SEGURIDAD Y PROTECC.DE SIST.INFORMÁTICOS (2018-2019)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 5. Puzles hash y blockchain

Patricia Maldonado Mancilla

Índice

1. Para la función H, realizad, en el lenguaje de programación que queráis, una función que tome como entrada un texto, un n umero de bits b y una cadena de n bits. Crear a un id que concatene dicha cadena de n bits con el texto. Pegar a a ese id cadenas aleatorias x de n bits hasta lograr que H(idjjx) tenga sus primeros b bits a cero. La salida ser a un bloque que contenga 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. 4

2. Construid una simulación de una blockchain en la que no emplearemos punteros. El primer bloque será la salida la función del apartado 1 teniendo como entrada vuestro nombre como mensaje, b = 2 y una cadena aleatoria de n bits. Los siguientes 9 bloques consistir an en la salida de la funci on del apartado 1 teniendo como entrada vuestro nombre como mensaje, b = 2 y el valor hash del bloque anterior como cadena de bits 8

3. Los siguientes 10 bloques consistirán en la salida de la función del apartado 1 teniendo como entrada vuestro nombre como mensaje, b = 3 y el valor hash del bloque anterior como cadena de bits. 12

4. Repetid el apartado 3 incrementando el valor de b iterativamente de uno en uno. Parad cuando vuestro ordenador tarde varios minutos en completar la tarea. 13

5. Construid una tabla/gráfica que tenga en el eje de abscisas el n umero de bits b y en el eje de ordenadas la media de n umero de intentos empleados en las 10 ejecuciones de la función del apartado 1 para cada valor de b. 15

Índice de figuras

1.1. Código ejercicio 1 5
1.2. Ejecuciones ejercicio 1 6
2.1. Código ejercicio 2 8
2.2. Código ejercicio 2 9
2.3. Ejecución ejercicio 2 10
2.4. Ejecución ejercicio 2 11
3.1. Código ejercicio 3 12
3.2. Ejecución ejercicio 2 13
4.1. Código ejercicio 4 14
4.2. Ejecución ejercicio 4 15
5.1. Tabla ejercicio 1 16
5.2. Grafica ejercicio 1 16

1. Para la función H, realizad, en el lenguaje de programación que queráis, una función que tome como entrada un texto, un n umero de bits b y una cadena de n bits. Crear a un id que concatene dicha cadena de n bits con el texto. Pegar a a ese id cadenas aleatorias x de n bits hasta lograr que H(idjjx) tenga sus primeros b bits a cero. La salida ser a un bloque que contenga 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.

Este ejercicio se ha realizado en Python y se ha utlizado la función sha-256. Consta de dos funciones, una para generar la cadena aleatoria de n bits y la función principal. La función principal:

- se le pasa el texto, un número de ceros y la cadena de n bits aleatorios. He utilizado como texto PatriciaMaldonado y el número de ceros deseado en cada caso.
- id: concatenamos la cadena de n bits aleatorios con el texto.
- auxID: pegamos las cadenas aleatorias al id.
- hash: calculamos el hash con sha-256.
- Cuando el valor de ceros sea al menos el introducido, mostramos por pantalla la cadena n bits, el id, el hash, y el número de intentos que ha tenido que realizar.

```python
#Patricia Maldonado Mancilla
from random import randint
import hashlib


num=256
numCeros = 8
text ="PatriciaMaldonado"

#cadRandom: funcion que devuelve la cadena de bits aleatorios
def cadRandom(num):
  bits=""
  for x in range(0,int(num)):
    bits=bits+str(randint(0,1))
  return bits

cad = cadRandom(num)

#A la funcion principal le pasamos un texto, un número de bits b y una cadena de n bits
def principal(texto,b,n):

  #id = cadena nbits + texto
  id = n+texto

  seguir = True
  #Contador de intentos
  cont = 0

  while(seguir):
    cont+=1
    #cadena hex
    cadRan = cadRandom(num)
    #concatena id y cadena  de bits aleatorios
    auxID = id + cadRan
    #calcular hash con sha256
    hash = hashlib.sha256(auxID.encode('utf-8')).hexdigest()
    #Pasamos de hexadecimal a binario quitando 0b
    traduc_hash =  bin(int(hash,16))[2:]
    #Con zfill agregamos tantos ceros al principio hasta que alcance la longitud num
    result = traduc_hash.zfill(num)
#Comprobamos el número de ceros
    if(int(result[0:b],2) == 0):
      seguir= False
      print ("\nCadena de bits:",cadRan)
      print ("\nID:",id)
      print ("\nHash:",hash)
      print ("\nNº de intentos:",cont)


principal(text,numCeros,cad)
```

