Recryptor: A Reconfigurable Cryptographic Cortex-M0 Processor With In-Memory and Near-Memory Computing for IoT Security

Yiqun Zhang, *Student Member, IEEE*, Li Xu, *Student Member, IEEE*, Qing Dong, *Student Member, IEEE*, Jingcheng Wang, *Student Member, IEEE*, David Blaauw, *Fellow, IEEE*, and Dennis Sylvester, *Fellow, IEEE*

***Abstract*— Providing security for the Internet of Things (IoT) is increasingly important, but supporting many different cryptographic algorithms and standards within the physical constraints of IoT devices is highly challenging. Software implementations are inefficient due to the high bitwidth cryptographic operations; domain-specific accelerators are often inflexible; and reconfigurable crypto processors generally have large area and power overhead. This paper proposes Recryptor, a reconfigurable cryptographic processor that augments the existing memory of a commercial general-purpose processor with compute capabilities. It supports in-memory bitline computing using a 10-transistor bitcell to support different bitwise operations up to 512-bits wide. Custom-designed shifter, rotator, and S-box modules sit near the memory, providing high-throughput near-memory computing capabilities. We demonstrate Recryptor's programmability by implementing the cryptographic primitives of various public/secret key cryptographies and hash functions. Recryptor runs at 28.8 MHz in 0.7 V, achieving 6.8× average speedup and 12.8× average energy improvements over the state-of-the-art software- and hardware-accelerated implementations with only 0.128 mm^2 area overhead in 40-nm CMOS.**

***Index Terms*— Advanced Encryption Standard (AES), cryptographic processor, differential power analysis (DPA), elliptic curve cryptography (ECC), in-memory computing, Internet of Things (IoT), reconfigurable hardware, Secure Hash Algorithm 3 (SHA-3), security.**

I. Introduction

SECURITY is of utmost concern for Internet of Things (IoT) applications due to the potential pervasiveness of IoT devices. In order to ensure secure communications, we need good authentication for trustworthiness, data encryption for confidentiality and integrity, and fault tolerance for resilient operation under attack. Cryptographic algorithms provide the insurance for functional security, and they are mostly divided into three types: symmetric, asymmetric, and hash functions. Typically, asymmetric (public key) cryptography (PKC) is used for key exchange, and the well-known Rivest-Shamir-Adleman (RSA) cryptosystem was proposed by Rivest *et al.* [1] in 1977. Elliptic curve cryptography (ECC) [2], [3] was discovered in 1985 and is a popular option for PKC; for example, the STSAFE-A100 chip uses ECC for authentication [4]. For the desired security level, ECC needs significantly smaller keys than RSA, which results in smaller energy and memory requirements. An ECC can be applied either to binary field or prime field, and the comparison of these two fields can be found in [32] regarding the energy, runtime, and toplevel protocols. Symmetric (secret key) cryptography is used for efficiently encrypting data; for example, the Advanced Encryption Standard (AES) is a commonly used symmetric block cipher, approved by National Institute of Standards and Technology (NIST) [5]. Hash functions irreversibly "encrypt" information and provide a digital fingerprint, which are used for message integrity. The Keccak hash function won the Secure Hash Algorithm 3 (SHA-3) competition hosted by NIST in 2012 [6]. In summary, the underlying mathematics varies greatly across cryptosystems, and security standards evolve over time. A full system design with secure communications would require different security algorithms of various types [7].

IoT platforms have limited computational resources for energy/area reasons. Cryptographic functions typically require high bitwidth calculations (64–512 bits), but embedded processors' datapaths tend to be 32-bit wide. Executing these crypto algorithms in software on microcontrollers is simple but energy inefficient and slow. The first option to address this is to optimize software for cryptographic calculations on microcontrollers. For example, Aranha *et al*. [9] and de Clercq *et al*. [10] propose efficient assembly implementations by manipulating the data flow to maximize the register use and achieve around two to three orders of magnitude of speedup. The second option is to use application-specific integrated circuits (ASICs), or accelerators, to run a specific algorithm. ASICs [11]–[13] achieve high performance with low-energy consumption, but they are only useful for a single purpose, so multiple ASICs are required to cover a range of applications. Even for the same function, there are many different designs optimized for various performance and power requirements [14], [15]. Finally, the third option is to build cryptographic coprocessors for supporting different algorithms, which can give higher throughput and

Manuscript received August 7, 2017; revised October 22, 2017; accepted November 10, 2017. Date of publication February 5, 2018; date of current version March 23, 2018. This paper was approved by Guest Editor Makoto Ikeda. *(Corresponding author: Yiqun Zhang.)*

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: zhyiqun@umich.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/JSSC.2017.2776302

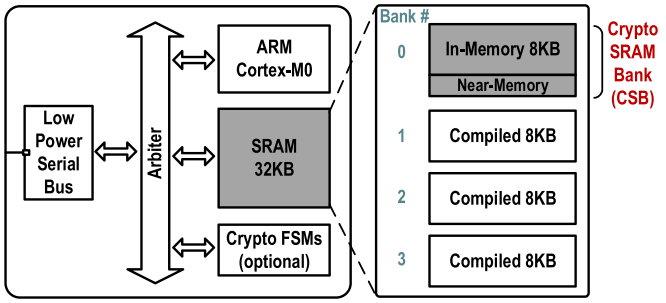


Fig. 1. Proposed Recryptor architecture.

maintain flexibility. The key idea in coprocessor designs is that they try to share as much logic as possible among the supporting algorithms to save area; nonetheless, they still tend to have high area and power overhead since they implement an entire separate processor with fetch, decode, register file, and local memory [16]–[18]. Also, existing coprocessor designs only cover a limited range of cryptographic algorithm types; for example, Hutter *et al.* [17] supports only symmetric and asymmetric crypto algorithms, while Sayilar and Chiou [18] supports only symmetric and hash functions.

The importance of security and the limitations in existing solutions motivate a new architecture in favor of the requirements for IoT devices. In this paper, we propose a reconfigurable cryptographic processor, called Recryptor [19], which exploits in-memory and near-memory computing to achieve energy efficiency, performance, and programmability for IoT security applications. We measure Recryptor's speedup and energy gains on core functions for symmetric and asymmetric cryptography as well as hash functions. Compared with a Cortex-M0 baseline, we achieve energy gains of 9.1× for AES, >6.7× for ECC finite field multiplication and reduction (FFMR), and 4.9× for SHA-3 Keccak function (Keccak-f), with energy gains of >4.1× across crypto algorithms relative to the literature. To the best of our knowledge, this is the first work that can accelerate public/secret key cryptography and hash functions at the same time.

The remainder of this paper is organized as follows. Section II gives the overview of the proposed Recryptor design. Sections III and IV provide the detailed explanation of in-memory and near-memory computing. Section V describes the programmability of Recryptor and our optimized algorithm implementations on Recryptor. Section VI talks about the testchips, measured results, and comparisons. Section VII gives some discussion about further optimizations on Recryptor. The conclusion is drawn in Section VIII.

II. Overview of Recryptor

Recryptor contains a standard ARM Cortex-M0 microcontroller with 32-kB memory, a low-power serial bus to access off-chip data, and an arbiter as its internal bus, as shown in Fig. 1. The optional finite state machines (FSMs) for further acceleration will be discussed in Section V-D. The 32-kB memory is composed of four 8-kB banks. Three are implemented using a standard memory compiler, while the fourth is a custom-designed crypto-SRAM bank (CSB).

