Journal of Computing and Security

Research Article, 2027, Volume 14, Number 1 (pp. 1–28)

http://www.jcomsec.org

Design of Memristive Lightweight Encryption For In-Memory Image Steganography

Seyed Erfan Fatemieh [a,*] Reza Shahdi Alizadeh [b] Esmail Zarezadeh [c]

[a] *MICE Research Group, Department of Computer Architecture, Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran.*

[b] *Department of Computer Engineering, Yadegar-e-Imam Khomeini (RAH) Shahre Rey Branch, Islamic Azad University, Tehran, Iran.*

[c] *Department of Electrical Engineering, Khatam al-Anbia University, Tehran, Iran.*

ARTICLE INFO.

Article history:
Received: 20 July 2026
Revised: 17 September 2026
Accepted: 10 October 2026
Published Online: 05 November 2026

Keywords:
Computation In-Memory (CIM), Memristor, Lightweight Cryptography, IMPLY Logic, Steganography.

ABSTRACT

With the expansion of data-intensive applications and increasing data volumes, providing an efficient solution to address growing energy consumption and performance degradation caused by the transfer of large amounts of data between the processor and the main memory has become a severe challenge. The frequent transfer of large amounts of data between internal chip units, memories, and their interconnections exacerbates the vulnerability of the data being accessed. Employing a memristive Computation In-Memory-Array (CIM-A) architecture limits data transfer, thereby addressing both challenges. By integrating lightweight cryptography, developed to be applied in hardware-constrained devices, with CIM-A architectures, the significant energy and latency overheads associated with a high rate of data transfers across the architecture's main blocks and interconnections can be mitigated. This paper implements two standard lightweight stream ciphers, Trivium and Grain-128a, for CIM using stateful material implication (IMPLY) logic to address the performance challenges. In addition to redesigning the cryptographic structures, we reduce the hardware complexity of conventional IMPLY-based implementations by proposing an efficient method for shifting data within the shift registers. Applying the proposed data-shifting method to the registers of these ciphers reduces the number of computational steps and decreases memristors' energy consumption by up to 42% and 44%, respectively, compared to conventional implementations. Finally, the performance of the proposed circuits is evaluated in a steganography application, demonstrating their practical efficiency.

1 Introduction

Today, data is one of the most important components of computation, and its fast, energy-efficient process-

* Corresponding author.
Email addresses: erfanfatemieh@eng.ui.ac.ir (S. E. Fatemieh), r.sh.alizadeh@gmail.com (R. Shahdi Alizadeh), zarezadeh@aut.ac.ir (E. Zarezadeh)

https://dx.doi.org/10.22108/jcs.2026.149790.1214

ISSN: 2322-4460

ing has become a challenge for computer architects more than ever. The Internet of Things (IoT) and Artificial Intelligence (AI) are among the most important processing applications, expanding in various dimensions every day. From the perspective of an architect of a high-volume data-processing system, the secure transmission of data, alongside fast and energy-efficient processing, is of particular importance. Maintaining data security and integrity poses numerous challenges, and using appropriate cryptographic mechanisms for secure data transmission, especially on edge devices and in IoT-related processing applications, is of high importance. The constant data movement between the processor and memory in the conventional von Neumann architecture is a major concern for designers and researchers, both in terms of resource consumption and circuit evaluation criteria, and in terms of the security of data accessed during inter-unit transfers, especially in interconnections [1, 2].

By the middle of the first decade of the new century, technological improvements and reductions in chip transistor dimensions had significantly improved performance and addressed various processing needs [3]. With the invalidation of Dennard's scaling principle, the mismatch between Gordon Moore's prediction and manufacturing processes, and the power wall problem, the continuous trend of improving processor efficiency faced a sharp decline due to numerous problems, such as increased leakage currents, which were overcome by applying alternative architectures and the use of multi-core processors for several years [3, 4]. One of the main solutions proposed by researchers to overcome the aforementioned problems has been the use of emerging technologies such as Carbon Nanotube Field-Effect Transistors (CNFETs), Quantum-dot Cellular Automata (QCA), and memristors [4–7]. In addition to emerging technologies, numerous computational methods have been proposed across various scientific communities to replace conventional methods. One of the most promising of these methods is Computation In-Memory (CIM). In the conventional von Neumann architecture, the main memory in the memory hierarchy is located far from the central processor. The significant difference between processor speed and memory bandwidth, combined with the massive volume of data transferred for processing and computation in this architecture, has become a major bottleneck in the design, known as the von Neumann (memory) bottleneck [8]. The problem of data transfer in modern processing applications is so serious that, according to estimates from companies such as Google, from 63% to 90% of total processing energy consumption is spent solely on data movement, without performing any actual computation [9]. The use of CIM architectures that leverage emerging memristive technologies, which enable simultaneous data storage and logic/arithmetic operations, is a promising approach to overcoming the problems posed by the memory wall. Several methods have been proposed for designing arithmetic and logical circuits using memristors, due to their favorable characteristics. In general, these methods can be classified into two categories: stateful and non-stateful [10]. Stateful methods are methods that can be used to store and process data completely within the memristive array [2, 4, 10]. This means that stateful methods such as material implication (IMPLY) [11] and Memristor Aided loGIC (MAGIC) [12] are memristive processing methods that can be used in the CIM-Array (CIM-A) architecture to significantly reduce data transfer volume [13].

Maintaining information security and privacy across various processing applications related to IoT and AI, where large volumes of data are generated, collected, and transmitted by edge devices, has become a major concern [1]. The use of cryptographic mechanisms in these areas is essential. Memristors can be an efficient option for implementing cryptographic platforms due to their small area and power consumption, and their potential integration into CIM-A architectures to eliminate redundant data transfers [1, 2, 13, 14]. Excessive data transfer between the processor and main memory in conventional cryptographic units, which is required to generate and store ciphertext, also increases the risk of attack; this vulnerability can be minimized by applying a CIM-A architecture [1]. In [15–17], several methods for generating random and pseudorandom numbers using memristors have been investigated. Implementation and acceleration of hash functions using memristors for processing within and near the memory array have also been discussed in [18]. In addition to generating random numbers and hash functions, designers have also considered methods for implementing cryptographic mechanisms that specifically utilize memristors. In [1], the characteristics of the process variations during transistor manufacturing in a single chip are applied to generate random numbers. The non-stateful method of One Transistor-One Resistor (1T-1R) is also employed to XOR the input value stored in the memristors with the aforementioned random number to generate the ciphertext [1]. In 2024, the lightweight block cipher GIFT, based on memristors, was used to generate ciphertext on edge devices [14]. In this study, the XOR structure is located in the peripheral circuits of the memristive crossbar array (sense amplifiers) [14]. Hence, the architecture considered by the authors was categorized as a CIM-Periphery (CIM-P) architecture [13].

Based on the results presented in [14, 19–22] and an analysis of various lightweight ciphers, it can be concluded that lightweight stream ciphers are more



suitable for implementation in devices with limited hardware resources. Using stateful methods in the design of memristor-based circuits significantly reduces data movement during inter-unit transfers, providing an efficient way to improve circuit metrics. Considering the limitations imposed by power and memory walls, the importance of applying cryptographic mechanisms in inter-unit communications, and the need to minimize data transfer between processing units and memory on a chip, the authors propose the design of IMPLY-based encryption/decryption units for integration into CIM-A architectures, ensuring high compatibility with conventional memristive crossbar arrays. The lightweight stream ciphers evaluated in this article were selected based on global evaluations and standards [20, 21, 23, 24]. The selected ciphers have been redesigned and improved based on a reliable stateful method, IMPLY [25], to align seamlessly with the structure of conventional memristive crossbar arrays. Furthermore, the functionality of the proposed designs has been evaluated behaviorally in a steganography application. The results confirmed the practical applicability of the proposed designs in this domain. The main contributions of this paper are as follows:

(1) Presenting a systematic methodology for porting the classic implementation algorithms of the Trivium [23, 26] and Grain-128a [27] lightweight stream ciphers to the structure of CIM-A architectures based on the stateful IMPLY design method;
(2) Developing an algorithm for reducing the computational steps and memristors' energy consumption of Linear Feedback Shift Registers (LFSRs) and Nonlinear Feedback Shift Registers (NFSRs) used in the Trivium and Grain-128a lightweight stream ciphers;
(3) Redesigning and evaluating the functional correctness of all the basic logic blocks required in the IMPLY-based improved implementation algorithms of the Trivium and Grain-128a lightweight stream ciphers; and
(4) Examining the functionality and image quality metrics of the proposed memristive encryption and decryption units in a steganography application.

The remainder of this paper is organized into four sections. Section 2 reviews the basic concepts, definitions, and related work. Section 3 presents the proposed algorithm for implementing LFSRs and NFSRs, along with a step-by-step method for redesigning the Trivium and Grain-128a lightweight stream ciphers based on the IMPLY design method for CIM. The circuit-level and application-level simulation results are reported in Section 4. Finally, the paper's conclusion is provided in Section 5.

2 Background and Related Works

2.1 Lightweight Stream Ciphers

Symmetric ciphers are divided into two categories: stream ciphers and block ciphers [23]. In stream ciphers, the message is encrypted by adding it (modulo-2 addition) to a pseudorandom keystream by applying XOR operations. Stream ciphers are often faster than block ciphers and require fewer hardware resources, making them suitable for resource-constrained devices [23, 28]. In general, block ciphers are more susceptible to error propagation when noise corrupts data, and stream ciphers are recommended when the message size is unknown or the messages are transmitted in continuous streams [28]. Stream ciphers were developed based on the One-Time Pad (OTP) cipher, but because of the OTP's limitations, they are used as an alternative [19]. Pseudorandom Number Generators (PRNGs) are a main pillar of stream ciphers, which use various structures, including LFSRs. The LFSR structure, in combination with nonlinear logic gates, acquires nonlinear characteristics and is applied in stream ciphers [23].

Over the last three decades, numerous symmetric ciphers have been proposed by various companies and research institutions and evaluated across aspects such as information security, complexity, and the resources required for evaluation [19–24, 27, 28]. Among the most important of these processes are the eSTREAM project and the evaluations of the US National Institute of Standards and Technology (NIST) [19–24, 27, 28]. Several stream ciphers have also been examined in these processes. Cryptography is widely applied in processing environments such as the IoT, but many devices in these and similar domains are embedded devices with limited resources [20, 22, 28]. Therefore, relying on conventional cryptographic algorithms with high computational complexity in these applications is associated with significant limitations. Lightweight cryptography is a different approach from conventional algorithms that enables the maintenance of confidentiality and data integrity on devices with limited resources. Most efficient lightweight ciphers are symmetric ciphers, and in addition to evaluating their security, assessing their circuit metrics, such as energy consumption and resource utilization, is of great importance. A complete introduction and review of the main features of lightweight stream and block ciphers, along with comprehensive comparisons for selecting the appropriate lightweight cipher according to different design constraints, have been reported in [19–23, 28]. Based on the evaluations conducted and the results reported in the eSTREAM project and NIST evaluation processes, and considering the structures and characteristics of various lightweight stream

ciphers, Trivium [26] and Grain-128a [27] are selected to be described in detail in the following subsections.

2.1.1 Trivium

Trivium is one of the selected lightweight stream ciphers of the eSTREAM project and is also a standardized cipher (ISO/IEC 29192-3:2012) [20, 21]. As shown in Figure 1, the structure of this cipher consists of three shift registers, a 93-bit register (A), an 84-bit register (B), and a 111-bit register (C), combined with nonlinear logic gates [23]. By calculating the modulo-2 addition of these three shift registers' outputs, the keystream (s_i) is produced [23]. The logical equations defining the structure of this lightweight stream cipher are expressed in (1)-(4). The 80-bit key and Initialization Vector (IV) are placed in the 80 Least Significant Bits (LSBs) of registers A and B, respectively, and all other bits of the registers are set to logical '0', except for bits 109-111 of register C, which are set to a logical '1' [23]. In the initialization phase, the cipher's structure is clocked 1152 times, and the keystream output from the 1153^{rd} cycle onward is used to produce the ciphertext [23].

