Journal of Intelligent Systems and Internet of Things
Vol. 17, No. 02, PP. 340-359, 2025
DOI: https://doi.org/10.54216/JISIoT.170222

A Secure and Efficient Novel Keystream Generator for Stream Ciphers

Chaithanya S.[1,*], Siddesh G. K.[2]

[1]Department of Electronics and Communication Engineering, Rajarajeswari College of Engineering, Bangalore, Karnataka, India

[2]Dean ECE-EEE, VidyaVikas Institute of Engineering and Technology, Mysore, Karnataka, India

Emails: chaithanya06@gmail.com; siddeshgundagatti@gmail.com

Abstract

The Internet of Things real-time communications depend on a secure stream of data. For the secure communications, a stream cipher with the features of ease and speediness is appropriate. The development and testing of a novel cryptographic algorithm with the goal of enhancing encryption performance. This paper introduces novel A matrix-based nonlinear pseudorandom key stream generation method inspired by the principles of fundamental recursive relationship of Reinforcement Learning, aiming to enhance diffusion and randomness in stream ciphers. We also incorporate the encryption approach based on the Counter based transformation of keystream generation (CBTKSG) method to enhance the speed, which is particularly well-suited for efficiently handling large file sizes since it delivers fast throughput. The technique was thoroughly bench marked and compared to other well-known encryption schemes. Performance has significantly improved without sacrificing security, according to the data. The keystream output was placed through the NIST SP 800-22 statistical test suite to verify its cryptographic strength. It passed every test with high p-values, indicating high randomness quality. The cipher has a strong avalanche effect, meets standard security criteria like IND-CPA and IND-CCA, and resists common cryptanalysis methods including related-key, differential, and linear attacks.

Received: January 02, 2025 Revised: March 02, 2025 Accepted: June 05, 2025

Keywords: Internet of Things; Encryption; Stream cipher; CBTKSG; Cryptography; Reinforcement Learning

1. Introduction

The Internet of Things [1] heavily drives the advancement of modern industrial processes. The creation of smart environments, transportation, environmental monitoring, integrated manufacturing, military and integrated health services are just a few of the technical sectors where IoT is crucial to advancement. Applications for smart homes, smart cities, health data management, incorporated manufacturing systems, and other systems that need sensors can be created using a range of these technology. Radio frequency identification (RFID), cloud computing, middleware, wireless sensor networks (WSN), and Internet of Things application software are the five fundamental categories of IoT technologies that are presently being effectively implemented. A crucial component of IoT security is confidentiality, which guards against unwanted access to or disclosure of private information. It's crucial because hackers may find it simple to target IoT devices in order to disrupt operations or obtain access to more extensive networks.[2] The Internet of Things envisage a world where ordinary objects exchange data, enabling users to make informed choices in a given situation. Although IoT has many benefits, its qualities making it incredibly susceptible to security breaches. IoT devices have limited computing and energy resources, making it difficult to apply contemporary ciphers, despite their potential to thwart assaults.[3] Because cloud computing involves outsourcing, security is a significant concern without a strong security strategy in place, cloud systems will be open to many types of assaults and user vulnerability. Data security is ensured by a number of security elements, including availability, confidentiality, integrity, authorization, and authentication. The three most crucial areas are authenticity, integrity, and secrecy. Since there are unknown dangers, attacks, and vulnerabilities, there

can never be complete security assurance. Cryptography is used to secure data during transmission, whether it be electronic or physical. The increasing demand for information confidentiality necessitates the development of new encryption techniques and algorithms. [4] The security of encrypted data can only be guaranteed by the encryption mechanism's strength and the key's confidentiality. These algorithms must be blazingly quick and sufficiently safe to avoid resource waste on low-resource devices. The field of cryptography is still in its infancy as academics try to develop a robust encryption method that keeps hackers from decoding the encrypted communication [5].

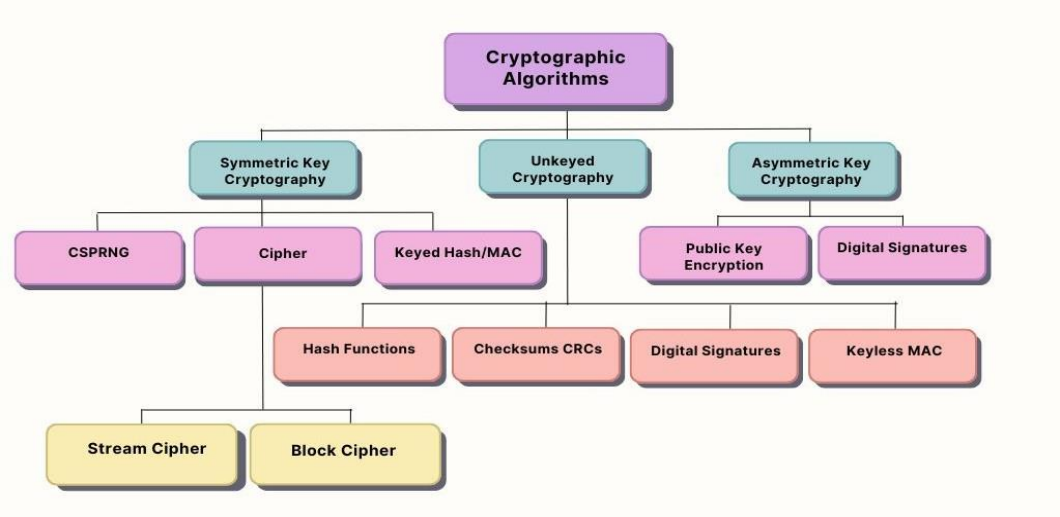


Figure 1. Different approaches of Cryptographic Algorithms.

As the figure.1 illustrates, there are three different methods in the realm of cryptography. One Symmetric encryption encrypts and decrypts data using a single key. Asymmetric key encryption uses two different keys: a public key that is shared with the public and a private key that is kept hidden. These techniques rely on mathematical processes, and the unkeyed method operates without a secret key. Symmetric encryption comes in two varieties: stream and block ciphers. The size of the key being used determines how effective symmetric key encryption is. The two primary components of stream ciphers are a key stream and a mixing function. The primary component of stream cipher encryption is the keystream generator, and the mixing function is typically an XOR function. If a stream cipher generates a string of zeros, the ciphered stream will be identical to the original plaintext. [6] A block cipher is an encryption technique that processes a fixed-size block of plaintext at once, converting it into a cipher text block of identical size.

2. Stream Ciphers

Stream ciphers are symmetric encryption algorithms that are faster and more effective than block ciphers. They are used in many different applications, but most notably in real-time broadcasts and digital communications. The most common stream cipher model is cipher text is plaintext xor stream shown in figure 2.

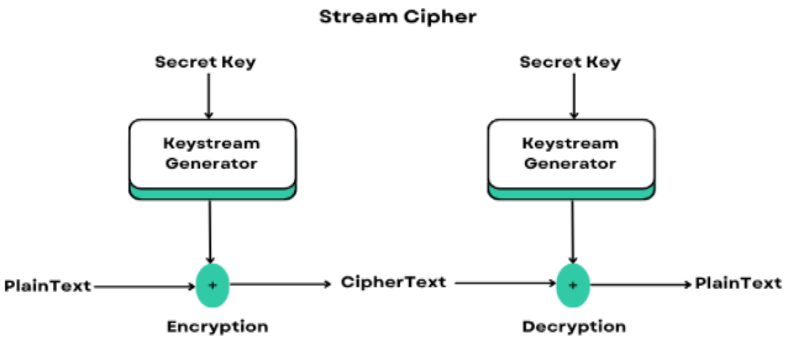


Figure 2. Stream Cipher

In order to encrypt a message, stream ciphers need to produce a lengthy stream of random numbers. Because this stream is generated so rapidly, it should be hard for an attacker to determine the sequence because it appears to be random. Their nature is synchronized and self-synchronizing. The previous cipher text has no bearing on the output of synchronous stream ciphers. Conversely, a stream cipher that self-synchronizes relies on the encrypted text that before it to produce its output. The location, key, nonce, and prior plain text (or, in the opposite case, cipher text) bytes all decide each byte in the streamA simpler instance of a stream cipher is one where the cipher text is a plain text xor stream, and the stream is determined by the nonce and the key. Neither the plain text nor the cipher text affects the keystream. This idea applies to any block cipher that runs in counter mode, such as Salsa20. Adding plain text to the stream more specifically, adding plain text bytes to the stream's subsequent byte is fundamentally justified by the fact that it allows message authentication at no cost. After the plain text has been encrypted, a specific amount of additional stream bytes can be produced to serve as an authenticator of the plain content. One counterargument is that it is a foolish exaggeration to say it is free. It takes time for each message to generate an authenticator and for each block in the stream to integrate the plain text. Another counterargument is that there is a security risk when plain text is added to the cipher state, requiring more communication from the cryptanalyst. Regaining the initial level of confidence requires adding rounds, and each block takes longer. Another counterargument is that contemporary 128-bit Wegman-Carter MACs, such as Poly1305, need only a few cycles per byte. Because it avoids the expense of decryption, this is significantly less costly than ”free” authentication for packets that have been tampered with, even though it still costs for authentic packets.

