applied sciences

IMPACT FACTOR 2.5

CITESCORE 5.3

Article

Enhancing Field Multiplication in IoT Nodes with Limited Resources: A Low-Complexity Systolic Array Solution

Atef Ibrahim and Fayez Gebali

Special Issue

Cybersecurity and Cryptography in the Internet of Things (IoT)

Edited by

Dr. Abdelkader Magdy Shaaban, Dr. Oliver Jung and Dr. Dejiu Chen

MDPI

https://doi.org/10.3390/app14104085

applied sciences

MDPI

Article

Enhancing Field Multiplication in IoT Nodes with Limited Resources: A Low-Complexity Systolic Array Solution

Atef Ibrahim [1,2,*] and Fayez Gebali [2]

[1] Computer Engineering Department, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj 16278, Saudi Arabia
[2] Electrical and Computer Engineering Department, University of Victroia, Victoria, BC V8P 5C2, Canada; fayez@ece.uvic.ca or fayez@uvic.ca
* Correspondence: aa.mohamed@psau.edu.sa

Abstract: Security and privacy concerns pose significant obstacles to the widespread adoption of IoT technology. One potential solution to address these concerns is the implementation of cryptographic protocols on resource-constrained IoT edge nodes. However, the limited resources available on these nodes make it challenging to effectively deploy such protocols. In cryptographic systems, finite-field multiplication plays a pivotal role, with its efficiency directly impacting overall performance. To tackle these challenges, we propose an innovative and compact bit-serial systolic layout specifically designed for Montgomery multiplication in the binary-extended field. This novel multiplier structure boasts regular cell architectures and localized communication connections, making it particularly well suited for VLSI implementation. Through a comprehensive complexity analysis, our suggested design demonstrates significant improvements in both area and area–time complexities when compared to existing competitive bit-serial multiplier structures. This makes it an ideal choice for cryptographic systems operating under strict area utilization constraints, such as resource-constrained IoT nodes and tiny embedded devices.

Keywords: modular multiplication algorithms; IoT device security; systolic array processors; parallel processing; hardware-based cryptography

Citation: Ibrahim, A.; Gebali, F. Enhancing Field Multiplication in IoT Nodes with Limited Resources: A Low-Complexity Systolic Array Solution. *Appl. Sci.* **2024**, *14*, 4085. https://doi.org/10.3390/app14104085

Academic Editor: Luis Javier Garcia Villalba

Received: 26 March 2024
Revised: 6 May 2024
Accepted: 7 May 2024
Published: 11 May 2024

1. Introduction and Related Work

The advent of the Internet of Things (IoT) has sparked a paradigm shift in sectors such as healthcare, entertainment, commercial appliances, transportation, agriculture, and housing. This revolutionary technology facilitates the effortless aggregation of data from interconnected devices, paving the way for cloud-based analysis and informed decision-making. Industries have been profoundly transformed as a result, with the IoT acting as a catalyst for innovation and connectivity in the digital era. However, the sensitive nature of IoT data necessitates robust security measures across all tiers of the network. This poses a challenge due to the resource limitations of IoT edge nodes. To address this, extensive research efforts have focused on developing tailored security methodologies for these resource-constrained nodes. Among the diverse cryptographic techniques available, Elliptic Curve Cryptography (ECC) has emerged as a highly optimized solution for IoT edge nodes. Finite-field multiplication is a key building block in the realm of finite-field arithmetic operations, which play a crucial role within ECC algorithms. Beyond its primary function of multiplication, it enables various other fundamental field operations, such as field inversion, field exponentiation, and field division. These operations are essential for the successful implementation of ECC algorithms, enabling efficient and secure cryptographic computations. By harnessing the power of finite-field multiplication, ECC algorithms leverage a comprehensive suite of field operations to deliver resilient and dependable cryptographic solutions. Consequently, there is a growing interest in adopting compact yet highly efficient cryptographic algorithms that can seamlessly operate within the resource

limitations of IoT edge nodes. By embracing such algorithms, IoT networks can ensure the integrity and security of data in diverse applications across industries.

The binary extension field, GF(2^k), plays a pivotal role in numerous crucial applications of computer algebra, including public-key cryptography and error-correcting codes [1–5]. These applications heavily rely on efficient multiplication operations within GF(2^k) [6–8]. Consequently, researchers have extensively explored strategies to minimize the space and delay overhead associated with this computationally expensive operation, which poses challenges in terms of area and delay complexity [9,10]. The literature abounds with discussions and studies dedicated to devising techniques and algorithms that optimize the performance of GF(2^k) multiplication [11–13]. These efforts aim to reduce resource requirements while maintaining high computational efficiency, enabling the seamless integration of Galois field computations into various applications [9–11]. By addressing the space and delay complexities associated with multiplication in GF(2^k), researchers strive to unlock the full potential of public-key cryptography and other computational tasks relying on the Galois field.

The binary extension field, GF(2^k), offers multiple base representations, including the polynomial basis (PB), normal basis (NB), and dual basis (DB), each with its own distinct characteristics and considerations. Among these, the polynomial basis (PB) emerges as a favourable choice due to its simplicity and compatibility with typical hardware constructions, eliminating the need for base conversion. By adopting the polynomial basis, hardware designers can streamline the implementation of GF(2^k) components without compromising efficiency [14]. Multiplication within the finite field GF(2^k) presents inherent complexities that involve two distinct stages: polynomial multiplication and modulo reduction utilizing an irreducible polynomial. The selection of the irreducible polynomial is of utmost importance and offers various options, including trinomials, All-One Polynomials (AOPs), pentanomials, and Equally Spaced Polynomials (ESPs). This choice significantly impacts the efficiency and security of the multiplication operation. Each type of irreducible polynomial possesses unique properties and considerations, influencing factors such as computational performance, implementation complexity, and resistance to cryptographic attacks. The careful evaluation and selection of the appropriate irreducible polynomial are critical in achieving optimal performance and security within GF(2^k) multiplication. While trinomials and pentanomials have been extensively studied, the exploration of AOPs for finite-field multiplication has received comparatively little attention. However, investigating the utilization of AOPs in GF(2^k) computations holds the potential for unlocking new insights and optimizing hardware implementations, offering fresh perspectives for enhancing the performance of Galois field operations.

The literature extensively explores various hardware architectures for implementing polynomial basis (PB) multiplication in GF(2^k), including bit-parallel, bit-serial, and digit-serial multipliers. Among these architectures, bit-parallel multipliers stand out for accomplishing field multiplication in a fixed number of iterations, regardless of the field size m, despite their significant space complexity [15]. On the other hand, systolic bit-parallel multiplier topologies, based on systolic arrays, offer an alternative approach that greatly enhances multiplier throughput at the expense of increased hardware complexity and latency [11–13]. These systolic architectures leverage parallelism and data flow to maximize the efficiency of GF(2^k) multiplication operations, making them particularly attractive for high-throughput applications where performance is of paramount importance [16–19]. However, when selecting the most appropriate multiplier architecture for a specific GF(2^k) implementation, it is vital to carefully consider the trade-off between hardware complexity, latency, and throughput requirements [20–23].