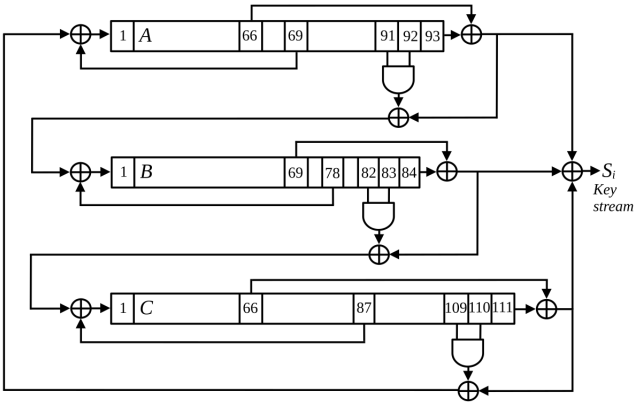


Figure 1. Architecture of the stream cipher Trivium [23, 26].

$$a_i \equiv c_{i-66} + c_{i-111} + c_{i-110} \cdot c_{i-109} + a_{i-69} \quad (1)$$

$$b_i \equiv a_{i-66} + a_{i-93} + a_{i-92} \cdot a_{i-91} + b_{i-78} \quad (2)$$

$$c_i \equiv b_{i-69} + b_{i-84} + b_{i-83} \cdot b_{i-82} + c_{i-87} \quad (3)$$

$$s_i \equiv a_{i-66} + a_{i-93} + b_{i-69} + b_{i-84} + c_{i-66} + c_{i-111} \quad (4)$$

2.1.2 Grain-128a

Another selected cipher of the eSTREAM project, which has been standardized in ISO/IEC 29167-13:2015 as one of the applicable ciphers for Radio Frequency Identification (RFID) systems, is Grain-128a [24]. This cipher can perform authentication in addition to producing a keystream for data encryption/decryption. Here, only the Grain-128a output generation process is discussed. The structure of this cipher is illustrated in Figure 2 for the pre-initialization and output generation stages. The main components in the structure of this cipher are the shift registers, each of which is 128 bits. (5) and (6) describe the update functions of these two shift registers, respectively [27]. The Boolean functions h(x) and y (the output generation function) are also computed according to (7) and (8), respectively. The 128-bit key and the 96-bit IV are placed in the NFSR and LFSR, respectively [27]. The other 32 bits of the LFSR are initialized from bits 96 to 126 with logical '1', and the last bit (the 127^{th} bit) with logical '0' [27]. The pre-initialization phase occupies 256 cycles, and from the 257^{th} cycle onward, keystream generation begins. In each cycle after the 256^{th} one, the modulo-2 addition (XOR) of the keystream and the input data produces the ciphertext [27].

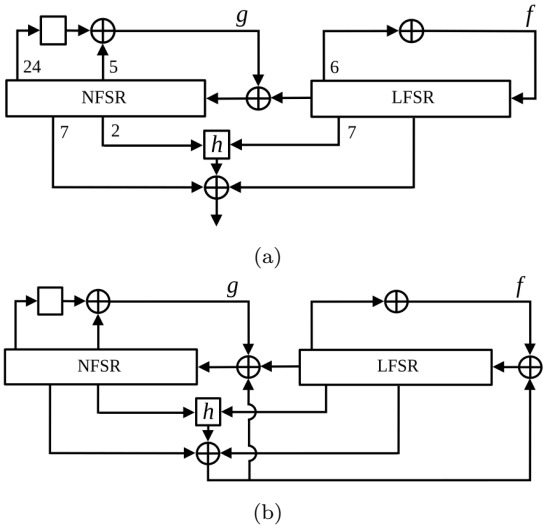


Figure 2. Architecture of Grain-128a: (a) the pre-output generator, and (b) the key initiaization [27].

$$s_{i+128} = s_i + s_{i+7} + s_{i+38} + s_{i+70} + s_{i+81} + s_{i+96} \quad (5)$$

$$b_{i+128} = s_i + b_i + b_{i+26} + b_{i+56} + b_{i+91} + b_{i+96} + b_{i+3} \cdot b_{i+67} + b_{i+11} \cdot b_{i+13} + b_{i+17} \cdot b_{i+18} + b_{i+27} \cdot b_{i+59} + b_{i+40} \cdot b_{i+48} + b_{i+61} \cdot b_{i+65} + b_{i+68} \cdot b_{i+84} + b_{i+22} \cdot b_{i+24} \cdot b_{i+25} + b_{i+70} \cdot b_{i+78} + b_{i+82} \cdot b_{i+88} + b_{i+92} \cdot b_{i+93} \cdot b_{i+95} \quad (6)$$

$$h(x) = b_{i+12} \cdot s_{i+8} + s_{i+13} \cdot s_{i+20} + b_{i+95} \cdot s_{i+42} + s_{i+60} \cdot s_{i+79} + b_{i+12} \cdot b_{i+95} \cdot s_{i+94} \quad (7)$$

$$y_i = h(x) + s_{i+93} + \sum_{j \in \mathcal{A}} b_{i+j}, \quad \mathcal{A} \in \{2, 15, 36, 45, 64, 73, 89\} \quad (8)$$

2.2 An Introduction to Memristor

Capacitors, inductors, and resistors are three basic electrical elements that define the relationship between electric charge and voltage, magnetic flux and current, and voltage and current, respectively. Professor L. Chua of the University of California at Berkeley introduced the memristor in the 1970s as the fourth basic electrical element, defining the relationship between electric charge and magnetic flux [10]. Four decades later, a team at HP Laboratories led by Professor S. Williams implemented the first physical memristor [10]. The memristor, whose symbol is illustrated in Figure 3(a), can act as a non-volatile memory cell, and its resistance determines its logic state. If the resistance of the memristor is at its minimum value (R_{ON}), it is generally equivalent to Boolean logic '1', and if the resistance is at its maximum value (R_{OFF}), it represents Boolean logic '0' [4, 10]. As shown in Figure 3(a), if a voltage/current is applied from the Top Electrode (TE) of the memristor, its resistance decreases, and if applied from the other terminal, its resistance increases [10]. The resistance of the memristor remains unchanged for years when no voltage/current is applied to its terminals, making it an emerging non-volatile memory cell. Memristive CIM has another fundamental aspect: the execution of logic and arithmetic operations. Researchers have proposed several methods for implementing logic and arithmetic functions using memristors, which are generally classified into two categories: stateful and non-stateful [10]. In stateful circuit design methods, the resistance of the memristor determines the logic state at the input, output, and intermediate nodes, whereas in non-stateful methods, the logic state is mainly determined by voltage [10]. The applicability of stateful design methods in memristive CIM-A has been widely evaluated, and several methods have been developed. Among the most important stateful methods are IMPLY [11], MAGIC [12], and fast and energy-efficient logic in memory (FELIX) [29]. Each of these methods has several advantages and disadvantages, and, considering the comparative summary in [10, 25, 29] and the reliability factor, the IMPLY method has been selected as the stateful method applied in this paper.

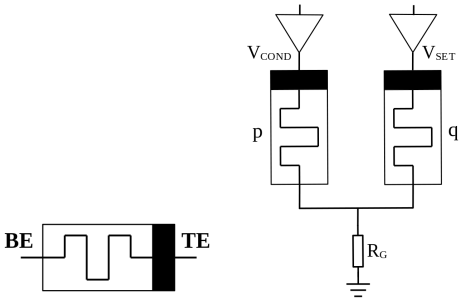


Figure 3. (a) Memristor Symbol, and (b) Memristive IMPLY Gate [4].

The IMPLY gate architecture and its truth table are illustrated in Figure 3(b) and Table 1, respectively. The IMPLY gate is universal; this means that any logic function can be computed by applying it multiple times in addition to the FALSE (zero) function. For example, the OR logic function can be implemented using two input memristors and one work memristor in two IMPLY-based computational steps, in addition to executing the FALSE function once, as shown in (9) [6]. Work memristors are used alongside input memristors to implement logic and arithmetic functions in the IMPLY method. Storing temporary variables during the execution of IMPLY-based logic and arithmetic functions is the main role of work memristors, although these memristors can also be used to store the outputs. To perform the IMPLY operation, two voltage, V_{COND} and V_{SET}, are simultaneously applied to memrsitors 'p' and 'q' in Figure 3(b), and the result of "$pIMPq$" is stored in the memristor 'q' [6]. For the correct execution of the IMPLY operation, two conditions, 1) $V_{COND} < V_c < V_{SET}$ (V_c: the threshold voltage of a memristor) and 2) $R_{ON} << R_G << R_{OFF}$, must be met [4, 6]. Four architectures, serial, parallel, semi-serial, and semi-parallel, have been introduced for the development of logic and arithmetic circuits using the IMPLY method [4, 6]. The serial architecture is the most compatible with the structure of memristive crossbar arrays; hence, it is an acceptable structure for use in CIM-A [4, 6]. In the serial architecture, the memristors are placed in a row or column of the crossbar array. This architecture is illustrated in Figure 4. The serial architecture has the least hardware complexity among the others, but it can perform only one IMPLY operation per computational step [4]. Numerous logic and arithmetic circuits, such as different types of adders and multipliers, have been

implemented in the serial architecture using the IMPLY design method [2, 4, 6, 11, 30–32]. Table 2 lists the implementation algorithms of serial IMPLY-based basic logic gates and their characteristics.

$$p\ OR\ q \equiv (p\ IMP\ 0)\ IMP\ q \equiv (p \rightarrow 0) \rightarrow q \quad (9)$$

Table 1. The Truth Table of a Universal IMPLY Logic Gate [4].

p	q	$p\ IMPLY\ q \equiv p\ IMP\ q \equiv p \rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

Table 2. Implementation of basic Boolean logic gates using IMPLY method's primitive functions (IMPLY and FALSE) [6].

Boolean logic gate	Equivalent IMPLY logic
NOT p	$p \rightarrow 0$
p OR q	$(p \rightarrow 0) \rightarrow q$
p NOR q	$((p \rightarrow 0) \rightarrow q) \rightarrow 0$
p NAND q	$p \rightarrow (q \rightarrow 0)$
p AND q	$(p \rightarrow (q \rightarrow 0)) \rightarrow 0$

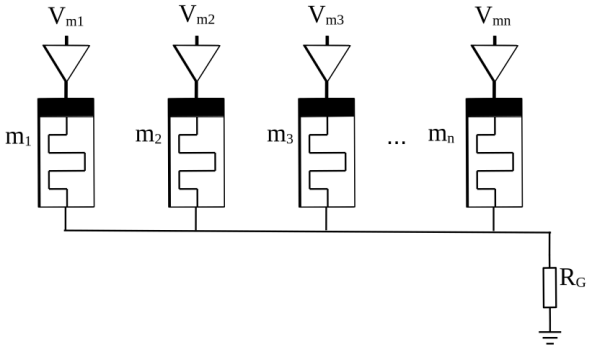


Figure 4. The serial architecture of IMPLY-based logical and arithmetic circuits [2, 4, 6].

2.3 Memristive Cryptography

Memristors can be applied in cryptography due to their electrical properties. This subsection briefly discusses a few of these applications.

In [1], an architecture is introduced that generates random numbers based on variations in the threshold slope of Metal Oxide Semiconductor FETs (MOSFETs) in a 1T-1R crossbar array, driven by manufacturing process variations. According to the architecture designed in [1], the threshold slope value is computed by dedicated extraction circuits and stored in a register. Then, this data is read and converted into a specific voltage [1]. By applying a non-stateful method introduced in [1], an XOR operation is performed between the data stored in the memristor and the aforementioned random voltage value, and the output ciphertext is generated.

Another application of memristors, widely used in cryptography, is the generation of random numbers. In [16, 17], different Ring Oscillators (ROs) are designed using varying numbers of transistors and memristors to generate random numbers. In the RO structure introduced in [16], a memristor is placed between the inverters of each row of the RO implemented based on Complementary MOS (CMOS) technology. If the propagation delays of two different rows are not the same, the generated output is logic '1', and otherwise, the output is logic '0' [16]. The role of memristors in the RO structure proposed in [33] is to replace the PMOS transistors of the inverter's pull-up network. The use of memristors in the RO structure increases the randomness of this circuit compared to an RO designed entirely with transistors [33]. In [17], two rows of single-bit ROs have been used to generate random numbers, where each row consists of multiple CMOS inverters, each connected to a single memristor. The random number generator proposed in [17] can be implemented in serial and parallel architectures, so that, if throughput needs to be increased, more data can be processed in the post-processing stage using a parallel-to-serial converter [17]. In the post-processing stage, the Trivium stream cipher is used, and after 1152 cycles, a random output bit is generated in each cycle [17].

The chaotic behavior of the memristor in [15] is used to generate noise. The output of the first stage of the proposed structure in [15] is a floating-point number. In the second stage, first, the output of the previous stage is multiplied by a fixed number (e.g., x_i), then the result $y_i = x_i\ mod\ 64$ is calculated and converted to binary format [15]. In the next phase of the second stage, five XOR gates are applied to compute the number of ones in y_i. Then, the raw bits of the second stage are applied to the Trivium unit in the third stage, and random outputs are generated bit-by-bit [15]. In the final stage of the proposed structure in [15], the random outputs are evaluated using the standard NIST statistical tests.

Lightweight encryption based on the GIFT block cipher has been applied within the structure of a memristive crossbar array for healthcare applications in [14]. The main idea in [14] is to integrate the compact

units of the GIFT cipher into the structure of the crossbar array. The assumption in [14] is that the generation and exchange of the encryption key between the transmitter and the receiver are handled by methods implemented in previous works. The XOR gate is one of the basic units of the GIFT cipher, which is implemented in [14] using Scouting Logic and Dual Sense Amplifier crossbar logic (DSA). This enhances parallel computation and is compatible with other parts of the GIFT cipher [14]. These two approaches are among the methods integrated into the CIM-P architecture.

Mozafari et al. [34] have presented a new design of the Piccolo-80 block cipher using the Voltage-To-Memristance (VTM) memristor design method. The authors believe that the VTM method and features such as the memristor's non-volatile nature enable compact, single-cycle implementation of the basic computational blocks in an energy-efficient manner [34]. The S-box and MixColumn operations have also been implemented in [34] using the VTM method. Hardware implementations of the Piccolo-80 cipher in [34] were carried out on an FPGA platform using the VTM logic gates proposed by the authors.

