StarGate: Pseudo-random number generator based on matrix transformations

Author: Daniel E. Baykalov

felix.trof@gmail.com

Abstract: The growing demand for efficient and reliable stream ciphers has stimulated the development of new pseudorandom generators for cryptographic applications. This article introduces StarGate, a new byte stream generator designed to achieve an optimal balance between computational efficiency, high diffusion, and unpredictability. StarGate uses a 16x16 byte matrix initialized from a 256-byte key, dynamic navigation with offset-controlled coordinates, modular matrix gates, and a fixed-size pool for non-linear mixing, ensuring high entropy of the output stream. The algorithm combines bitwise operations, transformations of rows and columns of a matrix, as well as nonlinear mixing through gates, which makes it promising for lightweight cryptographic applications such as Internet of Things devices and real-time data encryption. Key advantages include ease of implementation in high-level languages such as Go, flexibility in using keys of various sizes, and the potential for hardware optimization. Although full cryptographic robustness requires further cryptanalysis, preliminary estimates indicate that StarGate demonstrates competitive performance and randomness characteristics compared to well-known stream ciphers such as Salsa20. The article describes the design principles, operation, and results of experimental evaluation of StarGate, highlights its unique contributions to the field of cryptographic stream generators, and suggests directions for further research.

Keywords: pseudorandom number generator, stream cipher, matrix transformations, cryptographic diffusion, nonlinear mixing, high entropy, cryptographic applications, compact algorithms, key generation, IoT

Introduction

Modern cryptographic systems face increasing challenges due to both the growth of computing power and the emergence of new threats, including potential attacks using quantum computers. Stream ciphers, due to their high performance and flexibility, remain an important tool for ensuring data privacy in applications such as secure communications, real-time encryption, and cryptography for resource-constrained devices such as Internet of Things (IoT) devices. However, traditional stream generators such as RC4 or even more modern algorithms such as Salsa20 and ChaCha have their limitations[4], including vulnerabilities to certain types of attacks or high requirements for computing resources, which makes them less suitable for some scenarios.

This article introduces StarGate, a new byte stream generator designed to achieve an optimal balance between computational efficiency, ease of implementation, and a high degree of output stream diffusion. StarGate is based on a 16x16 byte matrix initialized from a 256-byte key, which is used in combination with dynamic offset-driven navigation, modular matrix gates, and a fixed-size pool for non-linear mixing. What makes StarGate unique is the integration of multiple levels of transformation-bitwise operations, row reversals, cyclic column shifts, and passing through matrix gates — which provides a complex relationship between input and output data, enhancing potential resistance to correlation and differential attacks. The algorithm is designed to be implemented in high-level languages such as Go, which makes it accessible to software applications, as well as promising for hardware implementation due to its compact data structure and minimal memory usage.

Key benefits of StarGate include:

- High diffusion. The combination of matrix transformations and nonlinear mixing ensures a uniform distribution of state changes, which contributes to the creation of a high-quality pseudorandom flow.
- Effectiveness. The algorithm is optimized for fast execution on modern platforms, including devices with limited resources.
- Easy implementation. The minimalistic structure and use of standard operations make StarGate accessible to developers without deep knowledge of cryptography.

While StarGate's full cryptographic strength requires further analysis, preliminary estimates show that it is competitive with existing stream ciphers such as Salsa20 in terms of performance and output stream quality. This article is structured as follows: section 2 discusses existing approaches to stream generation and their limitations; section 3 describes the StarGate architecture in detail; section 4 analyzes its key features and benefits; section 5 presents the results of an experimental evaluation; section 6 discusses potential applications and areas for further research; and finally, section 7 summarizes and formulates conclusions. This paper aims to present StarGate as a promising tool for cryptographic applications and encourage further research on its properties.

Overview of existing approaches

Stream ciphers and pseudo-random generators (PRNGs) play a key role in cryptography, providing efficient data encryption in scenarios that require high speed and minimal computing resources. This section discusses the main approaches to designing stream generators, their advantages and limitations, and positions the StarGate generator as an innovative solution that combines elements of matrix transformations and nonlinear mixing.

Classical stream ciphers, such as the well-known permutation-based algorithms, use simple bitwise operations to generate pseudorandom sequences.

