udp Escuela de Informática y Telecomunicaciones

Universidad Diego Portales

Criptografía y Seguridad en Redes

Laboratorio 3 - Hash

Autor:
Diego Esperidión

Profesor: Victor Manriquez

17 de Junio de 2022

Índice

1. Introducción

EN este informe se realizaran las comparaciones de los siguientes métodos de hashing con respecto al algoritmo creado por el autor del informe, el cual se basa en la integridad de los datos, estos son:

1. SHA1
2. SHA256
3. MD5
4. Hash-DiegoEsp

El proceso de análisis será a través de tablas comparativas entre los 4 métodos comparando el tiempo de demora en cada uno de los procesos y la entropía obtenida de cada uno, sacando una conclusión sobre cual de todos los métodos es el mas efectivo para encriptar datos.

La fórmula que se usara para la comparativa es la siguiente:

$$E = L * \log_2 N \tag{1}$$

Donde:

1. E = entropía
2. L = largo del texto
3. N = Base del algoritmo

2. Desarrollo

Antes de empezar a comparar vale recordar que todo lo que se habla a continuación también se encuentra disponible en el github, cuyo enlace es:

https://github.com/DiegoEsperidion/Laboratorio$_{3C}riptografia$

El algoritmo hash se realiza mediante el siguiente esquema:

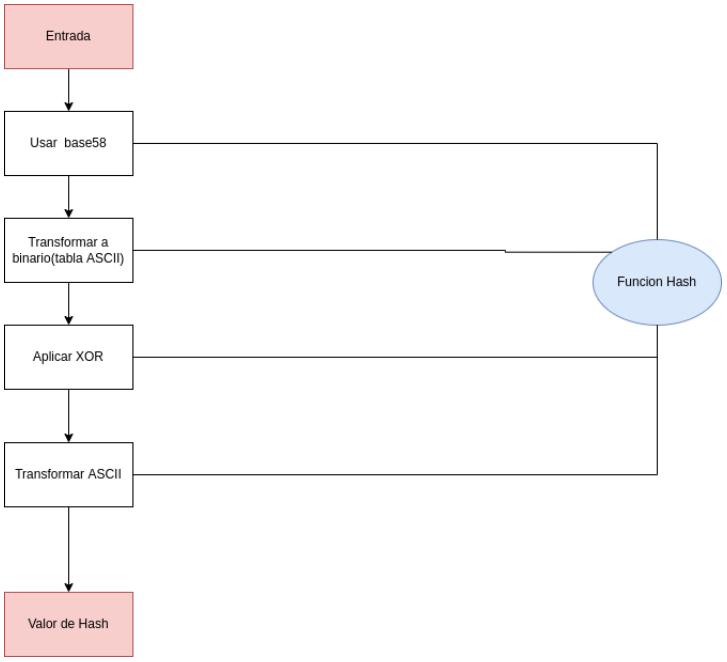

Figura 1: Proceso de hash

El proceso para realizar el hash se realiza de la siguiente forma:

1. Selecciona un texto de entrada, el cual puede ser cualquier dígito que este implementado en el teclado , el programa lo transformara posteriormente a base58, en el caso que sea un archivo de texto, el programa tiene la función de leerlo linea por linea para extraer los datos.
2. Las palabras deben ser iguales a 40, en caso contrario el programa se encarga de rellenar con una secuencia definida para cumplir con el requisito (de esta forma al pasarlo a base58 nos entrega un valor exacto de 55 caracteres).
3. Una vez tenemos la base, transformamos el texto plano en binario
4. Se aplica un XOR en base a la password y una llave que es definida en el código (1011101).

5. Transformar en ASCII para retornar finalmente el valor de Hash (el valor retorna en forma byte, por lo que finalmente se hace conversión de bytes a string para así calcular su entropía.)

2.1. Funciones

2.1.1. Librerías utilizadas

```
import base58 # Se usa la libreria base58 para convertir el texto plano en base58
import binascii #libreria para convertir el XOR obtenido en binario en ASCII
import hashlib #Libreria para encriptar con md5,sha1 y sha256
from math import log #Libreria para calcular la entropia
#Ingresamos la palabra a codificar en base58
```

Figura 2: Librería

Se utilizaron 3 librerías para el funcionamiento del hash, las siguientes son:

1. **base58:** Útil para transformar del string a base58
2. **binascii**: Librería para transformar el binario a ASCII
3. **hashlib**: librería para poder realizar los hash de md5, sha1 y sha256
4. **math:** para realizar las operaciones matemáticas (entropía)