The LFSR is one of the basic structures for generating random numbers. A 4-bit LFSR implementation algorithm using the IMPLY method within a serial architecture has been developed in [35]. In the cited paper, the authors designed an algorithm to implement a D-type flip-flop using the IMPLY method, which appears redundant because a memristor can inherently store data like a D flip-flop. However, in [35], an algorithm for implementing a 4-bit LFSR using four D flip-flops and an XOR gate is proposed, employing eight memristors. The IMPLY-based LFSR computes the results in 55 computational steps [35].

Table 3 presents a qualitative comparison of various CIM architectures recently used to design memristive cryptographic units or their components (such as shift registers). A direct quantitative comparison of the normalized circuit evaluation criteria for the methods in [14, 34, 35] is not possible because their computational paradigms differ fundamentally. The comparison in Table 3 includes factors such as the computational paradigm, data movement, the need for data converters, how intermediate computational variables are stored, and the role of peripheral circuits.

3 Implementation of IMPLY-based Trivium and Grain-128a Lightweight Stream Ciphers

In this section, we first examine how to redesign and implement the standard stream ciphers (Trivium and Grain-128a) by applying the serial IMPLY design method for the CIM-A architecture. Then, by applying an innovative method and redesigning the shift register structure, the number of computational steps for both selected stream ciphers is reduced compared to conventional designs, thereby directly reducing memristors' energy consumption.

3.1 IMPLY-based Logic Gates Required for the Implementation of Trivium and Grain-128a Lightweight Stream Ciphers

In Figures 1 and 2 and equations (1)-(8), the architecture and logic equations required for the implementation of Trivium and Grain-128a ciphers can be seen. In the Trivium's structure, there are nine two-input XOR gates, one three-input XOR gate, and three two-input AND gates. Forty-seven two-input XOR gates, 18 two-input AND gates, five three-input AND gates, and two four-input AND gates are the logic blocks that are employed in the Grain-128a cipher.

To implement logic gates using the IMPLY method, it should be noted that in some cases it is possible that the input data of the logic gate is overwritten and the original value is deleted; therefore, in some cases, it is necessary to design and use an algorithm that preserves the value of the input data in the input memristors in these gates. For example, the algorithm introduced in [6] for implementing a two-input XOR gate incur a loss of one of the inputs. Therefore, to implement a two-input XOR gate, two algorithms are used, as shown in Tables 4 and 5. The first algorithm computes the output in nine computational steps using two work memristors, and one of the input memristors loses its original value (see (10)). The second implementation algorithm obtains the output using three work memristors in eleven computational steps. In the second algorithm, the logic values of the input memristors are preserved.

$$p\ XOR\ q \equiv (p \rightarrow q) \rightarrow ((q \rightarrow p) \rightarrow 0) \qquad (10)$$

Two methods can be used to implement XOR gates with three inputs and beyond. If the designer's goal is to minimize the computational steps and input data preservation is not important, it is recommended to use the algorithm introduced in [6], where the XOR result of two inputs is first calculated in nine steps, then in each of the nine steps after that, the XOR result of the previous inputs is XORed with the next input and the final result is calculated. The second method, however, focuses on preserving the input data. In this method, the first two inputs are first XORed using three work memristors according to the algorithm presented in Table 5, and the result is placed in a work memristor. After that, the other input is XORed with the result of the previous step using the algorithm

Table 3. A qualitative comparison of various CIM architectures recently applied in the design of memristive cryptographic units or their components [14, 34, 35].

Metric/Feature	[14]	[35]	[34]
Computational Method	DSA and Scouting Logic (CIM-P)	IMPLY-Stateful (CIM-A)	VTM-Semi-Stateful
Data Movement	High	Low	Medium
Data Converters	Required	Not Required	Required
Bandwidth [13]	High-Maximum [13]	Maximum [13]	High-Maximum
Controller Design Effort [13]	Medium [13]	High [13]	Medium-High
Periphery Design Effort [13]	High [13]	Low [13]	Low-Medium
Periphery Overhead	High	Low	Medium
CMOS Periphery Size	High	Low	Medium
Role of Periphery	High	Low	Medium

Table 4. Implementation algorithm of a destructive IMPLY-based serial XOR logic gate introduced in [6]

Step	Operation	Equivalent Logic
1	$s_1 = 0$	FALSE(s_1)
2	$s_2 = 0$	FALSE(s_2)
3	$a \rightarrow s_1 = s_1^{'}$	$NOT(a)$
4	$b \rightarrow s_2 = s_2^{'}$	$NOT(b)$
5	$s_1^{'} \rightarrow s_2^{'} = s_2^{''}$	$NOT(a) \rightarrow NOT(b)$
6	$s_1 = 0$	FALSE(s_1)
7	$s_2^{''} \rightarrow s_1 = s_1^{'}$	$NOT(NOT(a) \rightarrow NOT(b))$
8	$a \rightarrow b = b^{'}$	$a \rightarrow b$
9	$b^{'} \rightarrow s_1^{'} = s_1^{''}$	$XOR(a,\ b)$

written in Table 4.

In addition to XOR logic gates, two-input, three-input, and four-input AND gates are required to design the two proposed lightweight ciphers. The two-input AND gate can be implemented in a serial architecture using the IMPLY method, with two work memristors and in five computational steps, as shown in Table 2. For the three-input AND gate, the third input can be ANDed with the result of the AND of the first two inputs, and the result can be stored in one of the work memristors in the final step (the 10^{th} computational step). By examining (11) and Table 6, it can be concluded that by overlapping the computational steps of two two-input AND gates, the number of computational steps of a three-input AND gate can be reduced to six by executing two consecutive AND algorithms. The four-input AND gate can also be implemented in

Table 5. Implementation algorithm of a non-destructive IMPLY-based serial XOR logic gate.

Step	Operation	Equivalent Logic
1	$s_1 = 0$	FALSE(s_1)
2	$s_2 = 0$	FALSE(s_2)
3	$s_3 = 0$	FALSE(s_3)
4	$a \rightarrow s_1 = s_1^{'}$	$NOT(a)$
5	$b \rightarrow s_2 = s_2^{'}$	$NOT(b)$
6	$s_2^{'} \rightarrow s_3 = s_3^{'}$	b
7	$s_1^{'} \rightarrow s_2^{'} = s_2^{''}$	$NOT(a) \rightarrow NOT(b)$
8	$a \rightarrow s_3^{'} = s_3^{''}$	$a \rightarrow b$
9	$s_1 = 0$	FALSE(s_1)
10	$s_2^{''} \rightarrow s_1 = s_1^{'}$	$NOT(NOT(a) \rightarrow NOT(b))$
11	$s_3^{''} \rightarrow s_1^{'} = s_1^{''}$	$XOR(a,\ b)$

11 computational steps by executing the algorithms for the three-input AND gate and the two-input AND gate, according to Table 7.

$$AND(a,\ b,\ c) \equiv (c \rightarrow [((b \rightarrow (a \rightarrow)) \rightarrow 0) \rightarrow 0]) \rightarrow 0 \equiv (c \rightarrow (b \rightarrow (a \rightarrow 0))) \rightarrow 0 \quad (11)$$

3.2 Proposed Method for Implementing Shift Registers Required by Lightweight Stream Ciphers

In conventional architectures, it is common to use D flip-flops for implementing shift registers. In the CIM-

Table 6. Implementation algorithm of a three-input IMPLY-based serial AND gate

Step	Operation	Equivalent Logic
1	$s_1 = 0$	FALSE(s_1)
2	$s_2 = 0$	FALSE(s_2)
3	$a \rightarrow s_1 = s_1^{'}$	$NOT(a)$
4	$b \rightarrow s_1^{'} = s_1^{''}$	$b \rightarrow NOT(a)$
5	$c \rightarrow s_1^{''} = s_1^{'''}$	$c \rightarrow (b \rightarrow NOT(a))$
6	$s_1^{'''} \rightarrow s_2 = s_2^{'}$	$AND(a,\ b,\ c)$

Table 7. Implementation algorithm of a four-input IMPLY-based serial AND gate.

Step	Operation	Equivalent Logic
1	$s_1 = 0$	FALSE(s_1)
2	$s_2 = 0$	FALSE(s_2)
3	$a \rightarrow s_1 = s_1^{'}$	$NOT(a)$
4	$b \rightarrow s_1^{'} = s_1^{''}$	$b \rightarrow NOT(a)$
5	$c \rightarrow s_1^{''} = s_1^{'''}$	$c \rightarrow (b \rightarrow NOT(a))$
6	$s_1^{'''} \rightarrow s_2 = s_2^{'}$	$AND(a,\ b,\ c)$
7	$s_1 = 0$	FALSE(s_1)
8	$d \rightarrow s_1 = s_1^{'}$	$NOT(d)$
9	$s_2^{'} \rightarrow s_1^{'} = s_1^{''}$	$AND(a,\ b,\ c) \rightarrow NOT(d)$
10	$s_2 = 0$	FALSE(s_2)
11	$s_1^{''} \rightarrow s_2 = s_2^{'}$	$AND(a,\ b,\ c,\ d)$

A architecture, as designers implement different logical and arithmetic circuits with memristors, there is no need to design or use memristive D flip-flops. The purpose of using memristors in the CIM-A architecture is the same as that of a D flip-flop, as memristors are inherently capable of storing data within the crossbar array. Memristors can also be used for data processing. In a shift register, the stored data is shifted bit-by-bit in each cycle from the m^{th} cell to the $m + 1^{th}$ or $m - 1^{th}$ cell, depending on the shift direction. In the LSB or MSB of each register, a new bit is entered in each cycle, computed using a special or random calculation method, or assigned a fixed value. Therefore, in each cycle, a new bit is placed in the register, and one is removed. In a D flip-flop, data is placed on the data input line before the rising edge of the clock pulse, and it is stored after the rising edge. This process is completely different in the IMPLY-based serial architecture considered here. In this architecture, a buffer should be applied to shift each bit from one cell to another. The buffer implementation algorithm (two consecutive inverters, see Table 2) shifts data in four computational steps using two work memristors.

Using a buffer to shift data in the shift registers of both the Trivium and Grain-128a ciphers is a reliable method. The disadvantage of this method is that each bit of data is shifted in four computational steps. From Figures 1 and 2, a key point for efficiently designing IMPLY-based shift registers can be identified. In each cycle of processing data in these two ciphers, a few cells in each register are directly involved in the computation, and a large number of bits are only shifted from one cell to its neighbor at the end of each cycle. According to Figure 1, only five bits (bits 66, 69, and 91-93) of Trivium's shift register A are used in the computations of the two-input XOR, two-input AND, and three-input XOR logic gates. The other bits of this register (in the Trivium cipher structure) are only shifted one position to the right (the more significant bit) at the end of each cycle. In this process, it is important that the aforementioned bits have the correct value in each processing cycle to maintain the security of the cipher. The same principle applies to the other two registers of the Trivium cipher and the two registers of the Grain-128a cipher. Considering the above points, the number of computational steps can be significantly reduced by replacing some buffers with inverters, so that each buffer is replaced by an inverter, reducing the number of computational steps by two.

The use of the aforementioned method is accompanied by certain design constraints. Specifically, the correctness of the bits used in the computations must be ensured in every processing cycle. Therefore, an algorithm must be proposed and applied that maintains the integrity of the data stored in the memristors required for logical and arithmetic processing in all cycles. Hence, in addition to improving circuit evaluation criteria such as the number of computational steps, the cipher operates correctly without any functional disruption. The proposed algorithm is as follows:

The data located in consecutive memristors and used directly in logical processing (such as two-input XOR gates or two-input AND gates) must be transferred from the main memristor (the first memristor, e.g., k) to its neighbor (the second memristor, e.g., $k^{'}$) in each cycle using only a buffer. In other words, a working memristor and a target memristor (the second memristor, $k^{'}$), where the data is stored, are used to transfer the bit of the first memristor to the target memristor in four computational steps using the IMPLY-based buffer implementation algorithm. This process is performed in all processing cycles. For example, consider Figure 1. In shift register A, bits 91

(k) and 92 ($k^{'}$) are both involved in the computations. In each of the 1152 initialization cycles (in the warm-up phase) or from cycle 1153 onwards, only the buffer implementation algorithm must be used to transfer data from the 91^{st} bit of shift register A to its 92^{nd} bit. The main problem, however, concerns the first memristor (k). If the distance of the input data from the first memristor (k) is an odd number of memristors (for example, a distance of one, three, five, etc.), the desired value is shifted with the buffer in the cycle preceding the data shift. Here, the input data is either the logical value stored in the memristor used in logical and arithmetic computations during the previous cycle, or the input bit of the register. For example, in the Trivium's first shift register (A), the input memristor for the 91^{st} cell (k) in the proposed algorithm is the output of the 66^{th} memristor ($k^{''}$), and for the 65^{th} cell ($k^{'''}$), the input of the register is considered as the input. The distance of the data from the target memristor is of great importance. If the distance is equal to "$2 \times n - 1$" bits and 'n' buffers and 'n' inverters are used, then only an inverter is used in the data shifts from cycle "$2 \times n + 1$" onwards. For example, if the distance between the input data and the target memristor is nine memristors, 'n' is equal to five, meaning that the combination of buffers and inverters (respectively) is used for 10 consecutive cycles, and from the 11^{th} cycle onwards, only the inverter is used. If the number of memristors placed between the input data and the target memristor is "$2 \times n$", the buffers and inverters are used 'n' times, and from the "$2 \times n + 1^{th}$" cycle onwards, only the buffer is applied to shift data. If the distance between the input data and the target memristor is 10 memristors, the buffers and inverters are used five times, and only the buffer is used to shift data from the 11^{th} cycle onwards. The other bits stored in the memristors before the first memristor (k) and after the second memristor ($k^{'}$) are shifted only by the inverters (see the inverter implementation algorithm in Table 2), and the shifted values are stored in the adjacent memristors. If the input data is directly stored in the first memristor, only the buffer should be used. For a better understanding of the proposed algorithm, please refer to Algorithm 1 and Tables 8 and 9. Algorithm 1 presents the formal algorithmic explanation (pseudocode) of the proposed method for implementing shift registers. In these toy examples, the conventional and proposed shift register implementation algorithms are shown over 12 cycles. In Tables 8 and 9, 'n' equals 5 and 6, respectively. In these tables, the columns marked in green refer to the memristors involved in the computations (k and $k^{'}$).