These algorithms provide high performance, but often have vulnerabilities related to the correlation between the output stream and the internal state. More modern developments based on addition, rotation, and bitwise elimination (Add-Rotate-XOR, ARX) operations have significantly improved cryptographic stability[3]. Such algorithms work with large states (for example, 64-byte matrices) and apply a series of transformations to achieve high diffusion and resistance to differential attacks. However, their initialization complexity and computational requirements may be redundant for resource-constrained devices such as Internet of Things (IoT) devices.

Matrix structures are widely used in block ciphers, where transformations such as byte substitution and column mixing provide obfuscation and diffusion[5]. In stream ciphers, matrix approaches are less common, but they are found in generators based on cellular automata. Such systems use simple rules to create complex pseudorandom sequences, but their implementation is often limited by high computational complexity or insufficient speed. Chaotic systems based on nonlinear dynamics are also used to generate pseudorandom flows due to their high sensitivity to initial conditions. However, their practical implementation is difficult due to the need for precise floating-point calculations, which reduces their efficiency in software and hardware systems.

Modern stream ciphers, despite their advantages, often require significant computational resources for initialization and state processing, which makes them less suitable for lightweight applications. Although the classical algorithms are effective, they are outdated due to the identified cryptographic weaknesses[2]. Generators based on cellular automata and chaotic systems face problems of scalability and performance, as well as complexity of implementation on universal platforms. In addition, many existing solutions are optimized for specific architectures, which limits their applicability in a variety of scenarios. Thus, there

is a need for generators that combine compactness, performance, and flexibility while maintaining the potential for high randomness and diffusion.

The proposed StarGate generator occupies a unique niche, combining the advantages of matrix structures, dynamic navigation and non-linear mixing. Unlike classical algorithms, StarGate uses complex transformations, including XOR crosses, row reversals, and cyclic column shifts, which ensures high diffusion and reduces the likelihood of correlations. Compared to modern ARX algorithms, StarGate uses a more compact but multi-layered state. This structure allows you to efficiently manage dependencies between states and the output stream, enabling complex data transformation. StarGate matrix gates that perform modular addition operations add an extra layer of non-linearity by borrowing ideas from block ciphers, but adapting them for stream generation. Unlike chaotic systems, StarGate relies exclusively on integer operations, which simplifies its implementation in both software and hardware. The Go implementation provides easy integration into modern software systems, and the algorithm structure opens up prospects for optimization at the hardware level.

Thus, StarGate represents a promising approach that combines matrix transformations, dynamic navigation, and non-linear blending to create a high-quality pseudorandom stream. Although its cryptographic robustness requires further analysis, preliminary characteristics indicate potential for application in lightweight cryptographic systems and high-performance applications. In the following sections, we will discuss the StarGate architecture in detail and evaluate its properties.

StarGate architecture. Introduction to Architecture.

The StarGate generator is a new approach to creating pseudo-random byte streams, designed to achieve an optimal balance between computational efficiency, state compactness, and high degree of diffusion. The main design goal of StarGate

is to provide high-quality pseudorandom output suitable for cryptographic applications, while maintaining ease of implementation and adaptability to various platforms, including resource-constrained devices such as Internet of Things (IoT) devices.

Unlike traditional stream generators, which often rely on linear transformations or complex operations with large states[1], StarGate uses a multi-layer architecture that combines matrix transformations, nonlinear operations, and dynamic state management. The central element is a 16x16 byte matrix initialized from a 256-byte key, which serves as the basis for storing and transforming state. Dynamic navigation through the matrix is performed using coordinates (X, Y), updated based on the offset accumulator (offsetSum), which ensures that each generated byte depends on previous states. Additionally, StarGate includes 16 4x4-byte matrix gates that perform modular addition operations to enhance non-linearity, and a fixed-size pool that stores the last generated bytes for subsequent non-linear mixing. These components together provide high diffusion, where a change in a single bit in the state can significantly affect the subsequent output stream.

АрхитектураThe StarGate architecture makes it promising for applications that require high performance and compactness, such as real-time encryption or pseudorandom sequence generation for cryptographic protocols. In the following subsections, you will learn more about the state components, basic operations, and benefits of StarGate design.

StarGate architecture. State components.

The StarGate generator state is a complex but compact structure designed to provide high diffusion and non-linearity of the output stream with minimal computational effort. It includes four key components: a 16x16-byte matrix, coordinates (X, Y) and an offset accumulator (offsetSum), a fixed-size pool for

storing the last generated bytes, and a set of 16 4x4-byte matrix gates. Each component plays a unique role in the pseudorandom stream generation process, providing complex dependencies between input, state, and output.

