Facultad de ingeniería y Ciencias – Escuela de Informática y Telecomunicaciones

Laboratorio N°4: Criptografía y Seguridad en Redes - CIT2113_CA01– 2021/02

Profesor: Víctor Manríquez
Alumno: Brian Ignacio Castro Farías

Introducción:

Luego de finalizar el trabajo de passwords, los alumnos del curso de criptografía Son contratados por el gremio de los dueños de los sitios de contraseñas expuestas. Los cuáles ofrecen una cuantiosa suma si es que se le entrega un algoritmo propio de Hashes a fin de poder resguardar sus sitios y por sobre todo, sus contraseñas. El representante de los dueños de sitios web vulnerados, exige de que el Algoritmo, debe tener cómo mínimo un set de características, para asegurar un estándar mínimo de seguridad. Las características son las siguientes:

- El número de caracteres final del texto procesado, debe ser fijo y no menor de 25 caracteres. Este número no debe variar, aunque el texto de entrada sea más largo.
- Se requiere que el programa pueda recibir como entrada, tanto un string o texto mediante STDIN, cómo también mediante archivos con múltiples entradas separadas por el salto de línea.
- Cualquier ligero cambio al texto de entrada, debe cambiar el resultado final del algoritmo.
- El procesamiento del texto de entrada, debe ser rápido, no debe tomar mucho tiempo, independiente de la cantidad de texto de entrada. * Se recomienda el uso de operaciones matemáticas al texto de entrada, a fin de optimizar la velocidad de procesamiento del texto de entrada*
- Adicionalmente se requiere que posea a lo menos dos opciones:

    1. una que procese el texto de entrada y calcule el Hash de estas entradas (sea por STDIN, cómo mediante un archivo)

    2. Y otra opción que sólo calcule la entropía del texto de entrada. En este último caso, debe arrojar mediante STDOUT que entropía posee cada texto de entrada (El formato de la salida a STDOUT, debe mostrar tanto el texto analizado, cómo la entropía calculada separada por un delimitador que permita diferenciar los diferentes campos).

Marco teórico:

Una función criptográfica hash- usualmente conocida como "hash"- es un algoritmo matemático que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija. Independientemente de la longitud de los datos de entrada, el valor hash de salida tendrá siempre la misma longitud.

Según un generador online de hash SHA-1, (SHA-1 es una de las funciones hash más ampliamente utilizadas en la informática, junto con MD 5 y SHA-2) el hash de mi nombre, Brian, es: 75c450c3f963befb912ee79f0b63e563652780f0. Versus uns cadena muy parecida "Brain" (cerebro, en inglés). El hash SHA-1 para "brain" (cerebro), siempre según el generador online de hash SHA-1, es: 8b9248a4e0b64bbccf82e7723a3734279bf9bbc4, totalmente distinto al primer hash.

Aplicaciones de las funciones hash:

Fundamentalmente, se utilizan para verificar que un conjunto de datos sea correcto y seguro.
Comprobación de archivos: Para comprobar que un archivo es original, que no está dañado y no tiene virus, el creador debe crear un hash y adjuntarlo. Una vez descargado, se obtiene la copia del código. Si este es igual al original, el archivo está en buen estado.
Almacenar contraseñas: Manera fácil de guardar miles de contraseñas en un sistema. Estas tienen que ser encriptadas para evitar ser mostradas en caso de un ataque.
Fácil acceso a un archivo en una base de datos: Con el hash del archivo puedes encontrar fácilmente cualquier archivo dentro de una base de datos, sin tener que hacer una búsqueda profunda.
Creación de una cadena de bloques: Se utilizan para crear criptomonedas. Esta cadena se crea con el hash del bloque anterior, por lo que si un hash se cambia, la cadena se rompería y no se podría acceder a su información.

Desarrollo:

Para comenzar con el desarrollo de la implementación se escoge el lenguaje de programación Python y se sabe que hay que trabajar con operaciones matemáticas para conservar rapidez y eficiencia en el código, mas aún hay que tener variables de aleatoriedad suficientemente capaces de no generar colisión, es decir que para la misma entrada, nunca se devuelva el mismo hash.