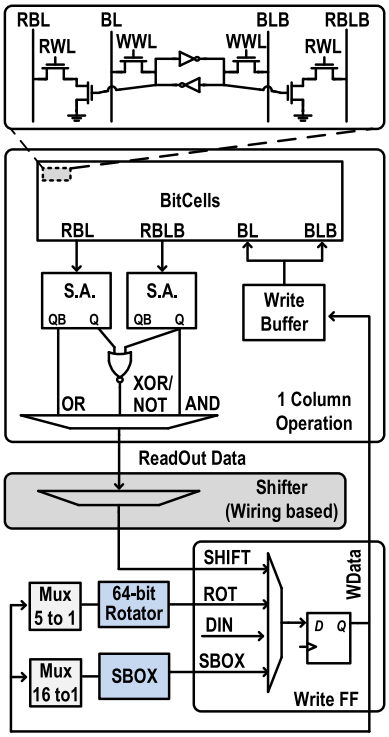


Fig. 2. Proposed CSB.

The CSB can operate as a normal memory with 32-bit read and write, but it also supports large bitwidth in-memory and near-memory computing. As shown in Fig. 2, the CSB uses a 10-transistor (10T) bitcell, which has dual read ports in order to enable different bitwise operations. In each cycle, two words are accessed simultaneously in the bank and perform a bitwise logic operation which is read out with standard sense amplifiers. The memory sub-banks are set to different widths to support different lengths of vector computation. After the readout, sense amplifiers are three near-memory logic functions: a shifter, an arbitrary 64-bit rotator, and an S-box. These three options, together with data-input from the arbiter interfacing with the processor, provide the write back data to memory.

Equipped with in-memory and near-memory computing capabilities, users can directly program the Cortex-M0 to use the CSB through a memory-mapped decoder to accelerate different crypto algorithms. We optimize and demonstrate AES, FFMR with four different word lengths, and the Keccak hash function in this paper.

III. Recryptor's In-Memory Computing

A. 10T Bitcell and Accelerated Bitwise Operations

The 10T bitcell is used in the CSB (Fig. 2). For normal read operation, only read bitline (RBL) is precharged to high. RBL will be discharged if the stored data is 1 and remain high if the stored data is 0. The data is read out by a skewed inverter-based sense amplifier. For in-memory computing, either or both RBL and RBL bar (RBLB) will be precharged, depending on the required bitwise operations. Fig. 3 shows how to get the bitwise operations on two words. First, A OR B is achieved by precharging RBL to high and

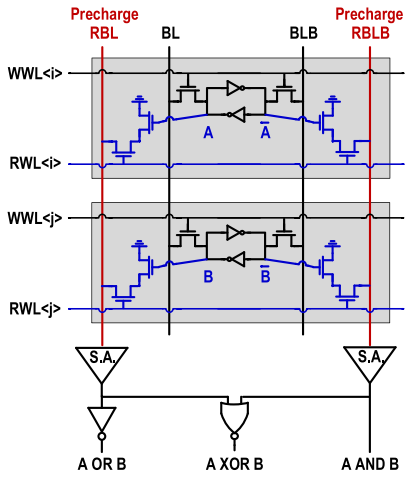


Fig. 3. 10T bitcell and supported bitwise operations on two words.

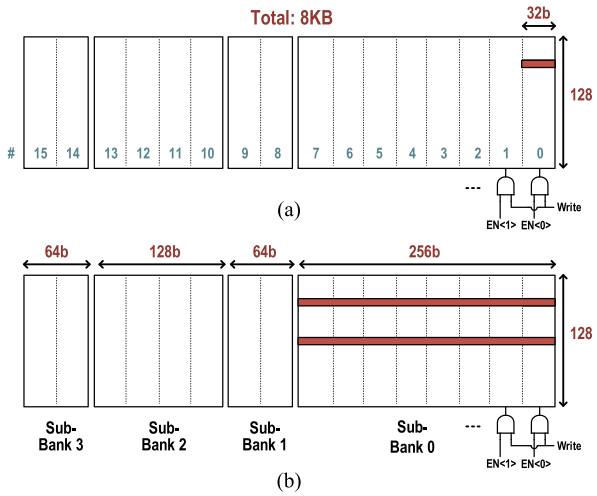


Fig. 4. Sub-bank configuration and implementation. (a) Normal 32-bit memory access by enabling 1 slice. (b) In-memory computing by enabling sub-banks.

then enabling two words' read-wordline. If at least one of the stored data A and B is 1, RBL will be discharged, which is the function of A NOR B. Similarly, A AND B works the same way, but operates on RBLB. Finally, A XOR B is achieved by precharging both RBL and RBLB, and adding an NOR gate between them.

The supported bitwise operations of CSB are OR/AND/XOR on two words and COPY/NOT on one word. All of these operations are performed in one memory cycle. Like an 8T bitcell, we achieve robust operation at low voltage using decoupled read. Since we need both RBL and RBLB for the three logic operations, we have two read ports leading to 10 transistors in the bit cell. However, smaller SRAM cells could be used for smaller area, however, the conventional 6T bitcell [20] has degraded read noise margins and worse performance at low voltages, and the 4 + 2T bitcell [21] requires additional voltage supplies.

B. Configuration of Bank Division

The sub-bank is configured and implemented as shown in Fig. 4. The 8-kB CSB is comprised of 16 slices, each with 128 32-bit words. During a normal 32-bit memory access, just one slice is activated to save energy. During in-memory computing, by enabling different sub-banks or a combination of them, we can support up to 512-bit single-cycle computations. The size and placement of these sub-banks were optimized at design time to support a wide range of security primitives with efficient signal gating, which will be discussed in Section V-A.

TABLE I
SHIFTER SUPPORTED FUNCTIONS

Shifter options	Function	Supported subBank
LS1	Left shift 1 bit	0,1,2,3
LS4	Left shift 4 bits	0,1,2,3
LS64	Left rotate 64 bits	0,1
RS64	Right rotate 64 bit	0,1
ROT1_w64	Right rotate 1 bit within 64 bits	0,1
ROT8_w64	Right rotate 1 bit within 64 bits	2
S.Row	Shift n_{th} bytes in i_{th} sub-subBank to $\{(n+i) \bmod 4\}_{th}$ sub-subBank	2
KG	Shift i_{th} sub-subBank words to $\{(i+2) \bmod 4\}_{th}$ sub-subBank	2

Algorithm 1 López-Dahb Multiplication and Reduction in F_2^m

Input: $x = (x_{m-1} \ldots x_0)$, $y = (y_{m-1} \ldots y_0)$, $= (r_{m-1} \ldots r_0)$
Output: $c = (c_{m-1} \ldots c_0) = xy \bmod r$
[Note: $x/y/r/c$ is at 1 physical line in CSB]
1: Compute $T(u) \leftarrow uy \bmod r$ for all polynomials u of degree lower than ω
2: Compute $T'(u) \leftarrow ur$ for all polynomials u of degree lower than ω
3: $c \leftarrow 0$
4: **for** $\leftarrow \lceil m/\omega - 1 \rceil$ down to 0 **do**
5: $u(x \gg j \cdot \omega)$ & 0xF
6: $c \leftarrow \text{SHIFT}(c \oplus T(u), \text{LS4})$ *
7: $u' = (c \gg m)$ & $0 \times$ F
8: $c \leftarrow c \oplus T'(u')$ ***[Note: reduction step, c stays to be m bits]***
9: **end for**
10: **return** c

