applied sciences

MDPI

Article

Strike: Stream Cipher Based on Stochastic Lightning Strike Behaviour

Khaled Suwais * and Sally Almanasra

Faculty of Computer Studies, Arab Open University, Riyadh 11681, Saudi Arabia
* Correspondence: khaled.suwais@arabou.edu.sa

Abstract: There is an increasing need for secure and fast encryption algorithms to support applications and communication protocols, and business models. In this paper, we present an alternative stream cipher (Strike) inspired by the stochastic behaviour of lightning strike phenomena. The novelty and originality of Strike stem from the utilisation of lightning strike behaviour as a source for generating random keystreams for encryption and decryption. Strike consists of three main functions: a function for setting up the security attributes, a function for generating lightning strikes and converting them to a keystream, and a function for plaintext encryption. The proposed stream cipher was tested against several cryptanalysis and statistical attacks in addition to other performance tests. The results show that Strike achieves high throughput on both high- and low-speed devices. Additionally, security analysis shows that our cipher is resistant to cryptanalysis and statistical attacks.

Keywords: stream cipher; lightning strike; encryption; pseudorandom generator; data security

Citation: Suwais, K.; Almanasra, S. Strike: Stream Cipher Based on Stochastic Lightning Strike Behaviour. *Appl. Sci.* **2023**, *13*, 4669. https://doi.org/10.3390/app13084669

Academic Editors: Luis Javier García Villalba and Ki-Hyun Jung

Received: 2 March 2023
Revised: 5 April 2023
Accepted: 5 April 2023
Published: 7 April 2023

1. Introduction

New technologies, particularly Internet of Things (IoT) networks, enable the connection of numerous devices through the Internet. These large-scale networks utilise vast amounts of obtained data to offer consumers a variety of applications. Nonetheless, this creates substantial security risks related to the exchanged data. Indeed, these systems are vulnerable to both traditional network attacks and novel threats that could compromise their availability, security and privacy. Security services like data confidentiality and privacy are crucial, and the challenge is exacerbated by the enormous amount of data and limited resources of some IoT devices. In this context, asymmetric encryption techniques are commonly used to protect data privacy and confidentiality [1,2].

The European Network of Excellence for Cryptology (ECRYPT), which coordinated the development of stream cipher algorithms in hardware and software, launched the research project eSTREAM in 2004 [3]. The project approved several stream ciphers, including Sprout [4], Fruit [5], LIZARD [6], Plantlet [7], Trivium [8], Mickey [9] and Grain series ciphers [10]. Some lightweight stream ciphers are currently deemed dangerous [11,12] despite significant progress in decoding techniques.

Numerous applications and systems process and communicate sensitive data. Such data must be safeguarded by adhering to the strictest guidelines and best practices for data confidentiality and privacy. Most symmetric encryption algorithms today are computationally costly, mainly because they require several complicated operations that are repeated numerous times [13,14]. They are thus unsuited for real-world applications or devices with limited resources. Accordingly, lightweight cryptographic approaches have been developed to lower the computing complexity of encryption.

Conventional encryption algorithms iterate a round function that includes complex operations using static (fixed) cryptographic primitives. Such algorithms suffer from high computing costs and are vulnerable to cryptanalysis attacks. According to NIST [15], lightweight ciphers should be designed to accommodate devices with constrained resources.

This is possible by designing a simplified round function or reducing the function's number of iterations. On the other hand, keystream generation should follow a dynamic approach in which the generator relies on regularly updating the set of security parameters to provide random properties for higher immunity against cryptanalysis and statistical attacks.

Even the most modern static-based lightweight ciphers require at least 6–18 cycles, requiring a substantial number of rounds [16,17]. In this study, we address the problems of operation complexity, encryption time and resource utilisation of existing stream ciphers. Stream ciphers that are designed to support low-resource computation devices usually either suffer from performance issues or are subjected to different cryptanalysis attacks. Hence, we present an alternative dynamic stream cipher to minimise complex operations, thus reducing the encryption time while preserving the requisite security level. In our method, each plaintext message is encrypted using a distinct set of cryptographic attributes. The keystream is formed by performing a set of mathematical operations in a single iteration. Upon generating one keystream block, all attributes are updated to generate a new keystream for a new encryption process.

Our proposed stream cipher (Strike) is novel, as it is the first stream cipher inspired by natural stochastic lightning strike phenomena. The novelty lies in the ability of designing fast encryption algorithm that is secure and less complex to work on computing devices with limited resources. The keys and parameters used in the model are extracted and calculated using mathematical models that express the behaviour of lightning strikes at different points in the given space. The strike starts randomly from a point x in a given space and hits another random point y on the ground. The behaviour of the strike is considered random and unpredictable, as a given strike is affected by several factors in addition to its start and end points in a large space; these factors include voltage, number of jumps, the distance of jumps and direction. In addition, Strike is considered feasible as it includes straightforward update procedures for the cryptographic attributes, reducing the computational complexity. As for error tolerance, Strike is resistant to channel failure since a bit error in one key affects only the corresponding byte of the encrypted message.

Our main contributions lie in presenting a high-performance, secure stream cipher. As for security, our design depends on the novel idea of utilising the stochastic behaviour of lightning strikes, which is proven mathematically [18]. On the one hand, the strike parameters are used as secure cryptographic primitives that are selected and updated randomly. The number of iterations is minimised to speed up keystream generation. Given that our cipher generates 16 bits of keystream in each round, it is usable in many resource-constrained devices and real-world applications.

The remaining sections of the paper are organised as follows: in Section 2, a background on stream ciphers and lightning strike phenomena is presented. Section 3 discusses the main issues related to existing stream ciphers. The main components of the Strike stream cipher are presented in Section 4. In Section 5, we conduct a security analysis by comparing the proposed cipher to the required cryptographic features. The efficiency of Strike is demonstrated in Section 6. Finally, Section 7 concludes the paper. However, the Abbreviations presents a list of abbreviations that appear in this document.

2. Preliminaries

2.1. Stream Ciphers

Stream ciphers are symmetric ciphers that generate pseudorandom sequences of bits from a given secret key. These pseudorandom bits are later used for encrypting plaintext using the XOR operation. Stream ciphers are widely and effectively used in securing TLS, Bluetooth and 4G connections [19].

Stream ciphers take two input values: a secret key (K) and an initial vector (IV). The length of the secret key is usually between 128 and 256 bits. The IV can be either a secret or known value, but keeping it secret adds an additional level of security. The general structure of stream ciphers is illustrated in Figure 1, where KS represents a keystream, and PT and CT represent the plaintext and ciphertext, respectively.

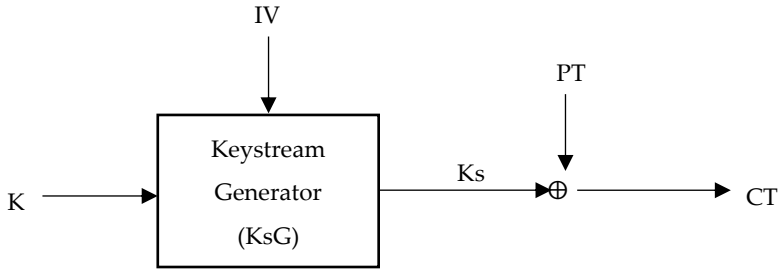

Figure 1. Stream cipher encryption operation.

The keystream is computed as a function KsG such that Ks = KsG (K, IV). The plaintext is encrypted as CT = PT ⊕ Ks, and the ciphertext is decrypted as PT = CT ⊕ Ks. Note that the encryption and decryption operations are the same, as they perform the same XOR operation. This justifies the absence of decrypt() functions in certain cryptographic libraries; the encrypt() function can be used for both encryption and decryption [19].

