Laboratorio Hash Criptografía

Diego Carrillo

Junio 2022

Índice

1. Introducción

En el presente informe, se explicaran los pasos a seguir para desarrollar el laboratorio numero 3 correspondiente al curso de criptografía.En donde, se desarrollará un cifrado creado por el alumno.

Este cifrado corresponderá a crear un hash para integridad, en donde se "codificarán" todos los inputs que sean ingresados.

2. Hito I

En primera instancia, se debe de entregar un repositorio en donde se encontraran todos los avances correspondientes en conjunto del código final.

Repositorio Github @Carro1331

En donde, se toma le decisión de crear un algoritmo para realizar integridad de datos.

3. Hito II

Una vez escogido el caso de uso de este algoritmo, se procede a desarrollar el código. Para el cual, se escoge trabajar con Python3, ya que es el lenguaje de programación que se ha utilizado, en su mayoría, durante este curso y a su vez, es el lenguaje mas dominado por el estudiante.

Cuando fue el momento de empezar a desarrollar este algoritmo, se empezó teniendo una idea de tomar el string o el texto que fuese recibido por consola o txt y se realizaría un juego de números binarios en conjunto con tablas ASCII. Con las pruebas que se fueron realizando y las situaciones que se fueron analizando, se decide no trabajar con binario, y simplemente realizar un "codificado" mediante el uso de tablas ASCII o en otras palabras, se utilizo y creo un hasheo mediante unicode(utf-8).

Una vez claro lo mencionado anteriormente, se empieza con la explicación correspondiente a la actividad. Las funciones se irán explicando según el orden que se mantengan en el código, y no, por el orden que fueron desarrolladas.

3.1. getUnicode()

En esta sección, se crea un archivo, donde se registraron los primeros 500 caracteres que pueden ser creados mediantes las funciones de unicode.Para esto, se creó un ciclo de 500 datos en donde se transformó el numero en carácter.

Esto fue realizado, para poder descubrir de forma manual, cuales fueron los caracteres que producían errores o bien, no eran compatibles del todo con visual estudio code o python3.

```
1  def getUnicode(): # type 0 if pass without has || type 1 for hash.
2      arch = open('unicode.txt','w')
3      x = 0
4      while(x < 500):
5          arch.write("Numero:")
6          arch.write(str(x))
7          arch.write("- Caracter: ")
8          arch.write(str(chr(x)))
9          arch.write("\n")
10         x = x + 1
11     arch.close()
```

Figura 1: getUnicode

3.2. getASCII(symbol)

Esta función, toma el ord() que retorna el valor de un carácter dentro de la tabla ASCII, esta se creó para tener una mayor claridad al momento de utilizar estas funciones, ya que fueron la base para este.

```
def getASCII(symbol):
    return ord(symbol)
```

Figura 2: getASCII

3.3. encode(string)

Esta, es la función principal de este programa, en donde se realiza todo el supuesto "hash". Para esta función se recibirá un string, la cual simularía la supuesta contraseña, a esta, se le obtendrá el largo en primera instancia para posteriormente ir trabajando con estos valores.

Primero, es necesario definir un arreglo con todos los posibles caracteres que no son aceptados por Python o Visual Estudio Code, como se mencionó en las funciones anteriores. Estos valores, se guardan en el arreglo para posteriormente, ir revisando si los caracteres resultantes son o no validos.

```
def encode(string):
    size = int(len(string))

    #Listado de caracteres que no es reconocido por vsc o python. SE PROBÓ UNO POR UNO
    Null=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,
        28,29,30,31,32,33,34,60,127,128,129,143,140,145,146,147,148,149,150,151,152,153,154,155,156,157,158
        ,159,160,161,162,175,183,186,217,305,385,390,399,424,425,441,446,447,450,453]
    hash = ""
    cont = 0
```

Figura 3: encode_List

Una vez realizado el arreglo para evitar los errores, se crean las variables donde se irá guardando el resultado final de este "codificado" creado, que tomara el nombre de **Carro**.

