Received 1 July 2021; revised 30 October 2021; accepted 7 November 2021. Date of current version 9 December 2021.
Digital Object Identifier 10.1109/OJCAS.2021.3127273

ChaCha20-in-Memory for Side-Channel Resistance in IoT Edge-Node Devices

M. AAMIR, SOMYA SHARMA, AND ANUJ GROVER
Department of ECE, Indraprastha Institute of Information Technology Delhi, New Delhi 110020, India
This article was recommended by Associate Editor A. Bermak.
CORRESPONDING AUTHOR: M. AAMIR (e-mail: aamir17167@iiitd.ac.in)

ABSTRACT IoT edge devices process the data collected, which can contain sensitive information related to the user. It is crucial to incorporate robust encryption algorithms considering the resource and power budget of these devices. In this paper, we present a power-based SCA-resistant implementation of the ChaCha20 encryption algorithm for low-end devices by utilizing memory arrays. The 10T SRAM-based implementation performs simple operations (like NAND, NOR, XOR) on the bitlines and other operations like addition/subtraction, shifting, rotation on custom-designed in-memory elements tightly coupled to sense amplifiers (SA). The design is verified for multiple test vectors to generate power consumption signatures. Welch's t-test is performed on these signatures to demonstrate that the design is highly resistant to power-based SCA. The proposed implementation of ChaCha20 runs at 250MHz at a 1.2V supply, in 65nm Low Standby Power (LSTP) technology, achieving a speedup of around 7 times in terms of execution time compared to the ARM Cortex A9 processor.

INDEX TERMS ChaCha20, edge computing, encryption, in-memory compute, IoT security, SCA.

I. INTRODUCTION

IOT EDGE-NODE devices are now being used everywhere, from smart homes to industries. They collect information from the surroundings, process it, and upload the necessary data to the cloud. The data collected by these devices often include sensitive or safety-critical information and needs to be protected from theft and tampering [1]. To achieve this, encryption algorithms are integrated into the devices. They use private keys to encrypt and decrypt the data, hence ensuring confidentiality [2]. The security of these algorithms is dependent on the safety of their private keys. In IoT devices, the private keys are generally stored during the manufacturing process and are optionally updated during the entire life-cycle of the device [3]. It makes these devices a vulnerable target for side-channel attacks (SCA). As an alternative, Physically Unclonable Functions (PUFs) are also being used to generate private keys, which can't be anticipated or replicated [5], [6].

In SCA, the adversary takes advantage of implementation-specific characteristics to uncover or build a logical relationship for some of the secret parameters involved in the algorithm [4]. These characteristics are the measurable parameters like power consumption, execution time generated due to the algorithm's execution in the underlying hardware. For instance, the work shown in [7] conducts a power-analysis-based attack on an AES-128 cipher, and [8], [9] demonstrates cache-based SCA.

In power-based SCA, the power consumption profile of the machine running the encryption algorithm is analyzed to extract some valuable information like the private key. These attacks work on the basis that the power consumed in computing bit value '1' is different than when computing bit value '0' [10]. This is valid when the algorithm is being executed in a processor. The existing countermeasures to SCA, as mentioned in [11], are resource and power-intensive. Therefore, these methods to secure the chip against SCA can't be used in edge devices that are resource-constrained and operate on a limited power budget. Ideally, methods to secure edge devices from SCA should have no impact on the energy, area, and performance of the system.

IMC offers a promising solution to secure low-end IoT edge devices. It enables part of the logic operations to be executed on the fly during read access. This architecture exploits the flexibility of the memory cells to be dually used for storing data and for computing. In this work, we show that the power consumption of logic operations implemented using in-memory-compute architecture can be designed to be independent of the data being processed. Here, it must be noted that, the design of the ChaCha20 algorithm makes it naturally resistant to timing-based SCA [12]. This paper focuses explicitly on designing a power-based SCA-resistant ChaCha20 stream cipher implementation for resource-constrained devices.

The rest of the paper is organized as follows. In Section II, we discussed the previous work done on memory-based implementations of encryption algorithms. In Section III, we elaborate on the advantages of using IMC architecture in effectively avoiding SCA in edge devices. In Section IV, we give a brief overview of the ChaCha20 stream cipher and its implementation. In Section V, we provide the detailed implementation of in-memory computing units, followed by Section VI, in which we describe the verification setup and explain the methodology of power-based side-channel leakage assessment. Then in Section VII, we present the results of side-channel leakage assessment and performance comparison. Finally, the conclusion is drawn in Section VIII.

II. RELATED WORKS

Many existing hardware implementations for encryption algorithms have been proposed in the academia that are based on cryptographic co-processors like [13]–[15]. On the other hand, the countermeasures for power-based side-channel attacks are also being studied as discussed in [11] but at the cost of significant area and energy overhead. However, memory-based hardware encryption is rarely used while addressing the problems of side-channel attacks. The existing memory-based implementations of encryption algorithms focus majorly on the power and performance gain, as discussed below.

In [16], the authors propose a reconfigurable cryptographic processor (Recryptor) implementing the AES algorithm. In this paper, a block of memory is configured to implement in-memory and near memory compute operations. Only the XOR operation is implemented as in-memory. Rest of the operations namely, shifting, rotation, and substitution box (SBOX) are implemented by cross-wiring, combinational logic, and latches added near the memory. Similar to this, [17] proposes an in-memory implementation of the Advanced Encryption Scheme (AES) algorithm for non-volatile memories (NVMs). In this, various components of the AES algorithm like the Substitution box (SBOX), Mix-column blocks are implemented using a combination of look-up tables (LUTs) and combinational logic.

The near memory circuitry used in these works was independent of the array and could not synergize effectively with in-memory elements, leading to a significant area overhead and consequently power consumption. The architecture proposed in our work focuses entirely on IMC elements. The proposed adder circuitry takes benefits from the IMC operation, thereby reducing the area overhead. To the best of our knowledge, our work is the first to propose IMC-based ChaCha20 algorithm implementation with resistance to power-based SCA.

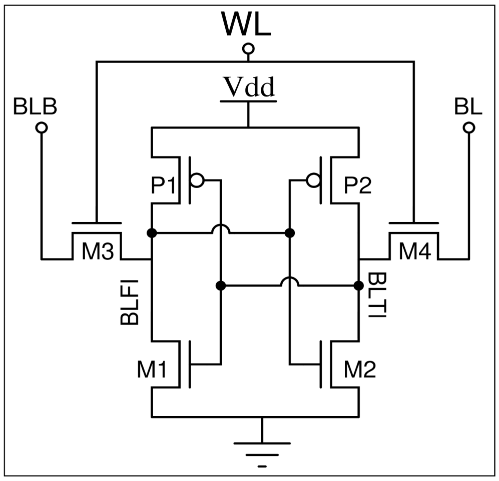


FIGURE 1. 6T SRAM Bitcell.

III. ADVANTAGES OF IMC BASED ARCHITECTURE

As discussed in [19], the IoT edge devices have limited budget for power and resources to implement robust security measures. Also, to enable edge computing, they are burdened with local processing. This leaves out minimal resource and power budget to incorporate encryption algorithms with effective measures against SCA without degrading the device's performance. Moreover, existing methods to mitigate power-based SCA as discussed in [11], require a lot of additional resources which are not available in these low-end devices. Furthermore, the ChaCha20 algorithm requires high bit-width computations and contains multiple rounds of operation on the input data. While a higher number of rounds increases the diffusion between input and cipher-text (hence the security), it results in higher execution time. Implementation of the algorithm on the lower-end processor results in huge performance overhead and is not within the budgeted capability of most IoT edge devices.

