International Journal of Safety and Security Engineering
Vol. 16, No. 5, May, 2026, pp. 1131-1145
Journal homepage: http://iieta.org/journals/ijsse

Multi-Level Optimization Strategies for FPGA-Based MixColumn and Linear Diffusion Layers: A Survey

Safa Hazim Alwan[1*], Yasir Amer Abbas[2], Mudhafar Hussien Ali[1]

[1] Department of Computer Engineering, AL-Iraqia University, Baghdad 10053, Iraq
[2] College of Engineering for Artificial Intelligence Technology, University of Diyala, Diyala 32001, Iraq

Corresponding Author Email: safahazim@uodiyala.edu.iq

https://doi.org/10.18280/ijsse.160518

Received: 22 February 2026
Revised: 30 April 2026
Accepted: 25 May 2026
Available online: 31 May 2026

Keywords:
MixColumn, FPGA, lightweight cryptography, diffusion layer, MDS matrix, Galois field, LUT mapping

ABSTRACT

The deployment of lightweight cryptography in resource-constrained platforms increasingly relies on FPGA-based accelerators that can balance performance, area, and energy under strict implementation constraints. In several lightweight permutations, the linear diffusion layer-implemented as MixColumn or diffusion-equivalent transformations-often represents a primary hardware cost center due to the finite-field constant multiplications, XOR-network depth, and routing/LUT-packing overhead. This review evaluates and generates FPGA-oriented diffusion-layer implementations across lightweight families, focusing on micro-architectural and platform-specific design decisions. The review assesses documented implementations using a standardized set of extraction fields, encompassing FPGA device family and tool flow, resource utilization, timing performance (Fmax), latency and cycle counts, and throughput computation basis. Where applicable, values are normalized to enhance comparability across papers, and absent metrics are clearly indicated as not reported. In addition, the survey organizes prior works into a taxonomy spanning arithmetic-level optimizations, architecture-level organizations, and platform-level considerations. Finally, the survey identifies recurring reporting limitations and open research gaps particularly inconsistent benchmarking assumptions and incomplete post-implementation evaluation-and outlines actionable directions for designing more efficient and comparable diffusion-layer architectures for lightweight cryptography on FPGA.

1. INTRODUCTION

In the realm of digital platforms with limited resources, such as IoT devices, embedded systems, RFID devices and security modules, the need for efficient cryptography has become a necessity. These platforms pose significant challenges, which must be balanced with the demand for reliable and high-throughput security mechanisms. As a result, researchers have been exploring cryptographic designs that are optimized for hardware, as well as implementations on FPGAs that carefully analyze trade-offs between area, frequency, and throughput [1, 2].

Diffusion layers implemented in various designs of FPGAs recently have drawn more attention from researchers and developers. those devices have limited resources, e.g. batteries, memory or small computing capabilities. Hence cryptographic algorithms which are widely used for secure data communication need to be adapted. It is essential to implement efficient cryptographic algorithms which consume low power and are able to process information in real time, in spite of hardware limitations. In order to prepare effective, scalable and practical solutions for secure communication, the focus is put on the efficiency improvement of MixColumn and diffusion layers.

One particular transformation that consistently proves MixColumn operation is or (diffusion equivalent linear layer) stand out as a bottleneck in lightweight primitive. Its crucial component in many cryptographic protocols. This operation involves a complex matrix multiplication in GF (2^8) that can result in XOR logic networks, shift, and conditional reduction factor that directly influence logic depth, routing complexity, and achievable clock frequency. Such as, optimizing the MixColumn operation has become a primary focus for many researchers, who seek to improve its performance without simply translating its mathematical specification into hardware description language (HDL) code. they aim to create more efficient and effective cryptographic implementations that can thrive in resource-constrained environments. The search for MixColumn optimization has resulted in multiple new methods which concentrate on making GF (2^8) arithmetic operations more efficient [3]. Research indicates that custom hardware implementations of MixColumn and Inverse MixColumn functions which perform Galois-field multiplication operations reveal how multiplier design structures impact FPGA resource usage and complete implementation costs. Scientists have achieved promising results through their research of MixColumn modification approaches by using Vedic multiplier designs and different

multiplication techniques to enhance delay-performance ratios [2, 4].

The MixColumn diffusion layer appears in AES but similar structures exist in lightweight cryptographic families and AES-like permutations which use compact MDS matrices and nibble-based mixing and structured linear layers for diffusion. FPGA implementations of these lightweight diffusion structures have demonstrated that linear diffusion often plays a critical role in determining implementation viability and performance [5]. The creation of lightweight hash functions which use AES-like diffusion processes has led to the development of platform-dependent implementations for the diffusion layer [6].

The main obstacle for implementing lightweight block ciphers on FPGAs stems from the fact that the diffusion layer requires most of the available resources. The examination of LED implementations shows that designers must make specific choices which affect the operational feasibility of lightweight diffusion when using these platforms for remote keyless entry systems and other applications [7]. The comparison of different studies about FPGAs using lightweight primitives such as LED and SIMECK requires standardized reporting parameters [8].

The total FPGA implementation expenses depend on three main factors which include the cipher structure, datapath design and diffusion scheduling approach of mCrypton and other lightweight ciphers [9, 10].

Although a lot of research has been recently done on stream ciphers, most of currently available surveys and literature reviews focus on lightweight cryptography algorithms in general or on their security applications on FPGAs, but do not dive deep enough into the details of optimizations. Finally, challenges encountered during benchmarking, normalization and reproducibility in the scope of FPGA implementations of considered diffusion layers are also not sufficiently discussed.

We present a comprehensive survey of speed optimizations of MixColumn layer and in general of any diffusion–equivalent linear layer. The surveyed works propose optimizations divided into three layers of optimizations: from the arithmetic-level to the architecture-level and to the platform-level, each representing a different abstraction layer of possible optimizations. In addition, FPGA implementation tradeoffs from area, speed, delay, netlength, and routing density viewpoints are discussed, while also highlighting device-dependent speed of instruction execution.

The process of implementing MixColumn and diffusion-equivalent linear layers for lightweight cryptography on FPGAs needs thorough examination. The main objective of this work involves establishing an equal basis for evaluating various design approaches which will help determine the best method to perform MixColumn transformations on FPGAs. Furthermore, this research aims to develop guidelines which help designers create improved lightweight cryptography solutions that use dependable data to produce easy-to-understand performance metrics [11, 12].

1.1 Motivation and major contribution

The internet of things (IoT) and edge computing have experienced rapid development which requires immediate solutions for secure cryptographic hardware that operates efficiently and in small form factors. The research community selects FPGA platforms for lightweight cryptography because these platforms provide flexible operation and fast development capabilities [13].

The linear diffusion layer creates a major barrier which blocks the achievement of maximum performance according to research findings. The component which exists in AES, LED, PHOTON, and SKINNY ciphers operates as a major performance limitation. The system slowdown results from three main factors which include finite-field constant multiplication difficulties and deep XOR network complexities and restrictions caused by routing congestion and LUT-packing pressure. The combination of these elements produces major problems which impact both the frequency closure and area efficiency of modern FPGAs [3-14].

The research community started working on different optimization solutions to address these problems since 2019. The research has used three methods to solve the problem which involve basic arithmetic operations and LUT-based system design and column-based data processing and pipeline-based system architecture. The methods have achieved significant performance enhancements during particular testing scenarios but their results remain scattered because of variations between FPGA families and synthesis toolflows and performance measurement systems. The various research studies about diffusion-layer design create difficulties when trying to determine how their design decisions impact the complete system operation [6-15].

The major contributions of our survey are:

1. Classifying FPGA diffusion-layer optimization methods into arithmetic-level, LUT-based, and architectural approaches.
2. Analyzing the impact of these methods on area, timing, routing complexity, and hardware trade-offs.
3. Providing a normalized framework for comparing FPGA implementation metrics across different studies.
4. Identifying research gaps and future directions for FPGA-based lightweight cryptographic implementations.

1.2 Research question and review objectives

The survey contains particular research questions which serve as analytical tools to conduct evaluation as shown in Table 1. The main objective of these investigations as shown in Figure 1 is a detailed analysis of FPGA-based MixColumn and diffusion-layer implementations to study their architectural, arithmetic, and platform-dependent design aspects. The research questions receive their corresponding benchmarks which include area utilization, throughput, latency and technology mapping strategies to establish direct evidence links with the complete evaluation system. The survey also includes the classification of FPGA-based diffusion-layer optimization techniques, the evaluation of hardware trade-offs, and the identification of benchmarking limitations and future research directions. In addition, the survey connects architectural analysis with FPGA implementation challenges in lightweight cryptographic systems.

The article organization is as follows: Section 2 highlights the background of MixColumn and diffusion-equivalent transformations. Section 3 explains the methodology used in the review and describes the method in detail to ensure the correctness and trustworthiness of the findings. Section 4 clarifies the taxonomy of FPGA MixColumn design techniques, while Section 5 presents a comparative evaluation of different resources.

Table 1. Specific research questions for a survey literature review

No.	Question	Rationale
RQ1	Which optimization strategies are most effective for reducing the hardware complexity of FPGA-based MixColumn and diffusion-layer implementations?	To investigate how arithmetic-level optimization methods reduce hardware complexity, and implementation cost in FPGA-based diffusion layers.
RQ2	How do FPGA architecture styles influence the balance between throughput, latency, and area efficiency in diffusion-layer implementations?	To analyze how different architectural organizations influence the trade-off between throughput, latency, and area efficiency in FPGA implementations.
RQ3	How do FPGA platform characteristics and mapping techniques impact the performance and resource utilization of diffusion-layer architectures?	To examine how FPGA platform characteristics, routing resources, and synthesis tool flows affect implementation efficiency.