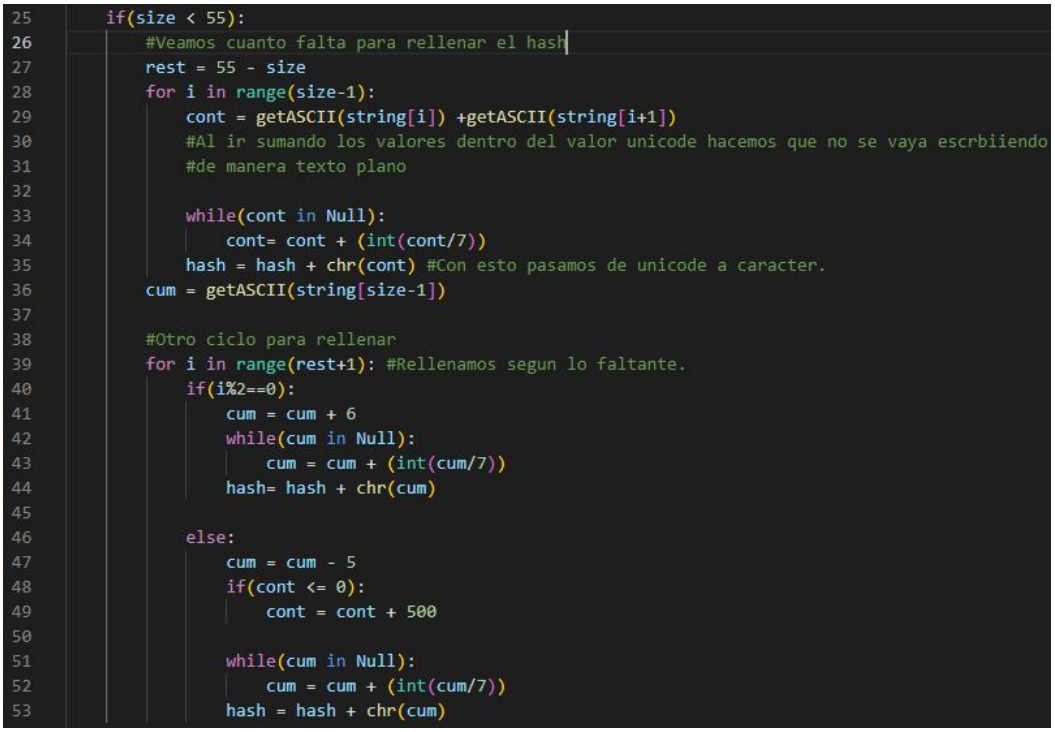
Se empieza tomando una par de condicionales, ya que el hash que se deberá crear tiene que tener un largo de 55 caracteres, ni mas ni menos. Es por esto, que apenas se reciben los caracteres que el usuario desee ingresar se revisara si es mayor o no al largo deseado.

Si el largo del dato recibido es menor a 55 se procede a ingresar a la condición menor a 55. En donde, se calcula, cuantos son los datos necesarios para llegar a cumplir el largo del hash deseado.

Una vez obtenido este valor, se empiezan a codificar los caracteres que se ingresaron. Esto mediante un ciclo que tendrá una variable **cont** para ir guardando los valores de la tabla ASCII y se irán sumando con el valor subsecuente, de esta manera se irán codificando los datos.

Obtenidos estos datos codificados, se procede a revisar si estos valores se encuentran dentro de la lista creada al comienzo de la función se procede a cambiar este valor sumándole una porción de su propio valor.

