República Bolivariana de Venezuela
Ministerio del Poder Popular para la Educación Universitaria
Universidad Rafael Urdaneta
Facultad de Ingeniería
Escuela de computación
Asignatura: Criptografía y Seguridad de Redes de Comunicación

Examen N°2

Alumno:
Erick Semprun CI:31085525
Docente:
Haller Bracho

Introducción

Las funciones hash criptográficas desempeñan un papel esencial en la seguridad informática moderna, ya que permiten garantizar la integridad de los datos, autenticar información y asegurar contraseñas. Una de sus propiedades más importantes es la resistencia a la búsqueda de preimágenes, es decir, la dificultad de encontrar un mensaje de entrada que produzca un valor hash específico. En particular, las preimágenes parciales —mensajes que generan hashes con ciertos patrones predefinidos, como ceros iniciales— permiten evaluar de manera experimental la robustez de una función hash ante ataques de fuerza bruta.

El presente experimento tiene como objetivo comparar el esfuerzo computacional requerido para encontrar preimágenes parciales en tres funciones hash ampliamente utilizadas: MD5, SHA-1 y SHA-256. Para ello, se implementó un programa que genera hashes a partir de mensajes personalizados, con formato fijo y contador incremental, y repite el proceso hasta obtener un hash con una cantidad específica de ceros al inicio. Los datos obtenidos —como el número de intentos y el tiempo de ejecución— permiten observar el crecimiento exponencial de la dificultad y analizar el comportamiento de cada algoritmo ante diferentes niveles de exigencia. Esta comparación práctica se relaciona directamente con los principios teóricos de complejidad y seguridad criptográfica, proporcionando una visión concreta del costo computacional asociado a la búsqueda de preimágenes.

Desarrollo

Metodología Utilizada

Hardware y software utilizado:

- CPU: AMD Ryzen 7 3ra Generacion
- RAM: 24GB
- Sistema operativo: Windows 11
- Lenguaje: Python 3.11.6

Librerías utilizadas:

- hashlib: Proporciona acceso a las funciones hash MD5, SHA-1 y SHA-256.
- time: Para medir el tiempo transcurrido entre el inicio y fin de cada experimento.
- csv: Para almacenar los resultados en un archivo CSV y facilitar su análisis posterior con herramientas de hojas de cálculo.

Formato del mensaje de entrada

Para cada intento, se construyó un mensaje de entrada de la forma:
m = [NOMBRE_COMPLETO_ESTUDIANTE] + ":" + [CONTADOR]
Por ejemplo:
Erick Jose Semprun Llanes:1243
Este formato fue elegido para garantizar que: El mensaje base es fijo y personalizado (único por estudiante).
Solo cambié la parte variable (el contador), lo cual facilita el análisis de cuántos intentos fueron necesarios para cada nivel de dificultad.

Funciones hash evaluadas

Se evaluaron las siguientes tres funciones hash criptográficas, ampliamente utilizadas en seguridad informática:

- MD5 (128 bits): Obsoleta para integridad fuerte, pero útil para fines comparativos.

- SHA-1 (160 bits): Vulnerable a colisiones, pero aún usada en ciertos sistemas legados.
- SHA-256 (256 bits): Estándar actual en muchas aplicaciones seguras como Bitcoin y TLS.

Definición de dificultad

Se definieron cuatro niveles de dificultad, consistentes con la cantidad de ceros hexadecimales requeridos al inicio del hash:

- Nivel 1: Hash comienza con 0 (un cero hexadecimal).
- Nivel 2: Hash comienza con 00.
- Nivel 3: Hash comienza con 000.
- Nivel 4: Hash comienza con 0000.
- Nivel 5: Hash comienza con 00000.

Proceso experimental

Para cada combinación de función hash y nivel de dificultad:

1. Se inicializa un contador en 0.
2. En cada iteración, se generó un nuevo mensaje del tipo nombre:contador.
3. Se calculó el hash del mensaje usando la función correspondiente.
4. Se verificó si el hash generado comenzaba con el número requerido de ceros.
    - Si no lo cumplía, se incrementa el contador y se repetía el proceso.
    - Si lo cumplía, se almacenaban los siguientes datos:
        - El mensaje exitoso encontrado.
        - El hash resultante.
        - El valor del contador (número total de intentos).
        - El tiempo total de ejecución.

Este procedimiento se repitió de forma independiente para cada nivel y para cada función hash, permitiendo una comparación directa del esfuerzo requerido por cada algoritmo.

