FlexENM: A Flexible Encrypting-Near-Memory with Refresh-less eDRAM-based Multi-mode AES

Hyunseob Shin
Department of Electrical Engineering
Korea University
Seoul, South Korea

Jaeha Kung
Department of Electrical Engineering
Korea University
Seoul, South Korea

***Abstract*—On-chip cryptography engines face significant challenges in efficiently processing large volumes of data while maintaining security and versatility. Most existing solutions support only a single AES mode, limiting their applicability across diverse use cases. This paper introduces FlexENM, a low-power and area-efficient near-eDRAM encryption engine. The FlexENM implements refresh-less operation by leveraging inherent characteristics of the AES algorithm, reordering AES stages, and employing a simultaneous read and write scheme using dual-port eDRAM. Furthermore, FlexENM supports three AES modes, parallelizing their operations and sharing hardware resources across different modes to improve compute efficiency. Compared to other AES engines, FlexENM achieves 16% lower power consumption and 83% higher throughput per unit area, on average, demonstrating improved power- and area-efficiency for on-chip data protection.**

***Index Terms*—AES encryption, embedded DRAM, hardware acceleration, near-memory computing**

I. Introduction

In the big data era, efficiently encrypting large data volumes poses security challenges. Memory bottlenecks constrain traditional Advanced Encryption Standard (AES) hardware implementations. Processing-near-memory (PNM) technology offers a solution through subarray parallelism, reduced data movement, and enhanced security via on-chip integration [1], [2]. This integration protects against physical tampering on off-chip memory/buses and side-channel attacks by creating a more isolated execution environment. The AES, a powerful symmetric key cryptography algorithm, supports various modes to enhance security for different purposes. Most previous AES hardware designs support only one specific mode, e.g., ECB, GCM [3], or XTS [4], limiting their versatility.

To address these limitations, we propose FlexENM, an on-chip AES accelerator based on PNM using embedded DRAM (eDRAM). FlexENM is designed to encrypt or decrypt data to or from the off-chip memory using several modes of AES (Fig. 1). Our approach eliminates the need for expensive refresh operations in the eDRAM memory during AES PNM operations by leveraging the frequent data updates in AES algorithms, reordering the execution of each stage, and utilizing dual-port eDRAM cells that allow simultaneous read and write operations. In addition, FlexENM efficiently supports three widely-used AES modes, i.e., ECB, GCM, and XTS, by sharing computing units and registers. When AES operations are not needed, FlexENM can act as a typical on-chip memory.

• *J. Kung is the corresponding author (`jhkung@korea.ac.kr`).*

The main contributions are summarized as follows:

- **Refresh-less eDRAM during AES**: We present a high-performance eDRAM-based AES accelerator for on-chip data encryption. The proposed architecture leverages inherent features of the AES algorithm to eliminate the need for refresh operations during encryption, thereby significantly enhancing power efficiency.
- **Hardware support for multiple AES modes**: Our design supports multiple AES modes by implementing general GF multipliers and α multipliers. Despite added functionality, the architecture maintains high throughput per unit area due to the compact size of the eDRAM memory.
- **Improved AES performance/efficiency**: We compare the FlexENM with previous AES accelerators and achieve 14$\sim$73% better power efficiency and 8$\sim$261% higher area efficiency even with the support of multiple modes.

II. Preliminaries and Threat Model

A. Advanced Encryption Standard (AES)

AES is a symmetric key cryptography algorithm based on operations in GF(2^8), approved by the National Institute of Standards and Technology (NIST). AES is a block cipher that processes plaintexts in fixed-size blocks of 128-bit. Each block is converted into a column-major 4$\times$4 matrix (with 1-byte entries), referred to as the *state*. The AES encryption starts with an initial secret key addition, followed by a series of regular rounds consisting of *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey* stages (Fig. 2).

1) Galois Field Arithmetic Operation: Elements of GF(2^m) are polynomials with binary coefficients and degrees less than m, which can be represented as m-bit binary data. The addition, denoted by $\oplus$, is calculated using coefficient-wise XOR. The multiplication, denoted by $\otimes$, involves polynomial multiplication followed by a reduction operation. The reduction operation is a modular operation using the irreducible polynomial $P_m(x)$, which reduces the degree of the multiplication output to smaller than m. The multiplication is typically implemented using "shift-and-add" followed by the reduction in hardware.

