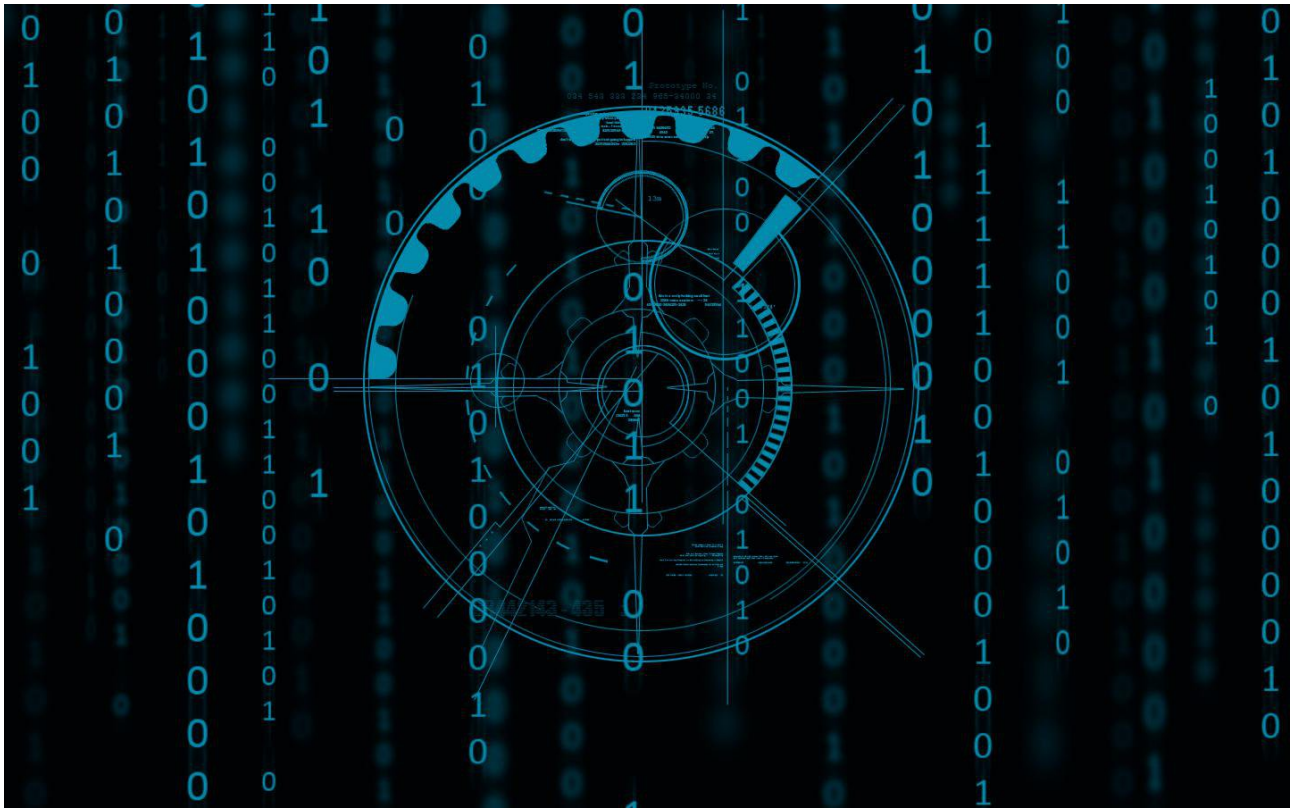
Práctica 5

PUZZLES HASH



Manuel Ruiz Maldonado

Índice

Para la realización de esta práctica he decidido utilizar Python3 como lenguaje de programación.

La función Hash escogida para calcular el valor hash requerido en este caso va a ser SHA-384 con salida de n = 384 bits.

Ejercicio 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 contatene 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.

