Revista Innovación y Software
Vol. 6, No. 2, Mes Septiembre - Febrero, 2025
ISSN: 2708-0935
Pág. 134-158
https://revistas.ulasalle.edu.pe/innosoft

Tipo de artículo: Artículos originales
Temática: Redes y seguridad informática
Recibido: 21/02/2025 | Aceptado: 21/03/2025 | Publicado: 30/09/2025

Identificadores persistentes:
DOI: 10.48168/innosoft.s24.a317
ARK: ark:/42411/s24.a317
PURL: 42411/s24.a317

A Case Study of the Salsa20 Encryption Algorithm Using Random Noisy Injection Enhanced by Artificial Intelligence

A Case Study of the Salsa20 Encryption Algorithm Using Random Noisy Injection Enhanced by Artificial Intelligence

Edgar Rangel Lugo[1][0000-0002-9611-8323]*, **Kevin Uriel Rangel Ríos**[2][0009-0001-7029-7183], **Carlos Alberto Bernal Beltrán**[3][0009-0009-3244-8268], **Leonel González Vidales**[4][0000-0002-2623-7626], **César Del Ángel Rodríguez Torres**[5][0000-0001-9198-4372], **Lucero De Jesús Ascencio Antúnez**[6][0009-0008-0923-4707], **Rosa Isabel Reynoso Andrés**[7][0009-0001-8896-8328]

[1]Tecnológico Nacional de México. I.T. de Ciudad Altamirano. Av. Pungarabato Ote. S/N, Col. Morelos, Ciudad Altamirano, Guerrero, México. erangel_lugo@hotmail.com
[2]Tecnológico Nacional de México. I.T. de Ciudad Altamirano. Av. Pungarabato Ote. S/N, Col. Morelos, Ciudad Altamirano, Guerrero, México. kgvppro@gmail.com
[3]Tecnológico Nacional de México. I.T. de Ciudad Altamirano. Av. Pungarabato Ote. S/N, Col. Morelos, Ciudad Altamirano, Guerrero, México. carlosalberto.bb@cdaltamirano.tecnm.mx
[4]Tecnológico Nacional de México. I.T. de Ciudad Altamirano. Av. Pungarabato Ote. S/N, Col. Morelos, Ciudad Altamirano, Guerrero, México. leonel.gv@cdaltamirano.tecnm.mx
[5]Tecnológico Nacional de México. I.T. de Ciudad Altamirano. Av. Pungarabato Ote. S/N, Col. Morelos, Ciudad Altamirano, Guerrero, México. cesardelangel.rt@cdaltamirano.tecnm.mx
[6]Tecnológico Nacional de México. I.T. de Ciudad Altamirano. Av. Pungarabato Ote. S/N, Col. Morelos, Ciudad Altamirano, Guerrero, México. lucerodejesus.aa@cdaltamirano.tecnm.mx
[7]Tecnológico Nacional de México. I.T. de Ciudad Altamirano. Av. Pungarabato Ote. S/N, Col. Morelos, Ciudad Altamirano, Guerrero, México. rosaisabel.ra@cdaltamirano.tecnm.mx

*Autor para correspondencia: erangel_lugo@hotmail.com

Resumen
La pérdida o robo digital de datos, puede generar severas consecuencias financieras para las organizaciones. Dicha situación puede ser manejada utilizando métodos de cifrado dinámicos, los cuales, son capaces de generar diferentes textos cifrados partiendo de una misma entrada de texto plano. La presente investigación, involucra el uso de inteligencia artificial (IA), aplicado a la ciberseguridad, a través del uso de estrategias: .ªleatorias ruidosas"(random noisy), siendo estudiados varios aspectos relacionados en dicho campo. La aplicación de estos esquemas: random noisy, sobre el texto cifrado, es recomendado como medida para mejorar la seguridad de los datos digitales. Estudios recientes refieren que ello puede ser efectivo para contrarrestar posibles ciberataques. Por lo tanto, el objetivo de la presente investigación fue enfocado sobre la aplicación de una nueva propuesta basada en estrategia: random noisy, siendo aplicada a textos cifrados obtenidos por el algoritmo Salsa20, debido a que, ha sido poco estudiado en la literatura. Lo anterior, sustentado en que las estrategias .ªleatorias ruidosas", han revelado resultados muy prometedores, en previas investigaciones. El presente trabajo, se considera importante, debido a que, los resultados experimentales han recomendado la efectividad del uso de IA, aplicado

Revista Innovación y Software
Vol. 6, No. 2, Mes Septiembre - Febrero, 2025
ISSN: 2708-0935
Pág. 134-158
https://revistas.ulasalle.edu.pe/innosoft

en la inyección de ruido a textos cifrados, mostrando incremento en el esquema de seguridad de los datos digitales encriptados. Por lo tanto, es presentada aquí una nueva metodología mejorada, para las organizaciones que utilizan cifrado de datos con el algoritmo Salsa20, así como, la comparación de resultados con cuatro random noisy.[es]trategias, basadas sobre el cifrado con los algoritmos: DES, 3DES, AES-256, y Blowfish. En conclusión, la nueva alternativa, aquí presentada, es recomendada porque mostró ser de difícil desencriptado, incluso para las herramientas estándar de los ciber-criminales.

Palabras claves: Aplicaciones de IA, criptografía, cifrado de datos dinámico, cifrado con inyección de ruido.

Abstract
The loss of digital data can have severe financial consequences for organisations. This situation can handle with a dynamic encryption approach, which it can generate multiple ciphertexts from a single plaintext. This paper delves into the intersection of artificial intelligence, random noisy schemes, and cybersecurity, analyzing several critical aspects of this emerging field. The application of random noisy scheme to ciphertext, it has been suggested as a means of enhancing cybersecurity, with recent studies indicating that this approach can be effective against cybercriminals. The objective of this research was focused on applying random noisy strategy on ciphertext of the Salsa20 encryption algorithm, as this area remains unexplored. Given the promising results of random noisy strategies, organisations that they employ Salsa20 may benefit from this approach. This research is important because recent experiments have revealed the effectiveness of using artificial intelligence for noisy injection on ciphertext. Therefore, is here presented a new opportunity for organisations regarding the employment of the novel random noisy Salsa20 encryption alternative because the experimental results have shown an increasing of the dynamic performance on ciphertext. This work introduces the random noisy Salsa20 strategy as a novel dynamic encryption alternative, and comparison of the performance to four random noisy schemes based on DES, 3DES, AES-256, and Blowfish algorithms. In conclusion, the novel alternative that it is here recommended, it can be difficult for the cybercriminals to decrypt.

Keywords: *Applications of AI, cryptography, dynamic encryption methods, noisy injection strategies.*

Introduction

A cybersecurity strategy [1–3], it is deemed insufficient if even one of its methods is susceptible to cyberattacks [4–8], which can lead to the theft of sensitive digital data [8, 9]. In practical applications, this can result in substantial financial losses for organisations [4, 5, 8, 10, 11]. These cases [12–17] typically involve cyber threats such as phishing, fraudulent calls, and financial scams, impacting social networking platforms, bank systems, large markets, energy networks, and e-commerce in organisations [4, 5]. Several approaches have been proposed to tackle the issue of digital data theft. These can be broadly categorised into three main strategies: Firstly, regularly updating cybersecurity strategies [4]. Secondly, utilising dynamic encryption methods [5]. Thirdly, employing noisy injection strategies on ciphertext [5–7, 9, 10], which they have shown promise in certain practical domains. This study focuses on encryption methods [8], that they utilise noisy injection strategies [5–7]. These methods involve computing statistics or mathematical equations to transform plaintext (Si) into ciphertext (Ci) [8], which can only be deciphered by authorised individuals or systems [10, 11]. The Si represents the original input data, while the Ci is the encrypted output. An encryption method is dynamic if it produces unique results with the same plaintext data input each time, whereas static encryption methods produce the