2.2. Lightning Strike Phenomena

Lightning is a natural phenomenon that originates randomly from thunderstorms and clouds [18,20,21]. In this section, we describe the mathematical model behind lightning strike development. According to [18], a lightning strike develops through several distinct phases. The first phase is called a *stepped leader* phase, where the head of the stroke is initiated from its origin (cloud base). This head makes a series of random jumps downward from its initialisation point toward the destination point (*priori*) on the earth's surface. The behaviour of the stepped leader creates a path between the cloud base and the ground. The average speed of this phase is approximately 0.001×300 m/µs, including pauses that the stepped leader makes between stochastic jumps.

As the stepped leader touches the surface, the *streamer phase* is initiated. This phase helps transfer charge between the earth and the cloud using the same path created by the stepped leader. The average speed of the streamer is higher than that of the stepped leader (0.01×300 m/µs), as it uses a previously created path with large currents with magnitudes exceeding hundreds of kilo-amperes (kA). A pause of a few milliseconds occurs upon completing the streamer phase. The end of this phase initiates the *dart leader* phase.

The dart leader starts from the cloud base and moves down to the ground. The dart leader is faster than the stepped leader since it uses a channel that is previously ionised. As soon as the dart leader touches priori on earth, a return stroke is initiated but with a much lower current. This procedure of dart leaders and return strokes continues for several iterations, resulting in a lightning flash composed of consecutive strokes [20,22].

As depicted in Figure 2 [18], the stepped leader head is initiated at height h_0 and randomly makes j jumps towards the earth's surface. The streamer is immediately initiated back to the cloud. As time t passes, several dart leaders occur with less surge current i. Once the current level reaches the minimum, the stroke stops.

The probability density function that represents the lightning strike is computed by Equation (1):

$$P(I) = 0.5 \cdot erfc(u_0) \tag{1}$$

where $erfc(u_0)$ is calculated in Equations (2) and (3) as follows:

$$erfc(u_0) = 1 - \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n \cdot u_0^{2n+1}}{n!(2n+1)} \tag{2}$$

$$u_0 = \frac{log(I) - \log(I_\mu)}{\sqrt{2} \cdot I_\sigma} \tag{3}$$

According to [23], the mean value (I_μ) and standard deviation (I_σ) of the current peak of the lightning can have the specific values shown in Table 1. The values are defined under the assumption that strokes can have a positive or negative polarity.

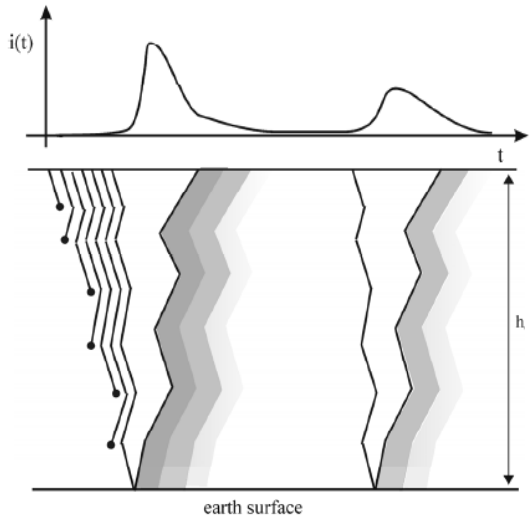

Figure 2. Lightning strike development through time t.

Table 1. Logarithmic normal distribution of lightning current parameters [23].

Mean (I_μ)	Stand. Dev. (I_σ)	Stoke Type
61.1	0.576	−stroke (less than 20 kA)
33.3	0.263	−stroke (greater than 20 kA)
33.9	0.527	+stroke

3. Related Works

Lizard is a lightweight stream cipher introduced by [6]. Lizard is designed for use with power-constrained devices such as passive RFID tags. The hardware efficiency is the consequence of merging a Grain-like design with the FP (1)-mode, a newly proposed building concept for the state initialisation of stream ciphers. Lizard employs 120-bit keys, 64-bit IVs, and a 121-bit inner state length as part of its core processes. As a result, Lizard can generate up to 218 keystream bits per key/IV pair, which makes it suitable with currently used communication protocols, such as Bluetooth, WLAN and HTTPS. The results show that Lizard's hardware design is more efficient than other hardware-based ciphers. In addition, Lizard is suitable for applications that do not require high-speed encryption rates.

In [24], the researchers presented a new lightweight RFID-based authentication solution that employs a stream cipher to ensure privacy between valid components. The suggested method eliminates the inherent linearity of LFSRs by using a variety of keystream generators such as And2LFSR, Geffe and majority-based. This allows for the construction of a nonlinear pseudorandom number generator (PRNG). For statistical analysis, the authors employed a multiple polynomial quadratic sieve. The simulation results show that the proposed authentication technique's performance is affected by the kind of error control code, nonlinear combinational functions and the degree of the generator polynomial.

The researchers in [25] developed a new deep learning-based key generation network called DeepKeyGen that acts as a stream cipher generator to produce the secret key for medical image encryption. DeepKeyGen's main goal is to learn the mapping relationship that describes how to convert a given image into a secret key. Security analysis demonstrates that the keystream generated by DeepKeyGen is highly sensitive to change, possesses a large key space, is pseudorandom, and is resistant to a variety of attacks. Statistically, only entropy tests are applied. Further statistical and performance analysis is needed to ensure the algorithm's efficiency.

Another stream cipher for image encryption is presented in [26]. The stream cipher is designed to ensure the safe transmission of images between different parties. As part of the pixel manipulation process, a combination of two-dimensional discrete wavelet transforms and the Arnold mapping algorithm are used. The experimental results demonstrate that the proposed stream cipher is secure in terms of randomness and statistics. However, the work does not provide sufficient detail about the performance of the cipher in terms of speed and throughput.

In [27], an image encryption cipher based on fixed-point chaotic maps is proposed. The correlation coefficient, differential attack, histogram and information entropy are some of the security analyses that are used in the process of evaluating the algorithm. Statistically, correlation coefficient analysis is performed, and the tests demonstrate resistance to entropy attacks. However, the algorithm's throughput is low (186.84 Mb/s) compared to other existing secure stream ciphers. In addition, other statistical tests are needed (e.g., NIST tests) to ensure its resistance to statistical attacks.

A stream cipher based on adding offsets to the two-dimensional coupled map lattice (2D-CML) is developed in [28]. The technique proposes adding distinct offsets to each lattice. The offset 2D CML model outperforms the original 2D-CML model in terms of its chaotic qualities; specifically, it has larger Lyapunov exponents (LE). It also produces more uniform chaotic sequences than the original model. Statistically, the Chi-square test, correlation coefficient and NIST SP800-22 suite analysis are performed. The research findings demonstrate that the algorithm is secure against statistical and cryptanalysis attacks. However, the throughput of the algorithm is low (around 22 Mbit/s), and this performance limitation would be challenging in many applications.

As part of the recent progress in IoT security, an alternative lightweight cipher scheme (LoRCA) is proposed in [29]. This cipher is based on a dynamic, key-dependent structure that offers data secrecy using minimal resources. Statistically, the cipher successfully passed several tests, which suggests that it is resistant to statistical attacks. However, the cipher must be verified in terms of performance, as no performance results on the throughput of the algorithm are indicated.

In [30], a unique resilient symmetric image encryption structure (IES) is introduced. IES generates a one-time session key by combining the image with a shared key. Statistically, IES is tested against entropy, PSNR and the correlation test. The results show that IES is secure against statistical attacks. However, its performance in terms of encryption throughput seems to be affected by the costly computations of chaos-based cryptosystems. Such performance makes IES suitable for highly secure applications that do not require high throughput rates.

