Comparación multi-algorítmica para la generación y el aumento de entropía con caracteres equivalentes de contraseñas: Un análisis Comparativo en R y C++

Anthonny Flores
Escuela de Matemáticas
Universidad de Costa Rica
San José, Costa Rica
Email: anthonnywilliamfloresrojas@gmail.com

Abstracto

El presente trabajo aborda la problemática de la baja seguridad en el uso de contraseñas escritas, las cuales suelen repetirse en múltiples cuentas de un mismo usuario y, en muchos casos, se basan en claves predecibles construidas a partir de diccionarios limitados de caracteres y palabras del idioma base. Estudios recientes, como los datos reportados por LastPass [1], evidencian que estas contraseñas rara vez se cambian, lo que incrementa su vulnerabilidad. A ello se suma la limitada capacidad humana para memorizar cadenas aleatorias a largo plazo [2] [3] [4], lo que plantea la necesidad de encontrar un equilibrio entre contraseñas triviales y largas secuencias aleatorias. En este contexto, se propone el uso de técnicas de mnemotecnia y el principio de semejanza de Gestalt [5] [6] [7] como herramientas para el parafraseamiento de caracteres, sustituyendo símbolos por equivalentes visuales que resulten familiares al ojo humano. Esta estrategia busca incrementar la entropía y robustez de las contraseñas sin sacrificar su memorabilidad, ofreciendo un enfoque intermedio que combina seguridad estadística con facilidad de uso.

I. Motivación.

Uno de los principales problemas que posee el uso de contraseñas escritas es la poca seguridad que estas proveen, ya que en muchos casos estas son las mismas en todas las cuentas relacionadas hacia una sola persona; y no solo esto, sino también se hace uso de claves predecibles utilizando un diccionario limitado de caracteres y palabras, normalmente el idioma base de esta misma. Sumado a ello, datos de LastPass apuntan a que estas claves normalmente nunca o casi nunca se cambian [1]. Esto, y la poca capacidad que tenemos los seres humanos para memorizar cosas aleatorias a largo plazo [2] [3] [4], nos lleva a la necesidad de llegar a un punto medio entre el uso de contraseñas triviales y el uso de largas cadenas de caracteres aleatorios. Con el uso de tecnicas de mnemotecnia, y la facilidad que tenemos para relacionar cosas conocidas con objetos que presenten características similares, formalmente llamado principio de semejanza de la Gestalt [5] [6] [7], podemos utilizar el parafraseamiento de los caracteres de nuestras contraseñas con caracteres parecidos o equivalentes hacia el ojo humano.

A esto anterior se le suma el hecho de que, en nuestra sociedad, no todas las personas están bien integradas con las tecnologías modernas como para utilizar tokens físicos u otras medidas; esto genera que el uso de llaves de seguridad en lugar de contraseñas sea un tema complicado.

II. Metodología.

La conversión algorítmica de cadenas en claves pseudoaleatorias se fundamenta en principios de teoría de la información y en modelos discretos de transformación simbólica [8]. En particular, el análisis se apoya en la consideración de la entropía como métrica cuantitativa del grado de imprevisibilidad o dispersión de los caracteres de una cadena [9]. Esto permite comparar objetivamente diferentes estrategias de sustitución, priorizando aquellas que maximizan la diversidad estadística y reducen patrones repetitivos [10].

Dentro de este contexto, la entropía de Shannon constituye el punto de partida como estimador clásico de incertidumbre en una distribución discreta [11]. Sin embargo, para el análisis se amplía este foco mediante variantes que captan otras propiedades que la entropía de Shannon no refleja adecuadamente, entre ellas: entropía normalizada para medir eficiencia relativa respecto al espacio simbólico disponible [9];

entropía cuadrática para penalizar concentraciones ponderadas de caracteres según pesos asignados [12]; entropía efectiva para evaluar el tamaño real del conjunto activo de símbolos; y entropía mínima como cota inferior basada en la frecuencia dominante [13]. Estas métricas permiten caracterizar no solo la cantidad de incertidumbre generada, sino también su calidad estructural y diversificación.