same output [8]. Symmetric encryption algorithms, they use a single secret key, while asymmetric encryption algorithms use a pair of keys, one private and one public [8]. Data encryption refers to the conversion of plaintext into ciphertext, whereas data decryption is the reverse process [8–10, 14, 18]. Asymmetric algorithms like RSA (Rivest-Shamir-Adleman) [3, 5, 6, 8, 13, 14, 19–27], ECC [5, 6, 8, 19–23, 28–31], and ElGamal [13, 19–21], they use asymmetric key cryptography, which involves a pair of keys - private and public. Dynamic encryption has shown promising results in recent research [5, 8, 11], when applied to asymmetric algorithms. Since asymmetric algorithms are not within the scope of this work, they are not examined here and could be considered for future works. There are also various symmetric key cryptography algorithms, such as DES (Data Encryption Standard) [3, 13, 14, 22, 23, 27, 32, 33], 3DES or TripleDES (Triple Data Encryption Standard) [14, 21–23, 34], Blowfish [22, 23, 35–38], Salsa20 [38, 39], and AES (Advanced Encryption Standard) [13, 21–23, 26, 27, 40, 41], to mention a few. In this research, the AES-256 version [38, 40, 41], it has been employed. In reference [22], it is noted that AES is a symmetric block cipher that can operate with different block sizes and supports key lengths of 128, 192, and 256 bits. DES processes 64 bits of plaintext to produce 64 bits of ciphertext, using a series of substitution and permutation rounds, and decryption is achieved by reversing the encryption process. Due to the limited key size of 64 bits, DES is considered insecure for modern standards. To improve security, 3DES was developed as a more robust alternative. Another symmetric algorithm, Blowfish, employs a variable-length block cipher with key lengths spanning from 32 to 448 bits [22, 23]. Although Blowfish is versatile, it is often used with a block size of 64 bits. Regarding the symmetric algorithm Salsa20 [39], it is ChaCha20's predecessor. Both algorithms are not block-cipher because they work with keystream generation. According to [39], Salsa20 is addition/rotation/XOR (ARX)–based cryptographic primitive. Their keystream generation algorithm comprises three simple operations: Addition modulo 2^{32} ($\boxplus$), constant distance left bit rotation (n), and bitwise XOR operation. These operations are swift in any circuit, and hence the cipher can achieve significant speed with a high security margin. The Salsa20 cipher takes a 256-bit key (k), a 128-bit constant (c), and a 128-bit initial vector (v), or IV, and generates a 512-bit keystream. The key is divided into eight 32-bit words $(k_0, k_1, \ldots, k_7)$. Similarly, the constant and the IV are also broken into 32-bit words (c_0, c_1, c_2, c_3) and (v_0, v_1, t_0, t_1). Due to the use of the 256-bit key, the cipher is also called 256-bit Salsa20. In the original Salsa20 cipher, the number of rounds is 20. After the final round, the initial state X is added modulo (2^{32}) to the updated state X^{20} word-by-word, and the keystream (Z) of 512 bits is achieved, that is, $Z = X \boxplus X^{20}$. This keystream is then bitwise XORed to plaintext to get the ciphertext. Besides, the Salsa20 scheme, it is commonly generated a 512-bit keystream. This proposal is appealing for encryption due to its high speed and security. Due to their efficient algorithm and fast performance, it has been attracted by cryptographic analysis since their release [39]. Some research [5, 8], they have revealed that symmetric encryption algorithms, they have reported static ciphertext results. Despite that it does not mean, in all cases that they are vulnerable schemes [5]. However, there exists some standard encryption alternatives that they have been threatened by quantum computing [8, 22, 23, 42]. In contrast to some views, other researchers [43–45], they emphasize that the AES and

RSA algorithms are at risk due to advancements in quantum computing [8,22,42]. This paper investigates several facets of this issue and, without using quantum computing, proposes measures that can be applied to symmetric algorithm variants to improve their security [5]. For example, as pointed out by some authors [4–11], the dynamic encryption methodologies, they offer a good alternative to increasing noisy and redundant information in ciphertext outputs [5,9,10]. Moreover, the employment of noisy injection strategies [5–7], they are recommended as good encryption measure. The noisy injection means that a plaintext or ciphertext contains some additional characters of the ASCII or UTF-8 encoding, which they are not part of the original input message [8]. A key hypothesis of this research is that these strategies can create confusion for cybercriminals, as indicated in [5,8]. These strategies [5–7], frequently utilize artificial intelligence (AI) [5–8,10,11,46,47], as AI-based cryptography [1–4, 8, 10, 11, 16, 17, 24, 48–50], it is a well-established field. According to [10, 46, 47], the purpose of AI is to enable machines to think [8]. In this context, various approaches like heuristic methods [5, 8, 10], they have been proposed, which involve predefined rules to solve problems. These can be applied to implement structured models such as decision trees [51–53] and graphs [53–55], among others. Moreover, there exists alternatives with AI based on random methods [10, 56–59], it consists in the selection of random numbers without replacement or with replacement (i.e., it does not repeat or it repeats values, respectively) [8]. These methodologies can be employed with intelligent models such as genetic algorithms (GAs) [1, 2, 9, 10, 16, 17, 56, 59–62], Monte Carlo (MC1) algorithm [59], artificial neural networks [57, 63, 64], to mention few. Several authors have explored AI-based cryptography alternatives [1–4, 8, 10, 11, 14, 16, 17, 24, 27, 48–50, 60–62]. For a comprehensive review of AI applications in cryptography, see [50]. In [48], it is analyzed trends and challenges, while in [3], it focuses on asymmetric and symmetric cryptographic methods. In [58], a study on pairing functions for AI-driven cryptography was conducted. Subsequently, the same author [16], he published a novel investigation on GAs in cryptography, specifically contributing to the field of e-commerce. In addition, to existing research, in [24] reports on an improved cryptanalysis of RSA using side-channel attacks. Meanwhile, reference [49], it employs a new approach to collision attacks through differentiated path construction. In line with established research practices, genetic algorithms for cryptography have been explored in [1, 62]. Similarly, in [60], it introduces modern optimization algorithms designed for cryptanalysis. On the flip side, reference [61], it reveals that genetic algorithms can successfully break certain simple cryptographic ciphers. In a similar vein, it is examined in [27], the application of genetic operators to symmetric cryptography using GAs. Moreover, in [49], it is introduced a quantum cryptanalysis technique for lattice-based protocols. Concerning the noisy injection strategies [4–11], there exists several alternatives that they can be grouped in several categories. Initially, pseudo-hexadecimal format is utilized. The "Noised" random pseudo-hexadecimal GAs methodology, in particular, it has been discussed in references [9,10]. A dynamic encryption scheme based on genetic algorithms was proposed. However, because of the disadvantages associated with pseudo-hexadecimal GAs, a successor methodology, "Noised" random pseudo-hexadecimal (without GAs), it was presented in [10]. Later, in [8], it presented four dynamic alternatives based on the pseudo-hexadecimal schema, known as «noisy random pseudo-hexadecimal» strategies.

The goal of these strategies is to confuse cybercriminals by injecting noise into ASCII characters when utilizing a novel pseudo-hexadecimal format. These schemes are limited to plaintext applications. The second approach involves combining AI-based noisy injection with the 1-NN rule, as discussed by other research [46,47,65–67]. In this regard, the Ñoised'random 1-NN with hexadecimal encoding based on AI was introduced in [11]. Similarly, the combination of "Noised" random pseudo-hexadecimal format with 1-NN was explored in [68]. These two approaches have shown that implementing double noisy injection over ciphertext outputs can improve data safety, though it requires sacrificing some disk storage space. These approaches were also used with plaintext. The pseudo-hexadecimal scheme, it was not examined in this paper, but it might be addressed in future works. Thirdly, the random Caesar II mod 120 [4–7, 11], it is utilized for noisy injection, applied to both plaintext [4, 5, 11], and ciphertext [5–7], which it was obtained through standard encryption algorithms. This method is referred to as the random noisy strategy [5–7]. Reference [5], it highlights that the random Caesar proposal, based on the traditional Caesar algorithm [4, 9, 13, 14], stands out due to its dynamic encryption with AI, utilizing heuristic methods. While the traditional Caesar cipher relies on a fixed shifting value K, as expressed in $Ci = Si + K$ (mód 26) [13, 14], the random Caesar utilizes varying shifting values (Ki) for each character Si, chosen randomly with replacement. The operation: $Ci = Si + Ki$ (mód N), it allows random Caesar to adapt to different N values. In the case of the traditional Caesar cipher with (mód 26), the alphabet is restricted to 26 characters for encryption and decryption purposes. While the (mód N) in random Caesar, it can be dynamic, recent studies [4, 6, 9, 11], they have focused on three specific modes: random Caesar I with (mód 9) and (mód 255) [6], random Caesar II with (mód 95) [4, 9, 11], and random Caesar II with (mód 120) [4–7]. The selection of alphabets in these schemes is defined by random conditions, and a heuristic method is applied to find the most effective Ki vector. Here, the (mód N) value indicates the size of the encryption alphabet and the maximum value that can be present in the Ki vector. With $N = 95$, ASCII characters from space (32) to '∼'(126) are used. $N = 120$ covers a range from 30 to 150 in the ASCII table. For different N values, Ki falls between 0 and N. The random Caesar scheme's procedure includes a second phase intended to confuse potential attackers [4]. This phase calculates the final package as $FinalPackage = Ci \ \& \ Ki \ \& \ OrdChr(Ci)$ [5]. The & operator indicates concatenation, and the $OrdChr$ procedure either appends the character Ci when $N = 120$ or converts it to an ordinal value depending on the mode. This method is limited to plaintext encryption as a dynamic strategy [4], while random Caesar II mod 120 has been employed in some studies [5–7], as a means of introducing randomness. Reference [5], it presents eight alternatives that integrate random noise with standard encryption algorithms: random noisy DES, random noisy 3DES, random noisy RC4, random noisy Blowfish, random noisy WEP, random noisy AES, random noisy RSA-2048, and random noisy ECIES SECP-256-R1. The approach integrates standard encryption techniques with random Caesar II mod 120 for noisy injection. Similarly, the concept of random noisy GOST was explored in [6], and the random noisy Camellia was highlighted in [7]. Some variants of random noisy strategies [4], they focus on noisy injection and they were designed to camouflage ciphertext [5]. Notable