Como se estuvo trabajando, en primera instancia, con los caracteres que fueron recibidos, es necesario empezar a rellenar los datos restantes, y para esto se fue crearon patrones como ir sumando dígitos dentro de una variable, que se convertirá pasará a la tabla ASCII para luego ir agregándola al hash final.



```
25      if(size < 55):
26          #Veamos cuanto falta para rellenar el hash
27          rest = 55 - size
28          for i in range(size-1):
29              cont = getASCII(string[i]) +getASCII(string[i+1])
30              #Al ir sumando los valores dentro del valor unicode hacemos que no se vaya escrbiiendo
31              #de manera texto plano
32
33              while(cont in Null):
34                  cont= cont + (int(cont/7))
35              hash = hash + chr(cont) #Con esto pasamos de unicode a caracter.
36          cum = getASCII(string[size-1])
37
38          #Otro ciclo para rellenar
39          for i in range(rest+1): #Rellenamos segun lo faltante.
40              if(i%2==0):
41                  cum = cum + 6
42                  while(cum in Null):
43                      cum = cum + (int(cum/7))
44                  hash= hash + chr(cum)
45
46              else:
47                  cum = cum - 5
48                  if(cont <= 0):
49                      cont = cont + 500
50
51                  while(cum in Null):
52                      cum = cum + (int(cum/7))
53                  hash = hash + chr(cum)
```

Figura 4: if menor 55

Terminada esta condicional, existe el caso contrario, en donde la cantidad de caracteres entregados por el usuario fueron mayores al hash que se realizara. Para esto, es necesario ver primero el largo que estamos trabajando para luego ir seccionándolo, este dividirá en 55 partes donde luego se multiplicará en 55 para tener un estimado de cuantas partes fueron creadas y así estimar un nuevo largo.

Se recorrerá el largo estimado y se empezara a crear el hash. en donde existirá una serie de condiciones dentro de este ciclo. Por ejemplo, si la posición que estoy revisando, su siguiente es un espacio vacío se codificara de una manera. En cambio, si no es un extremo, se codificará igual que en la parte anterior, tomando el valor de la tabla ASCII de una posición y sumándole la siguiente. Todo esto, para las sub partes creadas o a analizar en donde se ira concatenando los valores al hash final.

En caso contrario, se tomara los valores de la tabla ASCII para la posición que estoy investigando sumada a la siguiente y la resta de la mitad del sucesor. Para finalmente volver a concatenar al hash los resultados obtenidos.

Finalizadas las condiciones, se procede a retornar la palabra hasheada.

```
25      if(size < 55):
26          #Veamos cuanto falta para rellenar el hash
27          rest = 55 - size
28          for i in range(size-1):
29              cont = getASCII(string[i]) +getASCII(string[i+1])
30              #Al ir sumando los valores dentro del valor unicode hacemos que no se vaya escrbiiendo
31              #de manera texto plano
32
33              while(cont in Null):
34                  cont= cont + (int(cont/7))
35              hash = hash + chr(cont) #Con esto pasamos de unicode a caracter.
36          cum = getASCII(string[size-1])
37
38          #Otro ciclo para rellenar
39          for i in range(rest+1): #Rellenamos segun lo faltante.
40              if(i%2==0):
41                  cum = cum + 6
42                  while(cum in Null):
43                      cum = cum + (int(cum/7))
44                  hash= hash + chr(cum)
45
46              else:
47                  cum = cum - 5
48                  if(cont <= 0):
49                      cont = cont + 500
50
51                  while(cum in Null):
52                      cum = cum + (int(cum/7))
53                  hash = hash + chr(cum)
```

Figura 5: ifmenor55

3.4. writePass(text)

Esta función fue creada para poder realizar pruebas y generar un registro de los cambios que se fueron creando.

```
81  def writePass(text): # type 0 if pass without has || type 1 for hash.
82      with open('Resultados.txt','a', encoding="utf-8") as arch:
83          arch.write('Contraseña: '+ text + '\n')
84          hash = encode(text)
85          arch.write('\t'+'Hash: '+ hash  + '\n' )
86          arch.close()
```

Figura 6: writePass

3.5. entropía()

Al momento de ver la integridad de datos, se puede ver que tan segura puede llegar a ser una contraseña mediante la entropía que puede llegar a poseer esta contraseña. Es por esto que se procede a calcular la entropía.