*Syntax SHIFT(x, y): apply y shift to x vector using Shifter

IV. RECRYPTOR'S NEAR-MEMORY COMPUTING

A. Shifter

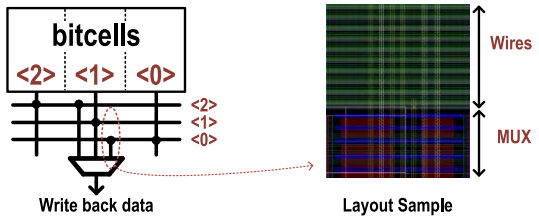
The shifter is a compact, wiring-based custom design, which is pitch matched with the SRAM bitcell. There are different multiplexer (MUX) options depending on the supported functionality. Table I shows the functions implemented under each sub-bank, which are used in Algorithms 1–3 in Section V. In the 3-to-1 MUX example in Fig. 5(a), vertical wires contain nearby bits' data and an MUX is used to select one of them as the write back data. Fig. 5(b) demonstrates a small sample of the physical implementation to show that it is a wiring intensive design. The top is wired with four metal layers for horizontal routing and 1.5 minimum spacing. The routing pattern is consistent by shifting 1 via for nearby bits so the design and layout complexity is low.

Algorithm 2 AES Encryption Function

Input: plaintext p, key k, where P/K is 128bits and at 1 physical line in Bank0, $k = [k[0]; k[1]; k[2]; k[3]]$
Output: ciphertext C

1: $C \leftarrow P$
2: **for** $i \leftarrow 0$ to $nr-1$ **do**
3: AddRoundKey: $C = C \oplus K$
4: ShiftRow: $D = \text{SHIFT}(C, \text{SRow})^*$
5: SubByte: $D[j] = \text{SBOX}(D[j]), \forall j \in [0, 15]$
6: **if** $(i \neq nr-1)$ **do**
7: MixColumn: $E = \text{SHIFT}(D, \text{ROT8})$; $F = \text{SHIFT}(E, \text{ROT8})$
8: $G = \text{SHIFT}(F, \text{ROT8})$; $H = D \oplus E$
9: $I[j] = (H[j][7])\ ?\ 0x1B : 0x0$
10: $C = H \oplus I \oplus E \oplus F \oplus G$
11: KeyGen: $k[4] = \text{SHIFT}(k[3], \text{KG})$
12: $k[4][j] = \text{SBOX}(k[4][j]), \forall J \in [0, 3]$
13: $k[4] = k[4] \oplus Rcon[i]$; $k[0] = k[0] \oplus k4$
14: $k[j+1] = k[j+1] \oplus \text{SHIFT}(k[j], \text{KG}), \forall j \in [0, 2]$
15: **end if**
16: **end for**
17: **return** C

∗Syntax SHIFT(x, y): apply y shift to x vector using shifter



(a) (b)

Fig. 5. (a) Shifter 3-to-1 MUX example. (b) Shifter physical implementation sample.

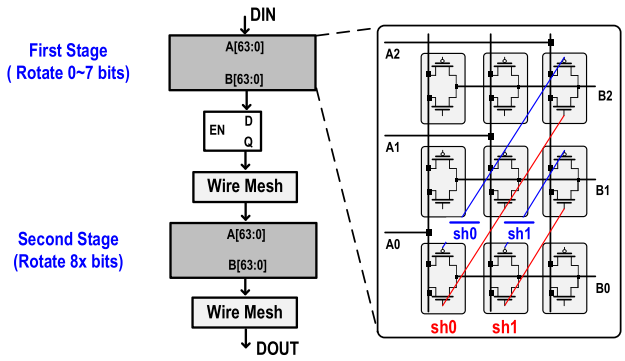


Fig. 6. Arbitrary 64-bit rotator.

In the Recryptor testchip, the memory read and shifter operations are implemented as a single-cycle operation, before writing back to the write flip-flop as shown in Fig. 2.

B. Rotator

The rotator is a custom two-stage design for arbitrary 64-bit rotation. As shown in Fig. 6, the first stage rotates 0–7 bits and the second stage rotates in multiples of 8 bits. This architecture is similar as [22], which requires much less area than the long wires needed of directly rotating from 0 to 63 bits. We use the same physical implementation, similar as a barrel shifter [23], for both stages in our design. However, transmission gates are used for the MUXes to achieve low energy and stable operation at low voltages. By using wire meshes, the same custom compact layout can be used for the first and second stage, which reduces the area of long wires for shifting multiples of 8 bits and design time.

Algorithm 3 KECCAK-f Function

Input: KECCAK[b](S), where $S' = S[0:4, y]$ is at 1 physical line, $\forall y \in [0, 4]$
Output: S

1: **for** $i \leftarrow 0$ to $nr-1$ **do**
2: θ step: $C = S'[0] \oplus S'[1] \oplus S'[2] \oplus S'[3] \oplus S'[4]$
3: $D = \text{SHIFT}(C, \text{LS64}) \oplus \text{SHIFT}(\text{SHIFT}(C, \text{RS64}), \text{ROT1})$
4: $S'[y] = S'[y] \oplus, \forall y \in [0, 4]$
5: ρ step: read $S'[y]$ in 1 cycle, then $S[x, y] = \text{ROT}(S[x, y], r[x, y])$
6: π' step: $S'[y] =$ **do** $\text{SHIFT}(S'[y], \text{LS64})$ **for** y iterations
7: ***[Note: π' step result is the transpose of π step in odd iterations]***
8: X step: $E[y] = \text{SHIFT}(S'[y], \text{LS64})$
9: $S'[y] = S'[y] \oplus (\text{NOT}\, E[y])$ AND $\text{SHIFT}(E[y], \text{LS64})$
10: ι step: $S[0, 0] = S[0, 0] \oplus RC[i]$
11: **end for**
12: **return** S

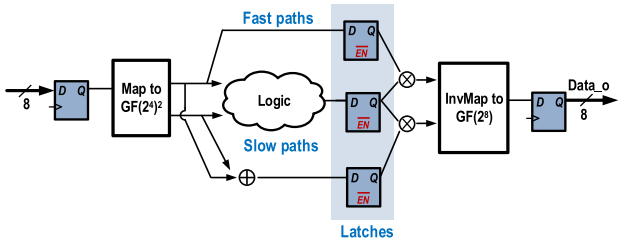


Fig. 7. Proposed S-box implementation.

C. S-Box

The S-box is a commonly used component in block ciphers for byte substitution. It is a nonlinear function that performs a multiplicative inversion on Galois field (GF) (2^8), followed by an affine transformation. This algorithm is more efficient with a transformation into the composite field GF $(2^4)^2$ [14]. Our previous design [11] proposed a two-stage implementation by adding flip-flops to reduce glitch power due to the presence of fast and slow paths. In this design as shown in Fig. 7, we further replace the middle flip-flops with latches due to a longer clock-cycle, which enables a 1 cycle latency of this block as well as saving a bit of area.

