Intelética 2(4) (2025), 75-101
Revista de Inteligencia Artificial, Ética y Sociedad

INTELETICA

https://inteletica.iberamia.org/

Mejorando la seguridad del algoritmo Camellia, mediante la inyección de ruido sobre textos cifrados utilizando procesos basados en inteligencia artificial

Edgar Rangel-Lugo [1], Kevin Uriel Rangel-Ríos [2]

[1] Tecnológico Nacional de México. Instituto Tecnológico de Ciudad Altamirano.
[2] Tecnológico Nacional de México. Instituto Tecnológico de Ciudad Altamirano.

[1] erangel_lugo@hotmail.com
[2] kgvppro@gmail.com

Abstract. The theft of digital data problem is receiving growing attention. This situation can be caused when a cybersecurity strategy, it has been employed (e.g., inadequate encryption methods). It has been observed in practical domains that it can produce an important losses in the organisation's finances. In this work, several aspects related with the subject, they are here studied. We have focused in the replacement of static encryption method by dynamic encryption alternatives based on *noisy injection* with artificial intelligence (AI). It has been applied over ciphertext that it was obtained with static encryption algorithms becoming it into dynamic encryption schema. The advantage of a *dynamic encryption method*, it is recommended due to that it can obtain different ciphertext results with the same plaintext input. Several alternatives for *noisy injection* based on *random noisy* strategies with AI, they are here suggested. Novel modification of *Camellia* algorithm based on *random noisy* strategies, it is here experimented and comparison results with traditional encryption strategies (*AES*, *GOST, 3DES, Blowfish*, and *DES*), they are also introduced.

Resumen. El problema de robo digital de datos está recibiendo gran atención. Esta situación, puede presentarse, debido al uso de una estrategia de ciberseguridad inadecuada (por ejemplo, utilizar algoritmos de cifrado estáticos). Se ha observado en dominios prácticos, que ello puede producir importantes pérdidas en las finanzas de las organizaciones. En la presente investigación, son estudiados varios aspectos relacionados con dicha área. Por lo tanto, este trabajo ha sido enfocado en recomendar alternativas para reemplazar el método de encriptado estático, por otras variantes dinámicas basadas en *inyección de ruido* con inteligencia artificial (IA), siendo aplicado sobre textos cifrados obtenidos por algoritmos estándar (estáticos), para ser convertidos en esquemas dinámicos. Un *método de cifrado dinámico,* permite generar diferentes resultados cifrados, aún utilizando la misma entrada de texto plano. Por último, en este trabajo, son recomendadas algunas alternativas para *inyección de ruido* con IA, basadas en estrategias del tipo: *random noisy* (*ruidosas aleatorias*). También, se introduce una nueva modificación del algoritmo: *Camellia* basado en metodología: *random noisy*. Finalmente, se comparan resultados con cinco alternativas *"ruidosas aleatorias"*, derivadas de algoritmos de encriptado estándar (*AES, GOST, 3DES, Blowfish* y *DES*).

Palabras clave: Metodología Random Caesar, criptografía, cifrado con inyección de ruido.
Keywords: Random Caesar methodology, cryptography, ciphertext based on noisy injection strategies.

ISSN: 3020-7444

1 Introducción

El problema de robo digital de datos está recibiendo gran atención en las organizaciones [1]. En dominios prácticos, no solamente se han observado pérdidas económicas en el campo empresarial, sino también, ha afectado su reputación [2]-[4], cuando se presentan situaciones de cibercrimen [2], [5]-[6], que traen como consecuencia la pérdida de información o robo digital de datos [2]. Este tipo de problemas, se ha detectado cuando una estrategia (de ciberseguridad) inadecuada ha sido empleada en las organizaciones, un caso muy particular, es el uso de métodos de cifrado estáticos [7]-[12].

Se entiende por cifrado de datos, la ocultación de la información, mediante la traducción de un mensaje original convirtiéndolo en un tipo de lenguaje o código, utilizando un alfabeto para cifrado/descifrado, que solamente podrá ser capaz de entender el software especializado o persona autorizada [7]. Al proceso inverso del cifrado, que transforma un código, secuencia o texto cifrado, se le conoce como desencriptado o descrifrado de datos [8], [12]. En este contexto, existen un gran número de alternativas. Siguiendo la común práctica, suelen ser empleados los algoritmos de cifrado simétricos [8]-[18], y también, existen los algoritmos de encriptado asimétricos [9], [11], [13]-[19]. El primer caso, consiste en el uso de una misma clave o llave secreta, que será utilizada para llevar a cabo el cifrado y descrifrado de datos. Se utiliza como parte de la entrada, junto con el texto plano. Por tal razón, se les conoce como algoritmos simétricos, debido a que, realizan el cifrado de llave simétrica [19]. Por otra parte, para realizar el cifrado/descrifrado con algoritmos asimétricos, se necesitan dos llaves o claves diferentes: una privada y otra pública [19]. A este tipo de estrategias, se les conoce como: algoritmos asimétricos porque realizan la encriptación mediante llave asimétrica. Para realizar este proceso, también se puede llegar a utilizar una clave secreta, que sirve para cifrar/descifrar la clave privada, otorgando mayor seguridad en los datos digitales. Ejemplos de estos algoritmos asimétricos, los más populares son: RSA (Rivest-Shamir-Adleman) [2], [9], [13]-[15], [17]-[19], ECC (Elliptic Curve Cryptography) [2], [14]-[15], [17]-[19], ElGamal [9], [14]-[15], [17], entre otros. Cabe aclarar que, en la presente investigación, no fueron utilizados ningún tipo de algoritmos asimétricos, es por ello, que no se aborda en este estudio dicho tipo de estrategias. Por lo tanto, solo nos hemos enfocado a señalar breves características de algunos algoritmos simétricos estándar, los cuales, fueron evaluados en el presente trabajo, aplicados en el cifrado de datos, durante la fase de experimentación. Lo anterior, con el propósito de comparar resultados con otras investigaciones [1]-[2], [18]-[19], así como, con la nueva propuesta aquí presentada, basada en el uso del algoritmo: *Camellia* , que se propone utilizar con inyección de ruido [1]-[2].

En la literatura, podemos encontrar que algunos algoritmos simétricos estándar, muy populares son: DES (Data Encryption Standard) [9], [13], [20]-]24], 3DES (Triple Data Encryption Standard) [13], [16]-[17], [20]-[21], Blowfish [25]-[26], AES-256 (Advanced Encryption Standard 256 bits) [9], [17], [20]-[21], [24], [27]-[29], GOST (GOsudarstvennyy STandart) estándar R 34.12-2015 versión Kuznyechik (Кузнечик) [30]-[31], Camellia [14], [17], [32]-[33], entre otros. Aunque los dos primeros algoritmos mencionados, ya no son recomendados, porque se han encontrado vulnerabilidades a determinados tipos de ciberataques, por ejemplo: el ataque de fuerza bruta [34]-[35], la técnica del diccionario [7], [34]-[36], por mencionar algunos casos. Una descripción rápida, sobre estos algoritmos simétricos populares, la podemos encontrar en [2], [19]-[21], donde se refiere lo siguiente: Baklaga [20], señala que "AES es un algoritmo simétrico de cifrado por bloques que opera con distintos tamaños y soporta diversas longitudes de clave (con 128, 192 y hasta 256 bits). En contraste, el algoritmo DES, procesa texto plano a 64 bits, produciendo también, 64 bits de texto cifrado. Esta operación involucra una serie de rondas basadas en sustitución y permutación que incluyen operaciones para descifrados en inversa o reversa. Sin embargo, estos 64 bits son considerados insuficientes para ambientes seguros, porque el esquema de seguridad, lo hace relativamente fácil de romper. Como resultado, fue desarrollado el algoritmo 3DES, como una versión mejorada del DES. Otro algoritmo simétrico popular es conocido como: Blowfish, el cual, utiliza bloques de cifrado de longitud variable y claves con rango de longitud entre 32 y 448 bits" [20]-[21]. Del mismo modo, el algoritmo simétrico: GOST [37]-[38], estándar R 34.12-2015 [30]-[31], versión Kuznyechik (Кузнечик) [31], realiza cifrado por bloques con tamaño de 128 bits, a través de un

número invariable de rondas (comúnmente usa de 10 a 32 rondas), que permite manejar claves con longitud variable hasta de 256 bits, permitiendo de este modo, el uso de configuraciones más flexibles.

En lo concerniente con el algoritmo simétrico: Camellia [14], [32]-[33], trabaja por bloques de 128 bits y permite manejar tamaños de longitud de clave con: 128, 192 y 256 bits, empleando un número variable de rondas, de acuerdo con la longitud de la clave, es decir, se realizan 18 rondas con claves de 128 bits, mientras que, para claves de 192 y 256 bits, se ejecutan 24 rondas. Para conseguir el cifrado, combina operaciones matemáticas, basadas en permutación y sustitución, empleando operador XOR [9], [17], haciendo uso de la llave secreta y el texto plano de entrada. El propósito del diseño de este algoritmo, fue adoptar un esquema altamente eficiente y seguro. Es por ello, que varios organismos internacionales lo recomiendan como un estándar de encriptado seguro. Empero, si observamos en la literatura, podemos encontrar en [17], que refiere a Camellia, como un cifrador por bloques, que encripta datos de 128 bits y acepta también 192 y 256 bits en llave secreta. Además, cuenta con 18 o 24 rondas dependiendo de la longitud de clave.

Recientes investigaciones [2], [7], [18], han revelado que los resultados generados por los algoritmos asimétricos estándar, producen cifrados dinámicos [2]. Mientras que, la mayoría de los algoritmos estándar para encriptado simétrico, regresan resultados estáticos. Aunque también, se ha hecho incapié, en qué esa situación no implica que sean vulnerables [2], [18]-[19], en todos los dominios prácticos. Sin embargo, algunos autores [7]-[8], [19], han considerado que el uso de un algoritmo estándar estáticos, puede poner en riesgo la seguridad de los datos digitales en las organizaciones. Debido a que, en este esquema estático, para una misma entrada de texto plano, es generado un mismo resultado cifrado, cuando se utiliza un mismo algoritmo con los mismos parámetros [8], [19]. Esta situación, resulta vulnerable a distintos tipos de estrategias de los ciber-delincuentes, por ejemplo, se pueden romper criptosistemas [9]-[15], [39]-[40], haciendo uso de la fuerza bruta [34], la técnica del diccionario [35], e incluso, estar amenazada la seguridad de la información ante posibles ataques de algoritmos basados en computación cuántica [2], [18]-[21], [42], [44]-[45], por ejemplo, mediante el uso de: Shor [21], o utilizando: Kyber512 [2], [42].

Empero, los algoritmos estándar para el cifrado dinámico, a pesar que son capaces de generar diferentes resultados cifrados, en cada ejecución del procedimiento, aún cuando sea utilizado el mismo algoritmo con los mismos parámetros y la misma secuencia de texto plano [1]-[2], [7]-[8]. Este tipo de esquema, basado en algoritmos de cifrado estándar, los cuales, son bien conocidos por los ciber-delincuentes, y que además, algunos lenguajes de programación, por ejemplo: Python [43[, ya cuentan con paquetes o librerías, que permiten el cifrado y descifrado de la mayoría de estos algoritmos estándar [8], [36], tal es el caso de: eciespy [46], gostcrypto [47], PyCryptodome [48], del paquete paquete Crypto.Cipher, así como, PyPI Pycryptodome [49], por mencionar algunas de ellas. En este contexto, aunque en algunos casos, estos algoritmos de encriptado permiten arrojar resultados dinámicos, ello no garantiza al 100%, la seguridad de los datos digitales en las organizaciones. Además, en algunas investigaciones [2], [18]-[21], [42], [44]-[45], se advierte que, varios de estos criptosistemas basados en algoritmos estándar, como es el caso de: AES, RSA y ECC, pueden resultar vulnerables o estar amenazados por la llegada de la computación cuántica. A pesar que en la presente investigación no se ha utilizado la computación cuántica, otras medidas basadas en la inyección de ruido [1]-[2], para confundir a los ciber-delincuentes, han sido propuestas.

En este respecto, es recomendable emplear otras medidas alternativas o complementarias, o combinar con el uso de otras estrategias mediante la fusión de técnicas [2], [19], ya que, los algoritmos simétricos y asimétricos, no son las únicas alternativas para encriptación de información. También existen otras estrategias de encriptado dinámico, las cuales, persiguen propósitos muy específicos. Por ejemplo, se encuentran las estrategias de cifrado de datos utilizando tecnología óptica [50]-[56], el cifrado basado en sistemas discretos caóticos [53], [57]-[71], el uso de criptosistemas basados en inteligencia artificial (IA) [1]-[2], [7]-[8], [20], [36], [72]-[78], por mencionar algunos casos.

En este contexto, Rangel & Rangel [7], señalan que: "Uno de los propósito de la inteligencia artificial (IA), es hacer que "la máquina piense" [7]-[8], [18]-[19], [76]-[77]. Para lograr dicho propósito, suelen ser empleadas una gran variedad de modelos, técnicas, estrategias, metodologías y/o paradigmas, los cuales, fundamentan sus bases, con métodos estadísticos [79], métodos heurísticos [2], [8], [18]. Ejemplos de estos casos son: los árboles de decisión [80], los gráfos [81], entre otros modelos.

También, existen los métodos aleatorios [2], [8], [18]-[19], [82]-[83], un caso muy particular son los algoritmos genéticos (AG) [1], [7]-[8], [24], [36], [39], [72], [75], [78], [82]-[85]. Algunas investigaciones [2], [7]-[8], [18], aseguran que una estrategia para amortiguar un poco el problema del robo digital de datos, consiste en la inyección de ruido, ya sea, durante el proceso de encriptación de un texto plano [1], [7]-[8], [18], o incluso, siendo aplicado directamente al texto encriptado por un algoritmo estándar [2], [19], sin que ello sea "mal interpretado" como un doble cifrado de datos [2].

Se entiende por inyección de ruido, cuando un texto plano o texto cifrado contiene algunos caracteres adicionales en formato UTF-8 (Unicode Transformation Format 8 bits) [86], o código ASCII (American Standard Code for Information Interchange) [87], los cuales, no son parte del mensaje cifrado o texto de entrada original [1]-[2]. Este tipo de estrategias, puede llegar a confundir a los ciber-delincuentes.

Algunas investigaciones [1]-[2], [7]-[8], [18]-[19], [36], relacionadas con inyección de ruido como medida de cifrado de datos dinámico, fueron aplicadas sobre texto plano [1], [7]-[8], [18], [36]. Una línea de investigación, consiste en la aplicación del formato pseudo-hexadecimal [7], [18], [36], para inyectar ruido usando un paquete de encriptado, mediante el uso de inteligencia artificial (IA), empleando métodos heurísticos aleatorios [8], así como, algoritmos genéticos (AG) [36]. Recientes estudios en este mismo contexto, fueron realizados en algunas investigaciones [8], [18], [36], las cuales, presentan varias alternativas para la implementación del denominado pseudo-hexadecimal. El primero de ellos [36], sugiere la implementación de AG para inyectar ruido en formato pseudo-hexadecimal, con el propósito de confundir a los ciber-delincuentes al momento de querer descifrar el mensaje encriptado, por una metodología denominada: "Noised" random pseudo-hexadecimal GAs [36]. A dicha metodología, posteriormente se descubrió deficiencias, ya que, producía demasiados errores, y además, este algoritmo genético, en ocasiones entraba en un ciclo infinito, y no podía terminar el proceso de encriptado. Por tal razón, surge una nueva adaptación o modificación al procedimiento, sustituyendo el AG por un método heurístico, que se considera hace uso de inteligencia artificial, porque utiliza métodos aleatorios, procedimientos muy similares al proceso de selección aleatoria de un algoritmo genético. A esta nueva propuesta se le ha denominado como: "Noised" random pseudo-hexadecimal [8]. Esta alternativa consiste en el uso de un alfabeto inicial que es definido por el usuario programador, o bien, por el cripto-analista [7], [36], el cual, haciendo uso de un método heurístico, con reglas previamente definidas, permite generar los alfabetos de cifrado/descrifrado, siendo seleccionados los caracteres ASCII, del alfabeto inicial, de manera aleatoria sin reemplazo, para posteriormente ser guardados en el alfabeto de cifrado en formato pseudo-hexadecimal, mientras que, el alfabeto para descifrado guarda caracteres ASCII en formato hexadecimal. Al final, se construye un paquete que guarda intercalados los valores de cada carácter guardado en los diccionarios de cifrado y descrifrado, así como, una secuencia de caracteres que incluye relleno hexadecimal, la cual, fue resultado de una previa operación denominada: cifrado parcial [8]. Por último, otro trabajo reciente [18], concerniente con la inyección de ruido pseudo-hexadecimal, presenta algunas adaptaciones al procedimiento, así como, cuatro alternativas de implementación, denominadas en dicha investigación como: noisy random pseudo-hexadecimal I (versión estándar), noisy random pseudo-hexadecimal por sustitución (by substitution), noisy random pseudo-hexadecimal por desplazamiento (by shifting) y noisy random pseudo-hexadecimal II (versión 2) [18]. En la presente investigación, no se hace uso de este tipo de estrategias basadas en formato pseudo-hexadecimal, es por ello, que no se describen más detalles de su procedimiento.

