Solución al Reto 7 de S21sec

Febrero 2009 © Daniel Kachakil

Introducción

Este documento describe una posible solución al Reto 7 de S21sec, publicado el 7 de enero de 2009 en la siguiente dirección web:

`http://blog.s21sec.com/2009/01/reto-7-te-haces-rico-con-nosotros.html`

El reto consiste en ver si somos capaces de "predecir el futuro" de una forma demostrable. Concretamente, se trataba de generar un documento PDF en el que debía aparecer la combinación ganadora del sorteo de EuroMillones del día 20/02/2009, debiendo publicar la firma MD5 del fichero antes de la fecha del sorteo, como es lógico. Posteriormente, se debía publicar el documento, demostrando que efectivamente había sido generado con anterioridad al sorteo y que nunca ha sido alterado, puesto que su firma MD5 debe permanecer intacta.

Además, para evitar trampas y trucos, se especifica que el documento PDF se abrirá en un ordenador sin conexión a Internet, para evitar que actúe como un simple visor y obtenga su contenido de forma dinámica.

En el reto hay tres elementos en los que deberemos profundizar para ver cómo podemos abordar la solución. Empezaremos explicando en qué consiste el juego de los EuroMillones, continuaremos con las debilidades del algoritmo MD5 y terminaremos con el análisis del formato PDF, viendo cómo podemos solucionar el reto.

El sorteo de EuroMillones

En este juego de lotería participan varios países de forma oficial (Francia, Reino Unido, España, Austria, etc). El juego consiste en acertar la combinación de números que se sorteará en una fecha determinada. Cada apuesta sencilla consta de dos partes: una tabla con 50 números (del 1 al 50), de la que se marcan 5 y otra tabla con 9 estrellas (del 1 al 9), de la que se marcan 2. Sin necesidad de profundizar en el reglamento del juego de EuroMillones, únicamente nos quedaremos con la idea de que, evidentemente, el premio más alto es el que se corresponde con todos los aciertos (5 números y 2 estrellas) y es el que tendremos que "pronosticar" en el reto.

Esta configuración admite 76.275.360 combinaciones posibles, por lo que la probabilidad de acertar el premio máximo es inferior a 1 entre 76 millones. Por tanto, en el caso de que nos hiciera falta, vemos que computacionalmente sería relativamente sencillo y factible obtener un documento con todas las combinaciones posibles, o un documento diferente para cada combinación posible. De todas formas, aún es pronto para adelantar acontecimientos y ya veremos si esto nos hace falta o no…

El algoritmo MD5 y sus debilidades

El algoritmo MD5 (*Message Digest algorithm 5*) es un algoritmo estandarizado (RFC 1321), diseñado por Ronald Rivest en 1991 y que se engloba dentro de las funciones de *hashing* o de resumen criptográfico. Estas funciones reciben como entrada un determinado conjunto y obtienen como salida otro conjunto normalmente menor. En nuestro caso, el algoritmo MD5 puede recibir como entrada una secuencia ilimitada de valores o bytes (los cuales serán procesados internamente en bloques de 512 bits, aplicando técnicas de relleno), pero siempre obtiene como salida una secuencia de 128 bits, cuya representación habitual se indica usando 32 caracteres en hexadecimal. La más mínima variación en la entrada, normalmente producirá un gran cambio en la salida, debido al efecto de avalancha, tal y como se muestra a continuación:

```
MD5("prueba") = c893bad68927b457dbed39460e6afd62
MD5("Prueba") = 5bc8c567a89112d5f408a8af4f17970d
MD5("PRUEBA") = 3614f7b8bf42dbb37b040c9387ddc1f0
MD5("PRUEBO") = b2240106aa37c4c8d7ccf6ad3be4b4b3
```

Las funciones de hash tienen la propiedad de que dos salidas diferentes siempre proceden de dos entradas diferentes, pero no necesariamente se cumple la propiedad inversa (es decir, dos entradas diferentes sí pueden coincidir en una misma salida o valor de hash, lo cual es lógico si tenemos en cuenta que reducen el conjunto de entrada). Cuando se utilizan en criptografía, es deseable que estos algoritmos sean resistentes a varios tipos de ataque: colisión, preimagen y segunda preimagen (descritos brevemente aquí):

- **Resistencia a colisión:** Dificultad para encontrar dos mensajes diferentes cualesquiera que tengan un mismo hash.
- **Resistencia a preimagen:** Dificultad para invertir la función de hash (o de encontrar un mensaje cualquiera cuyo hash coincida con un valor específico).
- **Resistencia a segunda preimagen:** Dificultad para encontrar un mensaje cualquiera cuyo hash coincida con el de otro mensaje específico.

