IMCRYPTO: An In-Memory Computing Fabric for AES Encryption and Decryption

Dayane Reis, Student Member, IEEE, Haoran Geng, Michael Niemier, Senior Member, IEEE and Xiaobo Sharon Hu, Fellow, IEEE

Abstract—This paper proposes IMCRYPTO, an in-memory computing (IMC) fabric for accelerating AES encryption and decryption. IMCRYPTO employs a unified structure to implement encryption and decryption in a single hardware architecture, with combined (Inv)SubBytes and (Inv)MixColumns steps. Because of this step-combination, as well as the high parallelism achieved by multiple units of random-access memory (RAM) and random-access/content-addressable memory (RA/CAM) arrays, IMCRYPTO achieves high throughput encryption and decryption without sacrificing area and power consumption. Additionally, due to the integration of a RISC-V core, IMCRYPTO offers programmability and flexibility. IMCRYPTO improves the throughput per area by a minimum (maximum) of 3.3× (223.1×) when compared to previous ASICs/IMC architectures for AES-128 encryption. Projections show added benefit from emerging technologies of up to 5.3× to the area-delay-power product of IMCRYPTO.

Index Terms—Advanced Encryption Standard, AES, In-Memory Computing, IMC, Random Access Memory, RAM, Content Addressable Memory, CAM

I. INTRODUCTION

Advanced Encryption Standard (AES) [1] is a widely used encryption method used in a variety of computer systems. Despite the excellent security of AES, its software-based implementations can be computationally expensive, resulting in low throughput. High throughput AES encryption/decryption is extremely desirable in many applications (e.g., virtual private networks, electronic financial transactions, etc.). As demonstrated in previous work (e.g., [2]–[7]), hardware accelerators designed for AES encryption/decryption can achieve high throughput while enabling area-efficient design alternatives for resource-constrained environments such as edge computing. AES hardware accelerators are usually application-specific integrated circuits (ASICs) or field-programmable gate array (FPGA)-based co-processors that implement the steps for the AES algorithm, i.e., AddRoundKey, SubBytes/InvSubBytes, ShiftRows, and MixColumns/InvMixColumns.

The core functions in AES involve mainly simple operations, which makes AES encryption/decryption memory bound. To address the memory bottleneck of AES, in-memory computing (IMC) architectures (i.e., with modified memory cells or peripherals) may be designed to perform operations commonly found in AES-based encryption/decryption (e.g., bit shifts, byte permutations and XOR). IMC-based architectures operate on memory words without the need for data transfers to a processing unit (e.g., [6], [7]). Despite the potential for reduced data traffic, work in [6], [7] must make tradeoffs between area/power efficiency and parallelism. To this end, accelerators based on lookup table (LUT) and pre-computed operations (e.g., [2], [3]) have demonstrated high levels of parallelism and throughput. In this regard, memories based on emerging technologies (e.g., phase-change memories (PCMs), resistive random-access memories (RRAMs), spin-transfer torque magnetoresistive random-access memories (STT-MRAMs), ferroelectric field-effect transistor-based random-access memories (FeFET-RAMs) can considerably improve the density and static power consumption of IMC architectures (albeit at the expense of longer write latencies) [8]–[10].

Besides architecture and technology considerations, security and flexibility in supporting different modes of AES are desirable features of hardware accelerators. For the former, the ability to run new encryption/decryption algorithms that resist emerging attacks is highly desirable [11]. For the latter, different modes of AES may be selected for different applications, e.g., cipher block chaining (CBC) and the counter (CTR) mode can be used for data streaming and for protecting data storage devices [12]–[14]. As another example, authenticated encryption guarantees both confidentiality (data secrecy) and integrity (data authenticity), which has utility in distinct application spaces and can be performed with Galois/Counter Mode (GCM). Considering this wide range of usage scenarios, having an AES accelerator that supports different modes of AES may reduce costs with system/intellectual property (IP) re-design. As an additional benefit, a general-purpose, programmable accelerator for AES could accommodate other symmetric block ciphers and hashing functions such as the Secure Hash Algorithm (SHA)-256 or the Message Digest Algorithm (MD)-5.

In this paper, we propose a programmable in-memory computing fabric for AES acceleration (IMCRYPTO) that achieves the same parallelism as LUT methods while avoiding large area/power overheads. IMCRYPTO is based on both conventional and compute-enabled random-access memory (RAM) arrays [15], as well as dual-function random access memory/content addressable memory (RA/CAM) arrays. Specifically, the RA/CAM arrays employed in IMCRYPTO enable encryption and decryption to be performed with a common structure. To improve the computational efficiency, this work was supported in part by ASCENT, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. Date of publication xx xx, xxxx; date of current version xx xx, xxxx.

Corresponding author: Xiaobo Sharon Hu.

D. Reis, H. Geng, M. Niemier, and X. S. Hu are with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, 46556, USA. e-mail: shu@nd.edu.
ciency, the steps SubByte/MixColumns (for encryption), and InvSubBytes/AddRoundKey/InvMixColumns (for decryption) are combined and executed in one step with our design. The proposed fabric supports pipelining and enables high-throughput execution. The contributions of our work are summarized below:

- We propose IMCRYPTO, a highly-parallel IMC fabric based on RAM and RA/CAM arrays that provide high-throughput AES encryption/decryption with a common hardware structure.
- We propose a compact implementation of the AES algorithm with IMCRYPTO, which combines certain steps in encryption (and decryption) to reduce computation load.
- We propose a RISC-V based controller that implements customized instructions in IMCRYPTO. The flexibility of IMCRYPTO enable multiple modes of AES and other block ciphers to be implemented.

We have performed detailed circuit-level, simulation-based evaluations of our proposed IMCRYPTO architecture. We consider figures of merit such as delay, area, power, throughput, area-delay-power product (ADPP), and throughput per area and compare them with state-of-the-art ASIC and IMC-based accelerators for AES encryption [4]–[7]. Results indicate minimum (maximum) throughput per area improvements of 3.3× (223.1×) and minimum (maximum) ADPP improvements of 1.2× (6,238.3×) for CMOS-based IMCRYPTO when compared to ASIC and IMC accelerators for AES-128 encryption of a 1MB plaintext. (The wide spectrum of improvement is a result of different circuit and design paradigms employed in each accelerator). IMCRYPTO can also leverage emerging memory technologies to further boost the ADPP improvements over a conventional CMOS-based implementation. Our analysis suggests that employing emerging technologies in the design of memory units in IMCRYPTO could improve the ADPP by an additional factor of up to 5.3×.