Por otra parte, existen varios estudios [1]-[2], [7], [19], [36], los cuales, no hacen uso del formato pseudo-hexadecimal, pero aunque siguen diferentes caminos de implementación, también promueven el tema de inyección de ruido [1], [7], [19], así como, otro nuevo concepto denominado: camuflaje de textos [1], [2]. Estas estrategias han sido aplicadas a texto plano [1], [7], y también, a textos cifrados [2], [19], alternativas que llevan como propósito confundir a los ciber-delincuentes, y por ende, mejorar la seguridad de los datos digitales en las organizaciones. En este mismo contexto, una propuesta para inyección de ruido, es aplicar una modificación al algoritmo de César [9], [12], haciendo uso de IA, empleando un método heurístico que incluye un proceso de selección aleatoria con reemplazo, muy similar a lo utilizado en la fase de selección de un algoritmo genético. A esta nueva metodología se le ha denominado como: random Caesar [1], [7], [36], la cual, consiste en aplicar desplazamientos aleatorios a cada caracter del texto plano. Es decir, mientras que el algoritmo del César (obtenido por:

$C_i = S_i + K$ *módulo 26*), define un solo valor de desplazamiento K, que se aplica al texto plano S_i, estando limitado a *26* caracteres del alfabeto, por el uso de un *módulo=26*. En cambio, el método: random Caesar (obtenido por: $C_i = S_i + K_i$ *módulo N),* define un vector K_i, de desplazamientos, uno para cada caracter del texto plano: S_i, teniendo una extensión del alfabeto definido por N. La razón del uso del método heurístico es para definir las reglas, respecto a la extensión del alfabeto de cifrado/descrifrado, el cual, es controlado por el vector K_i con extensión N. Ello ha dado lugar a tres versiones del random Caesar, empleando *módulos* (*mod*): *9* y *255* para la *versión I* [1], [19], [36], y *módulos*: *95* y *120*, para la *versión II* [1]-[2], [7]. Una breve descripción sobre el uso de los *módulos*, la podemos encontrar en [2], [19]. En términos generales, refiere que: "Los valores del alfabeto dependen del valor que tenga el caracter de texto plano S_i (a cifrar), con un rango de: $S_i \pm 9$, que equivale a: $(S_i - 9) \leq S_i \leq (S_i + 9)$. En cambio, para random Caesar II, existen dos tipos de módulos: *mod 95* y *mod 120*. En el primer caso, se tiene un alfabeto de *95* caracteres que incluye valores enteros u ordinales desde *32* hasta *126* de la tabla ASCII. Por lo tanto, el alfabeto incluye solamente caracteres imprimibles en pantalla (desde el espacio hasta la tilde: ~). Para el caso del uso del *módulo 120*, se incluyen *120* caracteres en el alfabeto, con valores ordinales dentro del rango entre *30* y *150*, correspondientes a la tabla ASCII, cuyos caracteres no imprimibles en pantalla, son considerados « *ruidosos* », los cuales, están dentro del rango entre *30* y *31*, así como, entre *127* y *150*, ambos casos, valores ordinales de la tabla ASCII" [2], [19].

Cabe aclarar que, los desplazamientos K_i, en random Caesar, definen el cifrado parcial C_i. Sin embargo, al terminar el procedimiento de encriptación, se construye un paquete final, obtenido por la operación: $FinalPackage = C_i \,\&\, K_i \,\&\, OrdChr(C_i)$. Dicho esquema, incluye de manera intercalada: la secuencia encriptada C_i, el valor de desplazamiento K_i, y se repite la secuencia cifrada C_i. Este valor debe ser ordinal cuando se usa la primera versión de random Caesar I. En cambio, para el caso de random Caesar II, el valor repetido de C_i, debe ser un caracter en formato ASCII, o al menos, dicha operación indica la función: *OrdChr* (porque para: random Caesar I, refiere a un valor ordinal), mientras que el operador: *&* , refiere a realizar una concatenación de caracteres. Cabe aclarar, que el motivo de guardar en el *paquete final*, la repetición de este valor C_i, se hace con el propósito de "inyectar ruido", lo cual, se diseñó con la intención de confundir a los ciber-delincuentes, porque cuando fue presentado random Caesar por primera vez [36], también se hizo la presentación del algoritmo genético: "Noised" random pseudo-hexadecimal GAs. Y debido a que este último guarda tres valores intercalados en su paquete final de encriptado, el simple hecho de que random Caesar, también guardara tres valores intercalados, y considerando que en ambos casos, estos valores fueron seleccionados de manera aleatoria, ello puede implicar que, para un ciber-delincuente sería difícil saber si el paquete final se encontraba cifrado con random Caesar o con la versión "Noised" random pseudo-hexadecimal GAs. Posteriormente, se descubrió que no era necesario este valor adicional en el paquete de cifrado final, siempre y cuando, se haya "camuflajeado" el paquete final como texto. Ello da lugar a dos versiones abreviadas de random Caesar, las cuales, fueron aplicadas inicialmente al texto plano, y han sido denominadas como: reduced random Caesar [1]-[2], y reduced random mutation [1]. Ambos casos, aplican el mismo procedimiento de cifrado parcial que utiliza random Caesar, solo difieren en el *empaquetado final* del encriptado. Para el primer caso, la secuencia cifrada: C_i, y el valor de desplazamiento: Ki, son convertidos a caracteres en formato ASCII, y se elimina el valor C_i repetido del *paquete final*. Este proceso puede ser definido formalmente [1], [19], como sigue: $FinalPackage = Char(C_i) \,\&\, Char(K_i)$. Dónde: $C_i = (\, Ord(S_i) + Ord(Char(K_i))\,)$ *módulo 120*, lo cual, implica la suma entera de dos números ordinales de la tabla ASCII, validados por la función *Ord*. Mientras que, el operador *&* , se refiere a la concatenación de dos valores camuflajeados en formato ASCII, definidos por la función: *Char*. En cambio, en caso de reduced random mutation [1], se calcula: C_i, utilizando el mismo procedimiento empleado en: reduced random Caesar, y se procede de igual manera con la construcción del *paquete final*, pero difieren en que: reduced random mutation, aplica un proceso basado en mutación a cada par de la secuencia cifrada, el valor par de (C_i, K_i), se intercambia por (K_i, C_i), procedimiento muy similar al empleado en la fase de mutación por intercambio, que suele incluirse en un algoritmo genético. Este procedimiento, puede ser definido formalmente [1], como se muestra enseguida: $FinalPackage = (\, Char(C_1) \,\&\, Char(K_1)\,) \,\&\, (\, Char(K_2) \,\&\, Char(C_2)\,) \,\&\, \ldots \,\&\, (\, Char(C_{i-1}) \,\&\, Char(K_{i-1})\,) \,\&\, (\, Char(K_i) \,\&\, Char(C_i)\,)$. Cómo puede observase, en el apartado del cifrado parcial: C_i, respecto al uso del vector K_i, tanto en reduced random Caesar, como en reduced random mutation, han sido delimitados sus valores en el alfabeto usando un *módulo=105*, y por ende, solo se admiten valores de K_i (aleatorios con reemplazo), dentro del rango entre *0* y *105*, para evitar que el caracter cifrado con desplazamiento, exceda el valor ordinal *255* de la tabla ASCII [2], [19].

Lo anterior, considerando que el valor máximo para un módulo *120*, es el ordinal *150*. Por lo tanto: $(C_i + K_i) = 150 + 105$ (máximo valor de C_i y K_i, respectivamente), se obtiene como resultado el valor máximo de la tabla ASCII [19]. Según Rangel et al. [19], se presume que dichos esquemas utilizan inteligencia artificial, ya que, sus procedimientos de cifrado, son equivalentes a los empleados en las fases de un algoritmo genético abreviado, porque solo se emplean las fases de selección aleatoria y mutación por intercambio, respectivamente, para cada método. En la presente investigación, no se llevó a cabo ningún experimento con reduced random Caesar o reduced random mutation, porque se ha considerado para otro posible trabajo futuro.

Otra línea de investigación, relacionada con la inyección de ruido, recomienda aplicar "el ruido", al texto cifrado previamente por un algoritmo de encriptación estándar [2], [19]. Este tipo de estrategias, de acuerdo con Rangel et al. [19], lleva como propósito confundir a los ciber-delincuentes, e incluso, estar prevenidos contra futuros ataques de algoritmos basados en computación cuántica [20]-[21], [42]. Las estrategias presentadas [2], en este contexto, se conocen como: *random noisy* y *reduced noisy*. El primer caso, sugiere el uso de: random Caesar II mod 120 [1], [7], siendo aplicado sobre textos cifrados por algunos algoritmos de encriptación estándar. Este esquema, puede ser definido por: $RandomNoisy_i = Char\ (Ord\ (StandardEncryption_i) + Ord\ (K_i)) \& Char\ (K_i) \& Char\ (StandardEncryption_i)\ módulo\ 120$. Dónde: la función: *Char*, regresa el caracter *ASCII* de su argumento, mientras que, la función: *Ord*, regresa el valor entero de su argumento o convierte un caracter ASCII a su valor ordinal entero. La estrategia: *reduced noisy*, del segundo caso, recomienda aplicar el método: reduced random Caesar, también sobre textos encriptados por un algoritmo estándar. Este procedimiento, puede ser obtenido por: $ReducedNoisy_i = Char\ (Ord\ (StandardEncryption_i) + Ord\ (K_i)) \& Char\ (K_i)$. Ambas alternativas, el vector: $StandardEncryption_i$, se refiere a un texto cifrado por algún algoritmo de encriptado estándar. En dicho estudio [2], fueron evaluados dos algoritmos de cifrado asimétrico: RSA-2048 y ECIES-SECP-256-R1, así como, seis algoritmos simétricos: RC4, WEP, AES-256, Blowfish, DES y 3DES. Según Rangel et al. [19], *"esta fusión de técnicas, en recientes investigaciones, ha sido empleada, primero aplicando el algoritmo de cifrado estándar sobre el texto plano, y posteriormente, se inyecta ruido al resultado encriptado, a través de la aplicación de random Caesar II mod 120, para el caso de estrategias: random noisy; mientras que, para el caso de estrategias: reduced noisy, se aplica camuflaje de textos con: reduced random Caesar* ". El resultado de esta fusión de técnicas, ha dado lugar al diseño de nuevas alternativas "ruidosas", las cuales, han sido denominadas como: random noisy AES (AES-256), random noisy DES, random noisy 3DES, random noisy Blowfish, random noisy RC4, random noisy WEP, random noisy RSA-2048 y random noisy ECIES SECP-256-R1, como resultado de la combinación del algoritmo estándar y random Caesar II mod 120. En cambio, los derivados de la fusión del cifrado estándar con reduced random Caesar, fueron denominados como: reduced noisy AES (AES-256), reduced noisy DES, reduced noisy 3DES, reduced noisy Blowfish, reduced noisy RC4, reduced noisy WEP, reduced noisy RSA-2048 y reduced noisy ECIES SECP-256-R1. En este mismo respecto, ha sido diseñada otra nueva alternativa [19], pero solamente basada en la estrategia: *random noisy* [2]. Se trata de un caso de estudio aplicado, mediante el uso del algoritmo: GOST R 34.12-2015, dando lugar a la estrategia denominada como: random noisy GOST [19], la cual, ha sido definida formalmente como: $RandomNoisy_i = Char\ (Ord\ (StandardEncryptionGOST_i) + Ord\ (K_i)) \& Char\ (K_i) \& OrdChr\ (StandardEncryptionGOST_i) \bmod N$. La función: *StandardEncryptionGOST*, regresa el resultado cifrado, después de la aplicación del algoritmo: *Kuznyechik*, del método: GOST R 34.12-2015. Según Rangel et al. [2], [19], lo anterior *"podría mal interpretarse como un doble cifrado de datos empleando diferentes algoritmos de encriptado. Sin embargo, no está enfocado de dicha manera, ya que, la aplicación repetida de dos o más algoritmos estándar sobre un texto plano, aplicado posteriormente, sobre el resultado de texto cifrado, ello podría ser desencriptado fácilmente por un ciber-delincuente, pero ello no ocurre con estrategias: random noisy, debido a que, en este caso, el ciber-delincuente desconoce dónde se encuentra localizado el «ruido», en el texto cifrado, y por ende, tendría prácticamente que «adivinar», dicha ubicación en la secuencia encriptada* " [19]. En la presente investigación, solamente se ha considerado el estudio de la estrategia: random noisy, siendo aplicada exclusivamente, en combinación con el algoritmo estándar: Camellia. Empero, el estudio de estrategias "ruidosas" con: Camellia, basadas en: reduced noisy, no fueron implementadas, porque ello podría ser considerado para un trabajo futuro.

La presente investigación, es continuidad de este tipo de estrategias, particularmente, las basadas sobre: *random noisy* [2], [19], porque se observó que estos esquemas, solamente se han experimentado para algunos algoritmos de cifrado estándar, pero no se han incluido pruebas sobre en el uso del algoritmo Camellia. Dicha situación, no se considera buena para las organizaciones, particularmente aquellas que han adoptado el uso de Camellia, considerando que las estrategias basadas en:*random noisy* , han revelado resultados muy prometedores [19]. Por ello, la importancia de esta investigación, porque se realizan experimentos basados en inteligencia artificial para la inyección de ruido a texto cifrado por el algoritmo Camellia, situación, que abre un gran abanico de oportunidades a las organizaciones, respecto al uso de Camellia con inyección de ruido. Del mismo modo, se considera importante el presente trabajo, para poder dar aportación empírica a favor de la metodología denominada: random noisy, debido a que, en recientes investigaciones [2], [19], a pesar que se señala que este esquema no es considerado aún, una metodología estándar, se resalta el beneficio de que la inyección de ruido, ha sido poco estudiada en la literatura, y por ende, tampoco han sido estudiadas vulnerabilidades mediante el uso de algoritmos basados en computación cuántica [19]. Adicionalmente, la presente investigación, se considera importante, porque se lleva a cabo una exploración del problema de inseguridad de datos digitales, presentando alternativas para el cifrado de dinámico basado en inyección de ruido con IA, con el propósito de confundir y/o retrasar, a los ciberdelincuentes, en el momento de querer descifrar nuestra información, y por ende, pueda mejorar la seguridad de datos en las organizaciones [7]. Esta es precisamente, una de las hipótesis que guía este trabajo, que ¿A través de la inyección de ruido en un texto cifrado, puede incrementarse el grado de seguridad, por ejemplo, en una cadena de texto encriptada e incluso estar prevenidos contra futuros ataques de computación cuántica?. Una aproximación para resolver dicho problema, la podemos encontrar en otras investigaciones [2], [19].