Automatización y almacenamiento

El experimento fue totalmente automatizado en un solo script Python. Los resultados fueron almacenados en un archivo CSV estructurado, que incluye las columnas:

- función: nombre de la función hash utilizada (MD5, SHA1, SHA256).
- dificultad: cantidad de ceros requeridos.
- mensaje: mensaje de entrada que generó el hash deseado.
- hash: valor hash obtenido.
- intentos: cantidad de mensajes evaluados hasta tener éxito.
- Tiempo: duración en segundos del proceso.

Resultados

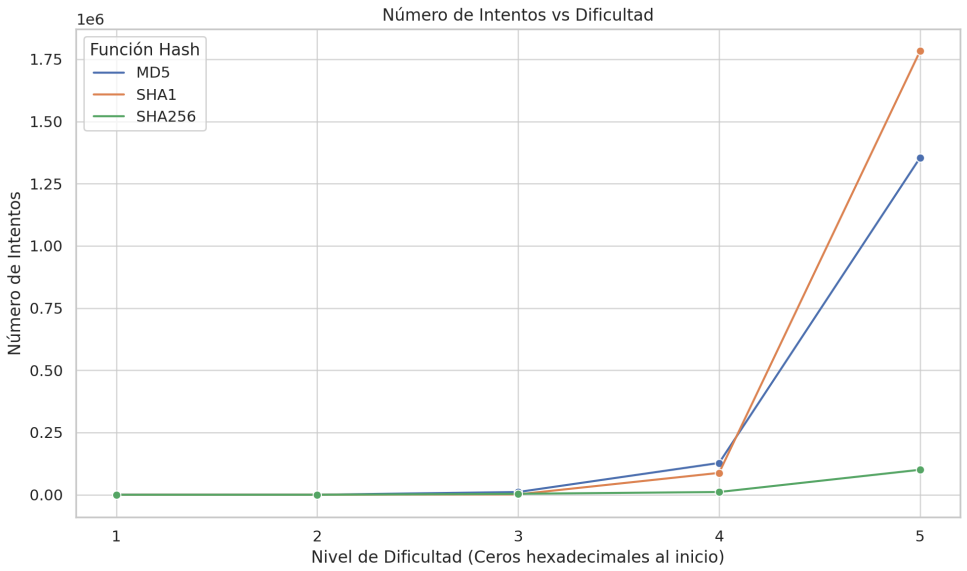
Funcion	Dificultad	Mensaje	Hash	Intentos	Tiempo
MD5	1	Erick Jose Semprun Llanes:2	03eea881 d5f149e65 c39b5e17 b146556	2	0.0
MD5	2	Erick Jose Semprun Llanes:121	004bd2d7 6f10335bc 091372f4b 4d0341	121	0.0
MD5	3	Erick Jose Semprun Llanes:110 01	00047203 45e2a80a 7677f2fc4 7d8c718	11001	0.0284051 89514160 156
MD5	4	Erick Jose Semprun Llanes:127	0000b3a2 39592dab 350031ef9	127660	0.2657942 771911621

		660	596341d		
MD5	5	Erick Jose Semprun Llanes:1353989	0000063ab46ff3804ef29d40062b4ea4	1353989	2.5271975994110107
SHA1	1	Erick Jose Semprun Llanes:5	0224116cc665b96c28c487bf2477c6f64e8cfdc2	5	0.0
SHA1	2	Erick Jose Semprun Llanes:125	00efcd0e8e26e53cd9c8356499b75939cc6f2939	125	0.0
SHA1	3	Erick Jose Semprun Llanes:1610	000e95592c3e2fee3e175ba27a7f326cef71d3ed	1610	0.003589868545532 2266
SHA1	4	Erick Jose Semprun Llanes:87825	0000f094cec657855c6a14730767ceff4b57cc56	87825	0.163513898849487 3
SHA1	5	Erick Jose Semprun Llanes:178	000004bbbbef980efbcfb4545d	1782471	3.236866712570190 4