A nivel de algoritmos, el estudio se apoya en métodos de sustitución iterativa diseñados en C++ y R, cada uno optimizado según la orientación de uso del lenguaje y la exploración del espacio de claves. Esto se desarrolla desde enfoques aleatorios sin memoria, hasta estrategias deterministas basadas en frecuencias dinámicas, pasando por modelos greedy que optimizan localmente la entropía incremental como un algoritmo de discriminación de caracteres. Estos algoritmos se consideran operadores estocásticos o heurísticos sobre un alfabeto expandido definido por un diccionario de equivalencias fijado y elegido.

En el caso de los datos utilizados en las pruebas, estos fueron obtenidos mediante la inteligencia artificial de Microsoft Copilot, generados a partir de prompts estructurados para producir principalmente dos tipos de información: contraseñas formadas por palabras aleatorias y contraseñas que se asemejaban más a una contraseña común, al mezclar símbolos y palabras. De estos dos tipos de datos, únicamente se evaluarán los primeros, ya que son más similares a las palabras simples que una persona podría utilizar como contraseña.

Para la comparación de datos se utilizarán gráficos de vela, ya que estos permiten, en un solo gráfico, denotar la dispersión, la mediana, los valores extremos y la variabilidad de las muestras. De esta manera, se logra visualizar de forma clara las diferencias estadísticas entre los distintos métodos de generación de contraseñas y las métricas de entropía asociadas. Además, los gráficos de vela facilitan la identificación de patrones, anomalías y rangos de confianza, lo que los convierte en una herramienta adecuada para comparar diversidad de la información contenida en cada conjunto de datos.

Sumado a lo anterior, también se tendrán en cuenta los tiempos de respuesta de cada uno de los algoritmos, los cuales estarán representados en una escala logarítmica para poder apreciar de la mejor manera las grandes diferencias entre ellos. Estos, al igual que los datos de las entropías, se representarán en gráficos de vela.

III. Metricas de entropia.

Las métricas de entropía utilizadas en el proyecto cumplen el rol de indicadores cuantitativos para evaluar la fortaleza estadística de las claves generadas. Las métricas consideradas fueron las siguientes:

1) Entropía de Shannon: mide la incertidumbre promedio y sirve como base teórica.

$$H(X)_{\text{shannon}} = -\sum_{n\in X} p_n \log_2 (p_n),$$

con n el número de datos disconjuntos.

2) Entropía normalizada: relaciona la entropía observada con la capacidad máxima del conjunto de símbolos utilizados.

$$H_{\text{norm}}(X) = \frac{H_{\text{shannon}}(X)}{\log_2 |\Omega|},$$

con Ω el tamaño del diccionario.

3) Entropía de densidad: evalúa la contribución de incertidumbre por carácter.

$$H_{\text{dens}}(X) = \frac{H_{\text{shannon}}(X)}{|X|}.$$

4) Entropía efectiva: aproxima el tamaño real del alfabeto activo dentro de la clave generada.

$$H(X)_{\text{efec}} = -|X| \log_2 (|\{n \in X\}|),$$

con n el número de datos disconjuntos.

5) Entropía mínima: captura la vulnerabilidad asociada al símbolo más frecuente.

$$H(X)_{\min} = -\log_2 (p_M),$$

con M el dato disconjunto de mayor frecuencia.

6) Entropía cuadrática (Rényi): incorpora pesos diferenciados para controlar la penalización de caracteres alfabéticos comunes.

$$H(X)_{Rnyi} = -\frac{1}{1-\alpha} \log_2 \left(\sum_{n\in X} p_n^{\alpha} \right),$$

con p_n ponderado al doble para caracteres no alfabéticos y $\alpha = 2$ el peso elegido.

En todas las entropías, el factor p_n denota la probabilidad de ese carácter. En este caso, se utilizó una distribución uniforme sobre los datos, ya que esta

permite otorgar la misma equivalencia a todos los reemplazos por igual.

Estas métricas permiten examinar simultáneamente la dispersión, la diversidad, la eficiencia y la robustez simbólica en el parafraseamiento de las claves.