3. Related Work

The unpredictable nature of the keystream is crucial to the security of stream ciphers. Since a stream cipher encrypts data by fusing plain text with a pseudorandom key stream, the quality and uniqueness of this key stream are crucial to integrity and secrecy of the encrypted data. Random numbers are crucial for network security, as widely acknowledged. [7] They appear to be values derived from independent realizations of a U (0, 1) random variable (r.v.). Cryptography states that the length of the key stream and the message should be identical. Two chaotic logistic maps in progress side by side and beginning with random independent initial conditions form the basis of the pseudo random bit generator. To address the drawbacks of logistic maps in creating chaos-based ciphers, piece-wise logistic maps are introduced. The PLM serves as the foundation for a brand-new pseudorandom number generator. Hybrid chaotic system with analog and digital components is used as the basis for a stream cipher system. By fulfilling the criteria outlined in Wiggins’ definition of chaos combining a chaotic strategy with a three-dimensional Chen chaotic system. [8] A customized logistic map in three dimensions is suggested to address the shortcomings of minimal complexity, constrained parameter space, and unequal distribution by concentrating on the features of low-dimensional chaotic maps. Compact chaotic keystream generators that utilize counter-based basic one-dimensional chaotic maps. [9]The suggested chaotic scheme, known as enlarged chaos, enhances the basic 1D chaos’s chaotic range limit. LDMLNCML’s spatiotemporal chaos is used to propose a new sensitive image encryption technique. [10] Using random diffusion, the method begins with a pending sequence determined by the number of pixels in the image. A conflict-handling approach is used to produce two sensitive index chains. Diffusion involves chaotic interference and two random non-adjacent pixels that affect each pixel’s cipher value. There might not be enough randomization when only one chaotic map is used. [11] Generalized symmetric chaotic maps and adaptive control settings provide consistent chaotic behaviour. In order to increase complexity, it uses a coupled map lattice structure in which each local map communicates with its neighbours through a coupling factor. [12] Three-dimensional CML exhibits more complex chaotic behaviour than one-dimensional and two-dimensional CML. Based on the 2D nonadjacent-coupled map lattice model designed, [13] a new high dimensional spatiotemporal chaotic system is put forth. Several random lattices in various dimensions have an impact on each local lattice, and the local map is described as a fractional fashion. The random behaviour of lightning strikes served as the inspiration for the Strike cipher. [14] Strike has three primary functions: one for creating dynamic strikes and turning it into a keystream, another for plaintext encryption, and a function for configuring security properties. [15] Dawson's Summation Generator as the Basis for a Composite Chaotic Keystream Generator. Combines multiple 1D chaotic maps with a Dawson Summation Generator (DSG), achieving excellent randomness for real-time IoT applications. [16] Integrates chaotic dynamics into the present cipher for improved diffusion and low resource usage. [17]Proposes a lightweight Zu Chongzhi (ZUC) variant enhanced with chaotic systems, improving unpredictability and key generation dynamics.

The Salsa20 hash algorithm combines 16 bytes of constants; a 32-byte key, an 8-byte block counter, and an 8-byte nonce make up its initial 16 words. After that, it carries out 320 Indestructible word alterations in a certain layout, every single of which xor’s a rotational sum of two other words into one word. Lastly, the resultant 16 words are added to the initial 16 words. The enhanced version of salsa20 is Cha-cha [18], ChaCha20 requires a 256-bit key. A new matrix generation method enhances speed on SIMD platforms and diffusion [19]. RC4 consists Key-Scheduling Algorithm to initialize the s-box and PRGA to generate the keystream. [20]RC4 can accept keys ranging from 8 to 2048 bits in length, and the period is determined by the key. The Blowfish works with 64-bit blocks and various key lengths ranging from 32 to 448 bits. Although it is small block size makes it less appropriate

for contemporary systems, it is renowned for its ease of use, speed, and robust Security. [21] While ECC has been successfully broken, Blowfish has not yet been broken. [22] Suggested alteration to the Twofish encryption algorithm already in use. Higher performance is guaranteed by the Twofish algorithm's recommended minimum latency. AES is a NIST-standardized symmetric block cipher that encrypts data in fixed 128-bit blocks utilizing key sizes of 128, 192, or 256 bits. Because of its high security, effectiveness, and defense against well-known cryptographic threats [23], it is widely utilized. Although SPECK, a lightweight block cipher created by the NSA [24], is tailored for high-speed software performance, there is ongoing discussion on its security and uptake. [25]ASCON is appropriate for devices with limited resources since it offers good resistance against cryptanalysis and lightweight authenticated encryption and hashing.

4. A Novel Keystream Generation Mechanism

The prime goal of this endeavor is on the creation of random numbers. The Novel Key stream generation method was inspired by the Bellman equation, which offers the mathematical underpinnings of reinforcement learning. The development process was directed by the recursive relationship that is inherent in the concepts of reinforcement learning, aiming to enhance diffusion and randomness in stream ciphers. A matrix-based nonlinear pseudo random key stream generator that uses delta-based amplification and row-wise diffusion to generate secure stream ciphers. The algorithm initializes a 16-element state array to hold the values needed for key stream generation. The following make up the state- four constant words, assist structure the initial state and prevent key-nonce collisions, making it more resilient to attacks such as differential cryptanalysis. Eight words of the key, i.e. 256-bit key, two nonce words, which are used to guarantee distinct encryption instances and One word of the counter, which is used to guarantee that each block has a distinct key stream and also used to assist transformation count. State array given to the key stream generator is depicted as an array of four by four consists of 32-bit words, written as:

$$S = \begin{bmatrix} C_0 & C_1 & C_2 & C_3 \\ K_0 & K_1 & K_2 & K_3 \\ K_4 & K_5 & K_6 & K_7 \\ \delta & N_0 & N_1 & N_2 \end{bmatrix} \quad (1)$$

The transformation function of key generation is as follows. All elements in state row i and column j are updated by the value Γ a for all the values of i and j.

$$S'_{ij} = S_{ij} + \Gamma, \forall i, j \in \{0,1,2,3\} \quad (2)$$

Where S'_{ij} represents the updated matrix element. Γ Is calculated as the sum of the counter value, ʌ, and the sum of the i^{th} row elements.

$$\Gamma = \sum_{x \in \{T, \Lambda, R_i\}} x (\mathrm{mod} 2^{32}) \quad (3)$$

By multiplying the RC_{max} with the key length to obtain δ. The purpose of multiplying RC_{max} by key length is to obtain a random sequence, even if the algorithm does not use constants or nonce.

$$\Lambda = L_k * RC_{\max} \quad (4)$$

RC_{max} is the maximum element of row indexed by column of S_{ij}

$$RC_{\max} = \max(S_{j\_row}) \quad (5)$$

The sum of the elements in i^{th} row that were applied to create Γ is known as the row sum.

$$R_i = \sum_{j=0}^{3} S_{i,j} \quad (6)$$

All state elements undergo bitwise rotation on the integer n by d positions once the State is updated to S". In the present instance, we allocated d=17 bits. By carrying out this transformation five times and beyond, we are producing a key stream of 64-bit sequences.

$$S'' = S' \ll d \quad (7)$$

The following algorithms are used to illustrate the suggested algorithm.

Algorithm 1: State Initialization

Input:

256-bit key K=(K0,K1,…,K7)

96-bit nonce N=(N0,N1,N2)

32-bit block counter δ

Define Constants: C0← 0x766F7270, C1← 0x20656469, C2← 0x75636573, C3← 0x79746972

Nonce Generation:

N←CSPRNG(n) Create a random bit string of length n using a cryptographically secure random number generator

Initial State Representation:

C → S[0…3]	▷Assign Constants
K → S[4…11]	▷Load 256 Key
δ → S[12]	▷ Initialize block Counter
N → S[13…15]	▷ Assign 96-bit Nonce
Save S	▷ Save Initial State

Output:

Initial 16-word state S[0…15]

Algorithm 2: Transformation Function

Transform State (S):

Reshape the 1D state S into a 4 × 4 matrix.

$R_i \leftarrow$ sum($S_{i,j}$), from j = 0 to 3

$RC_{max} \leftarrow$ max($S_{j,i}$) from i=0 to 3 ▷ Maximum element in row indexed by column j

Compute ∧

Find Γ

Update the element:

$$S[:] \leftarrow (S[:] + \Gamma) \mod 2$$

S[:]← RSHFTC(n, d, N)

Flatten the 4X4 matrix back into a 1D array S

Return the transformed state S.