The 16x16 byte matrix is the central element of the StarGate state. It is initialized from a 256-byte key divided into 16 lines of 16 bytes each, which provides a compact but large enough space for storing and converting data. The matrix serves as the basis for performing operations such as XOR crosses, row reversals, and cyclic column shifts that change state during the generation of each byte. Dynamic interaction with the matrix via (X, Y) coordinates allows efficient use of the entire state volume, increasing diffusion.

Coordinates (X, Y) and offset accumulator (offsetSum) control navigation through the matrix. The X and Y coordinates, each in the range from 0 to 15, define the current position in the matrix from which the value is extracted or written. They are initialized from the last bytes of the key and updated at each generation step using the offset accumulator, a single-byte variable that accumulates the sum of values extracted from the matrix. This mechanism allows dynamic movement through the matrix, creating complex dependencies between successive states and the output stream, which potentially complicates prediction.

A fixed-size pool (LastPool) stores the last generated bytes (up to 32 bytes), providing non-linear mixing of the current output with previous values. The pool acts as a fixed-length buffer, where new bytes are added and old ones are removed when the limit is reached. The mix operation uses pool values to modify the current byte through bitwise rotations and additions, which adds an additional layer of nonlinearity and increases the entropy of the stream.

Matrix gates (4x4) are a set of 16 matrices with a size of 4x4 bytes, each of which is initialized from a 16-byte key fragment. Gates perform modular addition operations on passing bytes, using coordinates calculated from the offset

accumulator. This process introduces non-linear transformations similar to the operations of block ciphers, but adapted for stream generation. Gates update their state with each pass, which further increases the dependence of the output on previous operations.

The combined use of these components provides StarGate with a high degree of diffusion and flexibility. The matrix and gates create complex state transformations, coordinate navigation adds dynamism, and pooling increases nonlinearity. This architecture makes StarGate suitable for cryptographic applications that require compactness and performance. In the following subsections, we will discuss operations that implement byte generation and their impact on stream properties.

StarGate architecture. Basic operations.

StarGate implements the generation of a pseudo-random byte stream through a sequence of carefully coordinated operations on internal state. Each operation is designed to provide diffusion, non-linearity, and dynamic coordinate change, which is a key element of the architecture. Basic operations can be classified as initialization, navigation, modular transformations, and mixing using a fixed-size pool.

Initialization

StarGate initialization is performed by converting a 256-byte key to a 16×16 matrix and forming 16 matrix gates of 4×4 size.

1. A 16x16 matrix is formed from consecutive blocks of 16 bytes each:

$$M[i] = K[16i: 16(i + 1) - 1],\ i = 0 \ldots 15$$

2. The X and Y coordinates are selected from the last two bytes of the key, modularly reduced to the range 0-15:

$$X = K[254] \bmod 16,\ Y = K[255] \bmod 16$$

3. 4 × 4 matrix gates are created from consecutive key blocks of 16 bytes each, divided into 4×4 submatrices.

4. A fixed-size pool is used to store the last bytes of generation and provides non-linear mixing in subsequent steps.

Initialization guarantees deterministic formation of the initial state based on the key, while ensuring a multi-layered structure.

Navigating the matrix

Dynamic navigation is performed using the coordinates (X,Y) and the offset accumulator offsetSum. At each generation step:

$$offsetSum \leftarrow (offsetSum + M[X][Y]) \bmod 256$$

Coordinates are updated using modified deltas:

$$\Delta X = (offsetSum \oplus M[(X + 1) mod 16][Y] \bmod 16$$

$$\Delta Y = (offsetSum + M[X][(Y + 1) \bmod 16]) \bmod 16$$

After the update:

$$X \leftarrow (X + \Delta X) \bmod 16,\ Y \leftarrow (Y + \Delta Y) \bmod 16$$

This mechanism provides non-linear movement through the matrix, depending on the current state and previous values.

Matrix transformations

StarGate uses several transformation modules to diffuse and complicate the structure:

XOR crosses

The XORCross function allows simultaneous mixing by row and column:

$$M[i][Y] \leftarrow M[i][Y] \oplus M[X][Y],\ i \neq X$$

$$M[X][i] \leftarrow M[X][j] \oplus M[X][Y],\ j \neq Y$$

Then the string is reversed:

$$M[X][i] \leftrightarrow M[X][15 - j],\ j = 0 \ldots 7$$

And a cyclic column shift:

$$M[i][Y] \leftarrow ((M[i][Y] << 1) \vee (M[i][Y] >> 7))\ mod\ 256$$

This step guarantees strong local mixing with the effect of each element on adjacent rows and columns.