Código:

```
#librerías necesarias
import requests
import time
from datetime import date, datetime
import json
import math
import hashlib
import csv


#función para obtener el valor de uf  a día de hoy en chile
def ValorUF():
    valor='uf'
    fecha=date.today()
    if(fecha.day<10):
        day='0'+str(fecha.day)
    else:
        day=str(fecha.day)
    if(fecha.month<10):
        month='0'+str(fecha.month)
    else:
        month=str(fecha.month)
    year=str(fecha.year)
    fecha=day+'-'+month+'-'+year


    # En este caso hacemos la solicitud para el caso de consulta de un
indicador en un año determinado
    url = f'https://mindicador.cl/api/{valor}/{fecha}'
    response = requests.get(url)
    data = json.loads(response.text.encode("utf-8"))
    # Para que el json se vea ordenado, retornar pretty_json
    pretty_json = json.dumps(data, indent=2)

    return data['serie'][0]['valor']



#función de hashing
```

```python
def HashBrianNasheeeeeeeeee(palabra):

    #defino variables de temporalidad y listas donde guardar información
    time=datetime.now()
    uf=int(ValorUF())
    #empiezo a construir el hash
    cadena=''
    #agrego el valor de la uf hoy
    cadena=cadena+str(uf)
    largo=len(palabra)
    ascii_values=list()
    values=list()
    #recorro la palabra de entrada y tomo cada uno de los carácteres para meterlo en una lista, y en otra con sus valores ascii
    for character in palabra:
        values.append(character)
        ascii_values.append(ord(character))
    #según variables de decisión voy agreganco cosas al hash
    if str(values[0])=='\n':
        cadena=cadena+str('a')
    else:
        cadena=cadena+str(values[0])
    cadena=cadena+str(int((ascii_values[int(largo/2)])/2))
    if str(values[int(largo/2)])=='\n':
        cadena=cadena+str('e')
    else:
        cadena=cadena+str(values[int(largo/2)])
    #agrego variables de temporalidad año y día
    cadena=cadena+str(time.year)
    cadena=cadena+str(time.day)
    #según variables de decisión voy agreganco cosas al hash
    if str(values[largo-1])=='\n':
        cadena=cadena+str('i')
    else:
        cadena=cadena+str(values[largo-1])
    #agrego variables de temporalidad mes, hora y minuto
    cadena=cadena+str(time.month)
    cadena=cadena+str(int((time.hour)/10))
    cadena=cadena+str(int((time.minute)/10))
    if str(values[int(largo/4)])=='\n':
        cadena=cadena+str('o')
    else:
        cadena=cadena+str(values[int(largo/4)])
    #agrego variables de temporalidad segundos
    cadena=cadena+str(int(((ascii_values[int(largo/2)])/2+(ascii_values[largo-1])/10)))
    cadena=cadena+str(time.second+10)
```

```
    return cadena
#formula para calcular la entropía

def Entropia(cadena):
    largo=len(cadena)
    #base ascii 128 caracteres
    base=128
    entropia=largo*math.log(base,2)
    return entropía

#menú de opciones
if __name__=="__main__":
    decision=0
    while decision!=6:
        print("Bienvenido al mecanismo de hashing de Brian Nasheeee:\nOpción
1: Hashear una cadena de texto \nOpción 2: Hashear un archivo de texto
\nOpción 3: Calcular entropía de una cadena de texto\nOpción 4: Comparación
entropy vs SHA1,SHA2,MD5\nOpción 5: Comparación time vs SHA1,SHA2,MD5 \nOpción
6: Salir del programa:)\n")
        decision=int(input())
        if decision==1:
            print("Ingrese su cadena de Texto:\n")
            cadena=str(input())
            aux=HashBrianNasheeeeeeeeee(cadena)
            print('Hashing: '+aux+', largo'+str(len(aux)))
            print('\n')
        elif decision==2:
            print("Ingrese la ruta de su archivo de Texto:\n")
            ruta=str(input())
            f = open (ruta,'r', encoding='utf-8')
            mensaje = f.readlines()
            for i in mensaje:
                if mensaje!='':
                    print('Mensaje:'+str(i)+'\nHasheando:')
                    aux=HashBrianNasheeeeeeeeee(str(i))
                    largo=len(aux)
                    if largo<25:
                        for i in range(largo,25):
                            aux=aux+'a'
                    print(aux+', largo'+str(len(aux)))
                    print('\n')


            f.close()
        elif decision==3:
            print("Ingrese la cadena de Texto:\n")
            cadena=str(input())
```

```
            aux=Entropia(cadena)
            print('La entropia es: '+str(aux)+'\n')
        elif decision==4:
            print("Ingrese la ruta de su archivo de Texto:\n")
            ruta=str(input())
            f = open (ruta,'r', encoding='utf-8')
            mensaje = f.readlines()