In cryptography applications where resources are limited, bit-parallel architectures are not the most suitable option due to their high hardware requirements. Instead, bit-serial multiplier layouts provide a better trade-off between speed and space, making them well suited for cryptographic applications with constrained space and power limitations [8,24,25]. These architectures offer efficient multiplication while accommodating

tighter resource constraints [9,10,26]. However, it is worth noting that bit-serial architectures lack performance scalability and require a minimum of m time steps to produce the multiplication result [27,28]. On the other hand, digit-serial multiplier architectures offer a slight improvement in hardware resources and performance scalability compared to bit-serial layouts [29–33]. This makes them a better fit for cryptography algorithms with fixed word sizes of data, striking a balance between resource utilization and performance requirements.

This study introduces a novel approach to implementing the Montgomery multiplication algorithm in GF(2^k) using a newly devised bit-serial systolic array configuration. The construction process involves a thorough examination of the dependency graph (DG) derived from the multiplication method, followed by the careful selection of scheduling and projection vectors. By assigning appropriate timing and mapping the DG nodes to appropriate processing elements (PEs), a bit-serial systolic array construction is formed. Notably, the proposed bit-serial multiplier exhibits a remarkable reduction in both area and area–time complexity compared to existing constructions documented in the previous literature. This significant reduction in complexity makes it an excellent choice for cryptographic techniques that operate within limited spatial constraints. Overall, this innovative construction presents an efficient solution for conducting Montgomery multiplication in GF(2^k) within resource-constrained cryptographic scenarios.

This paper is organized into distinct sections, each contributing to the overall understanding of the research. In Section 2, a concise elucidation of the adopted Montgomery multiplication algorithm within the context of $GF(2^k)$ is presented [20,21]. Section 3 delves into the exploration and analysis of DGs. Section 4 delves into an intricate examination of the bit-serial systolic array multiplier structure, encompassing its comprehensive design and insightful implementation considerations. To comprehensively evaluate the proposed approach, Section 5 conducts a thorough analysis of the recommended multiplier layout, meticulously assessing the spatial and temporal complexity of the observed effective bit-serial architectures. Finally, Section 6 serves as a fitting conclusion to the paper, concisely summarizing the key findings and significant contributions of this study.

2. Mathematical Basis of Montgomery Modular Multiplication over Binary Extension Field

The binary extension field, GF(2^k), can be constructed using an irreducible polynomial F of the form $F = \sum_{j=0}^{k} f_j \cdot F^j$, where $0 \leq j \leq k-1$ and $f_k = f_0 = 1$. The addition of polynomials in GF(2^k) is straightforward and involves performing bitwise exclusive-OR (XOR) operations [20,25]. However, the modular multiplication of polynomials within GF(2^k) presents additional complexities. This is because the intermediate results obtained during multiplication require a subsequent step of modular reduction using $F^k = \sum_{j=0}^{k-1} f_i \cdot F^i$ to ensure that the resulting polynomial remains within GF(2^k) and has a degree lower than k [25]. By applying this reduction, the integrity of the binary extension field is preserved, enabling accurate and valid multiplication operations to be carried out effectively within GF(2^k).

Let us consider σ and v as two elements in the binary extension field GF(2^k), and we aim to perform Montgomery Modular Multiplication (MMM) on them [20,21]. We assume the existence of a special element ϑ that satisfies the condition $gcd(\vartheta, F) = 1$. The Montgomery residues of σ and v are denoted by C and D, respectively. The computation $P = CD\vartheta^{-1} \bmod F = v\vartheta \bmod F = \sum_{j=0}^{k-1} p_j \cdot F^j$ represents the MMM operation between the polynomials $C = \sigma\vartheta \bmod F = \sum_{j=0}^{k-1} c_j \cdot F^j$ and $D = v\,\vartheta \bmod F = \sum_{j=0}^{k-1} d_j \cdot F^j$ [20,21]. To obtain the ultimate outcome, U, the deployment of Montgomery multiplication with the inputs P and 1 becomes paramount. This transformative operation culminates in the equation $U = P\vartheta^{-1} \bmod F = \sigma\, v \bmod F$. The utilization of Montgomery multiplication holds a distinct advantage, especially in a diverse range of applications characterized by repetitive multiplications. Notably, in the domains of elliptic curve point multiplication, inversion, and exponentiation, this technique assumes a central role. Its ability to seamlessly

handle pre- and post-transformations sets it apart, rendering it a powerful tool for achieving efficient and accurate results.

By selecting $\vartheta = F^{(k-1)/2}$, the multiplication $P = CD\vartheta^{-1} \bmod F$ can consistently be expressed in this form [21].

$$P = C(d_0 + d_1F + \cdots + d_{(k-1)/2}F^{(k-1)/2} + \cdots + d_{k-1}F^{k-1})F^{-(k-1)/2} \bmod F \quad (1)$$

Equation (1) can be alternatively presented in the following manner [21]:

$$\begin{aligned} P =& C(d_{(k-1)/2} + d_{(k+1)/2}F^1 + \cdots + d_{(k-1)}F^{(k-1)/2}) \\ &+ C(d_0F^{-(k-1)/2} + d_1F^{-(k-3)/2} + \\ &\cdots + d^{-1}_{(k-3)/2}) \bmod F \end{aligned} \quad (2)$$

It is worth highlighting that, in the realm of practical applications, the common practice is to utilize odd values for k. Therefore, when constructing the multiplier, the focus will primarily be on catering to the requirements and considerations specific to odd k values.

Equation (2) can be split into two separate polynomials, denoted by A and B, which can be expressed as follows. The sum of A and B should construct Equation (2) [21].

$$A = Cd_{(k-1)}F^{(k-1)/2} + Cd_{(k-2)}F^{(k-3)/2} + \cdots + Cd_{(k+1)/2}F^1 + Cd_{(k-1)/2} \bmod F \quad (3)$$

$$B = Cd_0F^{-(k-1)/2} + Cd_1F^{-(k-3)/2} + \cdots + Cd_{(k-5)/2}F^{-2} + Cd_{(k-3)/2}F^{-1} \bmod F \quad (4)$$

To derive the recursive forms of Equations (3) and (4), we can rearrange them into the following organized manner [21]:

$$\begin{aligned} A =& (\cdots((Cd_{(k-1)})F \bmod F + Cd_{(k-2)})F \bmod F + \cdots + \\ & Cd_{(k+1)/2})F \bmod F + Cd_{(k-1)/2} \end{aligned} \quad (5)$$

$$\begin{aligned} B =& (\cdots(((Cd_0)F^{-1} \bmod F + Cd_1)F^{-1} \bmod F + \cdots + \\ & Cd_{(k-5)/2}F^{-1} \bmod F + Cd_{(k-3)/2})F^{-1} \bmod F \end{aligned} \quad (6)$$

Let us denote the results of the ith iteration of Equations (5) and (6) by A_i and B_i, respectively. These values can be computed recursively based on the outcomes of the previous $(i-1)$th iteration. Specifically, for the ith step, where $1 \leq i \leq \frac{k+1}{2}$, the recursive version of Equation (5) can be stated in the following manner:

$$A_i = A_{i-1}F \bmod F + Cd_{(k-i)} \quad (7)$$

with $A_0 = 0$.

Equation (6) can be recursively formulated in a similar manner to Equation (5). The recursive form for Equation (6) at step i can be expressed as

$$B_i = B_{i-1}F^{-1} \bmod F + Cd_{(i-1)} \quad (8)$$

with $B_0 = d_{(k-1)/2} = 0$.