Figura 1.1: Código ejercicio 1

Ejecuciones para el número de ceros con valores: 4,8 y 12.

```
patri@patri:~/Escritorio$ python3 Ejercicio1.py

Cadena de bits: 1000000001101000101000101111110001110001100011101000010101000111
110010110000011010110010100111100011111001011000011100101110100111100011000100011
110100011010001010101101000000000110011111111110000100111010011110111011000101100
010100001100011101001111110100110

ID: 111101001000010111100011001001000111100111000001010001011100110110001010001100
011001110011000000010111111011100011101111000101011010101010011010010011001000000
101011011111000001111011010001011111000001100001010111101011010010001001001110100
1000110100100110100PatriciaMaldonado

Hash: 07936902718 94e49a3217f57b0496a424c0c1841706c7022bd9edb4b88abc87f

Nº de intentos: 4
patri@patri:~/Escritorio$ python3 Ejercicio1.py

Cadena de bits: 1011111010001100100111101011110001011001110111100111101011010111
000111110001001001100000011100110000101101001010111010000101101000000110100011110
000101111001011111111101100010010110001010000101101110011010000010010000101100001
011111000010101001111011001001111

ID: 101110000010100111110000100101000000101000000011011110011000101100111000101010
010100011101111110001101110100111011110010000101101001000011101000101011111100110
100001010101001010101100000000001010001001101110100001100100001011010100000111010
101111100010101001000PatriciaMaldonado

Hash: 001dfddae0652567e0a647e58c68d17354aafa364afddc4963a2630baa387754

Nº de intentos: 321
patri@patri:~/Escritorio$ python3 Ejercicio1.py

Cadena de bits: 1011010000101100111010010010011101001110011101011101101001000001
010100010101011010100000110100101000100011100101000111110011111001100110011010001
010100011011101011011100001110110000010110110101000111110001001111001101100010100
010001001100110000100110111111011

ID: 100001010101010010111001101100100010111110011011001111011101101111010011010
000111011011010101001100100101000000010000010001000111010011101001101101100100111
101010000110111001001011111011110001101101011000000110110111110001001111111100011
010101111001111010000PatriciaMaldonado

Hash: 000dc0d17aac2bd48a05ead40d79dfede36f72c93ccfb7d6247ebcba5c726d58

Nº de intentos: 7247
```

Figura 1.2: Ejecuciones ejercicio 1

2. **Construid una simulación de una blockchain en la que no emplearemos punteros. El primer bloque será la salida la función del apartado 1 teniendo como entrada vuestro nombre como mensaje, b = 2 y una cadena aleatoria de n bits. Los siguientes 9 bloques consistir an en la salida de la funci on del apartado 1 teniendo como entrada vuestro nombre como mensaje, b = 2 y el valor hash del bloque anterior como cadena de bits**

```python
#Patricia Maldonado Mancilla
from random import randint
import hashlib

#cadRandom: funcion que devuelve la cadena de bits aleatorios
def cadRandom(num):
  bits=""
  for x in range(0,int(num)):
    bits=bits+str(randint(0,1))
  return bits

def principal(texto,b,n):
  num = 256
  #id = cadena nbits + texto
  id = n+texto

  seguir = True
  #Contador de intentos
  cont = 0

  while(seguir):
    cont+=1
    #cadena hex
    cadRan = cadRandom(num)
    #concatena id y cadena hex
    auxID = id + cadRan
    #calcular hash con sha256
    hash = hashlib.sha256(auxID.encode('utf-8')).hexdigest()
    #Pasamos de hexadecimal a binario quitando 0b
    traduc_hash =  bin(int(hash,16))[2:]
    #Con zfill agregamos tantos ceros al principio hasta que alcance la longitud num
    result = traduc_hash.zfill(num)
    #Cuando encontremos el numero de ceros deseados paramos
    if(int(result[0:b],2) == 0):
      seguir= False

      print ("\nCadena de bits:",cadRan)
      print ("\nID:",id)
      print ("\nHash:",hash)
      print ("\nNº de intentos:",cont)
      return hash
```