IV. Algoritmos desarrollados en C++.

Para los algoritmos de C++ se hizo uso de punteros de caracteres constantes globales para la optimización en el uso de memoria, permitiendo múltiples salidas para cada carácter original. Cabe recalcar que tanto el diccionario utilizado en los algoritmos de R como en los de C++ es el mismo. Además, junto al diccionario de equivalencias se empleó un unordered_set [14], para identificar frases consideradas sin valor significativo. Sumado a ello, se desarrolló una función que limpiaba las frases, eliminando espacios vacíos y palabras con menos de tres caracteres. Los métodos utilizados fueron:

1) Generación aleatoria uniforme (aleatory_generation): este algoritmo recorre la frase carácter por carácter y, cuando existe más de un equivalente al carácter, selecciona uno completamente al azar con probabilidad uniforme, produciendo una contraseña rápida de generar pero potencialmente con repeticiones que reducen la entropía en general. Peso: $O(n \cdot m)$

2) Búsqueda discriminada (discriminatory_iteration): avanza ordenadamente por la cadena, eligiendo en la primera posición un equivalente aleatorio y en las siguientes el que menos veces haya aparecido en el segmento ya tratado, garantizando una distribución más uniforme y evitando aumentar en lo posible la concentración de símbolos similares. Peso: $O(n^2 \cdot m)$

3) Iteración aleatoria con umbral (aleatory_iteration): repite el proceso de generación aleatoria uniforme hasta que la entropía de Shannon de la contraseña resultante supere el valor de 3 o se alcance un límite basado en $2^{\text{longitud de la cadena}}$, asegurando un nivel mínimo de entropía incluso en casos de mala suerte inicial. Peso: $O(2^n \cdot n^2 \cdot m)$

4) Búsqueda por entropía local (greedy_function): se empieza probando en cada posición todas las alternativas disponibles y seleccionando aquella que, al incorporarla a la cadena actual, produce el mayor incremento en la entropía de Shannon, logrando una optimización local. Peso: $O(n^2 \cdot m)$

5) Búsqueda de árbol n-ario paralela (nary_password): explora todas las combinaciones posibles de equivalentes mediante un árbol de decisiones recursivo, generando una rama por cada opción en cada posición, evaluando la entropía en las hojas y conservando la mejor contraseña global, con paralelización por hilos para mitigar el costo exponencial del espacio de búsqueda completo. Peso: $O(m^n \cdot n)$

6) Búsqueda híbrida (hybrid_function): combina la evaluación local con una exploración selectiva, eligiendo en cada posición las dos alternativas que más aumentan la entropía de la cadena y continuando recursivamente solo por esas ramas en paralelo. Peso: $O(m^5 \cdot n)$ con $O(2^n \cdot n)$

Donde n es la longitud de la frase (número de caracteres), m es el número máximo de equivalencias por carácter, y k es el número de caracteres originales que tienen equivalencias.

Un punto clave a recalcar es que se está usando el algoritmo Mersenne Twister (MT19937), que permite calcular más aleatoriamente las elecciones aleatorias en C++ [15]. Además, se hizo uso de técnicas de manejo de bytes, funciones lambda y formatos numéricos acortados para optimizar el uso de memoria y la rapidez de los algoritmos.

V. Algoritmos desarrollados en R

Los algoritmos implementados en R operan sobre un diccionario de equivalencias en forma de vector. Para estos algoritmos se priorizo utilizar las propiedades vectoriales del programa, y de igual manera, hacer un buen uso de memoria. Los metodos utilizados fueron:

1) Método Aleatorio (aleatory.generation): recorre la frase carácter por carácter y, cuando existe más de un equivalente al carácter, selecciona uno completamente al azar con probabilidad uniforme, produciendo una contraseña rápida de generar pero potencialmente con repeticiones que reducen la entropía en general. Peso: $O(n \cdot m)$

2) Método Discriminatorio (discriminatory.iteration): avanza ordenadamente por la cadena, eligiendo en cada posición el equivalente menos frecuente según la tabla de frecuencias global de la frase original, garantizando una distribución más equilibrada y evitando en lo posible la concentración de símbolos similares. Peso: $O(n \cdot k \cdot m)$

3) Método Ponderado Inverso (weighted.iteration): recorre la frase carácter por carácter, asignando a cada equivalente una probabilidad inversamente proporcional a su frecuencia acumulada (más 1 para evitar división por cero), y selecciona uno con muestreo ponderado, actualizando dinámicamente las frecuen-

cias para favorecer símbolos no representados. Peso: $O(n \cdot k \cdot m)$

4) Método Greedy de Entropía (greedy.function): se construye la clave paso a paso, probando en cada posición todas las alternativas disponibles y seleccionando aquella que, al incorporarla a la cadena actual, produce el mayor incremento en la entropía de Shannon, logrando una optimización local agresiva. Peso: $O(n^2 \cdot m)$