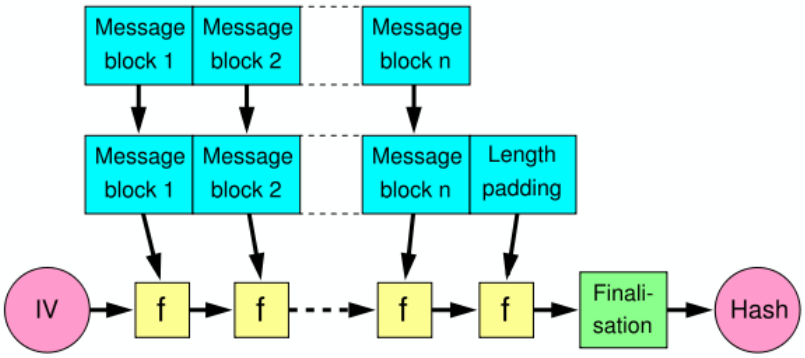
Inicialmente no se pudo demostrar ninguna vulnerabilidad del algoritmo MD5, por lo que se consideró válido para su uso criptográfico. Esta razón es la que hace que este algoritmo se utilice hoy en día de forma muy extendida como "garantía" para comprobar la integridad de un mensaje o fichero, e incluso para identificarlo con mucha fiabilidad (con una probabilidad de confusión teóricamente ínfima).

Por ejemplo, una aplicación de copias de seguridad remotas podría calcular el MD5 de un fichero para comprobar si ha sido modificado sin necesidad de tener que enviarlo. Una aplicación P2P o un repositorio de ficheros podrían usar el hash MD5 para identificar un mismo fichero y evitar duplicados, aunque tenga nombres diferentes. Este algoritmo también se suele utilizar en mecanismos de autenticación para la verificación y el almacenamiento de contraseñas, de forma que permita evitar el tener que enviar una contraseña en texto plano a través de un canal inseguro, a la vez que impide la obtención de la contraseña original a partir de su hash MD5 (bueno, eso si no tenemos en cuenta la existencia de tablas precalculadas, etc).

Sin embargo, es habitual que con el tiempo se acabe descubriendo alguna debilidad en este tipo de algoritmos, haciéndolos vulnerables a alguno de los ataques descritos antes. En este caso, decir que es vulnerable es equivalente a decir que existen

métodos capaces de reducir la complejidad computacional del algoritmo, comparando con un ataque de fuerza bruta o con un ataque de cumpleaños, según sea el caso. Precisamente debido a estas debilidades descubiertas y al incremento de la capacidad computacional, surge la continua necesidad de diseñar nuevos algoritmos (como SHA-1 o el futuro SHA-3).

Es bien sabido que el algoritmo MD5 no es una excepción a esta regla, ya que desde 1993 se han ido publicando diferentes documentos y técnicas que han revelado varias debilidades en el mismo. Los primeros documentos no tuvieron prácticamente ninguna repercusión, dado que no podían ser aprovechados por ningún ataque concreto, porque no usaban exactamente la misma configuración que el MD5. En agosto de 2004, Xiaoyun Wang publica el primer documento con colisiones de MD5 y desde ese momento comienzan a publicarse técnicas de ataque que aprovechan esa debilidad, basadas en los principios que describió Dan Kaminsky en este conocido documento: *"MD5 To Be Considered Harmful Someday"*.

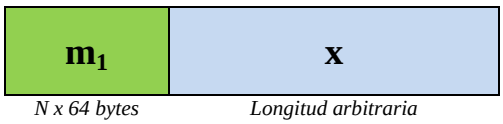


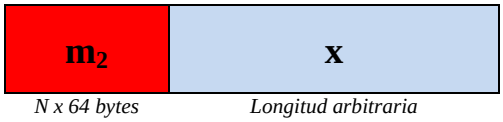
Puesto que el algoritmo MD5 utiliza la construcción de Merkle-Damgård para facilitar el procesamiento de un número ilimitado de bloques, entonces se cumple la siguiente condición matemática:

```
Si MD5(m1)=MD5(m2), entonces MD5(m1+x)=MD5(m2+x)
```

Es decir, dadas dos secuencias diferentes (m_1 y m_2) con el mismo hash (como resultado de una colisión), si concatenamos cualquier secuencia arbitraria (x) al final de ambas secuencias, entonces las dos secuencias resultantes también tendrán el mismo hash MD5. Nótese que no se está afirmando que el hash MD5(m_1) tenga que coincidir con el hash MD5(m_1+x).