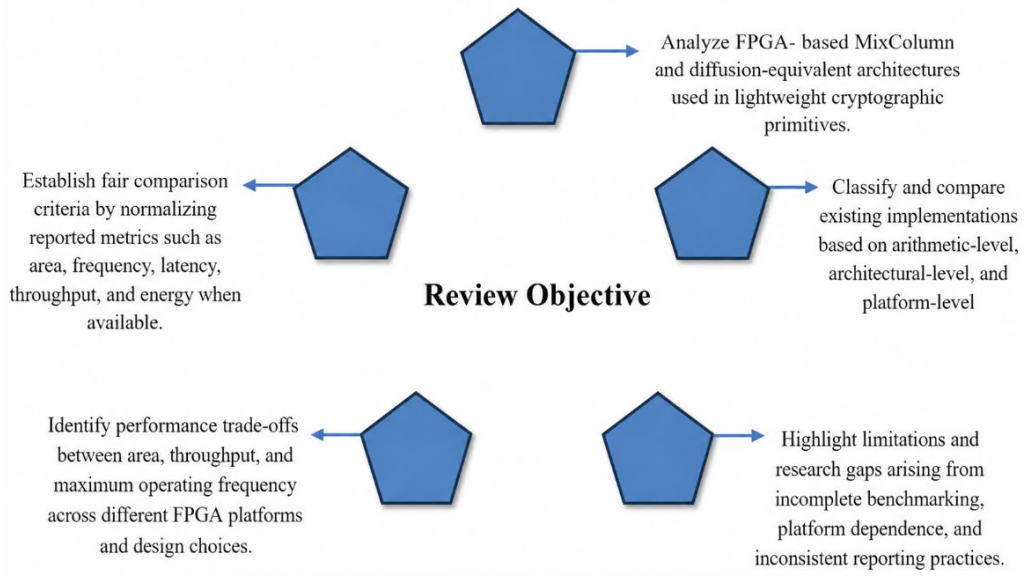

Figure 1. Framework of the review objectives for analyzing and classifying FPGA-based MixColumn and diffusion-layer optimization techniques

2. BACKGROUND

2.1 Lightweight cryptography and diffusion layers

The expanding deployment of resource-constrained devices including RFID tags and smart cards and low-power IoT devices requires cryptographic systems which can function under restricted space, memory and power consumption limitations. Existing encryption systems are often too resource-intensive, making them impractical for these devices. The development of lightweight cryptography has become a specific research area which Concentrates on creating small efficient cryptographic systems which operate well in environments with limited resources [16].

Scientists have studied lightweight block cipher operations on FPGAs but these designs encounter major problems when it comes to their operational speed and their ability to access system resources. The security of these lightweight cryptographic primitives depends on how their nonlinear substitution operations work together with their linear diffusion mechanisms. The encryption process achieves fast distribution of changes through its combination of S-box layers which introduce nonlinearity and linear diffusion layers which spread this nonlinearity throughout the system.

Good diffusion attempts to spread a single bit of input difference over the entire internal state of the cipher. This limits the effectiveness of both differential and linear cryptanalysis. In substitution-permutation networks, such as found in both AES and many of the lightweight ciphers, the MixColumn transformation is where much of this diffusion takes place. The goal of this transformation is to ensure that a single bit difference in the input plaintext is propagated to affect a large fraction of the state, ideally within a small number of rounds [17].

2.2 MixColumns as an FPGA hardware bottleneck

The core operation of AES depends on MixColumns which performs a linear transformation through finite field matrix multiplication. However, translating this mathematical concept into a hardware implementation is no straightforward task. The design process needs this essential element because it determines the complexity of XOR networks and routing systems and their logic depth which affects both system speed and power consumption. Research teams focus on MixColumns optimization because any reduction in logic depth and resource usage will produce major system performance and power efficiency improvements. The MixColumns equivalent in lightweight systems exists as either an MDS diffusion matrix or a structured linear layer.

Researchers have found that the linear layer can become a significant bottleneck due to the computational demands of diffusion. The problem requires two fundamental solutions which designers have developed through serial diffusion processing and round-based network structures. These approaches yield different results, with serialization reducing

area requirements but round-based designs boosting operational efficiency [17].

In the literature MixColumns acceleration is achieved using a mix of micro-architectural reconfiguration and FPGA-native implementation:

(1) Granularity of the Datapath and serialization.

Design that reduces datapath width decrease area but this comes at the cost of a higher cycle count. The LED architecture serves as an example which uses a 4-bit wide datapath that operates through serialization to perform processes by defining separate states which include diffusion state. The method demonstrates that hardware expenses decrease substantially through datapath serialization but increase latency.

(2) Variants of diffusion-matrix calculation (iterative versus direct).

The round-based LED implementations provide multiple options to calculate diffusion through three different methods which include base matrix iteration and squared variant addition and full diffusion matrix direct implementation. The different methods [17] produce distinct throughput/area trade-offs. The system generates identical output during MixColumns operations because different decomposition methods produce equivalent linear transformations which affect both the critical path duration and the hardware component dimensions.

(3) Exploitation of FPGA primitives and LUT-centric mapping.

In the quest for optimized FPGA designs, a striking example emerges in the form of Xilinx's shift-register LUT primitives, such as the SRL16. The research community uses these fundamental building blocks to develop small and power-efficient systems which implement LED and PHOTON functionality [8].

2.3 FPGA design trade-offs and optimization considerations

The survey of FPGA cryptography requires researchers to evaluate how survey results depend on both the cryptographic algorithm design and the particular FPGA platform and development environment. The evaluation of FPGAs in research studies focuses on three essential performance metrics which consist of resource utilization and top operating speed and processing speed and sporadic power consumption evaluation. The comparison of these metrics between different research studies becomes difficult because different studies used different FPGA devices and packaging types and speed grades and synthesis restrictions and tooling systems.

The current studies about lightweight block ciphers demonstrate that performance results depend strongly on the system design and data processing system structure. The selection of appropriate datapath components stands as a vital design factor which affects both FPGA system resource consumption and the achieved performance-speed balance [18, 19].

3. METHODOLOGY

The survey conducts a complete and fair evaluation through duplicable research design which studies all MixColumn and diffusion-equivalent implementation methods for lightweight cryptography on FPGAs. The paper follows IEEE survey guidelines to create a synthesis of different approaches instead of performing a meta-analysis which would unite results from different system implementations according to the study [20]. Review methodology process shown in Figure 2.

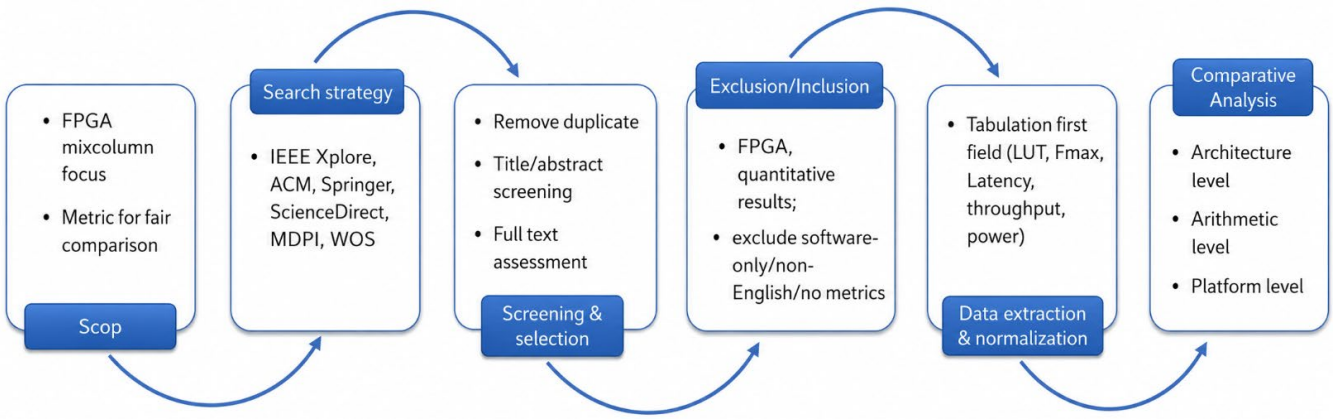

Figure 2. Review methodology for selecting, filtering, and analyzing FPGA-based MixColumn implementations

3.1 Scope of the review

The survey examines MixColumn transformation optimization methods for FPGA systems which focus on achieving maximum efficiency. The comparison process requires us to choose studies which show quantitative performance results and provide full information about their architectural and implementation details. The objective needs organizations to create uniform performance metrics which track resource utilization and speed and latency and throughput and power consumption to assess their implementation success. The research team wants to study how various design choices between iterative and unrolled data paths and pipeline depth and LUT mapping affect the tradeoff between area consumption and throughput performance and maximum operating frequency [21].