It is crucial to keep in mind that $d_{(k-1)/2} = 0$ is necessary in order to calculate the final product, $B_{(k+1)/2}$. It is also possible to determine A_i and B_i concurrently because they do not share any data. Equations (7) and (8) provide an illustration of this concurrent calculation. The expansion of F^k in Equation (7) can be changed to obtain the reduced form

of A_i for $1 \leq i \leq \frac{k+1}{2}$ at the bit level. Therefore, A_i can be written in the following manner in terms of its bit-level representation [21]:

$$\begin{aligned} A_i =& a_{k-2}^{i-1} F^{k-1} + \cdots + a_1^{i-1} F^2 + a_0^{i-1} F + \\ & a_{k-1}^{i-1} (f_{k-1} F^{k-1} + \cdots + f_1 F + f_0) + \\ & d_{k-i} (c_{k-1} F^{k-1} + \cdots + c_1 F + c_0) \end{aligned} \tag{9}$$

The expression for the recursive representation of A at step i should be

$$a_{k-1-j}^{i} = a_{k-2-j}^{i-1} + a_{k-1}^{i-1} f_{k-1-j} + d_{k-i} c_{k-1-j} \tag{10}$$

with $a_j^0 = a_{-1}^{i-1} = 0$ and $0 \leq j \leq k-1$.

By multiplying each side of equation F by F^{-1}, we can obtain $F^{-1} = \sum_{j=1}^{k} f_j F^{j-1}$. This manipulation is valid because F is a root of F, and for any irreducible polynomial, the coefficients f_0 and f_k are both equal to 1.

B_i can be expressed in a similar manner to Equation (9) by substituting the expansion of F^{-1} from Equation (8). The revised representation of B_i should be as follows [21]:

$$\begin{aligned} B_i =& b_{k-1}^{i-1} F^{k-2} + \cdots + b_1^{i-1} + b_0^{i-1} (f_k F^{k-1} + \cdots + \\ & f_2 F + f_1) + d_{i-1} (c_{k-1} F^{k-1} + \cdots + c_1 F + c_0) \end{aligned} \tag{11}$$

Once again, the recursive representation of B at step i can be written as follows:

$$b_j^i = b_{j+1}^{i-1} + b_0^{i-1} f_{j+1} + d_{i-1} c_j \tag{12}$$

with $b_j^0 = b_k^{i-1} = d_{(k-1)/2} = 0$ for $0 \leq j \leq k-1$.

To combine $A^{(k+1)/2}$ and $B^{(k+1)/2}$ to obtain the intended outcome P, a certain number of m two-input XOR gates are required. These XOR gates are utilized to carry out bitwise exclusive-OR operations between $A^{(k+1)/2}$ and $B^{(k+1)/2}$ corresponding bits. The bits of the final outcome P will be produced by the output of these XOR gates.

3. Dependency Graph

Within the Montgomery multiplication algorithm, the iterative section encompasses two recursive expressions, defined in Equations (10) and (12). These equations define the computational steps involved in the algorithm, highlighting its iterative nature. Although the computation structures of Equations (10) and (12) are fundamentally similar, they diverge in terms of the order in which input and output elements are processed.

To visualize the interdependencies within the algorithm, Figures 1 and 2 present the derived dependency graphs (DGs) specifically designed for a field size of $k = 5$. These DGs provide a graphical representation of the relationships between different elements within the algorithm, displayed in a two-dimensional integer domain $\mathbb{D}$ with indices i and j. By examining these graphs, one can gain insights into the flow of computations and the connections between various nodes. The set of nodes (squares) present within the DGs execute the iterative Formulas (10) and (12). Notably, the number of these nodes is determined by the formula $k \times (k+1)/2$. Each node performs specific calculations and contributes to the overall computation process, facilitating the execution of the Montgomery multiplication algorithm.

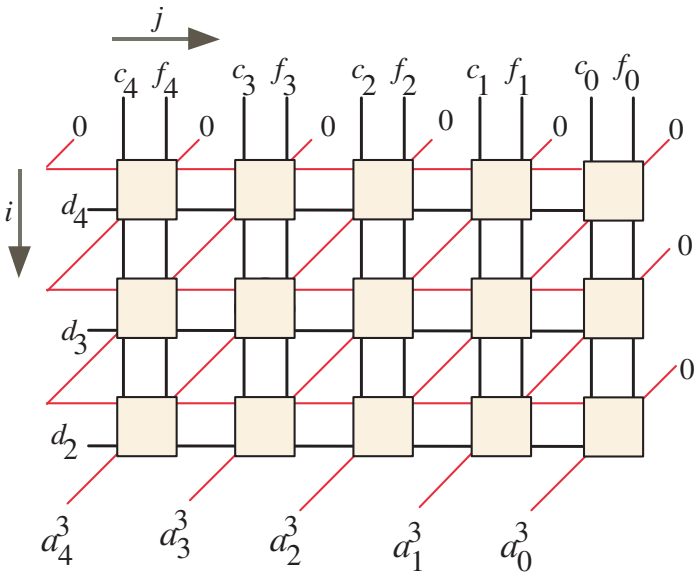

Figure 1. DG of recursive Formula (10) for $k = 5$.

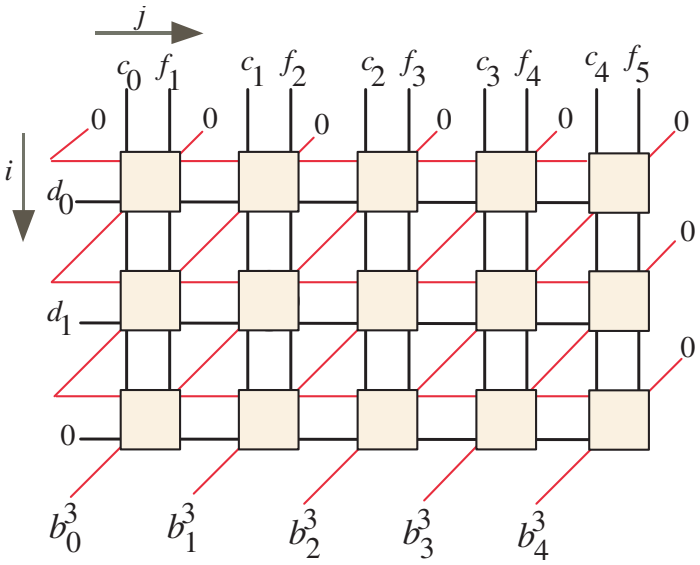

Figure 2. DG of recursive Formula (12) for $k = 5$.

The Montgomery multiplication algorithm employs a well-defined input strategy to ensure proper computation. During the initial stages, the input signals d_{k-i} and d_i are directed from left to right, following a consistent and orderly pattern. This left-to-right orientation facilitates the systematic entry of these signals, where i ranges from 1 to $(k+1)/2$. Simultaneously, at the topmost section of the dependency graphs (DGs), the input signals c_{k-1-j} and c_j are introduced, establishing a distinct entry point. Expanding further on the input procedure, additional signals, namely, f_{k-1-j} and f_{j+1}, are sequentially incorporated from the top of the DGs. These signals join the computational flow, contributing to the iterative process of the algorithm. To distinguish specific input parameters, visual cues in the form of red slanted lines are strategically placed at the right corners of the input nodes. These lines symbolize the insertion of input signals with initial values of a^0_{k-2-j} and b^0_{j+1}, both initialized to zero. Additionally, we have applied a red-coloured highlight to distinguish the intermediate partial products of the A and B signals. This visual distinction aids in easily identifying and understanding the computations involved.