5) Método Tensorial (tensor.function): explora todo el espacio de combinaciones posibles generando el producto cartesiano de todas las opciones por posición, evaluando la entropía de Shannon de cada clave candidata y seleccionando la de mayor valor, factible solo para cadenas cortas debido a su complejidad exponencial. Peso: $O(m^n \cdot n)$

6) Método Híbrido (hybrid.function): parte de una solución inicial greedy, identifica posiciones con alta entropía Shannon, construye un vecindario reducido mediante producto cartesiano limitado (hasta 5 dimensiones para un buen control de memoria), y selecciona la mejor combinación global. Peso: $O(m^5 \cdot n)$

Donde n es la longitud de la frase (número de caracteres), m es el número máximo de equivalencias por carácter, y k es el número de caracteres originales que tienen equivalencias.

Cabe recalcar que, tanto en los algoritmos hechos en C++ como en los desarrollados en R, se anula la evaluación de los algoritmos "perfectos teóricos", es decir, el árbol n-ario en C++ y el tensorial en R, ya que el consumo de memoria de estos es muy elevado, provocando que para cadenas largas de caracteres se produzca un desbordamiento.

VI. Resultados

Para el procesamiento de los datos, se utilizó un procesador Ryzen 5 3600 a frecuencia de serie y 32 GB de RAM DDR4 a 3600 MHz.

En el caso de los datos, en los prompts realizados se solicitaron 30 contraseñas de 4 a 6 caracteres, 50 de 7 a 9 caracteres, 50 de 10 a 12 caracteres, 40 de 13 a 15 caracteres y 30 de 16 a 20 caracteres. Estos datos están divididos en frases aleatorias y frases que se asemejan a contraseñas reales.

Las principales entropías a comparar serán la de Shannon, la cuadrática, la normalizada y la de densidad, ya que estas cuatro se consideraron como las más adecuadas debido a la diversidad de información que permiten obtener.

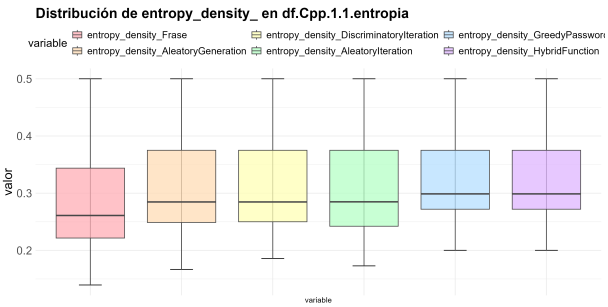

Figura 1: Entropía de Densidad C++

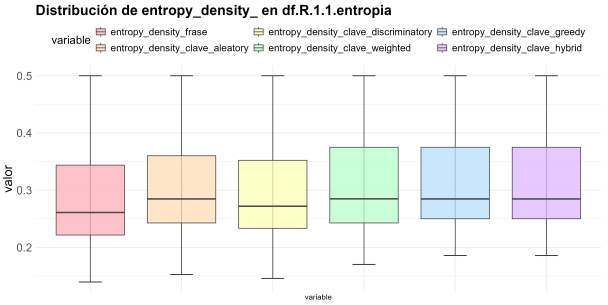

Figura 2: Entropía de Densidad R

Para el caso de la entropía de densidad, como es de esperar, las frases por sí solas obtuvieron la media y mediana más bajas tanto en R como en C++, siendo sus valores numéricos de 0.298 y 0.261 redondeados. Ahora, en los dos lenguajes, la función híbrida resaltó teniendo las medias más altas, siendo estas de 0.326 para el caso de la función híbrida en C++ y 0.312 para la función en R. La desviación estándar, por su parte, mostró una mayor dispersión en las frases, con valores que oscilaron entre 0.091 y 0.076 en los distintos escenarios, mientras que en las funciones híbridas y ponderadas se observaron desviaciones más controladas, entre 0.077 a 0.080.