TABLE II
ESTIMATED # OPERATIONS ACCORDING TO ALGORITHM 1

Step #	Mem read w/ XOR + Mem Write	Mem read w/ Shift + Mem Write	Mem read w/ Copy + Mem Write	Mem read w/ Shift
1	$2^{\omega}-2$	$\omega-1$	2	-
2*	$2^{\omega}-2$	$\omega-1$	2	-
3	-	-	1	-
5	-	-	-	$\lceil m/\omega-1\rceil$
6	$\lceil m/\omega-1\rceil$	$\lceil m/\omega-1\rceil$	-	-
7*	-	-	-	$\lceil m/\omega-1\rceil$
8*	$\lceil m/\omega-1\rceil$	-	-	-

V. PROGRAMMABILITY AND OPTIMIZED ALGORITHM IMPLEMENTATIONS

Users can write software to configure Recryptor to accelerate different cryptographic algorithms. We demonstrate one algorithm for each category to show Recryptor's flexibility and performance. First, we will analyze FFMR, the basic operations for ECC. Then, we will analyze AES for secret key cryptography and Keccak-f for hash functions.

A. Finite Field Multiplication and Reduction

Field multiplication computes $x \cdot y = z$, where x and y are the binary polynomials of degree at most $(m-1)$, and z is the degree at most $(2m-2)$. In order to return the multiplication results to degree of at most $(m-1)$, field reduction computes $z \bmod r = c$, where r is a polynomial of degree m.

The López-Dahab (LD) field multiplication algorithm [24] uses a windowing method with a precomputed table. With a window width of ω bits, the input x is shifted right to scan its last ω bits at a time. Then each ω bits are used as an index for the precompute table lookup. However, the number in the finite field needs m bits, and the inputs/output/intermediate values are stored in the memory. Due to register spilling on the M0, using this algorithm tends to create a large memory accesses. Aranha *et al*. [9] and de Clercq *et al*. [10] optimize the overflow to solve this spilling problem by maximizing the register reuse.

We propose a new optimization to combine the (LD) field multiplication and reduction algorithm with the goal of reducing the number of operations on Recryptor, as shown in Algorithm 1. The problem of register spilling does not need to be considered in this proposed method since all the calculations are computed in/or near memory with m bits of parallelism.[1] Table II lists the estimated number of operations needed for each step in Algorithm 1, with respect to the curve degree m and window width ω. The first step computes 2^{ω} of m bits numbers; $0 \cdot y$ and $1 \cdot y$ need memory reads with copy and data write back; and all others need the XOR operation between two words. For $2^t \cdot y$ $(t = 0 \ldots \omega-1)$, additional SHIFT operations before XOR are needed to remove the overflowing bits. Step 5 would apply SHIFT to the input x and use the first ω bits as the index to the precompute table in step 1.

[1]The Recryptor testchip in this paper supports up to 512 bits of parallelism, based upon our implementation.

TABLE III
ENABLED SUB-BANK OF DIFFERENT WORD LENGTH FOR FFM

Word length \ Sub Bank	3 (64bits)	2 (128bits)	1 (64bits)	0 (256bits)
163 bits		x	x	
233 bits				x
283 bits			x	x
409 bits		x	x	x

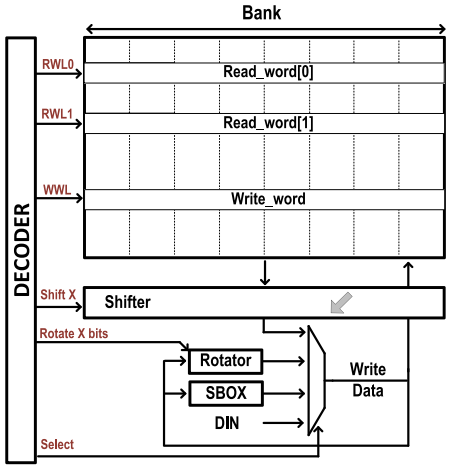


Fig. 8. Memory-mapped decoder.

Then, in step 6, we do an XOR and a 4-bit SHIFT. Reduction is considered within our algorithm with steps 2, 7, and 8, in order to reduce the size of intermediate values.

Table III shows where to perform calculations of different word lengths by activating different sub-banks. The sub-bank is configured to select nearby banks for the options listed, with efficient signal routing for shift operations. With the estimated number of operations in the columns 2, 3, and 4 to be 2, and the last column to be 1, the total number of estimated operations for multiplication in F_2^{233} is 330, compared to 4980 on a baseline Cortex-M0 software implementation and 2968 with register optimization [10].

B. Advanced Encryption Standard

Algorithm 2 shows in detail the implementation of AES on Recryptor. Steps 1, 3, and 4 use 128-bit operations, and step 5 uses the S-box block on 1 byte for 16 cycles in total. Steps 7–10 implement the MixColumn operation by left rotating by 1, 2, and 3 bytes, adding intermediate values D and E, and then multiplying based upon the MSB for each byte in $H[j]$. The KeyGen operation is achieved using 32-bit operations in steps 11, 13, and 14 and 8-bit operations in step 12.

The input/output/intermediate 128-bit values are all stored in the CSB's sub-bank 2, where more shifting options used by AES are supported (see Table I). Compared with an AES ASIC design, our solution only needs an additional S-box block, which occupies 30% of the area of the current smallest AES ASIC design [11].

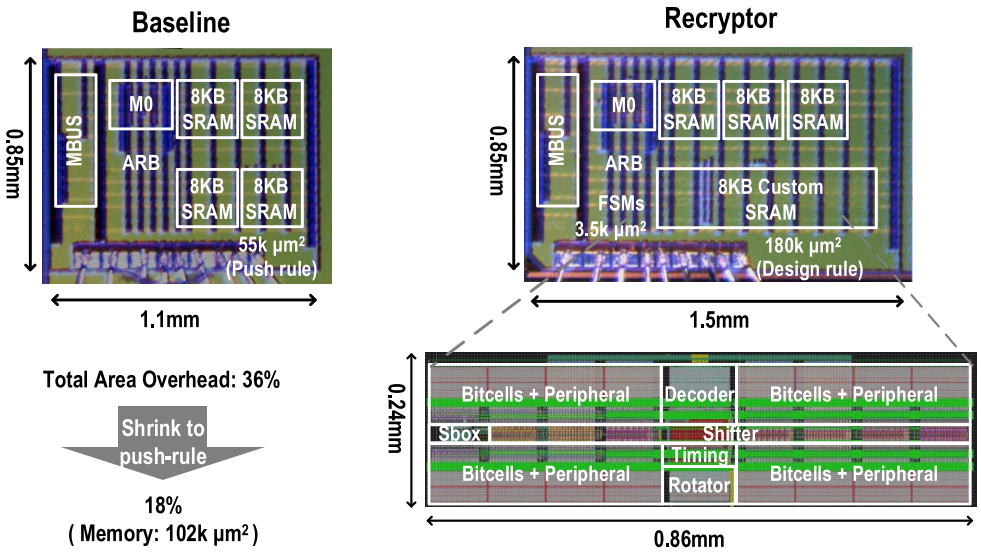


Fig. 9. Die photograph of baseline and Recryptor in 40-nm CMOS.

C. Keccak-f

The Keccak-f permutation function [6], with $b = 1600$, goes through 24 iterations of five steps on 1600 bits of data, which are treated as a block of 5×5 64-bit words. Algorithm 3 shows the detailed implementation of the Keccak-f on Recryptor. The 320-bit-wide operations include the in-memory bitwise operations of AND/NOT/XOR/COPY and the near-memory operations of left/right 64-bit shifts.