Within each node of the dependency graphs, intermediate partial products of the A and B coefficients are calculated. These intermediate results serve as crucial elements in the

overall computation, as they are subsequently transmitted to the nodes in the subsequent row. Through this iterative progression, the algorithm systematically processes the partial products, gradually building towards the final outcome. Ultimately, the final result, denoted by P, is derived by summing the last coefficients of $A^{(k+1)/2}$ and $B^{(k+1)/2}$ using XOR gates.

4. The Development of the Bit-Serial Systolic Array

To establish the bit-serial systolic array layout, it is necessary to carefully determine the suitable scheduling and projection vectors for the dependency graphs (DGs). Following the method outlined in previous works [13,34,35], we can select the scheduling vector, denoted by $\mathbf{S}$, and the projection vector, denoted by $\mathbf{P}$, that will effectively guide the design of the serial systolic structure [36,37]. Specifically, the recommended vectors are $\mathbf{S} = [2 \ -1]$ and $\mathbf{P} = [1\ 0]^T$. Through the application of these vectors to the individual DG nodes $\mathbf{p}(i, j)$, we are able to derive the associated scheduling function, referred to as $t(\mathbf{p})$, in addition to the projection function, denoted by $PE(\mathbf{p})$, as illustrated below. These functions play crucial roles in assigning time values to each DG node and mapping them to the appropriate PE within the systolic array [13,34,35]. Through this integration of scheduling and projection functions, the efficient coordination and execution of computations are achieved within the serial systolic array framework [36,37].

$$t(\mathbf{p}) = 2i - j \tag{13}$$

$$PE(\mathbf{p}) = i \tag{14}$$

The application of the scheduling function $t(\mathbf{p})$ on the DGs yields the node timing depicted in Figures 3 and 4. These timing values demonstrate the sequential progression of input application and the subsequent generation of DG products. In a step-by-step manner, the input bits c_{k-1-j}, f_{k-1-j}, c_j, f_{j+1}, a^0_{k-1-j}, and b^0_{j+1} are successively applied, where $0 \leq j \leq k-1$. As a result, the corresponding output bits $a^{(k+1)/2}_{k-1-j}$ and $b^{(k+1)/2}_{j}$ are generated. Notably, output A is computed from the most significant bit (MSB), $a^{(k+1)/2}_{k-1}$, at time $k-1$, and progresses to the least significant bit (LSB), $a^{(k+1)/2}_{0}$, at time $2k-2$. Similarly, output B is computed from the LSB, $b^{(k+1)/2}_{0}$, at time $k-1$ and progresses to the MSB, $b^{(k+1)/2}_{k-1}$, at time $2k-2$.

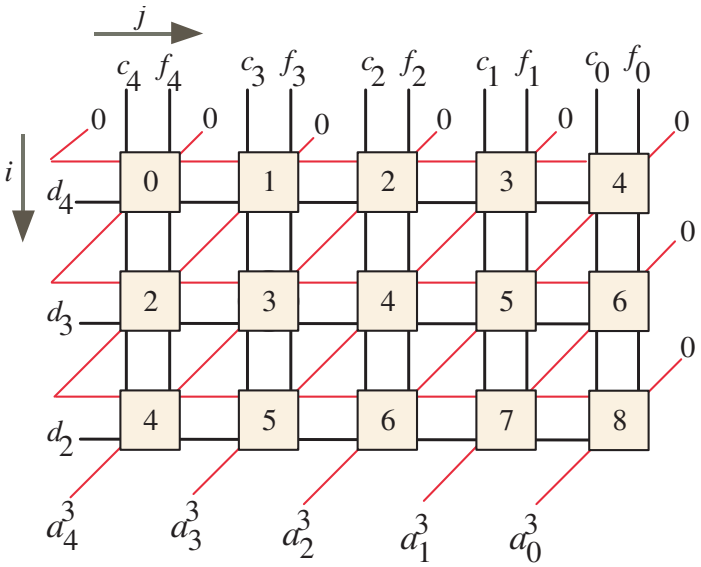

Figure 3. Node timing for $k = 5$.

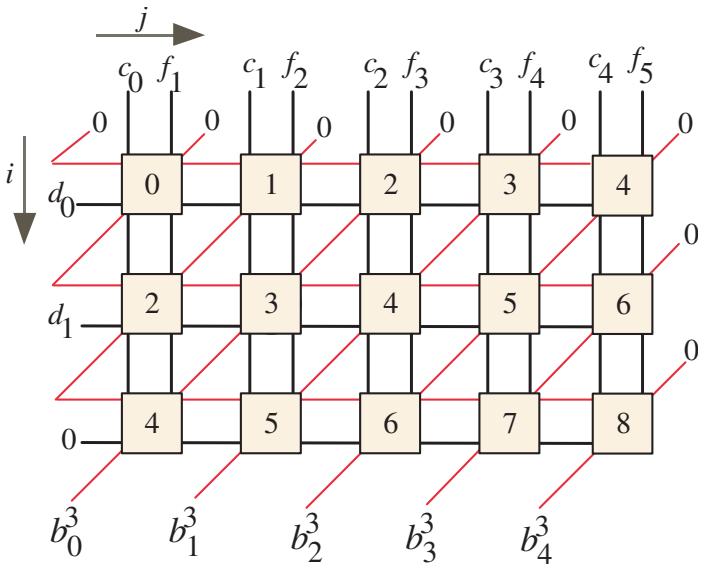

Figure 4. Node timing for $k = 5$.

Figure 5 visually depicts the extracted layout of a serial-in/serial-out (SISO) systolic array, which is achieved by applying the projection function $PE(\mathbf{p})$ to the dependency graphs (DGs). The structure consists of two identical processor arrays, one for the serial computation of coefficients of A and the other one for the serial computation of coefficients of B. The upper processor array produces the coefficients of output A, $a_{k-1-j}^{(k+1)/2}$, in a serial fashion and stores them in the upper shift register shown in Figure 5. The lower processor array produces the coefficients of output B, $b_j^{(k+1)/2}$, $0 \leq j \leq k-1$, concurrently with the coefficients of A and stores them in the lower shift register shown in Figure 5. Notice that bits a_{k-1}^i and b_0^i are held in a latch inside the PEs for the proper computation of coefficients A and B. Therefore, they have a separated serial bus from the remaining coefficient bits, as indicated in Figure 5. Also, the resulting bits $a_{k-1}^{(k+1)/2}$ and $b_0^{(k+1)/2}$ are stored in a separate latch from the output shift registers to avoid the disordering of the resulting coefficient bits. In order to obtain the final product P, the corresponding bits of coefficients A and B are XORed, as illustrated in Figure 5. The processor structure comprises $k+1$ processing elements (PEs) and operates over a span of $2k-2$ clock cycles to generate the ultimate output P. Features of the A and B PE logic are shown in Figures 6 and 7.