A stream cipher based on Linear Feedback Shift Register (LFSR) is proposed in [31]. The cipher aims to enhance the length of the sequence by integrating LFSR and a genetic algorithm (GA). The authors claim that the length of the generated sequence is more than the maximum length of LFSR. As for statistical analysis, key sequence, uniformity and autocorrelation analysis are performed. The results show that the proposed stream cipher is resistant to statistical and cryptanalysis attacks. Nevertheless, achieving a high encryption rate was not this work's main goal. Hence, it does not provide sufficient data on the performance of the stream cipher.

An alternative stream cipher system based on an analogue–digital hybrid chaotic system is proposed in [32]. The hybrid model can generate digital chaotic maps without degeneration. It also ensures the synchronisation of analogue chaotic systems, which is necessary for effective decryption. Moreover, the stream cipher offers the advantages of a massive key space, near-indefinite cycle duration, and strong security. Statistically, NIST and TestU01 analyses are performed. The results show that the cipher is resistant to statistical attacks. However, no proofs are provided concerning the efficiency of the cipher in terms of encryption rate and throughput.

In [15], a stream cipher based on a chaotic system in conjunction with two nonlinear feedback shift registers (NFSRs) is proposed. The cipher is presented as a lightweight

encryption algorithm for devices with limited computational resources. Several statistical analyses are discussed, including NIST and entropy tests. The results indicated that the cipher is secure against statistical attacks. The evaluation of hardware resources and throughput demonstrates the model's capability to operate on resource-limited devices. However, the throughput of the cipher is less than 100 Kb/s, which makes it unsuitable for high-speed encryption.

LESCA (LightwEight Stream Cipher Algorithm) is a lightweight stream cipher proposed by [17]. The cipher comprises two major functions: a round function based on cryptographic primitives and a function for updating those primitives. The round function is responsible for converting each plaintext/output input sequence to word precision, while the updating function updates all primitives using permutation tables. The proposed cipher outperforms other lightweight stream ciphers. LESCA is also validated statistically; the statistical tests show that LESCA is secure against statistical attacks. However, LESCA's potential applications for secure high-speed communication are limited.

Finally, a speech encryption algorithm is presented in [33]. The proposed algorithm pre-processes the original speech by deleting the unvoiced bits of the signal in order to choose the necessary data for encryption. The algorithm uses PRNG based on two 256-bit shift registers. One of the registers is linear, while the other is nonlinear. This allows the algorithm to expand its key space, which makes it secure against brute-force attacks. For statistical analysis, correlation coefficient, SNR and PSNR analysis are performed. The experimental results show that the generated keystream has high sensitivity, reduced correlation and a uniform histogram. However, the research does not provide sufficient analysis of the performance of the proposed cipher. Table 2 presents a summary of the comparison of related works presented in this study.

Table 2. Comparison of related works.

Work	Method(s)	Limitation
[6]	Self-shrinking generator method	Efficient as hardware implementation, but it is not suitable for applications that require high-speed encryption rates
[15]	Chaotic system in conjunction with two NFSRs	Low encryption speed, which makes it unsuitable for high-speed encryption-based applications.
[17]	Round function and permutation table	Limited potential application for secure high-speed communication.
[24]	And2LFSR, Geffe and majority-based	Performance is affected by the kind of error control code, nonlinear combinational functions and the degree of the generator polynomial
[25]	Deep learning–based key generation network	Lack of performance analysis to examine the algorithm's efficiency
[26]	2D discrete wavelet transforms and the Arnold mapping algorithm	Lack of performance analysis to examine the algorithm's efficiency
[27]	Fixed-point chaotic maps	Low performance and lack of sufficient statistical testing.
[28]	Two-dimensional coupled map lattice	Low encryption speed, which makes it unsuitable for high-speed encryption-based applications.
[29]	Dynamic key-dependent structure	Lack of performance analysis to examine the algorithm's efficiency.
[30]	Chaos-based cryptosystem	Suitable for highly secure applications that do not require high throughput rates.
[31]	LFSR and a genetic algorithm	Lack of performance analysis to examine the algorithm's efficiency
[32]	Analogue–digital hybrid chaotic system	Lack of performance analysis to examine the algorithm's efficiency
[33]	PRNG based on two 256-bit shift registers	Lack of performance analysis to examine the algorithm's efficiency

4. The Proposed Stream Cipher (Strike)

4.1. Mathematical Model of Strike Cipher

In this section, we introduce the detailed structure of our proposed stream cipher (Strike). The mathematical model of Strike is inspired by the lightning strike development model mathematically described in [18]. The model assumes that every single point can be presented in a Cartesian coordinate system (x, y, z). If a plane with $z = 0$ is chosen, the point will directly touch the surface of the ground. The stepped leader head starts at a given point at height h_0 and moves downward to the ground, as depicted in Figure 3.

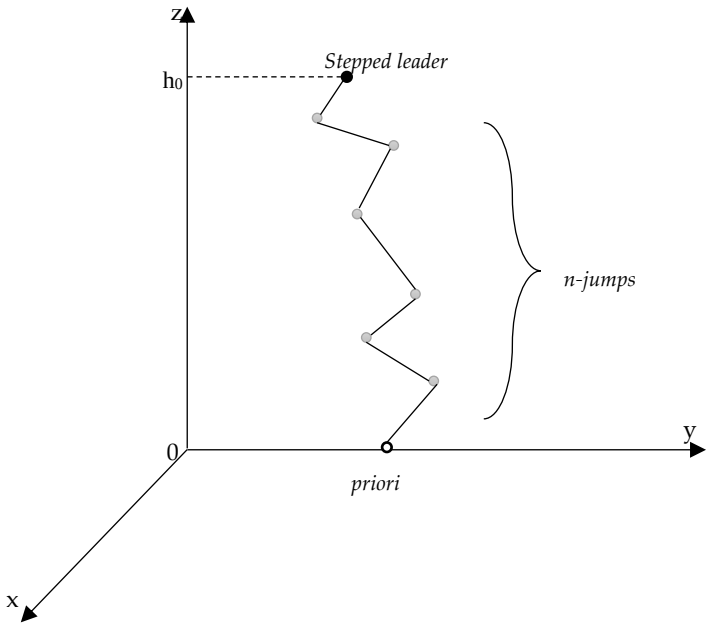

Figure 3. Simulating n jumps of lightning strike stepped leader head.

The starting point of the lightning strike is point T_0 of coordinate (x_0, y_0, h_0), where x_0, y_0 and h_0 are stochastically chosen according to the following equations:

$$x_0 = r \cdot a \tag{4}$$

$$y_0 = r \cdot b \tag{5}$$

$$h_0 = r \cdot c \tag{6}$$

where r is a pseudorandom value, $r \in [0, 1]$.

Choosing the peak value of the lightning current is also carried out stochastically. The lightning current peak values can be observed as an interval such that $I \in [I_{min}, I_{max}]$. This interval is divided into m classes. Accordingly, the lightning current's peak value is selected from a given class i as follows:

$$I = i_{I_{min}} + r \cdot \left(i_{I_{max}} - i_{I_{min}}\right) \tag{7}$$

where $i_{I_{min}}$ represents the minimal value and i_{max} represents the maximum peak value chosen from the i-th class. Note that both $i_{I_{min}}$ and i_{max} are calculated as follows:

$$i_{I_{min}} = I_{min} + (i - 1) \cdot \Delta I \tag{8}$$

$$i_{I_{max}} = i_{I_{min}} + \Delta I \tag{9}$$

$$\Delta I = \frac{I_{max} - I_{min}}{N_c} \tag{10}$$

As the stepped leader descends toward the surface, it makes several jumps of different distances in a stochastic path. The strike distance (jump) is calculated in metres as a function of the chosen lightning current peak as follows:

$$R = A \cdot \left(TP \cdot I^{\cos\left(I_{mean} + I_{std}\right)} \right) \tag{11}$$

With reference to Figure 3, the stepped leader head starts at a given point in space, then makes j jumps downward. The coordinates of the stepped leader head are calculated as follows:

$$x_j = x_{j-1} + R \cdot \sin\sigma_{j-1} \cdot \cos\varphi_{j-1} \tag{12}$$

$$y_j = y_{j-1} + R \cdot \sin\sigma_{j-1} \cdot \sin\varphi_{j-1} \tag{13}$$

$$h_j = h_{j-1} + R \cdot \cos\sigma_{j-1} \tag{14}$$

where the coordinates of the next position of the stepped leader head are determined stochastically as follows:

$$\varphi_{j-1} = r \cdot 2\pi \tag{15}$$

$$\sigma_{j-1} = (r+1) \cdot \frac{\pi}{2} \tag{16}$$

Equations (12)–(16) present continuous stochastic processes that continue until reaching the earth's surface. In addition, this process is repeated an arbitrary number of times, as new lightning strikes may start at different positions on any given cloud. This process summarises the randomness of the lightning strike phenomena through the undetermined random paths that each strike initiates.