            lista=list()
            encabezado=list()
            encabezado.append('Cadena')
            encabezado.append('HashBrian')
            encabezado.append('Entropy')
            encabezado.append('SHA1')
            encabezado.append('Entropy')
            encabezado.append('SHA256')
            encabezado.append('Entropy')
            encabezado.append('MD5')
            encabezado.append('Entropy')
            lista.append(encabezado)

            for i in mensaje:
                if mensaje!='':
                    lista2=list()
                    aux=HashBrianNasheeeeeeeeee(i)
                    print('HashBrian: '+aux+', entropy: '+str(Entropia(aux)))
                    sha1 = hashlib.new("sha1", bytes(i, encoding = "utf-8"))
                    sha1e=Entropia(sha1.hexdigest())
                    print('SHA1: '+sha1.hexdigest()+', entropy: '+str(sha1e))
                    sha256 = hashlib.new("sha3_256", bytes(i, encoding = "utf-
8"))
                    sha256e=Entropia(sha256.hexdigest())
                    print('SHA256: '+sha256.hexdigest()+', entropy:
'+str(sha256e))
                    md5 = hashlib.new("md5", bytes(i, encoding = "utf-8"))
                    md5e=Entropia(md5.hexdigest())
                    print('MD5: '+md5.hexdigest()+', entropy: '+str(md5e))
                    print('\n')
                    lista2.append(i)
                    lista2.append(aux)
                    lista2.append(str(Entropia(aux)))
                    lista2.append(sha1.hexdigest())
                    lista2.append(sha1e)
                    lista2.append(sha256.hexdigest())
                    lista2.append(sha256e)
                    lista2.append(md5.hexdigest())
                    lista2.append(md5e)
                    lista.append(lista2)
```

```
            with open('comparacion.csv', 'w',encoding='utf-8',newline='') as
file:
                writer = csv.writer(file, delimiter=';')
                writer.writerows(lista)

        elif decision==5:
            print("Ingrese la ruta de su archivo de Texto:\n")
            ruta=str(input())
            f = open (ruta,'r', encoding='utf-8')
            mensaje = f.readlines()
            lista=list()
            encabezado=list()
            encabezado.append('Nombre Hash')
            encabezado.append('Cantidad de Strings')
            encabezado.append('Tiempo en segundos')
            lista.append(encabezado)


            contador=0
            time1=time.time()
            for i in mensaje:
                if mensaje!='':
                    aux=HashBrianNasheeeeeeeeee(i)
                    contador+=1
            time2=time.time()
            lista2=list()
            lista2.append('HashBrian')
            lista2.append(contador)
            lista2.append(str(round(time2-time1,2)).replace('.',','))
            lista.append(lista2)

            contador=0
            time1=time.time()
            for i in mensaje:
                if mensaje!='':
                    sha1 = hashlib.new("sha1", bytes(i, encoding = "utf-8"))
                    contador+=1
            time2=time.time()
            lista2=list()
            lista2.append('SHA1')
            lista2.append(contador)
            lista2.append(str(round(time2-time1,2)).replace('.',','))
            lista.append(lista2)

            contador=0
            time1=time.time()
```

```
            for i in mensaje:
                if mensaje!='':
                    sha256 = hashlib.new("sha3_256", bytes(i, encoding = "utf-
8"))
                    contador+=1
            time2=time.time()
            lista2=list()
            lista2.append('SHA256')
            lista2.append(contador)
            lista2.append(str(round(time2-time1,2)).replace('.',','))
            lista.append(lista2)

            contador=0
            time1=time.time()
            for i in mensaje:
                if mensaje!='':
                    md5 = hashlib.new("md5", bytes(i, encoding = "utf-8"))
                    contador+=1
            time2=time.time()
            lista2=list()
            lista2.append('MD5')
            lista2.append(contador)
            lista2.append(str(round(time2-time1,2)).replace('.',','))
            lista.append(lista2)




            with open('comparaciontiempos.csv', 'w',encoding='utf-
8',newline='') as file:
                writer = csv.writer(file, delimiter=';')
                writer.writerows(lista)

        elif decision==6:
            print("Saludos!")
        else:
            print("Por favor ingrese una opción válida")
```