Algorithm 1 The formal algorithmic explanation (pseudocode) of the proposed method for implementing shift registers

Require: Array of Memristors $\mathbf{M}[0 \dots L-1]$
Require: Tap Points (Computational Memristors) $\mathbf{T}$
Require: Input bit IN_BIT
Require: Cycle count c
Ensure: Updated Array $\mathbf{M}$ with shifted and corrected data

procedure SHIFT_REGISTERS($\mathbf{M}, \mathbf{T}, IN_BIT, c$)
  for $i = L-1$ **down to** 1 **do**
    if $\mathbf{M}[i] \in \mathbf{T}$ **and** $\mathbf{M}[i-1] \in \mathbf{T}$ **then**
      ▷ Direct computation neighbors: Must maintain exact value
      BUFFER($\mathbf{M}[i-1], \mathbf{M}[i]$) ▷ 4 computational steps
    else
      $D \leftarrow$ Distance from previous input/tap to $\mathbf{M}[i]$
      $n \leftarrow \lfloor D/2 \rfloor$
      if D is ODD ($D = 2n - 1$) **then**
        if $c \leq 2n$ **then**
          Apply alternating BUFFER() and INVERTER() based on parity of c
        else
          INVERTER($\mathbf{M}[i-1], \mathbf{M}[i]$) ▷ 2 computational steps
        end if
      else if D is EVEN ($D = 2n$) **then**
        if $c \leq 2n$ **then**
          Apply alternating BUFFER() and INVERTER() based on parity of c
        else
          BUFFER($\mathbf{M}[i-1], \mathbf{M}[i]$) ▷ 4 computational steps
        end if
      end if
    end if
  end for
  ▷ Handle immediate input to first memristor
  if $\mathbf{M}[0] \in \mathbf{T}$ **then**
    BUFFER($IN_BIT, \mathbf{M}[0]$)
  else
    INVERTER($IN_BIT, \mathbf{M}[0]$)
  end if
end procedure

3.3 Step-by-step Implementation of the Trivium and Grain-128a Lightweight Stream Ciphers Applying the IMPLY Method in a Serial Architecture

In this subsection, after examining the implementation details of the basic logical and arithmetic blocks from

Table 8. A toy example of an 8-bit conventional (buffer-based) and an 8-bit proposed (inverter-based) shift register with a distance of 5 cells

Conventional buffer-based 8-bit shift register									
Cycle	Input	0	1	2	3	4	5	6	7
0	1	1	0	1	1	1	0	1	0
1	0	1	1	0	1	1	1	0	1
2	1	0	1	1	0	1	1	1	0
3	0	1	0	1	1	0	1	1	1
4	0	0	1	0	1	1	0	1	1
5	1	0	0	1	0	1	1	0	1
6	1	1	0	0	1	0	1	1	0
7	0	1	1	0	0	1	0	1	1
8	1	0	1	1	0	0	1	0	1
9	0	1	0	1	1	0	0	1	0
10	1	0	1	0	1	1	0	0	1
11	1	1	0	1	0	1	1	0	0
12		1	1	0	1	0	1	1	0

Proposed inverter-based 8-bit shift register									
Cycle	Input	0	1	2	3	4	5	6	7
0	1	1	0	1	1	1	0	1	0
1	0	0	0	1	0	0	1	0	0
2	1	1	1	1	0	1	1	1	1
3	0	0	0	0	0	1	1	1	0
4	0	1	1	1	1	1	0	1	0
5	1	1	0	0	0	0	1	0	0
6	1	0	0	1	1	1	1	1	1
7	0	0	1	1	0	0	0	1	0
8	1	1	1	0	0	1	1	0	0
9	0	0	0	0	1	1	0	1	1
10	1	1	1	1	1	0	0	0	0
11	1	0	0	0	0	0	1	0	1
12		0	1	1	1	1	1	1	1

Table 9. A toy example of an 8-bit conventional (buffer-based) and an 8-bit proposed (inverter-based) shift register with a distance of 6 cells.

Conventional buffer-based 8-bit shift register									
Cycle	Input	0	1	2	3	4	5	6	7
0	1	1	0	1	1	1	0	1	0
1	0	1	1	0	1	1	1	0	1
2	1	0	1	1	0	1	1	1	0
3	0	1	0	1	1	0	1	1	1
4	0	0	1	0	1	1	0	1	1
5	1	0	0	1	0	1	1	0	1
6	1	1	0	0	1	0	1	1	0
7	0	1	1	0	0	1	0	1	1
8	1	0	1	1	0	0	1	0	1
9	0	1	0	1	1	0	0	1	0
10	1	0	1	0	1	1	0	0	1
11	1	1	0	1	0	1	1	0	0
12		1	1	0	1	0	1	1	0

Proposed inverter-based 8-bit shift register									
Cycle	Input	0	1	2	3	4	5	6	7
0	1	1	0	1	1	1	0	1	0
1	0	0	0	1	0	0	0	0	1
2	1	1	1	1	0	1	1	1	0
3	0	0	0	0	0	1	0	1	1
4	0	1	1	1	1	1	0	1	1
5	1	1	0	0	0	0	0	0	1
6	1	0	0	1	1	1	1	1	0
7	0	0	1	1	0	0	0	1	1
8	1	1	1	0	0	1	1	0	1
9	0	0	0	0	1	1	0	1	0
10	1	1	1	1	1	0	0	0	1
11	1	0	0	0	0	0	1	0	0
12		0	1	1	1	1	1	1	0

the previous subsections, the implementation of each IMPLY-based lightweight stream cipher, the Trivium and Grain-128a, is analyzed.

3.3.1 Step-by-step Implementation Algorithm of the IMPLY-based Trivium Cipher

The implementation of this cipher is divided into three parts. The first part deals with the initialization of



the shift registers using random numbers, which, as in [14], is assumed by default in this research. The second part is the Trivium initialization phase, where data is calculated and processed 1152 times, and the third part starts at cycle 1153, where one bit of the keystream is generated in each cycle. To store data and implement this cipher using IMPLY logic, 288 memristors are required for implementing the registers, along with five work memristors s_0-s_4 and one output memristor.

In the computations related to register A in the structure shown in Figure 1, two two-input XOR gates and one two-input AND gate are required, yielding a total of 23 computational steps. Three work memristors are used in the computations of this part, and the results are stored in s_0 and s_2. At the end of this stage, the logical values of s_1 and A_{93} are no longer required, so these memristors can be used to store intermediate values in the next stages. The computations for registers B and C, similar to the computations for register A, are performed using two two-input XOR gates and one two-input AND gate, and the number of computational steps required for each of these registers is also 23 steps. The data required for subsequent processing steps related to registers B and C are stored in memristors A_{93} and s_3 (for register B) and in memristors B_{84} and s_4 (for register C).

After each processing cycle of the Trivium cipher, a two-input XOR gate is used to calculate the input bit for each of the registers (A_1, B_1, and C_1). Hence, three two-input XOR gates are required to compute the values to be stored in the first memristor of each register, requiring a total of 27 computational steps for this stage without the need for any new work memristors. After computing the input bits of the registers, the implementation algorithm for the two-input XOR gate must be executed twice to compute the output s_i, and this is also completed in 18 computational steps.

The remainder of this subsection explains the details of how data is shifted bit-by-bit in the shift registers of the Trivium lightweight stream cipher. In the previous subsection, the proposed method for implementing fast, energy-efficient shift registers using IMPLY-based inverters and buffers was described in detail. This method is applied in all three shift registers of the Trivium cipher's structure, as shown in Figure 1. The functionality of the shift register using the proposed method can be examined and observed in detail in Tables 10-12.

As an example, the details of how the stored bits in the memristors of register A are shifted are explained here. Based on the contents of Table 10, it can be concluded that two buffers and 88 inverters must be used to shift data directly in each processing cycle. Special bits are those stored in the memristors before they are involved in logical and arithmetic computations; shifting data in these memristors must be performed carefully. Accordingly, three buffers and three inverters in the first and second cycles, 40 buffers and 20 inverters in the third to 22^{nd} cycles, and 66 inverters and 66 buffers in the 23^{rd} to 66^{th} cycles are required to implement shift register A based on the proposed method, correctly shifting bits to specific memristors. From the 67^{th} cycle onwards, two inverters and one buffer are required in each cycle to transfer data in shift register A. The proposed method of shifting data in shift register A was examined in detail. A similar procedure is employed in the other two shift registers (B and C). Table 13 summarizes the number of inverters and buffers for different cycles of each register used in the Trivium lightweight stream cipher.

3.3.2 Step-by-step Implementation Algorithm of the IMPLY-based Grain-128a Cipher

The lightweight Grain-128a stream cipher also performs computations in three stages, like the Trivium cipher. The first stage initializes the shift registers, which will not be discussed in this research. The second stage, called the "pre-initialization" stage, is performed in 256 processing cycles, and the keystream is generated from the 257^{th} round, which is the third stage of implementing this lightweight stream cipher. In this subsection of the article, each part of the second and third stages of the Grain-128a cipher will be examined and implemented based on stateful IMPLY logic according to Figure 2. Two hundred and sixty-two memristors are needed to implement this cipher, including two sets of 128 memristors for implementing 128-bit LFSR and 128-bit NFSR, along with six work memristors. If the designer needs to store the output (y) separately in a specific memristor, the number of memristors increases to 263 (see Figure 2).

To calculate the output (y), two values, $h(x)$ and a set of $b-terms$, must be calculated according to (7) and (8). The final result is obtained by performing the two-input XOR algorithm of the values stored in the 93^{rd} bit of the LFSR with the two-input XOR product of the calculated values $h(x)$ and $b-terms$. The logical computations of the function $h(x)$ include four two-input AND gates, four two-input XOR gates, and one three-input AND gate, which are performed in 62 computational steps, and the final result is stored in the first work memristor (s_0). The implementation of the computations related to $b-terms$ is executed using six two-input XOR logic gates, one of which is implemented by applying the algorithm of Table 5, $XOR(b_2, b_{15})$, and the other gates are calculated using the algorithm of Table 4. The important point in these

Table 10. Details of the bit by bit data shifting process in Trivium's shift register A based on the proposed method.

Memristors involved in data shifting	**Applied circuits**	**Additional Details**
$A_{92} \rightarrow A_{93}$	1 buffer	
$A_{91} \rightarrow A_{92}$	1 buffer	
$A_{90} \rightarrow A_{91}$	Buffers and inverters	21 intervening memristors; 11 buffers/inverters, then inverters only.
$A_{70} \rightarrow A_{90}$	20 inverters	
$A_{69} \rightarrow A_{70}$	1 inverter	
$A_{68} \rightarrow A_{69}$	Buffers and inverters	2 intervening memristors; 1 buffers/inverters, then buffers only.
$A_{67} \rightarrow A_{68}$	1 inverter	
$A_{66} \rightarrow A_{67}$	1 inverter	
$A_{65} \rightarrow A_{66}$	Buffers and inverters	65 intervening memristors; 33 buffers/inverters, then inverters only.
$A_1 \rightarrow A_{65}$	64 inverters	
$Input \rightarrow A_1$	1 inverter	

Table 11. Details of the bit by bit data shifting process in Trivium's shift register B based on the proposed method

Memristors involved in data shifting	**Applied circuits**	**Additional Details**
$B_{83} \rightarrow B_{84}$	1 buffer	
$B_{82} \rightarrow B_{83}$	1 buffer	
$B_{81} \rightarrow B_{82}$	Buffers and inverters	3 intervening memristors; 2 buffers/inverters, then inverters only.
$B_{80} \rightarrow B_{81}$	1 inverter	
$B_{79} \rightarrow B_{80}$	1 inverter	
$B_{78} \rightarrow B_{79}$	1 inverter	
$B_{77} \rightarrow B_{78}$	Buffers and inverters	8 intervening memristors; 4 buffers/inverters, then buffers only.
$B_{70} \rightarrow B_{77}$	7 inverters	
$A_{69} \rightarrow A_{70}$	1 inverter	2 intervening memristors; 1 buffers/inverters, then buffers only.
$B_{68} \rightarrow B_{69}$	Buffers and inverters	68 intervening memristors; 34 buffers/inverters, then buffers only.
$B_1 \rightarrow B_{68}$	67 inverters	
$Input \rightarrow B_1$	1 inverter	

computations is that the values stored in the main memristors remain completely intact. The output of these calculations is obtained after 56 computational steps. The output of this step is also stored in a work

Table 12. Details of the bit by bit data shifting process in Trivium's shift register C based on the proposed method.