II. BACKGROUND AND RELATED WORK

In this section, we briefly review the basics of AES encryption/decryption, and discuss related work on hardware accelerators for AES encryption/decryption.

A. AES Basics

Fig. 1(a) and Fig. 1(b) illustrate the steps performed in $N$ rounds of AES encryption and decryption. As a variant of the Rijndael block cipher, AES employs a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits (for AES-128, AES-192, or AES-256, respectively). Round keys are derived from the main key for each one of the $N$ rounds using a key schedule algorithm [1].

The 128-bit block is arranged as a $4 \times 4$ array of bytes, which is the state array. Each step in the AES encryption (decryption) takes a state array as an input and produces an output state array that is passed to the next step (or returned as the final encryption or decryption result at the end of $N$ rounds). We represent elements of the input (output) state array as $a_{i,j}$ ($b_{i,j}$), where $i = 0, 1, 2, 3$ and $j = 0, 1, 2, 3$. The steps of AES encryption (decryption) are summarized below.

1) AddRoundKey: In the AddRoundKey step, every byte of a round key $r_k$ is combined to the corresponding byte of the input state through an XOR operation, i.e., $b_{i,j} = a_{i,j} \oplus r_{k_{i,j}}$.

2) SubBytes and InvSubBytes: During SubBytes and InvSubBytes, each byte $a_{i,j}$ in the input state array is replaced by a substitution byte to form the output state array, i.e., $b_{i,j} = Sbox(a_{i,j})$. Since each byte $a_{i,j}$ in the 0x00–0xFF interval maps to an unique substitution byte, encryption/decryption $Sbox$ tables can be pre-computed and stored in LUTs (or memory units) to improve the speed of AES. While pre-stored table-based implementations of the SubBytes and InvSubBytes steps may be vulnerable to cache timing attacks in a traditional central processing unit (CPU) architecture, they are well-suited for FPGA or ASIC-based accelerator designs [16].

3) ShiftRows and InvShiftRows: During ShiftRows and InvShiftRows, the bytes of the input state array are circularly shifted to the left (right) by a fixed offset amount (0, 1, 2 or 3 positions) depending on the index $i$ of the byte. The output state is then formed by the byte-shifted input state. For $i = 0$, no circular byte shifts are performed ($b_{i,j} = a_{i,j}$). For $i = 1$, 2, and 3, the offsets of the circular byte shifts are defined as 1, 2, and 3, respectively, with a left (right) direction for encryption (decryption).

4) MixColumns and InvMixColumns: During MixColumns and InvMixColumns, a linear transformation is applied to the input state matrix to form the output state matrix. This transformation consists of a matrix multiplication over the Galois Field ($GF(2^8)$) between a fixed matrix and the input state matrix (see Fig. 2). As with SubBytes (InvSubBytes), it is possible to speed-up the MixColumns (InvMixColumns) steps by pre-computing the products between every byte value in the range 0x00–0xFF and the multiplication coefficients.

5) Modes of Operation: Modes of operation ensure confidentiality and authenticity when a cipher's single-block operation encrypts/decrypts an amount of data larger than a 128-bit block. Different modes of operation are commonly used in addition to the basic version of the AES algorithm, a.k.a. the

![Fig. 1: The steps performed in the $N$ rounds of AES (a) encryption and (b) decryption. The values of $N$ are 10 (for AES-128), 12 (for AES-192), and 14 (for AES-256). (Inv)ShiftRows and (Inv)SubBytes are commutative steps [1].](image-url)
ECB (Electronic Code Book). For instance, the CBC, the CFB (Cipher FeedBack), the OFB (Output FeedBack), the CTR, and the GCM are AES modes that require simple operations (such as XOR or addition) to be performed between the state array and randomly-generated numbers (or counter values) at the start of every AES round. (For details on different modes of AES and their practical use, see [17], [18]).

B. Related Work

Numerous research efforts have been devoted to developing FPGA or ASIC-based hardware to accelerate AES (e.g., [2]–[7], [19], [20]). That said, designing a hardware accelerator for AES that can provide high throughput without large overheads in terms of area and power consumption can be a challenging task. Area-efficient AES accelerators that provide high throughput are highly sought-after intellectual property (IP) blocks for resource-constrained environments (e.g., in embedded systems [21]).

One of the strategies to minimize area without compromising performance is to re-use circuit blocks and to design shared datapaths for encryption and decryption [4], [22]. In this regard, [4] introduces an architecture for AES with shared encryption/decryption datapaths. Although their shared-datapath strategy can help reduce area, some circuits used for encryption and decryption still need to be implemented separately in this architecture. Furthermore, the design proposed in [4] is a standalone ASIC, which requires plaintexts and ciphertexts to be fetched/stored in memory units. As we show in our evaluation (see Sec. VI), the latency and energy overheads from data transfers can significantly impact the performance of hardware accelerators for AES.

Reducing the impact of data transfers on a system’s performance is a desired feature for AES accelerators. To this end, IMC could potentially alleviate the burden of data transfers between the memory and the accelerator. Reference [6] proposes a reconfigurable cryptographic processor that uses IMC and near-memory computing for supporting AES and elliptic curve cryptography (ECC). Even though their IMC-based design improves the performance, it lacks the flexibility of supporting multiple modes of AES and other cryptographic algorithms (e.g., SHA-256). In this regard, reference [5] proposes a highly reconfigurable ASIC cryptographic processor. It can support most of the existing cryptographic algorithms and potentially can support future algorithms. Although their throughput per area is much higher than the best CPU and graphic processing unit (GPU), they do not account for data movement in their evaluation (similar to [4]).

