

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 número 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 $H(idjx)$ 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 función 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 número de bits b y en el eje de ordenadas la media de número 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 número 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 $H(id||x)$ 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 utilizado 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.

```

#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
11001011000001101011001010011110001111100101100001110010111010011110001100010001
1101000110100010101011010000000001100111111110000100111010011110111011000101100
01010000110001110100111110100110

ID: 1111010010000101111000110010010001111001110000010100010111001101100010100011
011001110011000000010111110111000111011110001010110101010011010010011001000000
1010110111110000011101101000101111100000110000101011110101101001000100100111010
10001101001001101000PatriciaMaldonado

Hash: 0793690271894e49a3217f57b0496a424c0c1841706c7022bd9edb4b88abc87f

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

Cadena de bits: 1011111010001100100111101011110001011001110111100111101011010111
00011111000100100110000001110011000010110100101011101000010110100000011010001110
0001011110010111111101100010010110001010000101101110011010000010010000101100001
01111100001010100111101100100111

ID: 1011100000101001111100001001010000001010000000110111100110001011001110001010
01010001110111110001101110100111011110010000101101001000011101000101011111100110
10000101010100101010110000000001010001001101110100001100100001011010100000111010
10111110001010100100PatriciaMaldonado

Hash: 001dfddae0652567e0a647e58c68d17354aafa364afddc4963a2630baa387754

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

Cadena de bits: 1011010000101100111010010010011101001110011101011101101001000001
01010001010101101010000011010010100010001110010100011111001111100110011001101001
01010001101110101101110000111011000001011011010100011111000100111100110110001010
01000100110011000010011011111011

ID: 1000010101010100101110011011001000101111100110110011110111011011110100110100
00011101101101010100110010010100000001000001000100011101001110100110110110010011
10101000011011100100101111011110001101101011000000110110111110001001111111100011
01010111100111101000PatriciaMaldonado

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 consistirán en la salida de la función del apartado 1 teniendo como entrada vuestro nombre como mensaje, $b = 2$ y el valor hash del bloque anterior como cadena de bits

```
#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
011100111000100101011011011001111101011010110110001011101001111100111011001000

ID: 0000111010001111011010110010001011100100100110000001100010011011011110111011
0100110001111010110111001011111100110101111101000010000101000111110111010111010

Hash: 1ad2f8d3c56c3a8edaf788100cfe6d61fe2a6d17842dace10ef15dcad361e5e4

Nº de intentos: 1

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

Cadena de bits: 1111011100111111000101101101111000010110110001001011110100011111
00001001110010011011010100000001001001011000000101100100011000101100000100011111

ID: 1ad2f8d3c56c3a8edaf788100cfe6d61fe2a6d17842dace10ef15dcad361e5e4Patricia

Hash: 35715b4fb3e4d573e14a35e4f56eda9d737c4b4a112d1f896db30f094f4e0e1d

Nº de intentos: 3

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

Cadena de bits: 0100110011111010101110010001110001111110110000010101110101110011
100101110111001100000001010101011100000111000000001101011001011000100110101110110

ID: 35715b4fb3e4d573e14a35e4f56eda9d737c4b4a112d1f896db30f094f4e0e1dPatricia

Hash: 0434a4b7658f5adba56a5aa31109a24c48d6b6f9f68c2ec2d2fdb72f0b8beefa

Nº de intentos: 11

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

Cadena de bits: 1111000110101100010110100111011000110010101110000010110010100011
101110000100111101011000010001010110110100100010000000110101101000010111110010111

ID: 0434a4b7658f5adba56a5aa31109a24c48d6b6f9f68c2ec2d2fdb72f0b8beefaPatricia

Hash: 0cb9bcee58dbd242ffce1da43e080af4e23be18a0c55a231cab97a4c6812ac6e

Nº de intentos: 5

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

Cadena de bits: 1100000001011000111111101001100010011001100100001011010000101110
1110101101001011111111100000110111111001010010010000000110101001011011100100011

ID: 0cb9bcee58dbd242ffce1da43e080af4e23be18a0c55a231cab97a4c6812ac6ePatricia

Hash: 2d6d0eefe2394b1dfff01ebd48a53c6b2703e8f7ad8ec200e2da1bf5e919fcdd

Nº de intentos: 1

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

Cadena de bits: 0111001000001000111101100011010110101011111101100100001010011010
00110110001010001110001111001000101110011000101101100011100100100111101001001010

ID: 2d6d0eefe2394b1dfff01ebd48a53c6b2703e8f7ad8ec200e2da1bf5e919fcddPatricia

Hash: 359fe5d3e9605f177cf07767898b12b198ab7d23035136f3b362b280525db823

Nº de intentos: 1

```

```

*****Bloque 7 *****

Cadena de bits: 1101010001000001110001111000100010011001100101101100000010100100
10101101011011010101001010101101101101101010011010101000011011110101101011101100

ID: 359fe5d3e9605f177cf07767898b12b198ab7d23035136f3b362b280525db823Patricia

Hash: 1c3203bd5886b9415db22cd3e93e856994ecf16a390a2c9ce83ad8f71367b12f

Nº de intentos: 3

*****Bloque 8 *****

Cadena de bits: 0000000010010011001110101110000011101110111011010010010011011010
11101000100110010110000111011010110111110111000010111101101001100010011000100100

ID: 1c3203bd5886b9415db22cd3e93e856994ecf16a390a2c9ce83ad8f71367b12fPatricia

Hash: 24411e08cbda9f81860ab03f513a66b27750bd5438a7642ec47c849309060e96

Nº de intentos: 5

*****Bloque 9 *****

Cadena de bits: 0111011100100001101111010111010111000101101100110110010010001111
1010001110110111111100000010001010111111010000110000111100000000101101011100111