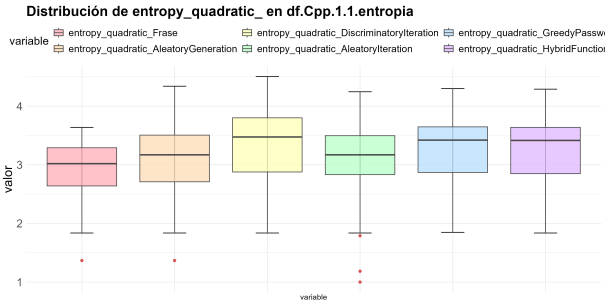

Figura 5: Entropía Quadratic C++

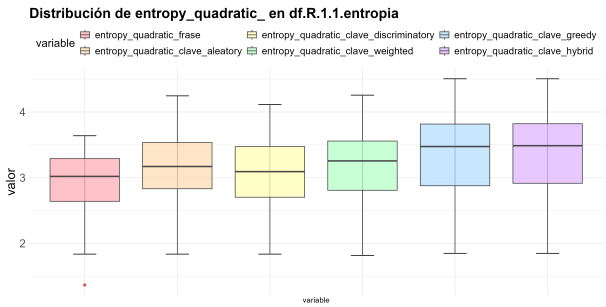

Figura 6: Entropía Quadratic R

En el análisis de la entropía cuadrática, las frases registraron las medias y medianas más bajas tanto en R como en C++, con valores aproximados de 2.906 y 3.019. A comparacion, las funciones para el parafraseamiento alcanzaron resultados más elevados, la función híbrida destacó en C++ con una media de 3.354 y una mediana de 3.487, mientras que la Discriminatory Iteration en R mostró cifras muy próximas, con 3.334 y 3.474. En cuanto a la desviación estándar, se evidenció una mayor variabilidad en las frases, con un valor de 0.473, mientras que en las funciones híbrida y discriminatoria las desviaciones fueron más altas pero consistentes, situándose alrededor de 0.624 y 0.623.

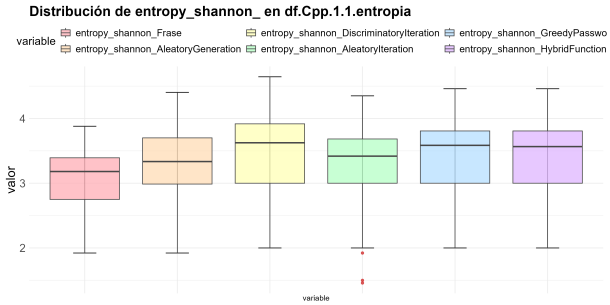

Figura 7: Entropía Shannon C++

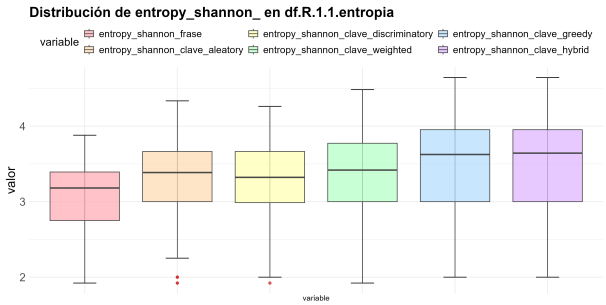

Figura 8: Entropía Shannon R

La entropía de Shannon mostró que las frases por sí solas obtuvieron las medias y medianas más bajas tanto en R como en C++, con valores aproximados de 3.037 y 3.181. En contraste, la funcion Discriminatory Iteration destacó en C++ con una media de 3.480 y una mediana de 3.625, mientras que la función híbrida en R presentó cifras muy cercanas, con 3.489 y 3.643. La desviación estándar, por su parte, evidenció una mayor variabilidad en las frases, con un valor de 0.489, mientras que en las funciones híbrida y discriminatoria las desviaciones fueron más altas pero consistentes, situándose alrededor de 0.628 en ambos casos.