Reference [7] presents another IM accelerator for AES based on PCMs, which is intended to protect the contents of the non-volatile, off-chip main memory in case an attacker gets physical access to it. The accelerator can simultaneously encrypt plaintext within each memory bank, and the entire encryption process can be completed without exposing the results to the memory bus. However, in a usage scenario where this IM accelerator is used to perform decryption and send the resulting plaintext to the CPU, the architecture may be vulnerable to side channel attacks as plaintext sent from the off-chip memory would be exposed to the memory bus. The design also requires expensive writes to the non-volatile PCM (including results from the SubBytes and MixColumns steps of AES), which is a concern due to the high write latency and high write energy of the PCM memory [8].

III. IMCRYPTO FABRIC

We propose the IMCRYPTO fabric for AES encryption/decryption. Our proposed in-memory accelerator is placed on the same chip as the CPU, at the level of the last level cache (LLC). Placing IMCRYPTO on-chip and at the level of the LLC facilitates data exchange with the CPU and creates a mechanism for storing ciphertexts inside IMCRYPTO’s internal memory (i.e., IMCRYPTO works as the LLC for secure data). Furthermore, data is properly protected before being sent to the main memory through the memory bus. IMCRYPTO employs a light-weight, general-purpose core to program IMCRYPTO’s functionality. For simplicity, we use a RISC-V core in IMCRYPTO, but other cores can also be designed for this purpose. Thanks to the programmability enabled by the RISC-V core, IMCRYPTO offers the needed flexibility at the application level (as mentioned in Sec. I).

Note that using IMCRYPTO as the LLC for secure data should not significantly impact the performance of the main CPU for executing non-secure workloads, since the L1/L2
caches can have their content accessed directly by the RISC-V core without going through IMCRYPTO. Furthermore, with IMCRYPTO working simultaneously as an accelerator for AES encryption/decryption and the LLC for secure data, the L1 and L2 caches (represented in Fig. 3) are reserved to the storage of non-sensitive data. Thus, they can freely transfer data from/to the main memory through the bus and from/to the RISC-V core without any negative impact to the security.

Fig. 3 provides an overview of the IMCRYPTO-based system, where the interaction between IMCRYPTO and the other components of a computer system (i.e., the main memory and L1/L2 caches) is explicitly shown. Encryption/decryption requests to IMCRYPTO can be made by the RISC-V core, which communicates with IMCRYPTO through a set of control (CTRL) signals. The RISC-V uses the customized instruction “TEXT” (to be described in Sec. III-D, Table Ia) to pass the address of a plaintext (PT) or a ciphertext (CT) for encryption/decryption with IMCRYPTO. Round keys are generated on-the-fly by the RISC-V core and directly provided to IMCRYPTO. Below, we describe how IMCRYPTO processes encryption/decryption requests by the RISC-V core:

- **At encryption**, the instruction “TEXT” is used to pass the data address of a plaintext (PT) to IMCRYPTO. The PT is fetched to (and temporarily stored in) the internal memory of IMCRYPTO from either the L2 cache (in case of a cache hit) or the main memory (in case of a cache miss). After the data transfer process, the PT is encrypted by the IMC infrastructure in IMCRYPTO, generating a ciphertext (CT). Once encryption finishes, the post-encryption PT is not kept in the internal memory of IMCRYPTO (which is reserved for secure data), but rather is transferred to the CPU’s L2 cache to be used by processes running on the CPU.

The IMCRYPTO fabric (Fig. 4(a)) consists of circuits that may be grouped into four distinct blocks. The blocks, numbered from 1 to 4 in Fig. 4(a), are (1) the compute-enabled memory (CEM), (2) the bi-directional shifter, (3) the LUT fabric (comprised of 4 modules), and (4) the RISC-V based controller. The circuits in each block as well as the communication between them are described below.

### A. The Compute-Enabled Memory

The AddRoundKey step of AES encryption (decryption) is the addition of a round key to a plaintext (ciphertext) through a simple XOR operation. **To enable the AddRoundKey step in AES encryption (and in the initial round of AES decryption) the proposed architecture implements the compute-enabled memory (CEM) block (block 1 in Fig. 4(a)).** The CEM of IMCRYPTO employ circuits similar to those in [15], [23] — i.e., customized sense amplifiers — which enable bitwise Boolean logic between two memory words stored in a memory array.

The CEM receives plaintext from the CPU L2 cache, performs logic operations on the AES state array, and stores the resulting ciphertext after encryption. The encrypted data in IMCRYPTO can be directly read from the CEM by the CPU cache structure and sent to external parties using communication protocols like the peer-to-peer (P2P) or Attribute-Based Encryption (ABE) (used in IoT systems). Another usage scenario for direct access to the CEM is when the CPU core performs operations on data that has been previously encrypted and stored in IMCRYPTO. In this case, IMCRYPTO allows for the data to be decrypted prior to sending it to be used by the processes running on the CPU core.

As an on-chip memory, the size of a CEM needs to be within the acceptable limits for a CPU cache (i.e., from tens of KB up to a few MB). A larger CEM enables more
ciphertexts to be stored in IMCRYPTO, which can be useful when cached ciphertexts are sent to external parties using communication protocols. On the other hand, bigger memories have longer access times and consume more power. To allow for a compromise between memory size, access times and energy consumption, the CEM block of IMCRYPTO is a 1 MB memory that consists of a tiled, compute-enabled SRAM structure [15]. With the aid of customized sense amplifiers, the CEM can execute XOR operations between two memory words without the need for reading the data out to an external processing unit.

B. The Bi-Directional Shifter Block

The bi-directional shifter block (labeled as block 2 in Fig. 4(a)) performs the byte permutations needed by the ShiftRows (InvShiftRows) steps of AES encryption (decryption). Internally, the circuit of the bi-directional shifter block is similar to that described in [20].

C. The LUT Fabric

The LUT fabric (labeled as block 3 in Fig. 4(a)) is a critical block in IMCRYPTO as it executes the SubBytes/InvSubBytes and MixColumns/InvMixColumns (the most time-consuming steps of AES). With the use of small memory elements (i.e., 6T-SRAM and RA/CAM arrays of size 256×8) and customized peripherals, it is possible to perform these steps in a highly parallel and combined fashion with low latency. The size of the memory elements (i.e., 256×8) is chosen so it is possible to store all the possible 256 pre-computed bytes of combined SubBytes and MixColumns.