Luego de crear el código, se solicita hacer una serie de pruebas de rendimiento, para esto es necesario usar la librería hashlib, y de ella se escogen sha1., sha256 y md5 para las comparativas.

La primera es revisar la entropía del hash generado , para esto se ocupa un txt con 10 contraseñas extraído de rockyou.txt .

Luego de correr las 10 contraseñas se extrae la siguiente tabla de resultados:

Cadena	HashBrian	Entropy	SHA1	Entropy	SHA256	Entropy	MD5	Entropy
123456	30322126420212 6i10222	175.0	c4f9375f9834b4e7f0a528	280.0	c99520333c68cd896f58d8	448.0	f447b20a7fcbf53a5d5be0	224.0
password	30322p59w202126i1022s	175.0	c8fed00eb2e87f1cee8e90	280.0	17eded3bf5ab67bb8e3729	448.0	286755fad04869ca52332(	224.0
12345678	30322126520212 6i10223	175.0	9806af3952e1380212b09	280.0	5847c9569c208a37fbb7ee	448.0	23cdc18507b52418db774(	224.0
qwerty	30322q57r202126i1022w	175.0	3c8b9f4b983afa9f644d26	280.0	dd8b296aaabf84ea792d73	448.0	a86850deb2742ec3cb415	224.0
123456789	30322127620212 6i10223	175.0	179c94cf45c6e383baf526	280.0	044a2cfbc4e8143f285228(	448.0	b2cfa4183267af678ea06c	224.0
12345	30322126420212 6i10222	175.0	2672275fe0c456fb671e4f	280.0	f627c8f9355399ef45e1a6b	448.0	d577273ff885c3f84dadb8	224.0
1234	30322125320212 6i10222	175.0	1be168ff837f043bde17c0	280.0	859a7a7603028deeb3b662	448.0	e7df7cd2ca07f4f1ab415d	224.0
111111	30322124120212 6i10221	175.0	3ee88a74d3722b336a69c	280.0	991b4361d768d1dfc3e113	448.0	77a319564621b96fa0656	224.0
1234567	30322126520212 6i10223	175.0	e017693e4a04a59d0b0f4(	280.0	1a39495b12b3f47eb98815	448.0	1b504d3328e16fdf281d1f	224.0
dragon	30322d51g202126n1022r	175.0	af8978b1797b72acfff9595	280.0	1b8cefb8384133f0916cfeb	448.0	8621ffdbc5698829397d97	224.0

Como se puede observar el largo del hash generado por el código es de 25 caracteres, luego de aplicar la fórmula de entropía resulta un número igual a 175 para todas las entradas.

Por otro lado el resto de los hash también arroja la misma entropía para cada una de las entradas, pero en este caso son entropías más altas que las del código presentado en este informe.

Se continúa con las pruebas relacionadas al tiempo, para esto se pide hacer pruebas con 1, 10, 20 y 50 strings del archivo rockyou.txt

Nombre Hash	Cantidad de Strings	Tiempo en segundos
HashBrian	1	2,59737682342529
SHA1	1	0,0000157356262207031
SHA256	1	0,0000061988830566406
MD5	1	0,0000052452087402343
HashBrian	10	18,045471
SHA1	10	0,0000360012054443359
SHA256	10	0,0000240802764892578
MD5	10	0,0000221729278564453
HashBrian	20	19,1109669
SHA1	20	0,0000667572021484375
SHA256	20	0,0000560283660888671
MD5	20	0,0000510215759277343
HashBrian	50	56,2307529
SHA1	50	0,00015712
SHA256	50	0,0001061
MD5	50	0,00010228

Como se puede observar el Hash presentado en el código es el que más lento actúa, mientras que los implementados por librería demoran milésimas de segundo, el presentado en el código demora alrededor de 1,xxx segundos por palabra ingresada, dejando en claro la menor eficiencia que posee el script presentado en este informe.

Finalmente se procede a verificar los puntos iniciales que debe cumplir el código para poder ser aceptado por los dueños de la empresa contratante.

- El número de caracteres final del texto procesado, debe ser fijo y no menor de 25 caracteres. Este número no debe variar, aunque el texto de entrada sea más largo:

  Para una entrada de texto :

  ”hola123hola123hola123hola123hola123hola123hola123” con un largo = 49

  Se arroja el siguiente hash:

  Hashing: 30322h48a2021263102225310, largo=25

- Se requiere que el programa pueda recibir como entrada, tanto un string o texto mediante STDIN, cómo también mediante archivos con múltiples entradas separadas por el salto de línea:

```
Bienvenido al mecanismo de hashing de Brian Nasheeee:
Opción 1: Hashear una cadena de texto
Opción 2: Hashear un archivo de texto
Opción 3: Calcular entropía de una cadena de texto
Opción 4: Comparación entropy vs SHA1,SHA2,MD5
Opción 5: Comparación time vs SHA1,SHA2,MD5
Opción 6: Salir del programa:)
```