En este ejercicio se nos pide crear una función cuyas entradas sean un texto y un número. A partir de ese texto se creará lo que se llama *id*, que es la concatenación de una cadena aleatoria de *n* bits (o *n/4* caracteres hexadecimales) junto con el texto. Además tendré que generar otra cadena *x* también aleatoria de *n* bits (o de nuevo, *n/4* caracteres hexadecimales). Por lo tanto, lo primero que he hecho ha sido crear una función que devuelve una cadena aleatoria de caracteres hexadecimales, dado el tamaño que se desee. El código de esta función se puede ver en la Imagen 1.

```
def generarHex(n):
    '''
    Función que devuelve una cadena aleatoria de caracteres hexadecimales de
    tamaño n.
    '''

    salida = ""
    for i in range(n):
        r = '%1x' % randint(0,15)
        salida += r
    return(salida)
```

Imagen 1: Código de la función que genera cadenas hexadecimales.

Esta función genera números aleatorios entre 0 y 15, y se transforman a valores hexadecimales que se van concatenando en una cadena de texto que será la salida.

Una vez hecho esto hay que seguir los siguientes pasos dentro de la función que realizará la tarea del ejercicio 1:

1. Crear *id*: concatenar una cadena hexadecimal aleatoria con el texto inicial.
2. Crear *x*: generar otra cadena hexadecimal aleatoria.
3. Calcular valor hash: concatenar *id* y *x* y calcular su valor hash con la función comentada al principio de la práctica.
4. Calcular bits iniciales: pasar ese valor hash a bits y calcular el número de ceros que hay al inicio de la cadena.
5. Si el número de ceros al inicio de la cadena es al menos el valor *b* indicado como parámetro de entrada de la función, entonces se devuelven los datos que se solicitan como salida, sino, se tiene que volver al paso 2.

Este proceso lo he recogido en una función utilizando la ya comentada función de la Imagen 1,

además de la función *sha384()* de la librería *hashlib* de Python para el cálculo del hash. El código de la función creada se puede observar en la Imagen 2.

```python
def Ejercicio1(texto, b):
    '''
    texto: cadena de caracteres
    b: número de bits que al inicio deben ser ceros
    '''

    n = 384
    idInicial = generarHex(int(n/4)) + texto
    encontrado = False
    intento = 0
    while(not encontrado):
        intento += 1
        x = generarHex(int(n/4))
        idAux = idInicial + x
        hashObtenido = sha384(idAux.encode('utf-8')).hexdigest()
        cadenaBits = bin(int(hashObtenido, 16))[2:].zfill(n)
        if(int(cadenaBits[0:b],2) == 0):
            encontrado = True
            print("ID = ", idInicial, "\nCadena x = ", x,
                  "\nHash = ", hashObtenido, "\nIntentos = ", intento)
            #print("Intentos = ", intento)
```

Imagen 2: Código de la función del Ejercicio 1.

El resultado obtenido tras ejecutar la función del Ejercicio 1 para un ejemplo concreto es el de la Imagen 3.

```
manolo@manolo-Parallels:~/Documentos/SPSI/P5$ python3 Ejercicio1.py

**************************************************
Ejercicio 1
Texto de entrada: Bienvenido a SPSI - Seguridad y Proteccion en Sistemas Informaticos
Número de ceros: 6

ID =  6b30bc275a5db8c1639cd86ccc5c10a7a82495ac0629e411e0f2b88e9bcc51635e7967abe4bb7f617f6eaf3b
ff22c9e1Bienvenido a SPSI - Seguridad y Proteccion en Sistemas Informaticos
Cadena x =  b34f2205856e18fd511725372e091054c4cc48e812e2a8fa269ec5687fdb35fb162c01e4554d9f3d31
090b48aedc9a48
Hash =  035d82f1434fe09946216e9cb4b045bd96bf46a05be6dc0b3679597aad11b0e8562ef91326dc85127613f3
21da6c029a
Intentos =  74
**************************************************

manolo@manolo-Parallels:~/Documentos/SPSI/P5$
```

Imagen 3: Resultado obtenido del Ejercicio 1.

Ejercicio 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.