ID: 24411e08cbda9f81860ab03f513a66b27750bd5438a7642ec47c849309060e96Patricia

Hash: 11707d13d22fa3862d5155ad121d6859d834b6747ddabfb263d7b85727580323

*****Bloque 10 *****

Cadena de bits: 1011101110111100111000111101011110101100001101111100101010100111
010001010101011111011010011000110010100001110001101101001001011100110010011110101

ID: 11707d13d22fa3862d5155ad121d6859d834b6747ddabfb263d7b85727580323Patricia

Hash: 187af5929442de28b62a08e422a28c3af16d9326e321b8cd4c98263dcd4235e6

Nº de intentos: 9

```

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

```

*****Bloque 10 *****

Cadena de bits: 0011100001101010111001100010010011110010110000101111010101000101
11100101110101011111101010110011111000111110001001001000100001000011011110111010
011011100110000101000110010001010010010000000111111001010101111110100110010111
00010010110110011100000110110111

ID: 0aaa39e177ed1378bc89cc7b0a2ba01618d616844173fd64c5f3700cd9a0c7cdPatricia
Hash: 305f9c7d2ca69599bdlf96ce241e46551453alebe0f19153a4a9723907ddf65a
Nº de intentos: 9

*****Bloque 11 *****

Cadena de bits: 0101010101000000000011011001001100000001011111100010001000010011
11001000010011000110010010011111010100000000110110011110110011011000111011010111
00001101100100010001101111101101100010010110100100100000011110011011110101
00010001011010111101011111100111

ID: 305f9c7d2ca69599bdlf96ce241e46551453alebe0f19153a4a9723907ddf65aPatricia
Hash: 041ec23dfc6f17fc7add8d04597b456b6bdd45dee0285463664894b4e1932c70
Nº de intentos: 6

*****Bloque 12 *****

Cadena de bits: 1111111100011110101111111000001101001011101010001000101110101110
01011100101001100110110110011010100100010101100110000001001011101011010010011100
00101000110011000001011111100001110111100110111100100100011010111111001000001100
101010000111110000100000010001010

ID: 041ec23dfc6f17fc7add8d04597b456b6bdd45dee0285463664894b4e1932c70Patricia
Hash: 07cc0d615770113d844737c11eb626ae016272668db3a8580617a8701ee33b64
Nº de intentos: 15

```

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.

```

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)~
    # # Sigüientes 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

```

*****Bloque 24 *****

Cadena de bits: 0010111001111101100000011100010101111101011101010110010101100111
01101011011001110011000100001011010111111001010011011000000001010000111011011011
1010010011010001100000110100000001110100100111111010000011111001000011000111110
10011010001011011111111100100001

ID: 00002ebcd9d2577cb63865e73b48dbe8f769168dcb2a6236d65955b95e10a58aPatricia
Hash: 000025a1ebcaceed88780310205248ccca8b5f7d0d087b42d296f7ce394b22d5
Nº de intentos: 32711
VALOR B: 16 <---

*****Bloque 25 *****

Cadena de bits: 1011101111110101011100111011110010110101101011111001000000111101
00001110100011011010100011000110111110010110001010000001011100011011110110111001
11000000001011001111100011101101100100011010010000010110001111010111001101001101
11000101101000010011000110001101

ID: 000025a1ebcaceed88780310205248ccca8b5f7d0d087b42d296f7ce394b22d5Patricia
Hash: 00006a50f2f199e1bd158831352f31f8c65b0137802acd1ab7bd8f0d0017f221
Nº de intentos: 199235
VALOR B: 17 <---

```

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

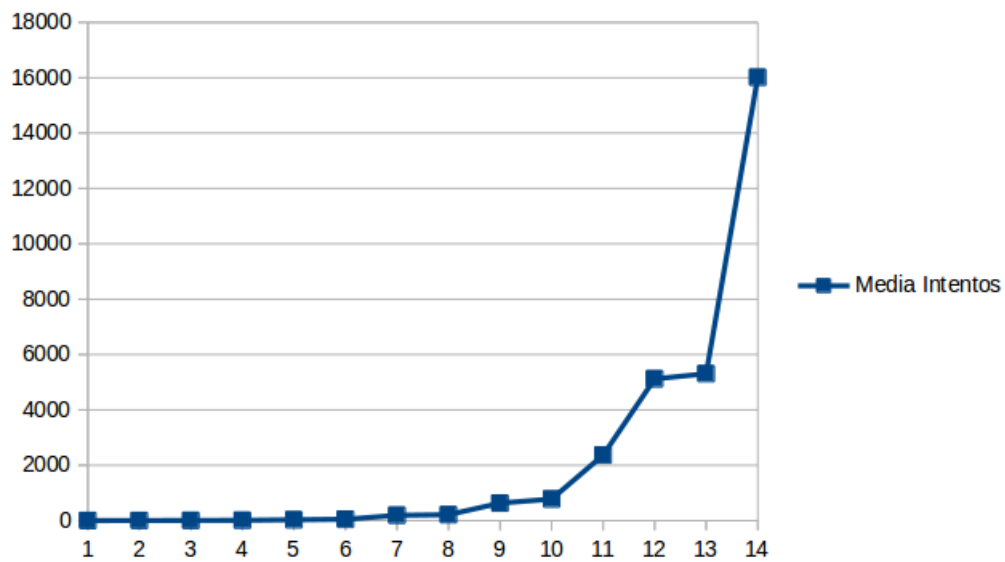


Figura 5.2: Grafica ejercicio 1

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