3.2 Literature search strategy

The review process will receive complete evaluation through searches which examine all major scientific databases including IEEE Xplore and ACM Digital Library and SpringerLink and ScienceDirect and MDPI Journals. The search process includes Scopus and Web of Science databases to reduce database bias while increasing the search results' scope. The search queries contain suitable keywords which

Boolean operators apply according to current survey best practices to retrieve both general search outcomes and specific search results. The literature search was conducted using combinations of keywords related to FPGA-based diffusion-layer optimization and lightweight cryptography. The main search terms included:

- "MixColumn FPGA implementation"
- "diffusion layer optimization FPGA"
- "lightweight cryptography FPGA"
- "MDS matrix FPGA"
- "GF multiplication optimization"
- "AES MixColumns hardware implementation"
- "FPGA lightweight cipher architecture"

Boolean operators such as AND and OR were used to refine the search results across scientific databases. The review mainly focused on studies published between 2018 and 2026 in order to capture both foundational and recent FPGA-oriented diffusion-layer optimization techniques.

3.3 Screening and study selection

Candidate records were filtered using a staged screening approach: (i) duplicate removal, (ii) title/abstract screening, and (iii) full-text assessment. This progressive filtering adheres review methodologies, guaranteeing that research included for synthesis correspond properly to the review scope and provide extractable evidence appropriate for cross-paper comparison [21, 22].

3.4 Inclusion and exclusion criteria

Studies were included if they:

1. Addressed FPGA implementations of cryptographic primitives or components where MixColumn/diffusion is part of the Datapath (AES-like or lightweight diffusion-equivalent)
2. Provided quantitative results supporting comparison analysis (e.g., LUT, slice, FF, BRAM use, Fmax, throughput, latency, or power/energy consumption),
3. Provided a detailed description of the architectural approach, including Datapath width.

Studies were-exclusive if they be:

1. absent empirical implementation outcomes (no quantified/synthesized metrics),
2. excluded from consideration (lacking MixColumn/diffusion-equivalent layer),
3. Non-English content was subjected to an English-only filter for uniformity [21, 22].

3.5 Data extraction and normalization

- The research team used a standardized method to collect data which they then organized into a single framework for complete outcome comparison [21].
- The extracted data contains essential information about the specific algorithm or primitive used which includes MixColumns or MDS-based and the diffusion method.
- The documentation includes information about architectural design choices between iterative and pipelined and serialized approaches as well as datapath width and parallelism degree.
- The platform details which include FPGA family and device and tool version need evaluation together with synthesis and place-and-route status.
- The system tracks both area metrics which include LUTs and slices and FFs and BRAM and DSPs and timing and performance metrics which include maximum frequency and latency and throughput.
- The data includes power and energy consumption information when such information becomes available.
- The study documents all its identified strengths together with its limitations and gaps which include missing timing information and absent power analysis results [21].

To enhance comparability, absent values were designated as NR (Not Reported). Throughput values were recalculated when adequate information was provided (e.g.,Throughput = (block size × F_max)/cycles/block), otherwise the initially stated throughput was preserved and annotated [23].

3.5.1 Normalization assumptions and cross-study comparison

Comparing results from the different studies reported in the literature is not straightforward because the tests were run on different platforms. While each study describes the particular FPGA family and synthesis tool(s) (and in one case an even custom architecture) used, in this survey common implementation metrics are compared and presented in a normalized fashion wherever possible. Area-related figures-of-merit such as the number of LUTs, slices or flip-flops are provided first, followed by the throughput, normalized to MPC wherever possible, using block size, clock frequency and number of cycles as parameters. Although both designs achieve a supported maximum frequency of 110 MHz and similar throughput, results on other FPGAs, available routing resources and synthesis constraints may differ. In this survey, we do not aim to compare numbers, but rather gain insights in relative design patterns and trade-offs for a variety of architectures and different design implementations on heterogeneous FPGA platforms.

3.6 Synthesis, categorization, and comparative

The researchers developed a complete taxonomy system to eliminate duplicate information from extracted papers through their analysis. The framework organizes implementation methods based on three essential parameters which include GF multiplication methods, constant matrix optimization, shared logic approaches and arithmetic-level methods. The framework includes three main architectural approaches which are iterative, pipelined, unrolled, sub-pipelined designs and serialized diffusion. The framework considers platform-specific requirements which include device family characteristics and LUT and BRAM usage and tool flow limitations. The taxonomy enables researchers to perform a comparative synthesis which includes standardized tables and a cross-paper analysis of how area and throughput and maximum frequency interact with each other [21].

3.7 Bias control and reproducibility measures

Our research team dedicated itself to reducing bias while ensuring maximum reliability of results through equal application of strict study evaluation methods to all investigated studies. The review process appears in full detail through our transparent methods which show all stages from database searches to keyword selection and from initial screening to final inclusion numbers in a comprehensive methodology and when appropriate using a PRISMA-inspired

flowchart to demonstrate the process of record selection and filtering which follows established guidelines [20, 21].

4. TAXONOMY OF FPGA MIXCOLUMN DESIGN TECHNIQUES

The MixColumn layer operates with reduced efficiency because it's hardware requirements include expensive finite-field multiplications and complex XOR networks and routing systems. Research teams use three fundamental methods to enhance MixColumn performance through operation simplification which reduces multiplier complexity and shortens logic paths and datapath redesign for achieving both performance and area efficiency and platform-level optimization for FPGA-specific advantages. Figure 3 illustrates the proposed taxonomy used to classify FPGA-based MixColumn and diffusion-layer optimization strategies. The taxonomy divides prior studies into arithmetic-level, architectural-level, and platform-level optimization categories, enabling clearer analysis of how different design strategies influence area utilization, throughput, timing performance, and implementation efficiency.

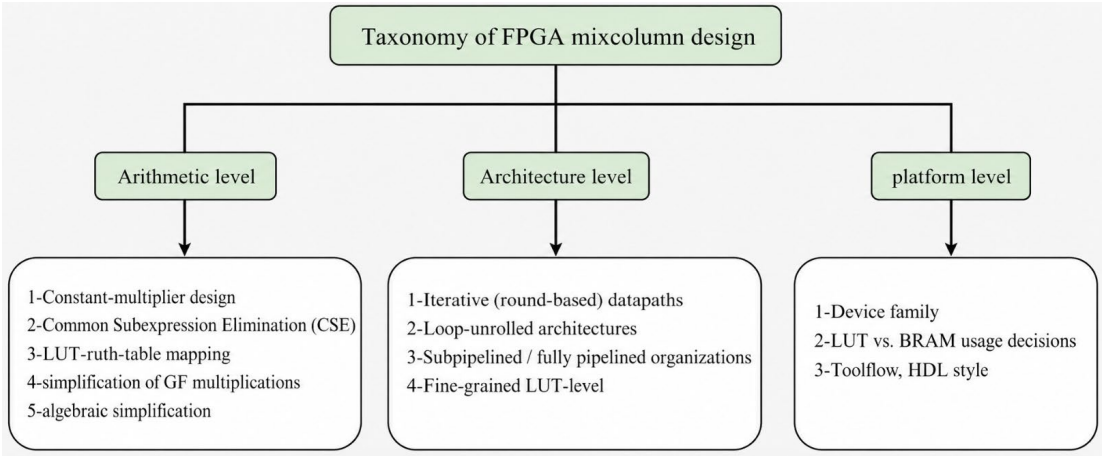

Figure 3. Taxonomy of FPGA-based MixColumn optimization techniques classified into arithmetic, architecture, and platform levels

4.1 Arithmetic-level techniques (GF multiplication variants, LUT/logic sharing)

(1) Constant-multiplier design and "xtime"-based formulations.

The MixColumn multiplication process needs shift-and-conditional-reduction networks which use specific constants to establish both the number of XOR operations and the critical path. This is particularly relevant in AES-like MixColumn transformations, where the choice of constant multiplier significantly impacts the complexity of the XOR operations and the depth of the critical path [24]. As a result, the timing closure of the design is directly affected when the XOR network becomes a performance bottleneck, with some constants proving more efficient than others [3, 25].

(2) Common Subexpression Elimination (CSE) and shared intermediate signals.

The process of arithmetic operations produces identical intermediate results which get used multiple times for different output bytes between {02} and {03}. The design approach of CSE effectively detects and removes duplicate logic which enables it to use existing signals to decrease the requirement for lookup tables and slices. The optimization method enables improved resource management which results in a favorable reduction of system latency according to reference [23].

(3) LUT-truth-table mapping (Boolean decomposition to FPGA-native LUTs).

Rather than directly generating matrix-multiplication circuitry, numerous studies generate each output bit as a Boolean function of the input bits and implement it using FPGA LUT primitives (e.g., 6-input LUTs). The system converts standard gate-level operations into LUT memory access operations which results in shorter logic paths. This method is especially appealing when outputs can be obtained by a single LUT-level evaluation stage, efficiently aiming for a one-cycle combinational implementation of MixColumn under an appropriate mapping [3].

(4) Automated algebraic simplification and logic-sharing via optimization scripts.

To optimize LUT utilization, some architectures tackle the LUT mapping process as a combinatorial optimization problem. The process requires output function analysis to detect recurring input patterns which get organized into efficient LUT mapping subsets. The optimization process uses MATLAB-based pipelines to eliminate unnecessary components while finding shared elements which results in a simplified design before HDL implementation according to [3].

(5) Replacing expensive GF multiplications with simplified operators.