5. Counter-Based Transformation of Key Stream Generation for Stream Cipher (CBTKSG-SC)

The Novel Key stream Generation mechanism serves as a key stream generator for encryption and Decryption. Symmetric ciphers use an identical key for both encryption and decryption. As a result, in cases of communication that employ symmetric key cryptography, both parties must be aware of the key. Symmetric ciphers are characterized by the functions (E, D), where: E and D represent Algorithms for encryption and decryption, respectively. Let K represents the key space, M denotes the message space, and C signifies the cipher text space. A general encryption system can be depicted as

$$E : K * M \rightarrow C \quad (8)$$

Where the $*$ symbol indicates that K and M are mapped to C via the function E. where The encryption feature E accepts a plaintext message m ϵ M and a key k ϵ K. and produces a cipher text c ϵ C:

$$E_k(m) = c \quad (9)$$

Similar to that, the decryption function is as follows:

$$D : K * C \rightarrow M \quad (10)$$

$$D_k(c) = m \quad (11)$$

$$D_k(E_k(m)) = m, \ \forall m \in M, \forall k \in K \quad (12)$$

As said above these ciphers usually encrypt data either bit-by-bit or byte by byte by forming a pseudo random keystream and merging it with the plain text. Modern stream ciphers work with data in 64-byte blocks, in contrast to traditional stream ciphers. They are perfect for modern cryptographic applications because of their architecture, which allows them to be processed effectively in parallel. Since the key stream is still generated in a pseudo-random fashion and merged with the plain text using XOR, the fundamental idea of stream ciphers is maintained. A certain process is repeated multiple times in most ciphers. In general, more rounds of the ciphers boost security but also take more time, even though some rounds are more powerful than others are. Large-scale data encryption and decryption operations can demand a lot of processing power, particularly when employing robust encryption techniques. This may result in considerable delays regarding data processing and access. In time-sensitive applications, the latency introduced by real-time encryption and decryption of large datasets may be unacceptable. The processes of encryption can utilize considerable CPU, memory, and storage resources, which may affect other operations of the system. In order to resolve this issue, here we implemented a counter-based transformation. To reduce computational cost, it reuses the previous state rather than initializing from the beginning. The transformation is applied five times in the first block and once in each of the following blocks shown in figure.3. Despite encrypting large file size it takes less time, compare to the other existing algorithms.

Here, we present a new Counter-Based Transformation for keystream generation algorithm (CBTKSG) used in stream cipher. Let (Pi,Ci) represent plaintext and ciphertext, such that P = $p_1 \| p_2 \| ...$ symbolizes the plaintext, divided into blocks of 64 bytes Pi. .The ciphertext is represented as: C = $c_1 \| c_2 \| ...$ where each ci corresponds to a 64-byte block of ciphertext. Every block of plaintext p_i is converted into a block of ciphertext c_i based on an encryption function. The CBTKS for stream cipher,- CBTKS: $\{0,1\}^{256} \times \{0,1\}^{96} \times \{0,1\}^* \rightarrow \{0,1\}^*$, mutate a plaintext P to a ciphertext C by employing a 12-byte nonce N and a 32 byte secret key K with constant and block counter. Inwardly, it can utilizes the block function:

CBT: $(0,1)^{512} \leftarrow *(0,1)^{128} * (0,1)^{256} * (0,1)^{32} * (0,1)^{96}$

Which, given key, nonce and block counter δ, produces a 64-byte key stream block k_δ^N. This process resembles a block cipher in counter-mode operation:

The plaintext P is divided into 64-byte segments (p_i): $0 \leq i < 2^{32}$

Every block of plaintext is encrypted by XOR-ing it with the corresponding keystream block k_i^N , yielding ciphertext blocks:

$$c_i = p_i \oplus k_i^N \quad (13)$$

Decryption follows the same principle in reverse:

$$p_i = c_i \oplus k_i^N \tag{14}$$

This approach ensures efficient also parallelize encryption and decryption.

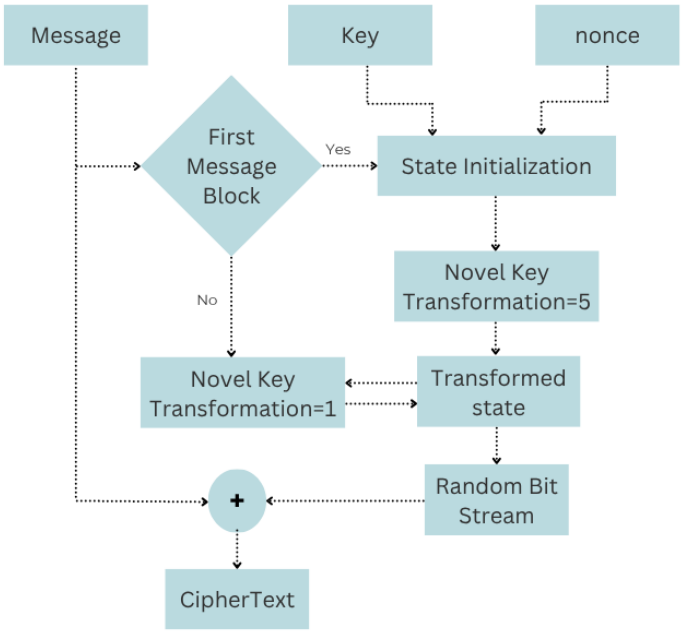


Figure 3. Counter-Based Transformation of Key Stream Generation for Stream Cipher

---

Algorithm 3: CBTKSG-SC: C = CBTKSG-SC(K, N, P)

Input : K Є F_2^{256}, N Є F_2^{96}, P Є F_2^*

Output : C Є $F_2^{|P|}$

▷ where F2 is the two-element finite field {0, 1}

for i from 0 to $\frac{|P|}{512} - 1$ **do** Keystream Generation:

for counter ← 0 to number of samples−1 do

if counter == 0 **then**

Apply the transformation 5 times:

for i ← 1 to 5 do

S ←Transform State(S)

end for

else

Apply the transformation once on state:

S ←Transform State(S)

end if

Pack the state S into 64 bytes to form the key Stream.

end for

Ki N ← S

Ci ← Pi ⊕ Ki N

end for

Return C

6. Experimental Results and Analysis

6.1. Performance Analysis

To evaluate and compare the performance of the encryption algorithms, a custom simulation was developed using the Python programming language. All algorithm were implemented in Python IDLE 3.13 and tested on a Windows 10 Home (64-bit) system featuring 4 GB of RAM and an Intel® CoreTM i5-2520M CPU running at 2.50 GHz. The rate at which a set of stream ciphers produces the key stream is a critical component of the entire encryption mechanism, as stream ciphers are mostly reliant on key stream generators. The running speed for producing a 1 MB binary stream using each of these techniques is listed in Table.1. The fastest speed achieved using the suggested Novel Keystream Generator is 24.67 Mbps. The suggested approach is the best overall, showing excellent throughput efficiency. With its competitive performance and top ranking among the compared PRNGs, the suggested approach is appropriate for high-speed cryptography applications. The encryption of the suggested CBTKSG cipher is compared to that of other ciphers in the following.

Table 1: Running Speed Comparison of Different keystreams/PRNGs

Method / Reference	Speed (MBps)
Proposed	24.67
Zhuo Liu [12]	23.85
X Huang [26]	21.5054
MAS AL-khatib [27]	-
LFSR[27]	20.51
LCG[27]	12.59
BBS[27]	1.41
SLA-LFSR[27]	2.13
Y Sun [13]	18.599
Khaled Suwais [14]	22.796
Wang, Y[28]	12.8

The Table.2. represents encryption time (in seconds) for a range of file sizes utilizing several encryption methods, such as the AES (CBC Mode), Twofish, Blowfish, ChaCha20, RC4, and Salsa20 with Proposed Cipher is shown in the table below. These metrics aid in assessing each algorithm's effectiveness and performance in terms of speed and scalability. In terms of encryption time, the suggested CBTKSG cipher turns out to be very effective. It performs better than more

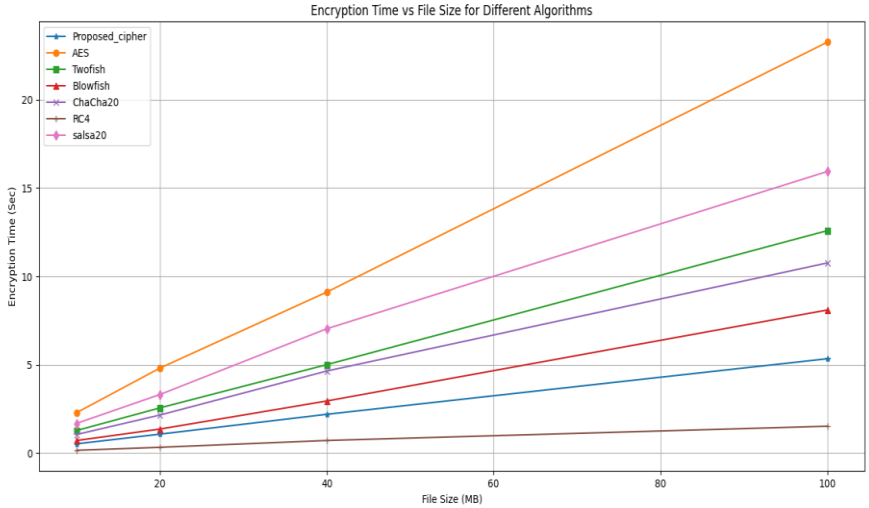
contemporary stream ciphers like ChaCha20 and Salsa20, as well as more conventional ciphers like AES and Twofish, and scales well with growing file sizes is shown in figure.4(a). This demonstrates its possible applicability in settings with limited resources and time-sensitive applications. Table 3. Represents decryption time for a range of file sizes for different algorithms shown in figure. 4(b).