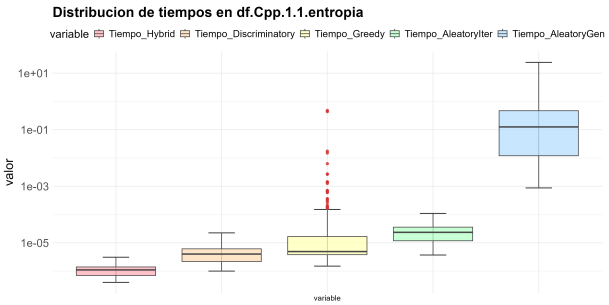

Figura 9: Tiempos C++

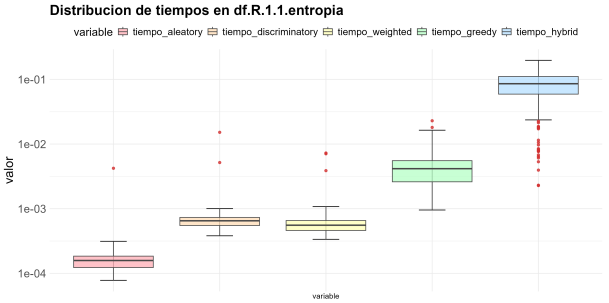

Figura 10: Tiempos R

Y como último, en el caso de los tiempos de respuesta, se observa que las funciones más simples como Greedy y Aleatory Iteration presentaron valores muy bajos en C++, con medias cercanas a 2.68×10^{-5} y 2.35×10^{-5} segundos, mientras que las funciones Hybrid y Aleatory Generation destacaron por ser aún más rápidas, con tiempos reducidos hasta el orden de 10^{-6} y 10^{-7} segundos. A comparación, en R los tiempos fueron considerablemente mayores: la función híbrida alcanzó valores de 0.086 y 0.044 en distintas ejecuciones, mientras que Greedy se mantuvo en torno a 0.004 y 0.003. Las funciones Weighted y Aleatory mostraron tiempos más bajos, entre 0.00065 y 0.00018 segundos, aunque con cierta variabilidad en las repeticiones. La desviación estándar, por su parte, reflejó una mayor dispersión en los métodos híbridos de R, con valores que oscilaron entre 0.043 y 0.086 segundos, mientras que en los algoritmos Greedy y Aleatory los tiempos fueron más estables y controlados, manteniéndose en rangos mucho más pequeños.

VII. Discusión.

En la comparación de los métodos, se observa

que la función híbrida en C++ presenta limitaciones al parafrasear frases cortas menores a 9 caracteres, como se muestra en la Figura 11. Este comportamiento se debe a que el algoritmo híbrido requiere un vecindario suficientemente amplio para explorar alternativas, lo cual no ocurre en cadenas pequeñas. En contraste, métodos como el aleatorio o el discriminatorio logran un parafraseamiento mayor en frases cortas que el algoritmo hibrido, ya que estas no dependen de un vecindario para la generacion del mismo.

Frase	Parafraseo
sol12	sol12
luz3	luz3
mar4	mar4
rio5	rio5
flor6	flor6
arena	4rEN@
bosque	bosque
campo	campo
viento	viento
hojas	hojas

Figura 11: Frases en C++ bajo el metodo hibrido.

Al analizar la frase "cereza fresca13" en cada método, tanto en C++ como en R, se aprecia que los resultados difieren en la forma de los equivalentes seleccionados. El método híbrido en C++ genera una clave con mayor complejidad visual y mucho mas variada en caracteres, mientras que en R la versión híbrida produce una variante más balanceada. El método discriminatorio en R muestra un desempeño muy cercano al híbrido de C++, lo que evidencia que ambos algoritmos tienden a distribuir mejor los símbolos y evitar repeticiones.

Método	Parafraseo
HybridFunction	(3rEZ4fRE$<a|e
DiscriminatoryIteration	(ER€2@Fr3$<4|e
GreedyPassword	<ER32@Fre$(4|e
AleatoryIteration	<eR3z@fRe$<@l€
AleatoryGeneration	C3R3z@fr35<@I3

Figura 12: Frase "cereza fresca13" en cada metodo en C++.

Método	Parafraseo
Aleatory	(€rE24frES<@|€
Discriminatory	(€REZ4FRES<4I€
Weighted	CEr€ZAFR3S(@|E
Greedy	<ER€2@Fr3$(4|e
Hybrid	<€RE2@Fr3$(4|e

Figura 13: Frase "cereza fresca13" en cada metodo en R.

Considerando los tiempos de ejecución, los métodos aleatorios y ponderados en R resultan más rápidos debido a su naturaleza vectorizada, mientras que en C++ la optimización con punteros y el uso de MT19937 permite un rendimiento competitivo en los métodos iterativos. Sin embargo, los algoritmos de búsqueda exhaustiva como el árbol n-ario en C++ y el tensorial en R presentan un costo exponencial, lo que los hace inviables para frases largas.