The MixColumn component in lightweight construction designs such as PHOTON and LED systems needs strong computational resources to perform multiplication operations which leads to higher system expenses. The problem needs designers to create new architectural solutions which implement comparator-based selection methods to achieve simplicity through the removal of complicated multiplication operations. The new system designs work to accomplish two essential goals which include minimizing area and power consumption while enhancing throughput and efficiency. The MixColumn operation becomes more efficient through a method which uses a progenerated multiplication result table that allows fast lookup with a basic comparison function to identify the needed output [5].

4.2 Architecture-level techniques

(1) Iterative (round-based) Datapath.

The MixColumn core of iterative design approaches uses space-efficient operations which span multiple rounds to minimize system size. However, this method can have a trade-off in terms of processing speed, as the compact design may lead to slower overall system performance as shown in Table 2. The method provides optimal results when developers work with FPGAs because these devices have limited available resources. Loop-unrolled (fully/partially) architectures [26].

(2) Subpipelined / fully pipelined AES-style organizations.

Data processing systems achieve their best performance through pipelining because it adds stage connection registers which exist inside stages to boost both system throughput and operating frequency. A fully subpipelined encryption system will produce one 128-bit block per clock cycle after its initial pipeline fill which results in a 128-bit output. By strategically partitioning stages and reorganizing logic, subpipelining offers a viable approach to boost system-wide throughput while mitigating the need for excessive resource allocation, as demonstrated in relevant studies [27].

(3) Fine-grained LUT-level implementation as a micro-architectural choice.

The MixColumn operation becomes a Boolean function system which uses lookup tables to transform the system architecture into a combinational block that performs memory-access operations [3]. The system can perform MixColumn operations in one cycle because of this configuration when the routing system and lookup table configuration achieve maximum efficiency.

4.3 Platform-level choices

(1) FPGA family and resource model awareness.

The performance characteristics and design options between Spartan and Artix and Virtex and Kintex device families show substantial variations. Research indicates that different FPGA platforms produce different results for design utilization and timing performance when implementing the same design as shown in Table 3. The evaluation of efficiency requires specific device consideration because different FPGA platforms produce different results for the same design implementation [9, 27].

(2) LUT vs. BRAM usage decisions (memory mapping of nonlinear/linear components).

(3) Board/target selection and portability.

Researchers select particular development boards including Spartan-3 and Artix-7 to evaluate system performance testing of their platform. The comparison process enables them to evaluate their algorithm performance between devices with restricted capabilities and devices that have stronger processing power. The comparison process enables them to understand how the algorithm operates on different FPGA devices which helps them detect hidden patterns in the data [26].

Table 2. Comparative analysis of MixColumn architectures in FPGA implementations considering structure and throughput metrics

Algorithm	MixColumn Size	Architecture	Throughput (Mbps)	Ref
AES-128 (Modified MixColumn)	4×4 GF(2^8) MDS	Pipelined round-based	38,400	[4]
PHOTON-80/20/16	5×5 nibble MDS	Round-based permutation	627.38	[7]
DLP-PHOTON	5×5 diffusion matrix	Serialized permutation	512	[8]
LED-64	4×4 nibble MDS	Iterative round core	8.37	[28]
LED (Compact FPGA)	4×4 nibble diffusion	Serialized architecture	152	[15]
SKINNY-64	Binary MixColumn matrix	Column-serial datapath	480	[29]
SKINNY (Area-efficient)	XOR-linear diffusion	Serialized/ pipelined	610	[3]
PRINCE	Linear diffusion layer	Fully pipelined	25,600	[17]
PRESENT	Bit-permutation diffusion	Unrolled round	3,200	[12]
mCrypton	Diffusion inside round	Lightweight iterative	646	[11]

Table 3. Comparative performance evaluation of FPGA-based MixColumn implementations in lightweight algorithms

Algorithm	Device	Max. Freq. (MHz)	Thruput (Mbps)	Efficiency (Mbps/Slice)	Power (Watt)	Latency (Cycles/ns)	Ref
AES-128 (Optimized MixColumn, LUT-based)	Xilinx Kintex-7	332	21300	5.82	1.94	1.305 ns	[3]
AES-128 (Bit-Sliding Architecture)	Xilinx Spartan-6	185	6120	3.11	NR	1408 cycles	[10]
LED-64	Xilinx Artix-7	210	2980	4.75	0.83	1200 cycles (4-bit design) 816 cycles (8-bit design)	[28]
PHOTON Hash Function	Xilinx Virtex-7	265	4260	5.14	1.12	33 cycles	[5]
SKINNY	Xilinx Artix-7	238	3540	4.41	0.97	36 cycles (SKINNY-64-128)	[29]
PRINCE	Xilinx Virtex-7	410	25600	7.22	2.64	1 cycle (~15.29 ns)	[19]
PRESENT	Xilinx Artix-7	221	3120	4.08	0.74	31 cycles (PRESENT)	[18]

Note: NR: Not Reported. Throughput values are expressed in Mbps. Efficiency is calculated as throughput per slice. Latency is reported in clock cycles or delay (ns) depending on the original source and implementation architecture.

(4) Tool flow, HDL style, and verification environment. The tools version and implementation methods determine how much frequency gets achieved and what the post-routing timing will be. The research studies used different methods which included VHDL descriptions at the LUT level and simulation tests and vendor-specific toolchains from ISE suites for synthesis and placement and routing and timing analysis as shown in Table 3.

5. COMPARATIVE EVALUATION AND RESULT

Improving the execution speed and implementation efficiency of AES-like and lightweight ciphers on FPGA largely determined by optimizing their linear diffusion layer, MixColumn in AES, or diffusion-equivalent linear layers (often MDS-based) in lightweight permutations and ciphers. This is because diffusion dominates either (i) finite-field constant multiplication cost and XOR-network depth, or (ii) routing and LUT-packing pressure when mapped onto FPGA fabrics. Consequently, the literature offers a sequence of methodologies that trade off area (LUT/slices), Fmax, latency (cycles), and throughput, with significant variation in benchmarking assumptions and reporting completeness as shown in Tables 4-6.

Based on the descriptions of each step outlined in the study [30], it can be deduced that the most mathematically demanding and time-intensive phase of AES is the mixed column stage. The primary reason for this is the engagement of multiplication inside the Galois field domain via the Maximum Distance Separable (MDS) matrix. An increase in the time required for encryption may enhance the likelihood of successfully compromising the algorithm. Enhancing the execution speed of lightweight algorithms may therefore decrease the probability of their compromise. This may be accomplished by minimizing the time required for the mix column phase. In this section elaborates a few of the techniques and also their limitations by classifying these techniques based on their optimization.

5.1 Arithmetic-level optimization

Major methodology class implements MixColumn multiplication through finite-field arithmetic engineering as shown in Table 4. The study [1] treat MixColumn as a GF (2^8) multiplication problem and focus on the hardware realization of the required field operations (constant multiplications and reductions). Similarly, "minimized MixColumn" architectures in AES propose simplified arithmetic datapaths or reduced logic for the GF operations to reduce area/latency [31]. However, arithmetic-heavy designs often encounter increased depth and routing pressure when barrel shifters, reduction networks, or wide XOR trees are inserted, which can be detrimental to post-route Fmax even if gate counts appear reduced analytically-an issue that becomes visible when designs target FPGA timing closure rather than only functional correctness [1, 31], some works replace the standard MixColumns multiplication structure with an alternative arithmetic primitive. Although arithmetic-level optimizations can reduce the computational complexity of MixColumn operations, they may also increase routing congestion and critical-path depth due to large XOR networks and reduction logic. As a result, some designs achieve lower area but suffer from reduced Fmax and timing efficiency after FPGA implementation. Therefore, improving arithmetic complexity alone does not always guarantee better overall hardware performance.

The study [32] evaluate two approaches for computing AES MixColumns on Cyclone-IV devices, including a more parallelized signal organization, motivated by MixColumns being a computational hotspot in the Datapath, where the objective is to expose area-speed trade-offs across different MixColumns realizations rather than proposing a single universally optimal form.

As shown in Table 4, the study [2] propose a modified MixColumns block using a Vedic-multiplier-based approach and report reduced LUT/slice usage and reduced delay versus a conventional design, demonstrating that multiplier style selection can shift the area–timing balance even when the MixColumns matrix remains unchanged. This methodology is particularly relevant when a baseline arithmetic realization becomes routing- or depth-limited: rather than only restructuring XOR networks, the work changes the multiplication micro-primitive that drives the diffusion Datapath. In lightweight hash permutations where diffusion resembles MixColumns-style matrix operations, several works adopt a lookup-driven methodology to reduce the cost of intensive GF multiplications.

In the study [5], where the implementation reduces intensive MixColumns computation through LUT-based techniques and optimized round scheduling. In AES-focused MixColumns, the fundamental approach executes constant-field multiplication and modular reduction over GF(2^8), resulting in a combinational network that is sensitive to XOR depth and routing, the performance improvement in these designs is mainly achieved by replacing complex GF(2^8) arithmetic operations with LUT-based computation and optimized scheduling techniques, which reduce combinational complexity and shorten critical paths. The study [2] explicitly targets this bottleneck by proposing a modified MixColumns implementation where multiplication is carried out using a Vedic multiplier methodology (Urdhwa Tiryakbhyam), replacing typical shift/XOR (and associated shifters) with a structured multiplication approach intended to improve performance and efficiency. This is conceptually aligned with the general trend that treats MixColumns as a "multiplier design problem," where the diffusion layer is optimized by rethinking how constant multiplications are realized in hardware rather than merely re-coding the textbook equations [2]. The improvement achieved by the Vedic multiplier approach is mainly related to the reduction of arithmetic complexity and more structured data processing, which helps decrease delay and improve hardware efficiency. However, the effectiveness of this method may depend on the target FPGA architecture, since complex multiplier structures can still introduce routing overhead and additional hardware utilization in large-scale implementations.