Memristors involved in data shifting	Applied circuits	Additional Details
$C_{110} \rightarrow C_{111}$	1 buffer	
$C_{109} \rightarrow C_{110}$	1 buffer	
$C_{108} \rightarrow C_{109}$	Buffers and inverters	21 intervening memristors; 11 buffers/inverters, then inverters only.
$C_{88} \rightarrow C_{108}$	20 inverters	
$C_{87} \rightarrow C_{88}$	1 inverter	
$C_{86} \rightarrow C_{87}$	Buffers and inverters	20 intervening memristors; 10 buffers/inverters, then buffers only.
$C_{67} \rightarrow C_{86}$	19 inverter	
$C_{66} \rightarrow C_{67}$	1 inverter	
$C_{65} \rightarrow C_{66}$	Buffers and inverters	65 intervening memristors; 33 buffers/inverters, then inverters only.
$C_1 \rightarrow C_{65}$	64 inverters	
$Input \rightarrow C_1$	1 inverter	

Table 13. Inverter and buffer counts for Trivium's shift registers A, B, and C per cycle.

Cycles	No. of buffers	No. of inverters	Cycles	No. of buffers	No. of inverters	Cycles	No. of buffers	No. of inverters
Shift register A			Shift register B			Shift register C		
1-2	7	179	1-2	7	161	1-20	70	2150
3-22	80	1780	3-4	7	179	21-22	8	214
23-66	154	3938	5-8	12	324	23-66	154	4730
67	3	90	9-68	210	4830	67	3	108
68 and onwards	3	90	69 and onwards	4	80	68 and onwards	3	108

memristor (s_2). The final output is also calculated using two two-input XOR gates in 18 computational steps, as shown in (8). The computations related to the LFSR feedback, performed in the "pre-initialization" and keystream stages according to (5), require five two-input XOR logic gates and a total of 47 computational steps. The first XOR is calculated using the algorithm of Table 5, and the other XORs are calculated using the algorithm of Table 4 to preserve the input values.

In the second and third stages of the Grain-128a cipher implementation, the NFSR feedback calculation is divided into two parts. It has a higher computational complexity than the other logical parts of this cipher. The first part computes the subsections mentioned in (6), which involve several XOR and AND gates. The first part specifically includes the computation of 15 two-input XOR gates: 14 are computed using the algorithm in Table 4, and one is computed using the algorithm in Table 5. The total number of steps in this stage is 137 (126 + 11 steps). Eight two-input AND gates (five computational steps), totaling 40 steps, and three three-input AND gates (six computational steps), using 18 steps, are also used to calculate (6). It should be noted that all the main memristors remain intact after these calculations, with 195 computational steps. The final result of this part is also stored in one of the work memristors (s_2).

In the "pre-initialization" stage of the Grain-128a

lightweight stream cipher, which is the first 256 rounds of its execution, the output of each round (the same value stored in the first work memristor (s_0)) is XORed as one of the inputs of two two-input XOR gates with the feedbacks of the registers in this cipher, and the results of each XOR are considered as the input bits of each register.

If the data shift in the Grain-128a's LFSR and NFSR, which consists of 256 bits in total, is performed using IMPLY-based buffers, it takes 1024 computational steps per cycle. By using the proposed method and replacing inverters with buffers in these registers, the computational steps per cycle can be significantly reduced. The implementation of Grain-128a's shift registers is much more complex than the registers in the Trivium cipher. A similar approach to Tables 10-12 is also applied to determine the details of performing data shifts in the Grain-128a cipher shift registers. According to the proposed approach, 114 inverters and two buffers are required to transfer data from each memristor to its neighbor in the LFSR per cycle. The problem's complexity, however, stems from the other 12 shifts directly involved in the computations. The number of inverters and buffers applied to shift bits that contribute to computations each cycle is tabulated in Table 14. A similar approach should be applied to examine the bit-by-bit shift process in an NFSR to reduce its hardware complexity significantly. In all processing cycles, 92 inverters and 12 buffers are required to shift data between memristors. The shift of the bits stored in the memristors that participate in the logical computations follows the specific characteristics of the proposed algorithm. A summary of the number of buffers and inverters required per cycle for the 24 bits involved in the logical computations of the NFSR is tabulated in Table 15.

4 Simulation Results, Discussion, and Evaluation

In the previous section, the implementation of IMPLY-based Trivium and Grain-128a, two lightweight stream ciphers, in the serial architecture was investigated and reported. In subsection 4.1, first, the memristor model applied in circuit-level simulation is introduced; second, the method of applying this model to perform the simulation is explained; and in the last part of this subsection, the simulation results are presented and expounded in detail. In the second subsection, the behavioral simulation results for the proposed encryption/decryption structures (based on Trivium and Grain-128a) and their application to image steganography will be reported.

4.1 Circuit-level Simulation, Method, and Results

For simulating memristor-based logical and arithmetic circuits, various behavioral and physical SPICE models have been presented. The logical and arithmetic blocks applied in the structure of Trivium and Grain-128a in this paper have been simulated by applying the modified open-source Voltage Threshold Adaptive Memristor (VTEAM) model, which has been fitted according to the parameters of a discrete memristor (BS-AF-W model) developed by "Knowm Inc." [2]. This model has been applied in [2, 4, 6, 36–38]. The model parameters and standard coefficients are shown in Table 16. The specific parameters considered for the simulation of the IMPLY-based blocks are listed in Table 16, too. The values reported in Table 16 are contemplated based on the parameters reported in [2, 4, 6, 36–38]. LTspice has been used to simulate the IMPLY-based logical and arithmetic blocks described in this paper.

To analyze the functionality of the simulated circuits, note that, since all logic and arithmetic blocks are stateful, the input and output logical values are determined by the memristors' resistance. The maximum and minimum resistance values of the memristor are equal to logic '0' and logic '1', respectively. It should be noted that the functionality of IMPLY-based implementation algorithms must be evaluated by considering all possible input states. First, the input values must be initialized to the input memristors. Then the implementation algorithm runs for each state, and the correctness of the output is evaluated across all states against the intended resistance value of the output memristor. The number of computational steps, which is proportional to the computational delay, determines the performance. The energy consumption is computed using the method described in [2, 4, 6, 36–39]. For each input state, the energy consumption of each memristor in the row or column of the array is calculated, and the average over all states is reported as the estimated energy consumption. The number of memristors required is another criterion examined. In the following, the memristors' energy consumption, number of computational steps, and number of memristors required for each basic logic and arithmetic unit are calculated after verification. Then, the circuit evaluation criteria for the IMPLY-based Trivium and Grain-128a stream ciphers are reported in two scenarios. In the first scenario for each of the two stream ciphers, conventional algorithms are used without the improvements detailed in the previous section, whereas in the second scenario, the proposed methods are used to improve the circuit evaluation criteria.

As mentioned, the functionality of the implementa-



Table 14. Number of buffers and inverters required per cycle for the bits involved in the logical computations of the Grain-128a's LFSR.

Cycles	No. of buffers	No. of inverters	Cycles	No. of buffers	No. of inverters	Cycles	No. of buffers	No. of inverters
1-2	12	12	3-4	10	14	5-6	10	14
7-8	12	12	9-10	13	11	11-12	12	12
13-18	33	39	19-32	63	105	33 and onwards	4	8

Table 15. Number of buffers and inverters required per cycle for the bits involved in the logical computations of the Grain-128a's NFSR.

Cycles	No. of buffers	No. of inverters	Cycles	No. of buffers	No. of inverters	Cycles	No. of buffers	No. of inverters
1-2	24	24	3-4	18	30	5-6	16	32
7-8	16	32	9-10	15	33	11-12	15	33
13-32	15	33	33-34	16	32	35 and onwards	8	16

Table 16. Setup values of IMPLY logic and VTEAM model [2, 4, 6].

Parameter	Value	Parameter	Value	Parameter	Value
v_{off}	0.7 V	v_{on}	-10 mV	α_{off}	3
α_{on}	3	R_{off}	1 MΩ	R_{on}	10 kΩ
k_{on}	-0.5 $\frac{nm}{s}$	k_{off}	1 $\frac{cm}{s}$	w_{off}	0 nm
w_{on}	3 nm	w_C	107 pm	a_{off}	3 nm
a_{on}	0 nm	v_{set}	1 V	v_{reset}	1 V
v_{cond}	900 mV	R_G	40 KΩ	t_{pulse}	30 μs

tion algorithms for all IMPLY-based logic gates and basic arithmetic circuits used in the design of the Trivium and Grain-128a ciphers has been examined by considering all possible input states and applying the model in the LTSPICE simulator. Two-input XOR gates (according to the algorithms of Tables 4 and 5), a three-input XOR gate with the ability to retain input values, two-input, three-input, and four-input AND gates, buffers, and inverters are the circuits that are evaluated. The output waveforms of three gates, as examples of simulation results, are shown in Figures 5-7. In Figure 5, the output waveform of a two-input XOR logic gate in which the logic value of one of the input memristors is changed is plotted. The inputs are assumed to be equal to "00", and the output is stored in the output memristor (the first work memristor, s_1) after nine computational cycles. According to Table 16 and the reported value of t_{step}, the output is computed and stored in 240-270 μs. The analysis of the functionality of the other two-input XOR gate mentioned in Table 4, which computes the output in the last step of the implementation algorithm (300-330 μs), is shown in Figure 6. The inputs of this gate equal "00", and the input values stored in the memristors remain unchanged after the execution of the implementation algorithm. The output is stored in the first work memristor (s_1) in the final step. In Figure 7, the simulation result of the three-input AND gate ($A_{in}B_{in}C_{in}$="111") is calculated in six computational steps according to (11) in the time interval 0 to 180 μs. The output is obtained in the last computational step (150-180 μs) and stored in the second work memristor (s_2).

After introducing the applied model, familiarizing with the simulation method and circuit evaluation criteria, and presenting a number of simulation outputs, the results of the circuit evaluation criteria for the IMPLY-based basic logic gates required in the implementation of the Trivium and Grain-128a, lightweight stream ciphers, are summarized in Table 17. The numbers reported in Table 17 are widely used in estimating the circuit evaluation criteria.

Subsection 3.3.1 explains the implementation details of the Trivium cipher using the IMPLY method. The previous paragraphs of this subsection also mentioned the details of the circuit-level implementation of the basic building blocks of this cipher. Tables 18 and 19 specify the number of required computational units, the number of computational steps, and the estimated memristors' energy consumption for the initialization and keystream generation stages of the

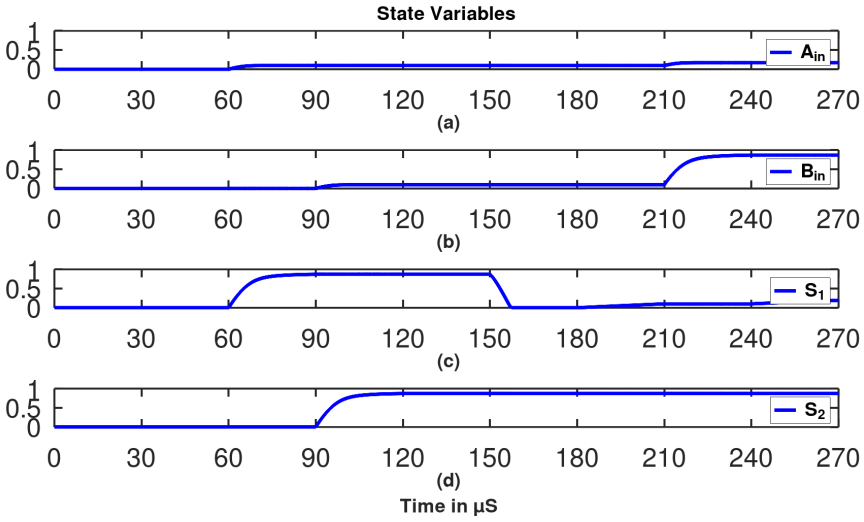


Figure 5. The output waveform of a destructive IMPLY-based serial two-input XOR gate introduced in [6] for input state "00".

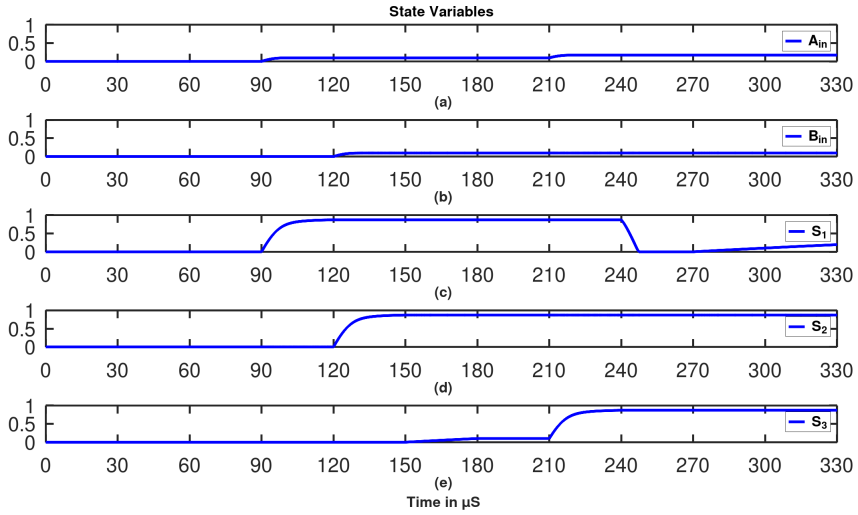


Figure 6. The output waveform of a non-destructive IMPLY-based serial two-input XOR gate for input state "00".