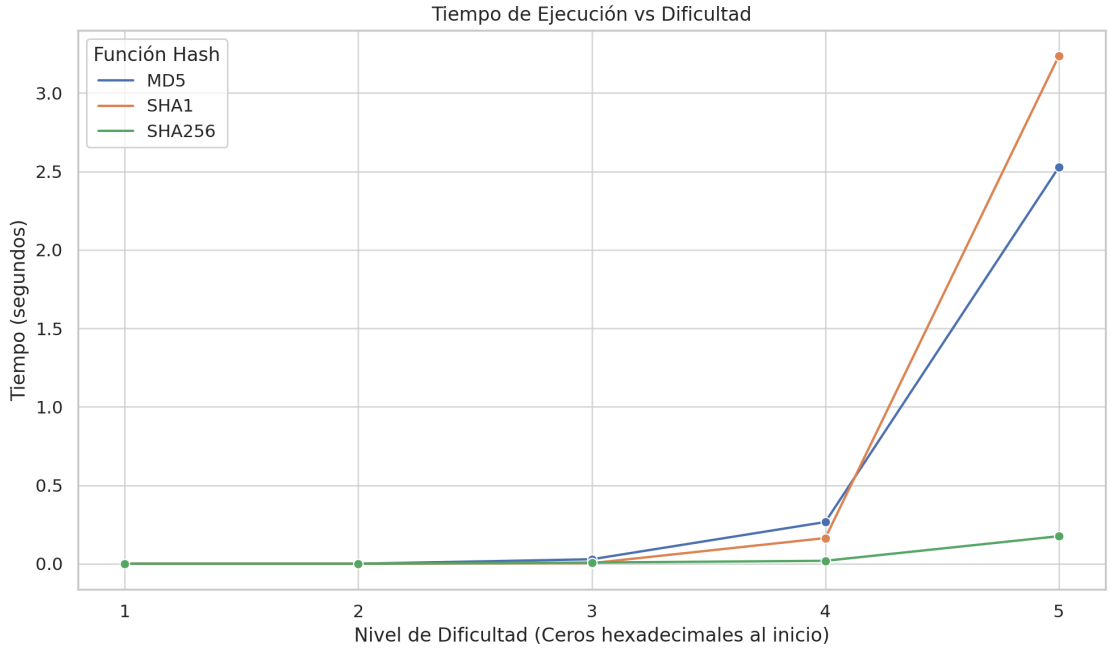
		2471	565444168628bf		
SHA256	1	Erick Jose Semprun Llanes:27	0766b5e9bfd07cb48b20230129cb8d794503309a1f0c1fe08e026ee83ae6b30e	27	0.0
SHA256	2	Erick Jose Semprun Llanes:37	00629ac429418982 4520486dc51aa7fe0 7b29dd4c5c70ea70c98dfec18dd58b1	37	0.0
SHA256	3	Erick Jose Semprun Llanes:3863	0007af9694b846742ed7ba255c6a3817f0686bf2e81799fa72411700739f7345	3863	0.007350206375122 07
SHA256	4	Erick Jose Semprun Llanes:109	0000e131bcdd3075b7a7c32d	10999	0.018597126007080078

		99	39ed3369 918efc5be 9e3451c8 05d84d18 79010c2		
SHA256	5	Erick Jose Semprun Llanes:100 358	00000f69d 6d208201 0eaede56 7435cbed 90c80b4d 20ab9ba11 4dc14d93 261f14	100358	0.1754074 09667968 75

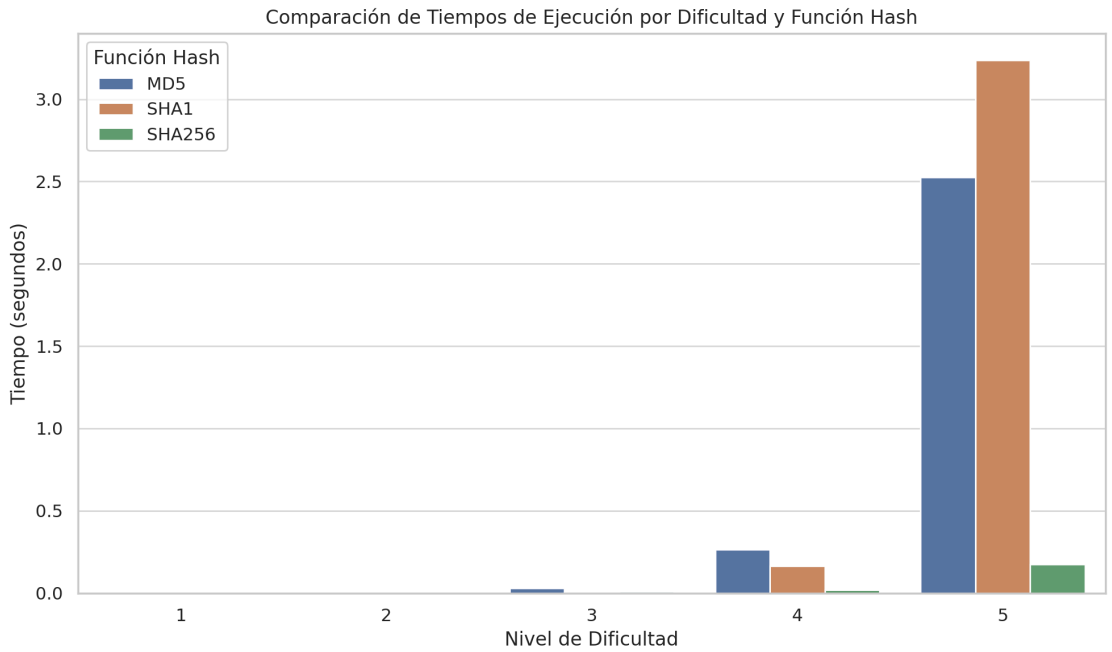
Durante el experimento se ejecutaron pruebas para cuatro niveles de dificultad (de 1 a 5 ceros hexadecimales al inicio del hash) utilizando las funciones hash MD5, SHA-1 y SHA-256. Para cada combinación, se registraron el mensaje exitoso, el hash resultante, el número de intentos necesarios y el tiempo total de ejecución. Los datos recopilados se resumen en las siguientes representaciones gráficas:



En esta gráfica se observa una clara tendencia exponencial en el número de intentos necesarios a medida que aumenta la dificultad. En todos los algoritmos, el salto de dificultad 3 a 4 muestra un incremento particularmente pronunciado, lo cual es coherente con la naturaleza aleatoria de los hashes y la baja probabilidad de encontrar coincidencias con múltiples ceros iniciales.



El tiempo de ejecución sigue un patrón paralelo al número de intentos. Para un mismo nivel de dificultad, SHA-256 demanda sistemáticamente más tiempo que SHA-1 y MD5, como consecuencia de su mayor complejidad computacional y tamaño de salida de 256 bits. MD5, al ser más ligera, completa las búsquedas en menos tiempo, aunque con menor seguridad.



Esta gráfica de barras permite visualizar de forma directa las diferencias entre algoritmos para cada nivel. Se refuerza la conclusión de que SHA-256 es el más exigente en términos de tiempo, especialmente a partir del nivel 3.

Observaciones

Se observa que:

Crecimiento exponencial del esfuerzo computacional:

Cada cero hexadecimal adicional en el hash representa una reducción del espacio de búsqueda en un factor de 16 (ya que cada dígito hexadecimal tiene 16 posibles valores). Esto implica que la probabilidad de éxito en cada intento es de 1/16^n , siendo n la cantidad de ceros requeridos. Esto se refleja en el aumento abrupto de intentos y tiempo entre niveles consecutivos, especialmente del nivel 3 al 5.

Comparación de rendimiento entre algoritmos:

- MD5 resultó ser el más rápido en todos los niveles, lo cual se debe a su simplicidad. Sin embargo, esta eficiencia viene acompañada de debilidades conocidas en términos de colisiones y seguridad, por lo que no es adecuado para sistemas que requieran alta resistencia criptográfica.

- SHA-1 ofrece un equilibrio entre velocidad y seguridad, aunque también ha sido objeto de ataques prácticos y se encuentra en proceso de desuso.
- SHA-256 mostró el mayor tiempo de cómputo, pero es actualmente una de las funciones más seguras y recomendadas para aplicaciones modernas como blockchain y firmas digitales.

Relación con la teoría de seguridad:

El experimento ilustra cómo, incluso para preimágenes parciales (una meta mucho más simple que encontrar una preimagen completa o una colisión), el esfuerzo necesario se incrementa significativamente. Esto resalta la importancia del diseño eficiente de algoritmos de minería, autenticación y firmas que dependen de este tipo de propiedades hash.

Aplicabilidad práctica:

En contextos como blockchain (por ejemplo, Bitcoin), se requieren hashes que comiencen con varios ceros como prueba de trabajo. Este experimento reproduce, a pequeña escala, el tipo de esfuerzo computacional requerido, reforzando la comprensión práctica de estos mecanismos de seguridad.

Resumen

Este experimento evalúa la resistencia a la búsqueda de preimágenes parciales en tres funciones hash criptográficas: MD5, SHA-1 y SHA-256. Se desarrolló un programa en Python que genera mensajes con un formato fijo personalizado y un contador incremental, calculando sus respectivos hashes hasta encontrar valores que comienzan con un número creciente de ceros hexadecimales (de uno a cuatro). Para cada caso, se registraron el mensaje hallado, el hash resultante, la cantidad de intentos y el tiempo de ejecución. Los resultados obtenidos evidencian un crecimiento exponencial en el esfuerzo requerido a medida que aumenta la dificultad, siendo SHA-256 la función más costosa computacionalmente, seguida de SHA-1 y MD5. Estos hallazgos coinciden con las expectativas teóricas sobre la complejidad de las funciones hash seguras y permiten reflexionar sobre su aplicabilidad en escenarios reales que requieren resistencia ante ataques por fuerza bruta.