efficient data encryption in non-volatile memory," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 11, pp. 2426–2440, Nov. 2016.

[30] H.-S. P. Wong *et al.*, "Metal–oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, Jun. 2012.

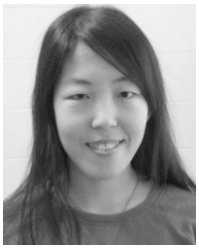
[31] V. Young, P. J. Nair, and M. K. Qureshi, "Deuce: Write-efficient encryption for non-volatile memories," in *Proc. 20th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2015, pp. 33–44.

[32] H. Zhang, C. Zhang, X. Zhang, G. Sun, and J. Shu, "Pin tumbler lock: A shift based encryption mechanism for racetrack memory," in *Proc. 21st Asia South Pacific Design Automat. Conf. (ASP-DAC)*, Jan. 2016, pp. 354–359.

[33] X. Zhang, C. Zhang, G. Sun, J. Di, and T. Zhang, "An efficient run-time encryption scheme for non-volatile main memory," in *Proc. Int. Conf. Compil., Archit. Synth. Embedded Syst.*, Sep./Oct. 2013, pp. 1–10.

[34] Y. Zhang *et al.*, "Recryptor: A reconfigurable in-memory cryptographic cortex-m0 processor for IoT," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2017, pp. C264–C265.

[35] P. Zhoum, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2009, pp. 14–23.



Mimi Xie (S'13) received the B.E. and M.S. degrees from the College of Computer Science, Chongqing University, Chongqing, China, in 2010 and 2013, respectively. She is currently working toward the Ph.D. degree at the School of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA, USA.

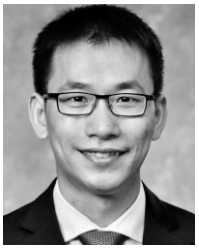
Her current research interests include compiler optimization, nonvolatile memory, and embedded systems.

Shuangchen Li (S'15) received the B.S. and M.S. degrees from the Department of Electrical Engineering, Tsinghua University, Beijing, China, in 2011 and 2014, respectively, and the Ph.D. degree from the University of California at Santa Barbara, Santa Barbara, CA, USA, in 2018.

He currently holds a postdoctoral position at the University of California at Santa Barbara. His current research interests include memory-centric architectures, emerging nonvolatile memory, and non-von Neumann architecture for emerging applications.

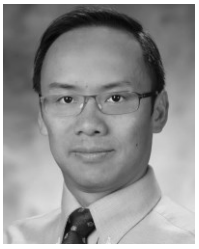
Alvin Oliver Glova (S'17) received the B.S. degree in computer engineering from the University of the Philippines Diliman, Quezon City, Philippines, in 2009 and the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2012. He is currently working toward the Ph.D. degree at the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, USA.

He was a Research Engineer at the New Memory Device Technology Group, Research and Development Division, SK Hynix, Icheon, South Korea, where he was involved in STT-MRAM development. His current research interests include emerging memory technologies, secure computation and machine learning.



Jingtong Hu (S'09–M'13) received the B.E. degree from the School of Computer Science and Technology, Shandong University, Jinan, China, in 2007, and the M.S. and Ph.D. degrees in computer science from The University of Texas at Dallas, Richardson, TX, USA, in 2010 and 2013, respectively.

He is currently an Assistant Professor at the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA, USA. His current research interests include embedded systems, field-programmable gate array, and nonvolatile memory.



Yuan Xie (F'15) received the B.S. degree from the Electronic Engineering Department, Tsinghua University, Beijing, China, and the M.S. and Ph.D. degrees from the Electrical Engineering Department, Princeton University, Princeton, NJ, USA.

Since 2003, he has been a Professor at Pennsylvania State University, State College, PA, USA. From 2012 to 2013, he was at the AMD Research China Laboratory, Beijing, China. From 2002 to 2003, he was at IBM, Armonk, NY, USA. He is currently a Professor at the Electrical and Computer Engineering Department, University of California at Santa Barbara, Santa Barbara, CA, USA. His current research interests include computer architecture, electronic design automation, and VLSI design.