Table 2: Encryption Time (in seconds) for Different Ciphers vs File Size

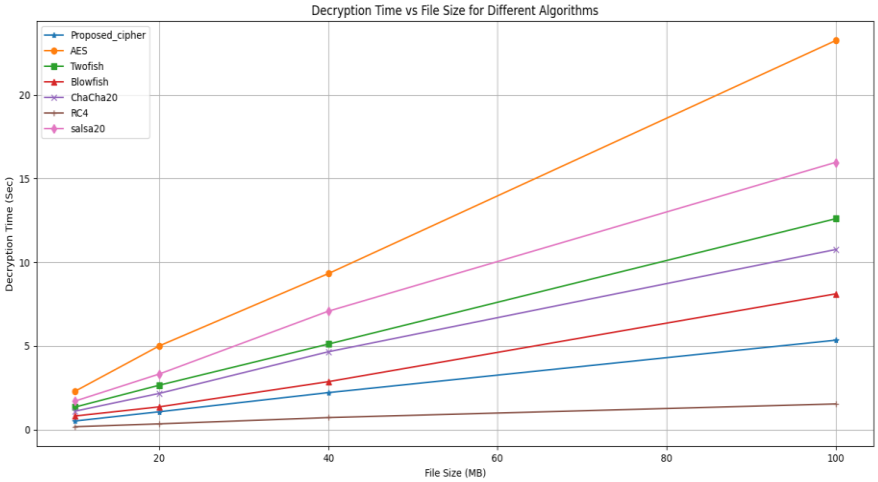
File size (MB)	Proposed	AES	Twofish	BlowFish	ChaCha20	RC4	Salsa20
10	0.52976	2.2911	1.2803	0.7161	1.0556	0.1631	1.6835
20	1.0766	4.816	2.5634	1.365	2.1581	0.3332	3.31898
40	2.2008	9.1168	5.0106	2.954	4.6494	0.7196	7.0413
100	5.3459	23.2603	12.5937	8.1032	10.766	1.526	15.9425

Table 3: Decryption Time (in seconds) for Different Ciphers vs File Size

File size (MB)	Proposed	AES	Twofish	BlowFish	ChaCha20	RC4	Salsa20
10	0.51354	2.2812	1.3412	0.8112	1.097	0.1725	1.6975
20	1.06782	5.002	2.6534	1.3598	2.1632	0.3434	3.3213
40	2.2103	9.3212	5.1104	2.865	4.6512	0.7195	7.0753
100	5.3425	23.2416	12.5981	8.1098	10.759	1.534	15.9654



(a). Encryption time of various algorithms



(b). Decryption time of various algorithms

Figure 4. Encryption and Decryption time for Different Ciphers vs File Size

6.2. Implementation on Raspberry Pi 4

The proposed algorithm was implemented on a Raspberry Pi 4 Model B, which is powered by a Broadcom BCM2711 SoC with a quad-core ARM Cortex-A72 64-bit processor operating at 1.5 GHz, LPDDR4 memory, and support for high-speed I/O interfaces. We performed encryption and decryption to evaluate the algorithm's performance on hardware with limited resources. The files of different sizes were encrypted and decrypted, and key performance metrics like execution time, throughput, memory usage, and CPU utilization were measured.

Table 4: Performance Metrics of proposed algorithm on Raspberry Pi 4

Data Size (MB)	Enc. Time (s)	Dec. Time (s)	Throughput (Mbps)	Memory Current (MB)	Memory Peak (MB)	CPU Avg (Enc)	CPU Avg (Dec)
0.98	0.01151	0.01227	728.56	1.96	1.96	25.20%	26.10%
9.77	0.12042	0.11827	696.6	19.53	19.53	25.10%	25.00%
97.66	1.17901	1.2445	711.49	195.31	195.31	25.00%	24.80%
976.56	12.8874	19.1290	650.91	1953.13	1953.13	24.80%	24.90%

The symmetric computational complexity of the solution is confirmed by the testing results on a Raspberry Pi 4, which demonstrate that the encryption and decryption times are almost the same for all dataset sizes given in table .4. The obtained throughput shows consistent and scalable performance as the data size grows, ranging from 728.56 Mbps for smaller files to roughly 650.91 Mbps for bigger ones. A 976 MB dataset's memory utilization shows a linear development pattern, roughly doubling the size of the original file to reach 1.95 GB. CPU use, on the other hand, stays steady at about 25% per core,

Overall, the findings show that the cryptographic solution runs smoothly on the Raspberry Pi 4, providing predictable memory scalability, fast throughput, and moderate CPU utilization. These results validate the approach's appropriateness for edge computing and Internet of Things applications requiring lightweight cryptography.

7. Security Analysis

The evaluation of the suggested CBTKSG-SC encryption scheme's security features is the objective of this section. The final goal is to thoroughly inspect the security assurances. By giving a comprehensive evaluation of CBTKSG-SC's security features and shedding light on both its advantages and disadvantages, this contributes to the field of cryptographic research.

7.1 Avalanche Test

The results of the avalanche test indicate that, on average, half of the output bits altered when a one bit in the key is flipped, which is perfect for a secure cipher. Table.5. Indicates Strong diffusion and sensitivity to input changes are indicated by the fact that approximately 50.14% of the 512 output bits are changed. The observed change ranged from roughly 44.92% at the least to 55.86% at the maximum, indicating some natural variation but no weak places with extremely small changes. All things considered, this shows that your solution successfully uses the avalanche effect, which makes it impervious to assaults that take advantage of inadequate bit diffusion.

Table 5: Avalanche Effect

Metric	Changed Bits	Percentage (%)
Average changed bits	256.70	50.14
Minimum changed bits	230	44.92
Maximum changed bits	286	55.86

7.2 Correlation Coefficient

The linear relationship between two variables is measured by the Pearson Correlation Coefficient (ρ) and is given by

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} \qquad (15)$$

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} \tag{16}$$

Where $\mathrm{Cov}(X,Y)$ is covariance given by,

$$\mathrm{Cov}(X,Y) = E[(X-\mu_X)(Y-\mu_Y)] \tag{17}$$

Thus, the Pearson correlation:

$$\rho_{X,Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \tag{18}$$

σ_X, σ_Y are the standard deviations of X and Y is given by,

$$\sigma_X = \sqrt{\mathrm{Var}(X)} \tag{19}$$

$$\sigma_Y = \sqrt{\mathrm{Var}(Y)} \tag{20}$$

$$\rho_{X,Y} = 0.000017$$

The Pearson Correlation Coefficient (ρ) was calculated in order to assess the bit stream generated by the suggested Key stream statistical independence. it results, Pearson ρ = 0.000017 There is no discernible linear link between neighbouring bits, as the calculated correlation value is quite near to 0. This implies that high statistical independence, a crucial characteristic for safe cryptography applications, is present in the key stream output.

7.3 Global Bit Balance Analysis

The raw number of zeros and ones throughout the whole keystream segment under analysis is shown in this table.6. A measurement of the absolute percentage departure from the ideal 50% distribution is called the Disequilibrium Degree:

$$\text{Disequilibrium Degree} = \frac{|b_0 - b_1|}{b_0 + b_1} x100\% \tag{21}$$

b_0, b_1 denote the counts of zeros and ones, respectively,
The decreasing percentage in table.4 indicates that when the sample size grows, the keystream approaches an ideally equitable distribution of zeros and ones, consistent with the law of large numbers.