IMC capabilities in SRAMs enable high bandwidth of operations within a small power envelope. So, as proposed, in-memory implementation can be considered as a solution for the trade-off mentioned above between security, performance, and resources. Since memory elements form a significant area in any computing system, memory bitcells are used for storage and computation purposes. Given below are a few benefits of the in-memory-compute architecture.

1) *Power-consumption pattern masking capability:* A conventional 6T SRAM cell is shown in Fig. 1. It is a pair of inverters (M1-P1 and M2-P2) cross-coupled to form a latch. M3 and M4 devices are used to access the latch and perform a read/write operation. The bitlines BL and BLB are initially precharged to Vdd (logic 1), followed by enabling the word-line, i.e., charging it to logic 1. Then, depending on the data-bit stored in the bitcell at BLTI and BLFI nodes, one of the bit lines

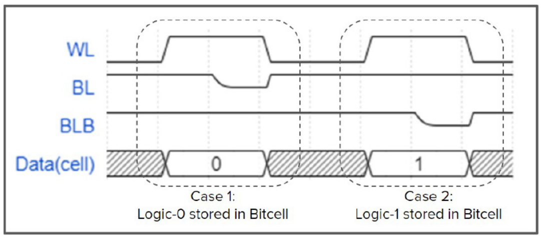


FIGURE 2. **Read operation waveform of the 6T bitcell.**

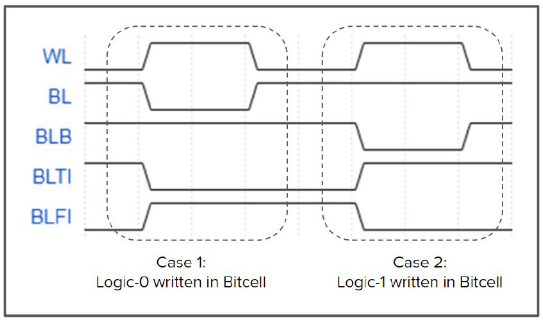


FIGURE 3. **Write operation waveform of the 6T bitcell.**

(BL or BLB) gets discharged through access and pull-down transistors path M4-M2 or M3-M1, respectively. This generates a differential voltage drop between the bit-lines, which is then detected using a sense amplifier (SA).

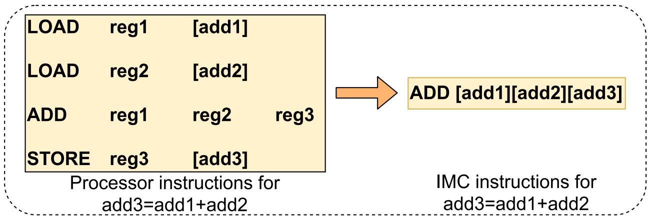
Similarly, to write a data-bit, either of the bitlines is precharged to gnd, and the other is precharged to Vdd. Then, on enabling the word-line, logic 0 and logic 1 are stored in the complementary nodes (BLTI and BLFI) depending on the discharge path formed.

It is observed that the discharge pattern (hence power consumption) of the bitlines as a pair is identical in the read-0&1 and write-0&1 as shown in Fig. 2 and Fig. 3, respectively. Hence, it can be safely concluded that power consumed in read and write operation in a memory array is independent of data being read from or written to the array.

Additionally, leakage current of other non-activated bitcells in the memory array helps to further obfuscate the relation between the data being processed and the power consumption patterns.

2) *Single cycle operation:* In-memory architecture does not require the data to be loaded in temporary registers from memory for computing. Here, the data is read from memory, computed by IMC elements on the fly, and the modified data is written back to the memory in the same clock cycle. This cuts back on the extra cycles required for load and store instructions, thereby reducing the number of clock cycles per instruction as explained in [18] and illustrated in Fig. 4.

3) *Overcomes von-Neumann Bottleneck:* In *von-Neumann* architecture, limited data bus width acts as a bottleneck to wide bandwidth computations, hitting a "Memory Wall" for processing speed. This significantly affects the throughput, and a large fraction of energy is consumed in moving data back and forth between memory and the processor. The proposed architecture completely bypasses the need for data transfers in SIMD operations by taking compute units to memory, thereby increasing throughput and reducing power consumption.



LOAD	reg1	[add1]	
LOAD	reg2	[add2]	
ADD	reg1	reg2	reg3
STORE	reg3	[add3]	

Processor instructions for add3=add1+add2

ADD [add1][add2][add3]

IMC instructions for add3=add1+add2

FIGURE 4. **Addition operation can be done in 1 cycle with the updated OPCODE in IMC instead of 4 cycles in a regular processor.**

4) *Parallel processing:* The input/output port width of a memory array is limited to standard data bus width, but the in-memory implementation allows us to work on custom bit widths. This enables us to exploit bit-level parallelism for better performance and constant-time execution, preventing timing-based SCA.

IV. CHACHA20

ChaCha20 is a stream cipher that is used to generate pseudo-random bits of data called keystream. The given plain-text data is combined with the generated keystream by XOR operation to generate the cipher-text. Similarly, the cipher-text can be decrypted by doing an XOR operation on the cipher-text and the keystream.

The ChaCha20 algorithm starts with an initial state of 16 words, 32-bits each. It comprises a 128-bit constant, a 256-bit private key, a 32-bit block counter which is incremented from zero, and a 96-bit nonce which is unique for a keystream. The initial state is arranged in the form of a 4×4 matrix as shown in Fig. 6(a). A series of 20 rounds, alternating between even and odd rounds are applied to the initial state-matrix, generating a 512-bit keystream. The flow of the ChaCha20 algorithm is shown in Fig. 5. Each round contains four quarter-round (QR) functions. The even and odd rounds differ by their inputs, as shown in Fig. 6(b). Each QR takes 4 words, 32-bits each, as input. It uses additions, XORs, and rotations to update the words, as shown in the flow diagram in Fig. 7.

V. CHACHA20 IN-MEMORY ARCHITECTURE

The top-level architecture of the in-memory-ChaCha20 is presented in Fig. 8. It uses IMC operations to implement the ChaCha20 algorithm inside the memory array without requiring any external processing unit. The 512-bit input data is stored in the memory array as a matrix of dimensions 4×128, as shown in Fig. 6(a), such that the words required in a single QR operation are aligned in the same column of the memory array. Since each column has a dedicated IMC unit, storing data in the above form allows four QRs to

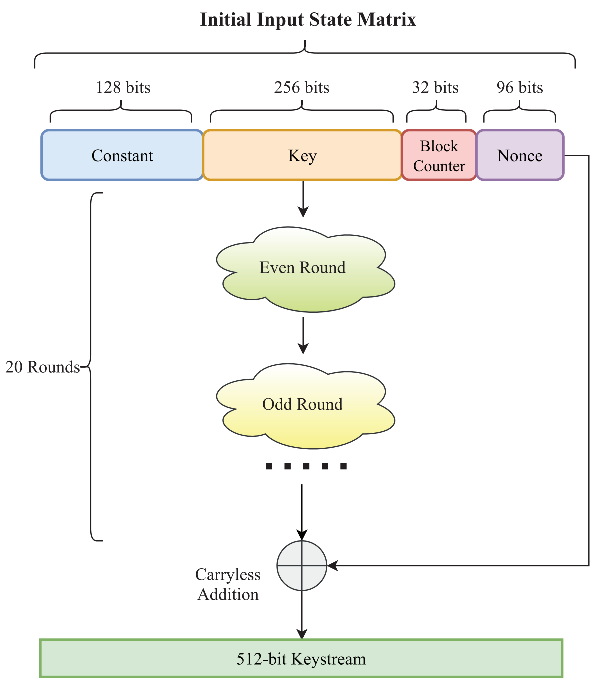


FIGURE 5. ChaCha20 algorithm flow.