examples are reduced random Caesar [4, 5], and reduced random mutation [4]. According to [4], the primary purpose of these schemes is to camouflage and reduce the size of at least 1/3 of the ciphertext. The reduced random Caesar strategy has been applied to both plaintext [4], and ciphertext [5], whereas the reduced random mutation [4], it has been used exclusively with plaintext. These reduced random and mutation strategies, they were not here experimented because it can be considered in future works. We assessed four random noisy strategies that they utilize standard symmetric encryption algorithms, including random noisy DES, random noisy 3DES, random noisy AES-256, and random noisy Blowfish. Due to that novel alternative, named here as random noisy Salsa20, it was developed and it is introduced in this study. The application of noisy injection with random noisy strategies [4–7], to plaintext [4], or ciphertext [5–7], they are recommended due to its strong correlation with dynamic encryption performance. There exists a lack of research on the vulnerabilities of these schemes when subjected to quantum computing [5, 8]. Given that random noisy strategies have shown promise, this research continues the work of [5–7], by examining the Salsa20 encryption algorithm's potential when applied to ciphertext, a gap in existing research that could benefit organizations employing Salsa20. This circumstance also allows organizations to consider employing noisy injection techniques derived from the Salsa20 scheme. This work explores the use of noisy random encryption in cybersecurity, specifically the technique of applying noisy injection to ciphertext from standard algorithms to potentially deceive cybercriminals [8]. The scope of this study was limited to two classes of situations. The initial step involved comparing five standard encryption algorithms (DES, 3DES, AES-256, Blowfish, and Salsa20) as static encryption schemes, with outcomes benchmarked against related research [5–7]. Furthermore, the random noisy Salsa20 scheme, it was developed to serve as a comparative benchmark for other random noisy strategies (random noisy DES, random noisy 3DES, random noisy AES-256, and random noisy Blowfish) in the context of noisy injection on ciphertext. This method uses random Caesar II mod 120 [4], being applied on ciphertext by standard encryption algorithms. The focus is on dynamic encryption as a means to achieve randomness in performance. It also makes an empirical contribution to cryptography using AI, as the literature on random noisy strategies is sparse. Despite that recent research [4, 5, 9–11], they have reported that random Caesar II [4], it might obtain random values for the ciphertext that they can be selected out of the range of the ASCII table [5]. In practical domains that they were evaluated with random noisy strategies [5], these problems have not been observed. The random noisy encryption strategies were previously assessed with five-fold cross-validation [5–7]. This work expresses their performance in terms of global average or accuracy [46, 47, 65, 66]. We present the findings of an extensive research project investigating digital data theft, including experimental results. This study analyzed cases where inadequate static encryption methods were used, causing security concerns [5]. Initially, the experiments were focused on replacing of the static encryption schema for recommending the random noisy strategies as dynamic encryption alternative [4]. Several ciphertext exemplars resulting from random noisy encryption schemes are shown here. These approaches were evaluated over five samples when a novel modification of cross-validation [4,5,10], it has been employed. Based on observations, noisy injection over

ciphertext output is recommended due to its excellent performance as a dynamic encryption indicator. This paper concludes by introducing the random noisy Salsa20 strategy, a novel dynamic data encryption approach. In addition, the comparison of results with other four random noisy schemes based on DES, 3DES, AES-256, and Blowfish, algorithms, they are also introduced. Finally, this work is justified because the novel random noisy Salsa20 is rarely studied, thereby it has not revealed vulnerabilities in literature regarding to cyberattacks or quantum computing strategies.

Computational methodology

The use of static encryption algorithms as a replacement for existing cybersecurity strategies does not ensure data protection for organizations. A dynamic encryption approach is suggested instead [4]. We evaluate several dynamic encryption approaches that utilize random noisy strategies [5], to address the issue of digital data theft. This research, it is considered experimental and exploratory. Due to that the novel alternative that it is based on random noisy Salsa20 scheme, it is here first introduced. This work required the use of hardware, software, and datasets. The experiments were conducted on a personal computer with a 2 GHz CPU, 4 GB of RAM, and 32 GB of free disk space. The software implementation of the encryption methods, including DES, 3DES, AES-256, Blowfish, and Salsa20, as well as the novel variants based on random noisy strategies [5], these experiments were carried out using Microsoft Windows 10 [69], and Python language [70]. To compare our results with those in [5–7], we repeated some experiments on a mobile computing device with the same hardware features as the personal computer mentioned above, but with Android 9 operating system [71], as well as the PyDroid3 app [72], for software development. Our experiments showed no significant differences. Our datasets were composed of training samples (TS) [6, 7, 46, 47, 65], which included 1000 randomly created exemplars. Each row in the dataset features a pattern with five columns or features. The data contains encryption and decryption details, including ciphertext (Ci) represented as a pair of exemplars ($Test1, Test2$), as well as metrics like encryption time (TC), decryption time (TD), error rate, and class label. The pattern format is $TP = [(Test1, Test2), TC, TD, Error, Label]$, which it allows for direct comparison with other studies [7]. In Table 1, we display two ciphertext exemplars for the label "#Welcome#?". The label feature corresponds to the plaintext (Si), which it is simulating a password and it includes noisy characters '#ánd '?'(with ordinal values 9619 and 65533, respectively). The encryption and decryption times, they were calculated in milliseconds, and the TC, TD, and error features were represented as double precision values. The encryption strategy converts the plaintext sequence into a ciphertext result, represented as a tuple ($Test1, Test2$), while TC is computed, allowing us to observe the dynamic nature of the encryption results. During decryption of the ciphertext sequence, TD was calculated. Both sequences are then stored in TS, including their respective TD, TC, and error rates, using a predefined structured pattern. The error rate was calculated based on the number of characters with errors. If the ciphertext obtained through the encryption strategy does not match the plaintext

(Si), the error rate is determined by the proportion of Si that contains errors. In these terms, if for a ciphertext of eight characters, its correspondent plaintext was also of eight characters, thereby an error value of 0.5, it means that four characters from $Test1$ or $Test2$, they have not be decrypted properly, and so on. The ciphertext and plaintext are represented as sequences of characters using ASCII or UTF-8 encoding, and each sequence is limited to a maximum of 255 characters. The 3DES and Blowfish algorithms were exceptions, as they couldn't support sequences of 255 characters. Consequently, Blowfish was experimented with up to 13 characters, and 3DES with up to 22 characters. The encryption times being evaluated mean that the comparison of algorithms is not entirely on equal terms. To address this, the study experimented with ciphertexts filled with random hexadecimal values to provide a more equitable assessment. This information was utilized to transform it into a new format based on cross-validation modifications [4–7]. The updated TS format was then applied to each encryption strategy individually. To enhance result interpretation, thereby, the Table 1 presents the arithmetic mean and standard deviation for a plaintext with the value "#Welcome#?", which it has been processed using most encryption algorithms under identical conditions. Finally, separate TS were created for each encryption algorithm tested, including those with random noisy strategies.