2) Encryption Stages: Fig. 2 illustrates the operations of each stage. The *SubBytes* stage involves substituting each byte of the state using an S-box constructed from a specific transformation function. During the *ShiftRows* stage, cyclic left shifts are performed on the state's rows by 0, 1, 2, and

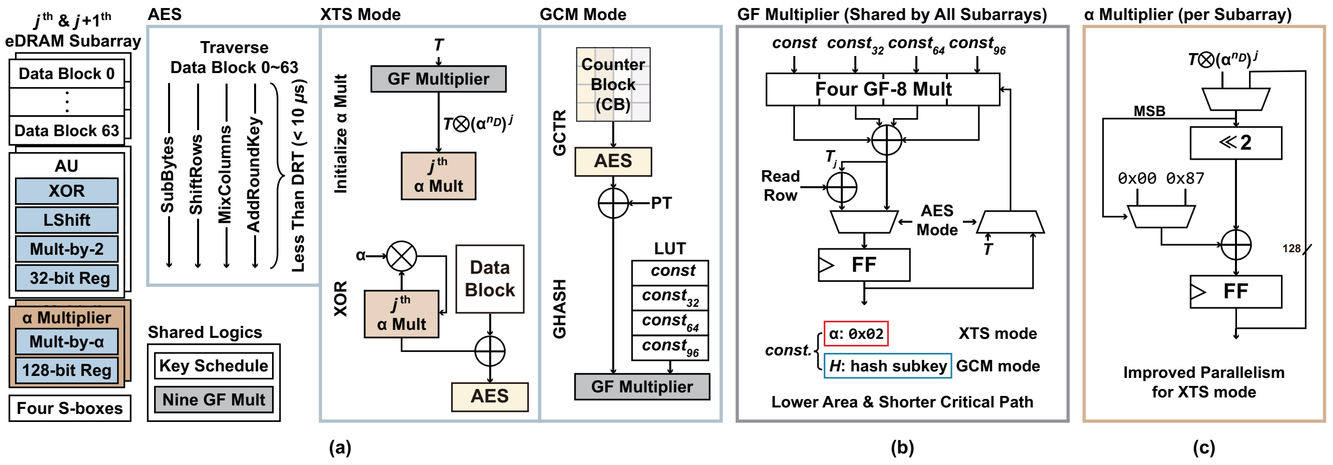


Fig. 1. (a) Overview of the proposed eDRAM-based AES engine supporting various AES modes, i.e., ECB, XTS, and GCM modes. Execution orders of the AES algorithm are determined to enable refresh-less eDRAM operations. (b) The proposed GF multiplier consists of small LUTs for pre-computed constant values and four GF-8 multipliers that result in a lower area and reduced critical path. (c) The α multiplier is placed at each eDRAM subarray to parallelize the sequential process of XTS mode. Here, n_D and j indicate the number of data blocks per subarray and the subarray index, respectively.

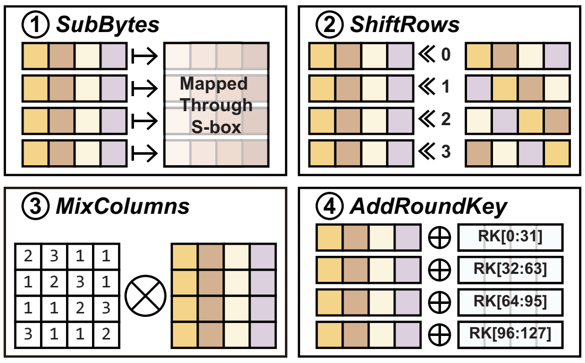


Fig. 2. Compute stages of an AES algorithm. The regular round, which consists of *SubBytes, ShiftRows, MixColumns, AddRoundKey* stages, repeats 9, 11, and 13 times for the key length of 128, 192, and 256-bit, respectively.

3 bytes, respectively. In the *MixColumns* stage, each column is transformed through multiplication with a constant matrix. Finally, in the *AddRoundKey* stage, the state is XORed with the round key (RK) generated from the key scheduler.

B. Various AES Modes

Electronic Codebook (ECB) is the most basic mode in AES, which simply passes plaintexts block-by-block to obtain ciphertexts. Since data blocks can be encrypted independently, computations can be highly parallelized. Although simple to implement, ECB is highly vulnerable to brute-force attacks due to its deterministic behavior and lack of randomness.

To enhance the security level of AES, several variations have been made on top of the ECB mode. XEX-based tweaked-codebook mode with ciphertext stealing (XTS), where XEX stands for xor-encrypt-xor, is designed for encrypting data stored in storage devices [4]. The XTS mode assigns a tweak value to a set of data blocks and uses two different keys. The state is XORed with $T \otimes \alpha^j$ before and after the AES algorithm using the first key, as shown in Fig. 3(a). Here, T is the encrypted tweak value using the second key, α is a constant, and j is the encrypted data block index. The XTS mode supports encryption of extra data blocks smaller than

