PRÁCTICA 5 PUZZLES HASH

1. Usaremos la función hash sha256, y crearemos un script en python. La cadena usada como mensaje será “holamundo”

He usado el siguiente script:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import hashlib
import sys
from random import randint
import time
import sys

"""
#Funcion que devuelve una cadena aleatoria de nBits binario
#devuelve la cadena en entero. Por ejemplo 3 bits, 0..a 7
"""
def randomBits(nBits):
	bits = ""
	for x in range(0,int(nBits)):
		bits = bits+""+str(randint(0,1))
	bits = int(bits,base=2)
	return bits

#Funcion que devuelve n numeros aleatorios
def random_with_N_digits(n):
	n = int(n)
	range_start = 10**(n-1)
	range_end = (10**n)-1
	return randint(range_start, range_end)


#Funcion que convierte una cadena ascii a binario
def asciiToBinary(chain):
	chain=bin(int(str(chain),16))
	return chain

#se le pasan string
def sumBinary(a,b):
	a = int(a)
	b = int(b)
	return (str(asciiToBinary(a+b)))

def hashing(msg, bits):
	idM = str(bits)[2:]+""+msg #quitamos el 0b
```

```
        idM = str(idM).encode('utf8')
        return (str(hashlib.sha256(idM).hexdigest()))
```

Aquí vemos que al devolver, quitamos los 3 primeros caracteres. Esto se produce porque al pasar de hexadecimal a binario siempre añade un uno, más el 0b inicial, 0b1.

Función principal, generamos la cadena aleatoria, el id, y aplicamos hash. Mientras los primeros n-bits sean distintos de 0,vamos concatenando números aleatorios al id.

```
def run(message,nbits):
        contador=0
        strZero="0"*nbits
        """
        Generamos el numero aleatorio en binario
        """
        x = randomBits(nbits)
        xBin = bin(x)

        """
        Creamos el id concatenando la cadena de bits y el mensaje.
        Aplicamos hash, todo ello con el metodo hashing
        """
        #idM = str(xBin)+""+message
        hashM = hashing(message,xBin)
        #idM  0b10101010holamundo


        hashMBin=bin(int(hashM, 16)) #hexadecimal a binario
        chainHashBin = hashMBin[3:] #binario pero quitando los 3 primeros digitos
0b y el bit extra que introduce hexadecimal a binario

        chainSumX=xBin[2:] #cadena a la que se le va sumano x



        #seguimos sumando uno a la cadena de bits mientras los primeros nBits sean
distintos de 0
        while((str(chainHashBin[:nbits])!=str(strZero))):
                contador+=1
                sumHash=randomBits(nbits)

                chainBin = bin(sumHash)[2:]
                chainSumX = chainBin+""+chainSumX


                #procedemos a concatenar y aplicar la funciona hash
                chainHashBin=hashing(message,chainSumX)
```

```
            chainHashBin=bin(int(chainHashBin,16))[3:] #hexadecimal a binario.
Quitamos el 0b y el primer numero, porque siempre introduce un 1 al pasar
hexadecimal a binario

        print("\nIteracion: "+str(contador))
        print("\nHash: "+str(chainHashBin))
        print("\nId: "+chainSumX+message)
        print("\nCadena bits: "+str(bin(sumHash)))