The LUT fabric block comprises 4 LUT fabric modules that execute 1 matrix multiplication in GF(2^8). These four identical LUT fabric modules in the LUT fabric block operate in parallel to compute each column of the output state array (i.e., j = 0,1,2,3) in the MixColumns/InvMixColumns and SubBytes/InvSubBytes steps. The internal structure of a LUT fabric module is depicted in Fig. 4(b). Each LUT fabric module consists of (i) 4 RA/CAM arrays with customized peripherals, (ii) 8 RAM arrays, and (iii) 2 sets of cascaded XOR gates (a.k.a. XOR trees, which are employed to perform additions over GF(2^8)). The number of memory elements of each type inside a LUT fabric module allows combined steps to be performed without serializing operations. More details about the execution of combined steps will be given in Sec. IV-C.

D. RISC-V based Controller

Encryption and decryption in AES requires that certain computational steps are performed in a certain order over N rounds (see Fig. [1]). Moreover, different modes of AES have different data flows and may require additional operations. IMCRYPTO employs a RISC-V based controller (block 4 in Fig. 4(a)) to perform the different steps of AES and support different modes of operation. Additionally, the RISC-V based controller makes it possible to extend the use of the IMCRYPTO fabric for other cryptographic algorithms that require similar operations as AES.

| Instruction | Function Code | Assembly | Description |
|-------------|---------------|----------|-------------|
| TEXT L      | 0000000       | TEXT s1  | Load 128 bits from IMCRYPTO to RISC-V registers during encryption. |
|             | 1             | s1       | Store 128 bits to IMCRYPTO from RISC-V registers during encryption. |
| SFTR E      | 0001000       | SFTR s1 | Trigger ShiftRows on s1 (encryption) |
|             | 1             | s1       | Trigger InvShiftRows on s1 (decryption) |
| SUBMX E     | 0001000       | SUBMX s1| Trigger combined SubBytes+MixColumns (encryption) on s1 |
|             | 1             | s1       | Trigger combined InvSubBytes+InvMixColumns (decryption) on s1 |
| SBOX E      | 0010000       | SBOX s1 | Trigger SubBytes (last round of encryption) on s1 |
|             | 1             | s1       | Trigger InvSubBytes (last round of decryption) on s1 |

(a) RV32I I-type Instructions for IMCRYPTO

(b) RV32I R-type Instructions for IMCRYPTO

To support IMCRYPTO operations, we introduce 8 RV32I I-type (Table 1a) and 10 RV32I R-type (Table 1b) customized instructions in the RISC-V instruction set architecture (ISA). Two RV32I I-type instructions are used to load/store from/to IMCRYPTO’s CEM (discussed in Sec. III-A) to/from registers in the RISC-V core. The rest of the I-type instructions are used to trigger each step in AES encryption/decryption. After
decoding these customized I-type instructions, the RISC-V based controller generates three main inputs to each block of IMCRYPTO—1 enable (EN) bit, 1 memory address (ADDR), and 1 piece of data (4 registers of 32 bits each). When IMCRYPTO finishes executing an instruction, it sends back a “DONE” signal to the RISC-V based controller, which then fetches the next instruction.

The 10 RV32I R-type customized instructions are used to trigger general-purpose bitwise and arithmetic in-memory operations, such as IMAND, IMOR, IMXOR, IMADD, etc. inside the CEM in IMCRYPTO. These instructions are used to realize the arithmetic and logic operation inside the memory (IMXOR is also used to achieve the AddRoundKey step of AES). As the in-memory customized R-type instructions are decoded, 1 enable bit (EN), 2 source memory addresses and 1 destination memory address (ADDR) are generated. The CEM operates on the two source memory addresses specified by instructions, performs the corresponding in-memory operation, and stores the result in the destination memory address. The aforementioned control signals (for both RV32I I-type and R-type instructions) are showed in Fig. 4(a).

We use the opcodes 0000111 and 1000111 to differentiate the regular RISC-V instructions from our customized instructions I-type and R-type. Furthermore, as IMCRYPTO’s instructions are executed in the CEM, the arithmetic logic unit (ALU) and memories inside the RISC-V processor can be simultaneously used to execute regular RISC-V instructions running in different threads. For example, IMCRYPTO can use one RISC-V thread to receive data. By employing this multi-threaded technique, the RISC-V core acts as a controller for our IMCRYPTO accelerator, while it is also available to be used as a general purpose processor.

By supporting different in-memory operations via the customized RISC-V R-type instructions, IMCRYPTO can achieve various modes of operation for AES without any hardware modifications. For instance, the CTR mode of AES leverages a counter value which can be stored inside the CEM of IMCRYPTO. The counter can then be implemented with the use of IMADD operations. As another example, the CBC mode of AES for processing streaming data can be achieved by an additional IMXOR operation between the output values of two consecutive AES 128-bit blocks.

In addition to the different modes of AES, IMCRYPTO can also support other block ciphers and hash algorithms such as Blowfish, CAST-128, SHA-256 and MD5, which has similar computation operations and complexity as AES. By implementing these algorithms in IMCRYPTO, we can take advantage of high parallelism, high throughput, low data transfer time, and low energy consumption offered by IMC. In Sec. V, we present a use case for an IMCRYPTO-based implementation of the SHA-256 hashing algorithm.

### IV. RA/CAM DESIGN

Here, we introduce the RA/CAM design in IMCRYPTO. Fig. 5(a) provides a high-level view of the components of the RA/CAM design: (1) the 256×8 RA/CAM array, (2) the row decoder, (3) the RAM sense amplifiers (SAs), (4) the search drivers, (5) the CAM sense amplifiers (SAs), and (6) the customized encoder.