First adopted strategy, it involves utilizing standard symmetric encryption algorithms, specifically DES, 3DES, AES-256, Blowfish, and Salsa20. They were experimented as static encryption schemes being applied over plaintext for results comparison with other research [5–7]. We carried out these experiments on five training samples containing the mentioned information, assessing each encryption algorithm individually through a modified cross-validation approach [4,5]. These standard encryption algorithms were implemented in Python [70], utilizing libraries such as cryptography [73], pycryptodome [74,75], and pycrypto/pycryptor, which they were installed as part of the process. Therefore, it needs to be imported into the source code as follows: *from cryptography.hazmat.primitives import padding ; from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes ; from cryptography.hazmat.backends import default_ backend, from Crypto.Cipher import DES, and from Crypto.Cipher import Salsa20.* Thereby, it is assumed that Si = "#Welcome#?. Using the DES algorithm, the ciphertext can be computed according to the following operation: $Ci = ((A(Key.encode(), Mode)).encrypt(p$ In the case of ciphertext produced by the 3DES and Blowfish algorithms, the computation source-code is: $Ci = (Ri.update(plaintext) + Ri.finalize()).hex(); print(Ci)$. Similarly, the ciphertext for the AES-256 encryption alternative can be computed as follows: $Ci = (IV.encode()+(Ri.update(plaintext)+Ri.finalize())).hex(); print(Ci)$. The ciphertext generated using the Salsa20 algorithm is presented below: $Ci = ((A(Key.encode(), Nonce.encode())).en$ Where, the A component refers to the used algorithm, while that the encoded Key parameter corresponds to the secret key. The IV value, it is the initialization vector. The $Mode$ argument, it is a valid operation mode of the algorithm. The $Nonce$ argument, it is a valid nonce value of the Salsa20 algorithm. The plaintext argument is the encoded Si. The *encrypt()* procedure returns ciphertext object, which is a class component. The N value indicates the maximum number of bytes in a character sequence. The Ri com-

ponent represents a partial ciphertext object that is either unpadded or incomplete. The Qi component denotes the padding used. The *updated()* and *finalize()* procedures allow for the completion of the encryption operation. In the end, the *hex()* function converts byte values to their corresponding hexadecimal representations. In these terms, some valid values for obtaining a ciphertext with DES algorithm, it can be computed as follows: *Key = "00000001"; A = DES.new ; N = 8 ; Mode = DES.MODE_ECB ; plaintext = Si.encode() + (b"\x00"* (N-len(Si.encode()) % N))*. The valid parameters for computing ciphertext with the Blowfish algorithm can be obtained through: *Key = "00000001"; A = algorithms.Blowfish ; N = 16; Mode = modes.ECB() ; Ri = Cipher(A(Key.encode()), Mode , default_backend()).encryptor() ; plaintext = (Si + + str(.join([for k in range(0,int(N-len(Si.encode())))]))).encode()*. To produce ciphertext with 3DES, the valid values can be obtained through: *Key = "000000000000000000000001"; A = algorithms.TripleDES ; N = 24 ; Mode = modes.ECB() ; Ri = Cipher(A(Key.encode()), Mode , default_backend()).encryptor() ; plaintext = (Si + + str(.join([for k in range(0,int(N-len(Si.encode())))]))).encode()*. Similarly, the AES-256 encryption version requires the following valid parameters for ciphertext generation: *Key = "00000000000000000000000000000001"; A = algorithms.AES ; N = 256 ; IV= "0000000000000001"; Mode = modes.CBC(IV.encode()) ; Ri = Cipher(A(Key.encode()), Mode , default_backend()).encryptor() ; Qi = padding.PKCS7(N).padder() ; plaintext = Qi.update(Si.encode()) + Qi.finalize()*. For the Salsa20 algorithm, the valid parameters for ciphertext generation are: *Key = "0000000000000000000000000000001"; N = 32 ; Nonce = "00000001"; paintext = Si.encode() ; A = Salsa20.new ; Mode = None*. Each of these statements needs to be included in the source code before invoking Ci, as applicable.

Second adopted strategy, it consists in application of random noisy alternatives [5], for dynamic data encryption. In [4], it is recommended because that these schemes can provide a good alternative for increasing the noise in ciphertext outputs. The strategies outlined in [5], they were applied to ciphertexts produced by standard encryption algorithms, specifically examining four cases: random noisy DES, random noisy 3DES, random noisy Blowfish, and random noisy AES-256. Due to that this study also introduces random noisy Salsa20 as a novel approach. We implemented the five random noisy strategies in Python [70], and they were tested using a noisy injection application that combines random Caesar II mod 120 with the ciphertext from each standard encryption algorithm. This allowed for a direct comparison with other studies [5–7]. Each encryption algorithm was evaluated separately on the five TS using an iterative process with five repetitions of cross-validation [4,5]. We applied modified cross-validation to calculate the global average and standard deviation for each encryption strategy. The novel proposals, as described in [5], involve noisy injection into ciphertext, and the procedure for computing random noisy strategies is detailed in [5–7]. We computed it as follows: $RandomNoisyi = Char(Ord(StandardEncryptioni)+Ord(Ki))$ & $Char(Ki)$ & $Char(StandardEncryptioni), with$ (mód 120). By replacing the plaintext with ciphertext, the operation was streamlined, as shown in [4]. The adjustment to (mód 120) is made because $FinalPackage$, or $RandomNoisyi$, only stores character types. As discussed

in [5], several versions of random noisy schemes have been explored. In this research, only four strategies for obtaining the $StandardEncryptioni$ ciphertext, they have been employed as mentioned above. In the same way, the random noisy Salsa20, it was implemented and can be computed as follows: $RandomNoisySalsai = Char(Ord(StandardSalsaEncryptioni)+Ord(Ki))$ & $Char(Ki)$ & $Char(StandardSalsaEncryptioni)$ (mód 120). Where: The + operator refers to the sum function, while the & operator corresponds to the concatenation function. The Ki shifting vector, it contains random integer values. The $StandardEncryptioni$ parameter, it refers to the ciphertext obtained using a standard encryption algorithm (e.g., DES, 3DES, Blowfish, and AES-256), as described in [5]. In a similar vein, the $StandardSalsaEncryptioni$ argument refers to the ciphertext obtained through the Salsa20 algorithm. Each strategy was applied separately. The $RandomNoisyi$ sequence is the $FinalPackage$ obtained through a random noisy strategy, as per [5]. In distinction, $RandomNoisySalsai$ denotes the $FinalPackage$ that results from employing random noisy Salsa20. The Ord function returns the ordinal value of a character or integer, and the $Char$ function converts a value to its corresponding character representation in ASCII or UTF-8. In the random noisy proposals, a standard encryption algorithm is first used to encrypt the plaintext, and then random Caesar II mod 120 is applied to the ciphertext, adding an extra layer of security [5–7]. In [5], it emphasizes that this approach differs from double encryption and warns against the repeated application of standard encryption algorithms, which could be exploited by cybercriminals using computational strategies to decrypt the data. As noted in [5], noisy injection should be applied selectively to the ciphertext to prevent arousing suspicion about the noise's location. By applying it to only a portion of the ciphertext, cybercriminals would encounter the daunting task of identifying the location of noisy characters, a challenge that remains substantial even with the advent of quantum computing. These strategies involve the use of random Caesar II mod 120 [5], applied to ciphertext previously obtained through a standard encryption algorithm. Hence, they are considered as dynamic encryption alternatives for random performance [4]. Utilizing random noisy strategies for encryption has resulted in dynamic ciphertext, which plays a crucial role in strengthening data security in organizational contexts. The random Caesar II methodology with (mód 120) is also an AI-based technique, leveraging random and heuristic methods to select the optimal Ki shift vector [5]. Thus, artificial intelligence was utilized when the heuristic method was applied to select the Ki vector that yields maximum values for the encryption alphabet. The similarity in procedures between the heuristic method and genetic algorithms leads to the consideration of AI application. This heuristic approach is employed to validate the selected ASCII characters [5,8], with further explanation provided in [8]. A genetic algorithm (GA) is a random procedure comprising selection, crossover, and mutation phases, followed by an evaluation stage utilizing a wrapper [9,10] or fitness function [10,16] for each GA generation [1,2,9,10,16,56,59–62]. In these terms, random Caesar methodology [4,5], only selection procedure of GA is employed for selecting the alphabets with (mód 120) and its Ki vector of shifts. In this case, the range of Ki values are delimited between 30 and 150, for it does not exceed the 255 ASCII value. Similarly, the reduced random Caesar strategy [4,5], can utilize the GA selection procedure with (mód 120). For reduced random mutation [4], the first phase (selection)

and third stage (mutation) of the GA model are employed. In both cases, the Ki shifting range is limited to ordinal values between 0 and 105 to avoid exceeding the ASCII table maximum. This research did not cover reduced random Caesar and reduced random mutation, but they could be explored in future work. Regarding result evaluation, a modified cross-validation method [4–7], it was proposed to internally bias the discrimination process, building on previous discussions. This information can help us recommend the employment of noisy injection as a safety measure in organisations.