run("holamundo",int(sys.argv[1]))
```

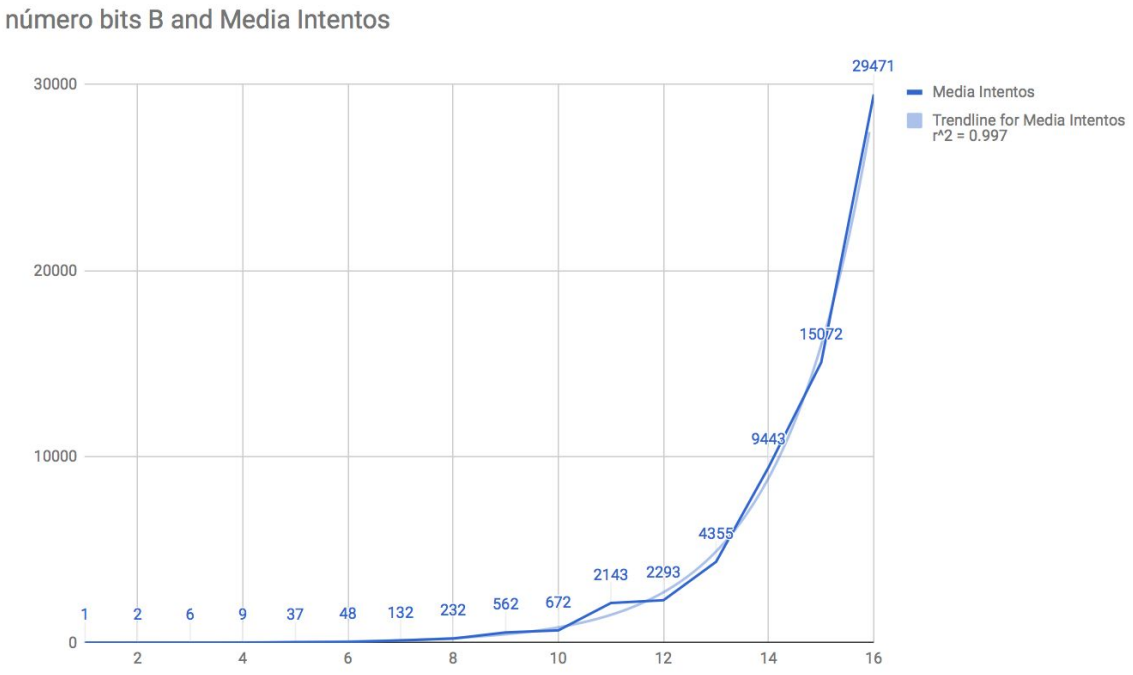
- Bits finales añadidos aleatorios: 0b110011001100110110
- Id mensaje:
  00101000100110111111110100011001111000011010001110100110111001001010001001011100100111011100111101101010000101110101100111011001010000111010111111111010011101101101010110100001101100101001011001101110000110010001111110111111111101001111010110101011010000110110010100101100110111000011001000110011001100100011101110011110110101000010111010110011101100101000011101011111111101001110110110101011010000110110010100101100110111000011001000110011001100100011101110011110110101000010111010110011101100101000011101011111111101001110110110101011010000110110010100101100110111000011001000110011001100100011101110011110110101000010111010110011101100101000011101011111111101001110110110101011010000110110010100101100110111000011001000110011001100100011101110011110110101000010111010110011101100101000011101011111111101001110110110101011010000110110010100101100110111000011001000110011001100100011101110011110110101000010111010110011101100101000011101011111111101001110110b110011001100110011011
- Mensaje hash:
  0b10000010111001110010101101110101100101000001001110100101111010000110101001100001000000011101010001101111110111010010111000010100011110011011110100001010110110011010010000110011011011010001110010000111100000100110101011101000110000111100100001111001011010
- Mensaje (Igual que el hash, excepto 3 primeros dígitos indican binario)

0000010111001110010101101110101100101000001001110100101111010000110101001100001000000011101010001101111110111010010111000010100011110011011110100001010110110011010010000110011011011010001110010000111100000100110101011101000110000111100100001111001011010

000010101101100100001100010000010111001110111101000011100100001111000001001101010111010001100001111001010110

- Número de iteraciones: 6
- Mensaje inicial: ABC
- Número de bits aleatorios inicial: 3

2. Calculad una tabla/gráfica que vaya calculando el número de intentos para cada valor de b. Con el objeto de que los resultados eviten ciertos sesgos, para cada tamaño b realizad el experimento 10 veces y calculad la media del número de intentos.



número bits B	Media Intentos
1	1
2	2
3	6
4	9
5	37
6	48

7	132
8	232
9	562
10	672
11	2143
12	2293
13	4355
14	9443
15	15072

3. Repetid la función anterior con el siguiente cambio: Se toma un primer valor aleatorio x y se va incrementando de 1 en 1 hasta obtener el hash requerido.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import hashlib
import sys
from random import randint
import time
import sys

"""
#Funcion que devuelve una cadena aleatoria de nBits binario
#devuelve la cadena en entero. Por ejemplo 3 bits, 0..a 7
"""
def randomBits(nBits):
	bits = ""
	for x in range(0,int(nBits)):
		bits = bits+""+str(randint(0,1))
	bits = int(bits,base=2)
	return bits

#Funcion que devuelve n numeros aleatorios
def random_with_N_digits(n):
	n = int(n)
	range_start = 10**(n-1)
	range_end = (10**n)-1
	return randint(range_start, range_end)


#Funcion que convierte una cadena ascii a binario
def asciiToBinary(chain):
	chain=bin(int(str(chain),16))
	return chain
```

```
#se le pasan string
def sumBinary(a,b):
	a = int(a)
	b = int(b)
	return (str(asciiToBinary(a+b)))

def hashing(msg, bits):
	idM = str(bits)[2:]+""+msg #quitamos el 0b
	idM = str(idM).encode('utf8')
	return (str(hashlib.sha256(idM).hexdigest()))
```