Para realizar este ejercicio voy a realizar una pequeña modificación en el Ejercicio 1, de modo que

en lugar de devolverme como salida todos los datos que allí se pedían se devuelva únicamente el número de intentos obtenidos en cada ejecución. Una vez hecho esto voy a repetir el ejercicio 10 veces y calcularé sus medias para poder dar una aproximación del número de intentos que esta función necesita para conseguir alcanzar el número de ceros al inicio que se pida. Además voy a calcularlo para un total de hasta 15 ceros iniciales.

La tabla que contiene todos los valores de las distintas ejecuciones así como sus medias se puede observar en la Imagen 4.

Número de ceros	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Intentos	4	1	18	11	12	26	88	229	780	1455	6365	1153	9968	207	11433
Intentos	1	6	1	33	39	115	204	2	474	639	788	7728	1512	40304	11066
Intentos	3	3	4	6	27	64	39	233	380	77	5715	3012	1161	9987	7496
Intentos	1	4	4	1	10	145	127	192	340	1046	407	4239	11496	12155	21786
Intentos	4	2	10	30	44	80	7	110	162	2251	75	6968	3482	4013	2961
Intentos	1	3	3	22	67	2	308	981	48	811	2170	1208	10502	13982	9927
Intentos	1	12	5	60	6	122	39	109	654	973	8460	9947	3603	21274	39592
Intentos	3	6	9	2	44	152	337	393	687	1606	4051	19268	9266	4258	11717
Intentos	2	9	7	1	2	19	196	209	905	782	1233	3637	3967	7908	9095
Intentos	1	5	6	2	23	36	97	603	1940	1737	149	27	2882	780	40466
Media	2	5,1	6,7	17	27	76,1	144	306	637	1138	2941	5718,7	5783,9	11487	16554

Imagen 4: Tabla resumen correspondiente a la función del Ejercicio 1.

A partir de dichas medias se puede crear una gráfica que muestre la relación entre el número de ceros que se quieren conseguir al inicio de la cadena y el número de intentos que se han realizado para conseguirlo. Esta gráfica, que se puede ver en la Imagen 5, muestra claramente una distribución exponencial en el número de intentos cada vez que se aumenta el número de ceros que se desean encontrar al inicio.