Figura 2.1: Código ejercicio 2

La modificación con respecto al ejercicio 1 se realiza en la siguiente captura, en la cual se llama a principal para el primer bloque con valor b=2 y a continuación usamos un for para los siguientes 9 bloques con el valor hash del primer bloque generado como cadena de bits.

```
def ejer2():
    b = 2
    cad = cadRandom(256)
    #primer bloque
    print ("\n**********************Bloque 1**********************")
    hash = principal("Patricia",b,cad)
    #Generamos los 9 siguientes bloques con el valor hash del bloque anterior
    for i in range(2,11):
        print ("\n**********************Bloque",str(i),"**********************")
        hash= principal("Patricia",b,hash)

ejer2()
```

Figura 2.2: Código ejercicio 2

A continuación vemos la salida de estos 10 bloques.

```
patri@patri:~/Escritorio$ python3 Ejercicio2.py

**********Bloque 1**********

Cadena de bits: 1110011000011110000110011101111010001011011111001100000101000000
0111001110001001010110110110011111010110101011011000101110100111110011101100100

ID: 0000111010001111011010110010001011100100100110000001100010011011011110111011
0100110001111010110111001011111110011010111110100001000010100011111011101011010

Hash: 1ad2f8d3c56c3a8edaf788100cfe6d61fe2a6d17842dace10ef15dcad361e5e4

Nº de intentos: 1

**********Bloque 2 **********

Cadena de bits: 1111011100111111000101101101111000010110110001001011110100011111
0000100111001001101101010000000100100101100000010110010001100010110000010001111

ID: 1ad2f8d3c56c3a8edaf788100cfe6d61fe2a6d17842dace10ef15dcad361e5e4Patricia

Hash: 35715b4fb3e4d573e14a35e4f56eda9d737c4b4a112d1f896db30f094f4e0e1d

Nº de intentos: 3

**********Bloque 3 **********

Cadena de bits: 0100110011111010101110010001110001111110110000010101110101110011
1001011101110011000000101010101110000011100000000110101100101100010011010111011 0

ID: 35715b4fb3e4d573e14a35e4f56eda9d737c4b4a112d1f896db30f094f4e0e1dPatricia

Hash: 0434a4b7658f5adba56a5aa31109a24c48d6b6f9f68c2ec2d2fdb72f0b8beefa

Nº de intentos: 11

**********Bloque 4 **********

Cadena de bits: 1111000110101100010110100111011000110010101110000010110010100011
1011100001001110101100001000101011011010010001000000011010110100001011111001011 1

ID: 0434a4b7658f5adba56a5aa31109a24c48d6b6f9f68c2ec2d2fdb72f0b8beefaPatricia

Hash: 0cb9bcee58dbd242ffce1da43e080af4e23be18a0c55a231cab97a4c6812ac6e

Nº de intentos: 5

**********Bloque 5 **********

Cadena de bits: 1100000001011000111111110100110001001100110010000101101000010111 0
1110101101001011111111111000001101111110010100100100000001101010010110111001000 11

ID: 0cb9bcee58dbd242ffce1da43e080af4e23be18a0c55a231cab97a4c6812ac6ePatricia

Hash: 2d6d0eefe2394bf1dff01ebd48a53c6b2703e8f7ad8ec200e2da1bf5e919fcdd

Nº de intentos: 1

**********Bloque 6 **********

Cadena de bits: 0111001000001000111101100011010110101011111110110010000101001101 0
0011011000101000111000111100100010111001100010110110001110010010011110100100101 0

ID: 2d6d0eefe2394bf1dff01ebd48a53c6b2703e8f7ad8ec200e2da1bf5e919fcddPatricia

Hash: 359fe5d3e9605f177cf07767898b12b198ab7d23035136f3b362b280525db823

Nº de intentos: 1
```

Figura 2.4: Ejecución ejercicio 2

3. Los siguientes 10 bloques consistirán en la salida de la función del apartado 1 teniendo como entrada vuestro nombre como mensaje, b = 3 y el valor hash del bloque anterior como cadena de bits.