Primero, es importante saber que para calcular la entropía es necesario saber una serie de datos:

- El largo de la contraseña
- La base del abecedario utilizado

Una vez obtenidos estos valores, se crearon condiciones para los casos que se deberá calcular, esto significa crear una condición para cada uno de los casos:

- MD5
- SHA1
- SHA256
- Carro

```
90  def entropia(string,type):
91      size = int(len(string))
92      trpy = 0
93      base = 0
94      if(type == 'md5'):
95          base = 32
96      elif(type == 'sha1'):
97          base = 40
98      elif(type == 'sha256'):
99          base = 64
100     else:
101         base = 1114112
102     trpy = size * (mat.log(base,10))
103     return trpy
```

Figura 7: Entropía

4. Hash

En este apartado, se crearon las funciones para obtener los resultados de un hash para cada tipo que sea solicitado, los tipos solicitados fueron los siguientes:

- MD5
- SHA1
- SHA256

Estas funciones, retornan de manera inmediata los hash que se soliciten.

```
108  def md5(string):
109      code  = hash.md5(string.encode())
110      ncode = code.hexdigest()
111      return ncode
112
113  def sha1(string):
114      code = hash.sha1(string.encode())
115      ncode = code.hexdigest()
116      return ncode
117
118  def sha256(string):
119      code = hash.sha256(string.encode())
120      ncode = code.hexdigest()
121      return ncode
```

Figura 8: Hashs

4.1. leerdatos()

Como para esta actividad, es necesario leer archivos de texto, en donde se reciben las contraseñas a encriptar, fue necesario crear esta función en donde retornará los datos que se encuentran en el texto.

```
123  def leerDatos():
124
125      arc = open("Rockyou.txt")
126      lineas = arc.readlines()
127      datos = list()
128
129      for lineas in lineas:
130          datos.append(lineas)
131      return datos
```

Figura 9: leerdatos

4.2. Main

En esta función, se llamaran todas las funciones que son creadas para luego obtener estos resultados y ponerlos a prueba. Para esto, se empieza creando un ciclo infinito en conjunto de un menú en donde el usuario que quiera utilizar este código, podrá escoger cuantas palabras quiere encriptar, para posteriormente ir recibiendo los valores de la entropía y los tiempos de ejecución.

Las opciones para trabajar con este programa es trabajar con 1, 10, 20 o 50 palabras a encriptar. En donde se empezarán a agregar a una lista enlazada las palabras que se desean utilizar. Para luego, empezar a recorrer esta lista mientras se están tomando los tiempos por cada encriptado que se esta realizando. Y los tiempos se irán guardando dentro de listas para luego mostrarlas por pantalla.

```
142 def main():
143     val = True
144     entro = list()
145 
146     while(val):
147         base = list()
148         tiempo = list()
149 
150         print("Bienvenido.")
151         print("¿Cuantos datos del Rockyou quiere compar
152         print("Ingrese la opcion que desea. (1 ,2 ,3, 4
153         print("1.- 1 ")
154         print("2.- 10 ")
155         print("3.- 20 ")
156         print("4.- 50 ")
157         entrada = input("5.- Salir\n")
158         data = fixArr()
159         if(entrada == '1'):
160             base.append(data[0])
161 
162         elif(entrada == '2' ):
163             for i in range(0,10):
164                 base.append(data[i])
165 
166         elif(entrada == '3' ):
167             for i in range(0,20):
168                 base.append(data[i])
169 
170         elif(entrada == '4' ):
171             for i in range(0,50):
172                 base.append(data[i])
173 
174         elif(entrada == '5'):
175             break
```