Row reverse and column shift

Additionally, the reverseRow(row) and shiftColDown(col) operations are used:

Reverse of the string:

$$M[row][j] \leftrightarrow M[row][15 - j]$$

Cyclic column shift down:

$$M[i][col] \leftarrow M[i - 1][col],\ i = 15 \ldots 1,\ M[0][col] \leftarrow M[15][col]$$

These operations enhance diffusion and prevent linear structures from being retained in the matrix.

Passing through matrix gates

Each byte (v) passes through 16 4x4 matrix gates. Modular addition is performed for each gate:

$$v' = (v + g[x][y]) \, mod \, 256$$

where x,y depend on the accumulator of offsets offsetSum. After passing, the value updates the internal gate matrix:

$$g[x][y] \leftarrow (g[x][y] + v) \, mod \, 256$$

This mechanism provides a local nonlinear transformation that adapts to the current state of the generator.

Non-linear mixing with a fixed-size pool

StarGate applies MixByte (val) for additional mixing. The algorithm uses the pool state of the last bytes:

$$v \leftarrow val$$

$$v \leftarrow ((v << 3) \vee (v >> 5)) + \sum_{i=1}^{n} (((a_i << 1) \vee (a_i >> 7)) + b_i) \, mod \, 256$$

where $a_{ai} = State[i], \; bi_i = State[i - 1]$ - previous values from the pool.

This operation adds a strong element of non-linearity and memory to the generator, since each new byte depends on several previous ones.

Byte Generation Algorithm (GetNext)

The complete byte generation sequence combines all the operations described above:

1. Counting the current byte: $currentVal = M[X][Y]$

2. Обновляем $offsetSum$:

$offsetSum \leftarrow (offsetSum + currentVal)\ mod\ 256$

3. Calculate$the\ deltas$: $\Delta x,\ \Delta yY$
4. Update coordinates: $X \leftarrow X + \Delta X,\ Y \leftarrow Y + \Delta Y$
5. Apply XORCross
6. Passing through the gates:

$r = PassThroughGates(transformedVal)$

7. Mixing with the pool: $r = MixByte(r)$
8. Returning the resulting byte and updating the pool

Key benefits of StarGate operations

The combination of XOR crosses, matrix gates, reversals, and shifts ensures that the influence of each byte is quickly propagated across the state.

Using a fixed-size pool (LastPool) introduces dependency on the generation history, making it more difficult to predict the output stream.

The total amount of internal state is less than 1 KB:

- 16x16 matrix — 256 bytes,
- 16 4×4 matrix gates — 256 bytes,
- a fixed — size pool (for example, 32) - 32 bytes.
- `service variables (X, Y, offsetSum) — 3 bytes.`

`In total, ≈ 547 bytes of state.`

All operations are based on integer shifts, additions, and XOR, which provides high performance in both software and potential hardware implementations.

Design Advantages

The StarGate generator architecture is designed to meet the requirements of modern cryptographic applications, such as real-time encryption, pseudorandom sequence generation, and data protection on resource-constrained devices. The unique combination of matrix transformations, dynamic navigation, modular gates, and non-linear mixing provides a number of key advantages that make StarGate promising for lightweight and high-performance systems. The main design advantages and their implications for potential applications are discussed below.

Multi-level diffusion

StarGate uses a combination of operations such as cross-XOR, row reversals, cyclic column shifts, and modular transformations through gates to achieve a high degree of diffusion. Each operation makes changes to the state, propagating the effect of a single bit on the set of matrix elements and subsequent bytes of the output stream. For example, the cross-XOR operation changes all row and column elements except for the center point, and subsequent row reversal and column bitwise shift further enhance the effect. This provides a property where even a minimal change in key or state results in significant changes in the output stream, which is critical for cryptographic generators.

Non-linearity with memory

Non-linear mixing implemented through a fixed-size pool (LastPool) introduces a dependency of the current byte on the history of previous generated values. The MixByte operation uses bitwise rotations and additions with pool elements, creating complex nonlinear dependencies. This complicates the analysis of the output stream by methods based on linear approximations and increases entropy. A fixed-size pool (32 bytes) provides a balance between preserving memory of previous states and computational efficiency, making StarGate suitable for applications that require high randomness.