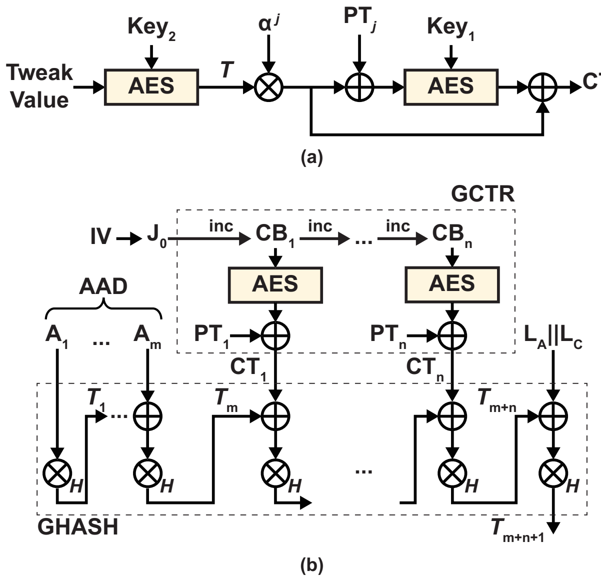


Fig. 3. (a) The operation of the XTS mode without the ciphertext stealing, and (b) GCTR and GHASH which are operations of the GCM mode.

16 bytes through ciphertext stealing, where the extra block is padded with part of the previous ciphertext before encryption.

Besides the XTS mode, Galois/Counter Mode (GCM) provides strong authentication assurance and encryption [3]. The GCM encryption algorithm consists of GCTR and GHASH, as shown in Fig. 3(b). The GCTR function, responsible for encryption, operates by applying AES to counter blocks (CBs). These CBs are generated from an initial vector (IV) and an increment function. The resulting output from AES is then XORed with plaintexts (PTs) to produce ciphertexts (CTs). Authentication in GCM is achieved through the GHASH function. It calculates $T_k = (T_{k-1} \oplus X_k) \otimes H$ and returns the final T_{m+n+1} of Fig. 3(b), i.e., GHASH$(X, H) = \sum_{k=1}^{N} X_k \otimes H^{N-k+1}$, where H is a hash subkey, X_k are the inputs (A, CT, or $L_A||L_C$), and N is the number of inputs ('$N = n+m+1$' in Fig. 3(b)). For the authentication, the GHASH function processes the additional authentication data (AAD), followed by ciphertext blocks. Finally, it processes L_A and L_C, i.e., lengths of AAD and CTs in 64-bit, respectively.

C. Embedded DRAM (eDRAM)

The eDRAM is a type of DRAM fabricated using the logic-compatible CMOS process, allowing it to be embedded on a chip. The eDRAM cell has multiple advantages over other on-chip memory technologies, such as SRAM or MRAM, although it also has several limitations. As a dynamic memory, the eDRAM cell does not require a constant supply voltage to maintain data, unlike SRAM. This results in lower leakage current (*power-efficient*) and typically fewer transistors per cell (*area-efficient*). Also, the eDRAM cell has better write speed and less write energy consumption than the MRAM cell [5].

However, due to the dynamic nature of an eDRAM cell, the storage node voltage decays over time due to the leakage current. This necessitates two specific operations: write-back and refresh. The write-back operation occurs immediately after a read operation, restoring the voltage level of the data stored in the cell. The refresh operation is a sequence of read operations followed by write-backs for all data within the memory, occurring periodically, preventing data loss. The interval between refresh operations, or the duration that an eDRAM cell safely retains its original data, is called data retention time (DRT). Due to the need for frequent refresh operations, eDRAM is often considered to consume more power than expected.

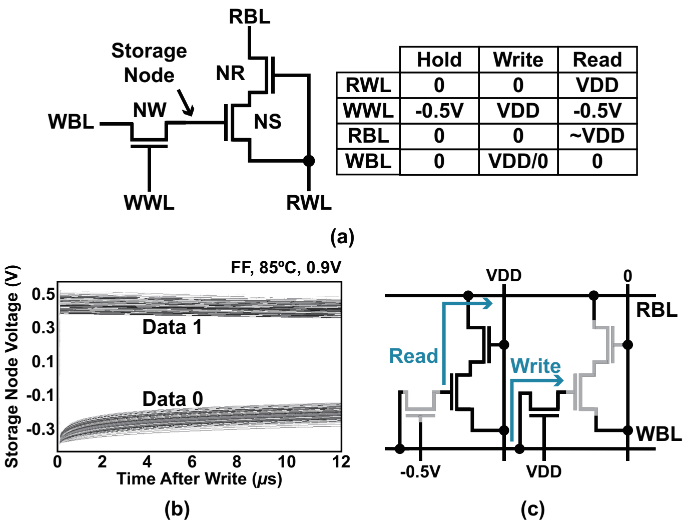
D. Threat Model

In our threat model, we assume that adversaries have the capability to probe the bus [6] or scan the off-chip memory using a technique like cold boot attack [7]. We consider only the processor chip to be trusted, while the bus and off-chip memory are deemed vulnerable and untrusted. Our model assumes a randomly generated secret key is embedded within the processor chip. This key is presumed to be inaccessible from external sources. Thus, our model focuses on protecting data leaving the chip, considering threats such as eavesdropping on the memory bus and physical attacks on off-chip memory.