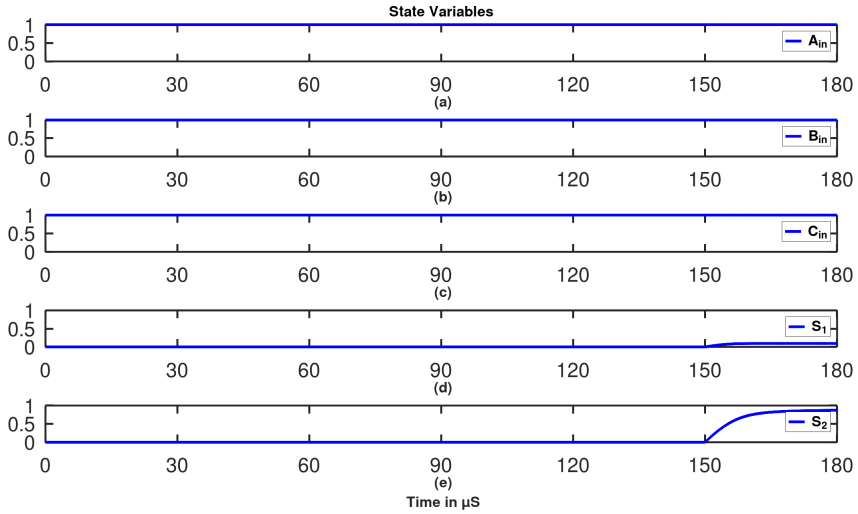


Figure 7. The output waveform of an IMPLY-based serial three-input AND gate for input state "111".

Table 17. Summary of the circuit evaluation criteria for the IMPLY-based basic logic gates required in the implementation of the Trivium and Grain-128a.

Gate	No. of Memristors	No. of computational steps	Estimated memristors' energy consumption (nJ)
Inverter	2	2	0.1291
Buffer	3	4	0.269
2-input AND	4	5	0.3833
3-input AND	5	6	0.5025
4-input AND	6	11	0.9131
2-input XOR (destructive) [6]	4	9	0.7426
2-input XOR (non-destructive)	5	11	0.9146
3-input XOR	6	20	1.711

Trivium cipher, respectively.

The initialization phase of the Trivium cipher requires 293 memristors, including 288 input/output memristors and five work memristors, and the number of memristors is not increased in all cycles. Overall, the initialization phase of the IMPLY-based Trivium cipher takes 797266 computational steps and consumes 53.4731 μJ. Each round of generating a keystream bit also requires the same number of memristors, takes 710 computational steps, and consumes 47.8983 nJ. So, if the number of keystream bits required by the user is considered to be 'n', the total number of computational steps and memristors' energy consumption estimation in the two phases are "$710 \times n + 797266$" steps and "$0.0478 \times n + 53.4731$" μJ, respectively.

The circuit evaluation criteria for the Grain-128a cipher can be examined in the same way as for the Trivium cipher. The "pre-initialization" phase of this cipher lasts 256 cycles, and the keystream is generated from the 257^{th} cycle. The number of memristors and logical units, the computational steps, and the estimated memristors' energy consumption of the units applied in the "pre-initialization" and keystream generation phases are reported in Tables 20 and 21, respectively.

Two hundred and sixty-two memristors participate in the "pre-initialization" phase of the Grain-128a cipher. This phase requires 245830 computational steps and estimates memristors' energy consumption of 17.6811 μJ. Each round of keystream generation requires 942 computational steps and 263 memristors. The memristors' energy consumption estimation for the second stage of the Grain-128a is 0.0666 μJ. The computed values can be generalized for an n-bit keystream. To produce an n-bit keystream of this cipher, "$942 \times n + 245830$" computational steps and 263 memristors are required. The memristors' energy consumption estimate for this cipher in generating an n-bit keystream is also "$0.0666 \times n + 17.6811$" μJ.

A comprehensive comparison of the circuit evaluation criteria for the Trivium and Grain-128a ciphers in the mentioned scenarios is presented in Table 22. The number of memristors required by both ciphers to generate an n-bit keystream is 293 and 263, respectively. The number of computational steps and memristors' energy consumption of the Trivium cipher applying the proposed shift register's implementation algorithm, considering the generation of 10,000 bits, are 18% and 22% less than those of the Grain-128a cipher, respectively, while by increasing the number of output bits by a factor of 10, the number of computational steps and memristors' energy consumption estimation are improved by 24% and 27%.

According to Table 22, it can be concluded that the computational delay and memristors' energy consumption of the Trivium cipher are lower than those of Grain-128a, and it is preferred over Grain-128a if computational delay and energy consumption are of higher priority for the designer. Grain-128a has higher priority than Trivium if the designer's main goal is to use a cipher with a 128-bit key and the circuit-level evaluation metrics are second-tier. As the number of bits tends to infinity, the number of computational steps and the memristors' energy consumption of the improved IMPLY-based Trivium and Grain-

Table 18. Details of the number of computational blocks, computational steps, and estimated memristors' energy consumption of the initialization stage of the serial IMPLY-based Trivium cipher.

Block	**No. of blocks**	**No. of computational steps**	**Estimated memristors' energy consumption (μJ)**
2-input XOR [6]	3456	17280	1.3246
2-input AND	10386	93312	7.699
Inverter in the proposed algorithm (shift register A)	103637	207274	13.3795
Buffer in the proposed algorithm (shift register A)	3499	13996	0.9412
Inverter in the proposed algorithm (shift register B)	92196	184392	11.9025
Buffer in the proposed algorithm (shift register B)	4572	18288	1.2298
Inverter in the proposed algorithm (shift register C)	124382	248764	16.0577
Buffer in the proposed algorithm (shift register C)	3490	13960	0.9388
Buffer in the conventional algorithm (shift register A)	107136	428544	28.8195
Buffer in the conventional algorithm (shift register B)	96768	387072	26.0305
Buffer in the conventional algorithm (shift register C)	127872	511488	34.3975

Table 19. Details of the number of computational blocks, computational steps, and estimated memristors' energy consumption of the keystream generation stage of the serial IMPLY-based Trivium cipher.

Block	**No. of blocks**	**No. of computational steps**	**Estimated memristors' energy consumption (nJ)**
2-input XOR [6]	11	99	8.1686
2-input AND	3	15	1.1499
Inverter in the proposed algorithm	278	556	35.8898
Buffer in the proposed algorithm	10	40	2.69
Buffer in the conventional algorithm	288	1152	77.472

Table 20. Details of the number of computational blocks, computational steps, and estimated memristors' energy consumption of the "pre-initialization" stage of the serial IMPLY-based Grain-128a cipher.

Block	No. of blocks	No. of computational steps	Estimated memristors' energy consumption (μJ)
2-input XOR (destructive) [6]	7936	71424	5.8932
2-input XOR (non-destructive)	768	8448	0.7024
2-input AND	2816	14080	1.0793
3-input AND	768	4608	0.3859
4-input AND	256	2816	0.2237
Inverter in the proposed algorithm (LFSR)	31195	62390	4.0272
Buffer in the proposed algorithm (LFSR)	1573	6292	0.4231
Buffer in the conventional algorithm (LFSR)	32768	131072	8.8145
Inverter in the proposed algorithm (NFSR)	27650	55300	3.5696
Buffer in the proposed algorithm (NFSR)	5118	20472	1.3767
Buffer in the conventional algorithm (NFSR)	32768	131072	8.8145

128a lightweight stream ciphers based on the proposed shift register algorithm are improved by 38%, 42%, 45%, and 32% compared to the conventional design, respectively.

4.1.1 Practical Limitations

This subsection addresses two categories of practical implementation constraints that this study has given less emphasis to. The objective of circuit-level simulations in this subsection was to ensure the functional correctness of the IMPLY-based basic-block implementation algorithms. The results reported in Tables 17-22 are based on the simulations using a valid behavioral model. This model has been used in several prior studies [2, 4, 6, 36–38], allowing others to examine and reuse the results. The algorithms and simulation were developed independently of the implementation platform technology, ensuring the proposed method can be implemented using alternative technologies compatible with stateful memristive methods.

The main contribution of this study is a step-by-step implementation algorithm for porting two lightweight stream ciphers to the CIM-A architecture using the IMPLY method, along with a creative implementation method for the two ciphers' shift-register structures to improve circuit evaluation criteria. It should be noted that peripheral circuits must also be taken into account when assessing improvements in the high-level circuit evaluation metrics of the proposed structures. As a result, improvements in computational steps and energy consumption in IMPLY-based structures are not applied as a 1:1 reduction of the architecture, even though they directly reflect array-level improvements.

Simulation and implementation results of memristor-based circuits, such as the IMPLY gate, can vary under the influence of various factors. Applied circuit voltages, pulse width, and simulation temperature are among the factors whose variations significantly affect

Table 21. Details of the number of computational blocks, computational steps, and estimated memristors' energy consumption of the keystream generation stage of the serial IMPLY-based Grain-128a cipher.

Block	**No. of blocks**	**No. of computational steps**	**Estimated memristors' energy consumption (nJ)**
2-input XOR (destructive) [6]	29	261	21.5354
2-input XOR (non-destructive)	3	33	2.7438
2-input AND	11	55	4.2163
3-input AND	3	18	1.5075
4-input AND	1	11	0.9131
Inverter in the proposed algorithm (LFSR)	122	244	15.7502
Buffer in the proposed algorithm (LFSR)	6	24	1.614
Buffer in the conventional algorithm (LFSR)	128	512	34.432
Inverter in the proposed algorithm (NFSR)	108	216	13.9428
Buffer in the proposed algorithm (NFSR)	20	80	5.38
Buffer in the conventional algorithm (NFSR)	128	512	34.432

Table 22. Comprehensive comparison of circuit evaluation metrics of IMPLY-based serial Trivium and Grain-128a ciphers.

Lightweight stream cipher	**No. of computational steps**	$n = 10000$	$n = 100000$	**Estimated memristors' energy consumption (μJ)**	$n = 10000$	$n = 100000$
Trivium (conventional)	$1152 \times n + 1437696$	12957696	116637696	$0.0867 \times n + 98.2711$	965.2711	8768.2711
Trivium (proposed)	$710 \times n + 797266$	7897266	71797266	$0.0478 \times n + 53.4731$	531.4731	4833.4731
Grain-128a (conventional)	$1646 \times n + 363520$	16823520	164963520	$0.09878 \times n + 25.9135$	1013.7135	9903.9135
Grain-128a (proposed)	$942 \times n + 245830$	9665830	94445830	$0.0666 \times n + 17.6811$	683.6811	6677.6811

the circuit responses [40]. In practical implementation, in addition to the aforementioned parameters, "device-to-device" and "cycle-to-cycle" variations, random switching behavior, resistance state drift, limited endurance, "interconnect resistance", "parasitic capacitances", memristor aging, and "sneak-path currents" have an unwanted effect on device behavior and, consequently, the designed structure [40]. Considering the main goal of the authors, which is to port the aforementioned ciphers from the conventional architecture to the CIM-A architecture while enhancing the circuit evaluation metrics of the ported baseline design, the investigation of Process, Voltage, and Temperature (PVT) robustness (via Monte Carlo analysis), voltage and pulse width sensitivity, and the overheads of the peripheral circuits (e.g., the switches required by the control unit and voltage drivers) are beyond the scope of this article. In [41–44], the

authors investigated the effects of these items on the performance of the IMPLY gate using a similar model.

As mentioned, the analyses and results reported in this research are based on circuit simulations. Security analysis and vulnerability assessment of the proposed architecture, especially the proposed shift register implementation algorithm, including power side-channel leakage, timing leakage, and fault susceptibility, are important in this research and similar work. Conducting experiments to measure these vulnerabilities is of great importance, and it will be an essential aim for future research.

4.2 Application-level Simulation

GNU Octave has been used to simulate the behavior of the serial IMPLY-based Trivium and Grain-128a stream ciphers in cryptography and steganography. All the logic gates and logical/arithmetic units have been simulated behaviorally using the algorithms mentioned in the previous sections, and their functionality has been validated. The proposed IMPLY-based shift register algorithm has also been considered in the behavioral simulation of both ciphers' implementations. The core structures of each cipher were implemented using basic logical circuits designed with the IMPLY method. After verifying the correctness of the cores' functionality, both ciphers were implemented in their two main stages of operation: initialization and keystream generation. In the final code, the desired keystream was generated, and encryption and decryption operations were performed based on the structures of both ciphers.

