Jaime Andres Torres Bermejo
202014866
andrestbermejoj@gmail.com
Alan Snade Fonseca Garcia
202111432
a.fonsecag@uniandes.edu.co

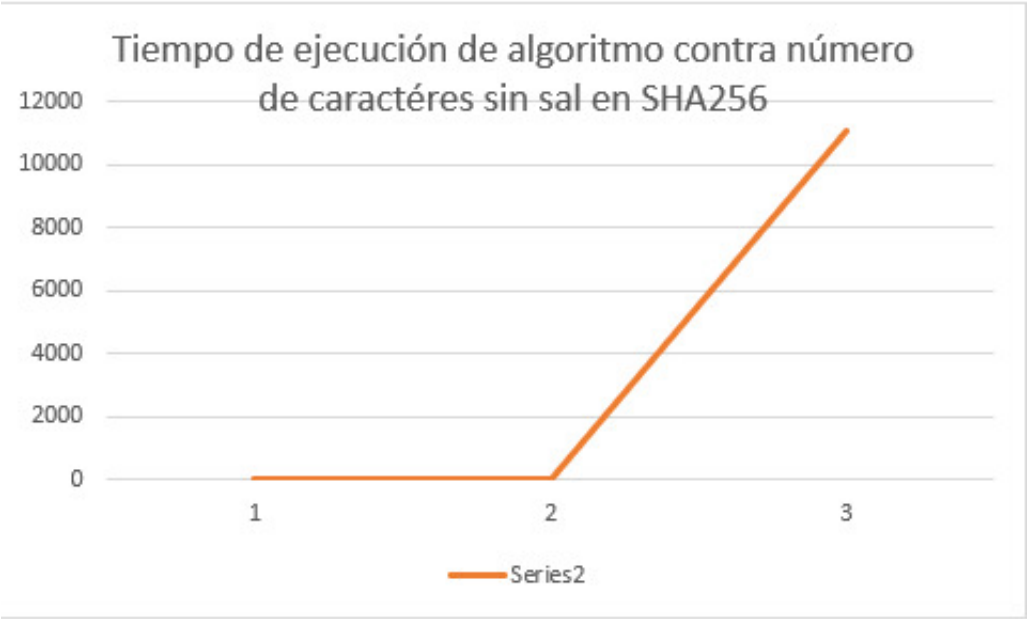
Caso 3
Infraestructura Computacional

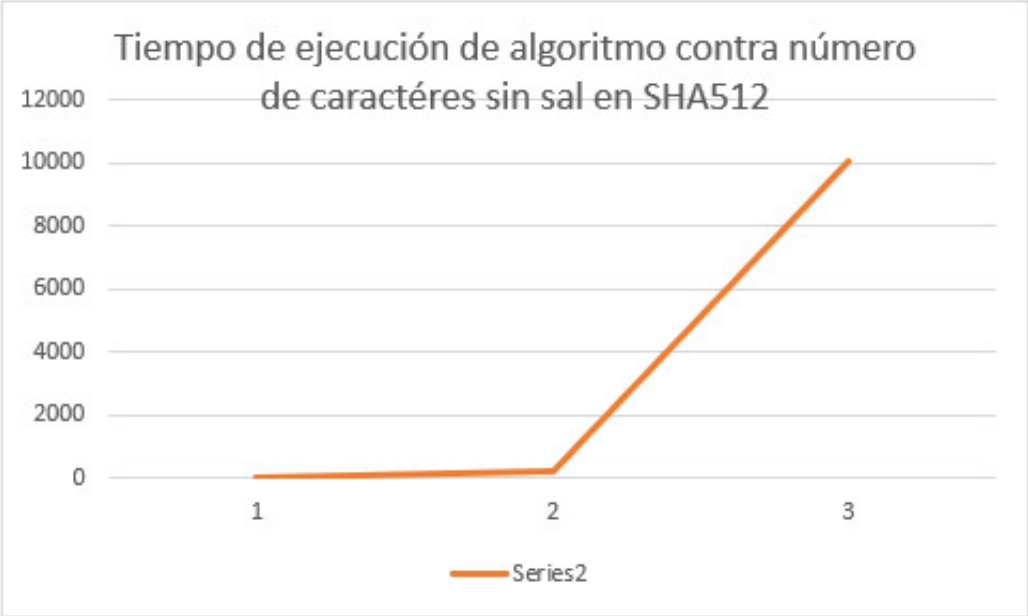
2023-05-04

1 Detalles de implementación

El código se divide en 2 clases, la clase principal Main y la clase donde se encuentra la lógica, PasswordCracker. En el Main se piden todos los paramétros de entrada: el algoritmo de hash, la sal que se va a utilizar, el número de threads y por último el código de hash a utilizar.
Una vez se tienen todos los paramétros se le pasan a PasswordCracker, cuyo constructor recibe todos los paramétros. En la clase se definen 2 constantes y un array de char, la cual contiene el alfabeto en minúsculas. Para llevar registro del tiempo que tarda la busqueda de una contraseña para distintos casos se lleva un contador desde el comienzo del metódo iniciar, se crean el número de threads ingresado por el usuario, si son 2 threads al primero se le pasa la mitad del alfabeto y al segundo la mitad final en el metódo de buscar que invoca cada uno.
El metódo de búsqueda recibe un inicio, un final, y un prefijo, adicionalmente la clase maneja una variable encontrado, que indica si la contraseña ya fue hallada, si este indicador es verdadero, se retorna el valor del mismo.
Si esto no se cumple se evalúa la longitud del prefijo, si es menor o igual a 7 se da una contraseña compuesta de la suma entre el prefijo y la sal, posteriormente a esta contraseña se le realiza el hash correspondiente con un metódo auxiliar, si el hash hallado por el metódo es igual al código criptográfico de hash se cambia el estado de encontrado a verdadero y se entrega el prefijo como contraseña. Si el hash es distinto se realiza un ciclo recursivo con la función search, en el cual se comienza desde el inicio hasta el final del alfabeto introducido, lo que cambia en la recursión es que al prefijo se le añade la letra del alfabeto en la posición i del ciclo.
Esto genera un montón de combinaciones y por lo tanto tiempos de búsqueda un poco altos, entre más adentrada dentro del alfabeto este la letra, más tiempo le va a costar al algoritmo, otro factor que afecta este tiempo son las combinaciones de letras distintas, entre más diversidad halla en las letras de la contraseña, más tiempo le cuesta al algoritmo descifrarla.