De forma gráfica, estas dos secuencias tendrían el mismo hash MD5, siempre que MD5(m_1)=MD5(m_2):

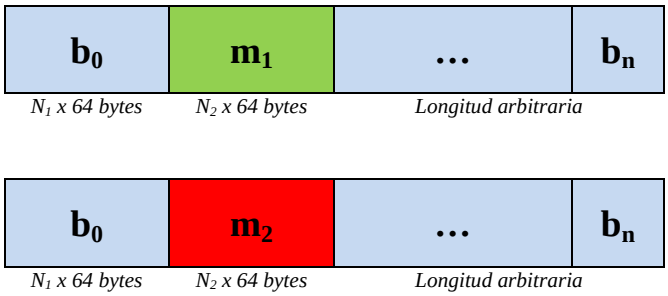




Puesto que la colisión entre m_1 y m_2 se genera de forma aleatoria, estos mensajes solo podrán usarse en determinadas condiciones particulares (por ejemplo, en cierto tipo de aplicaciones específicamente diseñadas para tal fin). De hecho, las técnicas de "falsificación" basadas en el uso de esta condición necesitan de cierto tipo de procesamiento o interpretación de instrucciones del tipo IF-ELSE (macros, lenguajes de script, etc) y además deben contener ambos mensajes (el que se visualiza y el que no), por lo que resultará fácil descubrir la trampa si se analiza el contenido del fichero.

Podríamos pensar que puesto que el algoritmo MD5 se aplica a nivel interno sobre bloques de 512 bits (64 bytes), entonces también podemos añadir cualquier cadena o secuencia inicial, siempre que su longitud sea un múltiplo de 64 bytes, pero cuidado porque esto no es cierto, es decir, $MD5(b_0+m_1)\neq MD5(b_0+m_2)$.

Teniendo en cuenta las condiciones de colisión descritas anteriormente, es decir, siendo $MD5(m_1)=MD5(m_2)$ y siendo el tamaño del bloque b_0 un múltiplo exacto de 64 bytes, estas dos secuencias **no** tendrían el mismo hash MD5:



Esto sucede porque el estado interno del algoritmo MD5 se modifica tras el procesamiento del bloque inicial b_0 y la colisión entre m_1 y m_2 estaba calculada cuando ambos mensajes eran los primeros en aparecer, es decir, cuando el vector de inicialización (IV) del algoritmo MD5 se corresponde con el vector inicial (IV_0), el cual fue establecido por su creador con el siguiente valor en notación hexadecimal: 0123456789ABCDEFFEDCBA9876543210

Sin embargo, existen métodos y aplicaciones software que permiten obtener colisiones MD5 a partir de cualquier vector de inicialización. Por ejemplo, el que implementa el *MD5 Collision Generator* de Marc Stevens es capaz de obtener una colisión MD5 aleatoria en pocos segundos, incluso ejecutándose en cualquier ordenador doméstico. Con este software sí que podemos calcular una o varias colisiones válidas tal que $MD5(b_0+c_1)=MD5(b_0+c_2)$, siendo c_1 y c_2 dos colisiones diferentes a m_1 y m_2.

Por ejemplo, existe alguna referencia describiendo cómo se pueden construir dos ficheros PostScript cualesquiera con el mismo hash MD5, pero visualizando mensajes completamente diferentes en cada uno de ellos. Estos documentos se generan con un bloque inicial (múltiplo de 64 bytes) con la cabecera del PostScript y con la apertura de una instrucción IF. A continuación, se calcula una colisión para ese vector de inicialización (que será reutilizable para cualquier documento generado de con ese mismo bloque inicial). A partir de ahí, tenemos dos posibles inicios de documento con el mismo hash MD5, por lo que solo falta añadir al IF una condición de igualdad con cualquiera de los valores de la colisión, de forma que si una de las colisiones es igual a sí misma, se muestra un mensaje y en caso contrario se muestra otro mensaje.