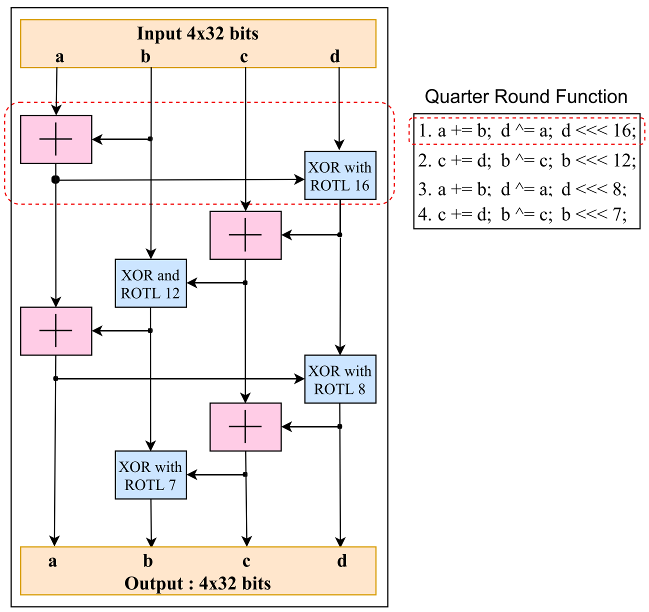
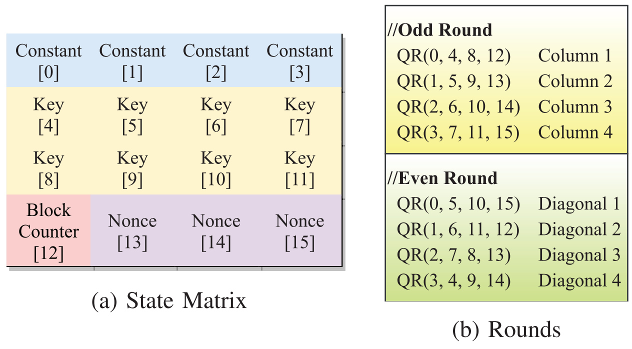


FIGURE 7. ChaCha20: Quarter-Round.



Constant [0]	Constant [1]	Constant [2]	Constant [3]
Key [4]	Key [5]	Key [6]	Key [7]
Key [8]	Key [9]	Key [10]	Key [11]
Block Counter [12]	Nonce [13]	Nonce [14]	Nonce [15]

(a) State Matrix

//Odd Round	
QR(0, 4, 8, 12)	Column 1
QR(1, 5, 9, 13)	Column 2
QR(2, 6, 10, 14)	Column 3
QR(3, 7, 11, 15)	Column 4
//Even Round	
QR(0, 5, 10, 15)	Diagonal 1
QR(1, 6, 11, 12)	Diagonal 2
QR(2, 7, 8, 13)	Diagonal 3
QR(3, 4, 9, 14)	Diagonal 4

(b) Rounds

FIGURE 6. Initial State Matrix and Even/Odd Rounds.

be executed in parallel. The row-decoders of the array are designed to enable multiple read-word lines simultaneously, allowing in-memory compute operations. I/O circuitry of each column of the memory array contains a separate single-ended SA for RBL and RBLB read-bitlines, write-drivers for write-bitlines, Adder/XOR unit, and rotation unit.

A. IMC OPERATION

In the proposed SRAM-based IMC array, NOR and AND logic functions are executed during the read operation on the data of selected rows, explained later in Section V-B. These results can be passed either to an XOR unit or the rotation circuit or the adder unit based on the requirement. The computed data is then written back to the memory bitcells in the same cycle. The in-memory-ChaCha20 architecture does the operations mentioned above in two parts. It uses bitlines for in-memory NOR and AND logic functions and custom-designed in-memory elements to execute the remaining operations. In a single round execution Fig. 7, all the required operands are aligned in the same column of the memory array to exploit parallel execution of QRs. The steps in a single QR are given below.

1) Firstly, the two rows which contain the operands for an addition/XOR operation, say A and B (aligned in the same column) are activated simultaneously. The RBL and RBLB of the respective column output the bitwise AND and NOR values of A and B.
2) Next, the output from the above step is passed to "Adder/XOR Unit".
3) The output is then passed to the "Rotation Unit", which rotates the input data by a specified number of bits according to the algorithm and writes back to the memory array.
4) Each row of the memory array contains four words (128-bits), all of which are operated upon in parallel.
5) The memory array now contains the updated data. These steps (starting from 1) are now repeated for different operands to complete the execution of a single round.

Here it is to be noted that when switching between even and odd rounds, the rows of the state matrix Fig. 16 are shifted using rotation block, such that the elements required in a single QRs are aligned in the same column.

B. IN-MEMORY COMPUTING USING READ BITLINES

The concept of IMC is based on the idea that simple boolean functions can be executed equivalently to a read operation. The only difference between a normal read operation and an IMC operation is that multiple rows are activated simultaneously in the latter, and all the activated bitcells in the same column are allowed to control the read-bitlines simultaneously. For the ChaCha20 algorithm, the addition and

