Universidad Diego Portales

Criptografía y seguridad de redes

# Laboratorio Nº3: Hash

---

*Autores:*
David Pazán
*Profesor:*
*Victor Manriquez*

Fecha: Sábado 18 de Junio

# Índice

# 1. Resumen actividad

Para la presente actividad se solicita crear un *Hash* el cual debe tener las siguientes características:

- Debe tener un largo fijo y no menor de 55 caracteres.
- Se requiere que el programa pueda recibir como entrada, tanto un **String** o texto mediante **STDIN**, cómo archivos y sobre dicho archivo, trabajar línea por línea.
- Cualquier cambio en el input debe dar un resultado distinto, mientras que el ingreso de la misma palabra entregue un mismo hash.
- El procesamiento del texto de entrada debe ser rápido.
- Calcular la entropía de la salida

Una vez realizado, se pide hacer uso de los distintos hash conocidos, MD5, SHA1, SHA256, con la finalidad de establecer una comparativa, tanto para la entropía como para el rendimiento respecto al tiempo en contraste con el nuestro, esto tanto para una entrada ingresada por el usuario, o mediantes archivos de texto, donde se tendrán archivos con 1, 10, 20 y 50 entradas.

# 2. Caso de usos

Un modelo de caso de usos respecto a como funcionaría a grande rasgos el programa.

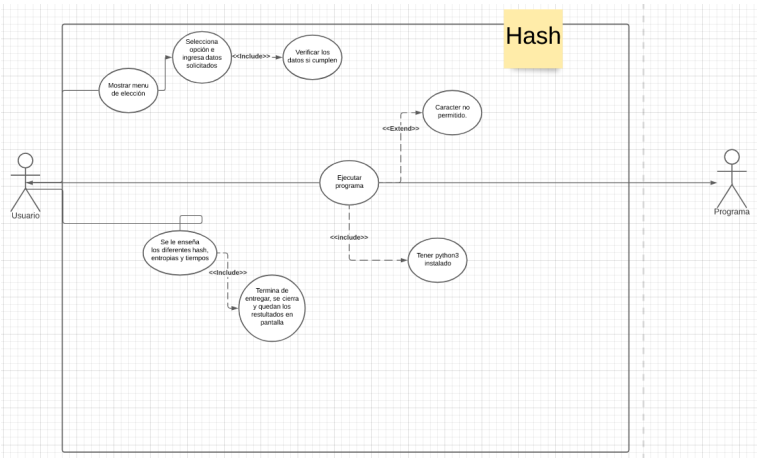


Figura 1: Cálculo entropía.

# 3. Explicación Código

A continuación se procede a explicar parte por parte el código realizado para la actividad.

En primer lugar se importan las librerías y se definen dos diccionarios. Las librerías que son usadas son:

1. **datetime:** De esta libreria se importa *date*, cuya funcionalidad es para dar el valor del día actual y posteriormente usarlo en la operación para hashear.
2. **math:** Encargada para la operación de cálculo de la entropía
3. **time:** Cuya función es usada para realizar la medición de tiempo que tarda en realizar el hasheo.
4. **haslib:** Permite hacer uso de los hash MD5, SHA1, SHA256.

Se parte definiendo una nueva función llamada **Hash**. Esta función pide como parámetro una variable, que se nombra como palabra, a continuación de esto, se definen 2 diccionarios, cuya funcionalidad es la de realizar conversiones entre los pesos y los caracteres usados para la base 74 que se escogió.

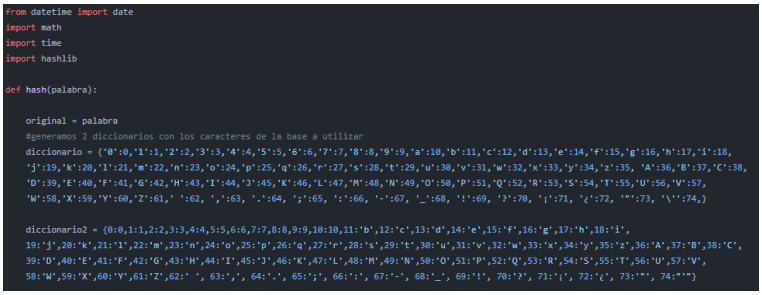


Figura 2: Definición de las librerías y diccionarios

Una vez que pasamos esta definición, entramos a la verificación del primer carácter. Se realiza esta operación mediante un condicional, donde en caso de que el primer valor de la palabra no se encuentre en el diccionario, el cual se comprueba mediante la función *key*, donde toma el *string* y lo pasa a un valor *int* para de esta forma confirmar su existencia. Una vez que se aprueba esta condición se guarda este valor y se mantiene un contador 0, donde estas dos variables han de ser usadas a continuación

```
#Verificamos si el primer caracter se encuentra en el diccionario
#Y verificamos su valor correspondiente
if palabra[0] not in diccionario.keys():
    return -1
primera = diccionario[palabra[0]]
contador = 0
```