% ... if (	c_1	== c_1) {*mensaje1*} else {*mensaje2*}
N_1 x 64 bytes	N_2 x 64 bytes	Longitud arbitraria

% ... if (	c_2	== c_1) {*mensaje1*} else {*mensaje2*}
N_1 x 64 bytes	N_2 x 64 bytes	Longitud arbitraria

Esta misma técnica también se ha utilizado para permitir crear dos ficheros ejecutables con el mismo hash MD5, de forma que uno de ellos ejecutará una parte del código (por ejemplo, un código inofensivo), mientras que el otro ejecutará otra parte (posiblemente un código malicioso).

Por otro lado, también existe alguna referencia a la aplicación de esta misma técnica sobre otros formatos de ficheros, como TIFF, Microsoft Word y PDF (en la introducción de este documento también se explica cómo se podría aplicar a ficheros PostScript). Puesto que el reto nos pide obtener un fichero PDF, este documento podrá sernos de gran utilidad, ya que describe una forma interesante de obtener dos ficheros PDF con el mismo hash, basándose en escalas de grises y funciones de transferencia que analizaremos en el siguiente punto.

Sin embargo, todo lo que hemos visto hasta el momento se centra en obtener únicamente dos ficheros diferentes con el mismo hash MD5, pero esto no es suficiente para poder generar nuestro PDF con la predicción de EuroMillones (a no ser que realmente seamos videntes y solo dudemos entre dos posibles combinaciones…). Aquí es cuando entra en juego el concepto de multicolisión, es decir, la sucesiva concatenación de colisiones para ampliar las posibilidades que ofrecería cada una de ellas por separado. En el documento *"Herding Hash Functions and the Nostradamus attack"* se describen varias técnicas que permiten hacer "predicciones" sobre el futuro para diversos casos. Si nos fijamos en la figura 3 de dicho documento, observamos que aparecen una serie de nombres de personas de las que tendríamos que predecir si se casarán o no en el año actual:

Fig. 3. Using Joux Multicollisions to Predict Who Will Get Married

Si esta técnica funciona para ese caso, es fácilmente deducible que también podremos utilizarla para nuestro caso particular, en el que cada nombre podría sustituirse por un número o estrella del sorteo de EuroMillones y en vez del concepto de casarse, podemos aplicar el concepto de si saldrá o no en el sorteo. Puesto que en nuestra predicción tenemos 59 variables (50 números y 9 estrellas), tendremos que formar una cadena de 59 colisiones, cuyo resultado final siempre será el mismo hash MD5 y en el que podemos sustituir cualquier colisión de forma independiente.

Obtención de colisiones MD5 sobre el formato PDF

Ya hemos visto cómo se podría sacar provecho de una colisión MD5 para el formato de fichero PostScript, mediante el uso de bloques IF-ELSE. El formato PDF no implementa una estructura de procesamiento similar (siempre que no se haga uso de JavaScript), pero también hemos comentado que existen técnicas que muestran y ocultan un texto de forma visual utilizando escalas de grises y funciones de transferencia.

Al igual que un documento PostScript, un PDF también comienza con una cabecera que lo caracteriza y luego se compone de una serie de secciones que contienen diversos tipos de objetos que forman el documento en sí. En el propio blog de S21Sec encontramos un documento de introducción al formato PDF y otro que describe la estructura de un documento básico en PDF. Teniendo presente esta información y por supuesto la referencia oficial del formato PDF de Adobe (es suficiente con la referencia del formato PDF en su versión 1.3), podemos empezar a crear un fichero PDF con el propio bloc de notas, pero antes de eso vamos a concretar nuestro objetivo concreto.

Vamos a basarnos en la técnica que describe G. Illies en el documento al que ya habíamos hecho referencia antes: *"A Note on the Practical Value of Single Hash Collisions for Special File Formats"*. En este documento se muestra un ejemplo parcial de un documento PDF que muestra un mensaje u otro según se haya incrustado en él un valor de la colisión o el otro valor.