One goal of implementing lightweight ciphers based on the IMPLY design method is to evaluate their functionality in steganography applications. Hiding data bits in the LSB of each pixel in an image is one of the most common and low-cost methods of steganography [45]. Therefore, to examine the functionality of the implemented circuits in this application, random messages were encrypted using the designed IMPLY-based Trivium and Grain-128a, and the output bits were inserted one bit at a time into the LSBs of the carrier images. The output messages from the previous step were also recovered from the images, and after comparing the recovered data with the original data, it can be deduced that the entire process was performed correctly.

Assessing image quality metrics such as Peak Signal to Noise Ratio (PSNR) and comparing the histograms of the steganographic output with the original image, ensures the reliability of the process. The human eye cannot distinguish the original image from the steganographic output if the PSNR is above 30 dB [46]. Hence, if the PSNR is higher than 30 dB, it can be concluded that the steganographic output is acceptable, as it is not possible to easily distinguish the two images from each other, and even human recognition is impossible. By drawing histograms of the original image and the output of the steganography process, and analyzing the outputs, the user can determine how well this technique works for hiding data.

According to the method applied to the behavioral simulation discussed in the previous paragraphs, five random messages of varying lengths were generated and, after processing, placed in the LSBs of the pixels of five standard 256×256 grayscale images. The output of the steganography process consists of two images, along with the standard grayscale image. The first and second output images are generated from the outputs of the lightweight stream ciphers Trivium and Grain-128a, respectively. Two sets of outputs from the steganography application are shown in Figures 8 and 9. The PSNR values of this application are tabulated in Table 23.

According to the results in Table 23 and Figures 8 and 9, it can be concluded that hiding the outputs of random messages of different lengths encrypted using the Trivium and Grain-128a ciphers in the LSBs of standard grayscale images is acceptable. As the number of encrypted bits increases, image quality decreases, but the results show a large gap between the output PSNR values and the acceptable image quality margin (30 dB). By increasing the number of bits, larger images can be used to improve image quality. Histograms corresponding to Figures 8 and 9 are also shown in Figures 10 and 11; thus, the pixel distributions of the outputs of the steganography application can be compared and evaluated with the reference images.

Since the authors did not present a new steganography algorithm in this section and a basic method has been used for this purpose, in addition, the proposed cryptographic blocks were ported from the conventional architecture to CIM-A without changing the fundamental principles and considering its algorithmic equivalent as the main contribution of the authors, criteria such as Structural Similarity Index Measure (SSIM), entropy, correlation, payload capacity, and steganalysis have not been evaluated, and only PSNR evaluation and histogram plotting have been performed.

5 Conclusion and Future Works

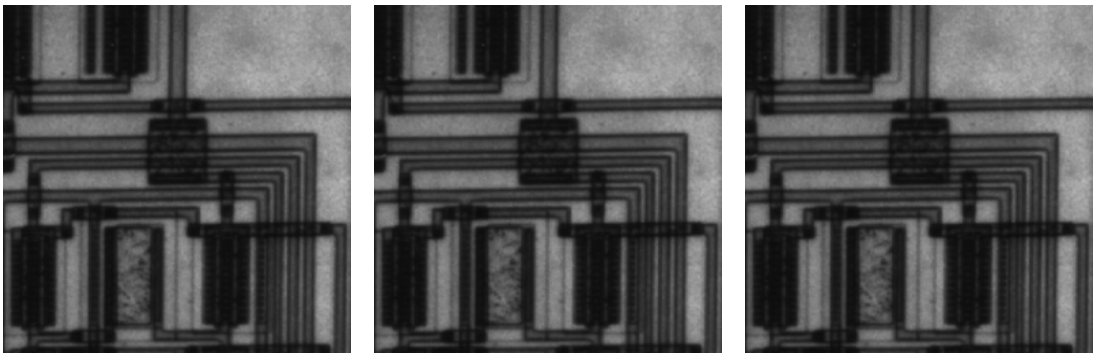
In this paper, two lightweight stream ciphers, Trivium and Grain-128a, are implemented using the serial IMPLY-based design method with a minimum number of memristors harmonized with the structure of the memristive crossbar array. In the redesigned ci-





(a) (b) (c)

Figure 8. Functionality analysis of IMPLY-based lightweight stream ciphers in steganography:(a) Cover image ("boat"), (b) Trivium-based stego image, and (c) Grain-128a stego image.



(a) (b) (c)

Figure 9. Functionality analysis of IMPLY-based lightweight stream ciphers in steganography:(a) Cover image ("circuit"), (b) Trivium-based stego image, and (c) Grain-128a stego image.

Table 23. Evaluation of the PSNR values of the stego images generated using the IMPLY-based Trivium and Grain-128a ciphers.

Cover image	**Trivium-based stego image PSNR value (dB)**	**Grain-128a-based stego image PSNR value (dB)**
Boat	71.642	71.966
Cameraman	69.822	69.668
Circuit	69.715	69.783
Rice	73.199	73.178
Walkbridge	65.139	65.258

phers, an attempt has been made to minimize the number of computational steps. Accordingly, in addition to investigating the possibility of overlapping steps in the implementation of basic logic and arithmetic blocks, by replacing the buffers required in the structure of shift registers of the architectures of the Trivium and Grain-128a ciphers with a combination of inverters and buffers, this study attempts to significantly improve the circuit evaluation criteria, such as the number of computational steps and memristors' energy consumption. Applying the proposed method to redesign the shift register structures in these two lightweight stream ciphers has resulted in improvements of 38%-42% in the number of computational steps and 32%-44% in memristors' energy consumption compared to the conventional design. The improved structure of the IMPLY-based Trivium cipher generates the keystream faster and consumes less energy than Grain-128a, such that by generating 10,000 valid bits employing the improved IMPLY-based Trivium cipher, the number of computational steps and memristors' energy consumption are 18% and 22% less than the circuit evaluation criteria of the IMPLY-based Grain-128a architecture, respectively. By exam-

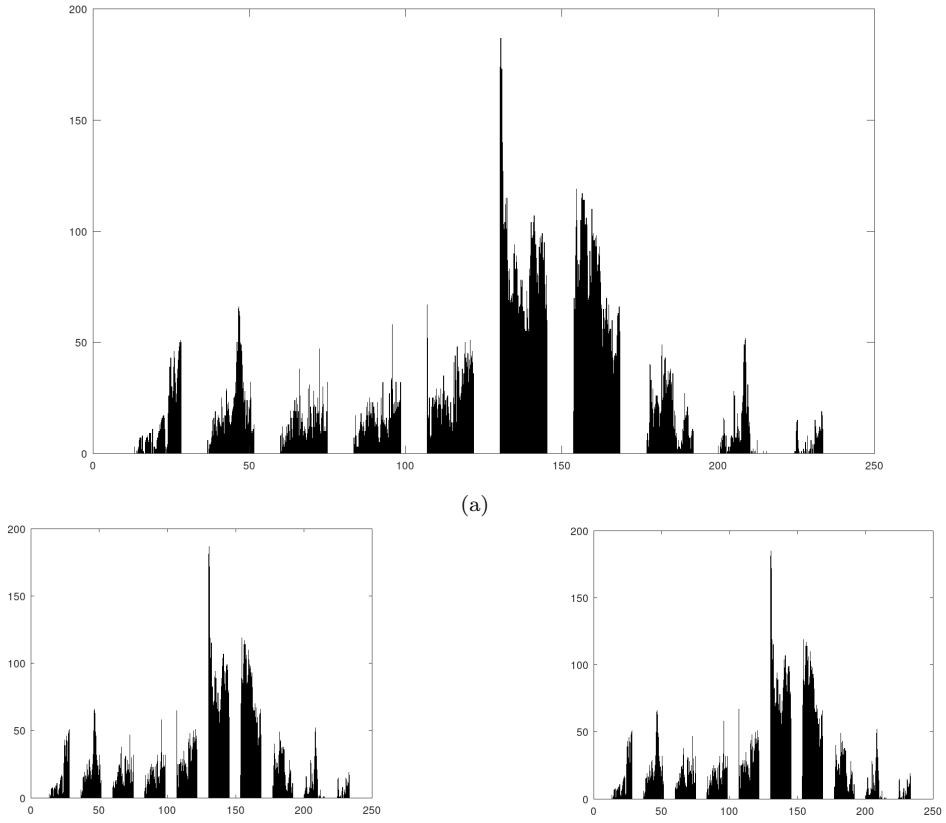


Figure 10. Histograms of (a) Figure 8(a), (b) Figure 8(b), and (c) Figure 8(c).

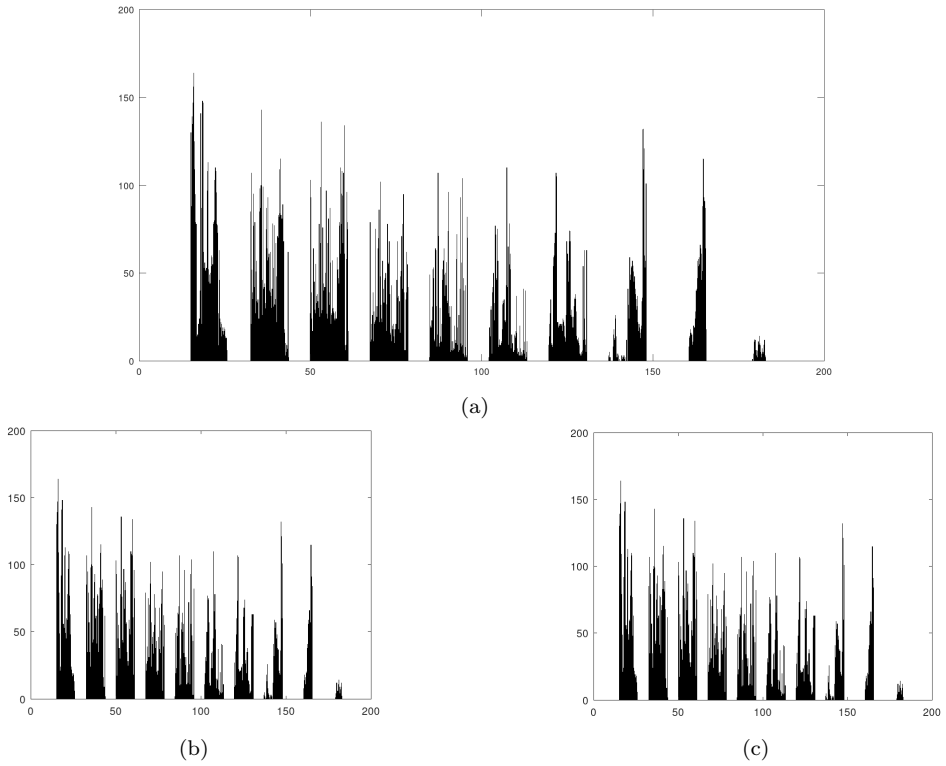


Figure 11. Histograms of (a) Figure 9(a), (b) Figure 9(b), and (c) Figure 9(c).

ining the PSNR criterion and the histogram diagrams of the different output images from the steganography application, it can be concluded that the functionality of the designed circuits in this application was acceptable.

This paper serves as a basis for continuing to design memristive stateful cryptographic units for implementation within the CIM-A architecture. The authors have outlined three of the most significant future works of this study as follows:

- Modifying the structure of conventional memories and utilizing memristive crossbar array structures to implement the proposed IMPLY-based lightweight stream ciphers Trivium and Grain-128a architectures requires designing a dedicated control unit according to the implementation algorithm. Accordingly, the authors consider developing an integrated framework for implementing IMPLY-based control units a key priority for future work,
- Investigating and analyzing the effect of memristor variability and sneak-path currents on the reliability of each of the basic logic blocks applied in the structure of proposed stateful cryptographic architectures is of great importance for advancing the other fundamental objectives of this research, and
- Analyzing the performance of the physical and behavioral model of the proposed architecture by examining the standard cryptographic test vectors to ensure the security of using the proposed structure in practical applications is another area that has been considered for future study. Furthermore, examining potential vulnerabilities through side-channel analysis and other common methods can assess the resilience of the proposed memristive cryptographic structures against various attacks when deployed in the CIM-A architecture.

Acknowledgements

During the preparation of this work, the authors utilized the "Google Gemini 3.1 Pro" to improve the readability, grammar, formal academic tone, and structural organization of the manuscript. After using this tool, the authors meticulously reviewed and edited the content. The authors take full responsibility for the scientific accuracy, originality, and overall integrity of the final publication.

References

[1] Ling Yang, Long Cheng, Yi Li, Haoyang Li, Jiancong Li, Ting-Chang Chang, and Xiangshui Miao. Cryptographic key generation and in situ encryption in one-transistor-one-resistor memristors for hardware security. *Advanced Electronic Materials*, 7(5):2001182, 2021. doi:10.1002/aelm.202170012.

[2] Seyed Erfan Fatemieh, Bahareh Bagheralmoosavi, and Mohammad Reza Reshadinezhad. Energy-efficient and fast memristor-based serial multipliers applicable in image processing. *Results in Engineering*, 25:104013, 2025. doi:10.1016/j.rineng.2025.104013.

[3] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Morgan kaufmann, 2017.

[4] Seyed Erfan Fatemieh and Mohammad Reza Reshadinezhad. Energy-efficient approximate full adders applying memristive serial imply logic for image processing. *Journal of Electrical and Computer Engineering*, 2026(1):7627964, 2026. doi:10.1155/jece/7627964.