4.2. Detailed Design of Strike Stream Cipher

Our Strike stream cipher is composed of three main phases. The first is responsible for setting up the security parameters. The input to this phase is the secret key (*SKey*) and an initial vector (*IVec*). Then, the second phase (keystream generation) is initiated. This phase generates lightning strikes which are later transferred to keystream sequences of keystream. The inputs to this phase are lightning strike-related attributes, including strike jumps distances, peak current intervals, and the stepped leader coordinates. Lastly, the generated keystream is passed to the encryption phase, which XOR the plaintext bits with the corresponding keystream. The general structure of the Strike cipher is depicted in Figure 4. The main security attributes used in our model are also listed and described in Table 3.

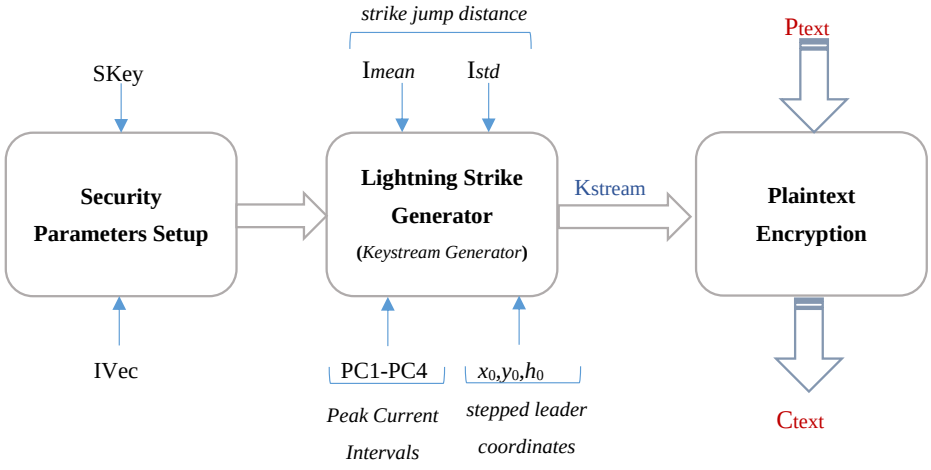

Figure 4. General structure of Strike stream cipher.

Table 3. Logarithmic normal distribution of lightning current parameters.

Parameter	Length	Description
SKey	128-bit	Secret key
IVec	32-bit	Initial vector
a	8-bit	*x*-coordinate on ground
b	8-bit	*y*-coordinate on ground
c	8-bit	*z*-coordinate in sky
r	8-bit	Random number
j	8-bit	Number of jumps composing the lightning strike
s	8-bit	The order number of selected class in PC*i*
*PC*1–*PC*4	16-bit/each	Peak current intervals
KSC	8-bit	*Skey* shifting controller
I_{mean}	8-bit	Mean value of current *I*
I_{std}	8-bit	Standard deviation of current *I*
IVSC	8-bit	*IVec* shifting controller
iclasses	8-bit	Total number of classes in each PC*i*
K_{stream}	16-bit	Keystream binary bits
C_{text}	16-bit	Ciphertext binary bits
P_{text}	16-bit	Plaintext binary bits
TP	4-bit	Total peak value

4.2.1. Security Parameters Setup

The setup process takes two input values: *SKey* of 128-bit length and the *IVec* of 32-bit length to initialise all other parameters used in our Strike cipher. The process follows the following steps:

- Extract bits from *SKey* and *IVec* to be assigned to all other parameters, such that *SKey* bits from 0–55 are assigned to the parameters *a*, *b*, *c*, *r*, *j*, and *s* sequentially.
- Initialise the peak current intervals (PC*i*) using the *SKey* bits from 56–119.
- Initialise the *KSC* parameter using the remaining *SKey* bits from 120–127, which will later help in shifting the *SKey*.
- Initialise the I_{mean} parameter using the *IVec* bits from 0–7.
- Extract the *IVec* bits from 8–15 and 16–23 to initialise I_{std} and *IVSC*, respectively.
- Extract the remaining *IVec* bits from 24–31 to initialise *iclasses* parameter.
- Initialise the coordinates of the lightning strike using Equations (4)–(6).

The span of each class in each PC*i* interval is computed to generate peak current classes for each PC*i*. Algorithm 1 describes the main processes carried out in the security parameter setup phase of the Strike cipher.

4.2.2. Keystream Generation

Our stream cipher's keystream generator acts as a lightning strike generator. The lightning strike process is composed of five stages: selecting the peak current, computing the probability density function, calculating the jump distances, locating the stepped leader head and generating the keystream. The structure of the lightning strike generator for the keystream generation process is illustrated in Figure 5.

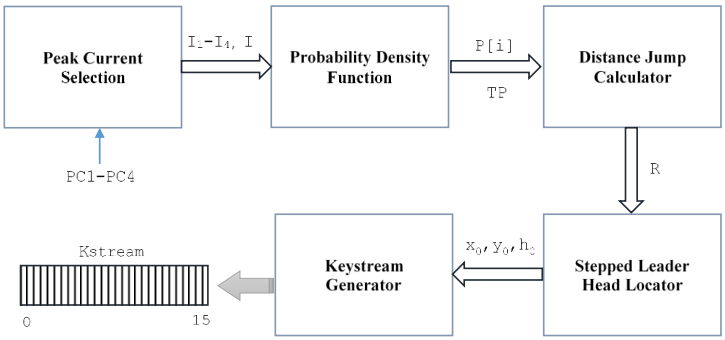

Figure 5. Lightning strike generator for keystream.

Algorithm 1 Skey/IVec Setup

Input: 128-bit value SKey
32-bit value IVec
Output: 8-bit values a, b, c, r, j, s
8-bit values KSC, I_{mean}, I_{std}, IVSC, iclasses
16-bit values PC1, PC2, PC3, PC4
16-bit values x_0, y_0, h_0