Keccak is slow to run on a 32-bit processor since it operates on 64-bit words. More specifically, there are two main issues with implementing this function on Recryptor. First, the arbitrary rotation of a 64-bit number in the ρ step would be expensive on 32-bit MCUs like the Cortex-M0. Schwabe *et al.* [25] propose the bit-interleaving technique in software, by collecting even and odd positions of a 64-bit data into two 32-bit data. In contrast, Recryptor applies a custom two-stage 64-bit rotator placed in the near-memory computation block to save energy and time on this step. The second problem is that the array transpose in the π step [26] would be very hard to do in-memory since in-memory operations require aligned data. To address this problem, the proposed π' step combines the even/odd iterations and modifies the intermediate results of each iteration, which helps avoid the large memory accesses needed for matrix transpose.

D. Cryptographic Finite State Machines

Programming Recryptor in software requires only writing a command to a memory-mapped decoder. The command indicates the opcode and which word lines to compute on and write back to. As shown in Fig. 8, the CSB configures itself based upon the detailed information sent by the decoder, i.e., addresses for in-memory computing and the number of shift/rotate bits for near-memory computing. Therefore, each bitwise operation can be written in detail inside the C doe.

For in-memory operations, input data must be aligned inside memory. Currently, this data alignment is done manually, but this could be optimized by operand locality and is addressed in [27].

Since loads and stores on the M0 are somewhat expensive, to further improve performance, we implemented optional FSMs to automatically issue the commands at a small area overhead.

VI. Recryptor Testchip Design and Experimental Results

A. Testchips and Measurements

Both the baseline and the Recryptor designs, based upon the ARM Cortex-M0 processor, were implemented on a testchip in 40-nm CMOS. Chip micrographs are shown in Fig. 9. The switch from an 8-kB compiled SRAM to a custom compute memory increases its area from 55k μm^2 to 180k μm^2. The layout of CSB is compact, with bitcell banks around the side, memory decoders in between banks and the shifter, timing generation, and rotator in the center. The area overhead of crypto-FSMs for AES/ECC/Keccak is 0.29k/2.67k/0.62k μm^2, respectively. The total area overhead of Recryptor over the baseline is 36%, including the interface bus.

However, we used design-rule bitcells for the CSB, and if we shrunk them to push-rule bitcells (with bitcell area shrunk by 50% and row peripherals area shrunk by 30%), the memory could be expected to drop to 102k μm^2. Therefore, area overhead over the existing SRAM is only 47k μm^2. The total area overhead would drop to 18%.

Fig. 10 shows the measured maximum frequency ($F_{\max}$) of the custom 10T SRAM. The blue line shows the $F_{\max}$ of normal 32-bit reads, while the red line shows the operation sequence of wide in-memory computing reads, near-memory shifts, and then data write back. Since the same sense amplifier is used for the normal and in-memory computing reads, the worst case $F_{\max}$ for them should be the same. At low voltages, the $F_{\max}$ of in-memory computing is limited by the shifter. Fig. 11 shows the minimum functional

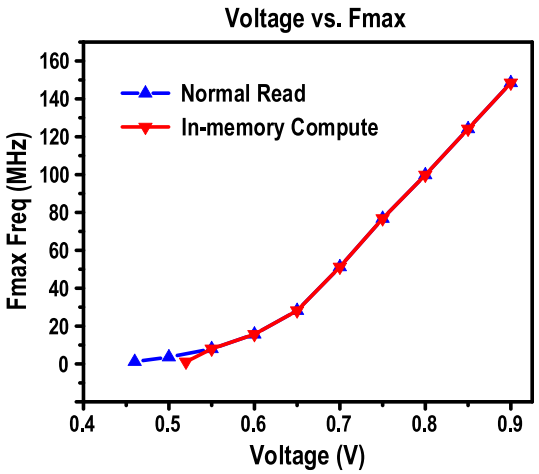


Fig. 10. Measured F_{max} of the custom 10T SRAM.

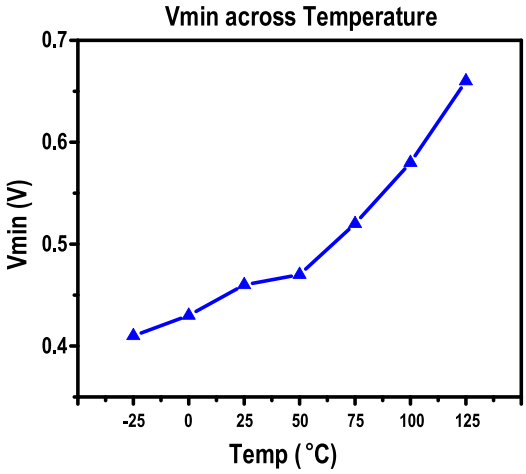


Fig. 11. Measured V_{min} across temperature of the custom 10T SRAM.

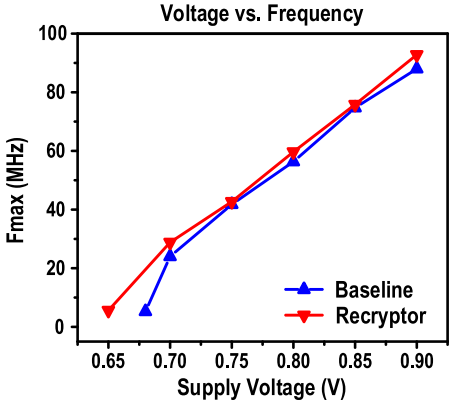


Fig. 12. Measured frequency of baseline and Recryptor across voltages.

voltage (V_{min}) across temperatures for normal CSB reads. The V_{min} of 25 °C/125 °C is 0.46/0.66 V, respectively.

Fig. 12 shows the measured performance of the baseline and Recryptor chips. They both achieve similar frequencies, with a slight difference attributed to die-to-die variation. Recryptor's power is at most 30% larger than the baseline across voltages (Fig. 13), but this overhead is outweighed by the

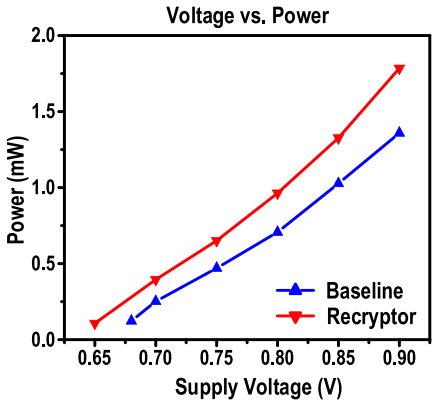


Fig. 13. Measured power of baseline and Recryptor across voltages.

application speedup. The overall performance improvement and energy reduction of each algorithm will be discussed in the following section.

B. Overall Comparison Among Algorithms

Different in-memory operations and near-memory logic are activated when running different crypto functions. Fig. 14 shows the simulated CSB power breakdown. Table IV provides the overall comparison results of three implemented algorithms running on the baseline Cortex-M0, Recryptor, and other processor-based implementations [10], [17], [18].

For FFMR, we support word lengths of 163–409 bits. Recryptor achieves $>11\times$ speedup and $>6.7\times$ energy savings over the baseline software [28]. The performance improvements and energy reductions increase as the word length increases, showing that Recryptor scales well to large bitwidth operations. An assembly code optimization [10] is included for 233-bit, and this method requires extra effort when the word width changes, which is inefficient regarding the design time.