[5] Seyed Erfan Fatemieh, Negin Mashayekhi, and Mohammad Reza Reshadinezhad. A high-speed and low-cost approximate full adder in qca technology. *The European Physical Journal Plus*, 140(8):726, 2025. doi:10.1140/epjp/s13360-025-06668-4.

[6] Bahareh Bagheralmoosavi, Seyed Erfan Fatemieh, Mohammad Reza Reshadinezhad, and Antonio Rubio. Power-area efficient serial imply-based 4: 2 compressor applied in data-intensive applications. *Cluster Computing*, 28 (8):512, 2025. doi:10.1007/s10586-025-05180-9.

[7] Samira Shirinabadi Farahani, Mohammad Reza Reshadinezhad, and Seyed Erfan Fatemieh. New design for error-resilient approximate multipliers used in image processing in cntfet technology. *The Journal of Supercomputing*, 80(3):3694–3712, 2024. doi:10.1007/s11227-023-05623-3.

[8] Carlos Carvalho. The gap between processor and memory speeds. In *Proc. of IEEE International Conference on Control and Automation*, pages 27–34, 2002.

[9] Nima TaheriNejad. In-memory computing: Global energy consumption, carbon footprint, technology, and products status quo. In *Proc. 24th IEEE Int. Conf. Nanotechnol.(IEEE-NANO)*, pages 1–6, 2024. doi:10.1109/NANO61778.2024.10628809.

[10] Khaled Alhaj Ali. *New design approaches for flexible architectures and in-memory computing based on memristor technologies*. PhD thesis, Ecole nationale superieure Mines-Telecom Atlantique, 2020. Available at `https://theses.hal.science/tel-03134905/file/2020IMTA0197_AlhajAli-Khaled.pdf`.

[11] Shahar Kvatinsky, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Memristor-based material implication (imply) logic: Design principles and methodolo-

gies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(10):2054–2066, 2013. doi:10.1109/TVLSI.2013.2282132.

[12] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Magic—memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014. doi:10.1109/TCSII.2014.2357292.

[13] Hoang Anh Du Nguyen, Jintao Yu, Muath Abu Lebdeh, Mottaqiallah Taouil, Said Hamdioui, and Francky Catthoor. A classification of memory-centric computing. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 16 (2):1–26, 2020. doi:10.1145/3365837.

[14] Muhammad Ali Siddiqi, Jan Andrés Galvan Hernández, Anteneh Gebreziorgis, Rajendra Bishnoi, Christos Strydis, Said Hamdioui, and Mottaqiallah Taouil. Memristor-based lightweight encryption. In *2023 26th Euromicro Conference on Digital System Design (DSD)*, pages 634–641. IEEE, 2023. doi:10.1109/DSD60849.2023.00092.

[15] Turgay Kaya. Memristor and trivium-based true random number generator. *Physica A: Statistical Mechanics and its Applications*, 542:124071, 2020. doi:10.1016/j.physa.2019.124071.

[16] Vikash Kumar Rai, Somanath Tripathy, and Jimson Mathew. Memristor based random number generator: Architectures and evaluation. *Procedia Computer Science*, 125:576–583, 2018. doi:10.1016/j.procs.2017.12.074.

[17] VK Rai, S Tripathy, and J Mathew. Multi-bit true random number generator for iot devices using memristor. *Security of Internet of Things Nodes: Challenges, Attacks, and Countermeasures*, page 35, 2021. doi:10.1201/9781003127598.

[18] Batel Oved, Orian Leitersdorf, Ronny Ronen, and Shahar Kvatinsky. Hashpim: High-throughput sha-3 via memristive digital processing-in-memory. In *2022 11th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–4. IEEE, 2022. doi:10.1109/MOCAST54814.2022.9837685.

[19] Nahom Gebeyehu Zinabu, Yihenew Wondie Marye, Kula Kekeba Tune, and Samuel Asferaw Demilew. Comprehensive analysis of lightweight cryptographic algorithms for battery-limited internet of things devices. *International Journal of Distributed Sensor Networks*, 2025(1):9639728, 2025. doi:10.1155/dsn/9639728.

[20] Noor Maher Naser and Jolan Rokan Naif. A systematic review of ultra-lightweight encryption algorithms. *International Journal of Nonlinear Analysis and Applications*, 13(1):3825–3851, 2022. doi:10.22075/ijnaa.2022.6167.

[21] Charalampos Manifavas, George Hatzivasilis, Konstantinos Fysarakis, and Yannis Papaefstathiou. A survey of lightweight stream ciphers for embedded systems. *Security and Communication Networks*, 9(10):1226–1246, 2016. doi:10.1002/sec.1399.

[22] Jesús Soto-Cruz, Erica Ruiz-Ibarra, Javier Vázquez-Castillo, Adolfo Espinoza-Ruiz, Alejandro Castillo-Atoche, and Joaquin Mass-Sanchez. A survey of efficient lightweight cryptography for power-constrained microcontrollers. *Technologies*, 13(1):3, 2024. doi:10.3390/technologies13010003.

[23] Christof Paar, Jan Pelzl, and Tim Güneysu. *Understanding cryptography: From established symmetric and asymmetric ciphers to post-quantum algorithms*. Springer Nature, 2024. doi:10.1007/978-3-662-69007-9.

[24] Martin Hell, Thomas Johansson, Willi Meier, Jonathan Sönnerup, and Hirotaka Yoshida. An aead variant of the grain stream cipher. In *International Conference on Codes, Cryptology, and Information Security*, pages 55–71. Springer, 2019. doi:10.1007/978-3-030-16458-4_5.

[25] Fabian Seiler. *Heterogeneous Approximate Adders for Memristive Processing in Array*. PhD thesis, Technische Universitat Wien, 2025.

[26] Christophe De Cannière and Bart Preneel. Trivium. In *New Stream Cipher Designs: The eSTREAM Finalists*, pages 244–266. Springer, 2008. doi:10.1007/978-3-540-68351-3_18.

[27] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of grain-128 with optional authentication. *International Journal of Wireless and Mobile Computing*, 5(1):48–59, 2011. doi:10.1504/IJWMC.2011.04410.

[28] Paul A. Psomiadis. *Security of lightweight cryptographic algorithms*. PhD thesis, National And Kapadistrian University of Athens, 2020.

[29] Saransh Gupta, Mohsen Imani, and Tajana Rosing. Felix: Fast and energy-efficient logic in memory. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7. IEEE, 2018. doi:10.1145/3240765.3240811.

[30] Shokat Ganjeheizadeh Rohani and Nima TaheriNejad. An improved algorithm for imply logic based memristive full-adder. In *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–4. IEEE, 2017. doi:10.1109/CCECE.2017.7946813.

[31] Fabian Seiler and Nima TaheriNejad. An improved serial imply adder algorithm for efficient neural network applications. In *2025 IEEE 16th Latin America Symposium on Circuits and Sys-*

tems (LASCAS), volume 1, pages 1–5. IEEE, 2025. doi:10.1109/LASCAS64004.2025.10966340.

[32] Fabian Seiler, Nima TaheriNejad, et al. An efficient robust serial imply-based in-memristor adder. In *2025 Cross-Disciplinary Conference on Memory-Centric Computing (CCMCC)*, pages 1–7. IEEE, 2025. doi:10.1109/CCMCC67628.2025.11380459.

[33] Noor Alia Binti Nor Hashim, Fazrena Azlee Binti Hamid, Julius Teo, and Muhammad Saiful Ariffin Hamid. Analysis of memristor based ring oscillators for hardware security. In *2016 IEEE International Conference on Semiconductor Electronics (ICSE)*, pages 181–184. IEEE, 2016. doi:10.1109/SMELEC.2016.7573621.

[34] Farzad Mozafari and Majid Ahmadi. Design and implementation of a low-power memristor-based piccolo-80 lightweight encryption algorithm using vtm logic gates. *IEEE Canadian Journal of Electrical and Computer Engineering*, 49(1): 25–38, 2025. doi:10.1109/ICJECE.2025.3628528.

[35] Mehri Teimoory, Amirali Amirsoleimani, Arash Ahmadi, Shahpour Alirezaee, Saeideh Salimpour, and Majid Ahmadi. Memristor-based linear feedback shift register based on material implication logic. In *2015 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4. IEEE, 2015. doi:10.1109/ECCTD.2015.7300100.

[36] Seyed Erfan Fatemieh, Mohammad Reza Reshadinezhad, and Nima TaheriNejad. Fast and compact serial imply-based approximate full adders applied in image processing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 13(1):175–188, 2023. doi:10.1109/JETCAS.2023.3241012.

[37] Samane Asgari, Mohammad Reza Reshadinezhad, and Seyed Erfan Fatemieh. Energy-efficient and fast imply-based approximate full adder applying nand gates for image processing. *Computers and Electrical Engineering*, 113:109053, 2024. doi:10.1016/j.compeleceng.2023.109053.

[38] Samane Asgari, Mohammad Reza Reshadinezhad, and Seyed Erfan Fatemieh. Energy-efficient and fast imply-based approximate 4: 2 compressor applying nand gates for image processing. *Journal of Engineering*, 2026(1): 9692096, 2026. doi:10.1155/je/9692096.

[39] Seyed Erfan Fatemieh, Samane Asgari, and Mohammad Reza Reshadinezhad. Fast and low-energy approximate full adder based on felix logic. *Computers and Electrical Engineering*, 136:111220, 2026. doi:10.1016/j.compeleceng.2026.111220.

[40] Lukas Rapp, Leandro Borzyk, Fabian Seiler, Nima Amirafshar, and Nima TaheriNejad. Rapprox: An adaptive approximate adder with look-back for efficient edge ai via memristive in-memory computing. *Electronics*, 15(15):3482, 2026. doi:10.3390/electronics15153482.

[41] Roya Rahimi Disfani, Mojtaba Valinataj, and Nima Taherinejad. An analysis of memristive stateful logics under various operational conditions. *IEEE Access*, 2026. doi:10.1109/ACCESS.2026.3688985.

[42] Roya Rahimi Disfani, Nima TaheriNejad, and Mojtaba Valinataj. Operational conditions analysis for memristive stateful logics-a study on imply and tmsl. In *2022 20th IEEE Interregional NEWCAS Conference (NEWCAS)*, pages 480–484. IEEE, 2022. doi:10.1109/NEWCAS52662.2022.9841969.

[43] David Radakovits and Nima Taherinejad. Behavioral leakage and inter-cycle variability emulator model for rerams (believer). *arXiv preprint arXiv:2103.04179*, 2021. doi:10.48550/arXiv.2103.04179.

[44] Simon Michael Laube and Nima TaheriNejad. Device variability analysis for memristive material implication. *arXiv preprint arXiv:2101.07231*, 2021. doi:10.48550/arXiv.2101.07231.

[45] Bikash Debnath, Jadav Chandra Das, and Debashis De. Design of image steganographic architecture using quantum-dot cellular automata for secure nanocommunication networks. *Nano communication networks*, 15:41–58, 2018. doi:10.1016/j.nancom.2017.11.001.

[46] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)*, 48(4):1–33, 2016. doi:10.1145/2893356.

Author Contributions

Seyed Erfan Fatemieh: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Software, Validation, Visualization, Writing-original draft, Writing-review & editing.

Reza Shahdi Alizadeh: Project Administration, Resources, Supervision, Validation, Writing-review & editing.

Esmail Zarezadeh: Project Administration, Supervision, Validation, Writing-review & editing.

Data Availability Statement

Data is contained within the article.

Competing interests

The authors declare no conflict of interest.

Funding

No funds, grants, or other support was received.



Seyed Erfan Fatemieh was born in Isfahan, Iran, in 1996. He earned his B.Sc. in Computer Engineering and his M.Sc. and Ph.D. in Computer Architecture from the University of Isfahan, Isfahan, Iran, in 2018, 2020, and 2024, respectively. He is currently a Postdoctoral Researcher with the Department of Computer Architecture at the University of Isfahan. His research interests include In-memory Computing, Digital VLSI, Computer Arithmetic, and Cryptography.



Reza Shahdi Alizadeh was born in Amlash, Iran, in 1981. He received his B.Sc. degree in software engineering from the Shahid Sattari University of Aeronautical Engineering, Tehran, and his M.Sc. degree in Information Technology from the Yadegar-e-Imam Khomeini (RAH) Shahre Rey Branch, Islamic Azad University, Tehran, in 2003 and 2025, respectively. His research interests include Blockchain, Artificial Intelligence, and Cryptography.



Esmail Zarezadeh was born in Tehran, Iran, in 1982. He received the B.Sc. degree from the Shahid Sattari University of Aeronautical Engineering, Tehran, Iran, in 2005, and the M.Sc. and Ph.D. degrees in Communications Engineering from the Amirkabir University of Technology, Tehran, Iran, in 2009 and 2022, respectively. He completed a postdoctoral research fellowship at the same institution in 2025. Since 2022, he has been an Associate Professor with the Department of Electrical Engineering at Khatam al-Anbia University, Tehran, Iran. He has taught courses in electrical circuits, microwave engineering, antenna theory, and radar. His research interests include electrical circuit modeling, electromagnetic issues in complex materials, and microwave components.