III. Proposed FlexENM Architecture

This section details the implementation of our proposed eDRAM-based encrypting-near-memory engine, i.e., FlexENM (Fig. 1(a)). FlexENM consists of multiple eDRAM subarrays, each with a size of 256×32. A data block, i.e., a 4×4 array of 16 bytes, is stored in a column-major order within the memory subarray, each occupying 4 rows and 32 columns (i.e., 128-bit). Thus, one subarray can accommodate up to 64 data blocks.

As near-memory compute logics, each subarray is equipped with an arithmetic unit (AU) and an α multiplier, while two subarrays share four S-boxes. Also, a key schedule unit and general GF multipliers are globally shared by all subarrays. The AU has XOR gates for GF addition (e.g., for the *AddRoundKey* stage), circular left shift logics for the *ShiftRows* stage, and multiply-by-2 logics with a 32-bit register for the *MixColumns* stage. The α multiplier includes a multiply-by-α logic and a 128-bit register to store intermediate values for the *MixColumns* stage and XTS mode operations.



	Hold	Write	Read
RWL	0	0	VDD
WWL	-0.5V	VDD	-0.5V
RBL	0	0	~VDD
WBL	0	VDD/0	0

Fig. 4. (a) The structure of a 3T gain cell eDRAM and its operating conditions. (b) The Monte Carlo simulation result measures the data retention time of an eDRAM cell (i.e., 10μs). (c) The simultaneous read and write scheme employed in the AES mode.

Our FlexENM can operate as either a normal on-chip memory or an AES accelerator supporting ECB, XTS, and GCM modes. When operating as normal memory, cell refresh and write-back operations are required, just as in typical eDRAM memory. When functioning as an AES accelerator, it employs a simultaneous read and write scheme of a 3T gain cell eDRAM (Fig. 4) and a proper selection of execution order (Fig. 1(a)), enabling refresh-less AES operations.

A. Refresh-less eDRAM Operations During AES

Our design leverages the AES algorithm's inherent frequent state updates to realize refresh-less AES operations in eDRAM subarrays. Unlike previous PNM approaches [1], [2], which complete all AES stages on a single data block, we *traverse all data blocks in the subarray for each AES stage*. Thus, we need to keep the number of stored data blocks per subarray less than 390 when operating at 625MHz so that state updates for all data blocks are completed within eDRAM data retention time (DRT), i.e., 10μs in our design (Fig. 4(b)). This approach *naturally refreshes* each memory row while performing the required AES computations.

To further reduce the end-to-end latency of traversing all rows and achieve a more stable refresh-less operation, we exploit the dual-port feature of the eDRAM cell, enabling simultaneous read and write operations, as illustrated in Fig. 4(c). This scheme hides write cycles, approximately halving the total clock cycle consumption. Consequently, *SubBytes, ShiftRows*, and *AddRoundKey* stages each take 4 clock cycles, while the *MixColumns* stage consumes 16 clock cycles per data block. As a result, each row is updated every '$n_D \times 16$' cycles, which translates to 1,024 cycles when the subarray is fully utilized, i.e., $n_D = 64$, where n_D is the number of data blocks to be encrypted per subarray. Operating FlexENM at 625MHz, the row update interval when traversing all 64 data blocks becomes 1.64μs, which is significantly shorter than the DRT (Fig. 4(b)), eliminating the need for energy-hungry refresh operations.

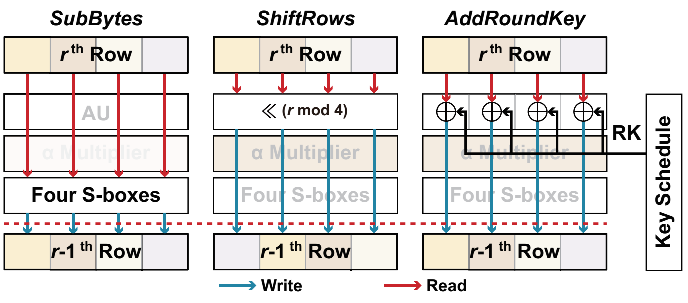


Fig. 5. Execution of three stages in AES, i.e., *SubBytes, ShiftRows*, and *AddRoundKey*, leveraging dual-port eDRAM cells (red dotted line is flip-flops).

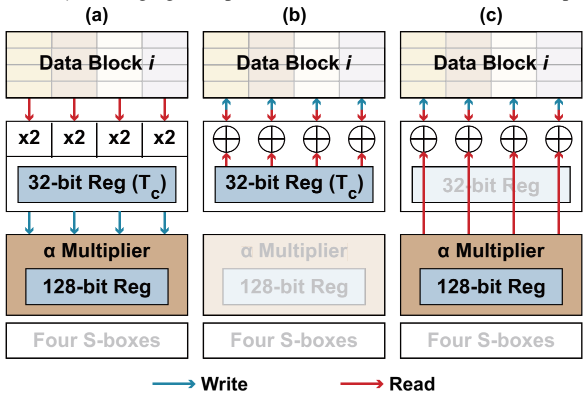


Fig. 6. Execution of the *MixColumns* stage: (a) each row in a data block is read and its doubled values are stored in the register of the α multiplier while T_c is stored and updated in the register of the AU, (b) each row is added with T_c and written back to the eDRAM subarray, and finally (c) each row is added with doubled values to obtain the final result of the *MixColumns* stage.