En su ejemplo, podemos analizar cómo hace uso de un espacio de color que define una paleta de grises indizada. Precisamente esta paleta es la que contendrá la colisión MD5, teniendo en cuenta que siempre aparece delimitada entre paréntesis y que admite todos los valores de byte posibles, salvo el de apertura y cierre de paréntesis (es decir, los valores 40 y 41 en ASCII). Esta afirmación es la que hace el autor, pero por las pruebas que hice, yo diría que se le olvidó mencionar otro carácter que tiene un comportamiento especial y que tampoco se admitiría tal cual en una paleta de grises: la barra inversa ("\", valor 92 en ASCII). Esta barra se usa normalmente como carácter de escape, haciendo que se interprete como un tabulador (\t) o como un salto de línea (\n), por lo que no admite cualquier carácter posterior.

Para crear un objeto que contenga una paleta de grises indizada usaremos la siguiente estructura, en la que el valor 100 corresponde al número de objeto, el valor 9 indicaría el índice máximo de la cadena de texto (es decir, la longitud total menos uno) y la cadena de valores posibles se indica entre paréntesis:

```
100 0 obj [
  /Indexed
  /DeviceGray 9
  (ABCDEFGHIJ)
] endobj
```

Tal vez la primera tentativa sea la de crear una única paleta con tantos valores como nos hiciera falta, pero comprobamos que está limitada a un máximo de 256 caracteres. Afortunadamente, en el documento podemos definir tantas paletas como sea necesario y en una página también podremos utilizar tantas como queramos, siempre que se definan en la correspondiente sección de recursos.

Ya que el software *MD5 Collision Generator* siempre genera colisiones de un tamaño de 128 bytes, podemos usar una misma paleta para albergar dos colisiones, de forma que en total necesitaremos 30 paletas diferentes. Pero en general no es suficiente con las paletas, ya que las colisiones se generarán de forma aleatoria y sería muy complicado obtener una colisión tal que uno de sus valores se corresponda con un color negro (o muy oscuro) y el otro se corresponda con un color blanco (o muy claro). Por tanto, el método requiere de la aplicación de funciones de transferencia y estados gráficos extendidos que vinculen una serie de valores de entrada a otros de salida controlables por nosotros a través del uso de ciertas instrucciones y cálculos básicos.

El formato PDF soporta ciertos tipos de funciones usadas normalmente para el ajuste de colores. Nos centraremos en las funciones de tipo 4 (*PostScript calculators*), ya que permiten interpretar un subconjunto de instrucciones del lenguaje PostScript, entre las que se encuentran bloques IF-ELSE. Estas funciones reciben un valor decimal entre 0 y 1, y devuelven un valor en el mismo rango (en nuestro caso, estos valores se corresponden con el color negro y el blanco, respectivamente).

Puesto que estamos trabajando con cadenas de texto que representan escalas de grises, cada valor ASCII se interpreta como valor decimal entre 0 y 1, por lo que se obtiene dividiendo el valor actual entre 255. Por ejemplo, el carácter "A" (65) se convierte en 0,2549 (65/255). Buscamos alguna forma de obtener una función que devuelva un valor determinado (por ejemplo, el correspondiente al color negro) cuando reciba como entrada un carácter concreto y que devuelva otro valor (por ejemplo, el correspondiente al color blanco) cuando recibe cualquier otro carácter.

Analizando las instrucciones que tenemos a nuestro alcance, podemos implementar una función que realice lo siguiente:

```
if (abs(entrada - 0.2549) < 0.0005) {return 0} else {return 0.9}
```

Esta función se modela de la siguiente forma, teniendo en cuenta que se procesa usando una pila y su sintaxis se basa en la notación polaca inversa:

```
200 0 obj
  << /FunctionType 4
     /Domain [0.0 1.0]
     /Range [0.0 1.0]
     /Length 42
  >>
stream
  { 0.2549 sub
    abs
    0.0005 lt
    {0}{0.9} ifelse
  }
endstream
endobj
```

Al igual que ocurría con las paletas de grises, nada impide que tengamos tantas funciones de este estilo como sea necesario, por lo que crearemos 59 (a no ser que algún valor coincida en dos colisiones y nos tomemos la molestia de no repetirlas, pero dado el número de funciones que tendremos que crear, automatizaremos su generación y no nos complicaremos la vida optimizando el proceso para minimizar espacio final que ocupará nuestro documento PDF)