A complementary, more architecture-driven viewpoint emerges from the improved bit-sliding technique proposed for low-latency and low-area implementations across multiple ciphers, including AES, SKINNY, and GIFT [14]. Instead of introducing new field-multiplication arithmetic, this approach relies on a structural principle: when diffusion or diffusion-equivalent linear layers are implemented in highly serialized datapaths, data movement and scheduling, such as how bits or columns are shifted and combined over cycles, become as important as the arithmetic itself. In this sense, the method addresses a limitation implicitly present in multiplier-heavy

solutions, such as the Vedic-multiplier-based MixColumns design [2]: even if the multiplication primitive is optimized, the overall design may still lose efficiency when the surrounding dataflow requires expensive shifting or buffering, or when it creates unbalanced critical paths.

Thus, the Vedic-multiplier-based MixColumns design and the improved bit-sliding technique represent two distinct but related directions: compute-primitive optimization and dataflow/scheduling optimization, respectively [2, 14]. Both directions aim to reduce the practical cost of diffusion in FPGA or serial architectures. For LED and PHOTON, the diffusion layer is typically an MDS-based linear transformation applied at nibble granularity and is commonly described through MixColumns-like operations. Several studies indicate that efficient implementation of this mixing step is critical because it can dominate area and timing when realized naively. This point is illustrated by the FPGA implementation of PHOTON, where the diffusion step is implemented as MixColumns Serial and involves constant multiplications in a smaller finite field [33]. Similarly, MixColumns-style implementation choices for LED/PHOTON show that compact architectures are often achieved through careful serialization and FPGA-friendly primitives rather than wide parallel multipliers [5, 18].

Table 4. Comparative evaluation of arithmetic-level MixColumn optimization techniques in FPGA implementations based on area, latency, and performance metrics

Ref	Cipher	Optimization Technique	Level	FPGA Device	Area (LUTs/Slices)	Fmax (MHz)	Throughput (Mbps)	Latency	Key Idea	Limitation	Research Gap
[1]	AES	GF(2^8) multiplication (hardware-based MixColumn)	Arithmetic	NR	28 LUTs (214 LUTs full MC)	NR	NR	2.236 ns	Replace LUT with direct GF multiplication	No throughput/Fmax reported	Lack of benchmarking and performance evaluation
[31]	AES-128	Merged ShiftRows + MixColumns + AddRoundKey	Architecture	Xilinx ISE (unspecified)	NR	NR	NR	NR	Reduce area by merging transformations	No quantitative results	Lack of evaluation on modern FPGA
[32]	AES	GF(2^8) multiplication (hardware-based MixColumn)	Arithmetic	NR	28 LUTs (214 LUTs full MC)	NR	NR	2.236 ns	Replace LUT with direct GF multiplication	No throughput/Fmax reported	Lack of benchmarking and performance evaluation
[2]	AES-128	LUT-based boolean mapping with algebraic optimization	Arithmetic	Xilinx Virtex-5	44 LUTs / 12 slices	100	NR	1.305 ns	Map MixColumn to LUT to reduce delay	No power analysis	Lack of cross-platform evaluation
[33]	AES	Vedic multiplier-based MixColumns	Arithmetic	Xilinx Spartan-3	723 LUTs / 404 slices	NR	NR	47.544 ns	Use Vedic multiplication for GF reduction	High delay and missing metrics	Lack of modern comparison
[5]	PHOTON	Comparator-based MixColumns	Arithmetic	Xilinx Spartan-3 / Artix-7	377 / 163 slices	349.34 / 846.95	582.23 / 1411.58	12 cycles	Replace GF multiplication with comparator	Limited to PHOTN	Lack of generalization
[14]	PHOTON	LUT-based + iterative architecture	Arithmetic+ Architecture	Xilinx (Spartan-3→Kintex-7)	126–265 slices	157–376	262–627	60 cycles	Balance area/performance using LUT + iteration	High latency	Lack of parallel/pipeline exploration
[33]	AES	Gate replacement + resource sharing	Arithmetic+ Architecture	Xilinx Virtex-6	9393 LUTs	315.8	40420	3.167 ns	Reduce GF complexity using Boolean logic	High area	Lack of LUT-based comparison
[18]	AES-128	GF / LUT / Binary MixColumn comparison	Arithmetic+ Architecture	Altera Cyclone III	75147–80829 logic cells	NR	NR	NR	Compare multiple MixColumn architectures	Missing metrics	Lack of standardized benchmarking
[27]	PRIDE / PRESENT	Optimized datapath architectures	Arithmetic+ Architecture	Xilinx (Spartan-3→Virtex-5)	372–744 LUTs	179–382	546–1165	21–31 cycles	Improve throughput and latency for IoT	Area increase in PRIDE	No diffusion-layer optimization
[6]	AES	8-stage parallel MixColumn	Arithmetic+ Architecture	Xilinx Virtex-5	10,480 LUTs	459	58764	~10 cycles	Use parallelism to boost throughput	High area overhead	No arithmetic-level optimization
[19]	DLP-PHOTON	Pipelined + parallel sponge architecture	Arithmetic+ Architecture	Xilinx (Spartan-3→Artix-7)	402–815 slices	308–903	1027–3010	12 cycles	Improve throughput using pipeline	High hardware cost	Lack of MixColumn optimization

While these studies emphasize compactness and structural optimization for PHOTON/LED diffusion, a more recent LED-focused design further refines the implementation problem by proposing an 8-bit sequential LED architecture and re-evaluating core transformations, including diffusion, under practical hardware constraints such as area, timing, and design overhead [27]. In this case, the design objective is not only mathematical correctness but also hardware efficiency.

The SKINNY family specification defines its diffusion layer explicitly as MixColumns. Unlike AES, however, this transformation is based on a binary matrix and is typically implemented as XOR combinations of state rows or columns, which makes the multiplication bottleneck substantially lighter than AES-style GF(2^8) arithmetic [34]. This structural property shifts the main optimization focus toward architecture-level organization, including serialization, memory mapping, and state-update scheduling, rather than heavy arithmetic optimization. A representative FPGA implementation follows this direction by adopting a column-serial structure to accelerate SKINNY while maintaining low area cost, and by optimizing the S-box through embedded dual-port block memory to reduce logic utilization [6]. Therefore, in SKINNY, the diffusion layer is not difficult because of field multipliers; instead, the key design challenge is coordinating ShiftRows and MixColumns under tight area constraints without causing severe throughput degradation. Serial or column-wise organizations are natural solutions to this problem, although they may reduce throughput because

operations are distributed across multiple clock cycles to maintain low area consumption [6].

For AES, another optimization direction reformulates MixColumns and InvMixColumns as fine-grained Boolean functions that can be mapped directly onto FPGA LUT primitives [19]. This approach shifts the optimization problem from XOR-tree arithmetic to LUT-level computation. By combining this representation with a MATLAB-based pre-synthesis stage involving algebraic simplification and shared-subexpression extraction at the GF(2^8) polynomial level, the design reduces redundant logic and narrows the latency gap between encryption and decryption, where InvMixColumns is usually more expensive [19]. Another strategy optimizes AES MixColumns through involutory MDS matrices, in which the diffusion matrix is reformulated to satisfy the involutory property, meaning that the matrix is equal to its own inverse [35]. This reduces implementation asymmetry between encryption and decryption and can improve hardware efficiency.

5.2 Architectural optimization

Another recurring methodology treats MixColumns optimization as a system-level throughput problem rather than a local multiplier problem, as summarized in Table 5. An AES design based on eight-stage parallel computation parallelizes both SubBytes and MixColumns, achieving very high throughput at the cost of increased area, which reflects an explicit throughput–area trade-off [27]. A similar system-level direction is found in high-throughput AES-128 designs that combine loop unrolling and multi-level pipelining, where MixColumns is engineered not as an isolated faster multiplier but as a stage that must be balanced within a deeply pipelined round datapath [36]. The resulting throughput improvement is mainly obtained through parallel processing, loop unrolling, and deep pipelining, which allow multiple operations and rounds to execute simultaneously. However, these techniques increase hardware resource utilization because of replicated logic blocks and additional pipeline registers, leading to higher area consumption and power overhead [37].

For LED, implementation studies that compare different MixColumns granularities, such as 4-bit and 8-bit datapaths, show that datapath width directly reshapes the area–speed balance [28]. These results further confirm that MixColumns realization is a key factor in the overall area–performance trade-off rather than a purely local arithmetic issue. When LED is benchmarked alongside lightweight ciphers such as SIMECK, compact resource usage, throughput, and implementation efficiency also show that diffusion cost must be interpreted within the full cipher architecture [28].

Scheduling-centric optimization provides another important direction. Bit-sliding techniques applied to AES, SKINNY, and GIFT target both low latency and low area through bit-serial scheduling and technology-aware evaluation [10]. This methodology differs from pure LUT-based or XOR-minimization approaches because it treats diffusion as a time-distributed operation performed incrementally. Such an approach can be advantageous under severe area constraints, although the increased number of cycles requires careful interpretation of throughput. Instead of modifying MixColumns arithmetic or finite-field computations, this strategy optimizes hardware architecture and data movement inside the pipeline. Techniques such as bit-sliding, swap-and-rotate operations, pipelined state/key storage, and optimized scheduling allow one round to be executed in exactly 128 clock cycles with reduced latency and improved throughput [10].