B. Supporting Compute Stages of AES in FlexENM

Fig. 5 illustrates the execution of the *SubBytes, ShiftRows*, and *AddRoundKey* stages in AES, along with the active submodules for each stage. Note that simultaneous read and write are allowed on the r^{th} and the $(r-1)^{\text{th}}$ rows, respectively, by using dual-port eDRAM cells. In the *SubBytes* stage, the read row is directly fed into four S-boxes, each converting one byte. For the *ShiftRows* stage, the read row is circularly left-shifted by $(r \bmod 4)$-bit. Due to the commutative nature of *SubBytes* and *ShiftRows* stages, their orders can be exchanged, allowing shared S-boxes between two subarrays in FlexENM as shown in Fig. 1(a). This approach reduces the number of S-boxes and enables a single-cycle substitution. During the *AddRoundKey* stage, the AU performs XORs between the read rows and the round keys generated by the on-the-fly key scheduling module.

The *MixColumns* stage involves calculating the left-hand side of the following equation:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,c} \\ a_{1,c} \\ a_{2,c} \\ a_{3,c} \end{bmatrix} = \begin{bmatrix} 2a_{0,c} \oplus 3a_{1,c} \oplus a_{2,c} \oplus a_{3,c} \\ a_{0,c} \oplus 2a_{1,c} \oplus 3a_{2,c} \oplus a_{3,c} \\ a_{0,c} \oplus a_{1,c} \oplus 2a_{2,c} \oplus 3a_{3,c} \\ 3a_{0,c} \oplus a_{1,c} \oplus a_{2,c} \oplus 2a_{3,c} \end{bmatrix} \quad (1)$$

where $a_{r,c}$ denotes the byte in row r and column c. We can factor out a common expression T_c in Eq. (1), which becomes

$$\begin{bmatrix} T_c \oplus 2a_{0,c} \oplus 2a_{1,c} \oplus a_{0,c} \\ T_c \oplus 2a_{1,c} \oplus 2a_{2,c} \oplus a_{1,c} \\ T_c \oplus 2a_{2,c} \oplus 2a_{3,c} \oplus a_{2,c} \\ T_c \oplus 2a_{3,c} \oplus 2a_{0,c} \oplus a_{3,c} \end{bmatrix}, \quad (2)$$

Algorithm 1 Parallelized GHASH

```
    Input: 128-bit $X_j$, $H^8$ and $H$
    Output: 128-bit GHASH($X$,$H$)
1:  for $k = 0$ to $7$ do          // parallelized w/ eight GF mult
2:      $M_k \leftarrow X_k$
3:      for $l = 1$ to $\frac{n_D \times n_A}{8} - 1$ do
4:          $M_k \leftarrow (M_k \otimes H^8 \oplus X_{k+8l})$
5:      end for
6:      $M_k \leftarrow M_k \otimes H^{8-k}$
7:  end for
8:  return $\sum_{k=0}^{7} M_k$
```

where $T_c = a_{0,c} \oplus a_{1,c} \oplus a_{2,c} \oplus a_{3,c}$. The AU in each subarray first performs the multiply-by-2 operation on each row, storing the results in the 128-bit register of the α multiplier. Concurrently, it computes T_c and stores it in the AU's internal 32-bit register (Fig. 6(a)). Then, the AU reads each row again to add it with T_c and writes the result back to the memory (Fig. 6(b)). Finally, it completes the computation in Eq. (2) over eight cycles by reading data block from both the eDRAM subarray and the register in the α multiplier (Fig. 6(c)).

C. XTS and GCM Mode Support in FlexENM

1) Mode Implementation: The XTS mode requires sequential operations equal to the number of all data blocks (i.e., $n_D \times n_A$) in generating α^j, where n_A is the number of subarrays involved in the encryption process (*active subarrays*) and j is the subarray index. To parallelize a portion of these sequential operations in FlexENM, we place a dedicated α multiplier per subarray. Initially, the general GF multiplier calculates $T \otimes (\alpha^{n_D})^j$ and stores the result in the 128-bit register in the j^{th} α multiplier as an initialization step (Fig. 1(a)). Subsequently, during the XOR operation prior to AES, mult-by-α unit calculates and updates α^j in the internal 128-bit register of each α multiplier, while the AU XORs the data block (PT$_j$) with the value stored in the α multiplier's register. This approach partially parallelizes the XTS mode, reducing the sequential parts by $1/n_D$.