Función principal, que llama a la función hash una primera vez, y mientras los primeros n-bits de la izquierda

```
def run(message,nbits):
	contador=0
	strZero="0"*nbits
	"""
	Generamos el numero aleatorio en binario
	"""
	x = randomBits(nbits)
	xBin = bin(x)

	"""
	Creamos el id concatenando la cadena de bits y el mensaje.
	Aplicamos hash, todo ello con el metodo hashing
	"""
	#idM = str(xBin)+""+message
	hashM = hashing(message,xBin)
	#idM  0b10101010holamundo


	hashMBin=bin(int(hashM, 16)) #hexadecimal a binario
	chainHashBin = hashMBin[3:] #binario pero quitando los 3 primeros digitos
0b y el bit extra que introduce hexadecimal a binario

	sumaHash=xBin


	sumHash=0
	#seguimos sumando uno a la cadena de bits mientras los primeros nBits sean
distintos de 0
	while((str(chainHashBin[:nbits])!=str(strZero))):
		contador+=1
		#print("\nNumero aleatorio binario en decimal"+str(int(xBin,base=2)))
		sumHash=int(xBin,base=2)+contador #sumamos uno en uno
		#print("\nSuma"+str(sumHash))
		sumBin = bin(sumHash)
```

```
            #procedemos a concatenar y aplicar la funciona hash
            chainHashBin=hashing(message,sumBin)

            chainHashBin=bin(int(chainHashBin,16))[3:] #hexadecimal a binario.
Quitamos el 0b y el primer numero, porque siempre introduce un 1 al pasar
hexadecimal a binario

        print("\nIteracion: "+str(contador))
        print("\nHash: "+str(chainHashBin))
        print("\nId: "+str(bin(sumHash))+message)
        print("\nCadena bits: "+str(bin(sumHash)))

run("holamundo",int(sys.argv[1]))
```

Como se puede comprobar cuando pasamos de hexadecimal a binario, tengo que eliminar los 3 primeros dígitos siempre, porque al hacer el cambio de base, siempre se añade 0b1 a la cadena en binario. Si simplemente se pasa de decimal a binario(caso de después de hacer la suma), quitamos los 2 primeros caracteres.

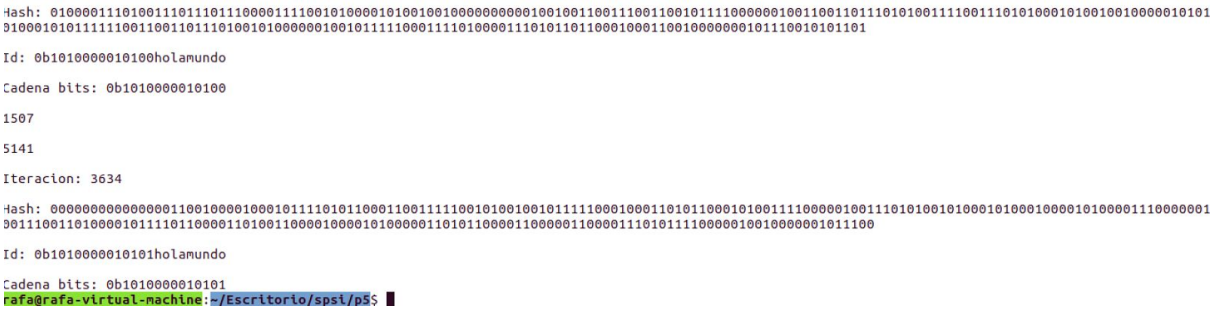
```
rafa@rafa-virtual-machine:~/Escritorio/spsi/p5$ python3 bhash.py 1

1

2

Iteracion: 1

Hash: 10010000010100010010110100001101110001011000011001011111010010111011101111010110000100110010111110101010011000001011010011101000000000011
1001000111001001111010010010010011100100001100010111001010100110000101000011101100010010001110010000010110010

Id: 0b10holamundo

Cadena bits: 0b10
```

4. Calculad una nueva tabla/gráfica similar a la obtenida en el punto 2 pero con la función construida en 3.

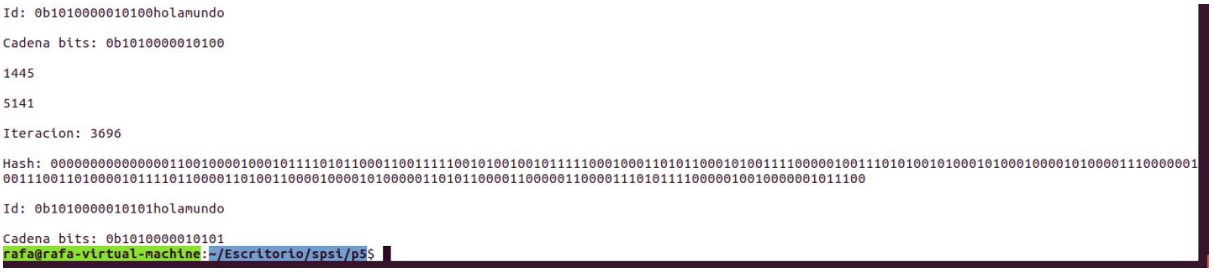
número bits B	Media Intentos
1	1
2	2
3	3
4	11
5	10
6	36
7	75
8	246
9	340
10	4493
11	4553