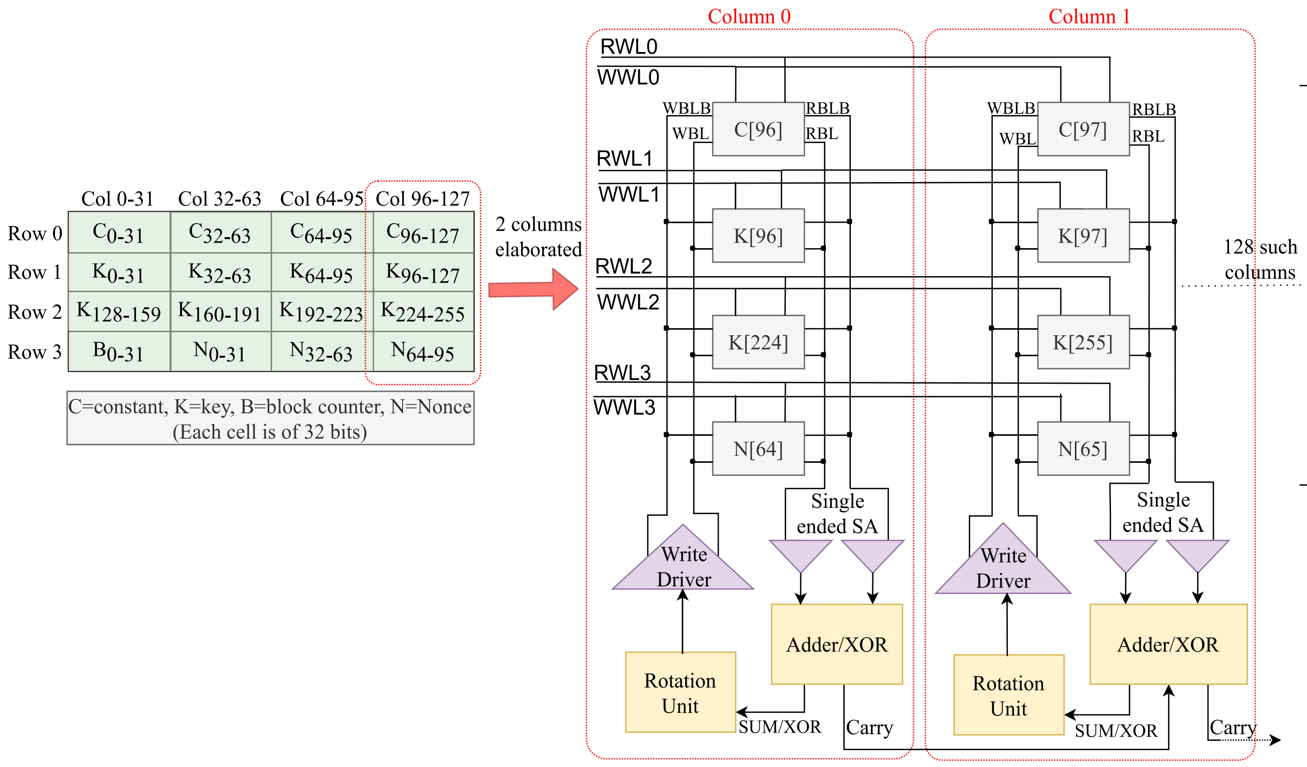

Col 0-31
Col 32-63
Col 64-95
Col 96-127
Row 0
Row 1
Row 2
Row 3
C_{0-31} C_{32-63} C_{64-95} C_{96-127}
K_{0-31} K_{32-63} K_{64-95} K_{96-127}
$K_{128-159}$ $K_{160-191}$ $K_{192-223}$ $K_{224-255}$
B_{0-31} N_{0-31} N_{32-63} N_{64-95}
C=constant, K=key, B=block counter, N=Nonce
(Each cell is of 32 bits)
2 columns elaborated
Column 0
Column 1
RWL0
WWL0
RWL1
WWL1
RWL2
WWL2
RWL3
WWL3
WBLB
WBL
RBLB
RBL
C[96]
C[97]
K[96]
K[97]
K[224]
K[255]
N[64]
N[65]
128 such columns
Single ended SA
Write Driver
Adder/XOR
Rotation Unit
SUM/XOR
Carry

FIGURE 8. Proposed In-memory-ChaCha20 Architecture.

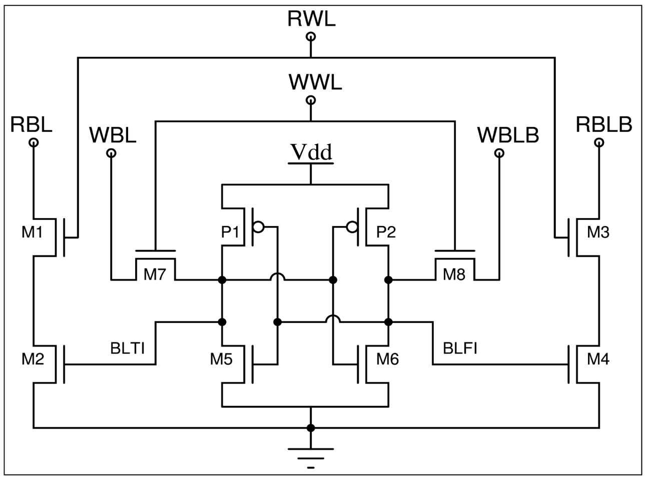


FIGURE 9. 10T SRAM Bitcell.

XOR operations on A and B (stored in the same column) are implemented as in-memory from $A.B$ (AND) and $\overline{A+B}$ (NOR) values. These AND and NOR boolean operations are executed efficiently by exploiting the read mechanism in the memory bitcell.

1) 10-TRANSISTOR SRAM BITCELL

The memory array used for in-memory compute in this paper is made using 10T SRAM bitcells. The bitcell is represented in Fig. 9. The bitcell consists of a conventional 6T SRAM cell used exclusively for a write operation and two single-ended read-ports, used for in-memory boolean logic execution and read operation. The read ports are added to the bitcell by using a pair of two series-connected NMOS transistors viz. (M2 and M1) and (M4 and M3). The gate terminals of these transistor pairs are connected to BLTI, RWL, BLFI ,and RWL, respectively. The read bitlines (RBL and RBLB) contact the bitcell at the drain end of the read transistors (M1 and M3), such that the internal nodes (BLTI/BLFI) of the bitcell are isolated from any direct contact with the read bitlines.

In the standard 6T memory array, read and write bitlines are shared. Short-circuit between multiple bitcells in a column storing complementary data could result in corruption of the data stored. For example, if one of the many bitcells stores a logic-0 and discharges the bitlines, this could cause a bit-flip at the internal nodes of the other participating cells of the same column. For this reason, 10T cell is selected for the IMC array in which decoupling of the read and write ports allows independent sizing of the transistors on the read and write path. It gives flexibility for implementing in-memory compute operations without corrupting the data. They allow having better readability and comparatively larger voltage swings at the read-bitlines without the risk of degrading the write-ability. Given below are the modes of operation of the bitcell.

- *Read-Mode:* In read mode, the read-bitlines RBL and RBLB are initially precharged to Vdd. RWL of the row of required bitcell is activated, providing the discharge path for bitlines. Based on the charge stored at BLTI

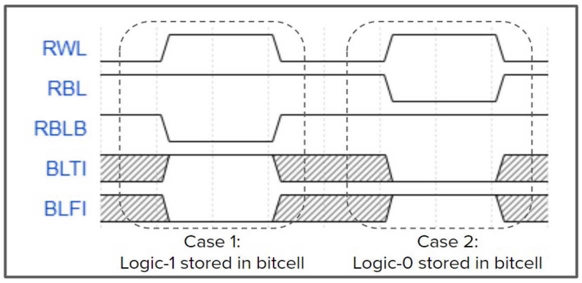


FIGURE 10. 10T Bitcell: Read Mode.

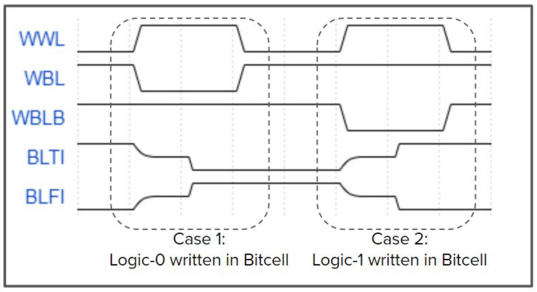


FIGURE 11. 10T Bitcell: Write Mode.

and BLFI nodes, either RBL or RBLB either discharges (logic 0) or both remain at a precharged voltage (logic 1) as shown in Fig. 10. The value read by the bitlines is then detected by an inverter-based SA at the I/O.