For the GCM mode, FlexENM supports GCTR with our AES computation flow (as in Section III-B) and GHASH using general GF multipliers, as shown in Fig. 1(a). The parallelization of the GHASH function was suggested in [8], and FlexENM allows this by placing nine (shared) GF multipliers. As described in Algorithm 1, eight GF multipliers compute line 2-5 in parallel, and the remaining one multiplier performs line 6 to compute the result of the GHASH function by iteratively multiplying it by H and accumulating them. For example, the second GF multiplier ($k = 1$) computes line 4 as $(((X_1 \otimes H^8 \oplus X_9) \otimes H^8 \oplus X_{17}) \cdots)$. FlexENM can process AAD and $L_A || L_C$ for the authentication in parallel with GCTR and after processing ciphertexts, respectively.

2) General GF Multiplier: In FlexENM, general GF multipliers are shared by all subarrays to support GF(2^{128}) multiplication in both XTS and GCM modes, where one of the inputs is a constant value. A naïve implementation of a GF(2^{128}) mul-

Algorithm 2 GF-8 Multiplication

```
Input: 8-bit $a_{r,c}$ and 128-bit $const$
Output: 128-bit $result$
1: $result \leftarrow 0$
2: $temp \leftarrow const$
3: for $r \bmod 4 < 4$ do                          // loop on a data block
4:     for $i < 8$ do                              // loop over 8-bit input
5:         if $a_{r,c}[i] = 1$ then
6:             $result \leftarrow result \oplus temp$
7:         end if
8:         $temp \leftarrow temp \ll 1$
9:         if $temp[MSB] = 1$ then
10:            $temp \leftarrow temp \oplus \texttt{0x87}$          // reduction step
11:        end if
12:    end for
13:    $r \leftarrow r + 1$                          // read next row
14: end for
15: return $result$
```

tiplier using the "shift-and-add" approach requires numerous XOR gates, logical shifters, and extensive wiring, which can significantly reduce the throughput. While look-up table (LUT) offers high-throughput implementation for constant inputs, the required 128×128-bit LUT incurs substantial area overhead.

To address these challenges, we propose a 4-cycle GF multiplier that utilizes two 4×128-bit LUTs with four GF-8 multipliers (Fig. 1(b)). This design is optimized for the data organization within the subarray. The data block $D = [d_0, \ldots, d_{15}]$ is stored in the memory subarray in column-major order as follows:

$$\begin{bmatrix} d_0 & d_4 & d_8 & d_{12} \\ d_1 & d_5 & d_9 & d_{13} \\ d_2 & d_6 & d_{10} & d_{14} \\ d_3 & d_7 & d_{11} & d_{15} \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}. \quad (3)$$

Each GF-8 multiplier operates on an 8-bit input $a_{r,c}$ and a 128-bit $const$ (i.e., a pre-computed constant for each mode). The proposed GF multiplier executes the multiplication defined in Eq. (4), considering the data layout in the eDRAM subarray.

$$a_{r,c} \otimes (const_{32 \cdot c} \ll (r \bmod 4) \cdot 8) \quad (4)$$

Here, $const$ represents α^{n_D} in XTS and H^8 or H in GCM. The term $const_x$ denotes $const$ left-shifted by x bits.

Algorithm 2 details the execution steps involved in the proposed GF-8 multiplication. As shown in line 4-12, only 8 iterations are required, significantly reducing the wiring overhead and shortening the critical path compared to the naïve 4-cycle implementation that directly processes 32-bit and 128-bit inputs, which requires 32 iterations. We maintain reasonable performance while minimizing hardware complexity by reducing the number of loops from 32 to 8 and storing pre-computed constants in the two 4×128-bit LUTs (for H and H^8). This design balances performance, area efficiency, and adaptability to specific data organization by processing data in 8-bit chunks, enabling effective support for both XTS and GCM modes.

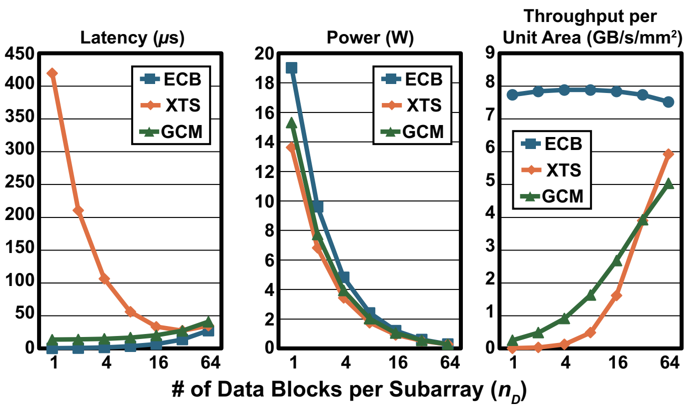


Fig. 7. Latency, power consumption, and throughput per unit area for encrypting 16,384 data blocks in each mode at various n_D values.