Upon closer examination of the SISO systolic arrangement, it becomes apparent that the input signals d_{k-i} and d_{i-1} are assigned to their respective PEs. Additionally, the intermediate signals, including c_{k-1-j}, f_{k-1-j}, a_{k-1}^i, a_{k-1-j}^i, c_j, f_{j+1}, b_0^i, and b_j^i, are efficiently pipelined between adjacent PEs. Notably, the tri-state buffers in Figures 6 and 7 are controlled by the signal t to ensure that the most significant bit (MSB) of A, a_{k-1}^i, and the least significant bit (LSB) of B, b_0^i, remain accessible throughout the subsequent clock cycles in the succeeding PE. Furthermore, to maintain proper timing, the control signal t is carefully propagated through two D-Latches before being applied to the next PE. This meticulous handling guarantees accurate synchronization within the systolic array.

Importantly, prior to the operation of the SISO systolic layout, it is imperative to clear all D-Latches to establish an appropriate initial state. This preparatory step ensures the reliable and consistent functioning of the structure.

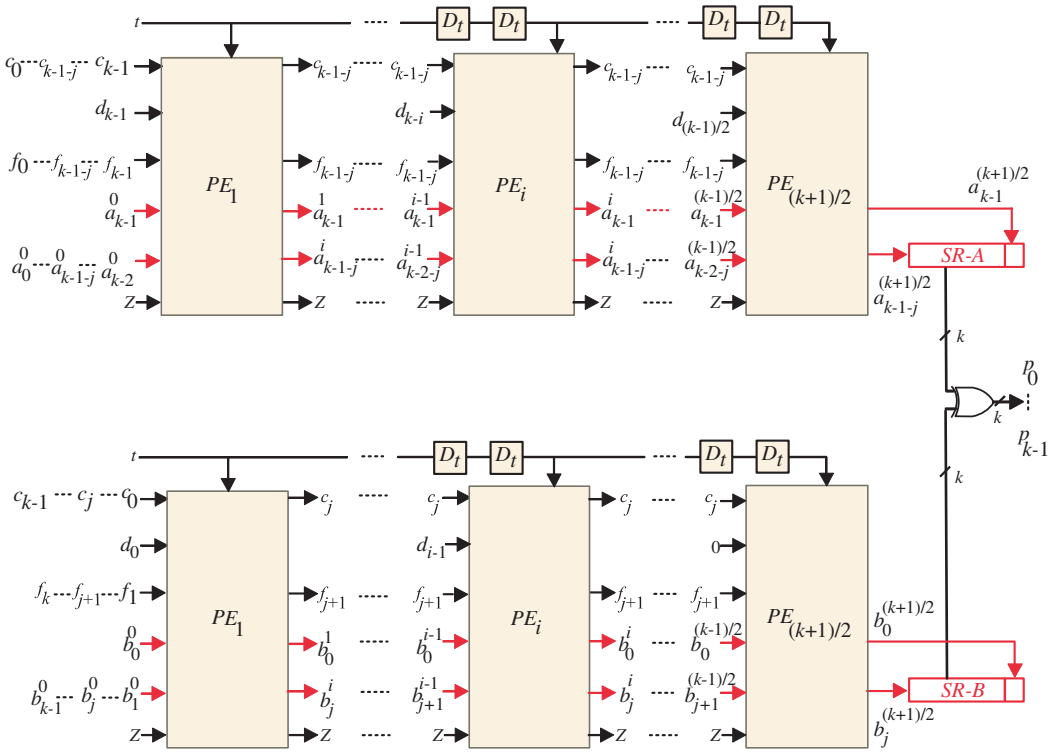

Figure 5. Suggested SISO systolic multiplier.

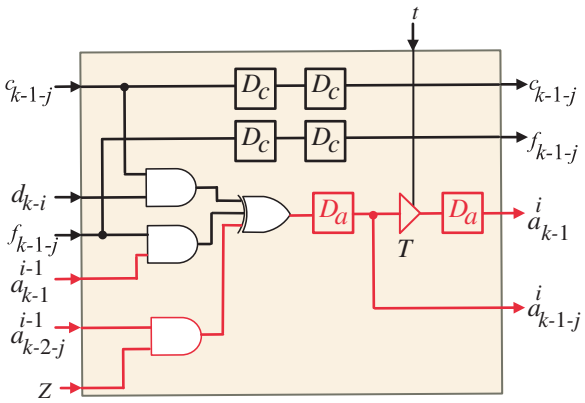

Figure 6. The PE_i logic circuit for the A systolic array. The square blocks represent D-Latches.

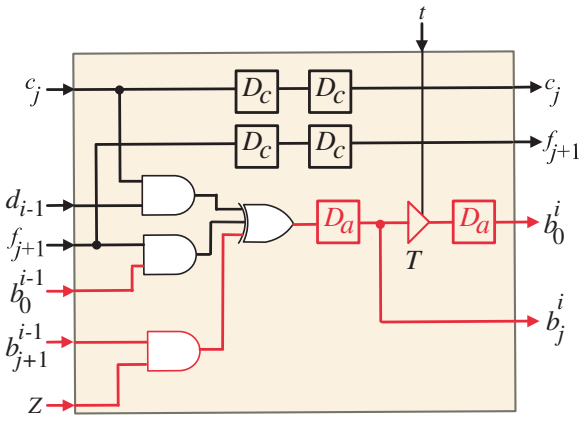

Figure 7. The PE_i logic circuit for the B systolic array. The square blocks represent D-Latches.

The functionality of the SISO systolic layout can be described as follows:

a. Initially, at time instance $t = 0$, the first processing element (PE1) of the upper systolic array receives input signals d_{k-1}, c_{k-1}, f_{k-1}, a^0_{k-1}, and a^0_{k-2} to compute the intermediate signal a^1_{k-1} using the AND-XOR construction depicted in Figure 6. Simultaneously, during the same time instance, the first PE (PE1) of the lower systolic array takes input signals d_0, c_0, f_1, b^0_0, and b^0_1 to compute the intermediate signal b^1_0 using the AND-XOR layout displayed in Figure 7. By the end of this time instance, the tri-state buffers in both systolic arrays are enabled to store the computed values of a^1_{k-1} and b^1_0.

b. At time instance $t = 1$, input signals d_{k-1}, c_{k-2}, f_{k-2}, a^0_{k-1}, and a^0_{k-3} are applied to the first PE (PE_1) of the upper systolic array to compute the intermediate signal a^1_{k-2} utilizing the AND-XOR construction shown in Figure 6. Also, at the same time instance, input signals d_0, c_1, f_2, b^0_0, and b^0_2 are applied to the first PE (PE_1) of the lower systolic array to compute the intermediate signal b^1_1 utilizing the AND-XOR configuration shown in Figure 7. Through this time instance, tri-state buffers in both systolic arrays are disabled to prevent writing the updated values a^1_{k-2} and b^1_1 on the previous values of a^0_{k-1} and b^0_0, respectively. At the end of this time instance, signals a^1_{k-1}, a^1_{k-2}, b^1_0, and b^1_1 will be available at the inputs of the second PE (PE_2).