Compact state

The total amount of StarGate state is approximately 547 bytes: 256 bytes for the 16x16 matrix, 256 bytes for the 16 4x4 gates, 32 bytes for the pool, and 3 bytes for the coordinates (X, Y) and offset accumulator (offsetSum). This compactness makes StarGate particularly attractive for resource-constrained devices, such as IoT devices, where minimizing memory is critical. Despite its small size, the multi-layered state structure provides complex transformations comparable to heavier algorithms.

Implementation efficiency

StarGate relies exclusively on integer operations (addition, bitwise shifts, XOR, and modulo operations), which provides high performance in both software and potential hardware implementations. The Go implementation demonstrates easy integration into modern software systems, and the absence of floating-point operations or complex calculations makes the algorithm suitable for hardware optimization, for example, on FPGAs or microcontrollers. Operations such as cross-XOR and gate passing require a minimum number of processor cycles, which ensures high byte generation speed.

Flexibility and adaptability

Dynamic navigation and modular gates provide flexibility in state management, and the compact structure makes it easy to scale the algorithm for different platforms. This makes StarGate a universal tool for cryptographic protocols that require both high performance and low resource consumption.

These advantages make StarGate promising for applications in lightweight cryptographic systems, including IoT data encryption, stream ciphers, and pseudo-random sequence generation for simulations. While full cryptographic robustness requires further analysis, the current design demonstrates the potential

to achieve competitive performance over current stream ciphers while maintaining simplicity and efficiency.

Experimental assessment

To evaluate the characteristics of the StarGate generator, a preliminary experimental evaluation was performed, aimed at analyzing the quality of the pseudo-random flow, performance, and resource consumption. The main goal was to test StarGate's ability to generate high-quality pseudorandom sequences suitable for cryptographic applications, and to evaluate its effectiveness on various platforms, including devices with limited resources.

Testing methodology

The quality of the pseudorandom flow was evaluated using the NIST Statistical Test Suite (SP 800-22), which includes 15 basic tests to check uniformity, independence, and lack of correlations. The StarGate generator was initialized with a 256-byte key generated using a cryptographically strong random number source. We tested a 10 MB binary file (≈83,886,080 bits) divided into 80 bitstreams of 800,000 bits (100,000 bytes) each. Performance was measured on the 2023 iMac platform (Apple M2, 8 cores, 3.5 GHz) with a Go implementation. The generation time of 10 MB of data and memory consumption were measured. For comparison, the Salsa20 stream cipher was used.

Results

NIST STS tests have demonstrated a high degree of randomness in the StarGate output stream. Most of the tests, including Frequency, BlockFrequency, CumulativeSums, Runs, LongestRun, FFT, ApproximateEntropy, Serial, and LinearComplexity, showed a passing ratio of 78–80/80 (97.5–100%), which is significantly higher than the NIST minimum threshold (≈76/80. 95%). The Rank test reached 80/80 (100%), confirming the effectiveness of matrix operations. The RandomExcursions and RandomExcursionsVariant tests (analyzed on 10 bitstreams) showed 10/10 (100%), which is above the threshold (8/10), demonstrating the absence of systematic deviations in transitions. The NonOverlappingTemplate test (188 subtests) showed 76–80 / 80 for most templates, confirming high uniformity. The distribution of p-values over the intervals (C1–C10) was close to uniform ($p > 0.01$ for most tests), which indicates the qualitative diffusion and nonlinearity implemented in StarGate.

In terms of performance, StarGate achieved a generation rate of 9.09 MB/s (10 MB in 1.1 seconds) on the 2023 iMac, which is comparable to the Salsa20 (≈10 MB / s on a similar platform). At the same time, StarGate uses a compact state of ≈547 bytes (16x16 matrix — 256 bytes, 16

4x4 gates — 256 bytes, 32 — byte pool, 3-byte service variables), which is significantly smaller than Salsa20 (≈672 bytes). This compactness makes StarGate particularly attractive for resource-constrained devices, such as IoT devices, where minimizing memory is critical.

Limitations

The current evaluation focused on statistical randomness and performance, but further cryptanalysis is needed to confirm resistance to attacks such as differential or linear attacks. Some NIST tests require additional analysis with a large number of bitstreams (for example, 100-1000) to increase statistical significance. Performance on microcontrollers and specialized platforms requires separate testing to confirm its applicability in IoT scenarios. Optimizing operations, such as passing through gates, can further improve speed on high-performance systems.