Our RA/CAM design is based on complementary metal–oxide–semiconductor (CMOS) static random-access memory (SRAM) memory cells. While 6T-SRAM cells can be employed for the design of RAM modules (e.g., [24]), our RA/CAM cell design employs 9 transistors (similar to the cell proposed in [19]). Although [19] emphasizes the usage of 9T-SRAM cells in the CAM setting only, in IMCRYPTO we introduce peripheral circuits (i.e., a row decoder and RAM sense amplifiers) that enables the array to work as both a RAM or a CAM (hence, a RA/CAM array is comprised of 9T-RA/CAM cells). The CAM functionality of the RA/CAM array enables both the SubBytes and InvSubBytes steps to be performed with a common structure for encryption and decryption.

Fig. 5: (a) A high level view of the RA/CAM array. (b) The schematic view of 1 column of the array (with 1 RA/CAM cell included).
At this point, if the logic state stored in SL (through n) is equal to the logic state applied to the CAM SA (i.e., OUT CAM SA = 0V), the ML is discharged, producing a mismatch at the output of the CAM SA (i.e., OUT CAM SA = 0V).

C. Step Combination in IMCRYPTO

To maximally exploit IMC and reduce computation cost, we leverage a step combination technique in IMCRYPTO. Specifically, we store the values of $1 \times \text{sbox}(a_{i,j})$ in the 4 RA/CAM arrays, and the values of $2 \times \text{sbox}(a_{i,j})$, and $3 \times \text{sbox}(a_{i,j})$ in the 8 RAM arrays of 1 LUT fabric module (instead of simply storing $\text{sbox}(a_{i,j})$). These values are needed by the matrix multiplication operation in the MixColumns step, hence storing them directly in the RA/CAM arrays enables us to easily combine the SubByte and MixColumns steps of AES encryption. The InvSubBytes, AddRoundKey, and InvMixColumns steps of AES decryption are also executed in a combined fashion, with the modification of peripheral circuits (i.e., the customized encoder) of the RA/CAM array.

1) Encryption: Fig. 6(a) illustrates the execution of the SubBytes and MixColumns in a LUT fabric module, with the RAM arrays and RA/CAM working in the RAM mode. Bytes $a_{i,j}$ in the same column of the input state array (i.e., bytes that have the same j index) are given as memory addresses to the row decoders of the 8 RAM arrays and 4 RA/CAM arrays in a transposed fashion. In the example of Fig. 6(a), we use $j = 0$. A pre-computed byte in each array is read out by the RAM SA at the given address. Once the bytes are read out from the RA/CAM arrays, they are XORed by the XOR tree to produce the result of the matrix multiplication in the MixColumns step for the column j in the output state array.

2) Decryption: To avoid unnecessary power and delay overheads, IMCRYPTO does not require RA/CAM arrays to be re-programmed when the accelerator switches its functionality from encryption to decryption. Therefore, during decryption, the 4 RA/CAM arrays of 1 LUT fabric module contain the same values of encryption (i.e., $1 \times \text{sbox}(a_{i,j})$). The RAM arrays in the LUT fabric module are used in AES decryption and can be turned off.

Fig. 6(b) depicts IMCRYPTO’s approach for combining steps in AES decryption. IMCRYPTO performs InvSubBytes in the periphery of the RA/CAM array operating in CAM mode (i.e., with customized encoders). Customized encoders allow the execution of InvSubBytes to be combined with AddRoundKey and InvMixColumns to enable efficient AES...
decryption. We divide the execution of combined steps in AES decryption with IMCRYPTO into stages. Stage 1 correspond to the operations of InvSubBytes, and leverages the invertible nature of the Sbox function to perform the SubBytes and InvSubBytes with distinct modes of a RA/CAM array. Stage 2 produces the result of AddRoundKey. Finally, stage 3 computes the multiplication of InvMixColumns.

In the example of Fig. 6(b), bytes $a_{i,j}$ with the same $j$ index are given as search data to the search drivers of the 4 RA/CAM arrays in a transposed fashion (similar to encryption). Although we illustrate only the operations in the last RA/CAM arrays in our LUT fabric module (i.e., where the search data is $a_{3,0}$), identical operations occur simultaneously in the other columns of arrays $(a_{0,0}, a_{1,0},$ and $a_{2,0})$. In stage 1 of our approach, we search for the byte $a_{3,0}$ in the RA/CAM arrays. The CAM SAs detect a single match for each array (i.e., while 1 row of the RA/CAM array produces a match, 255 rows produce mismatches). Since the RA/CAM array stores $1 \times sbox(a_{i,j})$, the row which returned a match is encoded into a byte as the equivalent result of the InvSubBytes step.

In stage 2, the customized encoder simply performs an XOR operation between each InvSubBytes result and $rk_{0,0}$ (i.e., a byte of the round key) in order to produce the result of the AddRoundKey step. In stage 3, the 8-bit result of the AddRoundKey step is multiplied by 9, 11, 13, and 14. This multiplication can be done with combinational logic inside the customized encoder that maps one byte value into another.

Similar to encryption, during decryption the 4 8-bit outputs of the customized encoder at stage 3 are XORed with the other outputs of the first row of RA/CAM arrays in the LUT fabric module (i.e., those where search data inputs were $a_{1,0}$, $a_{2,0}$, and $a_{3,0}$) with the XOR tree. The final result, which corresponds to the output of the InvSubBytes step, is forwarded to the next block in the AES decryption datapath.

V. A USE CASE OF IMCRYPTO BEYOND AES

The distinct security needs by various applications, in addition to the development of new algorithms for handling emerging attacks make it highly desirable for crypto engines to be flexible. Furthermore, configurable crypto engines that can “guarantee interoperability between countries and institutions”, are highly sought-after intellectual property (IP) blocks [11]. In IMCRYPTO, the use of a RISC-V co-processor allows for basic in-memory operations to be executed in an arbitrary order, which enables support for different modes of AES and other encryption/decryption algorithms beyond AES (e.g., hashing functions and other symmetric block ciphers). In this section, we provide an example for implementing SHA-256 on our IMCRYPTO fabric.