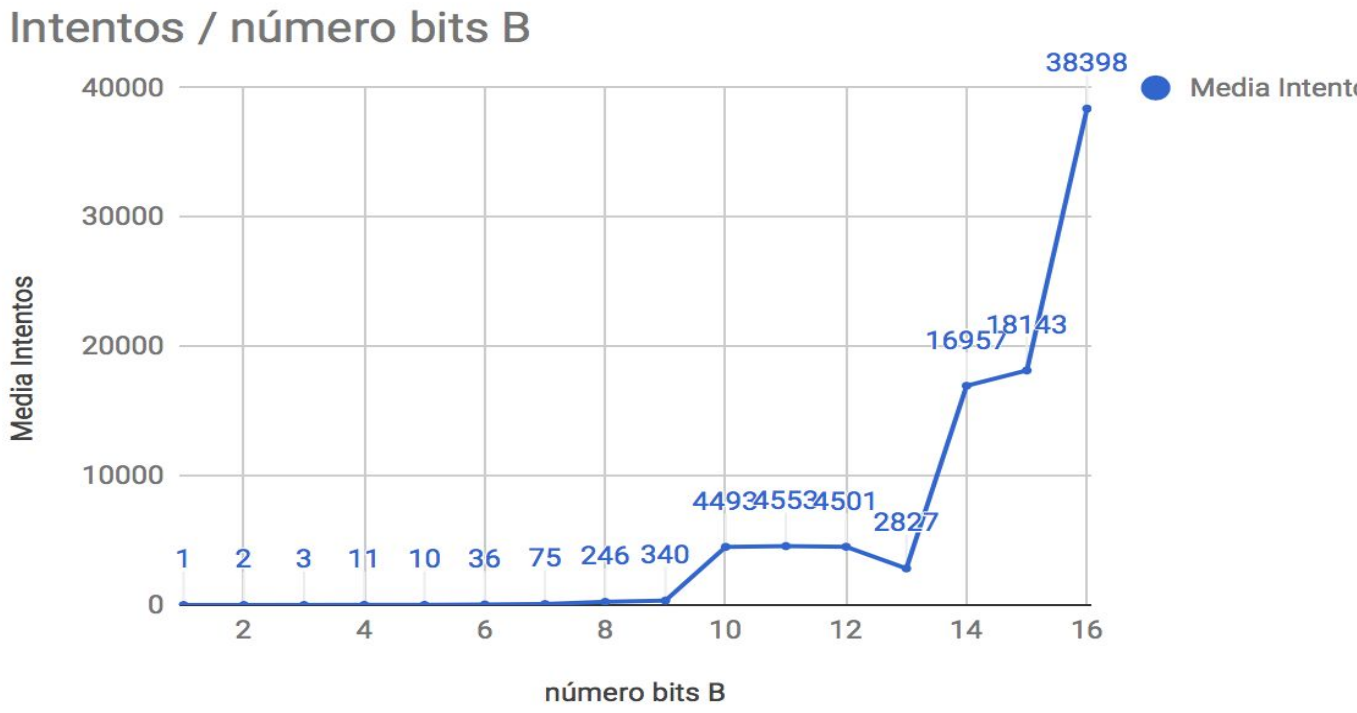
12	4501
13	2827
14	16957
15	18143
16	38398

Vemos que a partir de 10 bits, a diferencia del ejercicio anterior, genera más 0 a la izquierda de lo necesario, por lo que los resultados ya no son tan significativos, y la pendiente se suaviza.
Para 10 bits por ejemplo:



Para 11 bits:





Vemos como a partir de 9 bits, se suaviza la pendiente y se mantiene estable incluso repunta un poco hacia un menor número de intentos. Esto ocurre porque en esa franja, no sólo se hacen 0 los n primeros bits, sino algunos más, al hacer el hash, por lo que al incrementar el número de bits hasta 14, los intentos son más o menos iguales que en los 3-4 n-bits anteriores.
A partir de 14 bits, el número de intentos se dispara exponencialmente.