Table 6: Global distribution of bits in proposed keystream segments

Total Bits	Number of 0s	Number of 1s	Disequilibrium Degree (%)
10,000	5,026	4,974	0.52
50,000	25,102	24,898	0.41
100,000	49,805	50,195	0.39
1,000,000	498,906	501,094	0.22

7.4 Equilibrium Degree Analyse

The keystream is divided into multiple blocks of fixed size for each block i, the balance ratio r_i is calculated as,

$$\overline{r_i} = \frac{\text{number of ones in block i}}{\text{block size}} \tag{22}$$

The statistical measures are then computed as follows: Mean balance ratio:

$$\overline{r_i} = \frac{1}{N}\sum_{i=1}^{N} ri \tag{23}$$

Standard deviation:

$$\sigma_r = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(r_i - \overline{r_i})^2} \quad (24)$$

Equilibrium Degree (ED):

$$ED = 1 - \frac{\sigma_r}{\overline{r}} \quad (25)$$

Where N stands for the total number of blocks.

For this investigation, a 51,200,000-bit keystream was divided into 51,200 blocks, each consisting of 1,000 bits. An equal distribution of bits across blocks is shown by the low standard deviation of 0.015218, and the mean balance ratio of 0.498948 is quite close to the ideal 0.5. The high Equilibrium Degree of 0.969869, which shows no obvious bias, confirms global stability. The keystream's remarkable unpredictability and general homogeneity make it suitable for applications involving cryptography and secure communication.

7.5 Statistical Analysis

The statistical test suite NIST SP 800-22 was utilized to evaluate the keystream's randomness quality created by the suggested CBTKSG stream cipher. The purpose of this suite of hypothesis tests is to evaluate several facets of randomness in a binary sequence. The following hypotheses are assessed by each test in the suite: The sequence under test is random, according to the null hypothesis (H0). The sequence under test is not random, according to the alternative hypothesis (H1). For every test, a *p-value* is calculated, which is the likelihood that a perfect random number generator would produce a less arbitrary sequence than the one under examination. The specified significance level $\alpha = 0.01$ serves as the foundation for the decision rule: A $p-value \geq \alpha$ indicates that we are unable to reject H0, indicating that the test is successful and the series looks to be random. The sequence might not be random if the $p-value < \alpha$, which means we reject H_0. For every test, we are unable to reject the null hypothesis because the *p-values* above are all much higher than the significance level ($\alpha = 0.01$) according to table.7. This indicates that the proposed keystream does not appear to vary from predicted randomness. Consequently, we deduce that the keystream generated by proposed algorithm demonstrates significant randomness characteristics and fulfils all of the NIST SP 800-22 statistical tests. This demonstrates that proposed algorithm is cryptographically resistant in terms of statistical unpredictability, which is an essential characteristic for stream ciphers used in secure communications.

Table 7: Results of the NIST SP 800-22 Test for CBTKSG Keystream

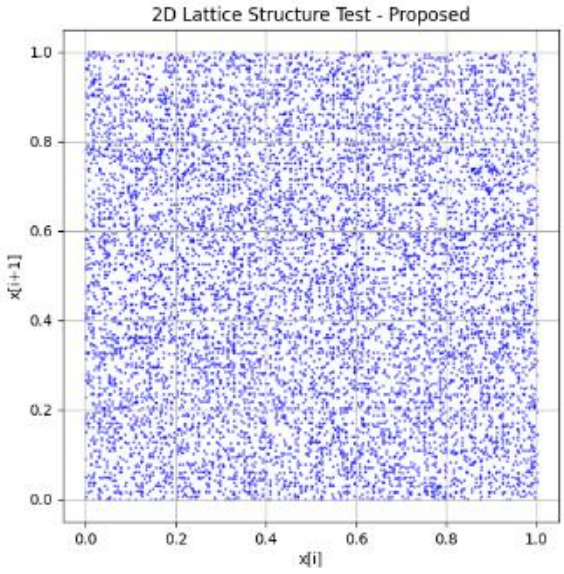
Statistical Test	P-value
Frequency (Monobit)	0.5117
Runs	0.4834
Longest Runs of Ones	0.5041
Discrete Fourier Transform	0.4869
Rank	0.5240
Block Frequency	0.4788
Overlapping Templates Matching	0.4580
Non-Overlapping Templates Matching	0.4967
Universal Statistical Test	0.4877
Serial	0.4030
Linear Complexity	0.5429
Cumulative Sums	0.3820
Approximate Entropy	0.4719
Random Excursions	0.4359
Random Excursions Variant	0.4365

7.6 Graphical Test

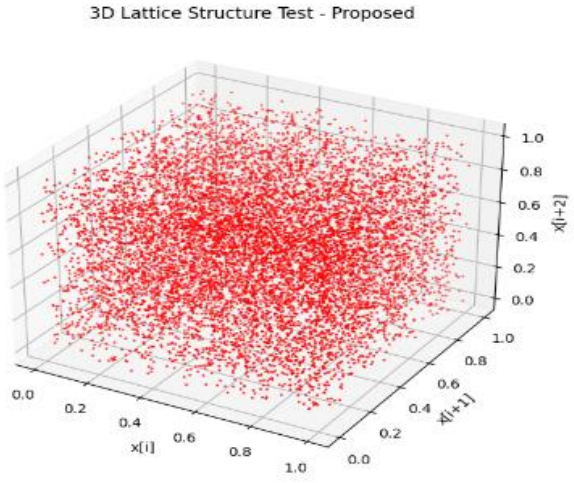
The distribution and organization of the keystream or random sequence are visualized through graphical testing. They aid in the comprehension of patterns that may not be apparent when using only numerical testing. There are a few chances of getting Type Type II errors or I in statistical tests leading to the incorrect result. Consequently, it is helpful to examine how the numbers appear on two-dimensional and three-dimensional plots. Plotting the numbers on a graph allows graphical testing to determine whether a pattern can be identified.

7.6.1. Lattice Test

The independence and randomness of a pseudorandom number generator are assessed using the Lattice Test. Examines the appearance of randomly distributed numerical sequences plotted in an n-dimensional space. This test determines whether there are patterns formed by random numbers. This is tested by pairing and plotting successive random numbers that are created from a seed. These normalized numbers are subjected to two different kinds of lattice test: the Lattice test in two dimensions (2D), which uses a point made up of consecutive integers, When viewed as a point on a plan, the two neighbouring random numbers (x_i, x_{i+1}) combine to form a lattice [13] and Lattice test in three dimensions (3D), which uses three consecutive numbers to make a point.



a. 2D Proposed



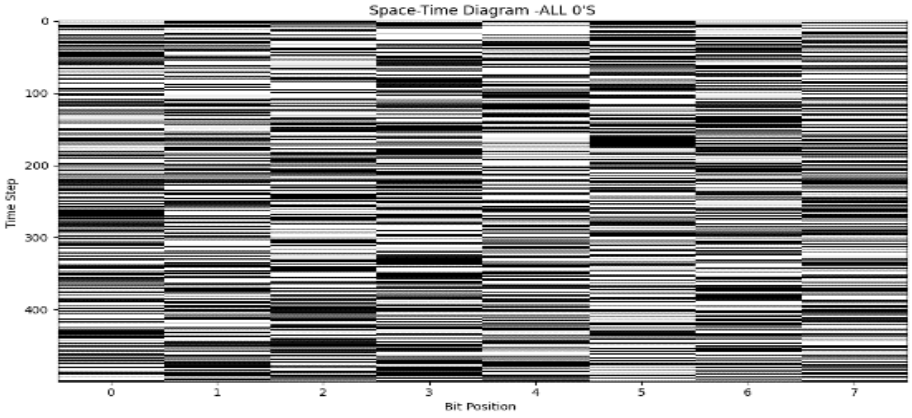
b. 2D Proposed

Figure 5. Lattice (2D & 3D) Tests of Proposed Algorithm

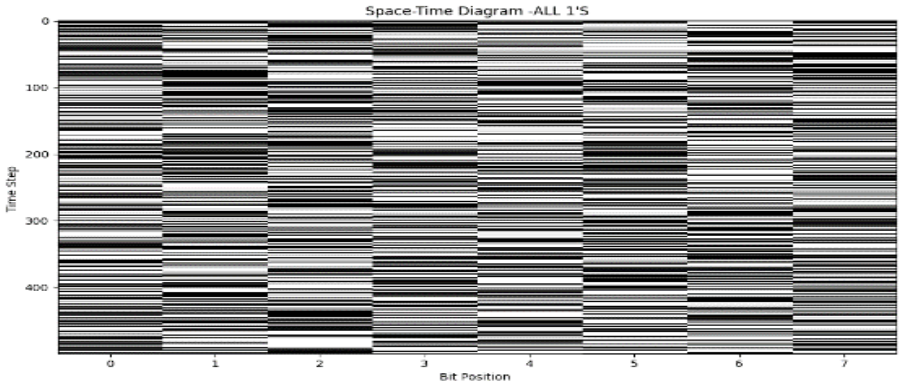
The points in 3D space may line up on parallel planes if a PRNG has weaknesses exposing the randomness's vulnerabilities. The charts display patterns if there is a correlation between the random integers. The two-dimensional and three-dimensional lattice tests shown in figure.5, No discernible alignment should be seen in the 2D or 3D lattice tests of a proposed algorithm contains 10,000 distinct points