For Keccak, we use [29] as a baseline implementation. To the best of our knowledge, there has not been any coprocessor implementation supporting Keccak, so an ASIC design [12] is included in Table IV for comparison.

For AES, we use the software implementation as our baseline from [30]. Compared to this, Recryptor achieves around $9\times$ speedup and energy. Fig. 15 shows further comparisons in log scale, including an ASIC [11] and a crypto coprocessor [18] design. Overall, Recryptor serves as an intermediate solution among the compared architectures in terms of area, throughput, energy, and programmability.

VII. Discussion

A. Memory Capacity

Table V shows the minimum required memory capacity for computing and the modules required for each algorithm, with the last row summarizing all implemented algorithms. Only 1.84 kB of memory is required to support the chosen algorithms, although the CSB as implemented here supports 8 kB of in-memory computing for greater flexibility. Future optimizations can explore different bank sizes, bitwidths of

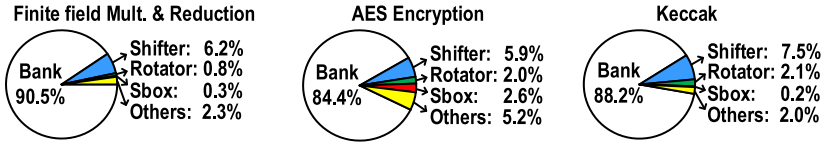


Fig. 14. Simulated power breakdown of different security functions.

TABLE IV
COMPARISON TABLE OF DIFFERENT CRYPTO ALGORITHMS AND DESIGNS

Applications		Designs	#cycles	Freq (MHz)	Time (us)	(Norm.)	Energy (nJ)	(Norm.)
AES		Baseline	6358	24	265	(1x)	64.2	(1x)
		[17]	5429	0.847	6410	(24x)	10259	(160x)
		[18]	20	1000	0.02	(7.5E-5x)	124	(1.93x)
		Recryptor	726	28.8	25.2	**(0.1x)**	7.05	**(0.11x)**
Finite Field Multiplication + Reduction	163 bits	Baseline	5966	24	249	(1x)	62.4	(1x)
		Recryptor	678	28.8	23.5	**(0.09x)**	9.30	**(0.15x)**
	233 bits	Baseline	8921	24	372	(1x)	93.4	(1x)
		[10]	3672	48	76.5	(0.21x)	45.9	(0.49x)
		Recryptor	826	28.8	28.7	**(0.08x)**	11.3	**(0.12x)**
	283 bits	Baseline	10809	24	450	(1x)	113	(1x)
		Recryptor	916	28.8	31.8	**(0.07x)**	12.6	**(0.11x)**
	409 bits	Baseline	19319	24	805	(1x)	202	(1x)
		Recryptor	1246	28.8	43.3	**(0.05x)**	17.1	**(0.08x)**
Keccak		Baseline	23015	24	959	(1x)	238	(1x)
		[12]	15427	100	154	(0.16x)	168	(0.7x)
		Recryptor	3329	28.8	116	**(0.12x)**	48.7	**(0.2x)**

[12,17,18]: Simulation only, no silicon implementation. [17]: 350nm; [18]: 45nm; [12]: 130nm; [10]: No technology given, Cortex M0+ processor; only include mult., no reduction.

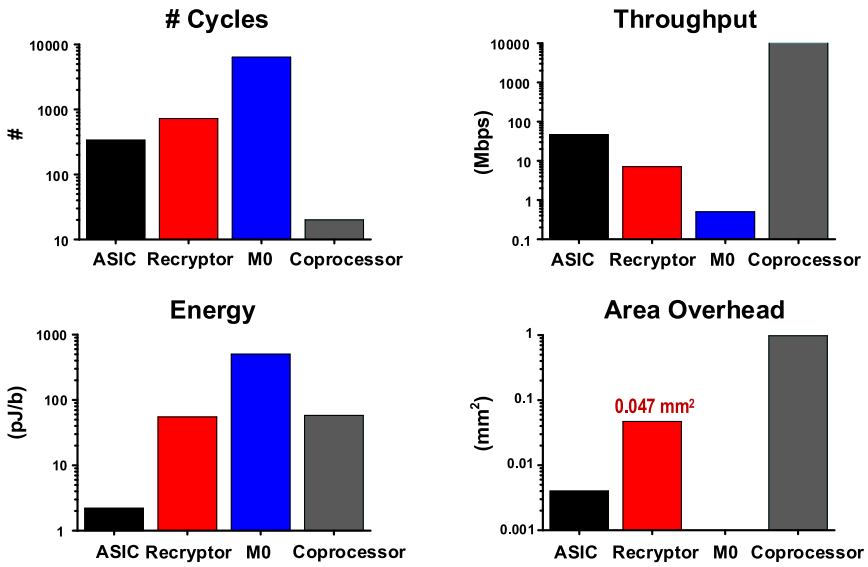


Fig. 15. Comparison of an ASIC [11], Recryptor, baseline, and a coprocessor [18] design for AES.

in-memory operations, and different near-memory modules, depending on the desired algorithms.

B. Performance and Overhead

The tradeoff between performance and hardware overhead is an important factor for the proposed Recryptor design. The hardware overhead of Recryptor is highly related with the supported functions' width, i.e., the parallelism of in-memory and near-memory computing. For the in-memory part, it is related with the peripherals required for each column (Fig.2), which includes two sense amplifiers an XOR and a 3-to-1 MUX. Regarding the implementation in this paper,

TABLE V
EFFECTIVE MEMORY CAPACITY FOR COMPUTING AND THE REQUIRED MODULES

Algorithm	Required Memory Size			Shifter	Rotator	Sbox
	bits/word	# Words	Total Size (bytes)			
AES - 128 bits	128	26	416	x		x
ECC - 163 bits	163	36	734	x		
ECC - 233 bits	233	36	1050	x		
ECC - 283 bits	283	36	1280	x		
ECC - 409 bits	409	36	1840	x		
Keccak	320	20	800	x	x	
All	512	36	1840	x	x	x

the column peripheral is 46% of height for the 128-row bitcell array. For the near-memory part, it is related with the added ASIC designs.

As shown in Section VI-B, the performance comparisons among Recryptor, ASICs, and coprocessors vary for different algorithms, which is dependent on the parallelism exploited by the hardware implementations. For example, on AES, Recryptor can perform 128-bit wide XORS, however, we used an 8-bit S-box design with a one cycle delay for area and energy reasons. This creates a performance bottleneck, which explains why the average performance of AES on Recryptor is lower than that of a pipelined ASIC design [11]. However, significantly higher performance can be achieved with a larger S-box or more shift functions on other sub-banks. On the other hand, on Keccak, our Recryptor implementation achieves throughput similar to that of an ASIC design. This is because Recryptor can exploit 320-bit wide parallelism in Keccak, which is also true of ASIC designs [12].

In addition, the memory bus width can also be a tradeoff factor, especially in the protocol level (e.g., transport layer security [36]). But just for the cryptographic operations, it is currently mostly done with long-width computation inside the CSB of Recryptor.