Results and Discussion

All experiments were carried out using TS, as previously mentioned. Thereby, the estimated error, TC, and TD, they were computed separately for each encryption strategy, as described above. A five-fold modified cross-validation [4], it was employed for each TS, allowing for direct comparison with the results of other studies [5–7]. According to [7], some studies employing the traditional cross-validation method [46,47,56,59,63,65–67] divide the TS into five subsamples, each containing approximately 20 % of the data, with a similar number of elements per class. One subsample is then used as the test sample (MC) to evaluate the model. The remaining four subsamples (roughly 80 % of TS), they are joined together to form a training set, which is used to train the model. The model's performance is then evaluated using the MC as if it were new, unseen patterns. To obtain the standard deviation after calculating the average or global accuracy, the process is repeated five times [66]. This traditional procedure is not applicable to data encryption/decryption, as the evaluation of the encryption algorithm in this research does not require a training model with TS or evaluation with MC. Consequently, the training and evaluation tasks are carried out before the ciphertext or $FinalPackage$ is generated. Initially, the standard encryption algorithm is used to encrypt the plaintext, producing a ciphertext. This ciphertext is then decrypted, and both the original and decrypted vectors are stored in the TS. For random noisy strategies, the standard encryption algorithm is applied to the plaintext to generate a ciphertext. Then, the heuristic method is used to simulate a partial phase training, utilizing only the random selection stage of the GA, as applicable. The Ki vector is obtained by partial training using the random Caesar strategy, which it is applied to the ciphertext for noisy injection. The encrypted sequence is decrypted, and both vectors are stored in the TS. The novel cross-validation modification [4, 5], it has been updated to adopt a new approach that does not rely on MC for assessing global accuracy [66]. The data encryption/decryption process in this research uses a modified cross-validation approach [4–7], which it omits MC pattern evaluation. The error is computed using four subsamples of the TS, and this process is repeated five times, with about 20 % of the TS being sequentially omitted. The schema simulates the estimated error in different environments by omitting a part of the TS, which it ensures a more convergent result with an optimistic bias (i.e., the obtained value can be better or equal in practical applications) [7]. This approach allows for the assessment of the estimated error and other numerical features of the TS. If the decrypted ciphertext does not match the original plaintext, the

error percentage is determined by calculating the proportion of coded characters that they were not decrypted correctly. Similarly, as the traditional cross-validation does, the operation is repeated five times, extracting sequentially, a different subsample in each iteration. With purpose of calculating the average and standard deviation of each attribute or column of numeric type, which in this research, it was applied to the encryption times (TC), decryption times (TD), and error percentage, globally, without distinguishing the elements by class. As encryption ambiguity was noted during the experimentation process, the cross-validation was performed without distinguishing between classes. This topic may be revisited in future research to provide more comprehensive explanations. However, this situation has not affected the global accuracy of the encryption strategies here evaluated. As a result, the experimentation in this work was limited to two classes of situations. First, the comparison of five standard encryption algorithms (DES, 3DES, AES-256, Blowfish, and Salsa20), they were studied being applied to plaintext as static encryption schema for results comparisons with other authors [5–7]. Secondly, we evaluated four random noisy strategies from [5], using ciphertext, as previously mentioned. Specifically, we studied random noisy DES, random noisy 3DES, random noisy AES-256, and random noisy Blowfish, while the fifth strategy, random noisy Salsa20, it is introduced here as a novel alternative. Therefore, we experimented with the five random noisy strategies separately to inject noise into the ciphertext as a dynamic encryption approach. After processing all the samples for each encryption strategy separately, the global average results, they were computed using the novel updated cross-validation method [4–7], as explained above. This information can be found in Table 1, which includes the standard deviation in parentheses. For comparison with other research [5, 7], in columns TC and TD are shown the encryption and decryption times respectively, which they were measured in milliseconds. The iterative procedure was stopped in this study after five repetitions of ciphertext were generated for each encryption method. The results in Table 1, which they include the standard deviation in parentheses, they are based on the average of five sequential experiments here evaluated using the updated cross-validation method [5–11]. The parameters used for the standard encryption methods, they are described below: The DES algorithm was configured with a 56-bit secret key ('00000001'), UTF-8 encoding, and ECB mode [20], with ciphertext output in hexadecimal format. The algorithm's development leveraged the pycryptodome library from Python's language [74, 75]. To obtain ciphertext with the TripleDES (3DES) algorithm, the ECB format [20], and OpenSSL [21, 34] scheme, they were used with a 24-bit secret key ('000000000000000000000001'), resulting in hexadecimal-encoded ciphertext. The 3DES algorithm was implemented in Python using the cryptography package [73]. In the Blowfish algorithm, ECB format [20], with OpenSSL-based default_backend() and a 16-bit secret key ('00000001'), they were employed, producing hexadecimal-encoded ciphertext. The implementation was also carried out using Python's Cipher module from cryptography package [73]. For AES-256 experiments, a 256-bit secret key ('00000000000000000000000000000001') and a 128-bit IV vector ('0000000000000001'), they were utilized. The implementation involved CBC mode [20], with OpenSSL [21, 34], and PKCS7 padding [21, 34], with 128 bits, generating ciphertext in hexadecimal format. Python's implementation leveraged the cryptography packa-

ge [73]. For the Salsa20 parameters, a 32-bit secret key with the value '00000000000000000000000000000001'was used, along with a nonce value of '00000001ánd a mode operation of Ñone'. The pycryptodome library [74,75], it was defined the other requirements using standard values, and the ciphertext outputs were obtained with hexadecimal encoding. The calculation of encryption/decryption times (in milliseconds) and estimated error, they are shown in Table 1. Some of tgese values were rounded to match the results presented in [5, 7]. Except for ciphertext with the random noisy strategies because these processes of the encryption method are random. Hence, it always generates different and dynamical results. The columns TC and TD, they are the average times of the estimated procedure for encryption and decryption tasks, respectively (the standard deviation among parentheses is shown). Two tests of ciphertext results for each encryption strategy, they are also shown. Two experimental tests with the same plaintext: "#Welcome#?", in some cases, different results have been obtained. In contrast, the majority of the experiments yielded successful results. With the exception of some tests using the DES algorithm and its associated random noisy scheme, which they have reported errors. Characters beyond the ASCII range, including '#'(ordinal 9619) and '?'(ordinal 65533), introduced into the plaintext, it could be the cause.

Encryption Strategies	TC	TD	% Error	Test 1	Test 2
Salsa20	1.0030 (0.4828)	0.0882 (0.0058)	0 (0)	c35d5f4be4-3284e385d7-77345fdf61d243	c35d5f4be4-3284e385d7-77345fdf61d243
random noisy Salsa20	1.0232 (0.3491)	0.2717 (0.0311)	0 (0)	±N±ººH6{F{-4ÇÇ*ⓒDⓒ±±-¿¿Ps;s··Û-vÛW$W³{³q-<qÖrÖˆl5lÉÉ¦s¦Å-Åj5j*ÇcÇææ··h7hä-äl:l[T	Â_ÂzGzhÌh-Ì±|±Á[Á\ãã7Xo-<oo=o»»··åån;nk3k-]2P$\frac{11}{44}$ÂÂU-c/c¯z¯ÊdÊ-:àzà¬v¬S-£?£QQ_+_ÁÁ
DES	1.1439 (0.5087)	0.1937 (0.0187)	1 (0)	d535c337be-1d94db5b94-0b75bce124-dcda93ffa35791bf07	d535c337be-1d94db5b94-0b75bce124-dcda93ffa35791bf07z

Encryption Strategies	TC	TD	% Error	Test 1	Test 2
random noisy DES	1.6333 (0.4897)	0.5848 (0.0389)	1 (0)	¿[¿²}²c0c¹¹à}ài6iÈ-ÈÀÀíí²M²-l;lUM0ºXº-j5j£A£11/22gj:j=Ui4i¥-C¥(«F«ÁÁly-ññÐmÐÐlÐôôi0i-a.a°J°&7[-n9na*a»»O-O7÷÷\,\i2i	“ððÉÉ]Y¾[¾\)\ m S¨F¨ÆaÆg-6gññM¬x¬-óóðð§r§ÃÃ«w«]«I«-Wr=r¿]¿¯-L¯ôôV%V{-I{11/44¸T¸åå2=ººRRÑ-kÑìôôdr=rY”YÆÆ-|à˜àßyßp@p®w®”
AES-256	7.3116 (3.7093)	0.6166 (0.0169)	0 (0)	3030303030-3030303030-3030303030-3130319421-780fc40c59-f14796a598-115d8e5139-f6f9666f67-cb30fcda01-f22739eeda	3030303030-3030303030-3030303030-3130319421-780fc40c59-f14796a598-115d8e5139-f6f9666f67-cb30fcda01-f22739eeda