c. As the second and third time instances ($t = 2$ and $t = 3$) occur, the second PE (PE2) in both systolic arrays continues its normal operation, similar to PE1, by preserving the most significant bit (MSB) of A (a^2_{k-1}) and the least significant bit (LSB) of B (b^2_0). It also updates the signals a^2_{k-2} and b^2_1 accordingly.

d. In the general case, at time instances $t = 2i$, where $2 \leq i \leq \frac{k+1}{2}$, the processing elements (PEi) function similarly to PE1 and PE2 to sustain the most significant bits (MSBs) of A (a^i_{k-1}) and the least significant bits (LSBs) of B (b^i_0). They also update the signals a^i_{k-2} and b^i_1. During the remaining time instances, all the processing elements work concurrently to compute the intermediate coefficients a^i_{k-1-j} and b^i_j, where $2 \leq j \leq k-1$.

e. At time instances $t = k + 2i - 1$, $0 \leq i \leq (k+1)/2$, control signal Z should be deactivated ($Z = 0$) to force zero values at the input of the lower AND gate shown in Figures 6 and 7. These zero values represent the initial zero values of a^{i-1}_{-1} and b^{i-1}_{-1} shown at the right edges of the DGs depicted in Figures 1 and 2.

f. During time instances $t = 2i$, where $0 \leq i \leq \frac{k+1}{2}$, it is necessary to activate the control signal t ($t = 1$). This activation is crucial in preserving the most significant bits (MSBs) of A as a^i_{k-1}, and the least significant bits (LSBs) of B as b^i_0. However, during the remaining time instances, the control signal t should be deactivated ($t = 0$). This deactivation prevents overwriting the previously stored values of a^i_{k-j-2} and b^i_{j+1}, where $0 \leq j \leq k-1$, on the respective stored values of a^i_{k-1} and b^i_0.

g. At time instance $2k - 2$, the shift registers depicted at the output of systolic arrays shown in Figure 5 will have all the resulting coefficient bits of A and B variables. The corresponding bits of A and B coefficients are applied to two-input XOR gates (m two-input XOR gates) to obtain the final product bits of p_j, $0 \leq j \leq k-1$, as indicated in Figure 5.

5. Results and Discussion

5.1. Comparison of Multiplier Structures

In this section, we conduct a comprehensive comparison between the recommended SISO systolic multiplier layout and previously published rival constructions, which include bit-serial systolic and non-systolic multiplier designs [8–10,13,24,26,38]. The evaluation encompasses several crucial factors, such as total gate counts, latency, and critical path delay (CPD), providing valuable insights into the performance and efficiency of each architecture. The results of this comparison, presented in Table 1, offer a clear understanding of

the strengths and weaknesses associated with different approaches. Within the table, the symbols Π_{tri}, Π_A, Π_X, and Π_M represent the latencies of specific logic components, including tri-state buffers, two-input AND gates, two-input XOR gates, and two-to-one MUXs, respectively. By thoroughly examining these metrics, designers and researchers can make well-informed decisions regarding the selection of the most suitable multiplier architecture.

Table 1. Comparing area and delay complexity: conventional vs. bit-serial multiplier architectures.

Design	Type	TSB	AND	XOR	MUX	Latches	Latency	CPD
[8]	Systolic	0	$3k$	$3k$	$3k$	$14k$	$3k$	$\Pi_A + \Pi_X + \Pi_M$
[24]	Systolic	0	$4k-1$	$2k-1$	k	$12k-7$	$3k-2$	$\Pi_A + \Pi_X$
[26]	Systolic	0	$3k$	$3k$	$3k$	$14k$	$3k-1$	$\Pi_A + T_X + \Pi_M$
[38]	Non-Systolic	0	$R1$ [(1)]	$R2$ [(1)]	0	$R3$ [(1)]	k	$\Pi_A + (2 + \lceil \log_2(k) \rceil)\Pi_X$
[9]	Non-Systolic	0	$R4$ [(2)]	$R5$ [(2)]	0	$R6$ [(2)]	k	$\Pi_A + \lceil \log_2(k) \rceil \Pi_X$
[10]	Semi-Systolic	0	$6y$ [(3)]	$6y+2$	$2y$	$27y-3$	$3y+k+1$	$\Pi_A + \Pi_X + \Pi_M$
[13]	Systolic	$2k$	$3k$	$3k$	k	$9k$	$3k-2$	$\Pi_A + \Pi_X + \Pi_M$
Proposed	Systolic	k	$3k+3$	$3k+2$	0	$8k-2$	$2k-2$	$\Pi_A + 2\Pi_X$

[(1)] $R1 = 5k+3$, $R2 = 5k+2q-4$, $R3 = 19k+4q-4$, where q denotes the exponent of the second term in the trinomial expression $(x^k + x^q + 1)$. [(2)] $R4 = 7k+6$, $R5 = 7k+6$, $R6 = 18k-2q-2$, [(3)] $y = \lceil k/2 \rceil$.

Before delving into the analysis of the area and delay complexity data presented in Table 1, it is worth noting an interesting observation. Among the compared multiplier layouts, the majority adhere to a systolic or semi-systolic design, whereas the multiplier constructions discussed in [9,38] stand out as non-systolic architectures. It is worth noting that the architectures proposed in [9,38] are built upon two different types of irreducible polynomials: trinomials and ϕ-nomials, which are irreducible polynomials characterized by having ϕ non-zero terms. Among these two types, trinomial-based solutions exhibit superior performance compared to ϕ-nomial solutions. Therefore, for the purpose of comparison, we choose to focus on the trinomial-based multiplier construction and evaluate it against the recommended SISO systolic multiplier layout.

The generated multiplier layout showcases several advantageous characteristics compared to existing multiplier structures. In terms of tri-state buffers, the presented multiplier construction requires only k tri-state buffers, whereas the Ibrahim multiplier structure [13] necessitates $2k$ tri-state buffers, and all other multiplier layouts do not require any tri-state buffers at all. Also, it is worth highlighting that the recommended multiplier layout significantly reduces the number of latches compared to all other multiplier structures.

In terms of gate counts, when compared to the Song multiplier [8], the proposed multiplier layout has similar numbers of AND and XOR gates but eliminates the need for the $3k$ MUXes present in the Song multiplier. Similarly, when compared to the Fenn multiplier structure [24], the proposed layout has a lower count of AND gates but a higher count of XOR gates while having the lowest number of MUXes between the two designs.

Compared to the multiplier structure proposed by Choi [26], the proposed design exhibits nearly the same number of AND and XOR gates but significantly reduces the number of required MUXes. Although both the multiplier structures of Masoleh [9,38] and the proposed layout have zero MUXes, the Masoleh structures feature a substantial number of AND and XOR gates, in contrast to the proposed design.

Lastly, when compared to the Ibrahim multiplier structures [10,13], the proposed layout demonstrates a similar count of AND and XOR gates while completely eliminating the need for the k MUXes required by Ibrahim's designs. These comparisons highlight the favourable characteristics of the proposed multiplier layout, such as a lower latch count and competitive gate counts in relation to existing multiplier structures.

When considering time complexity, it is evident that the non-systolic multiplier designs mentioned in [9,38] exhibit significantly reduced latency compared to all the compared designs, including the recommended one. However, they also have a considerably higher CPD.