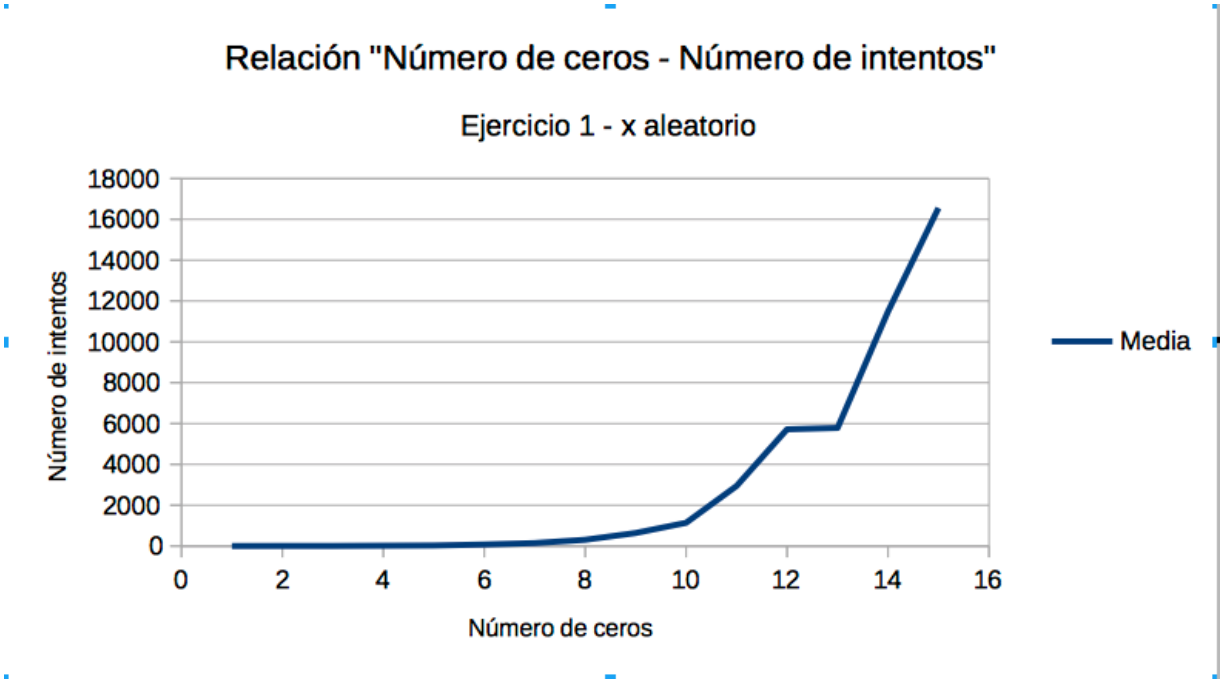


Imagen 5: Gráfica resumen correspondiente a la función del Ejercicio 1.

Ejercicio 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.

Para este ejercicio vamos a realizar una pequeña modificación con respecto al original del Ejercicio 1. Esta modificación tendrá lugar en el punto 2 de las tareas que debe realizar la función. Justamente, en la parte de "crear x". En este caso, en lugar de generar x aleatoriamente cada vez que se realiza una nueva iteración del bucle, se tendrá que generar x solamente en el primer caso, y en todos los siguientes lo que se realizará será aumentar en 1 el valor de x y continuar con el proceso normal.

Así, el código modificado de esta función queda como se puede observar en la Imagen 6.

```
def Ejercicio3(texto, b):
    '''
    texto: cadena de caracteres
    b: número de bits que al inicio deben ser ceros
    '''

    n = 384
    idInicial = generarHex(int(n/4)) + texto
    x = generarHex(int(n/4))
    encontrado = False
    intento = 0

    while(not encontrado):
        intento += 1
        if intento > 1:
            xAux = int('0x'+x, 16)+1
            x = '%lx' % xAux
        idAux = idInicial + x
        hashObtenido = sha384(idAux.encode('utf-8')).hexdigest()
        cadenaBits = bin(int(hashObtenido, 16))[2:].zfill(n)
        if(int(cadenaBits[0:b],2) == 0):
            encontrado = True
            print("ID = ", idInicial, "\nCadena x = ", x,
                  "\nHash = ", hashObtenido, "\nIntentos = ", intento)
            #print("Intentos = ", intento)
```

Imagen 6: Código de la función del Ejercicio 3.

El resultado obtenido tras ejecutar la función del Ejercicio 3 para un ejemplo concreto es el de la Imagen 7.

```
manolo@manolo-Parallels:~/Documentos/SPSI/P5$ python3 Ejercicio3.py

**************************************************
Ejercicio 3
Texto de entrada: Bienvenido a SPSI - Seguridad y Proteccion en Sistemas Informaticos
Número de ceros: 6

ID =  229e70543bc48aca231f9deb54f35e4420e476a2f750d9e6ea3835f9fc7e6b9e49e12bb927ab033ee77f4932
ad19c8c3Bienvenido a SPSI - Seguridad y Proteccion en Sistemas Informaticos
Cadena x =  fd47dd1d81abd576ef3366d36751074d11f58062657f846911ecfc2353c7d47bbdddfd5bbca3cc815f
ad3242b3d92ecf
Hash =  0236ff7e6603e75c48cb4f22336be75e7027ec47a39686efeed806ed292619dd86735d06adee1d2f6196d4
3ec0d2654e
Intentos =  81
**************************************************

manolo@manolo-Parallels:~/Documentos/SPSI/P5$
```

Imagen 7: Resultado obtenido del Ejercicio 3.

Ejercicio 4

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

De nuevo, repitiendo lo que hacía en el Ejercicio 2 puedo ejecutar varias veces con distintos valores de b y recoger a modo de resumen qué ocurre en cada caso. Para ello se pueden ver los resultados obtenidos en la tabla de la Imagen 8.