Encryption Strategies	TC	TD	% Error	Test 1	Test 2
random noisy AES-256	8.4637 (3.9351)	1.0021 (0.0119)	0 (0)	xExdj7jµµY-&Y¡q¡eÄÄ¼-¼W'Wk8ki9-iU¨x¨p=p¾-¾j7jZµµˆ.-ˆiWOQ!QV-#Vej7jR³³-{K{OUY§s§uCu«z« i ÍÍ®˜®ÍgÍ-ËhËs?svFv-÷÷{F{ÍÍ¯-I¯¸¸h£l£r-9rZ¿ˆ¿ÄÄ-³z³¤l¤Rˆ¯-z¯$ÌÌá\|á²}²¹¹²²}J}È-ÈÈbÈ°z°¤->¤¹¹⁴⁴h2hfq-;q¦@¦ËËS£-@£´R´ÀÀÄ-Äõõââ¥A¥-&[h7h#¸¸µ-µ²{²j7jn5nðð$¥D¥	"§t§P P[(-[xHxNuEuÇ-ÇnÁÁ´´Ox-Hx»»¹¹µµ-d¬y¬]ÁÁ_-/_n;nÀÀMS-#S'W'Wo<o-xHxªwª\&\V¯˜¯v=vÅ-Å¥s¥»»ÀÀ-p8pT²L²íí-£o£c3cµRµ-¹¹⁴⁴_&_Ä^Ä}-L}Z¶¶fTTí-í··G˜F˜V%-VN¢m¢÷÷¾-¾9u@uÄÄ]*-]\ÕoÕKÔn-ÔdQzDz¸¸-â|âX"Xh½Z½ÍkÍ¿¿h-*<4Á'ÁVk5U#U n [$[U"UXÉdÉõ-õÔpÔÀ_À"
Blowfish	7.5597 (3.7931)	0.6486 (0.0373)	0 (0)	4af747eaab-e473251f42-200cf8fda7f2	4af747eaab-e473251f42-200cf8fda7f2

Encryption Strategies	TC	TD	% Error	Test 1	Test 2
random noisy Blowfish	7.8544 (3.7619)	0.9457 (0.0349)	0 (0)	ÉÉïïóóc,c¦-r¦_3óóÖuÖ-ÇeÇÇbÇk¨q-¨hiY$Y_±K-±bY'Y$^{11}_{22}{}^{33}$-^{11}ÑnÑ¨B¨H-ĐjĐ»W»èè®-w®±K±¦t¦	YÓrÓ5'UcÁ-\ÁÃbÃ®M®-Á_Áéé['[-m6mi6ic1c£-n£$^{11}_{22}$µOµ©u-©^{33}])[±±b-×t×Â\Ât<t-.à|à¥D¥ÂÂ.]
3DES	7.6814 (3.9388)	0.5004 (0.0184)	0 (0)	d535c337be-1d94db5b94-0b75bce124-dc620c51e2380a3d5c	d535c337be-1d94db5b94-0b75bce124-dc620c51e2380a3d5c
random noisy 3DES	8.4594 (4.1892)	0.6742 (0.0179)	0 (0)	êêTTp=pµµ.-e'^{a}s^{a}ÈfÈÄ_Äk:k<§n§ÉÉµQµ-V*-y@yS[¤B¤À-ÀÂÂÖ°N°µRµ-,d3dd2dRRË-gËÂ_Â£m£Ã-ÃT¦C¦m8mh7h ; Z(Z m j2j p çç_-,_ÓoÓ¡l¡öö	8xCxxExˆ)ˆ-¤A¤dn;n[$-[çç7®}®&w->wxDxéé^{3}Q-^{3}µµ^{o}X^{o}¢i¢-^{3322o}X^{o}ÆÆ-$^{3\sim3}$+ÙvÙÓn-Ó£r£[)[])]&¡>¡iY'-Y´´ÊgÊ¡l¡-[óóYU"U°x°j:jìiXÂˆÂ$\frac{11}{44}$3
Average	5.2134 (3.2975)	0.5526 (0.2849)	0.2 (0.40)	- - - -	- - - -

While these characters might be unavoidable in real applications, in Table 1, it demonstrates that most encryption methods, they were able to hide this issue in the encrypted output. Reference [5], it notes that a common issue with random noisy alternatives is the inability to control the maximum random value selected from the ASCII table. Despite the success of these strategies, an ASCII value can be reinterpreted as a different encoding character, such as UTF-8. By employing mod 120, which yields sequences with allowed ASCII values, we avoided these problems in our study. Unless in particular instances where plaintext input was subject

to noisy injection, as discussed above. Compared to other standard symmetric proposals, the Salsa20-based encryption process, it was substantially faster, with speeds 1.1404 to 7.6577 times greater. This resulted in a millisecond difference of 0.1408 to 6.6783 (see Table 1). Despite the encryption/decryption times, they are much faster using Salsa20 algorithm, in this research, it was observed that this strategy supports plaintext or ciphertext with values greater than 255 characters. Thereby, it can be considered a secure schema if this situation is validated properly. Similarly, the 3DES alternative has not encountered any errors, but it is limited to supporting a maximum of 22 characters for both plaintext and ciphertext, similar to the Blowfish proposals with 13 maximum. In this research, DES has a character limit of 255 for plaintext or ciphertext. Given the average error rate of 1.0 % during data processing, the DES alternative, it is not considered a reliable option. Furthermore, Table 1 in this study presents several static ciphertext results obtained using standard encryption algorithms such as Salsa20, DES, 3DES, AES-256, and Blowfish. However, this does not necessarily imply that these schemes are vulnerable or insecure in all cases. Moreover, random Caesar applied to plaintext achieved the best balance. Notably, the random Caesar II with mod 120 significantly improved encryption times, outperforming the other strategies evaluated. The experiments yielded an average encryption time of 0.14 milliseconds with a standard deviation of 0.0108. In contrast, the decryption process had an average time of 0.05 milliseconds with a standard deviation of 0.0011. These results are not included in Table 1, as the focus of this research is on comparing standard encryption algorithms with their noisy injection applications. When applied to ciphertext, the random noisy Salsa20 strategy, which it combines standard Salsa20 with random Caesar II mod 120, outperformed the rest random noisy approaches here evaluated. The most balanced results are consistently achieved with Salsa20 strategy that it includes its random noisy strategy. Compared to the rest random noisy strategies, the novel random noisy Salsa20 alternative has showed a notable performance improvement, with speeds 1.5962 to 8.2712 times faster. The difference in milliseconds ranged from 0.6101 to 7.4404. The results show that the encryption speed difference between Salsa20 and its random noisy version, it was not significant. For instance, traditional Salsa20 is 1.0201 times faster, with a marginal difference of 0.0201 milliseconds (see Table 1). In this study, both Salsa20 alternatives, they were found to support a maximum plaintext length of 255 characters, as previously noted. Notably, the experiments did not reveal any errors related to this limitation. At any rate, this measure is deemed to provide a substantial improvement in the confidence of the encryption strategy's performance. The random noisy schemes demonstrate potential as an experimental tool, yet further investigation into other aspects is necessary, particularly since the experiments, they were restricted to plaintext inputs of up to 255 characters. The cases of 3DES, and Blowfish are exceptions, as previously discussed. Each encryption strategy was evaluated based on its own training sample, which was designed independently. Random Caesar schemes have proven effective in enhancing data security [4, 5, 9, 11], and the random noisy strategies, they have demonstrated similar efficacy. Table 1 confirms this effect in the global average calculation. Notably, random noisy alternatives result in slightly longer ciphertexts. The experiments revealed that DES-generated ciphertext can be quicker than 3DES, AES-256, and Blowfish. Nonetheless,

the use of DES may pose security risks due to its potential vulnerability to decryption, as evidenced by the 1.0 % average error rate observed during encryption and decryption. The use of random noisy strategies with standard encryption algorithms produced dynamic ciphertext results in every instance. Nonetheless, the time required for FinalPackage ciphertext generation exceeded that of the standard algorithms, as demonstrated in Table 1's Test1 and Test2. The execution times for Salsa20 proposals, they are shorter compared to the DES, 3DES, AES-256, and Blowfish strategies here evaluated. Notably, these random noisy schemes consistently produce dynamic ciphertext outputs. Cybercriminals would encounter significant obstacles in decrypting data, as they would need to determine each random Ki shifting value in advance, which has been previously hidden. When utilizing the PartialNoisy proposal, a novel partial noisy injection scheme introduced in [5], the decryption process becomes notably more complex. The complexity of these data discovery tasks could make them difficult to accomplish, even with quantum computing. The exploration of PartialNoisy was beyond the scope of this study and is suggested for future work. Repeatedly using these random noisy strategies, they can lead to better and more dynamic results in comparison to traditional static encryption methods. This noisy injection alternative, it can increase the security degree of the ciphertext or plaintext. Besides, this situation might warn us against future quantum computing attacks [8, 22, 23, 42], improving the digital data security of the organizations, as mentioned above. Additionally, the resilience of these random noisy alternatives to various cyberattacks remains unevaluated, leaving potential vulnerabilities unknown. Comparing to other studies [4, 5], our research suggests that downsized ciphertext using reduced random or mutation methodologies [4], which they can be effective indicators. These schemes provide a balance between efficiency and security, yielding short ciphertexts and fast encryption processes, while the Ki shifting of partial ciphertext ensures the protection of data security. These alternatives can mitigate the risk of digital data theft by incorporating a significant amount of noise into the ciphertext. In other words, proposals based on reduced random and mutation schemes show promise. However, this study focuses on other approaches, leaving reduced and mutation alternatives for future works. Notwithstanding the difficulties, the study's goals and hypotheses, they were fulfilled successfully. Due to that the random noisy Salsa20, it has been presented here as a novel proposal based on noisy injection because it allows obtaining dynamic encryption ciphertext outputs. In other words, it can generate varying outcomes, even with identical plaintext input and parameters, thereby confusing potential cybercriminals. This data is supported by the information presented in Table 1. By comparing standard encryption algorithms with random noisy strategies, we can infer that noisy injection is a promising approach for organizations seeking a safe alternative. This approach incorporates the novel random noisy Salsa20 strategy, especially in environments where traditional Salsa20 methods, they are already in use. Implementing the random noisy scheme is recommended to strengthen digital data security in these cryptosystems. This novel alternative based on noisy injection is considered a safe measure for organizations, as it opens up a wide range of opportunities for improving digital data security. Furthermore, a modification to the dynamic encryption methodology presented here could be worth exploring. By utilizing reduced noisy strategies [5], and reduced random mutation schemes [4], this