En lo que concierne con la presente investigación, se propone el uso de inyección de ruido con IA sobre textos cifrados, introduciendo una nueva definición basada en el algoritmo: Camellia [14], [32]-[33], con el propósito de mejorar o incrementar la seguridad de dicho criptosistema, incorporando mayor dificultad para ser descifrado. Se comparan cinco algoritmos, basados en estrategias "ruidosas aleatorias" (random noisy) [2], [9].

Los algoritmos de cifrado estándar estudiados son: DES, 3DES, AES-256, Blowfish, GOST R 34.12-2015 (versión Kuznyechik) y Camellia; así como, las variantes "ruidosas aleatorias" (random noisy), las siguientes: random noisy DES, random noisy 3DES, random noisy AES, random noisy Blowfish, random noisy GOST, y por supuesto, la nueva alternativa de cifrado aleatorio diseñada con inyección de ruido, que en la presente investigación, ha sido denominada como: random noisy Camellia.

2 Metodología

En el presente trabajo, se introduce una nueva propuesta basada en estrategias: random noisy, la cual, consiste en la fusión del algoritmo de cifrado estándar Camellia [14], [32]-[33], combinado con la inyección de ruido basada en el uso de: random Caesar II mod 120 [1]-[2], [7]. Es por tal razón, que el tipo de investigación se considera: experimental y exploratoria, debido a que, algoritmo Camellia combinado con el uso de estrategias "ruidosas aleatorias" [2], no se había llevado a cabo antes del desarrollo de la investigación, o al menos no con el mismo enfoque o propósito que aquí se presenta.

Para el desarrollo de la presente investigación, fue necesario utilizar algunos materiales y datos. Los materiales empleados fueron: hardware [88], y software [43], [89]-[91]. En el primer caso, se refiere a los equipos de cómputo que fueron utilizados. Se trata de dos computadoras con velocidad de procesamiento de 2 GHz, capacidad de 8 GB en memoria y espacio disponible en disco de 500 GB. También, se utilizó un equipo de cómputo móvil, con velocidad de procesamiento de 2 GHz, capacidad de 4 GB en memoria y un espacio disponible en almacenamiento interno de 16 GB. El uso del cómputo móvil, fue empleado con el propósito de corroborar resultados. En este aspecto, no se observó diferencia significativa entre los resultados obtenidos con las computadoras y el equipo móvil. Con

respecto al software utilizado, las computadoras tenían preinstalado, una actualización del sistema operativo Windows 10 [90], mientras que el equipo móvil, tenía instalado sistema operativo Android [91], versión 9. En lo concerniente con el lenguaje de programación, se utilizó Python 3 [43], en las dos computadoras, antes mencionadas. Sin embargo, en el equipo móvil, fue empleada la aplicación: PyDroid3 [92], para realizar algunas pruebas de implementación y experimentación. Tanto en Python 3, como en PyDroid3, fueron instaladas algunos paquetes o librerías [46]-[49], para poder trabajar la encriptación de datos con los algoritmos de cifrado estándar: DES, 3DES, Blowfish, AES-256, GOST R 34.12-2015 (Kuznyechik) y Camellia.

En lo que concierne con los datos utilizados, se trata de muestras de entrenamiento (ME) [76]-[77], donde cada patrón de entrenamiento [76], su etiqueta de clase es el texto plano. Este patrón, contiene valores reales en formato de doble precisión (tipo: double), que guardan los tiempos de cifrado y descrifrado, así como, el porcentaje de error ocurrido. Se considera como error, si la secuencia cifrada, al ser desencriptada no obtiene el mismo valor que el texto plano. Esta cadena cifrada, también se incluye como atributo del patrón de entrenamiento, la cual, consiste en una secuencia de caracteres en formato: hexadecimal, ASCII o UTF-8, ya que, son textos de palabras encriptadas por algún algoritmo de cifrado de datos, simulando ser claves (password) cifradas. En algunos casos, estas secuencias de texto, fueron almacenadas en formato de vectores con tipo de datos entero, porque guardan el valor ordinal de cada correspondiente caracter cifrado. Ello fue diseñado, de manera separada, para cada uno de los algoritmos o métodos de encriptación, aquí evaluados, incluyendo las estrategias: random noisy. Es por ello, que en algunos casos las secuencias cifradas son valores del tipo hexadecimal, porque ese tipo de datos arrojó como resultado las funciones de cifrado de las bibliotecas del lenguaje Python, que fueron empleadas, mientras que, las nuevas alternativas "aleatorias ruidosas", debido a que, no utilizan librerías de Python para su generación, se implementaron para que regresaran resultados del tipo caracter ASCII o UTF-8. El tamaño máximo de las secuencias encriptadas es de 255 caracteres, excepto para los algoritmos: 3DES y Blowfish. Lo anterior, debido a que, 3DES solamente regresaba secuencias cifradas de 22 caracteres, cortando la cadena de texto plano original. Del mismo modo, Blowfish regresaba la cadena cifrada incompleta, limitado sus resultados con un tamaño máximo de 13 caracteres. Esta situación, se tuvo que resolver en la implementación, realizando rellenos aleatorios en formato hexadecimal. Sin embargo, dicha acción, de acuerdo con lo observado en la etapa de experimentación, no se consideró viable, porque pone en desventaja al resto de las estrategias, ya que, en la presente investigación, se están evaluando los tiempos de cifrado y descrifrado de datos. Es por tal razón, que los resultados mostrados en la Tabla 1, presentan el promedio (y desviación estándar), de una muestra de experimentos, que fueron llevados a cabo, en la misma igualdad de condiciones, trabajando una cadena corta: "▓Welcome▓", que además incluye como "ruido", caracteres con valores ordinales que están fuera del rango de la tabla ASCII, tal es el caso de '▓' y '', con valores ordinales: 9619 y 65533. Por último, cabe recordar, que a cada algoritmo o método de cifrado, aquí experimentado, se le diseñó su propia ME [77], y fueron evaluados cada uno de manera separada. Debido a que, el procedimiento para medir el acierto y/o la estimación del error de cada experimento, utilizando las ME, antes mencionadas, consiste en la aplicación de una modificación del método validación cruzada [1], [7], empleado con cinco repeticiones, para poder comparar resultados con otras investigaciones [2], [19]. Por lo tanto, en cada iteración, ha sido empleado un 20% para "test" o muestra de control, y un 80% como muestra de entrenamiento.

En lo concerniente con el procedimiento empleado, así como, los algoritmos de cifrado y/o estrategias de inyección de ruido experimentadas, fue llevado a cabo, en la forma que se describe a continuación.

Primero, se aplicó cifrado al texto plano, siendo validado con el proceso de descrifrado correspondiente, calculando los tiempos de encriptado y desencriptado, así como, el porcentaje de error ocurrido, evaluando separadamente, cada algoritmo estándar: DES, 3DES, Blowfish, AES-256, GOST R 34.12-2015 y Camellia. Lo anterior, realizado con el propósito de comparar resultados con otras investigaciones [2], [7], [19]. El porcentaje de error se obtiene cuando la secuencia cifrada, al ser desencriptada, no es igual que el texto plano, pero tiene que calcularse, de acuerdo con el tamaño de la secuencia cifrada, para conocer, cuántos de los caracteres no pudieron ser descifrados, incluyendo los caracteres *ruidosos* : '▓

y '', obteniendo el porcentaje, en un rango de probabilidades del 0 al 1 (el valor de 1 significa 100%, es decir, no se pudo descifrar ningún caracter de esa secuencia de caracteres o vector con valores ordinales).

Tabla 1: *Resultados obtenidos de la experimentación con los algoritmos estándar y los métodos de cifrado de datos dinámico con inyección de ruido, basados en estrategias: random noisy [2], [19]. Se ha seleccionado una muestra como ejemplo usando el texto plano: Welcome. El cálculo de los tiempos para el cifrado/descrifrado de datos fue convertido en milisegundos. En caso del parámetro: ERROR, fue calculada la estimación en medidas de porcentaje. Todos los valores numéricos refieren a promedio generado por la validación cruzada [1], [7]. Aparece entre paréntesis, el cálculo de la desviación estándar. Se observa que las estrategias: random noisy, reportan siempre cifrados dinámicos, mientras que, los algoritmos estándar, aquí evaluados, generan resultados estáticos en todos los casos, aunque esta situación no implica necesariamente que sean vulnerables.*

METODO O ALGORITMO DE CIFRADO	Parámetros evaluados	Resultados del experimento
DES	Test 1	d535c337be1d94db5b940b75bce124dcda93ffa35791bf07
	Test 2	d535c337be1d94db5b940b75bce124dcda93ffa35791bf07
	TC	1.143980026 (0.50876953)
	TD	0.193710327 (0.01871141)
	ERROR	1.0 (0.0000)
random noisy DES	Test 1	è¤è¤o¤®{®»»ªGªZ'Z¤h¤¤e¤¤.¤¨C¨¹¹¤,¤,¤,g3gÇcÇæ¤æ³~³“1“'X‘ç¤ç£“ÃÅcÅ¤O¤ x ¤:¤ô‘ô¤4¤£r£¤rº¤€L€¤7¤¤:¤ÊfÊÁ`Á°w°¤]¤×q×ßyßºYºi6it?t‘Z‘¼¤¼zIz¤'¤ÕoÕY)Yp9p
	Test 1	Ä`ÄÂ¤Â¤N¤¤N¤°M°¤O¤`- `k4ké¤é¡<¡¤I¤¬H¬®u®\(\ËgËã¤ã€K€”2”¬s¬Ç“Çµ…µ¤9¤xAxX#X»Y»´Q´ç¤çf5f±¤±¶¤¶ I ¤:¤–2–®M®g.gg4g“-“î¤îï¤ïtAt•`•Ã¤ÃÀ¤Àp?p¤&¤Ä^Ä¤]¤Â¤Â
	TC	1.6333770751953125 (0.48977593822476057)
	TD	0.5848884582519531 (0.038977659805045535)
	ERROR	1.0 (0.0000)
3DES	Test 1	d535c337be1d94db5b940b75bce124dc620c51e2380a3d5c
	Test 2	d535c337be1d94db5b940b75bce124dc620c51e2380a3d5c
	TC	7.681427002 (3.938826769)
	TD	0.500459671 (0.018421763)
	ERROR	0 (0)
random noisy 3DES	Test 1	ÐlÐ—b—l9lxCxÜyÜs@sxEx¤U¤¤7¤¤'¤¤Y¤ò¤ò…L…¤M¤ñ¤ñ¤&¤g2g·U·¤N¤uAu{K{¤$¤¨q¨ TTËiËÕrÕÛvÚ¤_¤¤T¤Y%Yi¤¤- ¤'\'tBt¤h¤ÖsÖÈ“ÈU$U¤5¤¤Y¤r?r®v®¤h¤€€V#V¤¤¤d¤©F©
	Test 2	¬H¬¤X¤¯|¯¤d¤å¤å¼¤¼¤V¤f/fÂ`Â£>£Á¤ÁÎjÎx?x¬x¬¤#¤ºX°§r§¤&¤h/hh4hW'W¤,¤¨q¨¹¤¹° N°§D§¤$¤£r£°¤°ç¤ç'.'ÊgÊj4j¬z¬•e•¤*¤q<q¤R¤æ¤æ®|®Á¤ÁWW¤W¤¤%¤…R…î¤íÉ”ÉÕr Õ
	TC	8.45942497253418 (4.189240691107667)
	TD	0.674285888671875 (0.017911737048241966)
	ERROR	0 (0)

GOST R 34.12-2015	Test 1	2b2c98d7ebc5eb39ee563dc2c2abdc8fa40423ca32f7cbef6eb9bed261116b75
	Test 2	2b2c98d7ebc5eb39ee563dc2c2abdc8fa40423ca32f7cbef6eb9bed261116b75
	TC	21.26753807067871 (0.03488449571089547)
	TD	21.396074295043945 (0.022180206439834906)
	ERROR	0.2 (0)
random noisy GOST R 34.12-2015 (mod 120)	Test 1	j8jê▯ê®|®ªGª]Ì”Ìa)aâ~âh1hì▯▯íé▯é…P…▯3▯ÄbÄ m ´{´à{àÀ[Àm8mq;qyFyè▯è¶S¶vDvø•øxFxÎmÎØvØ▯5▯ÓpÓs;sÒlÒºYº▯Y▯ç†çl8Id2d▯P▯▯,▯ ì▯ìÃ▯Ã▯g▯ë…ë©r©ÕrÕÔrÔ▯4▯¯I¯▯K▯ªEª¸V¸ t —5—ªEªÍíÍ–d–▯X▯Á▯Áb1bd3d▯d▯ó‘ó °y°e0e
	Test 2	®|®»Y»ªxªë▯ëb)b▯N▯Ûw Ûv?vÒmÒ¡?¡ÎkÎzEzÙtÙ▯7▯zGzÅ▯ÅÝxÝ¹T¹SS▯g▯▯T▯▯ ▯…"…Ã‘ÃÒoÒPPã▯ã¶T¶¶R¶ç▯ç¾▯¾ : ºYºc/cÃ“ÃÇ“Çp>puBuÝzÝé▯êÁ▯Á]+]▯)▯¡j¡¡C¦¼Z¼ì▯▯)▯“]”®I®ÑoÑÀ▯▯Àß}ßá|á▯▯´▯´n8n Â‘Â”c”¯~¯´~´§E§▯V▯s>s
	TC	21.748065948486328 (0.0926154244121027)
	TD	21.419649124145508 (0.03516320800820544)
	ERROR	0.2 (0)
Blowfish	Test 1	4af747eaabe473251f42200cf8fda7f2
	Test 2	4af747eaabe473251f42200cf8fda7f2
	TC	7.559738159 (3.79314656)
	TD	0.648612976 (0.037382338)
	ERROR	0 (0)
random noisy Blowfish	Test 1	—c—ð▯ð▯&▯w@w§s§µ~µÒmÒ▯(▯▯▯▯.▯“.“Á▯Ád-dµ▯µÃ‘Ã[&[f5fÕoÕ¿▯¿Æ”ÆÃ‘Â¢ç£s£ä▯ä±K±Y!Y®H®▯‘▯▯<▯Ë”Ëø’ø▯]▯
	Test 2	▯j▯ô“ô▯)▯¼…¼j6jb+b¼W¼ÙxÙØwØ½[½ÔoÔ’]‘c,c§t§´▯´çm¢¬{¬²L²»▯»▯Å“ÅW%WW'WÅ •Å¹V¹Ä^Ä▯Q▯'+‘ < ¯N¯§p§é▯éZ(Z
	TC	7.854413986206055 (3.7619636933035645)
	TD	0.9457588195800781 (0.03495134646070494)
	ERROR	0 (0)
AES-256	Test 1	3030303030303030303030303030319421780fc40c59f14796a598115d8e513 9f6f9666f67cb30fcda01f22739eeda
	Test 2	3030303030303030303030303030319421780fc40c59f14796a598115d8e513 9f6f9666f67cb30fcda01f22739eeda
	TC	7.311677933 (3.709387166)
	TD	0.616607666 (0.016988446)
	ERROR	0 (0)
random noisy AES-256	Test 1	»▯»°€°£p£Â’Â▯\▯]-]i6iV&Vi6i▯V▯▯L▯€P€▯P▯¥u¥½▯½i9ig4g®~®▯T▯¥u¥i6itDt m ▯O▯|I|o?ol9l´▯´–c–“c“Â▯ÂOO¥l¥▯▯U▯|J|▯_▯…N…“[“OO¼V¼▯)▯¼▯¼¬|¬¡>¡▯L▯Ã▯Ã¼V¼ˆ‘°| °d-d▯O▯¨r¨ê▯êm8mc*cÀ▯À],]¾▯¾—b—ÛwÛÂ▯ÂØsØ▯Q▯Ä“Ä”a”Î•Î°J°yCyî▯î¾…¾Y#YX"X¯ y¯ßyß▯L▯▯T▯¥B¥▯9▯U"U▯_▯¿Y¿°M°ÏkÏ°O°yIy‘ˆ▯"▯°~°®|®▯Q▯||I¾…¾ç▯ç▯$▯ºVº§F§
	Test 2	▯S▯ } ®{®▯]▯T!T¬|¬Z'Z▯Y▯e2e▯k▯»▯»’b’▯k▯»▯»f3f] -]Ç”Ç▯Y▯]*]▯O▯▯i▯wGw\)\ } Æ“Æc3c¯|¯¿▯¿^+^\,\▯Q▯…T…,▯,f2fµ▯µj9j•^•¨p¨¨▯_▯¡¡Èe È£o£«{«—4—È”ÈÄ▯Ä»U»½▯½▯K▯¥n¥—^—³}³▯>▯©t©–]–––_—½▯½n=n¨s¨ I ªrª▯%▯▯ O▯°▯°vCv_&_®H®Ç’Ç▯)▯▯J▯V Vh2h¤n¤ÒlÒ▯b▯▯U▯è…è▯;▯n;n▯_▯▯"▯½Z½ð▯ðÐoÐW'W▯[▯ÛuÛº▯º▯k▯▯,▯,p=p–]–ø“øö’ö â~â▯'▯