Applications and prospects

The StarGate generator is designed as a universal solution for generating pseudorandom sequences suitable for cryptographic and non-cryptographic applications. Its unique architecture, which combines a 16x16 matrix, 4x4 gates and a compact state pool, provides a high degree of randomness and efficiency, which opens up a wide range of applications.

StarGate Applications

1. Stream ciphers: StarGate can be used for real-time data encryption, for example, in network protocols (VPN, TLS). Its speed (9.09 MB / s on a 2023 iMac) and compact state (547 bytes) make it competitive against algorithms like Salsa20, especially on systems with limited memory.

2. Cryptographic parameter generation: StarGate is suitable for creating nonces, keys, or initial vectors in cryptographic protocols due to the high statistical randomness confirmed by NIST tests (78-80 / 80).

3. Lightweight Systems(IoT): The compact state and low memory requirements make StarGate ideal for Internet of Things (IoT) devices, such as sensors or smart devices, where resources are limited. The Go implementation provides easy integration into software and hardware platforms.

4. Simulations and hashing: The high entropy and uniformity of the output stream allow you to use StarGate in modeling, generating test data, or as a component in hashing algorithms.

Advantages in the context of the application

1. A state of 547 bytes minimizes memory consumption, which is critical for IoT and embedded systems.

2. The 9.09 MB/s speed is comparable to industry leaders like Salsa20, with a smaller state size.

3. The algorithm supports keys of arbitrary length (256 bytes by default) and is easily adapted for various scenarios.

4. The Go implementation provides cross-platform functionality and easy integration into software systems, and the algorithm structure allows you to effectively implement it at the hardware level (for example, FPGA).

Conclusion

StarGate shows high statistical randomness, confirmed by NIST tests (78–80 / 80 for most tests), and competitive performance (9.09 MB / s) with a compact state (≈547 bytes). These characteristics make it promising for lightweight cryptographic applications, especially in

resource-constrained environments. Further research should include cryptanalysis and testing on specialized platforms to expand application scenarios.

The StarGate pseudorandom sequence generator is an innovative solution that combines compactness, performance, and high statistical randomness, making it promising for a wide range of cryptographic and non-cryptographic applications. Designed with a 16x16 matrix, 4x4 gates, and a compact state pool, StarGate demonstrates a unique approach to pseudorandom sequence generation that is optimized for resource-constrained systems.

The experimental evaluation confirmed the high quality of the StarGate output stream. NIST Statistical Test Suite tests showed a pass ratio of 78–80/80 (97.5–100%) for most tests, including Frequency, BlockFrequency, CumulativeSums, Runs, LongestRun, FFT, ApproximateEntropy, Serial, and LinearComplexity, as well as 10/10 for RandomExcursions and RandomExcursionsVariant. These results indicate high entropy and uniformity achieved due to multilevel diffusion and nonlinear mixing. Performance on the 2023 iMac platform (Apple M2, 8 cores, 3.5 GHz) was 9.09 MB / s (10 MB in 1.1 seconds), which is comparable to the Salsa20 stream cipher (≈10 MB / s), while the StarGate state (≈547 bytes) is much more compact (Salsa20 — ≈672 bytes). This highlights StarGate's suitability for lightweight systems such as Internet of Things (IoT) devices.

StarGate has potential applications in stream ciphers, cryptographic parameter generation, IoT, and simulations. Its compact size and cross-platform implementation in the Go language provide flexibility for integration into software and hardware platforms. Further research, including cryptanalysis and testing on microcontrollers, will expand the scope of applications and confirm the resistance to attacks. Optimizing operations such as gate vectorization can further improve performance.

In conclusion, StarGate offers a balanced combination of efficiency, compactness, and quality, making it a promising solution for today's cryptographic challenges. Its potential in lightweight systems and flexibility open the way for further development and implementation in real-world applications.

List of literature

1. Al-Shaarani F. "Securing matrix counting-based secret-sharing involving" Journal of King Saud University, 2022
2. Alshawi I. "Improved Salsa20 Stream Cipher Diffusion Based on Random Chaotic Maps." Informatica, 2022.
3. Chen X. et al. "Pseudorandom Number Generator Based on Three Kinds of Four-Wing Memristive Hyperchaotic Systems." Journal of Applied Mathematics, 2020.
4. Muhammed R.K. "Comparative Analysis of AES, Blowfish, Twofish, Salsa20" (arxiv, 2024)
5. Özpolat E. "Hyperchaotic System-Based PRNG and S-Box Design for a Secure Image Encryption Algorithm." PMC, 2025.