Cada función debe ir asociada a un estado de gráficos extendido, de forma que podamos aplicarla en el cuerpo del PDF. Para ello tendremos que crear tantos objetos de estado como funciones tengamos (59), usando una construcción como esta:

```
201 0 obj
  << /Type /ExtGState
     /TR 200 0 R    % Referenciando a la función descrita antes
  >>
endobj
```

Teniendo resuelto el tema de las escalas de grises y de las funciones de los estados de gráficos extendidos, solo queda aplicar una función diferente para cada número y estrella. Pero antes, para poder hacer uso de ellas en una página tendremos que declararlas en su correspondiente sección de recursos, asignándole un nombre diferente a cada uno de los objetos. Por ejemplo:

```
50 0 obj
  << /Type /Page
     /Parent 2 0 R
     /Resources <<
       % ...
       /ColorSpace <<
         /HelpCSxx   6 0 R   % Para los títulos y otros textos del documento
         /HelpCS00  10 0 R   % Para los números 0 y 1
         /HelpCS01  11 0 R   % Para los números 2 y 3
         % ...
         /HelpCS29  39 0 R   % Para las estrellas 8 y 9
       >>
       /ExtGState <<
         /HelpGSxx   7 0 R   % Para los títulos y otros textos del documento
         /HelpGS00  41 0 R   % Para el número 0
         /HelpGS01  43 0 R   % Para el número 1
         % ...
         /HelpGS58 159 0 R   % Para la estrella 9
       >>
     >>
  >>
```

Por último, solo falta generar el contenido de la página, aplicando a cada número su correspondiente espacio de color y estado gráfico extendido. De toda la paleta de grises, nos interesa seleccionar un valor que sea diferente en ambas colisiones, por lo que analizando las colisiones que genera el software que hemos usado, observamos que todas ellas difieren como mínimo en el byte de la posición 19 (empezando en 0). Por tanto, para el primer valor seleccionaremos el color del índice 19 y para el segundo valor de la misma paleta usaremos el 147 (128+19).

	Espacio de color	Estado gráfico	Color
Número 1	HelpCS00	HelpGS00	19
Número 2	HelpCS00	HelpGS01	147
Número 3	HelpCS01	HelpGS02	19
Número 4	HelpCS01	HelpGS03	147
…	…	…	…
Estrella 8	HelpCS29	HelpGS57	147
Estrella 9	HelpCS29	HelpGS58	19

Bueno, ahora que ya tenemos todas las piezas sobre la mesa, solo nos falta juntarlas de forma coherente y ya tenemos la solución del reto prácticamente lista.

Es importante es recordar que las colisiones se generan en bloques de 128 bytes y se ubican en posiciones que son múltiplos de 64 bytes. Puesto que el orden de los objetos no es relevante en el PDF, podemos empezar generando las paletas de grises (las colisiones), descartando todas aquellas que contengan cualquiera de los caracteres especiales (recordemos que eran los valores 40, 41 y 92). Esto lo podemos automatizar invocando al programa "*fastcoll*" y analizando los últimos 128 bytes de los dos ficheros de salida que genera. En caso de encontrar algún carácter inválido, descartaremos esa colisión y repetiremos el proceso. En caso contrario, nos guardaremos los dos valores de la colisión y añadiremos a cualquiera de ellos el siguiente contenido del PDF.

`%PDF-1.3`
 	
`% Relleno...`
 	
`10 0 obj [/Indexed /DeviceGray 255`
`(`
	Bloque inicial: encabezado del PDF y apertura de la primera paleta (N x 64 bytes)

`cðÎ□NÅ5□ iŽáç5□ .fm#□ Š{'±¡ÀžN...`	Colisión 1 (128 bytes)
`ÑìÄ^HÔugdàÑ›Ï□ WmeTrÒKKÞÉ¼ŸH ...`	Colisión 2 (128 bytes)

`)] endobj`
 	
`% Relleno...`
 	
`11 0 obj [/Indexed /DeviceGray 255`
`(`
	Cierre de la paleta anterior y apertura de la nueva (64 bytes)

`Êh®□kNê Zã-×dJg‡«"q<h1æK^lt ...`	Colisión 3 (128 bytes)
`Ù]□[Ó%.úv1□ ,¦Ì‡□ □ËÜdsÇËÿS< ...`	Colisión 4 (128 bytes)

`)] endobj`
 	
`% Relleno...`
 	
`12 0 obj [/Indexed /DeviceGray 255`
`(`
	Cierre de la paleta anterior y apertura de la nueva (64 bytes)