En el análisis de la entropía cuadrática, las frases originales registraron las medias y medianas más bajas tanto en R como en C++, con valores aproximados de 2.906 y 3.019. En comparación, la función híbrida destacó en C++ con una media de 3.354 y una mediana de 3.487, mientras que la Discriminatory Iteration en R mostró cifras muy próximas, con 3.334 y 3.474. En cuanto a la desviación estándar se pudo ver como esta se mantuvo muy baja en los metodos no aleatorios. Lo que permite que los casos excepcionales como el no parafreseamiento de una frase se eviten.

Cabe recalcar, que a pesar de que los tiempos son exageradamente mejores en C++, a la hora de manipular y escribir codigo este lenguaje presento muchas mayores complicaciones debido a ser un lenguaje de bajo nivel.

Como se puede apresar en los graficos, hay metodos que disvarian entre si aun teniendo un diagrama de flujo similar. A destacar esta el metodo discriminatorio, que su media y mediana presenta una diferencia de casi un 10 %. A pesar de ello, tanto el metodo hibrido en R como el discriminatorio en C++ aumentan la entropia Shannon y Cuadratica al rededor de un 15 %. Este porcetaje podra paraser un aumento insignificante, pero hay que tener en cuenta que estas entropias trabajan bajo caracteres locales, sumado a ello, el solo hecho de utilizar caracteres no alfabeticos es un gram avance en la mejora de la robuztes de una contrasena

VII. Conclusiones.

En conclusión, el análisis comparativo entre C++ y R demuestra que, aunque ambos lenguajes ofrecen estrategias sólidas para el parafraseamiento y la maximización de la entropía en contraseñas, C++ presenta ventajas significativas en términos de seguridad y tiempos de ejecución. La posibilidad de compilar librerías en formato ".o" permite ocultar las funciones internas y proteger la lógica de los algoritmos, mientras que los ejecutables ".exe" refuerzan esta seguridad al impedir la visualización directa de los diccionarios y equivalencias utilizados. Esto, sumado a los tiempos de respuesta mucho más bajos en C++

frente a R, convierte a este lenguaje en una opción altamente competitiva para servicios de gestión de contraseñas como LastPass, donde la rapidez en la generación y validación de claves es crítica para la experiencia del usuario y la robustez del sistema. Tal como se ha señala en el reporte anual de Lastpass [1], la mayoría de usuarios tiende a reutilizar contraseñas y rara vez las cambia, lo que incrementa la necesidad de mecanismos automáticos que garanticen diversidad y resistencia frente a ataques. En este contexto, la combinación de eficiencia computacional, ocultamiento de funciones y mayor seguridad en la distribución de ejecutables posiciona a C++ como un candidato idóneo para fortalecer la infraestructura de servicios de autenticación masiva, asegurando tanto la escalabilidad como la protección de los datos sensibles.

Referencias:

[1] LastPass, "Psychology of Passwords 2022," LastPass Annual Report, 2022. [En línea]. Disponible en: https://lastpass.com

[2] J. Yan, A. Blackwell, R. Anderson, and A. Grant, "The Memorability and Security of Passwords – Some Empirical Results," University of Cambridge Computer Laboratory Technical Report UCAM-CL-TR-500, Sept. 2000. [En línea]. Disponible en: http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-500.pdf

[3] W. Yang, N. Li, O. Chowdhury, A. Xiong, and R. W. Proctor, "An Empirical Study of Mnemonic Sentence-based Password Generation Strategies," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16), Vienna, Austria, Oct. 2016. [En línea]. Disponible en: https://doi.org/10.1145/2976749.2978346

[4] S. S. Woo and J. Mirkovic, "Improving Recall and Security of Passphrases Through Use of Mnemonics," University of Southern California, Information Sciences Institute, Technical Report, 2016. [En línea]. Disponible en: https://viterbik12.usc.edu/wp-content/uploads/2017/06/paper.pdf

[5] L. Zhang and A. F. Osth, "Modelling orthographic similarity effects in recognition memory reveals support for open bigram representations of letter coding," Cognitive Psychology, vol. 148, pp. 1–23, 2024, doi: 10.1016/j.cogpsych.2023.101619.

[6] D. J. Peterson and M. E. Berryhill, "The Gestalt principle of similarity benefits visual working memory," Psychonomic Bulletin & Review, vol. 20, no. 6, pp. 1282–1289, Dec. 2013, doi: 10.3758/s13423-013-0460-x.