7.6.2. Space–time diagram

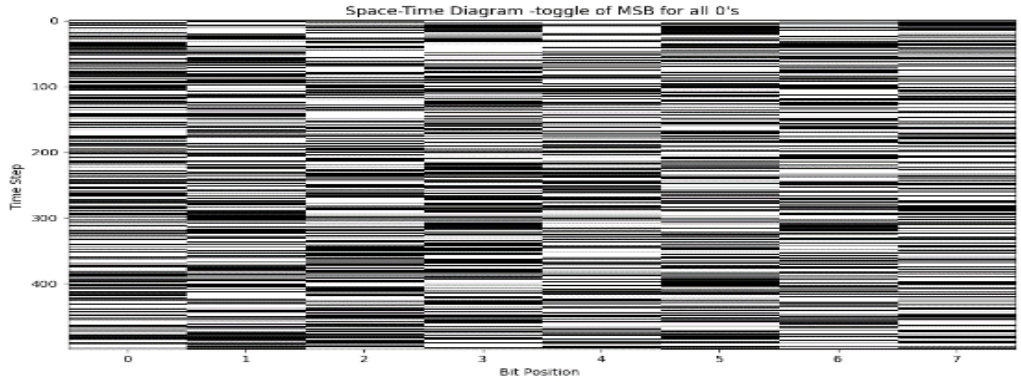
Another visual aid for examining how a system changes over time is a space-time diagram, this tool as a practical way to gauge a PRNG's randomness. The y-axis of a diagram shows time, while the x-axis displays a number created at any given moment. A PRNG with a space-time diagram requires non-normalized integers for testing. There must be b distinct colours to represent a number if it is in base b, with each colour denoting a specific base digit. A series of numbers is created over time t, starting with a seed, and every number is plotted in contrast to t. If a pattern appears in any area of a number or among any successive numbers, it is evident from this diagram, since colours highlight the pattern. The PRNG has a strong randomness quality if the numbers are noisy in colour and lack a pattern. Thus, a comprehensive understanding of a PRNG's randomness qualities may be built using this figure. The evolution of the bit locations over time for various initial conditions and adjustments is depicted in each space-time figure. The visuals shed light on the structure and randomness of bit changes in various scenarios. All 0's All of the bits in this situation are initially set to 0. Over time, the black and white variations show changes from 0's to 1s. Strong diffusion features are suggested by a high degree of randomness. All 1's In this case, all of the input bits are initially set to 1. The distribution of black and white pixels denotes bit transitions, just like in the first diagram. Similar randomness to that shown in the first diagram indicates that the algorithm behaves uniformly. By flipping LSB or MSB bits in the all 0's and all 1's it obtains space diagram as shown in figure.6.(c-f). This evaluates how sensitive the algorithm is to minor adjustments. Display localized changes that spread over time because only the LSB is toggled from an all-1 state. The system efficiently amplifies minor adjustments. Wide-ranging changes over various bit positions and time steps are caused by flipping one bit in the input. A system has strong diffusion if a single bit change causes the bit stream to shift widely over time. The system treats various input patterns equally, guaranteeing consistency, uniform distribution of 1's and 0's. This also ideally exhibits significant variances from the corresponding all-0 or all-1 scenarios, suggesting that they are sensitive to little variation in input.



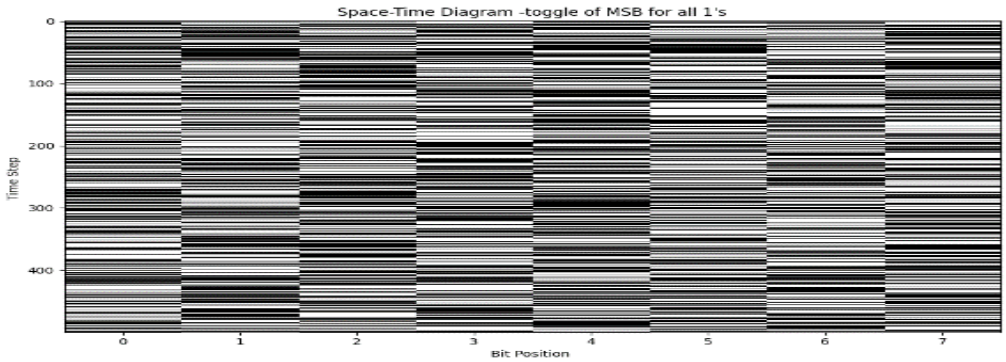
(a) All 0's



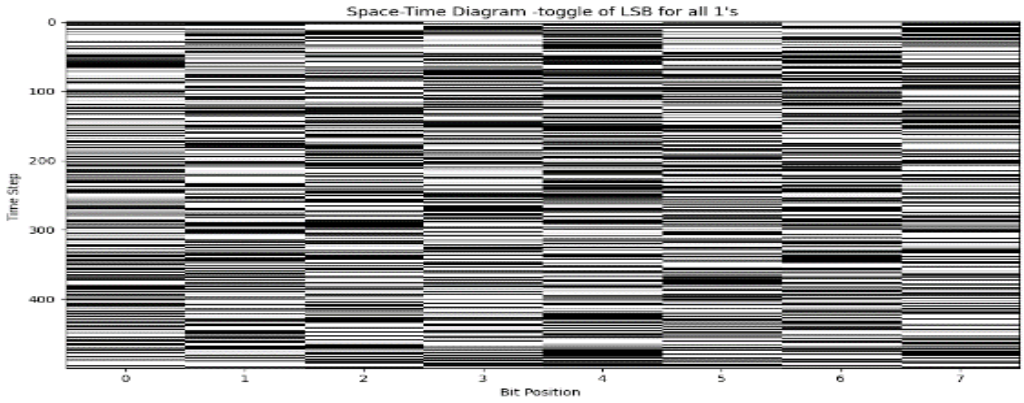
(b) All 1's



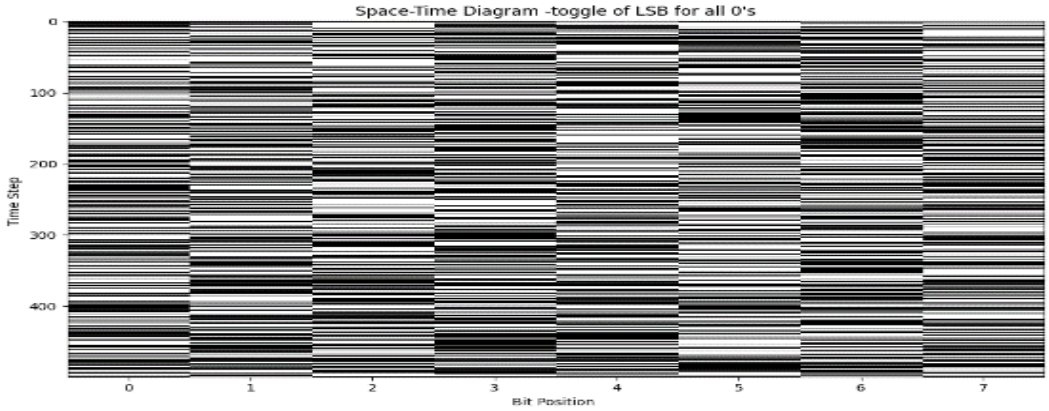
(c) Toggle of MSB bit in all 0's



(d) Toggle of MSB bit in all 1's



(e) Toggle of LSB bit in all 0's



(f) Toggle of LSB bit in all 1's

Figure 6. Space–Time diagrams under different input conditions using proposed algorithm

7.7 Bit Distribution Entropy analysis

Bit Distribution Plot shown in figure. 7 is to evaluate the degree to which the 1 bits are dispersed over the data's 8 bit locations (0–7).We anticipate an equitable distribution of 1 bits at every point for high randomness. Non-random patterns, which are undesirable for cryptography, may be indicated by skewed distributions. Entropy Plot over Blocks figure. 8 is to quantify the variation in uncertainty among data blocks. The maximal randomness, or ideal entropy, is near 8 bits/byte. Patterns or repetition are indicated by blocks with decreased entropy, which compromises encryption. Strong unpredictability, which is essential for cryptographic security, Overall entropy of 7.802 bits/byte is indicated a higher entropy around 8 bits/byte. If the bit distribution is uniform and the entropy is large, the charts indicate significant randomness. Figure.9 shows the normalized entropy for different key size. With increasing key size, this plot demonstrates that entropy efficiency approaches 1, suggesting near-optimal randomness for larger symbols. Normalized entropy of a source X is given by:

$$\eta(X) = \frac{H(X)}{H_{max}} = -\sum_{i=1}^{n} \frac{p(x_i)\log_b p(x_i)}{\log_b n} \tag{27}$$

Where: **η**(X): normalized Entropy, H(X): Shannon entropy of the source, $H_{max} = \log_b n$: Maximum possible entropy with n unique symbols, $p(x_i)$: Probability of symbol x_i.