Figura 3: Verificación del tamaño y el valor de la primera letra

En esta parte del código se verifica si el modulo de 55 es 0, con la finalida de poder dividirla luego en 55 partes y llegar a obtener un numero entero. En caso de que no sea el caso se agregan dos caracteres al inicio y al final de la palabra, esto siempre y cuando se cumpla la operación del módulo. El carácter a elegir se obtiene de la posición de la primera letra mas el contador del ciclo, es decir todo el procedimiento anterior, las variables que fueron guardadas ahora son utilizadas. Por ultimo se verifica si el numero del carácter está dentro del diccionario, si no es así se ubica la primera posición en el 0 y se calcula otra vez. El *string* que resulta de todo este procedimiento es la variable palabra, que es la usada para el *hash*.

```
#La idea de esta parte de código es verificar si el modulo de 55 es 0
#para poder dividirla luego en 55 tener un numero entero y terminar obteniendo "palabra"
booleano = True
while(booleano):
    if(len(palabra)%55==0):
        booleano = False
    else:
        if (len(palabra)+2)%55==0:
            contador = contador -1
            if (primera+contador) <= 74:
                palabra = palabra + str(diccionario2[primera+contador])
                palabra =  str(diccionario2[primera+contador]) + palabra
            else:
                primera = 0
        else:
            contador = contador + 1
            if (primera+contador) <= 74:
                palabra = palabra + str(diccionario2[primera+contador])
            else:
                primera = 0
```

Figura 4: Verificación del tamaño de la palabra y de las posiciones

A continuación se presentan las variables que han de guardar los nuevos largos y las divisiones mencionadas en el punto anterior, además guardamos el día actual, teniendo un valor entre [1-31] y le sumamos 5, muy parecido al

cifrado ROT conocido. Esta variable se usan en la siguiente sección.

```
#tomamos la nueva palabra, sus divisiones en 55 y el dia actual haciendo un corrimiento con este
guardarLargo = int(len(palabra))
division = int(guardarLargo/55)
diaHoy = date.today()
diaHoy = diaHoy.day + 5
```

Figura 5: Variables para realizar el corrimiento

Ahora se crea un arreglo que ha de tener los pesos de cada carácter, a dicho peso se le suma la variable anteriormente creada *diaHoy* y el contador inicializado, de esta forma se genera un corrimiento para los pesos. Además como se ve, se aprecian tres variables nuevas, las cuales son usadas para guardar un contador de la división que se efectuará a continuación, la palabra salida, que es el hash entregado y un nuevo contador.

```
#En la siguiente sección se crea un arreglo con los pesos de cada carácter sumado con el contador y la variable diaHoy.
Arr_Pesos = []
count = 1
for i in palabra:
    Arr_Pesos.append(int(diccionario.get(i))+diaHoy+count)
    count = count + 1

#Una vez que se generan los pesos del arreglo, se procede a convertilo en palabra
contadordivision = 0
salida = ""
suma = 0
```

Figura 6: Operación de corrimientos y variables auxiliares

En la siguiente sección se suman cada *contadordivision* obteniendo un valor que es transformado para que se encuentre en el diccionario, esta transformación es volver a la posición 0 cada vez que se pasen de 74 caracteres del diccionario. Ya teniendo la posición en los valores establecidos se guarda en la variable salida y se repite el ciclo. Por último se procede a imprimir la salida y la entropía de la palabra.

```
#Los números se convierten en letras, de esta forma se ajusta al hash
for i in range(guardarLargo):
    if(contadordivision == division-1):
        suma = suma + Arr_Pesos[i]
        contadordivision = 0

        if(suma<=74):
            salida = salida + str(diccionario2.get(suma))
        else:
            flag = True
            num = suma
            while(flag):
                if(num <= 74):
                    flag = False
                    salida = salida + str(diccionario2.get(num))
                    num = suma

                else:
                    num = num - 74

    else:
        suma = suma + Arr_Pesos[i]
        contadordivision = contadordivision + 1


print("El hash de: ",original," es: ")
print(salida)
entropia(salida)
```

Figura 7: Generación del hash de salida

El cálculo de la entropía se realiza mediante la fórmula vista en clases, cuya explicación es el largo de la salida por el logaritmo en base 2 de la base usada (74).

```
#la entropia se calcula como el largo de la palabra ingresada por el logaritmo en base 2 de la base utilizada:
def entropia(palabra):
    entropia =55*math.log(74,2)

    print("La entropia de ",palabra," es: ", entropia)
```

Figura 8: Cálculo entropía Hash Laboratorio.