[7] B. Pinna, D. Porcheddu, and J. Skilters, "Similarity and dissimilarity in perceptual organization: On the complexity of the Gestalt principle of similarity," Vision, vol. 6, no. 3, p. 39, Jun. 2022, doi: 10.3390/vision6030039.

[8] L. Chen, "Recommendation for Key Derivation Using Pseudorandom Functions," NIST Special Publication 800-108r1-upd1, U.S. Department of Commerce, National Institute of Standards and Technology, Aug. 2022, doi: https://doi.org/10.6028/NIST.SP.800-108r1-upd1

[9] K. Reaz and G. Wunder, "Expectation Entropy as a Password Strength Metric," arXiv preprint arXiv:2404.16853, Mar. 2024. [En línea]. Disponible en: https://arxiv.org/abs/2404.16853

[10] K. Janani, "Cybersecurity through Entropy Injection: A Paradigm Shift from Reactive Defense to Proactive Uncertainty," arXiv preprint arXiv:2504.11661, Apr. 2025. [En línea]. Disponible: https://arxiv.org/abs/2504.11661

[11] J. Jiang, A. Zhou, L. Liu, and L. Zhang, "OMECDN: A Password-Generation Model Based on an Ordered Markov Enumerator and Critic Discriminant Network," Applied Sciences, vol. 12, no. 23, p. 12379, Dec. 2022, doi: 10.3390/app122312379.

[12] A. Iosevich and T. Pham, "On an entropy inequality for quadratic forms and applications," arXiv preprint arXiv:2507.15196, Jul. 2025. [Online]. Available: https://arxiv.org/abs/2507.15196

[13] Y. Kim, C. Guyot, and Y.-S. Kim, "On the efficient estimation of min-entropy," arXiv preprint arXiv:2009.09570, Sep. 2020. [Online]. Available: https://arxiv.org/abs/2009.09570

[14] Microsoft Corporation, "Referencia del lenguaje C++ en el compilador Microsoft C++," Documentación de C++ en Visual Studio, 03-Abr-2023. [En línea]. Disponible en: https://learn.microsoft.com/es-es/cpp/cpp

[15] P. L'Ecuyer, "Maximally equidistributed combined Tausworthe generators," Math. Comput., vol. 65, no. 213, pp. 203–213, Jan. 1996, doi: 10.1090/S0025-5718-96-00696-5.

[16] C. Ş. Şahin, R. Lychev, and N. Wagner, "General Framework for Evaluating Password Complexity and Strength," arXiv preprint

arXiv:1512.05814, Dec. 2015. [En línea]. Disponible en: https://arxiv.org/abs/1512.05814

[17] W. Yu, Q. Yin, H. Yin, W. Xiao, T. Chang, L. He, L. Ni, and Q. Ji, "A Systematic Review on Password Guessing Tasks," Entropy, vol. 25, no. 9, p. 1303, Sep. 2023. doi: 10.3390/e25091303

[18] J. Jiang, A. Zhou, L. Liu, and L. Zhang, "OMECDN: A Password-Generation Model Based on an Ordered Markov Enumerator and Critic Discriminant Network," Applied Sciences, vol. 12, no. 23, p. 12379, Dec. 2022. doi: 10.3390/app122312379

[19] G. A. Alvarez and P. Cavanagh, "The capacity of visual short-term memory is set both by visual information load and by number of objects," Current Directions in Psychological Science, vol. 15, no. 3, pp. 106–111, Jun. 2004, doi: 10.1111/j.0963-7214.2004.01502006.x.

[20] E. Awh, B. Barton, and E. K. Vogel, "Visual working memory represents a fixed number of items regardless of complexity," Psychological Science, vol. 18, no. 7, pp. 622–628, Jul. 2007, doi: 10.1111/j.1467-9280.2007.01949.x.

[21] G. A. Alvarez and P. Cavanagh, "The capacity of visual short-term memory is set both by visual information load and by number of objects," Current Directions in Psychological Science, vol. 15, no. 3, pp. 106–111, Jun. 2004, doi: 10.1111/j.0963-7214.2004.01502006.x.

[22] L. Izakson, Y. Zeevi, and D. J. Levy, "Attraction to similar options: The Gestalt law of proximity is related to the attraction effect," PLOS ONE, vol. 15, no. 10, e0240937, Oct. 2020, doi: 10.1371/journal.pone.0240937.