C. Power Analysis Attack

The memory bus is not trusted and is vulnerable to physical attacks, for example, by probing [33]. There have been expensive techniques addressed to solve this problem, for example, oblivious RAM [34] uses address obfuscation and InvisimMem [35] requires 3-D integration to stack DRAM layers on top of logic layers. The Recryptor's idea of compute memories could provide inexpensive encryption to protect the memory bus, taking the attack model to treat in-memory and near-memory as a whole module, but not to probe the internal connection inside CSB.

In addition, a preliminary differential power analysis (DPA) attack is analyzed on Recryptor, for key extractions in AES algorithms. An AES ASIC with 8-bit datapath is also included as a baseline. The attack setup is to first run Hspice simulation on the netlist of the baseline and Recryptor's CSB, and then use MATLAB [31] to calculate correlations and extract every 8-bit key. Each power trace has 20 executions of the 128-bit plaintexts with 1 ns sampling time, while keep the same key each time. The DPA result of baseline and Recryptor's CSB is shown in Figs. 16 and 17. The minimal number of traces to reveal the first byte of the correct key (0×2 bit) is 20 traces for baseline, and 300 traces for Recryptor's CSB. An ideal analysis of DPA attack is shown here, but considerations of real-world problems (e.g., clock jitter) and other attacks (e.g., fault injection attack) are suggested for further analysis.

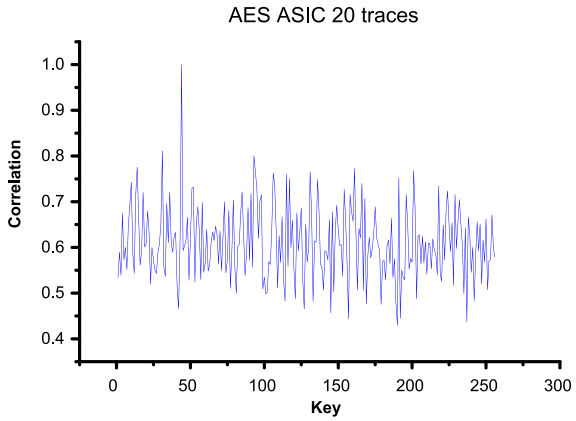


Fig. 16. Simulated DPA on baseline AES ASIC with 20 traces.

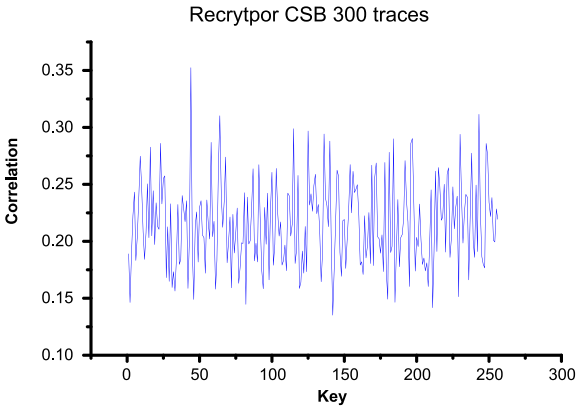


Fig. 17. Simulated DPA on recryptor's CSB with 300 traces.

VIII. CONCLUSION

There are many challenges in IoT security due to the limited computational resources and required flexibility. Current ASICs and coprocessors have limitations in different aspects. In this paper, we proposed a new architecture called Recryptor, which uses in-memory and near-memory computing to efficiently support large vector calculations for crypto algorithms. It maintains programmability and has over 80% runtime and energy savings compared with a baseline processor architecture. Overall, Recryptor is a good intermediate solution in terms of balancing area, energy, throughput, and programmability.

ACKNOWLEDGMENT

The authors would like to thank TSMC University Shuttle Program for chip fabrication. They also would like to thank Kaiyuan Yang and Supreet Jeloka for their input and discussions.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[2] V. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 218. New York, NY, USA: Springer-Verlag, 1986, pp. 417–426.

[3] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, no. 177, pp. 203–209, Jan. 1987.

[4] *STMicroelectronics STSAFE-A100*. Accessed: Oct. 2017. [Online]. Available: http://www.st.com/en/secure-mcus/stsafe-a100.html

[5] NIST. (Nov. 2001). *Advanced Encryption Standard (AES), FIPS PUBS 197*. [Online]. Available: http://csrc.nist.gov/archive/aes/

[6] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Keccak specifications," Submission to NIST (Round 3), 2011. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha3/Round3/submissions_rnd3.html

[7] *Algorithms, Key Size and Parameters Report*, ENISA, Heraklion, Greece, 2014.

[8] *ARM CORTEX-M Series*. Accessed: Oct. 2017. [Online]. Available: http://www.arm.com/products/processors/cortex-m

[9] D. F. Aranha, R. Dahab, J. López, and L. B. Oliveira, "Efficient implementation of elliptic curve cryptography in wireless sensors," *Adv. Math. Commun.*, vol. 4, no. 2, pp. 169–187, May 2010.

[10] R. de Clercq, L. Uhsadel, A. Van Herrewege, and I. Verbauwhede, "Ultra low-power implementation of ECC on the ARM cortex-M0+," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jun. 2014, pp. 1–6.

[11] Y. Zhang, K. Yang, M. Saligane, D. Blaauw, and D. Sylvester, "A compact 446 Gbps/W AES accelerator for mobile SoC and IoT in 40 nm," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Honolulu, HI, USA, Jun. 2016, pp. 1–2.

[12] P. Pessl and M. Hutter, "Pushing the limits of SHA-3 hardware implementations to fit on RFID," in *Cryptographic Hardware and Embedded Systems—CHES* (Lecture Notes in Computer Science), vol. 8086, G. Bertoni and J.-S. Coron, Eds. Heidelberg, Germany: Springer, 2013, pp. 126–141.

[13] E. Wenger, M. Feldhofer, and N. Felber, "Low-resource hardware design of an elliptic curve processor for contactless devices," in *Information Security Applications—WISA* (Lecture Notes in Computer Science), vol. 6513, Y. Chung and M. Yung, Eds. Heidelberg, Germany: Springer, 2010, pp. 92–106.

[14] S. Mathew *et al.*, "340 mV–1.1 V, 289 Gbps/W, 2090-gate NanoAES hardware accelerator with area-optimized encrypt/decrypt GF$(2^4)^2$ polynomials in 22 nm tri-gate CMOS," *IEEE J. Solid-State Circuits*, vol. 50, no. 4, pp. 1048–1058, Apr. 2015.

[15] 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 J. Solid-State Circuits*, vol. 46, no. 4, pp. 767–776, Apr. 2011.

[16] J.-W. Lee, S.-C. Chung, H.-C. Chang, and C.-Y. Lee, "Processor with side-channel attack resistance," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2013, pp. 50–51.

[17] M. Hutter, M. Feldhofer, and J. Wolkerstorfer, "A cryptographic processor for low-resource devices: Canning ECDSA and AES like sardines," in *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*, vol. 6633. New York, NY, USA: Springer-Verlag, 2011, pp. 144–159.

[18] G. Sayilar and D. Chiou, "Cryptoraptor: High throughput reconfigurable cryptographic processor," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, Nov. 2014, pp. 155–161.

[19] Y. Zhang *et al.*, "Recryptor: A reconfigurable in-memory cryptographic Cortex-M0 processor for IoT," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Jun. 2017, pp. 264–265.