Número de ceros	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Intentos	1	2	3	31	83	44	105	172	806	921	1170	6259	19786	1920	14752
Intentos	2	1	4	12	2	33	4	420	130	1013	3018	1248	43962	4276	152085
Intentos	4	13	5	15	2	27	39	4	297	803	661	1558	3728	28711	20529
Intentos	1	2	5	11	26	15	29	497	1778	91	4013	6538	9710	3790	11335
Intentos	2	3	14	1	14	40	538	83	1934	365	16	1158	4563	3517	883
Intentos	2	6	6	14	28	14	3	91	670	478	574	3322	1276	47016	46720
Intentos	2	5	8	30	9	54	14	150	461	1396	2671	1330	15006	3685	88691
Intentos	1	12	3	8	12	16	82	310	9	1122	782	11565	713	25930	40861
Intentos	1	1	4	1	6	12	222	12	1641	853	1181	9687	1934	11679	34056
Intentos	3	1	9	47	91	115	158	312	467	1680	538	829	3601	4538	22679
Media	2	4,6	6,1	17	27	37	119	205	819,3	872,2	1462	4349,4	10428	13506	43259,1

Imagen 8: Tabla resumen correspondiente a la función del Ejercicio 3.

Con esa tabla se puede volver a crear una gráfica, Imagen 9, que muestre la evolución de los resultados.

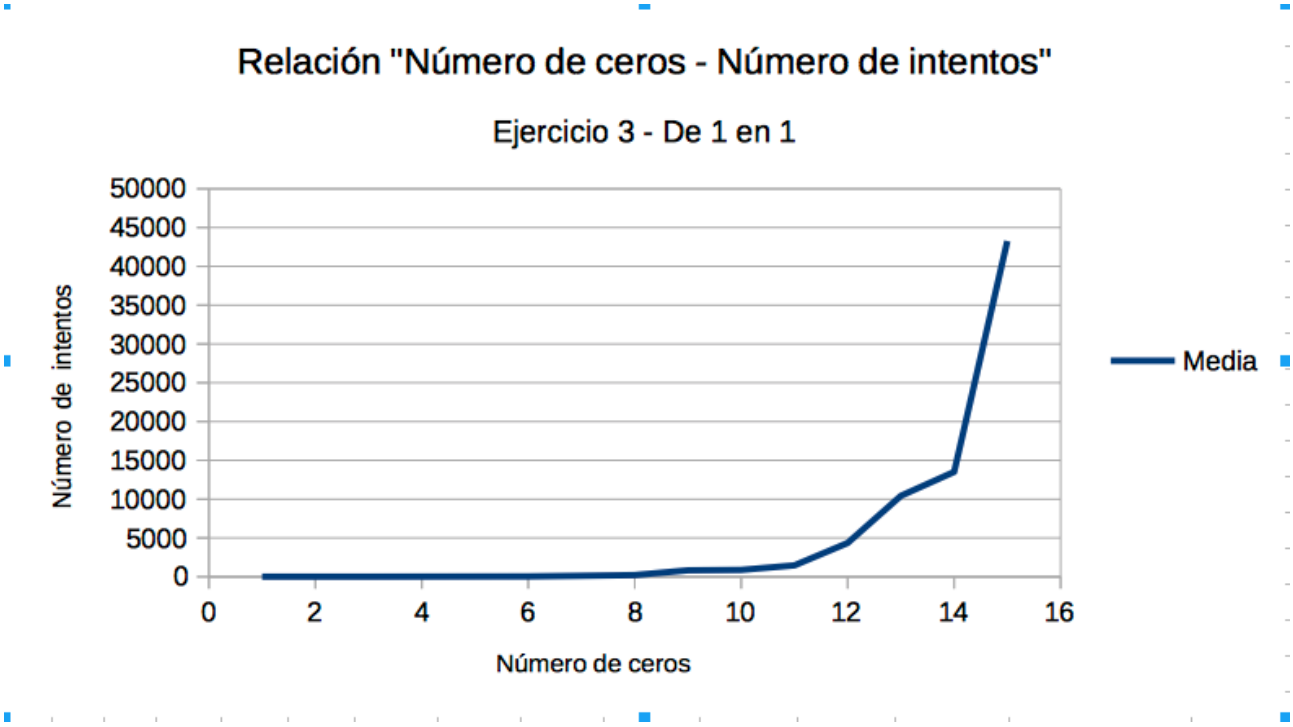


Imagen 9: Gráfica resumen correspondiente a la función del Ejercicio 3.