Para este ejercicio se ha agregado al ejercicio anterior los siguientes 10 bloques con valor b = 3 y el valor hash del bloque anterior como cadena de bits.

```
def ejer3():

    cad = cadRandom(256)
    #primer bloque
    print ("\n**********Bloque 1**********")
    hash = principal("Patricia",2,cad)
    print ("\nHash:",hash)
    #Generamos los 9 siguientes bloques con el valor hash del bloque anterior
    for i in range(2,11):
        print ("\n**********Bloque",str(i),"**********")
        hash= principal("Patricia",2,hash)
    # Siguientes 10 bloques con valor b = 3
    for i in range(11,22):
        print ("\n**********Bloque",str(i),"**********")
        hash= principal("Patricia",3,hash)
ejer3()
```

Figura 3.1: Código ejercicio 3

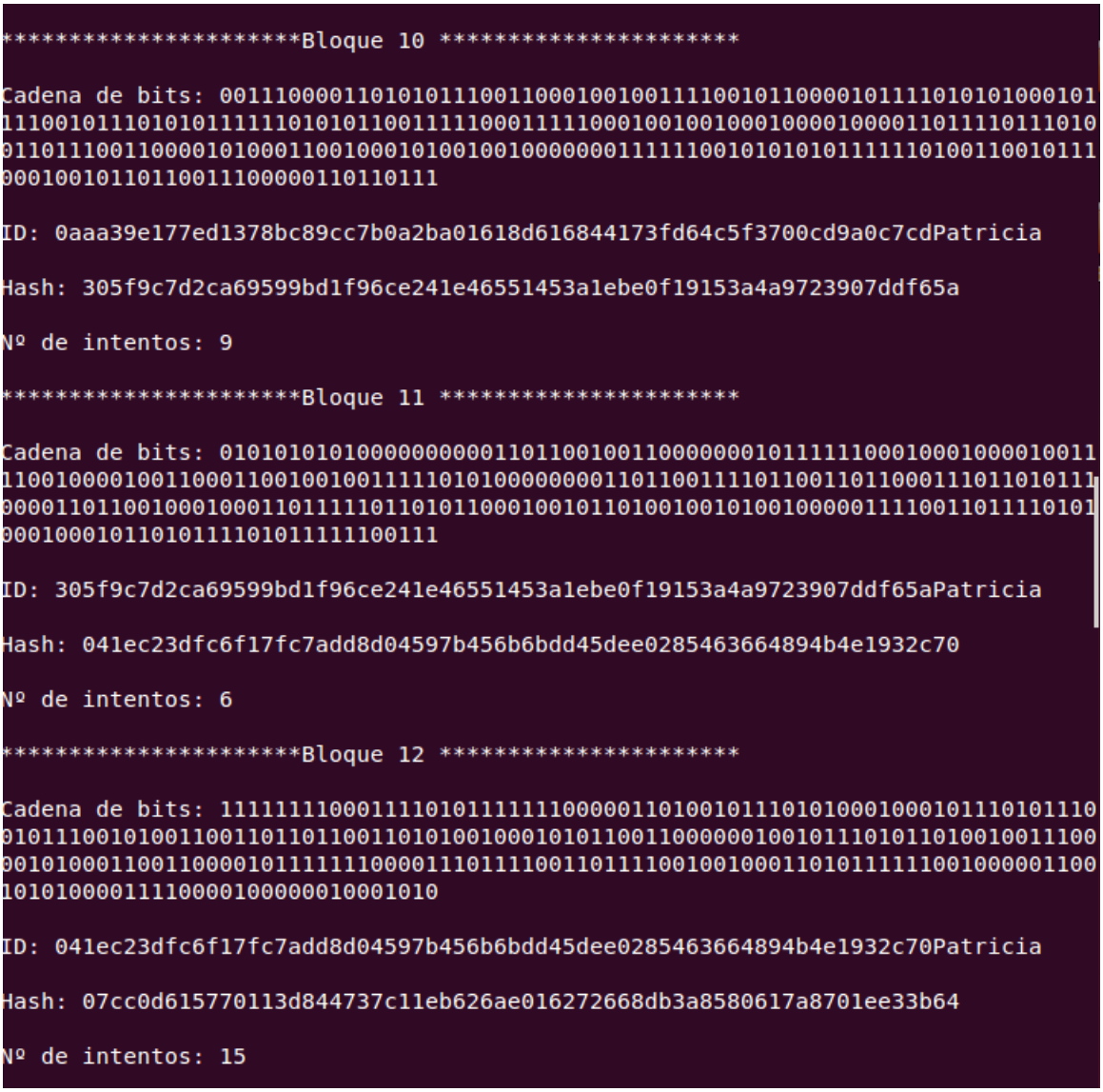

Figura 3.2: Ejecución ejercicio 2

4. Repetid el apartado 3 incrementando el valor de b iterativamente de uno en uno. Parad cuando vuestro ordenador tarde varios minutos en completar la tarea.