SHA-256 is a member of SHA-2 — a group of cryptographic hash functions designed by the National Security
Agency (NSA). Widely known for its high security and speed, SHA-256 is used in the block chain industry and encryption communication protocols. The SHA-256 algorithm can be divided into two parts: message padding (Algorithm 1) and main hashing function (Algorithm 2). While the former extends a 16-bit word message to a 64-bit word message, the latter generates fixed-sized outputs (i.e., hashes). Message padding uses the $\sigma_0$ and $\sigma_1$ functions for padding (lines 2 and 3 in Algorithm 1). $\sigma_0$ and $\sigma_1$ are implemented with bitwise “right-rotate” and “right-shift” operations, as defined below:

$$\sigma_0(A) = (A >>> 7) \oplus (A >>> 18) \oplus (A >> 3)$$

$$\sigma_1(A) = (A >>> 17) \oplus (A >>> 19) \oplus (A >> 10)$$

IMCRYPTO uses the customized instructions IMCSL, IMCSR, IMXR, IMXOR and IMADD to perform message padding (Fig. 7(a)). For instance, to implement rotations inside IMCRYPTO, the data are accessed through their memory addresses. Then, this data is read through the customized sense amplifier of the CEM and sent to hardware blocks that implement in-memory bit shifters. Fixed shift amounts are used with the bit shifters in order to slide the data bits to specific positions as required by the rotation operation.

The main hashing function (Algorithm 2) is executed after message padding and produces the final output (i.e., a set of 8, fixed-sized hashes). The main hashing function in SHA-256 takes 64 rounds to complete its execution. It is comprised of two functions — “majority” and “choice”:

$$\text{Majority}(A, B, C) = A \cdot B \oplus A \cdot C \oplus B \cdot C$$

$$\text{Choice}(A, B, C) := A \cdot B \oplus \overline{A} \cdot C$$

Each of these functions take 3 inputs A, B, and C, and performs computation as follows: (1) The “majority” function outputs ‘0’(‘1’) when half or more of its inputs are ‘0’(‘1’). (2) The “choice” function uses input A as a selector bit, i.e., when $A = '1'(0)$, the function outputs the value of ‘B’(‘C’). The main hashing function uses the $\Sigma_0$ and $\Sigma_1$ functions to hash the data, which can be implemented with bitwise “right-rotate” operations:

$$\Sigma_0(A) = (A >>> 2) \oplus (A >>> 13) \oplus (A >>> 22)$$

$$\Sigma_1(A) = (A >>> 6) \oplus (A >>> 11) \oplus (A >>> 25)$$

Besides these functions, 64 constants + 8 initial hashes (A – H) are needed. IMCRYPTO pre-stores these constants and the initial hashes inside the CEM, and uses the customized instructions IMCSR, IMXR, IMXOR, IMOR, IMNOT, IMADD and IMMOVE (Table II) to perform the operations in the main hashing function.

VI. EXPERIMENTAL EVALUATION

Here, we compare figures-of-merit (FoM) (i.e., delay, throughput, area, power, throughputs per area, and ADPP) of IMCRYPTO with accelerators for AES encryption/decryption from previous works [4–6].

TABLE II: Accelerators Considered in the Evaluation

| Design | Type    | Technology       | Supported Algorithms                          | Reference |
|--------|---------|------------------|----------------------------------------------|-----------|
| A      | ASIC    | CMOS 45 nm       | AES encryption/decryption                     | [4]       |
| B      | ASIC    | CMOS 45 nm       | AES encryption+Most symmetric ciphers         | [5]       |
| C      | IMC     | CMOS 40 nm       | AES encryption,Keccak, finite field multiply  | [6]       |
| D      | IMC     | PCM 65 nm        | AES encryption/decryption                     | [7]       |
| E      | IMC     | CMOS 45 nm       | AES encryption/decryption*                    | IMCRYPTO |

Notes: *IMCRYPTO can potentially support other symmetric ciphers by manipulating its custom instructions/reordering the steps executed in the IMC architecture.

A. Experimental Setup

In our evaluation (and comparison), we implement AES-128 encryption (ECB mode). The implementation of the IMCRYPTO instructions in the RISC-V based controller and their sequencing for AES-128 ECB mode is done in Verilog and evaluated through simulations. At the circuit level, we use the DESTINY simulator [25] to measure area, latency and energy for a 1MB compute-enabled SRAM, with a 2-Mat organization and a subarray size of 128×128. We have modified DESTINY to support the customized sense amplifiers and local write buffers of [15]. The compute-enabled SRAM is based on a 45nm CMOS predictive technology model (PTM) [27]. We choose the 45nm technology node as most designs in our comparison use this node. To estimate the area overhead introduced by compute circuits in the SRAM, we subtract the area of a baseline SRAM of 1 MB based on same technology node from the area of the compute-enabled RAM. We use DESTINY to simulate the baseline SRAM, which is a regular SRAM memory without customized peripherals.

The area, latency, and energy of the bi-directional shifter block are measured through synthesis using the Cadence Encounter RTL Compiler v14.10, with the NanGate 45nm open-cell library [28]. Finally, the 256×256 RA/CAM arrays in IMCRYPTO’s LUT fabric (block 4 of Fig. 4(a)) are simulated using HSPICE version O-2018.09-1 with the PTM for 45nm CMOS [27] and $V_{DD} = 1V$. We measure the latency and energy for the different types of memory accesses, i.e., reads/writes in RAM mode and searches in CAM mode. In order to estimate the area of the 256×256 RA/CAM array, we employ the OpenRAM memory compiler [29], in which we specify the 6T-SRAM and 9T-RA/CAM cells as well as the peripherals of our design. The area of the IMCRYPTO fabric (which we compare with the area of other accelerators in Sec. VI-B) includes the area overhead of compute-enabled SRAM, the area of the bi-directional shifter, and the area of all circuits in the LUT fabric.

1DESTINY is an open-source tool to simulate 2D and 3D memory arrays, which utilizes the 2D modeling framework of NVSim [26] for SRAM and non-volatile memories based on emerging technologies.
Fig. 8: (a) Delay, (b) throughput, (c) area, (d) power, (e) throughput per area, and (f) area-delay-power product (ADPP) of AES-128 encryption. We compare the AES accelerators listed in Table II. Filled (hollow) bars do not (do) include the latency and energy overhead of data transfers to/from a memory unit for the ASIC-based accelerators (designs A and B).

Fig. 9: Comparison between designs A-E for 7 different FoM (a) without and (b) with the overhead of data transfers.