2.1.2. Transformar a base58

```
#Transformar a Base58
def b58(hash):
    a = base58.b58encode(hash.encode()).decode()
    return a
```

Figura 3: texto a base58

2.1.3. rellenar palabra

```python
def rellenar(palabra,pos):
    #Esta funcion rellena una palabra menor a 55 caracteres con los
    # caracteres contenidos en salt, se termina al tener 55 caracteres.
    salt=["1","A","a","B","b","C","c"]
    if pos == len(salt):
        pos=0
    if len(palabra) <40:
        pal=palabra+salt[pos]
        pos=pos+1
        pal=rellenar(pal,pos)
        return pal
    else:
        return palabra
```

Figura 4: rellenar

La función se encarga de rellenar el string hasta 40, esto mediante un patrón de caracteres que es definido en la función (1AaBbCc), si el string llega a los 40 sale de la función, esto es con el objetivo de que al hashear en base58 el string se expanda a un total exacto de 55 caracteres.

2.1.4. Base58 a binario

```python
#Transformar de Base58 a Binario

def toBinary(a):
    l,m =[],[]
    for i in a:
        l.append(ord(i))
    for i in l:
        m.append(int(bin(i)[2:]))
    return m
```

Figura 5: base58 a binario

Transforma la base58 a binario

2.1.5. Implementación de XOR

```
# Usar XOR
def ConvertXOR(arr):
    c = []
    key="1011101"
    print("llave: ", key)
    for i in range(len(arr)):
        a = str(arr[i])
        c.append(bin(xor(a,key)))
    print("Aplicando XOR a los binarios obtenidos con la llave:  ",c)
    return c

def xor(a,b):
    y = int(a,2)^int(b,2)
    format(y,"b")
    return y
```

Figura 6: Uso de XOR

LA funcion que llama al binario para transformarlo a un XOR, se aprecia la llave mencionada anteriormente (1011101) que nos servirá para ejecutar un XOR de forma correcta.

2.1.6. Convertir de lo obtenido al XOR a ASCII y ajustar tamaño

```
#Transforma el binario de XOR a ASCII
def ConvertToASCII(X):
    arr  = b""
    for i in range(len(X)):
        c=X[i][2:]
        c=int(c,2)
        c=iterar(c)
        bi= binascii.unhexlify('%x' % c)
        arr = arr + bi
    return toString(arr)
#Ajusta el tamaño del hash a 55 caracteres como minimo
def tamaño(tam,hash):
    if(tam<55):
        hash = rellenar(hash,0)
        print(hash)
        return hash
```

Figura 7: XOR a ASCII

La función llama al binario obtenido del XOR para transformarlo a ASCII, para que se ejecute perfectamente se implementaron 2 funciones que son las que se verán a continuación

2.1.7. Iteración y transformar bytes en string

```
#Iterar el valor de c para satisfacer el requerimiento de la funcion unhexlify
def iterar(c):
    while(c<32):
        c=c+c/2

    while(c>126):
        c=c-c/2
    return int(c)

#Transforma los bytes en un string
def toString(arr):
    return ''.join(map(chr, arr))
```

Figura 8: Iterar y transformar bytes a string

La función iterar esta hecho con el fin de que los caracteres obtenidos satisfacen la tabla ASCII y que sean leíbles por el ordenador, esto es decir por ejemplo si nos hubiese entregado un valor igual a 30, seguramente el ordenador nos mande un error al no reconocer el carácter, por lo tanto para que esto no ocurra le sumamos su mitad y así obtenemos un carácter que si sea leído por el sistema operativo.

LA segunda función fue creada solamente para transformar el hash final de formato byte a string, ya que de esta manera resulta mas sencillo manipular los datos para poder calcular su entropía.

2.1.8. Inicio del programa

```
#Comienza el programa
def Iniciar(palabra):
    tam = len(palabra)
    palabra = tamaño(tam,palabra)
    a = b58(palabra)
    print("Mensaje codificado a base58: " + a)
    print("")
    print("Base58 a binario: ")
    print(toBinary(a))
    print("")
    arr = toBinary(a)
    X = ConvertXOR(arr)
    print("")
    pf=ConvertToASCII(X)
    print("Hash final en ASCII: ", pf)
    print("Largo del hash: ",len(pf))
    return pf
```

Figura 9: Iniciar Programa

Ésta es la función donde se llama a cada palabra del texto a realizar su correspondiente hash