- Cualquier ligero cambio al texto de entrada, debe cambiar el resultado final del algoritmo:

  Prueba 1:
  Ingrese su cadena de Texto:
  hola
  Hashing: 30322h54l202126a1022o6314, largo=25

  Prueba 2:
  Ingrese su cadena de Texto:
  hola1
  Hashing: 30322h54l20212611022o5820, largo25

- Opción que sólo calcule la entropía del texto de entrada. En este último caso, debe arrojar mediante STDOUT que entropía posee cada texto de entrada (El formato de la salida a STDOUT, debe mostrar tanto el texto analizado, cómo la entropía calculada separada por un delimitador que permita diferenciar los diferentes campos):

  Ingrese la cadena de Texto:
  hola
  Hash:30322h54l202126a1023o6340, largo:25, La entropía es: 175.0

Por otro lado se solicita que el hash sea resistente a colisiones. Se produce una colisión cuando dos elementos de entrada diferentes dan como resultado el mismo código hash. Aún así, hasta el momento en el que alguna persona encuentre este error, la función seguirá considerándose como 100% segura.

Esto se debe a que se requeriría millones y millones de años para que exista la posibilidad de encontrar una colisión. Un ejemplo de las funciones más resistentes es el SHA-256. Existen diferentes algoritmos SHA, como por ejemplo SHA-0 y SHA-1, que ya no se consideran funciones seguras debido a que se han encontrado en ellos diversas colisiones. Hoy en día, los que agrupa las funciones SHA-2 y SHA-3 son consideradas por el público como las más seguras y resistentes.

Análisis de Resultados:

Como se puede observar, en el código hay diversas variables de temporalidad que le dan un toque de "aleatoriedad" a sectores del hash, la verdad es que el componente que cambia de forma más rápido es el que implica los segundos del datetime, mientras que otra función que le da más versatilidad es la que involucra el valor de la uf en chile a día de hoy.

Luego de observar las tablas de comparación de resultados es posible observar que el hash presentado en el código tiene una menor entropía que el resto de lo hashes importados desde la librería hashlib, esto está relacionado directamente con el largo del hash que sale de la función. Mientras que el largo del Hash es 25, el que posee mayor entropía es el Sha256, lo cuál es de esperarse al saber que es una versión mejorada de los otros 2.

Al arrojar una menor entropía, se podría presumir que el tiempo demorado para expulsar el hash en el código presentado es menor al resto, esto porque debiera tener menor procesamiento matemático pero al observar las tablas se puede observar que esto no es así, ¿por qué?.

Luego de varias pruebas y un análisis del código, se puede llegar a la conclusión de que el elemento que hace lento el hashing no viene de dentro del procesamiento matemático del string, sino que es la función que se conecta a la api para traer el valor de la uf, de igual manera no es un tiempo excesivamente alto, ya que según se define en clases, pasado 4 segundos por palabra, el algoritmo de hash ya no serviría.

Conclusiones:

Luego de trabajar con los distintos algortimos de hashing existentes y crear una especie de algoritmo propio es posible concluir que el hashing posee varias propiedades que le dan un carácter especial a la seguridad de datos.
El hashing es importante para el análisis de diversos datos y su gestión, las funciones hash criptográficas son utilizadas para la seguridad de la información, como por ejemplo, en mensajes o huellas digitales. Asimismo, son una parte fundamental en el minado de Bitcoin, también es esencialmente útil para la tecnología blockchain. Las criptomonedas existen gracias a que el hash condensa grupos de transacciones en bloques.
Además, es verdaderamente útil para procesar archivos de gran tamaño o un conjunto de datos. Esto se debe a que el hash elaborará un output (datos finales) simplificado y único del input (datos iniciales). Esto es muy útil para almacenar grandes cantidades de datos.

El hashing ayuda a transformar diversas palabras, números o cualquier elemento en un hash final que suele ser difícil de descifrar. Esta función no solo ayuda a mantener la seguridad de los datos, sino que también ha ayudado a la creación de monedas virtuales, las cuales mueven una cantidad enorme de dinero.

Antes de utilizarlo, cabe destacar que hay que entender cómo funciona y qué es lo que consigue. Recordar que la codificación de diversos datos está al alcance de las manos, por lo que no hace falta ser un experto en criptografía para codificar lo que se quiera, las librerías son de código abierto y se pueden implementar en cualquier tipo de software.