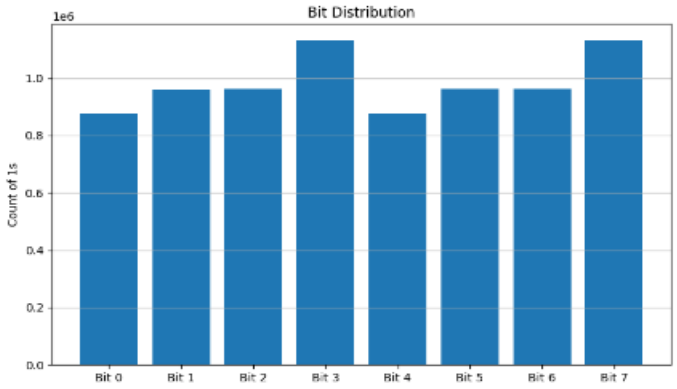


Figure 7. Bit distribution plot

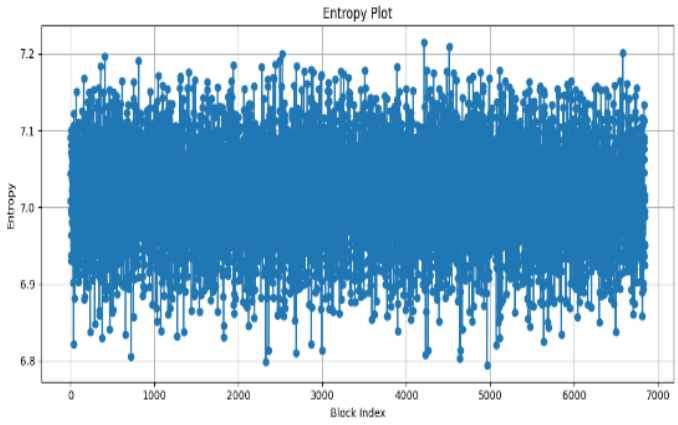


Figure 8. Entropy Plot over Blocks

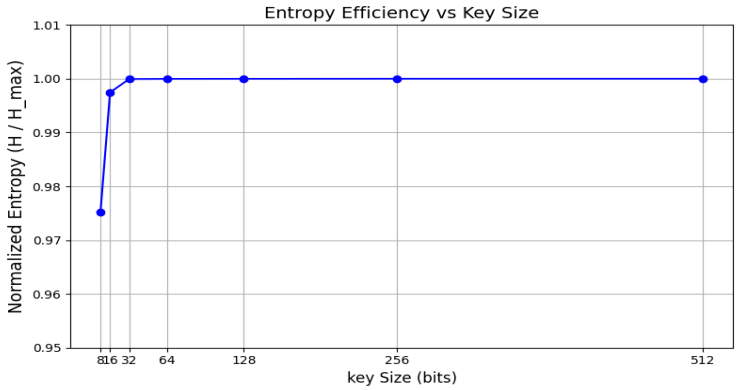


Figure 9. Entropy analysis of cryptographic keys

7.8 Differential cryptanalysis

A statistical analysis was conducted by introducing single-bit deviations in the input and measuring the ensuing variations in the output keystream in order to examine the diffusion qualities and resistance of the suggested innovative cipher to differential cryptanalysis. Using fixed key and 1000 rounds, the experiment introduced small variations in the plaintext. The results demonstrated that the generated 512-bit keystreams had an average of 253.18, with a minimum of 224 and a maximum of 281 different bits. An ideal cipher should, display the avalanche effect, in which each output bit has a 50% chance of flipping for every one-bit change in the input. For a 512-bit output, the expected number of differing bits is:

$\mathbb{E}[X] = n \cdot p = 512 \cdot 0.5 = 256,$

Where $X \sim B(n = 512, p = 0.5)$ follows a binomial distribution. The standard deviation σ of this distribution is:

$$\sigma = \sqrt{n \cdot p \cdot (1 - p)} = \sqrt{512 \cdot 0.5 \cdot 0.5} \approx 11.31$$

Hence, the expected statistical range within which most values should fall (with high confidence, using 2.8σ) is:

256 ± 2.8σ = 256 ± (2.8 x 11.31) ≈ 256 ± 31.67

=> Expected Range: [224.33, 287.67].

The cipher closely resembles the behaviour of an ideal cipher with strong diffusion qualities, as evidenced by the empirical range of observed different bits [224,281] falling well within this confidence interval.

7.9 Linear cryptanalysis

Let $Z = \text{bit}_0 \oplus \text{bit}_3$ be a linear combination of two output bits of the internal state of the cipher. We examine if bias affects this relationship. Define this event's probability across N encryptions as follows: Let $\Pr[Z = 1]$ denote the probability that the expression evaluates to 1. We estimate this over N=100,000 independent encryptions with randomly chosen keys and plaintexts:

$$\hat{p} = \Pr[\text{bit}_0 \oplus \text{bit}_3 = 1] = \frac{1}{N}\sum_{i=1}^{N} \left(\text{bit}_0^{(i)} \oplus \text{bit}_3^{(i)}\right) \tag{26}$$

From our experiments, we obtained: $\hat{p} = 0.50063$. The bias ϵ of the approximation is defined as:

$$\epsilon = \left|\hat{p} - \frac{1}{2}\right| = |0.50063 - 0.5| = 0.00063$$

The fact that ϵ is near 0 suggests that there is no discernible linear bias in the cipher.

7.10 Resistance to Related-Key Attacks

We conducted an experiment where a pair of encryption keys were generated that differed by exactly one bit in order to assess the suggested cipher's resilience to related-key attacks. The same nonce and plaintext were encrypted for every key pair, and the keystreams that were produced were compared. Over N = 100 iterations, the average Hamming distance between the two outputs was calculated. Let K and K′ be two keys such that:

$$K' = K \oplus 2^i \text{ for some } i \in [0,255] \quad (28)$$

Let CBTKSGC (K, P) denote the keystream output from the CBTKSG cipher with key K and plaintext P. Then, for each test iteration, the Hamming distance is computed as:

$$HD_i = \text{Hamming}\big(\text{E}(K,P), \text{E}(K',P)\big) \quad (29)$$

The average hamming distance over N tests is:

$$\overline{HD} = \frac{1}{N}\sum_{i=1}^{N} HD_i \quad (30)$$

In our experiments, we observed:

$$\overline{\text{HD}} \approx 253.98 \text{ bits}$$

The estimated average hamming distance for a cipher with perfect avalanche effect and full diffusion, given that the keystream output is 512 bits (64 bytes), is:

$$E[HD] = \frac{512}{2} = 256\ bits$$

The measured average Hamming distance of 253.98 bits is extremely near to the optimal value of 256 bits. This indicates that a single-bit difference in the key causes approximately half of the output bits to flip, confirming that the proposed cipher exhibits strong diffusion properties and is resistant to related-key attacks.

7.11. Indistinguishability under Chosen Plaintext Attack (IND-CPA)

If no effective adversary can differentiate between the ciphertext of two selected plaintexts with a probability substantially higher than 50%, the cipher is said to be IND-CPA secure.
The encryption strategy Π = (KeyGen, Enc, Dec) will be considered. A game between a challenger and a probabilistic polynomial-time adversary A defines IND CPA security:

- The challenger provides p_k to A and creates keys $(p_k,s_k) \leftarrow \text{KeyGen}(1^\lambda)$.
- An encryption oracle $\text{Enc}_{pk}(\cdot)$ may be queried polynomially many times by A.
- A selects m_0,m_1, two messages of equal length. A random bit $b \in \{0,1\}$ is sampled by the challenger, then returns the challenge ciphertext.

$$c^* \leftarrow \text{Enc}_{pk}(m_b) \quad (31)$$

- After repeatedly accessing the encryption oracle, A ultimately produces a guess b′.

The following describes the advantage of the opponent in this game:

$$Adv_\pi^{ind-cpa}(A) = \left|P_r[b' = b] - \frac{1}{2}\right| \quad (32)$$

If every probabilistic adversary A in polynomial time is secure. then an encryption system Π is IND-CPA secure.

$$Adv_\pi^{ind-cpa}(A) \leq \text{negl}(\lambda) \quad (33)$$

With 100,000 trials, the proposed cipher was evaluated using the IND-CPA model. The attacker's 50.05% success percentage is statistically comparable to that of random guessing (50%). This shows that the encryption does not leak differentiating information and is highly resistant to IND-CPA attacks.

7.12. Indistinguishability under Chosen Ciphertext Attack (IND-CCA)

By giving the attacker access to a decryption oracle, the IND-CCA security game expands upon IND-CPA.