In comparison to the Song design [8], the proposed multiplier layout has notably lower latency and almost the same CPD (assuming the delay of the MUX is similar to that of the XOR gate). The Fenn multiplier [24] has a higher latency but a smaller CPD with the delay of one XOR gate compared to the recommended design. In the case of Ibrahim's multiplier structures [10,13], they exhibit significantly larger latency but almost the same CPD as the proposed design.

Given the variations in gate counts, latency, and CPD among the different designs, relying solely on analytical analysis to estimate the required area and computation time for each design may be challenging. Therefore, it is crucial to validate the conducted qualitative analysis through real implementations and performance measurements.

5.2. Comparison of Time–Space Complexity

To gauge the effectiveness of the compared structures, a thorough evaluation was conducted, encompassing key performance indicators such as overall area (A), total processing time (T), and the area–time product (AT). This evaluation was carried out using the NanGate Open Cell Library, specifically designed for a cutting-edge 15nm process technology and operating at a voltage of 0.8V. The assessment focused on the values of $k = 233$ and $q = 74$. To determine the area of each fundamental component (referred to as S), a comprehensive gate count was performed, utilizing the two-input NAND gate as the reference point for size comparison. The gate count provides an estimation of the area occupied by each component within the design. Additionally, the delay of the crucial logic components (Π) was evaluated individually. The delay represents the time taken for a signal to propagate through a particular component. This assessment allows for understanding the performance characteristics of each component in terms of speed and latency.

The assessment conducted for each component provides the following findings:

1. Tri-state buffer: The area of a tri-state buffer (S_{tri}) is assessed to be 0.8, and the delay (Π_{tri}) is evaluated to be 7.9 ps.
2. Two-input AND gate: The area of a two-input AND gate (S_A) is calculated as 1.2, and the delay (Π_A) is estimated to be 11.3 ps.
3. Two-input XOR gate: The area of a two-input XOR gate (S_X) is determined to be 2.5, and the delay (Π_X) is evaluated to be 12.7 ps.
4. Two-to-one MUX: The area of a two-to-one MUX (S_M) is estimated to be 2.5, and the delay (Π_M) is calculated to be 12.4 ps.
5. D-Latch: The area of a D-Latch (S_{Latch}) is determined to be 2.8, and the delay (Π_{Latch}) is evaluated to be 16.6 ps.

Based on the results presented in Table 2, it is evident that the proposed SISO systolic array multiplier architecture exhibits significant advantages over recently reported efficient bit-serial multiplier layouts, particularly in terms of area (A) and the area–time (AT) product. The average improvement in area is at least 16.4%, while the average improvement in the area–time product is at least 33.9%. These findings strongly support the suitability of the suggested architecture for IoT applications, where space complexity is a critical concern due to limited resources. By offering superior performance in terms of resource utilization, the SISO systolic array multiplier addresses the stringent resource constraints of IoT applications, making it an ideal choice for implementing cryptosystems in such scenarios.

Table 2. Evaluating space and time trade-offs in serial constructions for $k = 233$ and $q = 74$.

Design	A [*Kgates*]	T [*ns*]	AT	%A	%AT
[8]	18.0	25.1	451.8	43.4	55.8
[24]	17.3	21.9	378.9	41.0	47.2
[26]	18.0	25.0	450.0	43.3	55.6
[38]	42.0	32.0	1344.0	75.7	85.1
[9]	41.0	26.0	1066.0	75.1	81.2
[10]	16.0	21.0	336.0	36.3	40.5
[13]	12.6	24.8	302.6	16.4	33.9
Proposed	10.2	19.6	199.9	-	-

6. Summary and Conclusions

This research paper introduces a new and innovative SISO systolic array design specifically tailored for performing Montgomery multiplication over the binary extension field. The proposed architecture brings forth a range of advantageous features that make it highly suitable for VLSI implementations. One key advantage lies in its regular processing element builds, which not only simplify the overall design but also facilitate efficient integration within VLSI systems. Moreover, the architecture incorporates localized interconnections, further enhancing its feasibility for VLSI implementation. What sets this design apart from existing multiplier architectures is its ability to significantly reduce area and delay overhead. This aspect is of utmost importance, as it optimizes resource utilization, making it particularly appealing for implementation in cryptosystems. In the context of IoT applications, where space complexity is a critical concern due to limited resources, this proposed architecture demonstrates its effectiveness. By offering improved efficiency in terms of space utilization, it effectively addresses the specific challenges faced by IoT applications, positioning itself as a highly promising solution for implementing efficient and reliable cryptosystems in the IoT domain. Moving forward, the main area of concentration for future research will be the practical hardware implementation of the proposed solution. By doing so, we can gain valuable insights into the feasibility and performance of the proposed solution in real-world scenarios.

Author Contributions: Conceptualization, A.I.; methodology, A.I. and F.G.; software, A.I.; validation, A.I.; formal analysis, A.I.; investigation, A.I.; resources, A.I.; data curation, A.I.; writing—original draft preparation, A.I.; writing—review and editing, A.I. and F.G.; visualization, A.I.; supervision, A.I.; project administration, A.I.; funding acquisition, A.I. All authors have read and agreed to the published version of the manuscript.

Funding: Prince Sattam bin Abdulaziz University provided funding for this research work through project number (PSAU/2023/01/26738).

Institutional Review Board Statement: Not Applicable.

Informed Consent Statement: Not Applicable.

Data Availability Statement: The datasets presented in this article are not readily available because the data are part of an ongoing study. Requests to access the datasets should be directed to the corresponding author.

Acknowledgments: The authors extend their appreciation to Prince Sattam bin Abdulaziz University for funding this research work through project number (PSAU/2023/01/26738).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things
GF	Galois field
PE	Processing element
ECC	Elliptic Curve Cryptography
DG	Dependency graph
AT	Area–time
AOP	All-One Polynomial
PB	Polynomial basis
NB	Normal basis
DB	Dual basis
ESP	Equally Spaced Polynomials
MMM	Montgomery Modular Multiplication
VLSI	Very Large-Scale Integrated Circuit
SISO	Serial-in/serial-out
CPD	Critical path delay