// extract Secrete Key (SKey) parameters
a = SKey[bit_0–bit_7]
b = SKey[bit_8–bit_15]
c = SKey[bit_16–bit_23]
r = (SKey[bit_24–bit_31] + SKey[bit_32–bit_39]) *mod* 255
j = SKey[bit_40–bit_47]
s = SKey[bit_48–bit_55]
// initialize the min and max peak currents of each PC interval
function Calculate_PCs (SKey)
    PC1 = [I_min = Min (SKey[bit_56–bit_63], SKey[bit_64–bit_71]),
        I_max = Max (SKey[bit_56–bit_63], SKey[bit_64–bit_71])]
    PC2 = [I_min = Min (SKey[bit_72–bit_79], SKey[bit_80–bit_87]),
        I_max = Max (SKey[bit_72–bit_79], SKey[bit_80–bit_87])]
    PC3 = [I_min = Min (SKey[bit_88–bit_95], SKey[bit_96–bit_103]),
        I_max = Max (SKey[bit_88–bit_95], SKey[bit_96–bit_103])]
    PC4 = [I_min = Min (SKey[bit_104–bit_111], SKey[bit_112–bit_119]),
        I_max = Max (SKey[bit_104–bit_111], SKey[bit_112–bit_119])]
end function
KSC = SKey[bit_120–bit_127]
// extract Initial Vector (IVec) parameters
I_{mean} = IVec[bit_0–bit_7]
I_{std} = IVec[bit_8–bit_15]
IVSC = IVec[bit_16–bit_23]
iclasses = IVec[bit_24–bit_31]
// calculate the initial lightning strike coordinates over 3D space
function Initialize_Coord (a, b, c, r)
    $x_0 = r \cdot a$
    $y_0 = r \cdot b$
    $h_0 = r \cdot c$
end function
//calculate span of each class in each PC
function Generate_Classes (PC1, PC2, PC3, PC4, iclasses)
    for PC = 1 **to** PC = 4 **do**
        PC_span_I[PC] = (I_max − I_min)/iclasses
    end for
    // generate peak current classes for each PC interval
    for PC = 1 **to** PC = 4 **do**
        for cls = 1 **to** cls = iclasses **do**
            iclass_I_min = I_min[PC] + (iclasses − 1) · PC_span_I[PC]
            iclass_I_max = iclass_I_min + PC_span_I[PC]
        end for
    end for
end function

The first step starts from the peak current selection by choosing four peaks (I_1–I_4) from their corresponding intervals (PC1–PC4), where [I_1, I_3] are chosen as the maximum peaks and [I_2, I_4] are chosen as the minimum peaks. Upon selecting the four peaks, the final general peak (I) is computed by XORing ($\oplus$) the four peaks as follows:

$$I = I_1 \oplus I_2 \oplus I_3 \oplus I_4 \tag{17}$$

Next, the probability density function is processed by applying Equations (1)–(3). Note that the series $\sum_{n=0}^{\infty} \frac{(-1)^n u_0^{2n+1}}{n!(2n+1)}$ presented in Equation (2) is found to be convergent. Hence, the infinite upper limit can be replaced by a fixed value based on the following ratio test:

$$\lim_{n\to\infty}\left|\frac{a_{n+1}}{a_n}\right| = \lim_{n\to\infty}\left|\frac{u_0^{2n+3}}{(n+1)!(2n+3)} \cdot \frac{n!(2n+1)}{u_0^{2n+1}}\right| \tag{18}$$

$$= \lim_{n\to\infty}\left|\frac{2u_0^2 n + u_0^2}{2n^2+5n+3}\right| = 0 < 1 \tag{19}$$

The probability density function will result in 4 bits (1 bit for each peak value). These bits are later concatenated to generate the total peak value (TP). As we calculated the TP, we proceeded to calculate the distance of each jump of the lightning strike using Equation (11). Consequently, the new coordinates of the stepped leader head are calculated using Equations (12)–(16). Lastly, the 16-bit length keystream is generated by XORing the x, y, and h coordinates using Equation (20):

$$K_{stream} = \left(x_j \oplus y_j \oplus h_j\right) << \text{TP} \tag{20}$$

Upon generating the keystream bits, a complete shuffling process is applied over all security parameters to ensure a sufficient level of randomness before generating new lightning strikes for new keystreams. Algorithm 2 presents the detailed processes of keystream generation inspired by the lightning strike phenomena.

4.2.3. Data Encryption

Plaintext encryption is a straightforward process that involves XORing the plaintext (P_{text}) bits with the keystream (K_{stream}) bits to generate the ciphertext (C_{text}). The length of each sequence is 16 bits. Once the keystream bits are completely used during encryption, the keystream generator is re-called to generate new keystream sequences. The plaintext encryption is illustrated in Figure 6, and the main processes are depicted in Algorithm 3.

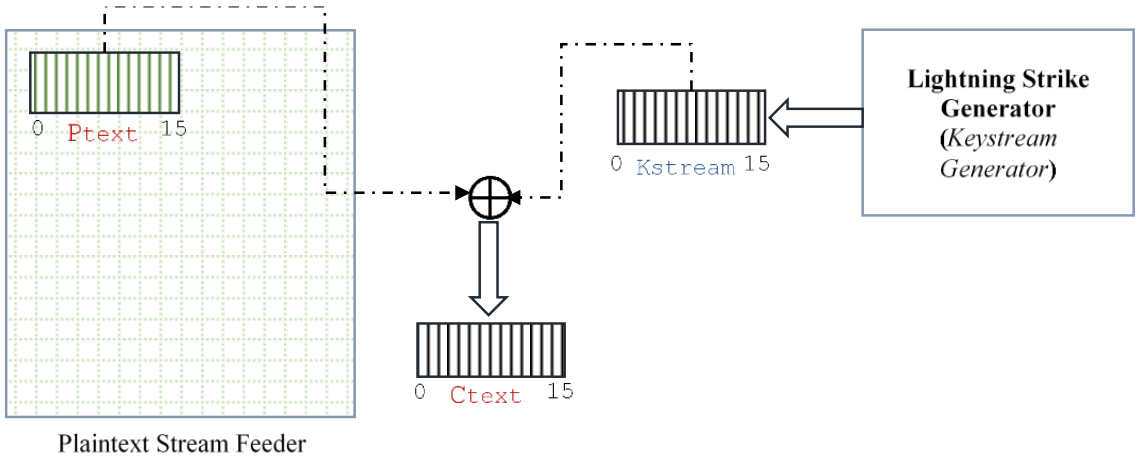

Figure 6. Plaintext Encryption.

Algorithm 2 Keystream Generation (Lightning Strike Generator)

Input: 8-bit values I_{mean}, I_{std}
16-bit value PC1, PC2, PC3, PC4, x_0, y_0, h_0
Output: 16-bit value K_{stream}

// select a peak current from each PC interval
I_1 = iclass_I_max(PC1, s mod iclasses)
I_2 = iclass_I_min(PC2, s mod iclasses)
I_3 = iclass_I_max(PC3, s mod iclasses)
I_4 = iclass_I_min(PC4, s mod iclasses)
// calculate the final peak current I
$I = I_1 \oplus I_1 \oplus I_1 \oplus I_1$
// calculate the probability density function of all selected currents
$u_1 = ((\log(I_1) - \log(I_{mean}))/(\mathrm{sqrt}(2) \cdot I_{std})$
$u_2 = ((\log(I_2) - \log(I_{mean}))/(\mathrm{sqrt}(2) \cdot I_{std})$
$u_3 = ((\log(I_3) - \log(I_{mean}))/(\mathrm{sqrt}(2) \cdot I_{std})$
$u_4 = ((\log(I_4) - \log(I_{mean}))/(\mathrm{sqrt}(2) \cdot I_{std})$
init = $1 - (2/\mathrm{sqrt}(Pi))$
sum = 0
for i = 1 **to** i = 4 **do**
    for n = 0 **to** n = 10 **do**
        sum = sum + $((-1)\hat{}n \cdot (u_i)\hat{}(2n + 1)/(n! \cdot (2n + 1))$
    end for
    erfc[i] = init · sum
end for
for i = 1 **to** n = 4 **do**
    P[i] = 1 · erfc[i]
end for
// concatenate P[i] to generate 4-bit TP value
TP = P [1] || P [2] || P [3] || P [4]
// calculate the distance of each jump
$R = 10 \cdot (TP \cdot I\hat{}(\cos(I_{mean} + I_{std}))$
// re-calculate the new coordinates of stepped leader head
for i = 1 **to** i = j **do**
    $phi_{j-1} = r \cdot (2 \cdot Pi)$
    $sigma_{j-1} = (r + 1) \cdot (Pi/2)$
    $x_j = x_{j-1} + R \cdot \sin(sigma_{j-1}) \cdot \cos(Phi_{j-1})$
    $y_j = y_{j-1} + R \cdot \sin(sigma_{j-1}) \cdot \sin(Phi_{j-1})$
    $h_j = h_{j-1} + R \cdot \cos(sigma_{j-1})$
end for
// calculate the keystream (K_{stream}) of 16-bit length
$K_{stream} = (x_j \oplus y_j \oplus h_j) << TP$
// shuffle security parameters
r << KSC
j >> KSC
s << KSC
Skey >> KSC
I_{mean} << IVSC
I_{std} >> IVSC
Iclasses << IVSC
IVSC >> KSC
KSC << IVSC
// re-calculate PC1-PC4
Calculate_PCs (SKey)
// re-initialize coordinates
a = SKey[bit_0–bit_7]
b = SKey[bit_8–bit_15]
c = SKey[bit_16–bit_23]
Initialize_Coord (a, b, c, r)
// re-generate peak current classes
Generate_Classes (PC1, PC2, PC3, PC4, iclasses)