Por último, el hash recibe una contraseña de entrada y devuelve su hash utilizando el algoritmo de hash especificado por la variable algorithm.
Primero, se comprueba si el algoritmo es SHA-256 o SHA-512 utilizando una estructura de control if-else. Si el algoritmo es desconocido, se lanza una excepción.
Luego, se crea una instancia de la clase MessageDigest utilizando el algoritmo especificado, y se llama a su método digest con el argumento input.getBytes(StandardCharsets.UTF8) para obtener el hash de la cadena de entrada.
Este hash se almacena en un arreglo de bytes hashBytes. A continuación, se convierte cada byte en una cadena hexadecimal de 2 dígitos y se concatenan todos los resultados para formar la cadena de hash completa.
Finalmente, se devuelve esta cadena de hash.

2 Gráficos de ejecución





En las gráficas se ve una tendencia exponencial muy marcada. Esto se debe a la forma como los algoritmos de hash funcionan y cómo el sistema que genera estos códigos esta diseñado para evitar ataques de fuerza bruta como el visto en este caso, también por esto es que cálculos de más valores, como podría ser el estandar de las contraseñas modernas (8 caractéres, por ejemplo) son tan ineficientes, en casos donde probamos 4 caracteres además de la sal demoraron más de media hora en completarse. Es de notar que es posible que dado que fueron corridos sin reiniciar el sistema o el IDE de prueba, estos resultados estuviesen afectados por algún tipo de caché que mantuviese las respuestas, pues las segundas o terceras ejecuciones de una longitud concreta eran significativamente menores que la primera, sobretodo si se corría un programa de debugging antes de hacer la prueba.

3 Preguntas realizadas.

Cálculo 3, Velocidad del procesador = 4.1 GHz, 1GHz = 2^30, se asume que 1GHz son 2^30 ciclos/segundo.

$4.1x2^30 = 4402341478,4$

Teniendo en cuenta que el alfabeto tiene 26 letras y una longitud máxima de 7 caracterés, esto se pue-

de ver tal que: $26^7 = 8091810176$ Entonces los ciclos por segundo serian: $\frac{26^7}{4.1x2^30} = 1,82444segundos$

Cálculo 4, mismos datos de antes excepto que un alfabeto tiene 78 caracterés y las contraseñas son de 8 caracterés, para el primer caso:

$\frac{78^8}{4.1x2^30} = 311224,01054segundos$

para el segundo caso mismo alfabeto pero ahora 10 caracterés para las contraseñas:

$\frac{78^10}{4.1x2^30} = 1,89349x10^9segundos$

para el tercer caso mismo alfabeto pero ahora 12 caracterés para las contraseñas:

$\frac{78^12}{4.1x2^30} = 1,125x10^13segundos$

Pregunta 1: Hoy en día, los algoritmos de generación de códigos criptográficos de hash más utilizados son:
SHA-256 (Secure Hash Algorithm 256 bits): Es un algoritmo de hash ampliamente utilizado para garantizar la integridad de los datos en aplicaciones de seguridad, como la autenticación de contraseñas y la firma digital de documentos.
SHA-3 (Secure Hash Algorithm 3): Es una familia de algoritmos de hash que incluyen varias variantes de longitud de salida, como SHA3-256 y SHA3-512. Fue diseñado como una alternativa al algoritmo SHA-2 y se utiliza en aplicaciones que requieren una mayor seguridad criptográfica.
bcrypt: Es un algoritmo de hash que se utiliza principalmente para almacenar contraseñas de forma segura en aplicaciones web. Utiliza una función de hash adaptativa que hace que la tarea de romper contraseñas sea mucho más difícil para los atacantes.
Argon2: Es un algoritmo de hash diseñado específicamente para la protección de contraseñas. Utiliza una función de hash adaptativa que hace que la tarea de romper contraseñas sea mucho más difícil para los atacantes, y también incluye medidas de seguridad adicionales para proteger contra ataques de canal lateral. Argon2 tiene 2 variaciones, Argon 2d que es más adecuado para el hash de contraseñas y Argon2i que es más adecuado para la derivación de claves de cifrado.
MD5 (Message-Digest Algorithm 5): Aunque ya no se considera seguro, aún se utiliza en algunas aplicaciones de compatibilidad y en algunos sistemas heredados para verificar la integridad de los datos.
Un algoritmo de hash se vuelve obsoleto y se deja de utilizar cuando se descrubre alguna vulnerabilidad en su diseño que lo vuelve inseguro contra ataques, vulnerabilidades que son fácilmente explotables para comprometer los datos protegidos por el algoritmo.

Pregunta 2: El mining de bitcoin consiste en resolver problemas criptográficos con el fin de validar transacciones y poder agregar bloques nuevos a la cadena de bloques de Bitcoin, el proceso se realiza en 4 pasos:
Los nodos de la red de Bitcoin recopilan las transacciones pendientes y las agrupan en un bloque.
Los mineros compiten por resolver un rompecabezas criptográfico complejo, conocido como prueba de trabajo (PoW), que está asociado con el bloque.
El minero que resuelve el rompecabezas criptográfico primero y agrega el bloque a la cadena de bloques recibe una recompensa en forma de nuevas bitcoins.
Una vez que un bloque ha sido agregado a la cadena de bloques, todas las transacciones que contiene son consideradas válidas y no pueden ser modificadas sin cambiar todo el bloque, lo que lo haría inválido.
Este proceso esta basado en el algoritmo criptográfico de hash SHA-256, cuyo fin es generar la prueba de trabajo para cada bloque. La idea es que resolver el rompecabezas criptográfico requiere una gran cantidad de energía y tiempo, lo que hace que el proceso sea difícil y costoso, pero también seguro contra ataques de doble gasto y otros tipos de manipulación.

Pregunta 3: El problema de seguridad asociado con el uso de Rainbow Tables es que existe la posibilidad de que un atacante pueda precomputar y tenga un gran número de hashes guardados en especifico para un conjunto de valores de entrada muy probables y luego en la tabla precalculada realizar una búsqueda para encontrar una coincidencia con un hash en particular, el rol de la sal para solucionar este problema es que vuelve cada hash único, inclusive para la misma entrada, volviendo mas complejo desencriptarlo y haciendo más grande el espacio requerido para realizar un ataque de tipo Rainbow Table.

Referencias:
- Ferguson, N., Schneier, B., Kohno, T. (2010). Cryptography engineering: Design principles and practical applications. John Wiley Sons.

- Menezes, A. J., van Oorschot, P. C., Vanstone, S. A. (1996). Handbook of applied cryptography. CRC Press.
- Stallings, W. (2017). Cryptography and network security: principles and practice. Pearson.
- Ristenpart, T., Shrimpton, T. (2012). Careful with composition: limitations of the ideal cipher and its use in composite hash functions. Journal of Cryptology, 25(4), 648-677.
- ”Bitcoin Mining, Explained”https://www.investopedia.com/terms/b/bitcoin-mining.asp
- ”The mathematics of Bitcoin”https://arxiv.org/abs/1609.08423
- ”Mastering Bitcoin: Unlocking Digital Cryptocurrencies”https://github.com/bitcoinbook/bitcoinbook/blob/develop/book.asci
- ”Bitcoin: A peer to-peer Electronic Cash System”https://bitcoin.org/bitcoin.pdf
- National Institute of Standards and Technology (NIST). (2020). Secure Hash Standard (SHS). Recuperado de https://csrc.nist.gov/projects/hash-functions

- OWASP. (2021). Password Storage Cheat Sheet. Recuperado de https://owasp.org/www-project-cheat-sheets/cheatsheets/PasswordStorageCheatSheet.html

- Bernstein, D. J. (2015). The Cost of Cryptography in Large-Scale Data Centers. Recuperado de https://cr.yp.to/highspeed/cool

- Wikipedia. (2021). Comparison of cryptographic hash functions. Recuperado de https://en.wikipedia.org/wiki/Comparison$_of$