	TC	8.463754653930664 (3.9351400019700353)
	TD	1.0021591186523438 (0.011940241309867648)
	ERROR	0 (0)
Camellia	Test 1	30303030303030303030303030303101e280d8699c96ac7dcc91714357fe176fd336eb2594aeb9619b2cfe3a1a5c90
	Test 2	30303030303030303030303030303101e280d8699c96ac7dcc91714357fe176fd336eb2594aeb9619b2cfe3a1a5c90
	TC	6.286382675 (3.222883801)
	TD	0.474720001 (0.011974389)
	ERROR	0 (0)
random noisy Camellia	Test 1	©v©▯[▯³€³wGw[([▯k▯{H{c3c{H{▯n▯d1d~N~Â▯Âd4dS S—g—▯T▯P P¯|¯"c"€M€▯T▯§t§▯Q▯"a"[+[▯P▯€P€}J}¿▯¿wDw±€±°€°S"S²M²±▯±_'_\,\±M±€H€▯Q▯j1j ½▯½Ã`Ã«r«}G}ÓrÓ"1""\"è▯èÖsÖë▯ë´{´▯Q▯p9p¦u¦n:n±~±](Jv?v°J°ÊeÊ▯\▯±z±®x®Ìfl`ò▯ ò▯i▯▯[▯▯W▯"."▯<▯}K}d/d▯d▯Ç"ÇÜ{ÜÞyÞ'/'°w°h2h`)`XXÁ_Á▯i▯ÔqÔ×q×¬G¬"`"ÅdÅ▯U▯▯" ▯xCxà}àªqªY)Y
	Test 2	¢o¢xHxU"Un>np=p¾▯¾▯P▯▯V▯d1d▯n▯▯l▯S#S▯i▯¡q¡▯]▯f6fuBu^.^▯N▯S#SX%Xªzª£p£«{ «j7j▯`▯▯h▯▯W▯Z'Zªzªb/b{J{Á'Á`/`ÏjÏV$Vl"ÌP PÛwÛ"[",▯h/h®u®ÝzÝ▯K▯l6l³R³▯▯"\"▯)▯▯%▯ÑnÑ®u®[*[▯L▯yHy²~²▯O▯·▯·«t«ÈbÈßzß …T…¹▯¹Y#YÓmÓ£?£¿▯¿r?r¨¨r¨³N³±O±r@rvAvªqª€L€í▯í▯+▯«l«^%^▯U▯yHy¨¨o¨¨ä▯ä"a"ë▯ë ▯8▯▯(▯¯|¯€€«z«£B£~l~ð▯ðd+d¹▯¹
	TC	10.436182022094727 (4.8905005416441565)
	TD	1.3324546813964844 (0.020668074692113732)
	ERROR	0 (0)
Random Caesar II (mod 120)	Test 1	◢O◢ÂkÂ▯'▯±E±ÍjÍ·H·ßrß▯▯&▯₽"₽Å -Å ((
	Test 2	◅1◅Õ~Õ"/""("ð▯ðã'ã▯▯´O´▯▯▯▯Ø W▯ :0:
	TC	0.146131516 (0.010897759)
	TD	0.051755905 (0.001176331)
	ERROR	0 (0)

Posteriormente, esta información se convierte en un patrón de entrenamiento, el cual, es guardado en las ME, antes descritas, hasta completar un total de 1000 patrones, información que se utiliza inmediatamente, para llevar a cabo la aplicación de una modificación de la validación cruzada [1], [7], sometida por cada algoritmo o método de encriptación empleados, separadamente, con la finalidad de obtener un promedio y su respectiva desviación estándar (ver Tabla 1).

Una segunda estrategia empleada, consiste en aplicar la metodología: random noisy, pero solo con algunas de las alternativas "aleatorias ruidosas" reportadas en [2], así como, la recomendada en [19]. Por lo tanto, se ha llevado a cabo la experimentación con las estrategias: random noisy AES, random noisy DES, random noisy 3DES y random noisy Blowfish, las cuales, han sido definidas formalmente como: $RandomNoisy_i = Char\ (\ Ord\ (\ StandardEncryption_i\ ) + \ Ord\ (\ K_i\ )\ )\ \&\ Char\ (\ K_i\ )\ \&\ Char\ (\ StandardEncryption_i\ )\ módulo\ 120$. Del mismo modo, se llevaron a cabo experimentos con: random noisy GOST módulo 120 [19], el cual, ha sido empleado como sigue: $RandomNoisy_i = Char\ (\ Ord\ (StandardEncryptionGOST_i\ ) + \ Ord\ (K_i\ )\ )\ \&\ Char\ (K_i\ )\ \&\ OrdChr\ (StandardEncryptionGOST_i\ )\ módulo\ 120$. El procedimiento de estas estrategias: random noisy, ya han sido descritas previamente. Del mismo modo, la explicación sobre el uso del módulo (mod) N, también se ha realizado anticipadamente. Estos experimentos, fueron realizados mediante la inyección de ruido, para cada uno de los algoritmos de cifrado estándar, de manera separada, todo con la finalidad de observar el comportamiento de cada estrategia, formulada por la fusión del algoritmo estándar, siendo combinado con la metodología: random Caesar II mod 120. Por lo tanto, se procede primero cifrando el texto plano con el algoritmo de

encriptado estándar, y posteriormente, a la secuencia cifrada, se le aplica: random Caesar II mod 120. El cálculo de los tiempos de cifrado/descrifrado, se inicia a contabilizar, desde el momento que comienza a aplicarse el algoritmo estándar de encriptación de datos. Por lo tanto, los tiempos calculados incluyen el proceso de retardo del algoritmo estándar y se le suma el tiempo que tarda en el cifrado por random Caesar II. La etiqueta del patrón de entrenamiento, es el mismo texto plano. Por lo tanto, para obtener el porcentaje de error, ello incluye doble procedimiento: primero, eliminar el ruido a la secuencia cifrada, y posteriormente, a dicho resultado, debe aplicarse el descrifrado usando el algoritmo estándar. Esta última secuencia obtenida, debe ser la misma que el texto plano, o de lo contrario, se calcula el porcentaje de error, de acuerdo con el tamaño de la secuencia de caracteres cifrada. Del mismo modo, toda esa información se incluye en los patrones de entrenamiento, que se guardan en la ME, para cada algoritmo estándar "aleatorio ruidoso", y se aplica la modificación [1], [2] de la validación cruzada, para obtener el promedio y la desviación estándar (ver Tabla 1), de cada conjunto de experimentos. Esta experimentación, también lleva como propósito, poder comparar resultados con otras investigaciones [2], [19]. Se ha considerado importante realizar esta evaluación de estrategias: random noisy, para poder comparar resultados con la nueva propuesta aquí diseñada, basada en el algoritmo de Camellia, la cual, se describe enseguida.

Finalmente, para mejorar la seguridad de los datos digitales, una tercer estrategia, consiste en implementar la alternativa "ruidosa aleatoria", correspondiente al algoritmo: Camellia [14], [32]-[33], la cual, ha sido aquí denominada como: *random noisy Camellia* . El procedimiento para cifrado de datos "ruidoso" con dicha estrategia, ha sido diseñado, siguiendo el mismo camino que: random noisy GOST [19], todo con la finalidad de poder comparar resultados. Por lo tanto, dicho proceso puede ser definido como: $RandomNoisy_i = Char\ (Ord\ (StandardEncryptionCamellia_i) + Ord\ (K_i)) \,\&\, Char\ (K_i) \,\&\, Char\ (StandardEncryptionCamellia_i)\ módulo\ 120$. Dónde: La función *StandardEncryptionCamellia* , permite encriptado de un texto plano utilizando el algoritmo estándar Camellia y regresa como resultado el texto cifrado correspondiente. La función *Char* , regresa un valor caracter de la tabla ASCII o UTF-8, según corresponda. La función *Ord* , regresa el valor entero de su argumento o convierte un valor caracter a su correspondiente ordinal. El operador *+* , realiza.la suma de dos enteros y el operador *&* , realiza la concatenación de caracteres. Los experimentos realizados con esta alternativa, fueron aplicados utilizando muestras de entrenamiento, también de 1000 ejemplares, siendo empleados los mismos atributos descritos previamente, para el patrón de entrenamiento. La operación presentada previamente para obtener el cifrado: *random noisy Camellia* , en términos generales, indica que primero se realizará el cifrado del texto plano, empleando el algoritmo estándar: Camellia. Este resultado puede ser obtenido en lenguaje Python [43], utilizando la librería: Cryptography [93], mediante el siguiente conjunto de instrucciones: *from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes ; from cryptography.hazmat.backends import default_backend ; from cryptography.hazmat.primitives import padding ; textoplano = "▒Welcome▒" ; IV= b"0000000000000001" ; Key256bits = b"00000000000000000000000000000001" ; Ci = Cipher(algorithms.Camellia(Key256bits), modes.CBC(IV), backend=default_backend()).encryptor() ; relleno = padding.PKCS7(algorithms.Camellia.block_size).padder() ; Ci = (IV + (Ci.update(relleno.update(textoplano.encode()) + relleno.finalize()) + Ci.finalize())).hex() ; print(Ci)* . Posteriormente, se utiliza el resultado C_i , texto cifrado por: Camellia, para aplicar la inyección de ruido con: random Caesar II mod 120 [2], [19]. Se calculan los tiempos de cifrado y descrifrado, así como, el porcentaje de error, construyendo un patrón de entrenamiento, cuya etiqueta es el texto plano, siendo guardado en una ME. Posteriormente, se aplica una modificación de validación cruzada [1], [7], a dicha ME, para calcular los resultados en promedio, con su respectiva desviación estándar (ver Tabla 1).

3 Resultados y discusión

Los experimentos fueron realizados sobre muestras de entrenamiento (ME), previamente diseñadas, utilizando cinco repeticiones de validación cruzada [2], [19], empleando la misma modificación que se describe en [1]-[2], y [7], [19], todo con la finalidad de poder comparar resultados con dichas investigaciones. En términos generales, dicha modificación [2], ha sido adaptada con enfoque de estrategias que no requieren el uso de una muestra de control (MC), para evaluar la precisión de un

modelo, tal es el caso, del proceso para cifrado/descrifrado de datos, experimentado en la presente investigación. Si observamos en otros trabajos [76]-[77], [82]-[83], [94]-[98], podemos apreciar que, se divide la ME, en cinco submuestras, que incluyen aproximadamente el mismo número de elementos por clase, y se extrae una de ellas para ser utilizada como: MC, y así, poder evaluar el modelo. Mientras tanto, se unen las cuatro submuestras restantes, formando una sola, para poder "entrenar el modelo", que es sometido a evaluación, siendo empleado como si fueran patrones nuevos, estos mismos contenidos en la MC. El proceso se repite cinco veces, para poder conocer la desviación estándar, una vez obtenido el promedio o *global accuracy* [99]. Este procedimiento, no pudo ser aplicado en el cifrado/descrifrado de datos, porque la evaluación del algoritmo de encriptado, en la presente investigación, no requieren de un entrenamiento con la ME, y tampoco ser evaluado con una MC, debido a que, el entrenamiento y evaluación se hace antes de crear la secuencia del texto cifrada, por ejemplo, primero se ejecuta el método heurístico o se hace el entrenamiento de la selección aleatoria del GAs, según sea el caso, para generar el vector K_i (de random Caesar), que debe ser aplicado sobre el texto plano, y poder generar el resultado encriptado. Inmediatamente, se descifra la secuencia cifrada, para ser guardados ambos vectores en la ME. En estos términos, la modificación de la validación cruzada, consiste en omitir la evaluación usando patrones de la MC. En su lugar, se procede a calcular el error, pero solo con una parte de la ME (es decir, con cuatro submuestras), si el vector de la secuencia cifrada, al ser desencriptado, no coincide con la entrada del texto plano (que en esta investigación aparece como etiqueta del patrón de entrenamiento), entonces se calcula el porcentaje de error, es decir, se anota cuántos caracteres cifrados, no pudieron ser descifrados, para calcular su porcentaje. Del mismo modo, como lo hace la validación cruzada estándar [76]-[77], se repite la operación cinco veces, extrayendo en forma secuencial, una submuestra distinta en cada iteracion, para poder calcular la desviación estándar y promedio de cada atributo o columna de tipo numérico, que en la presente investigación, se aplicó a los tiempos de cifrado (TC), tiempos de descifrado (TD), y porcentaje de error, en forma global, sin distinguir los elementos por clase, ya que, ello podría ser considerado para una futura investigación, porque requiere de un gran número de explicaciones, puesto que, durante el proceso de cifrado se observó que ha ocurrido ambigüedad, pero dicha situación no ha afectado en la precisión del algoritmo o estrategia evaluada. Ello lo podemos corroborar en la Tabla 1, donde se observa que la mayoría de los resultados, reportan 0% de error. Por lo tanto, la modificación de la validación cruzada, ahora permite evaluar el algoritmo o estrategia de encriptación, simulando la estimación del error en ambientes diferentes, porque se excluye un grupo de patrones de la ME global. Por lo tanto, aplicar el método sin contemplar un grupo de cada clase, asegura resultados con sesgo positivo u optimista. Es decir, garantiza que el resultado en una situación real, pueda ser mejor o igual, que el valor reportado por el método de estimación de error.

Durante el inicio de la experimentación, de manera separada, fueron evaluados sobre texto plano, los algoritmos de cifrado estándar: DES, 3DES, Blowfish, AES-256, GOST R 34.12-2015 y Camellia, tal como se ha descrito previamente. Del mismo modo, se aplicaron experimentos con: random Caesar II mod 120. Lo anterior, para poder comparar resultados con otras investigaciones [2], [7], [19], esta información la podemos apreciar en Tabla 1. Posteriormente, se llevaron a cabo los experimentos, con los métodos de cifrado basados en las estrategias: random noisy [2], [18]. Pero solamente, fueron seleccionados: random noisy DES, random noisy 3DES, random noisy Blowfish, random noisy AES y random noisy GOST. Cabe aclarar que, estos experimentos, fueron aplicados sobre texto cifrado, con el propósito de inyectar ruido. Por último, después de haber diseñado la nueva propuesta: random noisy Camellia, fueron aplicados, de la misma manera, sobre texto cifrado, algunos experimentos con esta nueva variante. Los resultados obtenidos de las estrategias: random noisy [2], [19], se muestran también en la Tabla 1. En su mayoría, los experimentos tuvieron éxito, con excepción de aquellos que reportaron error, tal es el caso de algunas pruebas realizadas con los algoritmos: DES y GOST R 34.12-2015, incluyendo su alternativa: "ruidosa aleatoria". Está situación se debe, al uso de los caracteres fuera del rango de la tabla ASCII, que no pudieron ser controlados por dichas alternativas, porque corresponden a valores de la entrada del texto plano, que no fueron generados por los procesos de cifrado/descifrado. Sin embargo, cabe mencionar, que se considera, falta profundizar un poco más, en los experimentos, extendiendo el uso de cadenas de texto plano y/o texto cifrado, con longitud mayor que 255 caracteres, excepto para el caso de 3DES y Blowfish, por las razones que se explican más adelante. Empero, cabe aclarar que, para cada método de cifrado utilizado, se diseñó su propia muestra de entrenamiento (ME), y fueron evaluados cada uno de ellos, de manera separada.