Algorithm 3 Encryption

```
Input:  16-bit values P_Text, K_stream
Output: 16-bit values C_Text
//use K_stream to be exclusively or'ed with plaintext P_Text
while P_Text != null do
    for bit = 0 to bit = 15 do
        C_Text[bit] = P_Text[bit] ⊕ K_stream[bit]
    end for
//generate more SSK by the four threads
Keystream_Generation()
end while
```

5. Security Analysis of Strike

5.1. Statistical Tests and Balance

Testing the stream cipher with statistical tests is critical to ensure that the cipher is secure against statistical attacks [34,35]. Our Strike stream cipher is tested against the standard statistical test suite NIST [36]. The NIST statistical test is composed of 15 tests that calculate 188 results values [35]. The test is carried out on 1500 sequences of keystream (1.5 GB of data), which represent the sample size used in this test. The size of each sequence is 1-MB resulting from 64 keystreams. The results presented in Table 4 show that the Strike cipher successfully passed all the NIST tests.

Table 4. NIST statistical tests results.

Test	*p*-Value	Passing Rate	Decision
Frequency	0.098956	0.994	passed
Serial	0.589774	0.991	passed
Block frequency	0.333145	0.985	passed
Random excursion	0.228764	0.991	passed
Approximate entropy	0.839894	0.982	passed
Universal	0.598752	0.991	passed
FFT	0.021888	0.981	passed
Longest run	0.129204	0.982	passed
Linear complexity	0.456098	0.988	passed
Rank	0.098780	0.986	passed
Random excursion variant	0.333248	0.992	passed
Overlapping templates	0.506503	0.988	passed
Non-overlapping templates	0.229941	0.981	passed
Cumulative sums	0.377852	0.983	passed
Runs	0.187012	0.991	passed

In addition to NIST tests, we carried out a balance test to examine whether the generated keystream has an equal number of zeros and ones. The test examined 300 different keystreams. The number of zeros and ones is counted in each keystream. The test shows that no biased is detected in the generated keystream. The number of zeros and ones was almost equal in all keystreams, with a minor difference that does not exceed 2 bits. Figure 7 visualises the balance test. These results indicate that Strike is able to generate balanced keystreams that are resistant to several statistical attacks.

Figure 7. Balance analysis.

5.2. Avalanche Effect

Our Strike cipher is inspired by the stochastic behaviour of lightning strike phenomena. This behaviour is completely random and unpredictable, as each strike starts at a random point and moves in different directions and distances (jumps). In addition, every single keystream is generated using a completely new set of scrambled parameters, which produces a new sequence of bits that differ from previously generated sequences.

In this test, we consider the cryptographic property known as the avalanche effect. It mainly measures the impact of altering (flipping) one bit of the secret key (*SKey*) on the generated keystream (K_{stream}). The test was applied to a data set of 500 keystreams. The results are presented in Table 5. The dots visualised in Figure 8 represent the percentage of changes resulting in each keystream after flipping one bit in the secret key (*SKey*). The testing results show that Strike achieved an average avalanche percentage of about 74%.

Table 5. Avalanche effect on K_{stream}.

# of K_{stream}	500
Minimum % of changes on K_{stream}	70%
Maximum % of changes on K_{stream}	77%
Avg. % of changes on K_{stream}	73.56%

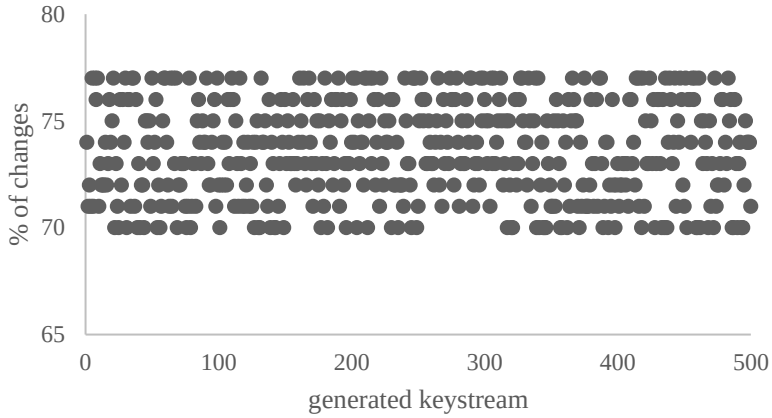

Figure 8. Avalanche effect on K_{stream}.

5.3. Cryptanalysis Attacks

5.3.1. Brute-Force Attacks

Brute-force techniques aim to guess all possible choices in a given search space. The search space of our Strike cipher is enormous. Strike cipher uses a 128-bit *SKey* and 32-bit *IVec* to generate one K_{stream} in each iteration. This requires the attacker to make $2^{128} + 2^{32}$ guesses to reveal both the *SKey* and *IVec*, which is practically impossible with currently available resources. Hence, the Strike cipher is found to be secure against brute-force attacks.

5.3.2. Known-Plaintext Attack

In this kind of attack, the attacker attempts to gain access to the complete list of ciphertexts and their mapped plaintext to retrieve *SKey*. In the Strike cipher, each sequence of C_{text} is obtained from an XOR operation between P_{text} and K_{stream}. A complete scrambling process is applied to every single parameter used in the keystream generator. Hence, the randomness of the behaviour of the generated strike will generate a completely different ciphertext, even for two identical plaintexts. Recall that each lightning strike is associated with different peak current intervals, stepped leader coordinates and jumps, all of which are extracted stochastically from *SKey*. Accordingly, the Strike cipher is found to be resistant to known-plaintext attacks.

5.3.3. Ciphertext-Only Attack

In this kind of attack, an attacker attempts to gain access to the ciphertext in order to retrieve the plaintext after guessing the secret key. Once the attacker succeeds, all the ciphertext can be decrypted using the key. However, our stream cipher generates a new lightning strike in each iteration. Hence, two identical ciphertext segments are mapped to two completely different lightning strikes. As a result, our Strike cipher is found to be secure against this kind of attack.

5.3.4. Differential Attack

Differential attacks try to reveal or reduce the time needed to reveal the secret key by performing intensive and comprehensive analysis over pairs of plaintext and ciphertext. In our Strike cipher, the keystream generator produces a completely new *SKey* at each iteration, where each lightning strike has its own unique set of attributes. Thus, attributes are scrambled and randomised at the end of each iteration, ensuring the uniqueness of each generated *SKey*. Accordingly, two identical plaintexts will be converted into two different ciphertexts. Hence, our Strike cipher is resistant to differential attacks.