[20] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory," *IEEE J. Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, Apr. 2016.

[21] Q. Dong *et al.*, "A 0.3 V VDDmin 4+2T SRAM for searching and in-memory computing using 55 nm DDC technology," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Jun. 2017, pp. 160–161.

[22] L. Sigal *et al.*, "Circuit design techniques for the high-performance CMOS IBM S/390 parallel enterprise server G4 microprocessor," *IBM J. Res. Develop.*, vol. 41, no. 4, pp. 489–503, Jul. 1997.

[23] J. Rabaey, *Digital Integrated Circuits: A Design Perspective*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1995.

[24] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. New York, NY, USA: Springer-Verlag, 2004.

[25] P. Schwabe, B.-Y. Yang, and S.-Y. Yang, "SHA-3 on ARM11 processors," in *Proc. 5th Int. Conf. Cryptol. Africa*, Jul. 2012, pp. 324–341.

[26] Y. Wang, Y. Shi, C. Wang, and Y. Ha, "FPGA-based SHA-3 acceleration on a 32-bit processor via instruction set extension," in *Proc. IEEE Int. Conf. Electron Devices Solid-State Circuits (EDSSC)*, Singapore, Jun. 2015, pp. 305–308.

[27] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Austin, TX, USA, Feb. 2017, pp. 481–492.

[28] D. Aranha and C. P. L. Gouvêa. (2013). *RELIC Cryptographic Toolkit*. [Online]. Available: https://code.google.com/p/relic-toolkit/

[29] *SUPERCOP*. Accessed: Oct. 2017. [Online]. Available: https://github.com/floodyberry/supercop

[30] *Tiny-AES128-C*. Accessed: Oct. 2017. [Online]. Available: https://github.com/kokke/tiny-AES128-C

[31] *DPAbook Online Material*. Accessed: Oct. 2017. [Online]. Available: http://dpabook.iaik.tugraz.at/onlinematerial/matlabscripts/

[32] E. Wenger and M. Hutter, *Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations*. Berlin, Germany: Springer, 2012, pp. 256–271.

[33] (2006). *I2C Bus Monitor*. Accessed: Apr. 1, 2016. [Online]. Available: http://www.jupiteri.com/

[34] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[35] S. Aga and S. Narayanasamy, "InvisiMem: Smart memory defenses for memory bus side channel," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 94–106.

[36] *The Transport Layer Security (TLS) Protocol, Version 1.2*. Accessed: Oct. 2017. [Online]. Available: https://tools.ietf.org/html/rfc5246

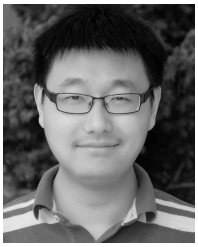
Yiqun Zhang (S'14) received the B.S. degree in electrical engineering and computer science from the University of Michigan, Ann Arbor, MI, USA, and Shanghai Jiaotong University, Shanghai, China, in 2013, and the M.S. degree from the University of Michigan in 2016, where she is currently pursuing the Ph.D. degree.

Her current research interests include security system, fault tolerance circuits, and error-resilient systems.

Ms. Zhang was a recipient of the Marian Sarah Parker Scholars Program Scholarship in 2012 from the University of Michigan.

Li Xu (S'15) received the B.Eng. degree in automation from Tongji University, Shanghai, China, in 2009, and the M.S. degree in electrical and computer engineering from Northeastern University, Boston, MA, USA, in 2016. He is pursuing the Ph.D. degree with the University of Michigan, Ann Arbor, MI, USA.

From 2009 to 2011, he was an IC Design Engineer with Ricoh Electronic Devices Shanghai Company, Ltd., where he was involved in LDO and dc/dc converter projects. In 2015, he was a Design Intern with Linear Technology Corporation, Colorado Springs, CO, USA. His current research interest includes energy-efficiency mixed-signal circuit design.



Qing Dong (S'14) received the B.S. and M.S. degrees in microelectronics from Fudan University, Shanghai, China, in 2010 and 2013, respectively. He is currently pursuing the Ph.D. degree with the University of Michigan, Ann Arbor, MI, USA.

His current research interests include memory circuits design, and monitoring circuits design for process variation and BTI.

Mr. Dong was a recipient of the Best Paper Awards at the 2012 IEEE International Conference on Solid-State and Integrated Circuit Technology, the 2015 IEEE International Symposium on Circuits and Systems, and the 2016 IEEE Symposium on Security and Privacy.

Jingcheng Wang (S'15) received the B.S. and M.S. degrees in electrical and computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 2014 and 2017, respectively, where he is currently pursuing the Ph.D. degree in electrical engineering.

His current research interests include high-speed and low-power VLSI circuits and systems.

Mr. Wang was a recipient of the Dwight F. Benton Fellowship in 2015 from the University of Michigan.

David Blaauw (M'94–SM'07–F'12) received the B.S. degree in physics and computer science from Duke University, Durham, NC, USA, in 1986, and the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1991.

He was with Motorola, Inc., Austin, TX, USA, where he was the Manager of the High Performance Design Technology Group. Since 2001, he has been on the faculty at the University of Michigan, Ann Arbor, MI, USA, where he is currently a Professor. He has authored or co-authored over 500 papers and holds 50 patents. His current research interests include VLSI design including near-threshold and subthreshold design for ultralow power mm-scale sensor nodes.

Dr. Blaauw was the Technical Program Chair and the General Chair of the International Symposium on low-power electronic and design. He was also the Technical Program Co-Chair of the ACM/IEEE Design Automation Conference and a member of the ISSCC Technical Program Committee.

Dennis Sylvester (S'95–M'00–SM'04–F'11) received the Ph.D. degree in electrical engineering from the University of California, Berkeley, CA, USA, in 1999.

He has held research staff positions with the Advanced Technology Group, Synopsys, Mountain View, CA, USA, the Hewlett-Packard Laboratories, Palo Alto, CA, USA, and Visiting Professorships at the National University of Singapore, Singapore, and Nanyang Technological University, Singapore. He is currently a Professor of electrical engineering and computer science with the University of Michigan, Ann Arbor, MI, USA, and the Director of the Michigan Integrated Circuits Laboratory, Ann Arbor, a group of 10 faculty and more than 70 graduate students. He is a Co-Founder of Ambiq Micro, Austin, TX, USA, a fabless semiconductor company developing ultralow-power mixed-signal solutions for compact wireless devices. He has authored or co-authored over 375 articles along with one book and several book chapters. He holds 20 U.S. patents. His current research interests include the design of millimeter-scale computing systems and energy-efficient near-threshold computing.

Dr. Sylvester served on the Executive Committee of the ACM/IEEE Design Automation Conference. He serves on the Technical Program Committee of the IEEE International Solid-State Circuits Conference and as a Consultant and Technical Advisory Board Member for electronic design automation and semiconductor firms in his research areas. He has served as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS and the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS, and a Guest Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II. He was a recipient of the NSF Career Award, the Beatrice Winner Award at ISSCC, an IBM Faculty Award, an SRC Inventor Recognition Award, eight Best Paper Awards and Nominations, the ACM SIGDA Outstanding New Faculty Award, the University of Michigan Henry Russel Award for distinguished scholarship, and the David J. Sakrison Memorial Prize as the most outstanding research for his dissertation in the Electrical Engineering and Computer Sciences Department, University of California, Berkeley.