Figura 10: main1

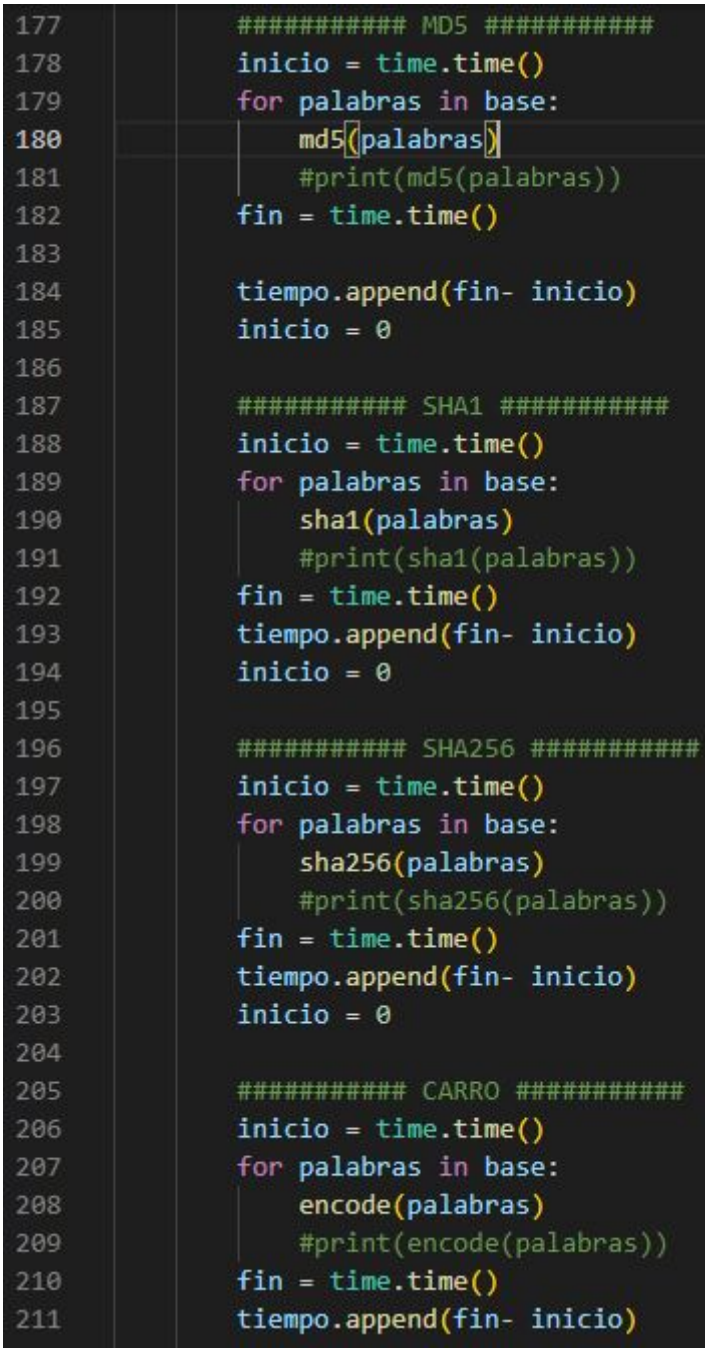


Figura 11: tiempos

```
213         entro.append(entropia(md5(base[0]),'md5'))
214         entro.append(entropia(sha1(base[0]),'sha1'))
215         entro.append(entropia(sha256(base[0]),'sha256'))
216         entro.append(entropia(encode(base[0]),'carro'))
217 
218         print(base)
219         print("Los tiempos para realizar el hash fueron los siguientes:")
220         print("MD5 :","{:.10f}".format(tiempo[0]))
221         print("SHA1 :","{:.10f}".format(tiempo[1]))
222         print("SHA256 :","{:.10f}".format(tiempo[2]))
223         print("Carro :","{:.10f}".format(tiempo[3]))
224         print("\nEntropias: ")
225         print("MD5 :",entro[0])
226         print("SHA1 :",entro[1])
227         print("SHA256 :",entro[2])
228         print("Carro's :",entro[3])
```

Figura 12: output