B. Quantitative Analysis with Different FoM

Table II summarizes previous work on accelerators for AES, along with our proposed design IMCRYPTO (design E). Designs A and B are ASIC accelerators for AES based on a 45nm CMOS technology node. Designs C, D and E are based on the IMC concept and consist of CEM arrays integrated with customized circuitry that compute at the memory periphery. While designs C and E are based on 40nm and 45nm CMOS technologies respectively, design D is based on a 65nm PCM technology. Designs B and C support other algorithms/functions beside AES encryption. Support for AES decryption is not discussed in [5], [6]. AES decryption is possible with designs A, D and E.

Fig. 8(a-f) and Fig. 9(a-b) present and compare different FoM for designs A-E for AES-128 encryption. While Fig. 8(a-f) presents the raw data for all FoM in separate bargraphs, Fig. 9(a-b) enables us to collectively compare the different designs in terms of all FoM evaluated. Two sets of data are considered: The filled bars in Fig. 8(a-f) (lines in Fig. 9(a)) do not include the latency and energy of data transfers to/from a memory unit for the two ASIC-based accelerators (designs A and B). Note that designs C—E are based on the IMC paradigm, which allow computation to be performed inside (or near) the memory without the need for data transfers. The second set of data (hollow bars in Fig. 8(a-f)/ lines in Fig. 9(b)) adds 744.49 µs latency (30.39 µJ energy) from memory transfers to the latency (energy) of designs A and B. This additional latency/energy is spent on transferring 1MB of plaintext from a 1MB SRAM to each ASIC accelerator, and writing back the resulting 1MB of ciphertext to the same 1MB SRAM. DESTINY [25] is used to estimate the latency and energy overheads of such data transfers.

We first consider delay and throughput when data transfers are not accounted for in designs A and B (i.e., the ASIC-based accelerators). In this scenario, design B has the shortest delay to encrypt 1MB of data and the highest throughput among all designs evaluated (Fig. 8(a) and Fig. 8(b)). IMCRYPTO has the highest throughput per area (Fig. 8(e)) among all designs.
when compared to design B, design E is 1.2 \times ASIC-type accelerators in our analysis, design E has the shortest delay and throughput. Importantly, fast and wide AES-128 encryption in IMCRYPTO (design E) comes with an improvement of 19.7 \times compared to design A-D, design E enables performance of IMC for security applications is worthwhile to pursue in further investigations.

### C. FoM Projection based on Emerging Technologies

Per Fig. [10] the CEM and the LUT fabric modules of IMCRYPTO are responsible for >99.9\% of the total area of our proposed AES encryption/decryption IMC accelerator. As discussed in Sec. [III] these two blocks of IMCRYPTO are built with CMOS 6T-SRAM and 9T-RA/CAM cells. Here, we investigate whether the use of memories based on emerging technologies such as PCM, RRAM, STT-MRAM, and FeFETs to replace CMOS-based memories could result in additional benefits to our proposed IMCRYPTO approach.

Per [8], [9], [30], emerging technologies could potentially improve the density of memories by factors of 1 \times – 25 \times. Furthermore, the read power of memories based on most emerging technologies may be comparable to that of CMOS SRAM, with static power savings due to non-volatility [9]. Despite these advantages, one of the downsides of emerging technologies is their long read and write latencies (up to 10 \times of CMOS SRAM), which may increase clock cycle time and reduce the frequency of operation of IMC accelerators such as IMCRYPTO. To analyze the impact of emerging technologies on the ADPP of IMCRYPTO, we study the trade-off between longer cycle times (thus lower operating frequencies) and density improvements enabled by emerging technologies with respect to CMOS SRAM (Fig. [11]).

In our projection, we consider the read power of emerging technologies to be on par with that of CMOS SRAM [9]. The green curves in the plot of Fig. [11] represent different memory densities (1 \times – 25 \times compared to SRAM). Given cycle times that are up to 10 \times longer than those of conventional SRAM (x-axis), improvements in ADPP with respect to CMOS-based IMCRYPTO (y-axis) are captured. From the data points depicted in Fig. [11] (from [8], [30]), we project that current state of the art memory technologies improve the ADPP of IMCRYPTO by 1.7 \times – 5.3 \times. For instance, by implementing a PCM-based IMCRYPTO, we project 2.5 \times benefits in ADPP relative to a CMOS-based IMCRYPTO (design E in Table II). As CMOS-based IMCRYPTO’s ADPP slightly outperforms PCM-based design D by a factor of 1.2 \times, when we compare the two designs implemented with same technology, we expect design E to outperform design D in terms of ADPP by a factor of 3.0 \times.

Despite the projections, we acknowledge that considerable challenges from the system/architecture perspective exist in employing emerging technologies for the design of memories and IMC-based accelerators [31]. For instance, technologies may be at different stages of maturity and not be as easily scalable to smaller nodes as is CMOS technology. Nevertheless, as memory technologies continue to evolve, opportunities emerge with the study of new materials, the possibility of stacked memory, etc. The impact of these advancements on the performance of IMC for security applications is worthwhile to pursue in further investigations.

### VII. Conclusion and Future Work

We propose IMCRYPTO, an IMC fabric for accelerating AES encryption and decryption that enables high throughput while avoiding large area/power overheads. IMCRYPTO
employs a novel approach that combines multiple RAM and RA/CAM arrays with customized encoders in the design of a modular LUT fabric. With the proposed LUT fabric, the (Inv)SubBytes and (Inv)MixColumns steps from AES encryption (decryption) can be combined and performed in one shot, in a highly parallel fashion. Furthermore, integration with RISC-V allows IMCRYPTO to have the flexibility of doing different modes of operation and opens the door for the implementation of other ciphers that require operations that are similar those of AES. Our future work will evaluate the execution of different AES modes and key lengths with IMCRYPTO, and will extend our customized instructions to enable the execution of other symmetric-key ciphers. Finally, to reduce the number of customized RISC-V instructions used in IMCRYPTO, we will exploit different levels of control granularity by replacing some of subset of customized instructions with “macro instructions” and provide special hardware control units for these macro instructions.