- The challenger provides p_k to A and generates $(p_k,s_k) \leftarrow \text{KeyGen}(1^\lambda)$.
- Both encryption and decryption oracles can be queried via A.

- Two messages m_0, m_1 are selected by A. After sampling a random bit $b \in \{0,1\}$, the challenger returns

$c^* \leftarrow Enc_{pk}(m_b)$

Under the condition that A cannot use $c*$ to query the decryption oracle.

- After using the oracles for a while, A generates an estimate b′.

In this game, the opponent's advantage is:

$$Adv_{\pi}^{ind-cca}(A) = \left| P_r[b' = b] - \frac{1}{2} \right| \quad (32)$$

If all probabilistic polynomial-time adversaries A are secure, then an encryption system Π is IND-CCA secure

$$Adv_{\pi}^{ind-cca}(A) \leq \text{negl}(\lambda) \quad (33)$$

Our experiment results the success rate of the attacker was 50.01%, which is almost equal to 50% chance guessing. This confirms the robustness of the suggested encryption method against adaptive adversaries with decryption gateway access by showing that it retains in distinguishability even under chosen ciphertext attacks.

7.13. Birthday attack

Using the suggested technique, 100,000,000 keystream blocks were generated and analysed; no collisions were found. The cipher's ability to withstand collisions under the testing conditions is highly supported by this result. The fact that there were no collisions among 10 million outputs in the birthday paradox, where the likelihood of a collision rises with sample size, emphasizes the high uniqueness and resilience of NOVEL keystream generation and supports its applicability for cryptographic applications.

8. Conclusion

The proposed high-speed cryptographic technique CBTKSG Stream Cipher algorithm shows notable increases in encryption performance over current algorithms like chacha20 and AES. Which makes it an excellent choice for resource-constrained applications such as the Internet of Things with good throughput. Large-scale data demonstrated significantly faster encryption speeds compared to current standard ciphers. Performance evaluation validated the algorithm's high effectiveness. Additionally, the statistical test suite NIST SP 800-22 was employed to evaluate the generated keystream's security. The findings showed strong p-values for all randomness tests, demonstrating high statistical randomness and unpredictability. The suggested stream cipher, which is based on the CBTKSG Stream Cipher algorithm, was also examined for resistance to a number of well-known cryptanalytic assaults, such as related-key attacks, IND-CPA, IND-CCA, differential cryptanalysis and linear cryptanalysis. The cipher's resilience and appropriateness for secure communication systems and lightweight encryption applications are confirmed by the results, which demonstrate its strong resistance to all assessed attack vectors. These results validate the feasibility of the suggested algorithm for real-time Internet of Things settings where encryption that is safe, quick, and resource-efficient is essential. Future work involves optimizing fundamental arithmetic operations to further improve cryptographic performance

References

[1] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *Proc. 49th Hawaii Int. Conf. Syst. Sci. (HICSS)*, 2016, pp. 3928–3937.

[2] F. Muhammad, C. Ahendyarti, and Masjudin, "Chacha stream cipher implementation for network security in wireless sensor network," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 673, no. 1, p. 012064, 2019.

[3] E. Lara, L. Aguilar, J. A. García, and M. A. Sanchez, "A lightweight cipher based on salsa20 for resource-constrained iot devices," *Sensors*, vol. 18, no. 10, p. 3326, 2018.

[4] O. Z. Akif, S. Ali, R. S. Ali, and A. K. Farhan, "A new pseudorandom bits generator based on a 2d-chaotic system and diffusion property," *Bull. Electr. Eng. Inform.*, vol. 10, no. 3, pp. 1580–1588, 2021.

[5] D. Khwailleh and F. Al-Balas, "A dynamic data encryption method based on addressing the data importance on the internet of things," *Int. J. Electr. Comput. Eng.*, vol. 12, no. 2, pp. 2139–2146, 2022.

[6] S. O. Sharif and S. Mansoor, "Performance analysis of stream and block cipher algorithms," in *Proc. 3rd Int. Conf. Adv. Comput. Theory Eng. (ICACTE)*, vol. 1, 2010, pp. 1–522.

[7] E. Almaraz Luengo, "A brief and understandable guide to pseudo-random number generators and specific models for security," *Statist. Surveys*, vol. 16, pp. 137–181, 2022.

[8] J. Zheng and H. Hu, "A highly secure stream cipher based on analog-digital hybrid chaotic system," *Inf. Sci.*, vol. 587, pp. 226–246, 2022.

[9] T.-T. Lee and S.-T. Wu, "A lightweight keystream generator based on expanded chaos with a counter for secure iot," *Electronics*, vol. 13, no. 24, p. 5019, 2024.

[10] X. Wang, L. Zhang, and Y. Liu, "A Novel Lightweight Image Encryption Algorithm Based on Chaotic Maps," *Entropy*, vol. 23, no. 5, pp. 1-15, 2021.

[11] U. Zia, M. McCartney, B. Scotney, *et al.*, "A novel pseudo-random number generator for IoT based on a coupled map lattice system using the generalized symmetric map," *SN Appl. Sci.*, vol. 4, p. 48, 2022.

[12] Z. Liu, Y. Wang, J. Liu, J. Feng, and L. Y. Zhang, "Characteristics of 3d coupled map lattice and its application in pseudo-random number generator," *Nonlinear Dyn.*, vol. 112, no. 23, pp. 21509–21531, 2024.

[13] Y.-j. Sun, H. Zhang, X.-y. Wang, X.-q. Wang, and P.-f. Yan, "2d non-adjacent coupled map lattice with q and its applications in image encryption," *Appl. Math. Comput.*, vol. 373, p. 125039, 2020.

[14] K. Suwais and S. Almanasra, "Strike: Stream cipher based on stochastic lightning strike behaviour," *Appl. Sci.*, vol. 13, no. 8, p. 4669, 2023.

[15] S.-T. Wu, "Hybrid chaotic keystream generator based on dawson's summation generator," *Sens. Mater.*, vol. 35, 2023.

[16] Abdelli, W. Youssef, F. Kharroubi, L. Khriji, and M. Machhout, "A novel enhanced chaos based present lightweight cipher scheme," *Phys. Scr.*, vol. 99, no. 1, p. 016004, 2024.

[17] Y. Guang, L. Yu, W. Dong, Y. Wang, J. Zeng, J. Zhao, and Q. Ding, "Chaos-based lightweight cryptographic algorithm design and fpga implementation," *Entropy*, vol. 24, no. 11, p. 1610, 2022.

[18] T. R. Campbell, "Daence: Salsa20 and ChaCha in deterministic authenticated encryption with no noncense," *Cryptol. ePrint Arch.*, 2020.

[19] Salkanovic, S. Ljubic, L. Stankovic, and J. Lerga, "Analysis of cryptography algorithms implemented in android mobile application," *Inf. Technol. Control*, vol. 50, no. 4, pp. 786–807, 2021.

[20] E. J. Madarro-Capó, *et al.*, "New Weak Keys with Parity Patterns in the RC4 Stream Cipher," *Cryptography*, vol. 8, no. 4, p. 54, 2024.

[21] M. Alabdulrazzaq and M. Alenezi, "Performance Evaluation of Cryptographic Algorithms for IoT Applications," *J. Inf. Secur. Appl.*, vol. 58, pp. 1-11, 2022.

[22] P. Gehlot, S. Biradar, and B. Singh, "Implementation of modified twofish algorithm using 128 and 192-bit keys on vhdl," *Int. J. Comput. Appl.*, vol. 70, no. 13, 2013.

[23] M. Babitha and K. R. Babu, "Secure cloud storage using aes encryption," in *Proc. Int. Conf. Autom. Control Dyn. Optim. Techn. (ICACDOT)*, 2016, pp. 859–864.

[24] J.-P. Aumasson, *Serious Cryptography: A Practical Introduction to Modern Encryption*. No Starch Press, Inc, 2024.

[25] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2: Lightweight authenticated encryption and hashing," *J. Cryptol.*, vol. 34, no. 3, p. 33, 2021.

[26] X. Huang, L. Liu, X. Li, M. Yu, and Z. Wu, "A new pseudorandom bit generator based on mixing three-dimensional chen chaotic system with a chaotic tactics," *Complexity*, vol. 2019, p. 6567198, 2019.

[27] M. A. S. AL-khatib, A. H. Lone, and M. Uddin, "Comparative analysis of sla-lfsr with traditional pseudo random number generators," *Int. J. Comput. Intell. Res.*, vol. 13, no. 6, pp. 1461–1470, 2017.

[28] Y. Wang, Z. Liu, L. Y. Zhang, F. Pareschi, G. Setti, and G. Chen, "From chaos to pseudorandomness: a case study on the 2-d coupled map lattice," *IEEE Trans. Cybern.*, vol. 53, no. 2, pp. 1324–1334, 2023.