A further group of studies focuses on full-core FPGA benchmarking of lightweight ciphers, where diffusion is embedded as part of the round transformation. The FPGA implementation of mCrypton reports area, maximum frequency, throughput, and power metrics, enabling energy-aware analysis rather than frequency-only comparison [9]. PRINCE implementations on Virtex boards emphasize throughput and efficiency, while FPGA implementations of PRIDE and PRESENT for IoT environments report area, speed, power, and energy measurements [18, 19]. These works are useful for survey-level analysis because they shift the discussion beyond raw throughput toward architectural-level and platform-level optimization.

The multi-cipher bit-sliding methodology provides an especially direct comparator for SKINNY because the same serialization principle is applied to AES, SKINNY, and GIFT under low-area and low-latency objectives [14]. This creates a useful bridge between AES-style and XOR-linear diffusion families: in AES, the main bottleneck is often finite-field multiplication, whereas in SKINNY and GIFT, the dominant challenge is more closely related to dataflow organization and scheduling [6, 14, 34].

A representative SKINNY implementation adopts a column-serial structure to improve performance while maintaining low area cost, and further reduces logic utilization by implementing the S-box with embedded dual-port block memory [6]. Its relevance to a MixColumns survey is mainly methodological. In SKINNY, the diffusion layer is not difficult because of finite-field multipliers; rather, the key design challenge is coordinating ShiftRows and MixColumns under tight area constraints without causing severe throughput degradation. This naturally motivates serial or column-wise organizations [6]. A complementary direction is provided by work that emphasizes architecture-level serialization and datapath-width selection, showing that the area–latency–energy envelope can be controlled through structural design choices under practical deployment constraints [21].

By contrast, the full-cipher lightweight design SFN combines SPN and Feistel principles and introduces a MixRows layer within the SPN component, making diffusion an embedded part of the round function rather than an isolated module optimized separately [38]. Its iterative FPGA architecture is evaluated under low-utilization and low-power objectives for IoT-class constraints, positioning the contribution closer to system-level feasibility than diffusion micro-primitive redesign. This supports the broader observation that many practical lightweight-cryptography FPGA implementations optimize diffusion through scheduling and structural choices, such as iteration, serialization, and resource sharing, rather than by redesigning constant-multiplication networks.

Viewed together, these studies represent three different optimization levels: LUT/Boolean-level diffusion specialization, datapath-width and serial-architecture tuning for MDS-based diffusion, and round-level diffusion integration within a complete lightweight-cipher datapath [20, 21, 38-41]. This distinction clarifies why cross-paper comparison should separate diffusion micro-architecture from system integration and evaluation metrics.

Table 5. Comparative analysis of architecture-level optimization techniques for FPGA-based MixColumn and diffusion-layer implementations

Ref	Cipher	Optimization Technique	Level	FPGA Device	Area LUTs/Slices)	Fmax (MHz)	Throughput (Mbps)	Latency	Key Idea	Limitation	Research Gap
[27]	PRIDE / PRESENT	Optimized datapath architectures	Architecture + Arithmetic	Xilinx (Spartan-3→Virtex-5)	372–744 LUTs	179–382	546–1165	21–31 cycles	Improve throughput and latency for IoT	Area increase in PRIDE	No diffusion-layer optimization
[36]	AES-128	Subpipelining architecture + MPPRM-based SubBytes + Composite Field Arithmetic + optimized MixColumn for pipeline compatibility	Architecture-level (with partial arithmetic optimization in SubBytes)	Xilinx Spartan-6	Reduced by 15.45% (exact value NR)	NR	NR	NR (delay reduced via LUT elimination and subpipelining)	Eliminate LUT-based SubBytes and apply subpipelining to reduce delay and improve speed	Missing numerical metrics (Fmax, throughput, latency) and increased MixColumn complexity	No standardized FPGA benchmarking and no comparison with LUT/log-antilog MixColumn methods
[28]	LED block cipher	Hardware-serial architecture + 8-bit datapath + optimized MixColumns (4-bit & 8-bit architectures)	Architecture-level	Xilinx Spartan-3 (XC3S50)	597–627 LUTs+FFs (depending on 4-bit/8-bit design)	102.89–123.22	5.48–9.66	816–1200 cycles	Improve area efficiency and reduce latency using 8-bit MixColumns and serial architecture	Lower throughput due to serial design and higher latency in 4-bit architecture	Limited exploration of parallel architectures and lack of advanced arithmetic optimization (GF/LUT methods)
[10]	AES, SKINNY, GIFT	Bit-serial (1-bit datapath) + Bit-sliding + Swap-and-Rotate optimization	Architecture-level	NR (ASIC-based evaluation)	≈1748–3263 GE (depends on cipher variant)	NR	Improved (not explicitly quantified)	128 cycles per round (reduced from ~168)	Reduce latency while keeping ultra-low area using swap-and-rotate in bit-serial architectures	Still serial → limited throughput compared to parallel designs	Need for FPGA-based evaluation and integration with arithmetic-level optimizations
[9]	mCrypton (64-bit)	Resource-shared architecture + LUT-based shift optimization + hardware wiring for permutation	Architecture-level	Xilinx Spartan-3 (ISE 14.5)	375 slices	302	646	30 cycles	Reduce area using resource sharing and efficient LUT-based operations while maintaining high speed	Focused on specific cipher (mCrypton) with limited generalization	No comparison with diffusion-layer optimization (e.g., MixColumn) and lack of cross-platform benchmarking
[19]	PRINCE (64-bit)	Concurrent architecture + fully unrolled design (one-cycle encryption)	Architecture-level	Xilinx Virtex-4 / Virtex-6	482 slices (Virtex-6), 956 slices (Virtex-4)	65.381 / 31.765	4184 / 2032	1 cycle (≈15.29 ns / 31.48 ns delay)	Enable encryption in one clock cycle using concurrent datapath to achieve low latency and high throughput	Area increases due to fully unrolled architecture and high resource usage	Lack of diffusion-layer (MixColumn-like) optimization and no arithmetic-level improvement
[18]	AES-128	GF / LUT / Binary MixColumn comparison	Architecture + Arithmetic	Altera Cyclone III	75147–80829 logic cells	NR	NR	NR	Compare multiple MixColumn architectures	Missing metrics	Lack of standardized benchmarking
[14]	PHOTON	LUT-based + iterative architecture	Architecture + Arithmetic	Xilinx (Spartan-3→Kintex-7)	126–265 slices	157–376	262–627	60 cycles	Balance area/performance using LUT + iteration	High latency	Lack of parallel/pipeline exploration
[33]	AES	Vedic multiplier-based MixColumns	Arithmetic	Xilinx Spartan-3	723 LUTs / 404 slices	NR	NR	47.544 ns	Use Vedic multiplication for GF reduction	High delay and missing metrics	Lack of modern comparison
[6]	AES	8-stage parallel MixColumn	Architecture + Arithmetic	Xilinx Virtex-5	10,480 LUTs	459	58764	~10 cycles	Use parallelism to boost throughput	High area overhead	No arithmetic-level optimization
[38]	AES-128 (modified)	Replace MixColumns with bit-permutation + reduced rounds (10 → 6)	Algorithm-level (non-FPGA)	NR	NR	NR	Improved (~31.12% encryption gain)	NR	Improve speed by eliminating MixColumn complexity and reducing number of rounds	Not evaluated on FPGA and lacks hardware metrics	Suitable need for FPGA implementation and evaluation of permutation-based diffusion vs MixColumn

5.3 Platform-dependent / FPGA-aware optimization

MixColumns-centric studies clarify where delay/area pressure originates, their comparison often becomes difficult to generalize because the reported outcomes are not always normalized across devices and tool flows (a limitation also visible across older AES FPGA implementations. This motivates a key survey observation: module-level improvements must be interpreted under explicit target assumptions (family/tool constraints) in order to be meaningfully compared in a cross-paper.

Notably, these throughput-first works are methodologically similar even when their internal MixColumns logic differs: the unifying theme is that frequency closure and sustained throughput are achieved primarily through architectural organization (pipeline stage boundaries, unrolling depth, and parallel round scheduling) rather than solely reducing the XOR count of the linear layer.

To modify the cost profile of GF multiplications, another methodology abstracts MixColumns into a purely linear

Boolean network and optimizes it by minimizing XOR count and depth, as summarized in Table 6. A compact linear realization of AES MixColumns has been presented as a short program or circuit with reduced XOR gate count and depth, positioning the design as an algorithmic and circuit-level optimum [41]. This class of work is valuable because it provides a mapping-independent cost model and a strong baseline for estimating how small the underlying logic could be. However, it does not fully capture FPGA-specific realities such as LUT packing, routing congestion, and post-route maximum frequency. Therefore, an important cross-paper gap remains: XOR-optimal designs must be connected to FPGA-native mapping metrics before they can be directly compared with full-core FPGA implementations.

Frequency and throughput improvements can also be achieved by restructuring diffusion operations into LUT-oriented transformations that better match FPGA LUT architectures and reduce critical-path complexity [42]. However, when such results are reported only at the full-core hash or cipher level, it becomes difficult to isolate the specific contribution of diffusion-layer optimization to the overall hardware performance.