Se puede realizar una comparación de estos dos métodos superponiendo una gráfica encima de otra como se puede ver en la Imagen 10.

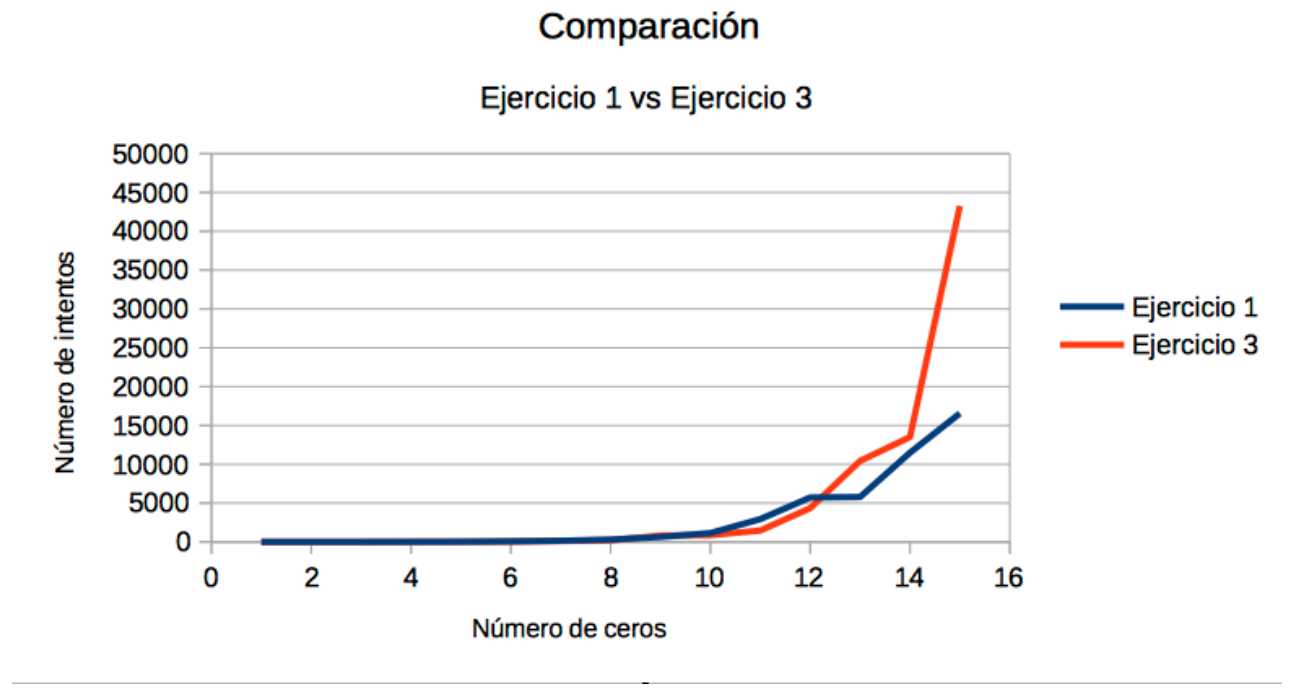


Imagen 10: Comparación de resultados obtenidos.

Como se puede observar el método utilizado en el Ejercicio 3 parece ser bastante peor en cuanto a número de intentos se refiere, según se va aumentando el número de ceros que se quiere conseguir.

En cambio, el tiempo requerido para calcular todas estas ejecuciones se reduce mucho en el Ejercicio 3 con respecto al Ejercicio 1. Con esto puedo decir que la forma de aumentar en 1 el valor de x es mucho más rápido que generarlo completamente de nuevo, pero esto no implica que se llegue en menos intentos al valor deseado.

Índice de imágenes