Para obtener el cálculo del tiempo, se realiza una diferencia entre el tiempo de inicio y final, del momento que se aplica el hash sobre la palabra ingresada.

```
def tiempo(palabra):
    inicio = time.time()
    hash(palabra)
    fin = time.time()
    print("El tiempo de ejecución HASH LABORATORIO es: ", fin-inicio, "\n")
```

Figura 9: Cálculo entropía.

Se ha de explicar uno solo de las funciones respecto a MD5, SHA1, SHA256, debio a que todas tienen la misma lógica. Para la obtención de estos hashes, se recuerda la libreria importada hashlib, hablada en un principio, esta aplica MD5 a la palabra o en su defecto los otros dos hash mencionados, lo único que ha de variar respecto a estos tres hash son la entropía, donde solo se diferencia en el largo, por ejemplo MD5 tiene largo 32, SHA1 con un largo de 40 y SHA256 un largo de 64. Por otro lado todos hacen uso de la base de 16.

```
def md5xd(palabra):
    md5 = hashlib.md5(palabra.encode())
    entropia = 32*math.log(16,2)
    print("El hash MD5 de: ",palabra," es: ")
    print(md5.hexdigest())
    print("La entropia de MD5",palabra," es: ", entropia,"\n")
```

Figura 10: Hash MD5

Para los archivos se solicita que ingrese el nombre del archivo, y en este se realiza el cálculo del tiempo instantáneamente, se abre el archivo y se va leyendo linea por linea hasta que se acaben. Se aplica el hash elegido, resaltando la linea 8 de la imagen, donde posteriormente se realiza para los otros tres hashes solicitados, mediante otras funciones que se encuentran posterior a esta, y se entrega el hash y tiempo al momento de imprimir.

```python
#al recibir el nombre lo convertimos como un string y lo pasamos al hash
def archivo2(palabra):
    inicio = time.time()
    f = open (palabra,'r')
    while(True):
        linea = f.readline()
        if not linea:
            break
        print(linea)
        hash(linea.rstrip())
    f.close()
    fin = time.time()
    print("Tiempo que demoro el hash laboratorio: ", fin-inicio,"\n")
```

Figura 11: Para los archivos

# 4. Pruebas de rendimiento en ComparaciónLAB

## 4.1. Análisis prueba Entropía

Como se puede observar gracias a la tabla anterior obtenida, la entropía que se presenta para los distintos *Hash* estudiados, el realizado en esta experiencia presenta una gran diferencia con respecto a los otros, dando a entender que ofrecemos mayor seguridad. Mientras que en los usados por la comunidad, el SHA256 presenta mayor entropía con respecto a sus pares, debido al largo del hash. Por lo cual, se saca la conclusión de que se debe alargar los hash, para obtener una mejor entropia.
De esto también se puede indicar que no porque la entropía sea alta el rendimiento igual va a ser bueno, por como se ha de poder observar en la tabla de a continuación para el rendimiento de exigencia.

## 4.2. Análisis prueba tiempo

Como se aprecia respecto a los tiempos, el hash realizado para la experiencia, siempre tuvo mayor tiempo con respecto a la competencia. Una de las ideas del porque ocurre esto, puede ser debido al gran diccionario que hace uso, recordando que en la tabla anterior se presenta este dato, haciendo operaciones de validación sobre los caracteres definidos y que son posibles de usar.

# 5. Conclusión final

Claramente los hash son mas conocidos y mas utilizados debido a su entropía y velocidad debido a la conversión que tienen y trabajan con bytes con funciones insertas en Python, en cambio el hash creado trabaja con operaciones mas lentas que los cambios de bytes como multiplicaciones y ciclos anidados además que el hash propio trabaja con los valores de conversión entre números y caracteres que vienen de un diccionario ya creado comparando cifras, lo que se traduce en más tiempo para la ejecución, por lo que es recomendable utilizar SHA256 que es la de mejor rendimiento

GitHub