6. Complexity and Performance Analysis

In this section, we analyse the complexity of the Strike cipher. The cipher is composed of three main stages: security parameters setup, lightning strike generation and data encryption. The complexity analysis revealed efficient results, as tabulated in Table 6.

Table 6. Complexity Analysis.

	Security Parameter Setup	**Lightning Strike Generator**	**Data Encryption**
Complexity	O(log$n$)	O(n)	O($n$log$n$)

The performance of the Strike cipher is also examined and analysed. The cipher is implemented in a Python environment installed on four computing devices with different computing capabilities to measure the algorithm's efficiency. The specifications of these machines are listed in Table 7.

Table 7. Specification of testing environments.

	Processor	**RAM Memory**	**Storage Capacity**	**Operating System**
Machine 1 (M1)	Intel Core i7®	6 GB	500 GB HDD, 128 GB SSD	Windows 10®
Machine 2 (M2)	Intel Core i5®	4 GB	500 GB HDD, 128 GB SSD	Windows 10®
Machine 3 (M3)	Intel Core i3®	2 GB	500 GB HDD	Windows 10®
Machine 4 (M4)	Intel Celeron®	2 GB	500 GB HDD	Windows 10®

The performance of the Strike stream cipher is tested and compared against well-known stream ciphers, including AES-128, Snow 2.0–128, Salsa-20, HC-128 [34,37], Trivium [8], Mickey-128 [9], Grain [10] and the cipher proposed by [28]. The testing results show that the Strike cipher achieved a throughput of 22,796.4 Mbit/s on the M1 machine. The results are tabulated in Table 8, which shows that our stream cipher outperforms other ciphers. The last column of Table 8 shows the ratio of efficiency of our cipher compared to others. The efficiency ranges from 2.5% (against [28]) to 99.9% (against Grain). Figure 9 visualises the throughputs of all tested ciphers.

Table 8. Stream ciphers performance results on M1.

Cipher	Throughput (Mbit/s)	Ratio of Efficiency
Strike	22,796.4	-
(Liu et al., 2020) [28]	22,222.2	2.5%
Snow 2.0–128	22,026.4	3.4%
Trivium	10,026.7	56.0%
Salsa-20	7987.2	65.0%
AES-128	6581.8	71.1%
HC-128	5980	73.8%
Mickey-128	192.4	99.2%
Grain	13.8	99.9%

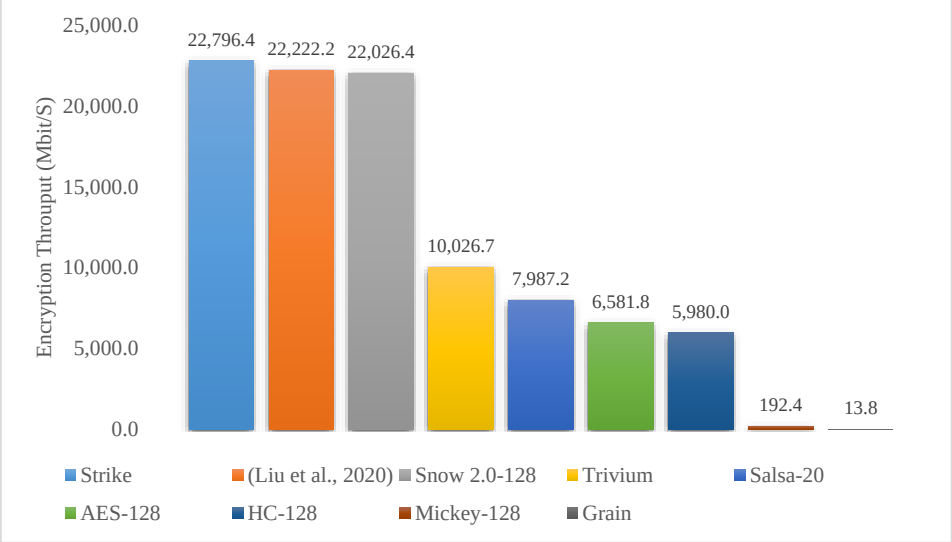

Figure 9. Stream ciphers throughput analysis on M1.

In addition, the Strike cipher is tested on different machines (M1–M4). The results presented in Table 9 show that the algorithm's structure was not affected by the computing power capabilities available on different machines. For instance, the major difference in performance is found between M1 and M4, where the performance is degraded by 7.25%; this is mainly due to the slower CPU processing speed and lower RAM capacity.

Table 9. Encryption throughout (Mbit/s) of Strike over different machines.

	M1	M2	M3	M4
Throughput (Mbit/s)	22,796.4	22,287.1	21,998.9	21,143.5
Efficiency Ratio	-	−2.23%	−3.5%	−7.25%

7. Conclusions and Future Works

In this research, we presented the Strike stream cipher. Strike is inspired by the stochastic behaviour of lightning strike phenomena. Each lightning strike has its own unique attributes, including the source and destination points, jump distances, peak intervals, power and pathways. These attributes represent the core security parameters of our lightning strike generator, which later generates keystreams for data encryption and decryption. In each iteration of keystream generation, the strike's attributes are updated and scrambled to ensure that the keystreams are correlation-free and resistant to statistical and cryptanalysis attacks.

The scientific novelty of Strike cipher lies in the utilisation of the stochastic behaviour of lightning strike phenomena to reduce operations complexity and enhance the performance

of data encryption. Such enhancements enabled Strike to work efficiently on a wide range of computing devices with limited resources. Strike outperforms other ciphers, including AES, Snow 2.0–128 and other well-known ciphers. The performance of Strike was tested on different environments with different computing capabilities. The results show that Strike managed to achieve a throughput of about 22,796 Mbit/s on a high-performance machine and about 21,143 Mbit/s on a low-performance machine. These results indicate that Strike is an efficient alternative for providing secure communication on various machines with different capabilities. In addition, further enhancement of Strike's performance is also possible through parallelism. The structure of Strike is designed to support multithreading, where multiple threads can be initiated to generate multiple keystreams simultaneously.

Author Contributions: All authors contributed equally to this research work. All authors discussed the results and contributed to the final manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created.

Acknowledgments: The authors would like to thank Arab Open University, Saudi Arabia, for supporting this study.

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

Abbreviations

Abbreviation	**Description**
IoT	Internet of Things
ECRYPT	European Network of Excellence for Cryptology
NIST	National Institute of Standards and Technology
XOR	Exclusive-OR
TLS	Transport Layer Security
4G	Fourth Generation Communication
RFID	Radio-Frequency IDentification
LFSR	Linear Feedback Shift Register
PRNG	Pseudo Random Number Generator
CML	Coupled Map Lattice
LE	Lyapunov exponents
IES	Image encryption structure
PSNR	Peak signal-to-noise ratio
GA	Genetic Algorithm
NFSR	Nonlinear feedback shift registers
LESCA	LightwEight Stream Cipher Algorithm
WLAN	Wireless Local Area Network
HTTPS	Hypertext Transfer Protocol
Mathematical Symbol	**Description**
ΔI	a change in the value of I in calculus.
σ	Standard deviation
φ	Phi (approx. 1.61803.)
$\oplus$	Exclusive-OR operation
$\sum$	Summation
lim	limit
log	logarithm
$sqrt$	Square root
sin	trigonometric functions of an angle
cos	trigonometric functions of an angle
p-value	Probability-value