IV. Experimental Results and Evaluation

The proposed FlexENM is implemented in 28nm FD-SOI CMOS technology. We estimated the DRT of 10μs by running 1,000 Monte Carlo trials in FF corner at 85$°C$ (Fig. 4(c)), which is sufficient for the refresh-less operation with a 1.6ns clock period (refer to Section III-A). S-boxes are implemented using the composite-field approach as described in EE [9], enabling both encryption and decryption operations. We compare the performance of FlexENM with other PNM architectures, i.e., Sealer [1] and AIM [2], and previous ASIC or FPGA accelerators for the ECB, XTS, or GCM mode [8]–[12]. For ASIC works [8], [9], we estimated the power consumption and area of 256KB SRAM by using CACTI [13].

A. Operating Points of FlexENM

Depending on the level of parallelism inherent to each AES mode, the optimal operating point may differ. Fig. 7 shows the latency (left), power consumption (middle), and throughput per unit area (right) for encrypting 16,384 data blocks (= 256KB) using different AES modes at various n_D values. The n_D represents the number of data blocks to be handled by each subarray. Thus, if $n_D = 1$, 16,384 subarrays are required to encrypt all data blocks (*full parallelism*). On the other hand, if $n_D = 64$, 256 subarrays are required to be active to encrypt all data blocks (*full utilization*).

The latency increases as the level of parallelism decreases (i.e., $n_D = 1 \rightarrow 64$) for ECB and GCM modes. However, the latency of XTS mode decreases since the α multiplier initialization overhead increases with the number of subarrays. Power consumption shows an inverse relationship with the number of active subarrays, as fewer active AUs and α multipliers consume power. If we look at the throughput per unit area, the ECB mode shows similar performance regardless of n_D, while XTS and GCM show increasing performance as n_D increases. This analysis demonstrates various operating conditions, allowing users to select different FlexENM configurations on latency-critical, power-constrained, or area-efficient deployment scenarios.

B. Performance/Power/Efficiency Analysis

1) Baselines: For the basic AES-ECB mode, there are two categories of the prior work: i) near-memory AES engines

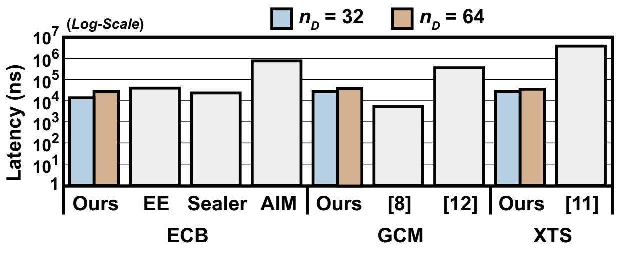


Fig. 8. Latency comparison among various architectures in each AES mode.

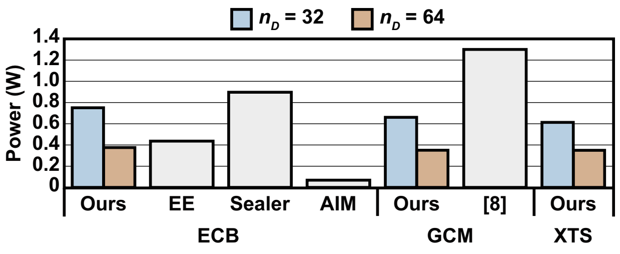


Fig. 9. Comparison on power consumption among various architectures.

(Sealer [1] and AIM [2]) and ii) ASIC hardware (EE [9]). The Sealer is an SRAM-based PNM architecture that performs XOR between two rows in a single read by performing NOR on two bitlines. The AIM utilizes non-volatile memory and modifies sense amplifier circuits to carry out bitwise operations to accelerate XOR operations. The EE proposes an on-die AES engine employing a composite-field S-box for reduced delay and area, a fused MixColumns circuit, and a folded datapath for further area reduction.

There are several studies on supporting XTS or GCM mode for enhanced security [8], [10]–[12]. For the XTS mode acceleration, the authors in [10] have presented synchronized dual AES cores, combining a standard core with a modified one for tweak value calculation and encryption/decryption for improving throughput. In [8], the computation of exponentiation of the hash subkey in the GHASH function is parallelized by using pre-computed H^{2j}, achieving lower latency. However, none of them support both XTS and GCM modes in the same hardware, while FlexENM supports all ECB, XTS, and GCM modes.

2) Performance: We evaluate FlexENM with n_D set to 32 and 64 against existing architectures, analyzing and extrapolating the reported latency for encrypting 256KB data. Fig. 8 illustrates the encryption latency across different designs for ECB, GCM, and XTS modes. In ECB mode, our design with $n_D = 64$ achieves 31% lower latency than EE, i.e., a dedicated hardware solution. Sealer shows 18% lower latency because it stores fewer data blocks ($n_D = 51$) in an equal-sized subarray with low utilization. For $n_D = 32$, our architecture achieves 65% and 41% lower latency than EE and Sealer, respectively. Using simultaneous read and write of compact eDRAM cells, our approach proves to be fast even at lower operating frequencies than others. For GCM and XTS modes, our design consistently shows lower latency than most compared implementations. However, [8] achieves the lowest latency in GCM mode but with significant power consumption.