2.1.9. Leer archivo

```python
#Leer un archivo .txt y extraer una palabra por linea en formato de texto plano
def leerArchivo(a):
    archivo = open(a,"r")
    palabras = []
    for linea in archivo:
        print("Palabra n° ", len(palabras)+1)
        print(linea)
        pf = Iniciar(linea)
        entropia(pf)
        print("============================================================")
        palabras.append(linea)
    return pf


def leerArchivosinImprimir(a):
    archivo = open(a,"r")
    palabras = []
    for linea in archivo:
        palabras.append(linea)
    return palabras
```

Figura 10: Lectura de textos (.txt)

Básicamente, es el encargado de leer los archivos.txt correspondientes, es donde se va palabra por palabra para calcular los hash, además de calcular la entropía.

2.1.10. Algoritmo de hash ya definidos

```python
#Calculo de hash de palabras en md5,sha1 y sha256
def md5(palabras):
    for word in palabras:
        hash = hashlib.md5(word.encode())
        print("Hash de la palabra en md5: ", hash.hexdigest())
        entropiahash(hash.hexdigest())
        print("============================================================")
    return hash.hexdigest()

def sha1(palabras):
    for word in palabras:
        hash = hashlib.sha1(word.encode())
        print("Hash de la palabra en sha1: ", hash.hexdigest())
        entropiahash(hash.hexdigest())
        print("============================================================")
    return hash.hexdigest()

def sha256(palabras):
    for word in palabras:
        hash = hashlib.sha256(word.encode())
        print("Hash de la palabra en sha256: ", hash.hexdigest())
        entropiahash(hash.hexdigest())
        print("============================================================")
    return hash.hexdigest()
```

Figura 11: MD5 , SHA1, SHA256

DIrectamente calcula el hash de los algoritmos de hashing md5, sha1 y sha256

```python
#Calcular md5 de solo 1 palabra
def md5solo(palabra):
    hash = hashlib.md5(palabra.encode())
    print("Hash de la palabra en md5: ", hash.hexdigest())
    return hash.hexdigest()

#Lo mismo con sha1 y sha256
def sha1solo(palabra):
    hash = hashlib.sha1(palabra.encode())
    print("Hash de la palabra en sha1: ", hash.hexdigest())
    return hash.hexdigest()

def sha256solo(palabra):
    hash = hashlib.sha256(palabra.encode())
    print("Hash de la palabra en sha256: ", hash.hexdigest())
    return hash.hexdigest()
```

Figura 12: MD5 , SHA1, SHA256 (sin imprimir)

(Esta función se uso para calcular la entropía de forma manual, sin el uso de archivos)

2.1.11. Entropía

```python
def entropia(palabra):
    L = len(palabra)
    N = 94
    H = L*log(N,2)
    print("Entropia del hash: ", H)
    print("")
    print("")
    return H

def entropiahash(palabra):
    L = len(palabra)
    N = 36
    H = L*log(N,2)
    print("Entropia del hash: ", H)
    print("")
    print("")
    return H
```

Figura 13: Entropía

Función que , como bien dice el nombre, se encarga de calcular la entropía tanto del hash implementados como los demás

Todo el proceso se va imprimiendo en consola, de manera que el usuario tenga una mejor visualización de como se esta haciendo todo el proceso de hash.