- *Write-mode:* In write-mode, WWL of the desired bitcell is activated, and data to be written is loaded on the write-bit line pair by the write drivers, as shown in Fig. 11.
- *IMC mode:* In IMC mode, both the read-bitlines RBL and RBLB are precharged initially to Vdd. The RWLs corresponding to the rows containing the two operands are activated simultaneously in the same read-cycle. The precharged bitlines will either discharge or remain constant based on the short-circuit between the two values sensed by the bitlines.
    - *NOR and AND operation:* The voltage level at the read-bitlines provides the output of the boolean operation. The read bitline, RBL, will discharge when any of the two activated bitcell stores a "0," equivalent to an "AND" boolean operation. On the contrary, the RBLB senses the complementary value stored in the bitcells, and it will be discharged only when both the bitcells stores a logic "1," equivalent to a logical NOR operation. The above functionality is shown in the Fig. 12.

C. CUSTOM-DESIGNED IN-MEMORY-COMPUTE ELEMENTS

Additional circuitry required for adder, XOR, and rotation operations is added near the SA to benefit from unbound bandwidth inside the memory, thereby allowing operations on 128-bits in parallel.

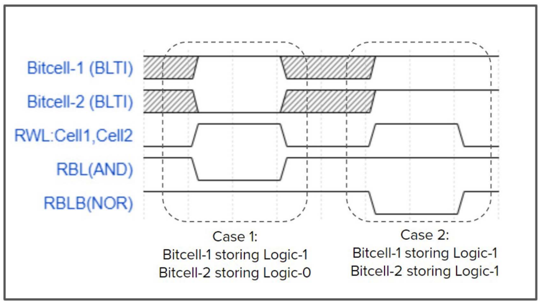

Bitcell-1 (BLTI)
Bitcell-2 (BLTI)
RWL:Cell1,Cell2
RBL(AND)
RBLB(NOR)
Case 1:
Bitcell-1 storing Logic-1
Bitcell-2 storing Logic-0
Case 2:
Bitcell-1 storing Logic-1
Bitcell-2 storing Logic-1

FIGURE 12. 10T Bitcell: IMC Mode.

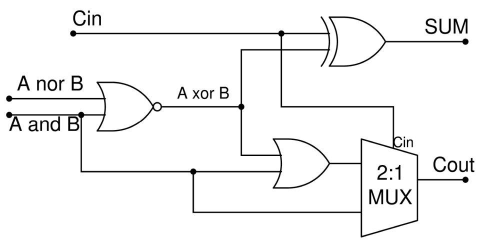


FIGURE 13. Proposed Adder.

TABLE 1. Area and latency comparison of RCA, CLA and IMC adders.

Adder	Area Overhead (per stage)	Latency (per stage)
RCA	2 2-input AND gates 2 2-input XOR gates 1 2-input OR gate	$\tau_{XOR} + N(\tau_{AND} + \tau_{OR})$ $+ \max((\tau_{AND} + \tau_{OR}), \tau_{XOR})$
Proposed IMC Adder	1 IMC-XOR (NOR Gate) 1 2-input XOR gate 1 2-input OR gate 1 2 : 1 MUX (Transmission) gate logic)	$\tau_{IMC-XOR} + \tau_{OR}$ $+N(\tau_{MUX}) + \tau_{XOR}$ (Midway between CLA and RCA)
CLA	N-1 multi-input AND gates 1 2-input AND gate 1 multi-input OR gate 1 XOR gate	$\max(\tau_{XOR}, \tau_{AND}) + \tau_{AND}$ $+\tau_{OR} + \tau_{XOR}$

- *XOR Circuit:* Outputs of in-memory boolean operations AND and NOR are passed to the NOR gate at the IO, resulting in XOR operation as demonstrated in [16].
- *Adder Circuit:* For a single-cycle 32-bit addition, a low latency adder is required while maintaining low area constraints. Ripple carry adders utilize the least area, but rippling carry at every stage leads to very low throughput. While on the other hand, carry look-ahead adders provide the lowest latency but at the cost of huge area overhead.

  Hence, an adder mid-way between ripple carry and carry look-ahead adder is proposed, as shown in Fig. 13. The latency and area of all the three adders area compared in the Table 1. This Adder takes the IMC-NOR and IMC-AND output of the input bits from the bitlines. The carry output for every bit is generated parallelly for both cases when input carry would be 1 or 0. The final output carry is selected using a MUX, with the carry-in signal (output carry from previous bit) as select-line. This reduces the carry propagation delay per stage. The sum, on the other hand, is generated in two parts. Firstly, the carry independent part, i.e., $A+B$ generated

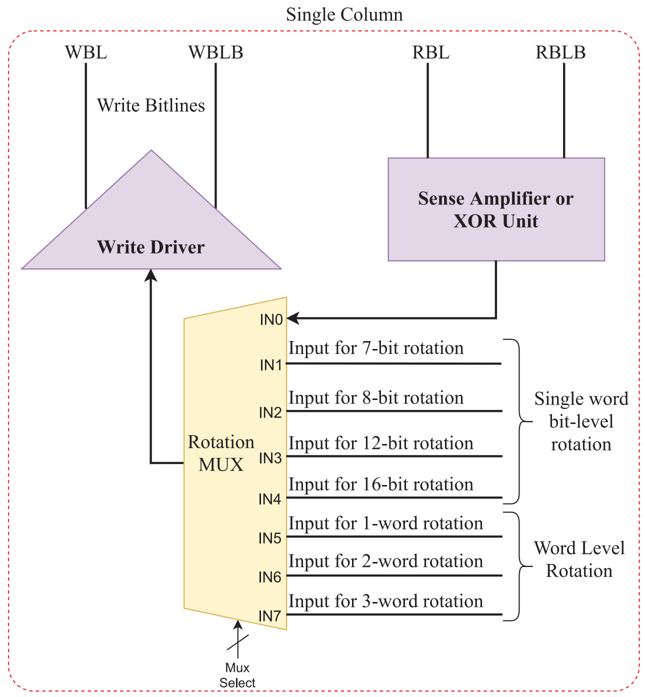

Single Column
WBL
WBLB
RBL
RBLB
Write Bitlines
Write Driver
Sense Amplifier or XOR Unit
Rotation MUX
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
Input for 7-bit rotation
Input for 8-bit rotation
Input for 12-bit rotation
Input for 16-bit rotation
Input for 1-word rotation
Input for 2-word rotation
Input for 3-word rotation
Single word bit-level rotation
Word Level Rotation
Mux Select

FIGURE 14. Rotator Circuit: Array output to write driver mapping.

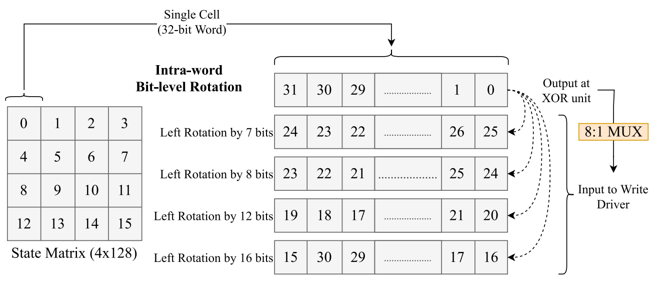


FIGURE 15. Bit-level Rotation in single word.