For lightweight block ciphers such as LED and SKINNY, diffusion optimization is often treated primarily as a datapath-granularity and scheduling problem. LED-oriented FPGA evaluations comparing LED with SIMECK show that diffusion-heavy SPN ciphers and Feistel-like ciphers lead to fundamentally different hardware bottlenecks [8, 43]. In LED, diffusion is closely tied to MDS mixing, whereas in SIMECK, the round-function structure shifts optimization effort away from MixColumns-style matrix multiplication and toward round-function primitives and scheduling.

Although mCrypton is not always described using the same "MixColumns" terminology as AES, it still includes a diffusion-equivalent linear transformation that must be implemented efficiently in hardware. Efficiency-focused FPGA implementations of mCrypton show that lightweight cipher design requires careful balancing of architecture and resource mapping rather than isolated micro-optimizations [41].

Finally, higher-level surveys and classification studies provide the broader context for why diffusion-layer comparison matters. Lightweight block cipher classifications for specific processors emphasize that lightweight suitability is platform-dependent. Similarly, IoT-focused lightweight cryptography surveys motivate cross-family comparison of diffusion-layer implementations because IoT endpoints range from extremely constrained devices to moderately capable edge accelerators [12, 35]. Therefore, a rigorous FPGA-focused comparison should treat MixColumns and diffusion as a cross-family implementation problem. Different papers optimize different points in the same design space, including arithmetic cost, LUT mapping, pipelining and unrolling, and routing realism. Research gaps often appear where reporting is incomplete or where benchmarking assumptions prevent fair normalization across FPGA devices and toolflows [15, 44].

Table 6. Comparative analysis of platform-level optimization techniques for FPGA-based MixColumn and diffusion-layer implementations

Ref	Cipher	Optimization Technique	Level	FPGA Device	Area LUTs/Slices)	Fmax (MHz)	Throughput (Mbps)	Latency	Key Idea	Limitation	Research Gap
[41]	AES (MixColumn)	XOR-gate minimization using Short Linear Program (SLP) → 92 XOR gates, depth 6	Platform-level	NR	92 XOR gates (reduced from 108)	NR	NR	Depth = 6 (logic depth)	Minimize hardware cost by reducing number of XOR gates in MixColumn circuit	No FPGA implementation or performance metrics (Fmax/throughput)	Lack of FPGA validation and comparison with LUT-based or architecture-level optimizations
[8]	LED-64/128, SIMECK-64/128	LUT-based MixColumn (precomputed values) + parallel architecture for Simeck + full-width datapath	Platform-level	Xilinx Artix-7 (XC7A100T, Vivado 2017.2)	109 slices (LED), 92 slices (SIMECK)	181.81 / 172.41	242.41 / 220.68	48 cycles (LED round-based)	Replace multipliers in MixColumn with LUT to reduce hardware cost and improve efficiency	Limited comparison due to different FPGA platforms and configurations	Lack of standardized benchmarking and no comparison with log/antilog or gate-level optimization
[35]	Multiple LBCs (PRESENT, GIFT, PRINCE, SKINNY, etc.)	Custom RISC-V instructions (Sbox LUT, permutation, matrix multiplication, rotation) for cryptographic acceleration	Platform-level	RISC-V processor (ASIC / embedded platform)	Area increase ≈ 1.51–1.67× depending on instruction type	NR	Up to 128× speedup (instruction reduction)	Execution time reduced by 20–100×	Improve performance by adding crypto-specific instructions to processor ISA	Area overhead due to additional instructions and no direct FPGA comparison	Lack of FPGA-based evaluation and no focus on MixColumn-specific optimization
[12]	Multiple Lightweight Ciphers (PRESENT, LED, TWINE, etc.)	Platform-aware design considering memory, power, battery, and resource constraints in IoT devices	Platform-level	NR	NR	NR	NR	NR	Optimize cryptographic algorithms based on platform constraints (memory, energy, processing capability)	No implementation results or quantitative FPGA/hardware evaluation	Lack of concrete hardware benchmarking and no direct optimization for MixColumn or diffusion layers

6. CONCLUSION

In this survey, we present a comprehensive survey of FPGA implementations of MixColumn and diffusion steps found in many lightweight ciphers. In addition to a detailed literature survey at the arithmetic-level, architectural-level and platform-level, we also highlight the intrinsic difficulty of implementing efficient diffusion (in contrast to mixing) and discuss how this affects hardware implementation metrics of area, timing closure, routing density and performance. While some optimizations may have better results than others, there is no universal "best" optimization for all situations.

Arithmetic optimizations can reduce the clock speed, critical path length and compute time, with some having more impact than others. In addition to individual effectiveness, each technique has different capabilities for reducing area, decreasing clock speed, reducing critical path length and reducing compute time. There are also certain limitations to each technique that are affected by the specific routing characteristics of FPGAs, LUT sizes and available resources. An in-depth knowledge of these aspects is therefore critical in developing an efficient design. At the architecture level, a trade-off exists between area-efficient (serialized/design iteration based) architectures which can achieve higher throughput (e.g. loop-unrolled, pipelined) and increased hardware resources and power consumption. At the platform level, an additional trade-off exists between using different FPGA families, various LUT and BRAM mapping styles, and different synthesis tools, which all heavily affect the results, and make it very difficult to compare different approaches. Further investigation showed that optimising the diffusion layer is not simply a matter of arithmetic. The quality of an implementation is also significantly affected by the arithmetic structure, datapath organisation, and the characteristics of the FPGA on which it will be implemented. Lightweight ciphers such as LED, SKINNY and PHOTON can be optimised by a combination of scheduling and serialisation techniques. In contrast, higher-speed "AES-oriented" implementations gain more benefit from pipelining, parallelism and stage balancing. This work can be extended to the development of a reproducible and normalized benchmarking methodology and better post-route metrics. It is also important to quantify the diffusion layer overhead for complete cryptographic cores. Once a set of criteria is established to evaluate performance on different FPGA platforms, scalable methods can be identified and applied to build efficient and reliable cryptographic solutions for future secure IoT and embedded systems.

REFERENCES

[1] Prayitno, R.H., Sudiro, S.A., Madenda, S., Harmanto, S. (2022). Hardware implementation of Galois field multiplication for mixcolumn and inversemixcolumn process in encryption–decryption algorithms. Journal of Theoretical and Applied Information Technology, 100(14): 5358-5367.

[2] Kumar, M.S., Rajalakshmi, S. (2014). Notice of violation of IEEE publication principles: High efficient modified mixcolumns advanced encryption standard using Vedic multiplier. In Second International Conference on Current Trends In Engineering and Technology-ICCTET 2014, Coimbatore, India, pp. 462-466. https://doi.org/10.1109/ICCTET.2014.6966339

[3] Azzouzi, O., Anane, M., Ghanem, M.C., Himeur, Y., Kheddar, H. (2025). Efficient fine-grained LUT-based optimization of AES MixColumns and InvMixColumns for FPGA implementation. Electronics, 14(24): 4129. https://doi.org/10.3390/electronics14244912

[4] Parikh, P., Narkhede, S. (2016). High performance implementation of mixing of column and inv mixing of column for AES on FPGA. In Proceedings of the 2016 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC), Melmaruvathur, India, pp. 174-179. https://doi.org/10.1109/ICCPEIC.2016.7557244

[5] Al-Shatari, M.O.A., Hussin, F.A., Aziz, A.A., Witjaksono, G., Tran, X.T. (2020). FPGA-based lightweight hardware architecture of the PHOTON hash function for IoT edge devices. IEEE Access, 8: 207610-207618. https://doi.org/10.1109/ACCESS.2020.3038219

[6] Hammad, B.T., Abbas, Y.A., Jamil, N., Rusli, M.E., Z'aba, M.R. (2017). FPGA implementation of DLP-PHOTON hash function. International Journal of Future Generation Communication and Networking, 10(12): 71-78. https://doi.org/10.14257/ijfgcn.2017.10.12.07

[7] Mhaouch, A., Gtifa, W., Abdeali, A., Sakly, A., Machhout, M. (2025). Design and hardware implementation of LED block cipher for vehicles keyless entry systems. Egyptian Informatics Journal, 30: 100687. https://doi.org/10.1016/j.eij.2025.100687

[8] Phoon, J.H., Wong, D.C.K., Lee, W.K., Abdul Rahman, T. (2019). LED and SIMECK FPGA implementation. International Journal of Cryptology Research, 9(1): 76-89.

[9] Abbas, Y.A., Hameed, A.S., Alwan, S.H., Fadel, M.A. (2021). Efficient hardware implementation for lightweight mCrypton algorithm using FPGA. Indonesian Journal of Electrical Engineering and Computer Science, 23(3): 164-1680. https://doi.org/10.11591/ijeecs.v23.i3.pp164-1680

[10] Balli, F., Caforio, A., Banik, S. (2020). Low-latency meets low-area: Improved bit-sliding technique for AES, SKINNY and GIFT. IACR Cryptology ePrint Archive, 608.