La estructura de las ME, ya se ha descrito previamente. Sin embargo, cabe recordar que cada una de ellas, fue diseñada, en forma separada, porque corresponde una distinta, a cada algoritmo o método de cifrado de datos que en la presente investigación, ha sido evaluado. Los patrones de entrenamiento que contiene cada una de las ME, tal como se ha descrito anticipadamente, tiene la estructura de vector: *PatrónEntrenamiento = [Test1, Test2, TC, TD, ERROR, Etiqueta]*, y se han almacenado 1000 ejemplares en cada ME. Es por tal razón, que en la Tabla 1, aparece la información estructurada de esa misma manera. Solamente, hay que recordar, que en dicha Tabla 1, lo que se muestra es resultado del promedio, de los experimentos evaluados con la validación cruzada [2], [7], incluyendo entre paréntesis la respectiva desviación estándar. En caso de los: *Test1* y Test2, no son promedios, son ejemplares encriptados, que se han elegido, de manera aleatoria, solamente dos muestras de ese experimento, cuya etiqueta de clase corresponde al valor: "▒Welcome▒". La información que incluye: *PatrónEntrenamiento*, refiere al tiempo de cifrado (*TC*) y tiempo de descifrado (*TD*), que ha demorado el algoritmo o método evaluado para realizar dicha operación, lo cual, ha sido medido en *milisegundos*. También, incluye el porcentaje de error (*ERROR*), tal como se ha explicado previamente, el valor 1 implica 100% de error, mientras que 0.5 significa 50% de error, en esa operación, que se calcula de acuerdo con el tamaño de *Test 1* o *Test 2*, los cuales, tienen la misma longitud. Por ejemplo, si para un texto plano de 10 caracteres, su texto cifrado fuera también de 10 caracteres, el valor 0.5 significa que no pudieron descifrarse 5 caracteres de *Test 1* y/o *Test 2*, y así sucesivamente. Estos valores: *Test*, guardan el texto cifrado por el algoritmo o método de encriptado que se está evaluando con esa misma ME. Se guardan dos valores de prueba, para determinar si los resultados obtenidos son cifrados dinámicos. El tamaño máximo de *Test 1* y *Test 2*, se ha explicado previamente que tiene longitud de 255 caracteres, excepto para los algoritmos estándar: 3DES y Blowfish, los cuales, solo admitieron un máximo de 22 y 13 caracteres, respectivamente. El último valor, denominado: *Etiqueta*, es el texto plano que ha sido encriptado.

También, se ha explicado anteriormente, que los experimentos fueron realizados, utilizando dos computadoras con sistema operativo Windows 10 y lenguaje de programación Python 3, previamente cargados. Además, para corroborar resultados, fue utilizado un equipo de cómputo móvil con sistema operativo Android 9 y la aplicación: PyDroid3, previamente instalados. Las computadoras tenían capacidad de memoria de 8 GB y espacio disponible en disco de 500 GB, mientras que, el equipo móvil cuenta con 4 GB de memoria y espacio libre en almacenamiento interno de 16 GB. En todos los casos, la velocidad del procesador fue de: 2 GHz.

Del mismo modo, se ha mencionado que, para la implementación de los algoritmos de cifrado estándar en lenguaje Python, fue necesario instalar las librerías: Gostcrypto [47], Crypto.Cipher de PyCryptodome [48], Pycryptodome de PyPI [49], y Cryptography [93]. Debido a que, los algoritmos de cifrado de datos estándar, que fueron evaluados en la presente investigación son: DES, 3DES, Blowfish, AES-256, Kuznyechik del método GOST R 34.12-2015 y Camellia. Los algoritmos no estándar, evaluados en la presente investigación son: random Caesar II mod 120, así como, las versiones [2], [19], "aleatorias ruidosas": random noisy DES, random noisy 3DES, random noisy Blowfish, random noisy AES, random noisy GOST, y por supuesto, la nueva alternativa aquí presentada como: random noisy Camellia. Dichas alternativas, también fueron implementadas en lenguaje Python.

Los parámetros utilizados para la encriptación de datos con los algoritmos estándar, se describen enseguida.

Para el caso del algoritmo: DES, ha sido usado el valor: "00000001", como clave secreta con longitud de 56 bits codificada en formato: UTF-8, siendo empleado el modo de libro de código electrónico: ECB (Electronic Codebook Mode) [15]. Los resultados cifrados fueron arrojados en codificación hexadecimal. La implementación del algoritmo: DES, fue llevada a cabo utilizando el conjunto de funciones o métodos de la clase o componente: Crypto.Cipher, que corresponde al paquete o librería: PyCryptodome [48], del lenguaje Python [43]. Una aproximación a este tipo de implementación, puede ser la siguiente: *from Crypto.Cipher import DES ; textoplano = "▒Welcome▒" ; Clave56bits="00000001" ; Ci = ((DES.new(Clave56bits.encode("utf-8"), DES.MODE_ECB)).encrypt(textoplano.encode("utf-8") + (b"\x00" * (8-len(textoplano.encode("utf-8")) % 8)))).hex(); print(Ci)*.

Del mismo modo, los parámetros empleados en el cifrado de datos con el algoritmo 3DES (TripleDES), consisten en el uso de una clave secreta de 24 bits, utilizando el valor: "000000000000000000000001". También, se ha empleado el modo de libro de código electrónico, conocido como: ECB (Electronic Codebook Mode) [15], usando OpenSSL [16]-[17], como "default_backend", generando resultados de texto cifrado en formato hexadecimal. La implementación de 3DES, también fue realizada en lenguaje Python, como sigue: *from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes; from cryptography.hazmat.backends import default_backend ; textoplano = "▒Welcome▒".encode() ; Clave24bits=b"000000000000000000000001" ; Cifrado = Cipher(algorithms.TripleDES(Clave24bits), mode=modes.ECB(), backend=default_backend()); Ci = Cifrado.encryptor() ; Ci = (Ci.update((textoplano.decode() + "" + str("".join([" " for k in range(0,int(24-len(textoplano)))]))).encode()) + Ci.finalize()).hex() ; print(Ci)* . Lo anterior, haciendo uso del conjunto de métodos o funciones: Cipher, importando como argumento el componente de clase: algorithms.TripleDES, que corresponde a la librería o paquete: Cryptography [93].

En lo concerniente con el cifrado de datos usando: Blowfish, ha sido empleada una clave secreta de 16 bits, con el siguiente valor: "00000001". También, fue utilizado: OpenSSL como "default_backend", en el formato o modo de libro de código electrónico: ECB (Electronic Codebook Mode) [15], obteniendo resultados cifrados en formato hexadecimal. La implementación de Blowfish, en lenguaje Python, también se llevó a cabo, utilizando el conjunto de métodos o funciones de los componentes de clase: Cipher, que corresponden a la librería o paquete: Cryptography [93]. Lo anterior, importando como parámetro el componente: algorithms.Blowfish, tal como se muestra a continuación: *from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes ; from cryptography.hazmat.backends import default_backend ; textoplano = "▒Welcome▒".encode() ; Clave16bits = b"00000001" ; Ci = Cipher(algorithms.Blowfish(Clave16bits), mode=modes.ECB(), backend=default_backend()).encryptor() ; Ci = (Ci.update((textoplano.decode() + "" + str("".join([" " for k in range(0,int(16-len(textoplano)))]))).encode()) + Ci.finalize()).hex() ; print(Ci)* .

Cabe mencionar que, en los códigos fuente presentados en los algoritmos: 3DES y Blowfish, muestran implementación de relleno usando el caracter espacio. Sin embargo, en la experimentación se ha utilizado relleno aleatorio con reemplazo, en formato hexadecimal. Por ejemplo: *import random; N=16; R = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "a", "b", "c", "d", "e", "f"]; textoplano = "▒Welcome▒" ; textoplano = textoplano + str("".join([(""+str(R[random.randrange(16)])) for k in range(0,int(N-len(textoplano)))])) ; print(textoplano)* .

Por otra parte, en lo que refiere a los parámetros empleados para el cifrado de datos con el algoritmo: AES-256, se ha utilizado una clave secreta (Key) con longitud (KeySize) de 256 bits y un valor de "0000000000000000000000000000001" lo cual, corresponde al uso de 32 bytes. Además, se hizo uso del formato o modo de encadenamiento de bloque de cifrado: CBC (Cipher Block Chaining Mode) [15], con OpenSSL [16]-[17], como "default_backend" y usando un vector de inicialización: IV= "0000000000000001" con 128 bits, empleando el estándar de criptografía de llave pública: PKCS7 (Public Key Cryptography Standard #7) [16]-[17], como método de relleno (padding). El resultado cifrado fue obtenido en formato hexadecimal. Este algoritmo: AES-256, fue implementado, también en lenguaje Python [43], empleando el paquete o librería: Cryptography [93], haciendo uso del conjunto de métodos de la clase: Cipher, importando el componente: algorithms.AES. Un ejemplo de este tipo de implementación, se muestra enseguida: *from cryptography.hazmat.primitives import padding ; from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes ; from cryptography.hazmat.backends import default_backend ; textoplano = "▒Welcome▒".encode() ; IV= b"0000000000000001" ; Key = b"0000000000000000000000000000001" ; KeySize=256 ; Ci = Cipher(algorithms.AES(Key), modes.CBC(IV), backend = default_backend()).encryptor(); Ri = padding.PKCS7(KeySize).padder(); Ri = Ri.update(textoplano) + Ri.finalize(); Ci = (IV + (Ci.update(Ri) + Ci.finalize())).hex() ; print(Ci)* .

Con respecto al uso de GOST, se utilizó el algoritmo: kuznechik, de la versión: GOST R 34.12-2015, para llevar a cabo el cifrado de datos, siguiendo el mismo camino presentado en [19]. Por lo tanto, *« los parámetros utilizados para la encriptación con el algoritmo GOST R 34.12-2015, se ha empleado una clave secreta de 16 bytes o 128 bits, siendo asignado el valor: "0000000000000000000000000000001", utilizando el valor "kuznechik" en la función "gostcrypto.gostcipher.new" de la librería "gostcrypto" [47], de lenguaje Python, para definir el estándar: Kuznyechik (Кузнечик). También, ha sido empleado un modo de operación ECB (Electronic Codebook Mode) [15], utilizando el valor de "gostcrypto.gostcipher.MODE_ECB", así como, el "Modo 1" para desplazamiento o relleno (padding o pad_mode) utilizando el valor "gostcrypto.gostcipher.PAD_MODE_1", ambos valores de dichas referencias han sido asignadas por la*

librería "gostcrypto" del lenguaje Python. Los resultados cifrados fueron arrojados en formato hexadecimal » [19]. Una aproximación de código fuente, se ha descrito en esa misma investigación [19], del siguiente modo: *import gostcrypto ; plaintext = "▒Welcome▒" ; key = b"00000000000000000000000000000001" ; Ci = (gostcrypto.gostcipher.new ("kuznechik" , key, gostcrypto.gostcipher.MODE_ECB, pad_mode = gostcrypto.gostcipher.PAD_MODE_1)).encrypt(plaintext.encode()).hex() ; print(Ci)* .

Por último, en lo que concierne con los parámetros que fueron empleados para realizar el cifrado de datos con el algoritmo: Camellia, también, fue utilizado un modo de encadenamiento de bloque de cifrado: CBC (Cipher Block Chaining Mode) [15], con OpenSSL [16]-[17], como "default_backend" y usando un vector de inicialización: IV= "0000000000000001" con 128 bits. Por lo tanto, el tamaño del cifrado por bloques fue de: Camellia.block_size=128, haciendo uso del estándar de criptografía de llave pública: PKCS7 (Public Key Cryptography Standard #7) [16]-[17], utilizado como método de relleno (padding). El valor de la clave secreta con longitud de 256 bits fue: "0000000000000000000000000000001", esta secuencia representa manejo de claves con 32 bytes, que se indican en el código fuente como: "algorithms.Camellia(Key256bits)". Los resultados cifrados, han sido obtenidos en formato hexadecimal. De la misma forma, que se ha descrito anteriormente, la implementación de: Camellia, también fue realizada en lenguaje Python, tal como se muestra enseguida: *from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes ; from cryptography.hazmat.backends import default_backend ; from cryptography.hazmat.primitives import padding ; textoplano = "▒Welcome▒" ; IV= b"0000000000000001" ; Key256bits = b"00000000000000000000000000000001" ; Ci = Cipher (algorithms.Camellia (Key256bits), modes.CBC(IV), backend=default_backend()).encryptor() ; relleno = padding.PKCS7 (algorithms.Camellia.block_size).padder() ; Ci = (IV + (Ci.update (relleno.update (textoplano.encode()) + relleno.finalize()) + Ci.finalize())).hex() ; print(Ci)* . Para dicha implementación, fue necesario hacer uso del conjunto de métodos de la clase: Cipher, importando como argumento el componente: algorithms.Camellia, que corresponde a la librería o paquete: Cryptography [93].

Aunque, los tiempos de procesamiento para el cifrado y descrifrado de datos son mucho más rápidos con el uso del algoritmo estándar: DES, se advierte que esta estrategia no es considerada un esquema seguro. Empero, el mejor balance fue obtenido con: random Caesar II mod 120, cuando ha sido aplicado al texto plano. Con respecto a las estrategias basadas en: random noisy, se ha observado mayor velocidad en el cifrado con el uso de la alternativa: random noisy DES. En este caso, la alternativa "ruidosa aleatoria", no ha sido evaluada, a la fecha, por ningún tipo de ataque, para asegurar vulnerabilidades. Sin embargo, tampoco es recomendable, porque se pudo observar la ocurrencia de errores, al momento del descrifrado de datos hasta de *1%* en promedio. Del mismo modo, el algoritmo: GOST R 34.12-2015, además que su proceso de cifrado/descifrado es más lento que el resto de los algoritmos, también ha reportado un *0.2%* de error en promedio. Situación que, durante los experimentos realizados, se le atribuye a la presencia de caracteres fuera del rango de la tabla ASCII, que han sido inyectados en el texto plano, durante la evaluación del algoritmo, tal es el caso del caracter: "▒" (con ordinal: 9619), así como, el caracter: "" (cuyo ordinal es: 65533).

Por otra parte, se logró cumplir la hipótesis de la presente investigación, debido a que, la nueva propuesta denominada: random noisy Camellia, genera resultados cifrados dinámicos, utilizando la inyección de ruido con IA, siendo capaces de producir distintos resultados, incluso utilizando la misma entrada de texto plano, empleando los mismos parámetros. Está situación puede ser corroborada en los resultados presentados en la Tabla 1, lo cual, cumple con el propósito de confundir a los ciberdelincuentes, en el momento de querer descifrar nuestra información, ya que, para descubrir la posición dónde se encuentra alojado el "ruido", podría constituir una tarea muy difícil de lograr, incluso para la computación cuántica, ya que, se tendría prácticamente que "adivinar", donde fue inyectado ese "ruido", haciendo más difícil la tarea de descubrimiento, si se utiliza la alternativa adicional presentada en [2], la cual, sugiere no inyectar ruido a todo el texto cifrado, siendo definido formalmente como: $RuidoParcial = [\ N\ ,\ Pos_i\ ,\ K_i\ ,\ Texto\text{-}Cifrado_t\ ]$. Donde: Todos los valores, incluyendo los contenidos en los vectores, son camuflajeados como caracteres en formato ASCII. El parámetro N, es el número de localidades que se está inyectando ruido. El valor máximo valor de i es N. El vector Pos_i, son las posiciones del $Texto\text{-}Cifrado_t$, que se ha inyectado ruido, mientras que, el vector K_i es el

desplazamiento para cada Pos_i del $Texto\text{-}Cifrado_t$. El máximo valor de t es el tamaño del texto cifrado. A través de esta inyección de ruido en un texto cifrado, puede incrementarse el grado de seguridad, por ejemplo, en una cadena de texto encriptada e incluso estar prevenidos contra futuros ataques de computación cuántica [2], [19], y por ende, puede mejorar la seguridad de datos en las organizaciones.

