Seguridad y Protección de Sistemas Informáticos Práctica 5

Miguel Morales Castillo

1. Para la función H, realizad, en el lenguaje de programación que queráis, una función que tome como entrada un texto y un número de bits b. Creará un id que concatene una cadena aleatoria de n bits con el texto. Pegará a ese id cadenas aleatorias x de n bits hasta lograr que H(id||x) tenga sus primeros b bits a cero. La salida será el id, la cadena x que haya proporcionado el hash requerido, el valor del hash y el número de intentos llevados a cabo hasta encontrar el valor x apropiado.

Para este apartado y los siguientes hemos cogido como función H SHA-256. Como lenguaje de programación hemos escogido Python, mas concretamente Jupyter Notebook. El código es el que presentamos a continuación:

```
def randomNBits(numBits):
    cadena=""
    for i in range(0,int(numBits)):
        cadena=cadena+str(r.randint(0,1))

    return bin(int(cadena,2))

def puzzle (mensaje,b):
    zeros="0"*b
    contador=0
    cadena=randomNBits(256)
    identificador=mensaje+str(cadena[2:])
    valorHash=bin(int(h.sha256(str(identificador).encode('utf-8')).hexdigest(),16))[3:]
    cadenaTotal=identificador
    while(str(valorHash[:b])!=zeros):
        contador+=1
        x=randomNBits(256)
        cadenaTotal=identificador +str(x[2:])
        valorHash=bin(int(h.sha256(str(cadenaTotal).encode('utf-8')).hexdigest(),16))[3:]

    '''print ("Identificador  ",identificador)
    print ("\n\n Valor Hash   ",valorHash)
    print ("\n\n Valor X    ", cadenaTotal[len(identificador):] )
    print ("\n\n Numero de intentos   ", contador)'''
    return contador
```

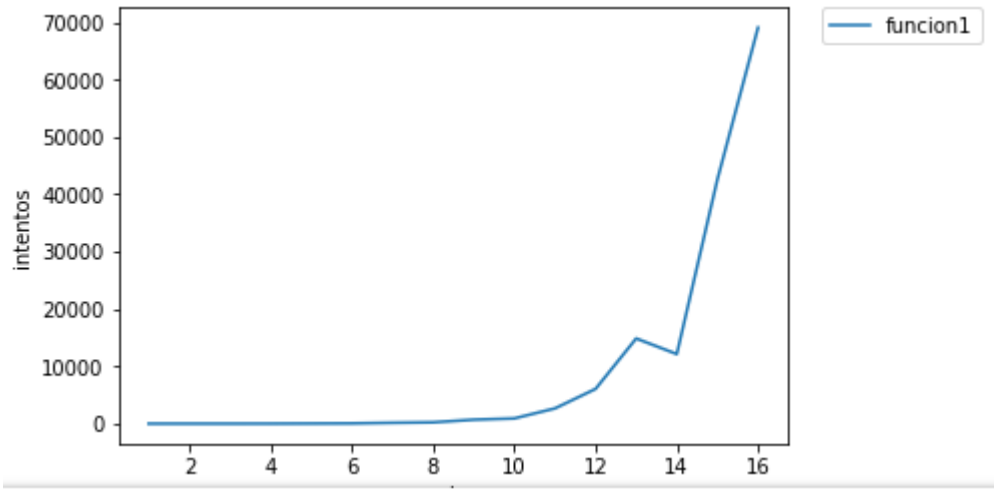
El proceso es basicamente lo que se pide en el enunciado, generar un id, y concatenar con cadenas aleatorias x de n bits hasta que el valor hash tenga sus b primeros bits a 0, imprimiendo el id, el valor hash, la cadena x que ha logrado el hash y el número de intentos. Como valor de return devolvemos el contador de intentos para después hacer la media y dibujar la gráfica que se pide en el apartado siguiente.

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.

Hemos calculado el número de intentos para b desde 1 a 16 ambos inclusive, el resultado es el siguiente.

b	número de intentos
1	1.2
2	4.2
3	3.7
4	8.0
5	28.3
6	52.9
7	156.7
8	236.7
9	695.6
10	918.4
11	2665.1
12	6067.1
13	14864.5
14	12141.4
15	42624.8
16	69091.9



Como ya sabiamos crece de forma exponencial.

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.

   La función desarrollada es la siguiente:

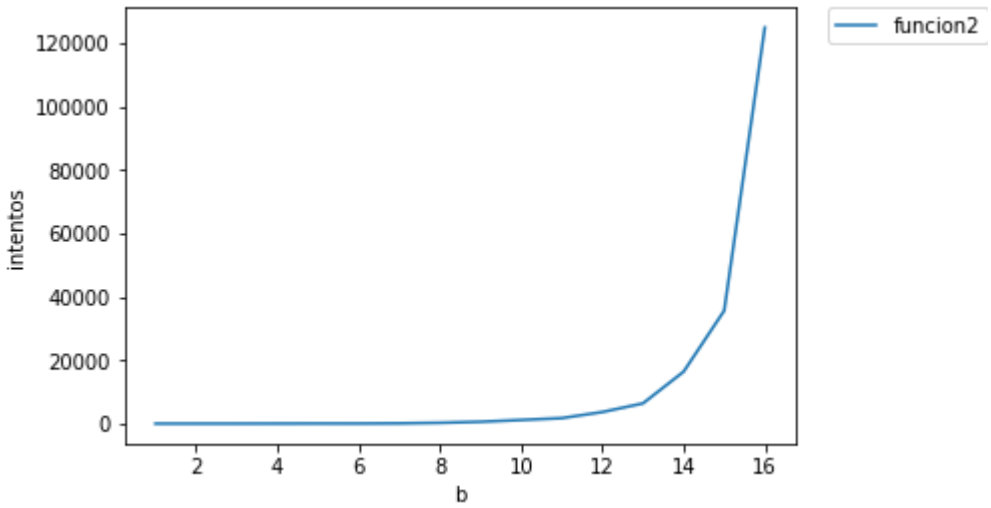
```
def puzzle2 (mensaje,b):
    zeros="0"*b
    contador=0
    cadena=randomNBits(256)
    identificador=mensaje+str(cadena[2:])
    x=randomNBits(256)
    cadenaTotal=identificador +str(x[2:])
    valorHash=bin(int(h.sha256(str(cadenaTotal).encode('utf-8')).hexdigest(),16))[3:]
    while(str(valorHash[:b])!=zeros):
        contador+=1
        x=bin(int(x,2)+contador)
        cadenaTotal=identificador +str(x[2:])
        valorHash=bin(int(h.sha256(str(cadenaTotal).encode('utf-8')).hexdigest(),16))[3:]

    '''print ("\n\n Identificador  ",identificador)
    print ("\n\n Valor Hash   ",valorHash)
    print ("\n\n Valor X    ", cadenaTotal[len(identificador):] )
    print ("\n\n Numero de intentos   ", contador)'''
    return contador
```

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

   Los resultados obtenidos son los siguientes:

b	número de intentos
1	1.3
2	2.8
3	7.4
4	23.3
5	41.1
6	35.8
7	91.5
8	267.3
9	588.6
10	1155.6
11	1736.7
12	3657.3
13	6380.9
14	16367.3
15	35610.8
16	124801.1

   Y la gráfica producida es la siguiente:



Como podemos observar la gráfica en este caso se suaviza gracias al cambio introducido.