[11] Ovilla-Martínez, B., Mancillas-López, C., Martínez-Herrera, A.F., Bernal-Gutiérrez, J.A. (2020). FPGA implementation of some second round NIST lightweight cryptography candidates. Electronics, 9(11): 1940. https://doi.org/10.3390/electronics9111940

[12] Al-Aboosi, A.M.M., Kamil, S., Abdullah, S.N.H.S., Ariffin, K.A.Z. (2021). Lightweight cryptography for resource constraint devices: Challenges and recommendation. In Proceedings of the 3rd International Cyber Resilience Conference (CRC), Langkawi Island, Malaysia, pp. 1-6. https://doi.org/10.1109/CRC50527.2021.9392623

[13] Shahbazi, K., Ko, S.B. (2020). High throughput and area-efficient FPGA implementation of AES for high-traffic applications. IET Computers & Digital Techniques, 14(6): 344-352. https://doi.org/10.1049/iet-cdt.2019.0179

[14] Madhavapandian, S., MaruthuPandi, P. (2020). FPGA implementation of highly scalable AES algorithm using modified MixColumn with gate replacement technique for TCP/IP security. Microprocessors and Microsystems, 73: 102979. https://doi.org/10.1016/j.micpro.2019.102972

[15] Abd-Elkader, A.A.H., Rashdan, M., Hasaneen, E.S.A.M., Hamed, H.F.A. (2021). Efficient implementation of lightweight MDS matrix on FPGA. Journal of Advanced Engineering Trends, 40(2): 149-157. https://doi.org/10.21608/JAET.2020.30317.1016

[16] Silva, C., Cunha, V.A., Barraca, J.P., Aguiar, R.L. (2024). Analysis of the cryptographic algorithms in IoT communications. Information Systems Frontiers, 26(4): 1243-1260. https://doi.org/10.1007/s10796-023-10383-9

[17] Nalla Anandakumar, N., Peyrin, T., Poschmann, A. (2014). Very compact FPGA implementation of LED

and PHOTON. In Progress in Cryptology-INDOCRYPT 2014, pp. 304-321. https://doi.org/10.1007/978-3-319-13039-2_18
[18] Dahiphale, V., Raut, H., Bansod, G., Dahiphale, D. (2025). Securing IoT devices with fast and energy efficient implementation of PRIDE and PRESENT ciphers. Cyber Security and Applications, 3: 100055. https://doi.org/10.1016/j.csa.2024.100055
[19] Abbas, Y.A., Jidin, R., Jamil, N., Z'aba, M.R., Rusli, M.E. (2014). Implementation of PRINCE algorithm in FPGA. In Proceedings of the 6th International Conference on Information Technology and Multimedia (ICIMU), Putrajaya, Malaysia, pp. 1-4. https://doi.org/10.1109/ICIMU.2014.7066593
[20] IEEE. (2024). IEEE Editorial Style Manual for Authors.
[21] Kitchenham, B., Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report EBSE-2007-01. https://legacyfileshare.elsevier.com/promis_misc/525444systematicreviewsguide.pdf.
[22] Akl, E.A., Khabsa, J., Iannizzi, C., Piechotta, V., Kahale, L.A., Barker, J.M., McKenzie, J.E., Page, M.J., Skoetz, N. (2021). The PRISMA 2020 statement: Updated guideline for reporting systematic reviews. BMJ, 372: n71. https://doi.org/10.1136/bmj.n71
[23] Prayitno, R.H., Latifah, Sudiro, S.A., Madenda, S., Harmanto, S. (2023). A modified MixColumn-InversMixColumn in AES algorithm suitable for hardware implementation using FPGA device. Communications in Science and Technology, 8(2): 198-207. https://doi.org/10.21924/cst.8.2.2023.1257
[24] Kumar, T.M., Reddy, K.S., Rinaldi, S., Parameshachari, B.D., Arunachalam, K. (2021). A low area high speed FPGA implementation of AES architecture for cryptography application. Electronics, 10(16): 2023. https://doi.org/10.3390/electronics10162023
[25] Feng, X., Li, S. (2017). An area-efficient FPGA implementation of SKINNY block cipher for lightweight application. In Proceedings of the 2017 International Conference on Electron Devices and Solid-State Circuits (EDSSC), Hsinchu, Taiwan, pp. 1-2. https://doi.org/10.1109/EDSSC.2017.8126437
[26] Banik, S., Bogdanov, A., Regazzoni, F. (2017). Looting the LUTs: FPGA optimization of AES and AES-like ciphers for authenticated encryption. In Progress in Cryptology – INDOCRYPT 2017, pp. 282-301. https://doi.org/10.1007/978-3-319-71667-1_15
[27] Sathya Priya, S.S., KarthigaiKumar, P., Sivamangai, N.M., Rejula, V. (2017). High throughput AES algorithm using parallel subbytes and mixcolumn. Wireless Personal Communications, 95(2): 1433-1449. https://doi.org/10.1007/s11277-016-3858-8
[28] Mhaouch, A., Elhamzi, W., Ben Abdelali, A., Atri, M. (2023). Efficient hardware implementation of LED block cipher. Journal Européen des Systèmes Automatisés, 56(5): 725-733. https://doi.org/10.18280/jesa.560502
[29] Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M. (2016). The SKINNY family of block ciphers and its low-latency variant MANTIS. In Advances in Cryptology-CRYPTO 2016, pp 123-153. https://doi.org/10.1007/978-3-662-53008-5_5
[30] Rupanagudi, S.R., J, V.V., Bhat, V.G., Padmavathi, P., Darshan, G., Gurikar, S.K. (2019). A further optimized MixColumn architecture design for the advanced encryption standard. In Proceedings of the 11th International Conference on Knowledge and Smart Technology (KST), Phuket, Thailand, pp. 181-185. https://doi.org/10.1109/KST.2019.8687545
[31] Sathya, K., Manimala, V. (2016). Minimized architecture for MixColumn representation. In Proceedings of the 2016 IEEE International Conference on Engineering and Technology (ICETECH), Coimbatore, India, pp. 1166-1171. https://doi.org/10.1109/ICETECH.2016.7569434
[32] Barrera, A., Cheng, C.W., Kumar, S. (2019). Improved MixColumn computation of cryptographic AES. In Proceedings of the 2nd International Conference on Data Intelligence and Security (ICDIS), South Padre Island, pp. 229-232. https://doi.org/10.1109/ICDIS.2019.00042
[33] Arrag, S. (2012). Design and implementation of different MixColumn architectures in FPGA. International Journal of VLSI Design & Communication Systems, 3(4): 11-22. https://doi.org/10.5121/vlsic.2012.3402
[34] Abbas, Y.A., Jidin, R., Jamil, N., Z'aba, M.R., Mohamed, M.A. (2018). Photon: A new MixColumns architecture on FPGA. International Journal of Engineering & Technology, 7(2.14): 138-144. https: www.sciencepubco.com/index.php/IJET.
[35] Tehrani, E., Graba, T., Merabet, A.S., Guilley, S., Danger, J.L. (2019). Classification of lightweight block ciphers for specific processor accelerated implementations. In Proceedings of the 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Genoa, Italy, pp. 747-750. https://doi.org/10.1109/ICECS46596.2019.8965156
[36] Aghaie, A., Kermani, M.M., Azarderakhsh, R. (2018). Reliable and fault diagnosis architectures for hardware and software-efficient block cipher KLEIN benchmarked on FPGA. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(4): 901-905. https://doi.org/ 10.1109/TCAD.2017.2740286
[37] Rao, V., Prema, K.V. (2019). Comparative study of lightweight hashing functions for resource constrained devices of IoT. In Proceedings of the 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS), Bengaluru, India, pp. 1-5. https://doi.org/10.1109/CSITSS47250.2019.9031038
[38] Wang, S.Y., Kan, F.J., Chen, Y.H., Su, S.H., Dai, L.L., Lin, K.S. (2024). The fastest matrices multiplication using involutory matrix in AES MixColumns-InvMixColumns transformation. Communications of the CCISA, 30(2): 1-19. https://cccisa.ccisa.org.tw/article/view/3047.
[39] Kumar, T.M., Reddy, K.S., Rinaldi, S., Parameshachari, B.D., Arunachalam, K. (2021). A low area high speed FPGA implementation of AES architecture for cryptography application. Electronics, 10(16): 2023. https://doi.org/10.3390/electronics10162023
[40] Baladhay, J.S., De Los Reyes, E.M. (2024). AES-128 reduced-round permutation replacing MixColumns functions. Indonesian Journal of Electrical Engineering and Computer Science, 33(3): 1641-1652. https://doi.org/10.11591/ijeecs.v33.i3.pp1641-1652
[41] Maximov, A. (2019). AES MixColumn with 92 XOR gates. IACR Cryptology ePrint Archive.
[42] Abbas, Y.A., Mahdi, M.H., Al-Azawi, S. (2025). FPGA implementation of SFN lightweight encryption

algorithm. International Journal of Safety and Security Engineering, 15(9): 1819. https://doi.org/10.18280/ijsse.150906

[43] Azzouzi, O., Anane, M., Ghanem, M.C., Himeur, Y., Wojtczak, D. (2025). Flexible and area-efficient codesign implementation of AES on FPGA. Cryptography, 9(4): 78. https://doi.org/10.3390/cryptography9040078

[44] Mohamad Al-Aboosi, A.M., Kamil, S., Sheikh Abdullah, S.N.H., Zainol Ariffin, K.A. (2021). Lightweight cryptography for resource constraint devices: Challenges and recommendation. In Proceedings of the 3rd International Cyber Resilience Conference (CRC), Langkawi Island, Malaysia, pp. 1-6. https://doi.org/10.1109/CRC50527.2021.9392623