approach enables a significant reduction in ciphertext size, up to 33 %, in FinalPackage compared to random noisy proposals. Another measure that it might be explored, it is the application of different ways for achieving the noisy injection. Especially, those that merge the simultaneous random noisy methodology with artificial intelligence utilizing the nearest neighbor rule [8,10,11,46,47,65,67], and pseudo-hexadecimal encoding [8–10], as noted above. This implies a vast array of possibilities that could be explored in future research.

Conclusions

Regular updates to cybersecurity strategies, including encryption methods, they are vital for maintaining digital data safety in organizations. However, it does not guarantee their digital data security. Studies cited above [4, 8], they have revealed that the challenges posed by inadequate encryption measures. If a method is well-known, it will become in inadequate strategy. Previous studies [4–11], they have proposed various dynamic encryption alternatives to address the issue of digital data theft. We present a novel approach that fuses standard encryption techniques with a random Caesar strategy, tailored for real-world use in organizations. A new dynamical encryption proposal, known as the random noisy Salsa20 strategy, is presented in this paper. Furthermore, the evaluation included a comparison of five dynamic encryption alternatives that utilize random noisy strategies. These strategies leverage artificial intelligence for noisy injection into ciphertext, utilizing random and heuristic methods as previously discussed. Therefore, it has the potential for practical application in organizational contexts. The methodology based on noisy injection makes a valuable contribution to overcoming the limitations of standard encryption strategies, which in turn boosts its effectiveness. Experimental results with the dynamic encryption random noisy alternatives, they have revealed that can cope with the cyberattacks and data security problems with high levels of trust. Hence, despite of the occurred difficulties, the research's hypotheses and objectives, they were fulfilled successfully. Due to that the random noisy Salsa20, it has been presented here as a novel proposal based on noisy injection because it allows obtaining dynamic encryption ciphertext outputs. Besides, in all cases, these random noisy strategies, they allow the dynamic generalized results even better than those obtained with the standard encryption algorithms (see Table 1). We plan to conduct further investigation into this matter. One of the methods we'll be exploring is the application of measures to decrease the size of ciphertext resulting from random noisy strategies. The use of reduced random schemes [5], or reduced mutation strategies [4], it might be an option, since they enable the masking of ciphertext operations. Another potential approach to consider is the use of various methods for achieving noisy injection. In particular, the methods that synergize random noisy approaches with nearest neighbor rule-based AI and pseudo-hexadecimal encoding, as noted earlier. As expected, this involves a multitude of possibilities that we can cover in future works.

Acknowledgment

This work was supported by Tecnológico Nacional de México (I.T. Ciudad Altamirano). This research is future work of a last project identified by: 19329.24-P.

Contribución de Autoría

Edgar Rangel Lugo: Administración de proyectos, Investigación, Metodología, Software, Validación (random noisy strategies), Redacción - borrador original. Kevin Uriel Rangel Ríos: Conceptualización, Investigación, Software, Validación (several experiments, random noisy strategies), Escritura, revisión y edición (reviewer, editing, and translation). Carlos Alberto Bernal Beltrán: Investigación, Validación (standard encryption algorithms). Leonel González Vidales: Investigación, Validación (standard encryption algorithms). César Del Ángel Rodríguez Torres: Investigación. Lucero De Jesús Ascencio Antúnez: Validación (standard encryption algorithms). Rosa Isabel Reynoso Andrés: Validación (standard encryption algorithms).

Referencias

[1] B. Delman, "Genetic algorithms in cryptography," Master's thesis, Rochester Institute of Technology, 2004, rIT Scholar Works.

[2] S. Kalsi, H. Kaur, and V. Chang, "Dna cryptography and deep learning using genetic algorithm with nw algorithm for key generation," *Journal of Medical Systems*, vol. 42, no. 17, 2018.

[3] J. C. Mendoza, "Demostración de cifrado simetrico y asimétrico," *Ingenius, Revista de Ciencia y Tecnología*, no. 3, pp. 46–53, 2008.

[4] E. Rangel and K. U. Rangel, "Novel random encryption methods based on mutation strategies of artificial intelligence," *SPCSJ: Scientific and Practical Cyber Security Journal*, vol. 8, no. 3, pp. 84–91, 2024.

[5] E. Rangel, K. U. Rangel, and L. González, "Dynamic encryption methods based on noisy injection and camouflaging ciphertext strategies with artificial intelligence," *SPCSJ, Scientific and Practical Cyber Security Journal*, vol. 9, no. 1, pp. 82–104, 2025.

[6] ——, "Inyección de ruido para encriptado de datos dinámico con inteligencia artificial. caso de estudio: Algoritmo gost r 34.12-2015," *Revista Electrónica de Divulgación de la Investigación del SABES*, vol. 29, 2025.

[7] E. Rangel and K. U. Rangel, "Mejorando la seguridad del algoritmo camellia, mediante la inyección de ruido sobre textos cifrados utilizando procesos basados en inteligencia artificial," *INTELETICA, Revista de Inteligencia Artificial, Ética y Sociedad*, vol. 2, no. 3, 2025.

[8] E. Rangel, K. U. Rangel, L. González, A. Ortiz, and C. A. Rodríguez, "Four dynamic encryption alternatives with artificial intelligence based on pseudo-hexadecimal noisy injection schema for handling the theft of digital data problem," *SPCSJ, Scientific and Practical Cyber Security Journal*, vol. 9, no. 3, pp. 59–77, 2025.

[9] E. Rangel, K. U. Rangel, J. Medrano, C. A. Bernal, and L. González, "Algoritmo genético para cifrado de datos, basado en un nuevo concepto pseudo-hexadecimal con inteligencia artificial," in *VI Congreso Nacional De Investigación En Ciencia E Innovación De Tecnologías Productivas*, Cd. Altamirano, Guerrero, México, 2023.

[10] E. Rangel, K. U. Rangel, and L. González, "Cifrado de datos dinámico con inteligencia artificial, utilizando el nuevo formato pseudo-hexadecimal," *Revista Electrónica de Divulgación de la Investigación del SABES*, vol. 28, 2024.

[11] E. Rangel and K. U. Rangel, "La regla del vecino más cercano como alternativa para inyectar ruido a mensajes encriptados por el algoritmo: Noised random hexadecimal," *INTELETICA, Revista de Inteligencia Artificial, Ética y Sociedad*, vol. 1, no. 2, pp. 1–15, 2024.

[12] D. Álvarez, "Algunos aspectos jurídicos del cifrado de comunicaciones," *Derecho PUCP*, no. 83, pp. 241–264, 2019.

[13] F. Barranco and C. Galindo, *Criptografía básica y algunas aplicaciones*. Universidad Jaume I, 2022.

[14] S. Gómez, J. D. Arias, and D. Agudelo, "Cripto-análisis sobre métodos clásicos de cifrado," *Scientia Et Technica*, vol. 2, no. 50, pp. 97–102, 2012.

[15] B. Javidi and J. L. Horner, "Optical pattern recognition for validation and security verification," *Optical Engineering*, vol. 33, no. 6, pp. 1752–1756, 1994.

[16] B. Reddaiah, "A study on genetic algorithms for cryptography," *International Journal of Computer Applications*, vol. 177, no. 28, pp. 1–4, 2019.

[17] C. Sebas, "¿qué son los algoritmos genéticos en las inteligencias artificiales?" 2023.

[18] S. Paul, P. Dasgupta, P. K. Naskar, and A. Chaudhuri, "Secured image encryption scheme based on dna encoding and chaotic map," *Review Of Computer Engineering Studies*, vol. 4, no. 2, pp. 70–75, 2017.