En lo concerniente con el algoritmo estándar: Camellia, se puede observar que su proceso de cifrado es *1.16* veces más rápido que: AES-256, con una diferencia de *2.21* milisegundos en promedio. Empero, la alternativa "ruidosa aleatoria", denominada: random noisy Camellia, reporta un proceso de cifrado más lento que: random noisy AES, hasta *1.23* veces, con una diferencia de *1.97* milisegundos (ver Tabla 1). Sin embargo, a pesar de estos menesteres, se considera a Camellia, una opción segura para las organizaciones, y el simple hecho de contar con esta nueva alternativa basada en inyección de ruido, ello abre un gran abanico de oportunidades a las organizaciones respecto a su uso, porque garantiza mejoramiento en la seguridad de los datos digitales.

Por último, con respecto a la evaluación de resultados con los algoritmos estándar y las estrategias basadas en: random noisy, empleando la variante de validación cruzada [1]-[2], [7], en la presente investigación, dicha información puede ayudarnos a sugerir, el uso de inyección de ruido con estrategias: random noisy Camellia, como alternativa segura en las organizaciones, sobretodo en aquellas, que se ha adoptado el uso de Camellia tradicional, y para poder incrementar la seguridad de este tipo de criptosistemas, a través del uso de inyección de ruido, aquí recomendada. Finalmente, una modificación del esquema: random noisy Camellia, que podría ser de interés, es la aplicación de estrategias del tipo: reduced noisy, algo muy similar a las presentadas en [2], ya que, permiten reducir hasta ⅓ en espacio de almacenamiento para un texto cifrado con alternativa "ruidosa reducida", haciendo uso de la metodología: reduced random Caesar. Por su puesto, ello involucra una gran variedad de posibilidades que podrían ser cubiertas en un trabajo futuro.

4 Conclusión

A pesar que el uso de métodos de encriptación es considerado un factor de gran influencia para la seguridad digital de los datos en las organizaciones, el simple hecho de cambiar periódicamente la estrategia para el cifrado de datos, no garantiza la seguridad de los mismos. Tal como se mencionó anticipadamente, existen varios estudios que se han diseñado para enfrentar este tipo de problemas, enfocados a la pérdida o robo de datos digitales, como parte de una consecuencia de ciberataques. En recientes investigaciones, antes mencionadas, se ha discutido acerca de estrategias basadas en el cifrado de datos dinámico, como alternativa para salvaguardar la información de ciber-delincuentes. Estos métodos, algoritmos o estrategias, logran el cifrado de datos, empleando métodos aleatorios, y en algunos casos, utilizando la misma inteligencia artificial con métodos heurísticos. Lo anterior, se considera indispensable, porque algunos estudios han revelado que, los algoritmos de cifrado simétricos estándar, a pesar que su procesamiento es muy rápido, la mayoría de ellos, presentan resultados estáticos. Aunque, ya se ha comentado anteriormente, que dicha situación no significa que sean vulnerables estos esquemas. Empero, existen investigaciones donde aseguran que mediante el uso de un diccionario, la fuerza bruta, e incluso, la misma computación cuántica, proveen estrategias que tienen la capacidad de romper este tipo de criptosistemas, debido a que, los resultados cifrados estáticos, no presentan variaciones en el texto encriptado, en cada distinta ejecución. Por lo tanto, esta situación puede comprometer la seguridad de los datos en las organizaciones. En cambio, las alternativas para el cifrado dinámico, son consideradas más seguras para encriptado de los datos digitales, debido a que, las secuencias de cifrado obtenidas, producen distintos resultados en cada ejecución del proceso, incluso cuando es utilizada la misma cadena de texto plano de entrada, aún siendo cifrada con los mismos parámetros (ver Tabla 1). Del mismo modo, ocurre con las propuestas basadas en inyección de ruido: *random noisy*, incrementando la seguridad de los criptosistemas, porque se genera un paquete cifrado de mayor tamaño. Ello lo hace menos vulnerable a ciberataques, en comparación con los resultados arrojados por los algoritmos de cifrado de datos estándar, evaluados en la presente investigación (ver Tabla 1), aunque se tenga que sacrificar un poco el aspecto de espacio de almacenamiento.

En lo que concierne con la aportación de la presente investigación, se introduce el estudio de varias estrategias: random noisy, siendo comparadas con el diseño de una nueva alternativa, basada en el algoritmo: Camellia, que incluye inyección de ruido, razón por la cual, fué denominada: random noisy Camellia. A pesar que no supera la expectativas de velocidad en comparación con: random noisy AES, la diferencia no es muy sustancial (ver Tabla 1). Sin embargo, cabe destacar que la nueva propuesta, aquí denominada como: random noisy Camellia, es de gran relevancia, porque permite dar aportación científica, no solamente en el campo de la criptografía, sino también, como recurso adicional en el contexto de la inteligencia artificial, ya que, se ha presentado como una nueva alternativa para aumentar la seguridad como variante del criptosistema: Camellia. Adicionalmente, la propuesta: random noisy Camellia, abre un abanico de oportunidades a las organizaciones que hacen uso de este tipo de algoritmos, ofreciendo una importante contribución para incrementar su uso. Lo anterior, porque está relacionado con la presente investigación, debido a que, han sido evaluadas distintas alternativas basadas en estrategias: random noisy, que siguiendo el mismo camino, la alternativa aquí propuesta: random noisy Camellia, también puede ser útil para estar prevenidos contra ataques basados en computación cuántica.

Por tal razón, se considera que el propósito de la investigación fue alcanzado. En primer término, porque se hizo una adaptación al procedimiento del estándar: Camellia, siendo combinado con estrategias del tipo: random noisy, muy similares a las presentadas en previas investigaciones, que ya se han descrito anticipadamente. Esta nueva alternativa: random noisy Camellia, es recomendada como nueva estrategia en el cifrado de datos dinámico, puesto que, se observó durante la experimentación, que produce 0% de error, alcanzando una aceptable convergencia, siendo capaz de producir diferentes resultados cifrados, aún cuando sea utilizada la misma entrada de texto plano (ver Tabla 1). Situación que intuye, no solamente poder incrementar la seguridad de los datos, sino también, cabe reiterar, se puede estar prevenidos ante futuros ataques basados en computación cuántica, ya que, la propuesta: random noisy Camellia, aún no se han estudiado vulnerabilidades, por parte de ese tipo de ciberataques.

Por otra parte, con la finalidad de presentar un balance final sobre la investigación realizada, se puede observar en la Tabla 1, que la diferencia promedio de velocidades para el cifrado/descifrado de datos, no es muy sustancial, ya que, no tarda en ningún caso más de un segundo en procesar. Sin embargo, hay que recordar que los textos planos empleados en la experimentación son menores de 255 caracteres. Por lo tanto, se tendría que analizar nuevos experimentos, probablemente empleando archivos de texto con distintos tamaños, incluyendo medidas como: megabyte o gigabyte. Sin embargo, esto podría ser considerado para un trabajo futuro.

En lo concerniente con los algoritmos estándar para el cifrado de datos, que han sido estudiados en la presente investigación, ninguno de ellos, logró reportar cifrados dinámicos. Sin embargo, las propuestas: random noisy, a pesar que demoran más tiempo en procesar, en todos los casos, han mostrado resultados dinámicos (ver columnas: Test 1 y Test 2, de la Tabla 1).

Con respecto a la nueva propuesta: random noisy Camellia, ha mostrado ser *2.08* veces más rápida para realizar el cifrado de datos, en comparación con la alternativa: random noisy GOST mod 120. Empero, los resultados (ver Tabla 1), han reportado que: random noisy Camellia, puede demorar más en su proceso de encriptación, cuando es comparado con las estrategias: random noisy AES, random noisy 3DES, random noisy Blowfish y random noisy DES, mostrando ser la alternativa de Camellia "ruidosa aleatoria", entre *1.23* y *6.38* veces más lento su cifrado de datos. En estos términos, el cifrado con: random noisy Camellia, puede demorarse entre *1.97* y *8.80* milisegundos, en comparación con las alternativas, antes mencionadas, excepto random noisy GOST. Lo anterior, teniendo en cuenta, que las cadenas de texto plano procesadas son de un tamaño máximo de 255 caracteres.

Con relación a las dificultades observadas en el cifrado de datos dinámico, una de ellas, es que puede llegar a seleccionar valores fuera del rango de la tabla ASCII o algunos caracteres que no son imprimibles en pantalla, ello puede producir pérdida de información y reportar errores al momento del descrifrado de datos, lo cual, constituye una desventaja. Sin embargo, en el presente trabajo, solamente las propuestas basadas en algoritmos: DES y GOST, han mostrado ser vulnerables a este tipo de situación, debido a que, hubo ocurrencia de error, al momento del descrifrado de datos (ver Tabla 1).

En este mismo contexto, los resultados experimentales con las estrategias: random noisy, han revelado la capacidad de hacer frente al problema de robo digital de datos, y por ende, puede amortiguar un poco los ciberataques en este respecto, incluso estar prevenidos ante futuros ataques basados en computación cuántica. Por lo tanto, se ha considerado un buen indicador para la seguridad de datos digitales en las organizaciones, sobretodo, cuando es aplicado este esquema, de forma adicional, con la alternativa parcialmente "ruidosa", la cual, ha sido aquí denominada como: *RuidoParcial* . Lo anterior, resulta más efectivo, cuando es empleado, tomando como entrada el texto cifrado por alguna alternativa basada en: random noisy.

En general, todos los métodos de cifrado de datos, aquí estudiados, reportan buen margen de acierto, excepto los basados en algoritmos: DES y GOST. Del mismo modo, la velocidad de encriptación entre las estrategias aquí mismo evaluadas, en promedio, difieren aproximadamente entre 11.31 y 20.11 milisegundos, para el caso de los métodos basados en: random noisy, mientras que, los algoritmos de encriptado estándar, difieren entre 13.58 y 20.12 milisegundos, siendo evaluados en conjunto. Por lo tanto, se puede deducir que, en términos generales, el uso de los algoritmos estándar, comparado con las estrategias: random noisy, su diferencia de velocidad de encriptación, no es muy sustancial, cuando son evaluados en conjunto. Sin embargo, los resultados reportados por: random Caesar II mod 120, superan las expectativas de velocidad y ahorro de espacio de almacenamiento, en comparación con el resto de las estrategias aquí experimentadas. Empero, la inyección de ruido no se aprecia con mayor proyección (ver Tabla 1).

Por último, en relación con la inyección de ruido en textos cifrados, aún pretendemos promover dichas temáticas. Una de las estrategias que podrían ser consideradas a explorar en dicho contexto, es precisamente, el uso de estrategias del tipo: reduced noisy, presentadas en [2], pero siendo aplicable al criptosistema de Camellia. Lo anterior, con el propósito de indagar, si es posible reducir un poco el espacio de almacenamiento de los resultados cifrados con la inyección de ruido basada en Camellia. Por supuesto, ello involucra una gran variedad de posibilidades, las cuales, podrían ser consideradas en un trabajo futuro.

Agradecimientos

Este trabajo fue soportado parcialmente por el Instituto Tecnológico de Ciudad Altamirano, campus del Tecnológico Nacional de México.

Referencias

[1] Rangel, E., & Rangel, KU. (2024a). Novel Random Encryption Methods Based On Mutation Strategies Of Artificial Intelligence. SPCSJ: Scientific and Practical Cyber Security Journal (Published by Scientific Cyber Security Association in Tbilisi, Georgia), ISSN: 2587-4667, vol. 8, no. 3, pp. 84-91, september issue. Recuperado de: https://journal.scsa.ge/ issue/september-2024/, (05/11/2024).

[2] Rangel, E., Rangel, KU., & González, L. (2025). Dynamic Encryption Methods Based On Noisy Injection And Camouflaging Ciphertext Strategies With Artificial Intelligence. SPCSJ: Scientific and Practical Cyber Security Journal (Published by Scientific Cyber Security Association in Tbilisi, Georgia), ISSN: 2587–4667, vol. 9, no. 1, pp. 82-104, march issue. Recuperado de: https://journal.scsa.ge/papers/dynamic-encryption-methods-based-on-noisy-injection-and-camoufla ging-ciphertext-strategies-with-artificial-intelligence/, (23/04/2025).

[3] Seh AH., Zarour M., Alenezi M., Sarkar AK., Agrawal A., Kumar R., & Khan RA. (2020). Healthcare Data Breaches: Insights and Implications. Healthcare (Basel). May 13;8(2):133. DOI:10.3390/healthcare8020133. PMID: 32414183; PMCID: PMC7349636.

[4] Simiyu, J., Rambim, D., & Ondulo, J. (2024). System Survivability Threats And Factors Influencing Attacks In Health Facilities. Scientific and Practical Cyber Security Journal (SPCSJ), 8(2): 68-75. ISSN: 2587-4667. Scientific Cyber Security Association (SCSA).Tbilisi, Georgia. Recuperado de: https://journal.scsa.ge/issues-archive/, (23/03/2025).

[5] Drzani, M. (2014). Securing E-learning platforms. 2014 International Conference on Web and Open Access to Learning (ICWOAL), Dubai, United Arab Emirates, 1-4. Recuperado de: https://doi.org/10.1109/ICWOAL.2014.7009237, (23/03/2025).

[6] Mamman, J., Onoja, O., & Blessing, E. (2024). Mitigating The Impact Of Phishing Attacks On The E-learning Infrastructure". Scientific and Practical Cyber Security Journal (SPCSJ), 8(3): 21-34. ISSN: 2587-4667. Scientific Cyber Security Association (SCSA). Tbilisi, Georgia. Recuperado de: https://journal.scsa.ge/issues-archive/, (23/03/2025).

[7] Rangel, E., & Rangel, KU. (2024b). 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. 1, 2 (Dec. 2024), 1–15. Editado por IBERAMIA (Iberoamerican Society of Artificial Intelligence: Sociedad Iberoamericana de Inteligencia Artificial). ISSN: 3020-7444. España. Recuperado de: https://inteletica.iberamia.org/index.php/journal/article/view/16, (23/03/2025).

[8] Rangel, E., Rangel, KU., & González, L. (2024). Cifrado De Datos Dinámico Con Inteligencia Artificial, Utilizando El Nuevo Formato Pseudo-Hexadecimal. Revista Electrónica de Divulgación de la Investigación del SABES (Revista de la universidad del SABES), vol. 28, edición diciembre 2024. ISSN: 2007-3542. Editada por Sistema Avanzado de Bachillerato y Educación Superior en el Estado de Guanajuato. México. Recuperado de: https://sabes.edu.mx/revista-electronica/27/# , (17/12/2024).

[9] Barranco, F., & Galindo, C. (2022). Criptografía básica y algunas aplicaciones. Universidad Jaume I, Departamento de Matemáticas, Castellón, España, octubre, 2022. URI: http://hdl.handle.net/10234/ 201359. Recuperado de: https://repositori.uji.es/items/35da2f29-ee4a-4dbc-a82f-c450a81cf9be, (13/04/2025).

[10] Centellas, LS., Blanco, L., & Sandoval, JP. (2022). Estudio comparativo de los algoritmos de criptografía simétrica AES, 3DES y ChaCha20. Revista ActaNova (ISSN: 1683-0768, e-ISSN: 1683-0789), vol. 10, no. 3, Epub 31-Mar-2022, Cochabamba, marzo. Recuperado de: http://www.scielo.org.bo/scielo.php?script=sci_arttext&pid=S1683-07892022000100283, (13/04/2025). Recuperado de: https://dialnet.unirioja.es/servlet/articulo?codigo=10071717, (16/04/2025).