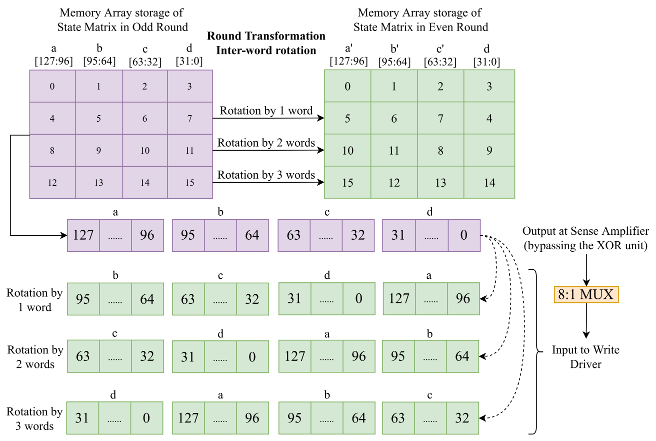

Memory Array storage of State Matrix in Odd Round
Round Transformation Inter-word rotation
Memory Array storage of State Matrix in Even Round
Rotation by 1 word
Rotation by 2 words
Rotation by 3 words
Output at Sense Amplifier (bypassing the XOR unit)
8:1 MUX
Input to Write Driver

FIGURE 16. Word-level Rotation.

parallelly for all the bits, and then the actual carry signal is used to calculate the final sum for each stage. The delay of the N-bit proposed adder can be calculated by the expression 1.

$$\tau_{delay} = \tau_{IMC-XOR} + \tau_{OR} + N(\tau_{MUX}) + \tau_{XOR}. \quad (1)$$

- *Rotation Circuit:* The circuit for rotation is shown in Fig. 14. There are two types of rotations required to implement the algorithm. First is the bit-level rotation, i.e., within a single word required in a QR Fig. 7. Second is the word-level rotation needed to align the words (required in the next QR) in the same column when the algorithm switches between even and odd rounds, as shown in Fig. 16. Rotation is executed by first reading a single row from the array, then latching the data at the output of SA, and then finally redirecting to different write drivers (based on rotation requirement) of the same row via IMC-XOR unit. The redirection is done using an 8×1 MUX, one for each column, having inputs from other columns and the output of the MUX connected to the write driver of its column. The select line for rotation MUX, *MUX Select*, is controlled by the memory array's control block. The rotation MUX, one for each column, is placed at the output of the XOR/Adder unit. The implementation for rotation using MUX for each column is shown in Fig. 14.
The mapping between XOR units to write driver using MUX for the two different types of rotation is discussed below.
  - *Bit-level Rotation (Quarter Round):* In QR, each of the 32-bits words is bit-rotated by a fixed amount after the XOR operation Fig. 7. The rotation is executed by passing the XOR output of required columns at the input of rotation MUX. The inputs to the MUX of 0^{th} column is shown in Fig. 15.
  - *Word-level Rotation (Column Alignment):* The data processed in the ChaCha20 algorithm is stored in the form of 4×128 state matrix (Fig. 16) in the memory array. It is done so that all the elements required in a single QR function can be placed in the same column (for IMC operations). This allows for all the four QR functions of an even/odd round to be executed parallelly. In order to align the words of the matrix in the same column, word-level rotation is performed. Here, when a row is read from the array, the words as a whole are redirected to specified write drivers. The output of the SA is redirected to write driver after passing through XOR/Adder unit. The mapping for this word-level rotation is shown in Fig. 16.

VI. VERIFICATION SETUP

The complete IMC circuit for the algorithm is designed in the 65 nm LSTP technology of STMicroelectronics by using Cadence Virtuoso and simulated for functionality verification and power analysis using the Eldo Circuit Simulator.

A. POWER SIDE CHANNEL LEAKAGE ASSESSMENT METHODOLOGY

The power consumption of a circuit is the sum of its dynamic and static power consumption. It is measured by probing the total current drawn from the power supply directly during the circuit simulations. The power-consumption data for the complete memory array is extracted using the Eldo Simulator.

Power-based SCA exploits the vulnerability that different inputs to a cryptographic system will produce different power consumption patterns. The Test Vector Leakage Assessment (TVLA) methodology, demonstrated in [22], [25], is used to verify whether this vulnerability exists in the IMC implementation also or not by analyzing side-channel leakage statistics. So, if different inputs to the system always result in a similar power consumption pattern on average, there is nothing for an attacker to exploit. Here, specific test vectors are used, and corresponding power consumption patterns are sampled. Further, statistical tools like *t-tests* are used to analyze the vulnerability of the device based on the sampled power consumption patterns.

1) TEST VECTOR GENERATION

We have generated 200 random input test vectors and categorized them into two sets. Each generated vector has a bit-width of 512 bits, i.e., a total of 16 32-bits words. The structure of each vector looks like $\{c_0, c_1, c_2, c_3, k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, bc_0, n_0, n_1\}$, where each element is a 32-bit word, k_0–k_7 represents the 256-bit key, where k_i^{th} represent the 32-bit ith sub-key, (bc_0) represent the block counter, (n_0, n_1) represent the nonce and (c_0, c_1) represent constants.

2) WELCH'S T-TEST

In this paper, Welch's t-test is the statistical tool that is used for comparing the two sets of power consumption patterns [22]. It is used to compare the mean values of two groups of samples when the variances are different. Welch's t-statistic score is calculated as follow:

$$t = \frac{\mu_A - \mu_B}{\sqrt{\frac{\sigma_A^2}{\eta_A} + \frac{\sigma_B^2}{\eta_B}}} \tag{2}$$

where μ_A and μ_B are the means, σ_A and σ_B are the standard deviations, and η_A and η_B are the sample sizes of the two groups A and B, respectively. Higher values of the t-statistic score indicate a significant difference between the two sample sets, whereas a smaller t-statistic score suggests that the two sample sets are very similar.

The TVLA framework specifies a threshold t-statistic value of |4.5| for deciding that the implementation is leaking information at any particular point in the power trace [22]. The t-statistic value of the design less than |4.5| represents that the power consumption patterns of the two different data sets are nearly identical even if the input differs in terms of key [22]. So, the circuit is classified as resistant to power-based SCA. While if the score exceeds the threshold of |4.5|, the power consumption pattern differs significantly for the two sets of input vectors. Hence, an adversary can extract information regarding the key via the power consumption patterns.

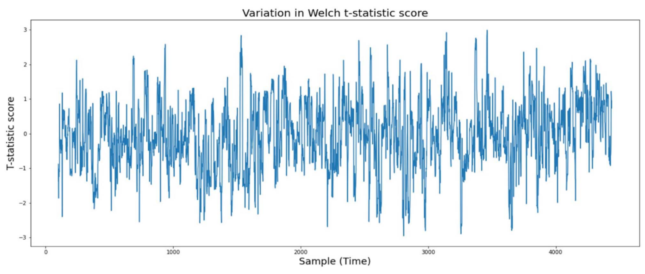


FIGURE 17. Welch's t-statistic score (5GHz Sampling).