ACKNOWLEDGMENTS

This work was supported in part by ASCENT, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] J. Daemen and V. Rijmen, “AES proposal: Rijndael,” 1999.
[2] K. Stevens and O. A. Mohamed, “Single-chip FPGA implementation of a pipelined, memory-based AES Rijndael encryption design,” in CCECE, 2005, pp. 1296–1299.
[3] R. Chaves, G. Kuzmanov, S. Vassiliadis, and L. Sousa, “Reconfigurable memory based AES co-processor,” in IIPPS, 2006.
[4] R. Ueno, S. Morioka, N. Miura, K. Matsuda, M. Nagata, S. Bhasin, Y. Mathieu, T. Graba, J. Danger, and N. Homma, “High Throughput/Get AES Hardware Architectures Based on Datapath Compression,” IEEE Trans Comput., vol. 69, pp. 534–548, 2020.
[5] G. Sayilar and D. Choudhury, “Cryptoraptor: A Reconfigurable Cryptographic Cortex-M0 Processor With In-Memory and Near-Memory Computing for IoT Security,” IEEE J. Solid-State Circuits, vol. 53, pp. 995–1005, 2018.
[6] M. Xie, S. Li, A. O. Glova, J. Hu, and Y. Xie, “Securing emerging nonvolatile main memory with fast and energy-efficient aes in-memory implementation,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, pp. 2443–2455, 2018.
[7] R. A. Sasonko, I. M. N. Kumara, A. Wicaksana, F. Rousseau, and O. Muller, “Hardware context switch-based cryptographic accelerator for handling multiple streams,” ACM Trans. Reconfigurable Technol. Syst., vol. 14, Aug. 2021. [Online]. Available: https://doi.org/10.1145/3460941
[8] B. Ege, E. B. Kavun, and T. Yalçın, “Memory encryption for smart cards,” in International Conference on Smart Card Research and Advanced Applications. Springer, 2011, pp. 190–216.
[9] W. Shi, H. Lee, M. Ghosh, C. Lu, and A. Boldyreva, “High efficiency counter mode security architecture via prediction and precomputation,” in 32nd International Symposium on Computer Architecture (ISCA’05), 2005, pp. 14–24.
[10] S. Aga, S. Jeloka, A. Subramaniyam, S. Narayanasamy, B. Blauuw, and R. Das, “Compute caches,” in IEEE International Symposium on High Performance Computer Architecture (HPCA), Feb 2017, pp. 481–492.
[11] E. Tromer, D. A. Osvik, and A. Shamir, “Efficient Cache Attacks on AES, and Countermeasures,” J. Cryptol., vol. 23, p. 37–71, Jan. 2010.
[12] D. Blazhevski, A. Bozhinovski, B. Stoichevska, and V. Pachovski, “Modes of operation of the aes algorithm,” 2013.
[13] A. Almuhannadi and I. Al-Hejri, “A comparative analysis of aes common modes of operation,” in 2017 IEEE 30th Canadian conference on electrical and computer engineering (CCECE). IEEE, 2017, pp. 1–4.
[14] Hua Li, “A new CAM based S/S/sup /+-/box look-up table in AES,” in ISCAS, 2005, pp. 4634–4636 Vol. 5.
[15] S. Aga, S. Jeloka, A. Subramaniyam, S. Narayanasamy, B. Blauuw, and R. Das, “Compute caches,” in IEEE International Symposium on High Performance Computer Architecture (HPCA), Feb 2017, pp. 481–492.
[16] E. Tromer, D. A. Osvik, and A. Shamir, “Efficient Cache Attacks on AES, and Countermeasures,” J. Cryptol., vol. 23, p. 37–71, Jan. 2010.
[17] D. Blazhevski, A. Bozhinovski, B. Stoichevska, and V. Pachovski, “Modes of operation of the aes algorithm,” 2013.
[18] A. Almuhannadi and I. Al-Hejri, “A comparative analysis of aes common modes of operation,” in 2017 IEEE 30th Canadian conference on electrical and computer engineering (CCECE). IEEE, 2017, pp. 1–4.
[19] J. Wolkerstorfer, E. Oswald, and M. Lamberton, “Anasic implementation of the aes sboxes,” in Topics in Cryptology — CT-RSA 2002, B. Preneel, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 67–78.
[20] D. Reis, M. Niemier, and X. S. Hu, “A Computing-in-Memory Engine for Searching on Homomorphically Encrypted Data,” IEEE Journal on Exploratory Solid-State Computational Devices and Circuits, pp. 1–1, 2019.
[21] X. Si, W. Khwa, J. Chen, J. Li, X. Sun, R. Liu, S. Yu, H. Yamauchi, Q. Li, and M. Chang, “A Dual-Split 6T SRAM-Based Computing-in-Memory Unit-Macro With Fully Parallel Product-Sum Operation for Binarized DNN Edge Processors,” IEEE TCAD, vol. 66, pp. 4172–4185, 2019.
[22] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, “DESTINY: A Tool for Modeling Emerging 3D NVM and eDRAM Caches,” in IEEE/ACM Design, Automation & Test in Europe Conference & Exhibition (DATE), San Jose, CA, USA, 2015, pp. 1543–1546.
[23] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, “NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 31, pp. 994–1007, July 2012.
[24] Y. Cao, Predictive technology model for robust nanoelectronic design. Springer Science & Business Media, 2011.
[25] J. Knudsen, “Nangate 45nm open cell library,” CDNLive, EMEA, 2008.
[26] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, “OpenRAM: An open-source memory compiler,” in ICCAD. IEEE, 2016, pp. 1–6.
[27] D. Reis, K. Ni, W. Chakraborty, X. Yin, M. Trentzsch, S. D. Dinkel, T. Mielde, J. Mullers, S. Beyer, S. Datta, M. T. Niemier, and X. S. Hu, “Design and analysis of an ultra-dense, low-leakage, and fast fefet-based random access memory array,” IEEE Journal on Exploratory Solid-State Computational Devices and Circuits, vol. 5, pp. 103–112, 2019.
[28] O. Mutlu, “Memory scaling: A systems architecture perspective,” in 2013 5th IEEE International Memory Workshop, 2013, pp. 21–25.