[11] Gómez, H., & Díaz, J. (2021). Análisis de algoritmos de cifrado simétricos y asimétricos para la protección de datos. Revista de Ingeniería y Tecnología, 20(2), 1-12.

[12] Gómez, S., Arias, JD., & Agudelo, D. (2012). Cripto-Análisis Sobre Métodos Clásicos De Cifrado. Scientia Et Technica, ISSN: 0122-1701 (Universidad Tecnológica de Pereira Pereira, Colombia), Año: XVII, vol. 2, no. 50, pp. 97-102, abril. DOI: https://doi.org/10.22517/23447214.6681. Recuperado de: https://revistas.utp.edu.co/index.php/revistaciencia/article/view/6681, (16/04/2025).

[13] Mendoza, JC. (2008). Demostración De Cifrado Simetrico Y Asimétrico. Ingenius. Revista de Ciencia y Tecnología, núm. 3, pp. 46-53. Universidad Politécnica Salesian. Cuenca, Ecuador. ISSN: 1390-650X. Recuperado de: http://www.redalyc.org/articulo.oa?id=505554806007, (09/11/2024).

[14] Oppliger, R. (2005). Contemporary cryptography, (1ra. ed.). Artech House Computer Security Library, Boston/London (ISBN: 1-58053-642-5, 978-8189265038), 510p. Recuperado de: https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://theswissbay.ch/pdf/Gentoomen%2520Library/Cryptography/Contemporary%2520Cryptography%2520-%2520Rolf%2520Oppliger.pdf&ved=2ahUKEwiT1tm1xNWMAxUAmO4BHWYLNa0QFnoECBoQAQ&usg=AOvVaw3GvxLI6QzJhvEXqcl9efPz, (13/04/2025).

[15] Stinson, DR., & Paterson, MB. (2019). Cryptography: Theory and Practice, (4ta ed.). Chapman and Hall Book/CRC Press (Taylor & Francis Group, ISBN: 978-1-1381-9701-5). Recuperado de: https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.ic.unicamp.br/~rdahab

/cursos/mo421-mc889/ Welcome_files/ Stinson-Paterson_CryptographyTheoryAndPractice-CRC%2520Press%2520%25282019%2529.pdf&ve d=2ahUKEwi8_euUtdWMAxWzh-4BHShGAaQQFnoECCUQAQ&usg=AOvVaw1DCBGkDbaZMeHuyR45xZZC, (13/04/2025).

[16] Van, HC., & Jajodia, S. (2011). Encyclopedia Of Cryptography And Security. Springer Science & Business Media, 2011. 1416p. ISBN: 978-14419-5907-2.

[17] Van-Tilborg, HCA. (2005). Encyclopedia Of Cryptography And Security. pp. 114-115, 201-202. TUE Research portal. Springer. DOI: https://doi.org/10.1007/0-387-23483-7, (09/11/2024).

[18] Rangel, E., Rangel, KU., González, L., Ortiz, A., & Rodríguez, CA. (en revisión, 2025a). Four Dynamic Encryption Alternatives With Artificial Intelligence Based On Pseudo-Hexadecimal Noisy Injection Schema For Handling The Theft Of Digital Data Problem. Enviado a: Scientific and Practical Cyber Security Journal (SPCSJ), Vol. 9, No. 2, June Issue. ISSN: 2587-4667 (electronic). Edited by SCSA - SPCSJ - BOAI. Published by Scientific Cyber Security Association in Georgia. Tbilisi, Georgia. URL: https://journal.scsa.ge/papers/.

[19] Rangel, E., Rangel, KU., & González, L. (en revisión, 2025b). Inyección De Ruido Para Encriptado De Datos Dinámico Con Inteligencia Artificial. Caso De Estudio: Algoritmo GOST R 34.12-2015. Enviado a: Revista Electrónica de Divulgación de la Investigación del SABES (Revista de la universidad del SABES), vol. 29, edición junio — 2025. ISSN: 2007-3542. Editada por Sistema Avanzado de Bachillerato y Educación Superior en el Estado de Guanajuato. México. URL: https://sabes.edu.mx/revista-electronica/.

[20] Baklaga, L. (2024). Leading The Way In Quantum-Resistant Cryptography For Everyday Safety. Scientific and Practical Cyber Security Journal (SPCSJ), 8(3): 65-73. ISSN: 2587-4667. Scientific Cyber Security Association (SCSA).Tbilisi, Georgia. Recuperado de: https://journal.scsa.ge/papers/leading-the-way-in-quantum-resistant-cryptography-for-everyday-safety/, (23/03/2025).

[21] Bavdekar, R., Eashan-Jayant C., Ankit A., & Tiwari, K. (2023). Post Quantum Cryptography: A Review of Techniques, Challenges, and Standardizations. In 2023 International Conference on Information Networking (ICOIN).

[22] Dhany, HW., Izhari, F., Fahmi, H.,Tulus, M., & Sutarman, M. (2018). Encryption and Decryption using Password Based Encryption, MD5, and DES. Published by Atlantis Press. ISSN: 2352-5398. Open Access: CC BY-NC license: http://creativecommons.org/licenses/by-nc/4.0/.

[23] Kumar, A., & Sharma, S. (2021). A Study on Historical Cryptographic Techniques: Caesar Cipher to DES. International Journal of Advanced Science and Technology, 30(2), 555-564.

[24] Rodríguez, J. (2020). Operadores Genéticos Aplicados A La Criptografía Simétrica. Proyecto De Grado. Universidad Distrital Francisco José De Caldas. Facultad De Ingeniería. Ingeniería De Sistemas. Bogotá, Colombia. Recuperado de: https://repository.udistrital.edu.co/handle/11349/28192, (06/05/2024).

[25] Schneier, B. (1994). Description of a new variable-length key, 64-bit block cipher (Blowfish). In Fast Software Encryption.

[26] Schneier, B. (2000). Secrets and lies: Digital security in a networked world. Wiley.

[27] Daemen, J., & Rijmen, V. (2002). The Design of Rijndael: AES - The Advanced Encryption Standard. Springer. DOI:10.1007/978-3-662-04722-4, (23/03/2025).

[28] NIST. (2001). FIPS PUB 197: Advanced Encryption Standard (AES). U.S. Department of Commerce.

[29] Rahman, MS., & Hossain, MS. (2021). A Secure Private Key Cryptography Scheme Using RSA and AES. Journal of Cybersecurity, 1(1), 1-9.

[30] Dunkelman, O., Keller, N., & Weizmann, A. (2023). Practical-Time Related-Key Attack on GOST with Secret S-boxes. In: Handschuh, H., Lysyanskaya, A. (eds) Advances in Cryptology - CRYPTO 2023. CRYPTO 2023. Lecture Notes in Computer Science (Annual International Cryptology Conference. Publisher by Springer, Cham., eBook Packages: Computer Science R0), ISBN: 978-3-031-38547-6,

e-ISBN: 978-3-031-38548-3, vol. 14083, no. 1, pp. 177-208, august, 2023, DOI: https://doi.org/10.1007/978-3-031-38548-3_7.

[31] Ishchukova, E., Maro, E., & Pristalov, P. (2020). Algebraic Analysis of a Simplified Encryption Algorithm GOST R 34.12-2015. Computation, 8(2), 51. https://doi.org/10.3390/computation8020051. Recuperado de: https://www.mdpi.com/2079-3197/8/2/51, (06/06/2024).

[32] Sami Sulaiman, A., & M. Hammood, M. (2025). Enhancing AES Security based on Camellia Key Schedule. Samarra Journal of Pure and Applied Science (e-ISSN: 2789-6838), 7(1), march issue, pp. 299–309. DOI: https://doi.org/10.54153/sjpas.2025.v7i1.1020. Recuperado de: https://sjpas.com/index.php/sjpas/article/view/1020/400, (30/03/2025).

[33] Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., & Tokita, T. (2000). Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms—Design and Analysis. Proceedings of the Seventh Annual Workshop on Selected Areas in Cryptography (SAC 2000), Springer-Verlag, LNCS vol. 2012, August Issue, eds. D.R. Stinson and S.E. Tavares. Springer-Verlag, Berlin, pp. 39–56.

[34] Alkhwaja, I., Albugami, M., Alkhwaja, A., Alghamdi, M., Abahussain, H., Alfawaz, F., Almurayh, A., & Min-Allah, N. (2023). Password Cracking with Brute Force Algorithm and Dictionary Attack Using Parallel Programming. Applied Sciences 13 (10), 5979. E-ISSN: 2076-3417. DOI: https://doi.org/10.3390/app13105979, may., 2023. Recuperado de: https://www.mdpi.com/2076-3417/13/10/5979, (14/04/2025).

[35] Bošnjak, L., Sres, J., & Brumen, B. (2018). Brute-force and dictionary attack on hashed real-world passwords. Conference: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). DOI: 10.23919/MIPRO.2018.8400211. Recuperado de: https://www.researchgate.net/publication/326700354_Brute-force_and_dictionary_attack_on_hashed_real-world_passwords, (23/05/2025).

[36] Rangel, E., Rangel, KU., Medrano, J., Bernal, CA., & González, L. (2023). Algoritmo Genético Para Cifrado De Datos, Basado En Un Nuevo Concepto Pseudo-Hexadecimal Con Inteligencia Artificial. Tecnológico Nacional De México, Instituto Tecnológico de Cd. Altamirano. Sexto (VI) Congreso Nacional De Investigación En Ciencia E Innovación De Tecnologías Productivas. Noviembre, 2023. Cd. Altamirano, Estado De Guerrero, México. Recuperado de: https://www.cdaltamirano.tecnm.mx/index.php/17-vi-congreso-nacional-de-investigacion-en-ciencia-e-innovacion-de-tecnologias-productivas/140-tecnm-40 , (23/03/2024).

[37] Courtois, NT. (2012). Security Evaluation Of GOST 28147-89. In: View Of International Standardisation. Cryptologia, Vol. 36, no. 1, 2012, pp. 2-13.

[38] Fulgueira, M., Hernández, OA., & Henry, V. (2015). Paralelización Del Algoritmo Criptográfico GOST Empleando El Paradigma De Memoria Compartida. Lámpsakos, núm. 14, pp. 18-24. Fundación Universitaria Luis Amigó Medellín, Colombia. E-ISSN: 2145-4086; julio-diciembre. DOI: http://dx.doi.org/10.21501/21454086.1633. Recuperado de: http://www.redalyc.org/articulo.oa?id=613965326004, (09/11/2024).

[39] Clark, A. (1994). Modern optimisation algorithms for cryptanalysis. In Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems, November 29 December 2, (pp. 258-262).

[40] Luciano, D., & Prichett, G. (1987). Cryptology: From Caesar Ciphers To Public-key Cryptosystems. The College Mathematics Journal, vol 18 pp 2-17. Recuperado de: http://www.jstor.org/stable/2686311, (06/06/2025).

[41] Rivest, RL., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126. Recuperado de: https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://people.csail.mit.edu/rivest/Rsapaper.pdf&ved=2ahUKEwjg2fiM2qGMAxXbhu4BHe9iCSsQFnoECDUQAQ&us

g=AOvVaw1FtI2T7P32pKAb6jnZSqxi, (23/03/2025).

[42] Iavich, M., Kuchukhidze, T., & Gagnidze, A. 2024. Post-quantum Digital Signature Using Verkle Trees And Lattices. Scientific and Practical Cyber Security Journal (SPCSJ), 8(3): 35-52. ISSN: 2587-4667. Scientific Cyber Security Association (SCSA).Tbilisi, Georgia. Recuperado de: https://journal.scsa.ge/issues-archive/, (23/03/2025).

[43] Python.org (2024). The Python Network. Python.org. Recuperado de: https://www.python.org/downloads/, (18/11/2024).

[44] Fuegner, P. (2024). Are RSA and AES Both at Risk From the Quantum Threat?. QuSecure, Inc. Recuperado de: https://www.qusecure.com/are-rsa-and-aes-both-at-risk-from-the-quantum-threat/#:~:text=The%20emergence%20of%20quantum%20computers,efficiently%20factoring%20large%20prime%20numbers, (2025-02-08).

[45] Sengupta, S., & Ghosh, S. (2023). Quantum Computing Encryption Threats: Why RSA and AES Are at Risk. Journal of Cryptographic Research.

[46] ECIESPy (2025). ECIESPy 0.4.4. Python Software Foundation. Recuperado de: https://pypi.org/project/eciespy/, (29/03/2025).

[47] Gostcrypto (2021). Gostcrypto 1.2.5. Python Software Foundation. Recuperado de: https://pypi.org/project/gostcrypto/, (06/06/2025).

[48] PyCryptodome (2025). Crypto.Cipher package. Introduction. Recuperado de: https://pycryptodome.readthedocs.io/en/latest/src/cipher/cipher.html, (30/03/2025).

[49] PyPI (2024). Pycryptodome 3.21.0. Python Software Foundation. Recuperado de: https://pypi.org/project/pycryptodome/, (13/12/2024).

[50] Goodman, J. (1996). Introduction To Fourier Optics. (2da. ed.). New York: McGraw-Hill. ISBN: 0-07-024254-2, 441p, . Recuperado de: https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.hlevkin.com/hlevkin/90MathPhysBioBooks/Physics/Physics/Mix/Introduction%2520to%2520Fourier%2520Optics.pdf&ved=2ahUKEwjb7r_d49aMAxVx4MkDHYrxDhsQFnoECCEQAQ&usg=AOvVaw3kMFxGyY-jUJZgWsFvFC54, (13/04/2025).

[51] Javidi, B., & Horner, JL. (1994). Optical Pattern Recognition for Validation and Security Verification. Optical Engineering (ISSN: 0091-3286), vol. 33, issue: 6, pp. 1752-1756, june, 1994, DOI: https://doi.org/10.1117/12.170736. Recuperado de: https://www.spiedigitallibrary.org/journals/optical-engineering/volume-33/issue-6/00 00/Optical-pattern-recognition-for-validation-and-security-verification/10.1117/12.170736.short, (14/ 04/2025).

[52] Khalil, M., Chan, A., Plant, DV., Chen, LR., & Kuang, R. (2024). Experimental demonstration of quantum encryption in phase space with displacement operator in coherent optical communications. EPJ Quantum Technology, SpringerOpen Journal (ISSN: 2196-0763), vol. 11, no. 49, pp. 1-12, july issue. DOI: https://doi.org/10.1140/epjqt/s40507-024-00260-y. Recuperado de: https://epjquantumtechnology. springeropen.com/articles/10.1140/epjqt/s40507-024-00260-y#citeas, (14/04/2025).

[53] Linfei, C., & Daomu, Z. (2005). Optical Image Encryption Based On Fractional Wavelet Transform. Optics Communications (ISSN: 0030-4018), vol. 254, issue: 4-6, pp. 361-367, october issue. DOI: 10.1016/j.optcom.2005.05.052. Recuperado de: https://ui.adsabs.harvard.edu/abs/2005OptCo.25 4..361C/abstract, (14/04/2025).

[54] Rueda, AS., & Lasprilla, M. (2002). Encriptación Por Conjugación De Fase En Un BSO Utilizando Señales Ópticas De Baja Potencia. Rev. Col. Fís., Vol. 34, No.2, P.P.636-640.

[55] Rueda, JE., Romero, AL., & Castro, LM. (2005). Criptografía Digital Basada En Tecnología Óptica. Bistua: Revista de la Facultad de Ciencias Básicas (Universidad de Pamplona, Colombia, ISSN: 0120-4211), vol. 3, no. 2, pp. 19-25, july, 2005, Recuperado de: http://www.redalyc.org/articulo.oa?id=90330203, (12/04/2025).

[56] Wang, X., & Chen, X. (2022). Optical image encryption based on compressive sensing and double random phase encoding. Journal of Optics, 24(2), 025401.

[57] Chang, C.-C., Hwang, M.-S., & Chen, T.-S. (2001). A new encryption algorithm for image cryptosystems. Journal of Systems and Software (ISSN: 0164-1212, e-ISSN: 1873-1228), vol. 58, issue: 2, pp. 83-91, september. DOI: https://doi.org/10.1016/S0164-1212(01)00029-2. Recuperado de: https://www.sciencedirect.com/science/article/pii/S0164121201000292, (16/04/2025).