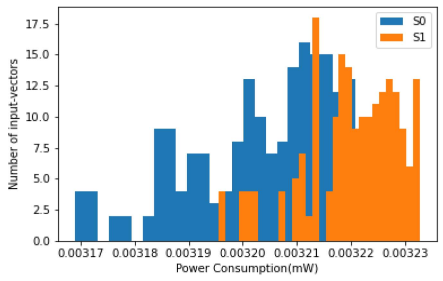


FIGURE 18. Histogram: Power consumption at a time-instant.

3) SIDE CHANNEL LEAKAGE ASSESSMENT METHODOLOGY

For each input vector, the circuit is simulated, and the power consumption is recorded throughout the execution of the algorithm in the hardware. The hence generated power traces are sampled at the rate of 5.4 GHz. To carry out the Welch's t-test, the power-traces are categorized into two groups corresponding to their input test vector set. We receive two sets of power consumption values belonging to two different sets of test vectors at each sampling point. These values are then used to create a probability density function (PDF) for the Welch's test. The statistical difference between the distributions of these two sets directly indicates the amount of side-channel leakage at that particular instant. This difference is quantified with the help of Welch's t-statistic scores. Here, the Welch's score is evaluated at each of the sampling points when the and presented in Fig. 17. The given example in Fig. 18 shows the captured histogram at one such specific time instant for power consumption corresponding to input vectors of the two sets.

B. PERFORMANCE ANALYSIS

The performance of the proposed hardware implementation is compared with a software implementation of the algorithm in terms of execution time and the number of clock cycles. We took a standard software implementation of the ChaCha20 algorithm from GitHub's repository of the OpenSSL cryptography toolkit library. The code is executed

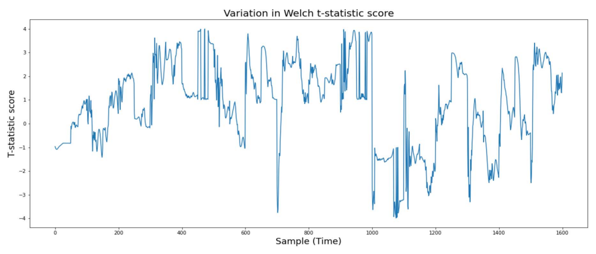


FIGURE 19. Welch's t-test scores at 25°C for single round (50Ghz power sampling).

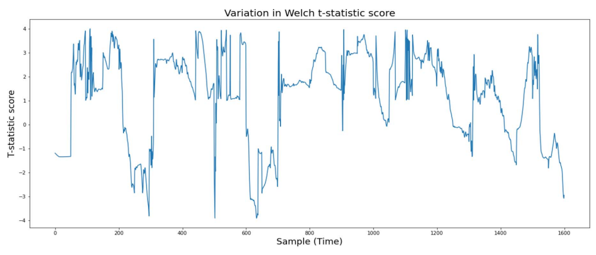


FIGURE 20. Welch's t-test scores at 125°C for single round (50Ghz power sampling).

on the Cortex A9 processor, and the number of clock cycles and execution time are sampled.

VII. RESULTS

We begin with simulating the proposed circuit to generate the power traces for multiple input vectors. These traces are then analyzed by the tools mentioned above to check for any side-channel leakage. Then, for performance analysis, the circuit is implemented on the ARM Cortex A9 processor, and its latency is noted and compared with that of the proposed implementation.

A. POWER BASED SIDE-CHANNEL ATTACK ASSESSMENT

Each of the captured power traces for complete execution was sampled at a frequency of around 50Ghz (1600 samples in 32ns). The circuit was simulated for a single ChaCha20 round at 25°C and 125°C temperature values along with Monte-Carlo analysis to model the effects of fabrication and temperature variations. The Welch's t-test scores for these cases are represented in Fig. 19 and Fig. 20 respectively for different sampling frequencies. It was observed that the t-statistic score for comparison between the provided two sets of input data in all the above experiments was well within the bounded value of |4.5|.

B. PERFORMANCE ASSESSMENT

We used Zedboard (Xilinx Zynq-7000 APSoC) as the target platform for the software implementation, which includes the required processor core. The SDSoC software is used to generate the boot image file from the C++ specification of the algorithm. It contains the instruction set (assembly generated from the C++ program) required to execute the algorithm on the processor. Finally, Tera Term, a terminal

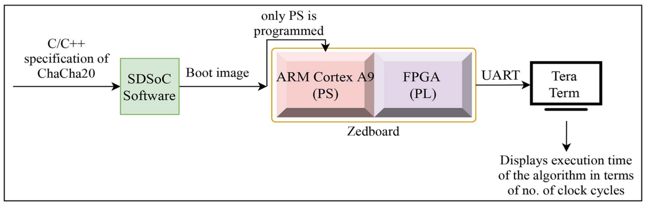

C/C++ specification of ChaCha20
SDSoC Software
Boot image
only PS is programmed
ARM Cortex A9 (PS)
FPGA (PL)
Zedboard
UART
Tera Term
Displays execution time of the algorithm in terms of no. of clock cycles

FIGURE 21. Steps to implement the ChaCha20 algorithm on the ARM Processor (Zedboard).

TABLE 2. Performance comparison of ARM cortex A9 and proposed IMC implementation.

Platform	Clock Frequency (in MHz)	Latency (in clock cycles)	Average Execution Time (in μs)
ARM Cortex A9	666	4094	6
Proposed IMC implementation	250	220	0.81

emulator, was used to display the results from the Zedboard. The process is shown in the Fig. 21. A speedup of approximately 7× in terms of the execution time was observed with the proposed in-memory implementation.

VIII. CONCLUSION

In this paper, we have presented a memory array-based implementation of the ChaCha20 algorithm to ensure its safety against power-based SCA. The proposed in-memory-ChaCha20 architecture is designed in ST Microelectronics 65nm LSTP technology and simulated. The Welch's t-test is employed to demonstrate the effectiveness of the circuit against power-based side-channel attacks. The results show that power consumption of the circuit is independent of the input vector or the data being processed, thereby warding off power-based SCA. Furthermore, the IMC technique offers a low area and low power solution for securing IoT edge devices since it bypasses the need for a separate co-processor.

Additionally, the proposed design overcomes the von-Neumann bottleneck as the memory array is now dually used as a storage and computing unit. We also show that by utilizing in-built parallelism during memory accesses and enabling high-bitwidth encryption, significant improvement in performance can be achieved. We demonstrate a 7X performance gain for the execution of the ChaCha20 algorithm on the proposed in-memory architecture as compared to the ARM Cortex A9 processor.

REFERENCES

[1] S. Rizvi, A. Kurtz, J. Pfeffer, and M. Rizvi, "Securing the Internet of Things (IoT): A security taxonomy for IoT," in *Proc. 17th IEEE Int. Conf. Trust Security Privacy Comput. Commun. 12th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, New York, NY, USA, Aug. 2018, pp. 163–168.
[2] A. Safi, "Improving the security of Internet of Things using encryption algorithms," *Int. J. Comput. Inf. Eng.*, vol. 11, no. 5, pp. 558–561, Apr. 2017.
[3] "From the Internet of Things to the Internet of Trust." NXP. Jan. 2019. [Online]. Available: https://www.nxp.com/docs/en/white-paper/NXP-FROM-IOT-TO-IOTRUST-WP.pdf
[4] Y. Zhou and D. Feng, "Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security Testing," IACR Cryptol. ePrint Arch., Lyon, France, Rep. 2005/388, Sep. 2005.