Para este ejercicio he modificado el ejercicio 3 incrementando el valor de b de uno en uno, empezando por el valor de b = 3, llegando hasta el bloque 25 con valor de b = 17 que es cuando se empieza a ralentizar.

```python
def ejer4():
    b = 2
    cad = cadRandom(256)
    #primer bloque
    print ("\n**********Bloque 1**********")
    hash = principal("Patricia",2,cad)
    for i in range(2,11):
        print ("\n**********Bloque",str(i),"**********")
        hash= principal("Patricia",2,hash)
    # # Siguientes 10 bloques con valor b = 3
    # for i in range(11,22):
    #     print ("\n**********Bloque",str(i),"**********")
    #     hash= principal("Patricia",3,hash)
    # Incrementamos b

    #modificación del ejercicio 3 para que b se incremente de 1 en 1
    for i in range(11,26):
        b+=1
        print ("\n**********Bloque",str(i),"**********")
        hash= principal("Patricia",b,hash)
        print ("\nVALOR B:",b,"<---")

ejer4()
```

Figura 4.1: Código ejercicio 4

Figura 4.2: Ejecución ejercicio 4

5. Construid una tabla/gráfica que tenga en el eje de abscisas el n umero de bits b y en el eje de ordenadas la media de n umero de intentos empleados en las 10 ejecuciones de la función del apartado 1 para cada valor de b.

En la siguiente tabla se muestran los valores de b hasta b =14 y la media de intentos que se ha obtenido en 10 ejecuciones para cada valor de b.

Valor b	Media Intentos
1	2
2	3,2
3	6,9
4	10,8
5	29,4
6	47,3
7	189,6
8	211,2
9	631,4
10	777,9
11	2365,8
12	5119,4
13	5306,1
14	16014,7

Figura 5.1: Tabla ejercicio 1

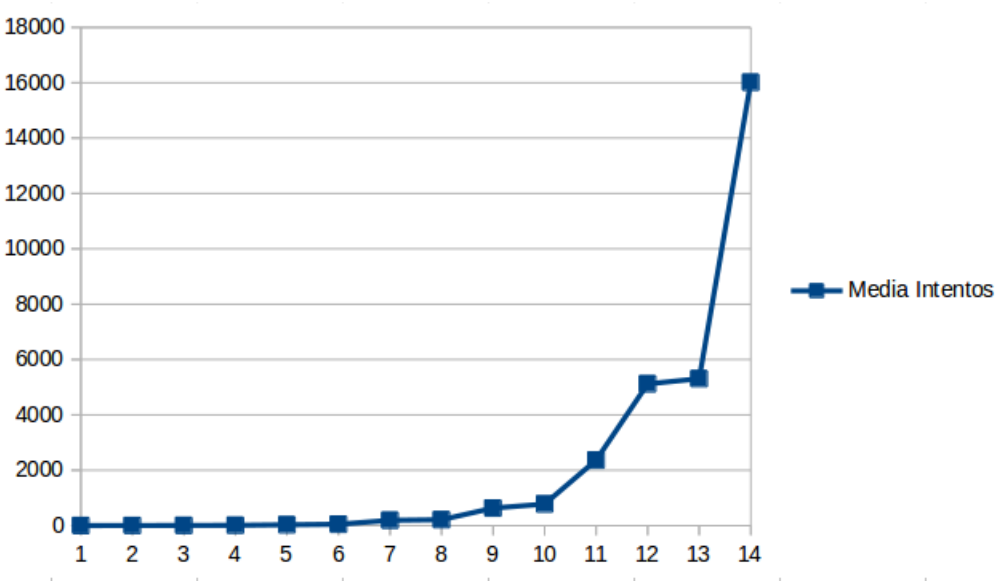

Figura 5.2: Grafica ejercicio 1

Podemos observar que conforme el valor de b (número de ceros que queremos encontrar) incrementa, el número de intentos que realiza es mayor.