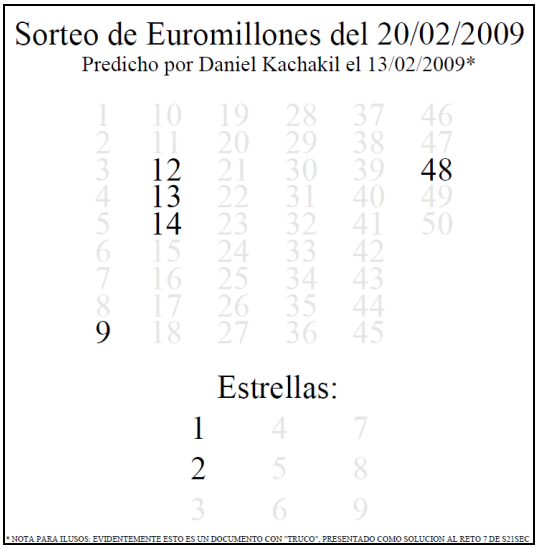
Al final del proceso habremos obtenido un fichero de texto que nos servirá como encabezado del PDF en su estado inicial (almacenando la mitad de las colisiones) y por otro lado tendremos guardados tantos ficheros como colisiones hayamos generado (en nuestro caso 59), que nos servirán para poder sustituir cualquiera de las colisiones generadas por su correspondiente pareja, todo ello sin alterar el hash del fichero.

Llegados a este punto, analizamos todas las parejas de colisiones generadas y obtenemos el primer valor diferente en cada una de ellas (ya vimos que siempre ocurre en la posición 19), de forma que generaremos las funciones de transferencia basándonos en dicho valor y las añadiremos al fichero PDF, junto con sus correspondientes objetos de estados gráficos.

Por último, añadimos todos los objetos que requiera el PDF (catálogo, páginas, sección de referencias cruzadas que tendremos que recalcular, etc). En mi caso opté por una solución bastante sencilla (y algo cutre), ya que mi PDF únicamente contiene un título y todos los números del sorteo, dispuestos de la misma forma que en el boleto real del juego, aunque se podría haber añadido cualquier otro elemento decorativo (imágenes, fuentes, gráficos, etc).

Para evitar que quedara mucho hueco que llamara la atención (en el que se podrían copiar y pegar todos los valores a pesar de no verlos), en vez de optar por un color blanco puro, opté por un gris al 10%, ya que disimulaba mejor la técnica. De esta forma, los números premiados aparecerán en negro, destacando sobre el resto, que aparecerían en un tono muy sutil pero perfectamente visible.

Una vez publicado el resultado del sorteo (20/02/2009), se toma el documento generado como base (en la que todos los números aparecen en gris clarito) y se reemplazan los bloques de colisión correspondientes a los números premiados:



http://www.kachakil.com/retos/Euromillones_20-02-2009.pdf

MD5 = 71dad4b530ff3a78aae6599de477b1a5 (tal y como publiqué en su día)

Agradecimientos y comentarios

Parece que cada día aparecen nuevas técnicas que ya permiten hacer prácticamente lo que nos venga en gana con el algoritmo MD5, así que el reto podía parecer trivial y algunos pensarán (con parte de razón) que esto no es nada nuevo. Sin embargo, en este reto no hemos hecho uso de las últimas técnicas descubiertas (de tipo *chosen-prefix*), que son mucho más potentes, a la vez que requieren una capacidad computacional muchísimo más elevada y de las que tampoco han sido publicadas sus herramientas. Recordemos que las técnicas que hemos utilizado fueron publicadas en 2005 y no tienen prácticamente nada que ver con éstas últimas.

A pesar de todo, sabemos que el algoritmo MD5 se considera ya obsoleto desde hace unos años y se recomienda sustituir su uso por alguno de los nuevos algoritmos con propiedades equivalentes (ej: SHA1), pero de momento más seguros. Esperemos que este reto haya puesto de manifiesto que este tipo de técnicas están al alcance de cualquiera y que es mejor pensárselo dos veces antes de confiar en el algoritmo MD5 para según qué cosas...

Como siempre, me gustaría terminar este documento dando las gracias a Mikel Gastesi por este reto y a todo el que haya colaborado en el mismo.

Saludos,

Daniel Kachakil
dani@kachakil.com