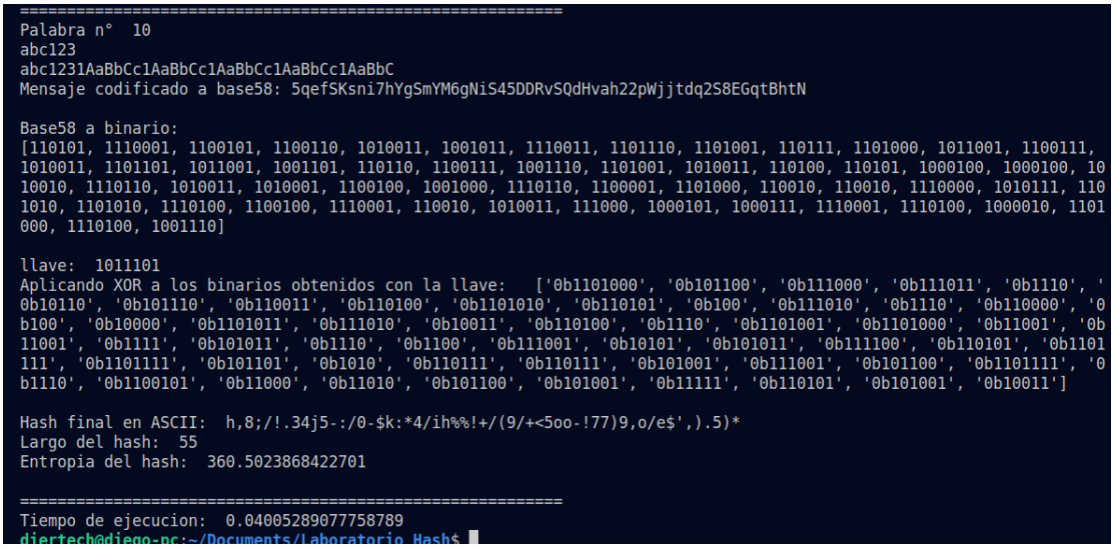

Figura 14: Ejemplo en consola de hash

3. Análisis

3.1. Tabla Tiempo

A continuación se presenta los registros obtenidos del tiempo de demora en realizar los algoritmos de cada texto

De ahora en adelante se llamara hash.yo al hash creado por el usuario.

IMPORTANTE: El tiempo puede variar según el momento que ejecute el code y el rendimiento del ordenador, todos los registros fueron realizados el mismo día.

txt/Hash	Hash.yo	MD5	SHA1	SHA256
1.txt	0.004149436950683594	0.0006771087646484375	0.0008707046508789062	0.0002727508544921875
10.txt	0.0225222110748291	0.0015511512756347656	0.0016777515411376953	0.002208709716796875
20.txt	0.049103498458862305	0.0021021366119384766	0.0005826950073242188	0.004169940948486328
50.txt	0.07008481025695801	0.007513284683227539	0.0022487640380859375	0.007145881652832031
Promedio	0,03646498919	0,002960920334	0,001344978809	0,003449320793

Cuadro 1: Tiempo de demora

Se puede apreciar que de todos los programas de hashing, el SHA1 es el de los mas rapido en realizar el proceso de hasheo, por otra parte el hash.yo ha sido el que mas tiempo se ha

demorado en realizar todo el proceso de hasheo, por lo tanto si es que se busca rapidez en procesos de hasheo quizás el creado no sea el mas apto para ello, en ese caso el ideal para este ocasión seria el SHA1. La posible razón del rendimiento sea debido a que presenta una mayor cantidad de pasos para hashear un dato que si fuera otro de los mencionados.Además de que al aplicar base58 el string se expande (y se exige un largo igual a 55, por lo que si no llega se debe incrementar) y eso conlleva a encriptar mayor cantidad de datos por muy pequeño que sea el string de entrada.

3.2. Tabla Entropía

E/Hash	Hash.yo	MD5	SHA1	SHA256
Entropia	360.5023868422701	165.437600046154	206.7970000576925	330.875200092308

Cuadro 2: Entropia

Sin embargo, si nos fijamos en la entropía el que se corona para esta ocasión es Hash.yo, por lo tanto si lo que se busca es la mayor variedad en cuanto a cifrado de información, éste seria ideal para su uso, y el que se encuentra por debajo es MD5. por lo que no es el mas seguro a comparación con el creado. Las posibles razones de estos datos son por las bases de cada uno, mientras que md5, sha1 y sha256 abarcan 36 caracteres, el hash.yo abarca 94 caracteres, esto quiere decir que la dificultad aumenta casi al triple al tener una mayor cantidad de combinaciones. Otra cosa puede ser porque al ser un largo fijo de 55 caracteres, representa a uno de los algoritmos mas largos junto con sha256.

4. Conclusión

El hash.yo fue creado para la integración de los datos, se a visto todo el proceso desde su entrada en forma de base58 a su salida en formato ASCII y luego se comparo con los otros sistemas de hash, de acuerdo a lo que se vio en el análisis se puede decir que el algoritmo creado tiene una muy buena seguridad debido a que su entropía es bastante alta, sin embargo si lo que se busca es una rapidez del hasheo quizás lo mas apto seria aplicar sha1 ya que a pesar de igual ser rápido, el algoritmo sha1 es bastante mas. No es un algoritmo tan lento como para compararlo con bcrypt ya que hash.yo no llega ni al segundo de tiempo de demora, pero si es similar al uso de scrypt ya que utiliza la misma base de encriptación para criptomonedas, solo que ahora se usa para encriptar datos, y esto se puede apreciar en la consola en la sección "mensaje codificado a base58"(Figura 14).