[58] Chong, F., Bib, L., Yu, SM., Xiao, L., & Jun-jie, C. (2011). A novel chaos-based bit-level permutation scheme for digital image encryption. Optics Communications (ISSN: 0030-4018), vol. 284, issue: 23, pp. 5415-5423, november, issue. DOI: https://doi.org/10.1016/j.optcom.2011.08.013. Recuperado de: https://www.sciencedirect.com/science/article/pii/S0030401811008431, (15/04/2025).

[59] François, M., Grosges, T., Barchiesi, D., & Erra, R. (2012). A new image encryption scheme based on a chaotic function. Signal Processing: Image Communication (e-ISSN:1879-2677), vol. 27, issue: 3, pp. 249-259, march. DOI: https://doi.org/10.1016/j.image.2011.11.003. Recuperado de: https://www.sciencedirect.com/science/article/abs/pii/S0923596511001354?via%3Dihub, (16/04/2025).

[60] Gan, WS. (2014). Techniques in the Application of Chaos Theory in Signal and Image Processing. Control and Dynamic Systems (ISSN: 0090-5267), vol. 77:1996, pp. 339-387, ed. Cornelius T. Leondes, Academic Press (ISBN: 9780120127771), june issue. DOI: https://doi.org/10.1016/S0090-5267(96)80034-X. Recuperado de: https://www.sciencedirect.com/science/article/pii/S009052679680034X, (15/04/2025).

[61] Gong, M., Chai, X., Lu, Y., & Zhang, Y. (2024). Exploiting Four-Dimensional Chaotic Systems With Dissipation and Optimized Logical Operations for Secure Image Compression and Encryption. In IEEE Transactions on Circuits and Systems for Video Technology (ISSN: 1051-8215), vol. 34, no. 8, pp. 7628-7642, august issue. DOI: 10.1109/TCSVT.2024.3375868. Recuperado de: https://ieeexplore.ieee.org/document/10466550, (16/04/2025).

[62] Hossam, EHA., Hamdy, K., & Osama, SFA. (2007). An Efficient Chaos-Based Feedback Stream Cipher (ECBFSC) for Image Encryption and Decryption. Informática (ISSN: 0350-5596), vol. 31, no. 1, pp. 121-129, march issue. Recuperado de: https://scholar.google.com.mx/scholar?hl=es&as_sdt=0%2C5&as_vis=1&q=An+Efficient+Chaos-Based+Feedback+Stream+Cipher+%28ECBFSC%29+For+Image+Encryption+And+Decryption&btnG=#d=gs_qabs&t=1744811219149&u=%23p%3D--4HB7oG_14J, (16/04/2025).

[63] Ji, Z., Chang, J., Huang, Y., Wu, Y., Cao, J., Zhang, J., & Jiang, H. (2023). Multi-key optical encryption based on two-channel incoherent scattering imaging. Opt. Express (ISSN:1094-4087), vol. 31, issue: 13, pp. 21507-21520, june. DOI: https://doi.org/10.1364/OE.491958. Recuperado de: https://opg.optica.org/oe/fulltext.cfm?uri=oe-31-13-21507&id=531451#articleMetrics, (14/04/2025).

[64] Jiménez, M., Flores, O., & González, MG. (2015). Sistema para codificar información implementando varias órbitas caóticas. Ingeniería, Investigación y Tecnología (Universidad Nacional Autónoma de México, ISSN: 1405-7743), vol. 16, no. 3, pp. 335-343, june. DOI: https://doi.org/10.1016/j.riit.2015.05.004. Recuperado de: https://www.sciencedirect.com/science/article/pii/S1405774315000220. URI: http://www.redalyc.org/articulo.oa?id=40440683002, (16/04/2025).

[65] Kuo, CJ. (1993). Novel image encryption technique and its application in progressive transmission. Journal of Electronic Imaging (ISSN: 1017-9909), vol. 2, issue: 4, pp. 345-351, october. DOI: https://doi.org/10.1117/12.148572. Recuperado de: https://www.spiedigitallibrary.org/journals/journal-of-electronic-imaging/volume-2/issue-4/0000/Novel-image-encryption-technique-and-its-application-in-progressive-transmission/10.1117/12.148572.short, (16/04/2025).

[66] Paul, S., Dasgupta, P., Naskar, PK., & Chaudhuri, A. (2017). Secured image encryption scheme based on DNA encoding and chaotic map. Review Of Computer Engineering Studies, IIETA: International Information and Engineering Technology Association. ISSN: 2369-0755, e-ISSN: 2369-0763, vol. 4, no. 2, pp. 70-75, june issue. DOI: 10.18280/rces.040206. Recuperado de: https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://iieta.org/sites/default/files/Journals/RCES/04.02_06.pdf&ved=2ahUKEwjVlYfordOMAxVVLUQIHXknBWsQFnoECB4QAQ&usg=AOvVaw3yvem4PsuKbMM9009owuAJ, (12/04/2015).

[67] Pisarchik, AN., & Flores-Carmona, NJ. (2006). Computer Algorithms For Direct Encryption And Decryption Of Digital Images For Secure Communication. Proceeding of the 6th WSEAS International Conference On Applied Computer Science (Canary Islands, Spain), pp. 29-34.

[68] Pisarchik, AN., & Zanin, M. (2008). Imagen Encryption Witch Chaotically Coupled Chaotic Maps. Elsevier Physica, abril [en línea], D 237. Recuperado de: www.elsevier.com/locate/physd.

[69] Rajan, B., & Saumitr, P.A. (2006). Novel Compression And Encryption Scheme Using Variable Model Arithmetic Coding And Coupled Chaotic System. IEEE Transactions on circuits and system- I, april, vol. 53 (No. 4). Recuperado de: https://www.researchgate.net/publication/3451216_A_novel_compression_and_encryption_scheme_using_variable_model_arithmetic_coding_and_coupled_chaotic_system, (06/06/2023).

[70] Scharinger, J. (1998). Fast encryption of image data using chaotic Kolmogrov flow. Journal of Electronic Imaging, vol. 7, issue: 2, pp. 318-325, april. DOI: https://doi.org/10.1117/1.482647. Recuperado de: https://www.spiedigitallibrary.org/journals/Journal-of-Electronic-Imaging/volume-7/issue-2/0000/Fast-encryption-of-image-data-using-chaotic-Kolmogorov-flows/10.1117/1.482647.short, (16/04/2025).

[71] Shikder, A., & Nishchal, BK. (2023). Image encryption using binary polarization states of light beam. Scientific Reports (e-ISSN: 2045-2322), vol. (13): 14028,, pp. 1-10, august issue. DOI: https://doi.org/10.1038/s41598-023-41251-w. Recuperado de: https://www.nature.com/articles/ s41598-023-41251-w, (14/04/2025).

[72] Griindlingh, W., & Van-Vuuren, JH. (2002). Using Genetic Algorithms to Break a Simple Cryptographic Cipher. Retrieved March 31, 2003. Recuperado de: http://dip.sun.ac.za/~vuuren/abstracts/abstr_genetic.htm, (06/06/2024).

[73] Iyengar, SS., Kannan, R., & Ganapathi, S. (2021a). Inteligencia artificial y criptografía: Tendencias y desafíos. IEEE Transactions on Emerging Topics in Computing, vol. 9, no. 2, pp. 833-844.

[74] Iyengar, SS., Kannan, R., & Ganapathi, S. (2021b). Análisis de patrones en datos cifrados utilizando inteligencia artificial. Pattern Recognition Letters, 146, pp. 168-175.

[75] Kalsi, S., Kaur, H., & Chang, V. (2018). DNA Cryptography and Deep Learning using Genetic Algorithm with NW algorithm for Key Generation. Convergence of Deep Machine Learning and Nature Inspired Computing Paradigms for Medical Informatics. Image & Signal Processing, in Journal of Medical Systems (e-ISSN: 1573-689X), vol. 42, no. 17, december, 2018. DOI: https://doi.org/10.1007/s10916-017-0851-z.

[76] Rangel, E. (2002). Vecinos Envolventes para Variantes de la Regla del Vecino más Cercano. M.Sc.Thesis [tesis de maestría en ciencias], División de Estudios de Posgrado e Investigación, Instituto Tecnológico de Toluca, Metepec, Estado de México, México, 2002.

[77] Rangel, E. (2022). La Regla De Los k Vecinos Más Cercanos (k-NN) Basada En Distancia De Manhattan (City-Block) Para Mejorar La Clasificación De Patrones. En Quinto (V) Congreso Nacional De Investigación En Ciencia E Innovación De Tecnologías Productivas del Tecnológico Nacional De México campus Instituto Tecnológico de Cd. Altamirano, Estado De Guerrero, México, Noviembre, 2022. Recuperado de: https://erangel.coolpage.biz/pappers/edgarrangel2022.pdf, (23/03/2024).

[78] Reddaiah, B. (2019). A Study on Genetic Algorithms for Cryptography. International Journal of Computer Applications ISSN: 0975-8887 (Department of Computer Applications. Yogi Vemana University Kadapa, A.P, India), vol. 177, no. 28, pp. 1-4, december, 2019. DOI:

http://dx.doi.org/10.5120/ijca2019919509. Recuperado de: https://www.researchgate.net/publication/338012809_A_Study_on_Genetic_Algorithms_for_Cryptography, (23/03/2024).

[79] Kanal, LN. (1963). Statical methods for pattern classification. Philco Rept, originally appeared in T. Harley et al., Semi-automatic imagery screening research study and experimental investigation, Philco Reports, V043-2 and v043-3, Vol. I, sec. 6 and Appendix H, prepared for U.S. Army Electronics Research and Development Lab. Under Contract DA-36-039-sc-90742, March 29.

[80] Ross-Quinlan, J. (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA.

[81] Sánchez, JS., Pla, F., & Ferri, FJ. (1997). Prototype selection for the nearest neighbor rule through proximity graphs. Pattern Recognition Letters 18, 507-513.

[82] Skalak, DB. (1994). Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms. In: Proceedings of the Eleventh International Conference on Machine Learning (ML94). Morgan Kaufmann, pp. 293-301.

[83] Kuncheva, LI., & Jain LC. (1999). Nearest Neighbor Classifier: Simultaneous editing and feature selection. Pattern Recognition Letters, 20, 1149-1156.

[84] Delman, B. (2004). Genetic Algorithms in Cryptography. M.S. thesis (Thesis for the Degree of Master of Science in Computer Engineering), Department of Computer Engineering, Rochester Institute of Technology, RIT Scholar Works, Rochester, New York. Recuperado de: https://scholar.google.com.mx/scholar_url?url=https://repository.rit.edu/cgi/viewcontent.cgi%3Farticle%3D6460%26context%3Dtheses&hl=es&sa=X&ei=cbaZZoadNt246rQPnd604AU&scisig=AFWwaeaMfCM5ORUFQN6DU4LA3aEG&oi=scholarr, (23/03/2024).

[85] Matthews, RAJ. (1993). The use of genetic algorithms in cryptanalysis. Cryptologia, 17(4), 187-201.

[86] Unicode Consortium (2022). The Unicode Standard, Version 14.0. Unicode Consortium. Recuperado de: https://www.unicode.org/consortium/consort.html, (11/11/2024).

[87] American National Standards Institute (1963). American Standard Code for Information Interchange (ASCII). ANSI X3.4-1963. New York: ANSI. Recuperado de: https://www.ascii-code.com/, (11/11/2024).

[88] Monterrubio, E. (2024). Hardware. Con-Ciencia Serrana Boletín Científico de la Escuela Preparatoria Ixtlahuaco 6 (11), pp. 6-8. DOI: https://doi.org/10.29057/ixtlahuaco.v6i11.11967. Recuperado de: https://repository.uaeh.edu.mx/revistas/index.php/ixtlahuaco/article/view/11967, (01/06/2025).

[89] Porras, AA. (2023). Metodologías ágiles para el desarrollo de software. Editorial Universidad Distrital Francisco José de Caldas. Recuperado de: https://books.google.com.mx/books?hl=es&lr=&id=JfXBEAAAQBAJ&oi=fnd&pg=PA12&dq=info:6z3i0_j9d7MJ:scholar.google.com/&ots=YKMck_SoE5&sig=ZO_wnCKjo1bJerRCEBV2jJe2Mcg#v=onepage&q&f=false, (01/06/2025).

[90] Microsoft (2025). Descarga de software. Microsoft. Recuperado de: https://www.microsoft.com/es-mx/software-download, (01/06/2015).

[91] Android 12 (2024). [Sistema operativo para dispositivos móviles]. Google. Recuperado de: https://www.android.com/intl/es_es/android-12/, (01/06/2025).

[92] Pydroid3 versión 7.4_arm64 (2025). [IDE for Python 3. Lenguaje de programación y compilador]. Google Play Store. Recuperado de: https://play.google.com/store/apps/details?id=ru.iiec.pydroid3&hl=en&pli=1, (01/06/2025).

[93] Cryptography (2025). Cryptography 45.0.4. Python Software Foundation. Recuperado de: https://pypi.org/project/cryptography/, (01/06/2025).

[94] Barandela, R., Sánchez, JS., García, V., & Rangel, E. (2003). Strategies for Learning in Class Imbalance Problems. Pattern Recognition, Vol. 36, No. 3, pp. 849-851. Rapid and Brief Comunication

(Pergamon) ISBN: (PII: S0031-3203(02)00257-1.0031-3203/02/). DOI: https://doi.org/10.1016/S0031-3203(02)00257-1 , (01/06/2025).

[95] Barandela, R., Sánchez, JS., & Rangel, E. (2003). Two Modifications of the Decontamination Methodology. IASTED International Conference On Artificial Intelligence and Applications , pp. 391-396, Benalmádena (Spain), ISBN 0-88986-390-3. Recuperado de: https://www.actapress.com/PaperInfo.aspx?PaperID=15031&reason=500, (01/06/2025).

[96] Barandela, R., Rangel, E., Sánchez, JS., & Ferri FJ. (2003). Restricted Decontamination for the Imbalanced Training Sample Problem. 8th Iberoamerican Conference on Pattern Recognition, Havana (Cuba), Progress in Pattern Recognition, Speech and Image Analysis, Lecture Notes in Computer Science 2905 , A. Sanfeliu and J. Ruíz-Shulcloper (eds.), Springer-Verlag, pp. 424-431, ISBN 3-540-20590-X. DOI: https://doi.org/10.1007/978-3-540-24586-5_52. Recuperado de: https://link.springer.com/chapter/10.1007/978-3-540-24586-5_52, (01/06/2025).

[97] Rangel, E., & García, O. (2002). Nearest Centroid Neighbour, An Alternative in the Speech Recognition for the Execution from a Mobile Robot Simulator (Applied to the Imbalanced Training Sample Problem). In: 9th International Congress On Computer Science Research (CIICC 02). October 23-25 2002 - Puebla, México. ISBN: 970-18-8526-0. Recuperado de: https://erangel.coolpage.biz/pappers/p2002.jpg, (01/06/2025).

[98] Bruzzone, L., & Serpico, S.B. 1997. Classification of Imbalanced remote-sensing data by neural networks. Elsevier Science B.V. , 0167-8655, 97. PH S0167-8655 (97) 00109-8. DOI: https://doi.org/10.1016/S0167-8655(97)00109-8. Recuperado de: https://www.sciencedirect.com/science/article/abs/pii/S0167865597001098, (01/06/2025).

[99] Lewis, D., & Catlett, J. (1994). Heterogeneous Uncertainty Sampling for Supervised Learning. Proceedings of the 11th International Conference on Machine Learning, ICML'94 (pp. 148-156), New Brunswick, New Jersey, Morgan Kaufmann. DOI: https://doi.org/10.1016/B978-1-55860-335-6.50026-X. Recuperado de: https://www.sciencedirect.com/science/article/abs/pii/B978155860335650026X, (01/06/2025).