[19] R. Oppliger, *Contemporary cryptography*. Artech House Computer Security Library, 2005.

[20] D. R. Stinson and M. B. Paterson, *Cryptography: Theory and Practice*, 4th ed. Chapman and Hall/CRC, 2019.

[21] H. C. A. Van-Tilborg, *Encyclopedia Of Cryptography And Security*. Springer, 2005.

[22] L. Baklaga, "Leading the way in quantum-resistant cryptography for everyday safety," *SPCSJ, Scientific and Practical Cyber Security Journal*, vol. 8, no. 3, pp. 65–73, 2024.

[23] R. Bavdekar, C. Eashan-Jayant, A. Ankit, and K. Tiwari, "Post quantum cryptography: A review of techniques, challenges, and standardizations," in *International Conference on Information Networking (ICOIN)*, 2023.

[24] Q. H. Dang and H. Q. Le, "Improved cryptanalysis of the rsa algorithm using side-channel attacks," *Journal of Information Security and Applications*, vol. 65, 2022.

[25] D. Luciano and G. Prichett, "Cryptology: From caesar ciphers to public-key cryptosystems," *The College Mathematics Journal*, vol. 18, pp. 2–17, 1987.

[26] M. S. Rahman and M. S. Hossain, "A secure private key cryptography scheme using rsa and aes," *Journal of Cybersecurity*, vol. 1, no. 1, pp. 1–9, 2021.

[27] J. Rodríguez, *Operadores Genéticos Aplicados A La Criptografía Simétrica*. Universidad Distrital Francisco José De Caldas, 2020.

[28] M. Baker and J. Schiller, "Ecies: Elliptic curve integrated encryption scheme," in *Cryptography and Network Security*. Springer, 2015, pp. 245–263.

[29] D. Hankerson, J. L. Hernandez, and A. J. Menezes, "Software implementation of elliptic curve cryptography over binary fields," in *CHES 2000, LNCS 1965*. Springer-Verlag, 2004, pp. 1–24.

[30] P. L. Montgomery, "Speeding up the pollard rho method," *Mathematics of Computation*, vol. 48, no. 177, pp. 453–456, 1987.

[31] NIST, "Recommended methods for key establishment using public key cryptography," NIST Special Publication 800-56A, Tech. Rep., 2013.

[32] H. W. Dhany, F. Izhari, H. Fahmi, M. Tulus, and M. Sutarman, "Encryption and decryption using password based encryption, md5, and des," *Atlantis Press*, 2018.

[33] A. Kumar and S. Sharma, "A study on historical cryptographic techniques: Caesar cipher to des," *International Journal of Advanced Science and Technology*, vol. 30, no. 2, pp. 555–564, 2021.

[34] H. C. Van and S. Jajodia, *Encyclopedia Of Cryptography And Security*. Springer Science & Business Media, 2011.

[35] B. Schneier, "Description of a new variable-length key, 64-bit block cipher (blowfish)," in *Fast Software Encryption*, 1994.

[36] ——, *Secrets and lies: Digital security in a networked world*. Wiley, 2000.

[37] E. A. AL-Maqtari, "Performance evaluation for aes, blowfish, des, and 3des cryptography algorithms," *PUIRP: Partners Universal Innovative Research Publication*, vol. 2, no. 5, pp. 86–95, 2024.

[38] R. K. Muhammed, R. R. Aziz, A. A. Hassan, A. M. Aladdin, S. J. Saydah, T. A. Rashid, and B. A. Hassan, "Comparative analysis of aes, blowfish, twofish, salsa20, and chacha20 for image encryption," *Kurdistan Journal of Applied Research*, vol. 9, no. 1, pp. 52–65, 2024.

[39] H. K. Garai and S. Dey, "A multi-step key recovery attack on reduced round salsa and chacha," *Cryptologia*, vol. 49, no. 3, pp. 252–267, 2024.

[40] A. Saini, A. Tsokanos, and R. Kirner, "Cryptoqnrg: a new framework for evaluation of cryptographic strength," *The Journal of Supercomputing*, vol. 79, pp. 12 219–12 237, 2023.

[41] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[42] M. Iavich, T. Kuchukhidze, and A. Gagnidze, "Post-quantum digital signature using verkle trees and lattices," *SPCSJ, Scientific and Practical Cyber Security Journal*, vol. 8, no. 3, pp. 35–52, 2024.

[43] P. Fuegner, "Are rsa and aes both at risk from the quantum threat?" 2024.

[44] S. Sengupta and S. Ghosh, "Quantum computing encryption threats: Why rsa and aes are at risk," *Journal of Cryptographic Research*, 2023.

[45] J. Thakur and N. Kumar, "Des, aes and blowfish: Symmetric key cryptography algorithms simulation based performance analysis," *International Journal of Emerging Technology and Advanced Engineering*, pp. 6–12, 2011.

[46] E. Rangel, "Vecinos envolventes para variantes de la regla del vecino más cercano," Master's thesis, Instituto Tecnológico de Toluca, 2002.

[47] ——, "La regla de los k vecinos más cercanos (k-nn) basada en distancia de manhattan (city-block) para mejorar la clasificación de patrones," in *V Congreso Nacional De Investigación En Ciencia E Innovación De Tecnologías Productivas*, 2022.

[48] S. S. Iyengar, R. Kannan, and S. Ganapathi, "Inteligencia artificial y criptografía: Tendencias y desafíos," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 2, pp. 833–844, 2021.

[49] X. Liu and X. Wang, "Quantum cryptanalysis of lattice-based cryptographic protocols," *Physical Review X*, vol. 12, no. 2, 2022.

[50] A. K. Singh, P. Kumar, and R. Singh, "Aplicación de la inteligencia artificial en la criptografía: Una revisión," *Journal of Intelligent Information Systems*, vol. 67, no. 2, pp. 257–275, 2021.

[51] T. M. Mitchell, *Machine learning*, 2nd ed. McGraw-Hill, 2020.

[52] J. Ross-Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[53] S. J. Russell and P. Norvig, *Inteligencia artificial: Un enfoque moderno*, 4th ed. Pearson, 2020.

[54] A. K. Hartmann, "Heuristic search in graphs," *Journal of Artificial Intelligence Research*, vol. 68, pp. 1–33, 2020.

[55] J. S. Sánchez, F. Pla, and F. J. Ferri, "Prototype selection for the nearest neighbor rule through proximity graphs," *Pattern Recognition Letters*, vol. 18, pp. 507–513, 1997.

[56] L. I. Kuncheva and L. C. Jain, "Nearest neighbor classifier: Simultaneous editing and feature selection," *Pattern Recognition Letters*, vol. 20, pp. 1149–1156, 1999.

[57] K. P. Murphy, *Probabilistic machine learning: An introduction*. MIT Press, 2022.

[58] B. Reddaiah, "A study on pairing functions for cryptography," *IJCA*, vol. 149, no. 10, pp. 4–7, 2016.

[59] D. B. Skalak, "Prototype and feature selection by sampling and random mutation hill climbing algorithms," in *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, 1994, pp. 293–301.

[60] A. Clark, "Modern optimisation algorithms for cryptanalysis," in *Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems*, 1994, pp. 258–262.

[61] W. Griindlingh and J. H. Van-Vuuren, "Using genetic algorithms to break a simple cryptographic cipher," 2002, submitted.

[62] R. A. J. Matthews, "The use of genetic algorithms in cryptanalysis," *Cryptologia*, vol. 17, no. 4, pp. 187–201, 1993.

[63] L. Bruzzone and S. B. Serpico, "Classification of imbalanced remote-sensing data by neural networks," *Pattern Recognition Letters*, 1997.

[64] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2021.

[65] R. Barandela, J. S. Sánchez, V. García, and E. Rangel, "Strategies for learning in class imbalance problems," *Pattern Recognition*, vol. 36, no. 3, pp. 849–851, 2003.

[66] D. Lewis and J. Catlett, "Heterogeneous uncertainty sampling for supervised learning," in *Proceedings of the 11th International Conference on Machine Learning*, 1994, pp. 148–156.

[67] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1967.

[68] E. Rangel and K. U. Rangel, "Novel pseudo-hexadecimal encryption strategies for camouflaging ciphertext based on nearest neighbor with artificial intelligence," *IJCOPI*, 2024.

[69] Microsoft, "Descarga de software," 2025.

[70] Python.org, "The python network," 2024.

[71] Google, "Android 12, sistema operativo para dispositivos móviles," 2024.

[72] Pydroid3, "Pydroid3 versión 7.4_arm64," 2025.

[73] Python, "Cryptography 45.0.4," 2025.

[74] PyCryptodome, "Crypto.cipher package," 2025.

[75] PyPI, "Pycryptodome 3.21.0," 2024.