References

1. Atawneh, B.; Abutaha, M.; Al-hammoury, L. Power Consumption of a Chaos-Based Stream Cipher Algorithm. In Proceedings of the 3rd International Conference on Computer Applications & Information Security, Riyadh, Saudi Arabia, 19–21 March 2020. [CrossRef]
2. Vavrenyuk, A.B.; Makarov, V.V.; Shurygin, V.A. Synchronous Stream Encryption Using an Additional Channel to Set the Key. *Procedia Comput. Sci.* **2021**, *190*, 797–802. [CrossRef]
3. ECRYPT Stream Cipher Project, March 2012. Available online: https://www.ecrypt.eu.org/stream/ (accessed on 20 September 2022).
4. Armknecht, F.; Mikhalev, V. On Lightweight Stream Ciphers with Shorter Internal States. In *Fast Software Encryption*; International Workshop on Fast Software Encryption; Springer: Berlin/Heidelberg, Germany, 2015. [CrossRef]
5. Ghafari, V.A.; Hu, H. Fruit-80: A Secure Ultra-Lightweight Stream Cipher for Constrained Environments. *Entropy* **2018**, *20*, 180. [CrossRef]
6. Hamannm, M.; Krause, M.; Meier, W. LIZARD—A Lightweight Stream Cipher for Power-Constrained Devices. *IACR Trans. Symmetric Cryptol.* **2017**, *2017*, 45–79. [CrossRef]
7. Mikhalev, V.; Armknecht, F.; Müller, C. On Ciphers That Continuously Access the Non-Volatile Key. *IACR Trans. Symmetric Cryptol.* **2016**, *2016*, 52–79. [CrossRef]
8. Cannière, C. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. *Lect. Notes Comput. Sci.* **2006**, *4176*, 71–186. [CrossRef]
9. Babbage, S.; Dodd, M. The Stream Cipher MICKEY 2.0. December 2022. Available online: http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf (accessed on 10 November 2022).
10. Ågren, M.; Hell, M.; Johansson, T.; Meier, W. Grain-128a: A New Version of Grain-128 with Optional. *Int. J. Wirel. Mob. Comput.* **2011**, *5*, 48–59. [CrossRef]
11. Mihaljevic, M.; Gangopadhyay, S.; Paul, G.; Imai, H. Generic Cryptographic Weakness of K-Normal Boolean Functions in Certain Stream Ciphers and Cryptanalysis of Grain-128. *Period. Math. Hung* **2012**, *65*, 205–227. [CrossRef]
12. Stankovski, P. Greedy Distinguishers and Nonrandomness Detectors. *Lect. Notes Comput. Sci.* **2010**, *6498*, 210–226. [CrossRef]
13. Almanasra, S. Parallel Platform for Supporting Stream Ciphers Over Multi-core Processors. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*, 181–190. [CrossRef]
14. Daldoul, I.; Tlili, S. Secured Transmission Design Schemes Based On Chaotic Synchronization and Optimal High Gain Observers. *Simul. Model. Pract. Theory* **2022**, *120*, 102625. [CrossRef]
15. Ding, L.; Liu, C.; Zhang, Y.; Ding, Q. A New Lightweight Stream Cipher Based on Chaos. *Symmetry* **2019**, *11*, 853. [CrossRef]
16. Turan, M.; McKay, K.; Chang, D.; Calik, C.; Bassham, L.; Kang, J.; Kelsey, J.E. *Status Report on the Second Round of the NIST Lightweight Cryptography Standardization Process*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2021. [CrossRef]
17. Noura, H.; Salman, O.; Couturier, R.; Chehab, A. LESCA: LightwEight Stream Cipher Algorithm for Emerging Systems. *Ad Hoc Netw.* **2023**, *138*, 102999. [CrossRef]
18. Sarajčev, I.; Sarajčev, P.; Vujević, S. Mathematical Model of Lightning Stroke Development. In Proceedings of the 16th International Conference on Software, Telecommunications and Computer Networks, Split, Croatia, 25–27 September 2008. [CrossRef]
19. Aumasson, J. *Serious Cryptography: A Practical Introduction to Modern Encryption*; No Starch Press: San Francisco, CA, USA, 2018; ISSN 978-1593278267.
20. Rakov, V.; Uman, M. *Lightning: Physics and Effects*; Cambridge University Press: Cambridge, UK, 2007; ISSN 9780521035415.
21. Golde, R.H. *Lightning, Volume 1: Physiscs of Lightning*; Academic Press: London, UK, 1977; ISSN 978-0122878015.
22. Su, R.; Wang, J.; Cai, L.; Zhou, M.; Fan, Y.; Cao, J.; Wang, F.; Wang, J. Characteristics of Dart Leader and Attempted Leader in A Triggered Lightning. *Electr. Power Syst. Res.* **2023**, *214*, 108812. [CrossRef]
23. IEC. *Protection against Lightning—Part 1: General Principles*; IEC: Geneva, Switzerland, 2006.
24. Ghasemi, F.; Babaie, S. A lightweight Secure Authentication Approach Based on Stream Ciphering for RFID-based Internet of Things. *Comput. Electr. Eng.* **2022**, *102*, 108288. [CrossRef]
25. Ding, Y.; Tan, F.; Qin, Z.; Cao, M.; Choo, R.; Qin, Z. DeepKeyGen: A Deep Learning-Based Stream Cipher Generator for Medical Image Encryption and Decryption. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *33*, 4915–4929. [CrossRef] [PubMed]
26. Fan, C.; Ding, Q. A Novel Image Encryption Scheme Based On Self-Synchronous Chaotic Stream Cipher and Wavelet Transform. *Entropy* **2018**, *20*, 445. [CrossRef]
27. Hasan, F.S.; Saffo, M.A. FPGA Hardware Co-Simulation of Image Encryption Using Stream Cipher Based on Chaotic Maps. *Sens. Imaging* **2020**, *12*, 35. [CrossRef]
28. Liu, Z.; Wang, Y.; Zhao, Y.; Zhang, L. A Stream Cipher Algorithm Based On 2D Coupled Map Lattice and Partitioned Cellular Automata. *Nonlinear Dyn.* **2020**, *101*, 1383–1396. [CrossRef]
29. Noura, H.N.; Salman, O.; Couturier, R.; Chehab, A. LoRCA: Lightweight Round Block and Stream Cipher Algorithms for IoV Systems. *Veh. Commun.* **2022**, *34*, 100416. [CrossRef]
30. Khedr, W.I. A New Efficient and Configurable Image Encryption Structure for Secure Transmission. *Multimed. Tools Appl.* **2020**, *79*, 16797–16821. [CrossRef]
31. Sudeepa, K.B.; Aithal, G.; Rajinikanth, V.; Satapathy, S.C. Genetic Algorithm Based Key Sequence Generation for Cipher System. *Pattern Recognit. Lett.* **2020**, *133*, 341–348. [CrossRef]

32. Zheng, J.; Hu, H. A Highly Secure Stream Cipher Based on Analog-Digital Hybrid Chaotic System. *Inf. Sci.* **2022**, *587*, 226–246. [CrossRef]
33. Belmeguenai, A.; Ahmida, Z.; Ouchtati, S.; Djemii, R. A Novel Approach Based On Stream Cipher for Selective Speech Encryption. *Int. J. Speech Technol.* **2017**, *20*, 685–698. [CrossRef]
34. Suwais, K. Stream Cipher Based on Game Theory and DNA Coding. *Intell. Autom. Soft Comput.* **2022**, *33*, 1815–1834. [CrossRef]
35. Maksymovych, V.; Shabatura, M.; Harasymchuk, O.; Shevchuk, R.; Sawicki, P.; Zajac, T. Combined Pseudo-Random Sequence Generator for Cybersecurity. *Sensors* **2022**, *22*, 9700. [CrossRef]
36. Rukhin, A.; Soto, J.; Nechvatal, J. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2010.
37. Kuznetsov, O.; Potii, O.; Perepelitsyn, A.; Ivanenko, D.; Poluyanenko, N. Lightweight Stream Ciphers for Green IT Engineering. In *Green IT Engineering: Social, Business and Industrial Applications*; Studies in Systems, Decision and Control; Springer: Berlin/Heidelberg, Germany, 2019; Volume 171. [CrossRef]