References

1. Blahut, R.E.; Katz, J.; van Oorschot, P.C.; Vanstone, S.A. *Handbook of Applied Cryptography*; CRC Press: Boca Raton, FL, USA, 1996.
2. Blahut, R.E. *Theory and Practice of Error Control Codes*; Addison-Wesley: Reading, MA, USA, 1983.
3. Pöpper, C. Applied Cryptography and Network Security. In Proceedings of the 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab Emirates, 5–8 March 2024; Proceedings, Part I; Springer Nature: Berlin/Heidelberg, Germany, 2024.
4. Tang, Q. Public-Key Cryptography–PKC 2024. In Proceedings of the 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, 15–17 April 2024; Proceedings, Part II; Springer Nature: Berlin/Heidelberg, Germany, 2024.
5. Seedorf, J.; Mazhar, K.; Schwabe, F.; Omerovic, I. Applied cryptography in the internet-of-things. In *Online-Labs in Education*; Nomos Verlagsgesellschaft mbH & Co. KG: Baden, Germany, 2022; pp. 361–374.
6. Haa, J.-C.; Moon, S.-J. A common-multiplicand method to the montgomery algorithm for speeding up exponentiation. *Inf. Process. Lett.* **1998**, *66*, 105–107. [CrossRef]
7. Lee, K.-J.; Yoo, K.-Y. Linear systolic multiplier/squarer for fast exponentiation. *Inf. Process. Lett.* **2000**, *76*, 105–111. [CrossRef]
8. Song, L.; Parhi, K.K. Efficient finite field serial/parallel multiplication. In Proceedings of the IEEE 1996 International Conference on Application-Specific Architectures and Processors, Chicago, IL, USA, 19–23 August 1996; pp. 72–82.
9. Abdulrahman, E.A.; Reyhani-Masoleh, A.h. High-speed hybrid-double multiplication architectures using new serial-out bit-level mastrovito multipliers. *IEEE Trans. Comput.* **2016**, *65*, 1734–1747. [CrossRef]
10. Ibrahim, A. Novel bit-serial semi-systolic array structure for simultaneously computing field multiplication and squaring. *IEICE Electron. Express* **2019**, *16*, 20190600. [CrossRef]
11. Kim, K.W.; Lee, J.D. Efficient unified semi-systolic arrays for multiplication and squaring over gf(2^m). *IEICE Electron. Express* **2017**, *14*, 1–10. [CrossRef]
12. Kim, K.W.; Kim, S.H. Efficient bit-parallel systolic architecture for multiplication and squaring over gf(2^m). *IEICE Electron. Express* **2018**, *15*, 1–6. [CrossRef]
13. Ibrahim, A. Efficient parallel and serial systolic structures for multiplication and squaring over gf(2^m). *Can. J. Electr. Comput. Eng.* **2019**, *42*, 114–120. [CrossRef]
14. Hsu, I.S.; Truong, T.K.; Deutsch, L.J.; Reed, I. A comparison of vlsi architecture of finite field multipliers using dual, normal, or standard bases. *IEEE Trans. Comput.* **1988**, *37*, 735–739. [CrossRef]
15. Wu, H. Bit-parallel finite field multiplier and squarer using polynomial basis. *IEEE Trans. Comput.* **2002**, *51*, 750–758.
16. Lee, C.-Y. Low-latency bit-pararallel systolic multiplier for irreducible $x^m + x^n + 1$ with $GCD(m, n) = 1$. *IEICE Trans. Fund. Elect. Commun. Comp. Sci.* **2008**, *55*, 828–837.
17. Ting, Y.R.; Lu, E.H.; Lu, Y.C. Ringed bit-parallel systolic multipliers over a class of fields gf(2^m). *Integration* **2005**, *38*, 371–384. [CrossRef]
18. Fournaris, A.P.; Koufopavlou, O. Versatile multiplier architectures in gf(2^k) fields using the montgomery multiplication algorithm. *Integration* **2008**, *41*, 571–578. [CrossRef]
19. Kim, K.-W.; Lee, H.-H.; Kim, S.-H. Efficient combined algorithm for multiplication and squaring for fast exponentiation over finite fields gf(2^m). In *Proceedings of the 7th International Conference on Emerging Databases, LNEE 461*; Springer: Singapore, 2017; pp. 50–57.
20. Lee, K. Resource and delay efficient polynomial multiplier over finite fields gf (2^m). *J. Korea Soc. Digit. Ind. Inf. Manag.* **2020**, *16*, 1–9.

21. Lee, K. Low complexity systolic Montgomery multiplication over finite fields gf (2^m). *J. Korea Soc. Digit. Ind. Inf. Manag.* **2022**, *18*, 1–9.
22. Imaña, J.L.; Piñuel, L.; Kuo, Y.-M.; Ruano, O.; García-Herrero, F. Efficient low-latency multiplication architecture for nist trinomials with risc-v integration. In *IEEE Transactions on Circuits and Systems II: Express Briefs*; IEEE: New York, NY, USA, 2024.
23. Zhao, S.; Hu, D.; Liu, Z.; Yu, B.; Huang, H.; Ma, C. High-performance unified modular multiplication algorithm and hardware architecture over gf(2^m). *Integration* **2024**, *96*, 102124. [CrossRef]
24. Fenn, S.T.J.; Taylor, D.; Benaissa, M. A dual basis bit serial systolic multiplier for gf(2^m). *Integration* **1995**, *18*, 139–149. [CrossRef]
25. Kim, K.W.; Jeon, J.C. A semi-systolic montgomery multiplier over gf(2^m). *IEICE Electron. Express* **2015**, *12*, 1–6. [CrossRef]
26. Choi, S.; Lee, K. Efficient systolic modular multiplier/squarer for fast exponentiation over gf(2^m). *IEICE Electron. Express* **2015**, *12*, 1–6. [CrossRef]
27. Kitsos, P.; Theodoridis, G.; Koufopavlou, O. An efficient reconfigurable multiplier architecture for galois field gf(2^m). *Microelectron. J.* **2003**, *34*, 975–980. [CrossRef]
28. Selimis, G.N.; Fournaris, A.P.; Michail, H.E.; Koufopavlou, O. Improved throughput bit-serial multiplier for gf(2^m) fields. *Integration* **2009**, *42*, 371–384. [CrossRef]
29. Guo, J.-H.; Wang, C.-L. Digit-serial systolic multiplier for finite fields gf(2^m). *IEEE Proc. Comput. Digital Tech.* **1998**, *145*, 143–148. [CrossRef]
30. Hutter, M.; Grobschald, J.; Kamenje, G.-A. A versatile and scalable digit-serial/parallel multiplier architecture for finite fields gf(2^m). In Proceedings of the 2003 4th International Conference on InformationTechnology: Coding and Computing (ITTCC2003), Las Vegas, NV, USA, 28–30 April 2003; pp. 692–700.
31. Song, L.; Parhi, K.K. Low-energy digit serial/parallel finite field multipliers. *J. Vlsi Signal Process. Syst.* **1998**, *19*, 149–166. [CrossRef]
32. Kim, C.H.; Hong, C.P.; Kwon, S. A digit-serial multiplier for finite field $GF(2^m)$. *IEEE Trans. Very Large Scale Integr. (VLSI) Sys.* **2005**, *13*, 476–483.
33. Hariri, A.; Reyhani-Masoleh, A. Digit-serial structures for the shifted polynomial basis multiplication over binary extension fields. In Proceedings of the LNCS Intl Workshop Arithmetic of Finite Fields (WAIFI), Siena, Italy, 6–9 July 2008; pp. 103–116.
34. Gebali, F. *Algorithms and Parallel Computers*; John Wiley: New York, NY, USA, 2011.
35. Ibrahim, A.; Elsimary, H.; Gebali, F. New systolic array architecture for finite field division. *IEICE Electron. Express* **2018**, *15*, 1–11. [CrossRef]
36. Ibrahim, A. Scalable digit-serial processor array architecture for finite field division. *Microelectron. J.* **2019**, *85*, 83–91. [CrossRef]
37. Ibrahim, A. Low-space bit-serial systolic array architecture for interleaved multiplication over gf (2^m). *IET Comput. Digit. Tech.* **2021**, *15*, 223–229. [CrossRef]
38. Reyhani-Masoleh, A. A new bit-serial architecture for field multiplication using polynomial bases. In Proceedings of the 7th International Workshop Cryptographic Hardware Embedded Systems (CHES 2008), Washington, DC, USA, 10–13 August 2008; pp. 314–330.