[5] M. S. E. Quadir and J. A. Chandy, “Embedded systems authentication and encryption using strong PUF modeling,” in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Las Vegas, NV, USA, 2020, pp. 1–6, doi: 10.1109/ICCE46568.2020.9043104.

[6] M. Bhargava and K. Mai, “An efficient reliable PUF-based cryptographic key generator in 65nm CMOS,” in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, 2014, pp. 1–6, doi: 10.7873/DATE.2014.083.

[7] O. Lo, W. J. Buchanan, and D. Carson, “Power analysis attacks on the AES-128 S-box using differential power analysis (DPA) and correlation power analysis (CPA),” *J. Cyber Security Technol.*, vol. 1, no. 2 pp. 88–107, Apr. 2017, doi: 10.1080/23742917.2016.1231523.

[8] J. Bonneau and I. Mironov, “Cache-collision timing attacks against AES.” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Berlin, Germany: Springer, Oct. 2006, pp. 201–215.

[9] A. Akram, M. Mushtaq, M. K. Bhatti, V. Lapotre, and G. Gogniat, “Meet the Sherlock Holmes’ of side channel leakage: A survey of cache SCA detection techniques,” *IEEE Access*, vol. 8, pp. 70836–70860, 2020, doi: 10.1109/ACCESS.2020.2980522.

[10] M. Randolph and W. Diehl, “Power side-channel attack analysis: A review of 20 years of study for the layman,” *Cryptography*, vol. 4, no. 2, p. 15, Jun. 2020.

[11] T. Popp, S. Mangard, and E. Oswald, “Power analysis attacks and countermeasures,” *IEEE Design Test Comput.*, vol. 24, no. 6, pp. 535–543, Nov./Dec. 2007, doi: 10.1109/MDT.2007.200.

[12] Z. Najm, D. Jap, B. Jungk, S. Picek, and S. Bhasin, “On comparing side-channel properties of AES and ChaCha20 on microcontrollers,” in *Proc. IEEE Asia–Pac. Conf. Circuits Syst. (APCCAS)*, Chengdu, China, 2018, pp. 552–555, doi: 10.1109/APCCAS.2018.8605653.

[13] 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.

[14] M. Hutter, M. Feldhofer, and J. Wolkerstorfer, “A cryptographic processor for low-resource devices: Canning ECDSA and AES like sardines,” in *IFIP International Workshop on Information Security Theory and Practices*. Berlin, Germany: Springer-Verlag, 2011, pp. 144–159.

[15] 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.

[16] Y. Zhang, L. Xu, Q. Dong, J. Wang, D. Blaauw, and D. Sylvester, “Recryptor: A reconfigurable cryptographic cortex-M0 processor with in-memory and near-memory computing for IoT security,” *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 995–1005, Apr. 2018, doi: 10.1109/JSSC.2017.2776302.

[17] M. Xie, S. Li, A. O. Glova, J. Hu, Y. Wang, and Y. Xie, “AIM: Fast and energy-efficient AES in-memory implementation for emerging non-volatile main memory,” in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, 2018, pp. 625–628, doi: 10.23919/DATE.2018.8342085.

[18] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, “X-SRAM: Enabling in-memory Boolean computations in CMOS static random access memories,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 12, pp. 4219–4232, Dec. 2018, doi: 10.1109/TCSI.2018.2848999.

[19] I. Beavers and E. MacLean, *Intelligence at the Edge Part 4: Edge Node Security*, Analog Devices, Inc., Norwood, MA, USA. Accessed: Jan. 1, 2021. [Online]. Available: https://www.analog.com/en/technical-articles/intelligence-at-the-edge-part-4-edge-node-security.html

[20] L. E. Kane, J. J. Chen, R. Thomas, V. Liu, and M. McKague, “Security and performance in IoT: A balancing act,” *IEEE Access*, vol. 8, pp. 121969–121986, 2020.

[21] D. J. Bernstein, “ChaCha, a variant of salsa20,” in *Proc. Workshop Rec. SASC* vol. 8, Jan. 2008, pp. 3–5.

[22] B. J. G. Goodwill, J. Jaffe, and P. Rohatgi, “A testing methodology for side-channel resistance validation,” in *Proc. NIST Non-Invasive Attack Test. Workshop*, vol. 7, Sep. 2011, pp. 115–136.

[23] *SDSoC Environment User Guide*, document UG1027 (v2019.1), Xilinx, San Jose, CA, USA, May 2019. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug1027-sdsoc-user-guide.pdf

[24] K. C. Akyel *et al.*, “DRC^2: Dynamically reconfigurable computing circuit based on memory architecture,” in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, Oct. 2016, pp. 1–8, doi: 10.1109/ICRC.2016.7738698.

[25] M. He, J. Park, A. Nahiyan, A. Vassilev, Y. Jin, and M. Tehranipoor, “RTL-PSC: Automated power side-channel leakage assessment at register-transfer level,” in *Proc. IEEE 37th VLSI Test Symp. (VTS)*, 2019, pp. 1–6, doi: 10.1109/VTS.2019.8758600.

[26] P. Slpsk, P. K. Vairam, C. Rebeiro, and V. Kamakoti, “Karna: A gate-sizing based security aware EDA flow for improved power side-channel attack protection,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8, doi: 10.1109/ICCAD45719.2019.8942173.

[27] I. Beavers, *Intelligence at the Edge Part 1: The Edge Node*, Analog Devices, Inc., Norwood, MA, USA. Accessed: Jun. 1, 2021. [Online] Available: https://www.analog.com/en/technical-articles/intelligence-at-the-edge-part-1-the-edge-node.html

[28] P. Sethi and S. R. Sarangi, “Internet of Things: Architectures, protocols, and applications,” *J. Elect. Comput. Eng.*, vol. 2017, pp. 1–25, Jan. 2017, doi: 10.1155/2017/9324035.

[29] J. Pfau, M. Reuter, T. Harbaum, K. Hofmann, and J. Becker, “A hardware perspective on the ChaCha ciphers: Scalable Chacha8/12/20 implementations ranging from 476 slices to bitrates of 175 Gbit/s,” in *Proc. 32nd IEEE Int. Syst. Chip Conf. (SOCC)*, Singapore, 2019, pp. 294–299, doi: 10.1109/SOCC46988.2019.1570548289.

M. AAMIR is currently pursuing the undergraduate degree in electronics and communication engineering with the Indraprastha Institute of Information Technology Delhi, New Delhi, India. His current research interests include memory design and in-memory compute for security applications.

SOMYA SHARMA is currently pursuing the undergraduate degree in electronics and communication engineering with the Indraprastha Institute of Information Technology Delhi, New Delhi, India. Her current research interests include in-memory designing, and implementing efficient and reconfigurable architectures for signal processing algorithms on FPGA.

ANUJ GROVER received the B.Tech. degree in electrical engineering from IIT Delhi, the M.S. degree in electronic circuits and systems from the University of California at San Diego, San Diego, USA, and the Ph.D. degree in electrical engineering from IIT Delhi.

He currently teaches with IIIT Delhi as an Associate Professor. Before joining IIIT Delhi, he worked with STMicroelectronics for over 18 years and has led large teams on multimillion dollar projects. His research interests include memory design, in-memory compute, digital circuits and system design, safety and security in hardware, and inventive problem-solving methods.