3) Power Consumption: Fig. 9 compares the power consumption. In ECB mode, our design ($n_D = 64$) consumes 14% and 58% less power than EE and Sealer, respectively. This reduction highlights the improvement in the power efficiency of eDRAM subarrays as n_D increases due to refresh-free operation. For GCM mode, our architecture with $n_D = 64$ consumes 73% less power than [8], which consumes large power during the data transfer. In XTS mode, our design consumes less power compared to our own ECB implementation.

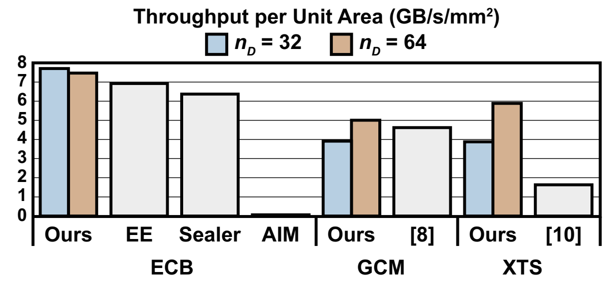


Fig. 10. Comparison on throughput per unit area among various architectures.

4) Efficiency: Fig. 10 presents the throughput per unit area. Our design demonstrates higher area efficiency across all modes. In ECB mode, it outperforms EE and Sealer by 8% and 17%, respectively. This indicates the effectiveness of eDRAM-based PNM in maximizing computational density. FlexENM achieves 8% and 261% higher throughput per unit area compared to the prior work on GCM and XTS modes, respectively. This consistent efficiency benefit at various modes emphasizes the versatility of FlexENM, which leverages the compact size of eDRAM cells and its parallel architecture.

V. Conclusion

In this paper, we presented an on-chip eDRAM-based encryption engine, named FlexENM, supporting multiple AES modes, i.e., ECB, XTS, and GCM. The FlexENM features compact eDRAM subarrays, simultaneous read and write support for improved AES performance, and near-memory compute logics for supporting various AES modes. To mitigate the overhead of refresh operations in eDRAM cells, we proposed to traverse all data blocks in each AES step so that eDRAM cells naturally refresh their contents while encrypting. The FlexENM equipped with these features outperformed state-of-the-art AES accelerators in terms of latency, power consumption, and area efficiency at each AES mode. Last but not least, we explored the operating conditions of FlexENM to help determine the optimal configuration at various constraint scenarios.

Acknowledgment

This work was partially supported by Korea Research Institute for defense Technology planning and advancement (KRIT) - grant funded by the Defense Acquisition Program Administration (DAPA) (KRIT-CT-22-078), and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (2022-0-01170, PIM Semiconductor Design Research Center). The EDA tool was supported by the IC Design Education Center (IDEC) in South Korea.

References

[1] J. Zhang *et al.*, "Sealer: In-SRAM AES for high-performance and low-overhead memory encryption," in *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2022, pp. 1–6.

[2] M. Xie *et al.*, "AIM: Fast and energy-efficient AES in-memory implementation for emerging non-volatile main memory," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 625–628.

[3] M. J. Dworkin, "Recommendation for block cipher modes of operation: Galois/Counter mode (GCM) and GMAC," National Institute of Standards and Technology (NIST), Special Publication 800-38D, 2007.

[4] ——, "Recommendation for block cipher modes of operation: The XTS-AES mode for confidentiality on storage devices," National Institute of Standards and Technology (NIST), Special Publication 800-38E, 2010.

[5] M.-T. Chang *et al.*, "Technology comparison for large last-level caches (L^{3}Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM," in *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 143–154.

[6] D. Lee *et al.*, "An off-chip attack on hardware enclaves via the memory bus," in *USENIX Security Symposium (USENIX Security 20)*, 2020.

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

[8] M. Mozaffari-Kermani *et al.*, "Efficient and high-performance parallel hardware architectures for the AES-GCM," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1165–1178, 2011.

[9] S. K. Mathew *et al.*, "53 Gbps native GF(2^4)^2 composite-field AES-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 46, no. 4, pp. 767–776, 2011.

[10] A. P. Kakarountas *et al.*, "A survey on throughput-efficient architectures for IEEE P1619 for shared storage media," in *IEEE Symposium on Computers and Communications (ISCC)*, 2011, pp. 758–763.

[11] Y. Wang *et al.*, "FPGA-based high throughput XTS-AES encryption/decryption for storage area network," in *International Conference on Field-Programmable Technology (FPT)*. IEEE, 2014, pp. 268–271.

[12] K. M. Abdellatif *et al.*, "AES-GCM and AEGIS: Efficient and high speed hardware implementations," *Journal of Signal Processing Systems*, vol. 88, pp. 1–12, 2017.

[13] N. Muralimanohar *et al.*, "Cacti 6.0: A tool to model large caches," *HP Laboratories*, vol. 27, p. 28, 2009.