



MÁSTER DE INGENIERÍA TELEMÁTICA
Seguridad y Protección de la Información
HASH, GENERADOR DE CARTAS Y
ATAQUE DE CUMPLEAÑOS

JESÚS VIDAL PANALÉS

Índice de contenido

1.REQUISITOS DEL SISTEMA Y LISTADO DE ARCHIVOS.....	3
2.ARCHIVOS DE LA APLICACIÓN Y SU FUNCIONAMIENTO.....	3
2.1.Visión general.....	3
2.2.SDES.java.....	4
2.3.Hash.java.....	4
2.4.HashGrafico.java.....	5
2.5.HashUI.java.....	7
2.6.Generador.java.....	8
2.7.SeleccionarRutas.java.....	9
2.8.AtaqueHash.java.....	9
2.9.AbrirMensajes.java.....	10
3.FUNCIONAMIENTO DE LA APLICACIÓN.....	10

1.REQUISITOS DEL SISTEMA Y LISTADO DE ARCHIVOS

Para poder ejecutar la aplicación es necesario instalar la última versión de java disponible (JRE 6.0), ya que el programa se ha creado en el entorno de desarrollo JDK 6.

El contenido es el siguiente:

1. La carpeta **src** con el código fuente (.java) y archivos .class. También se encuentra en su interior la carpeta **images** con las imágenes usadas y la carpeta **javadoc** con la explicación de cada clase y sus métodos.
2. El archivo **hash.jar** que contiene todo lo necesario para ejecutar el programa.
3. La carpeta **mensajes** que contiene dos carpetas: **grupo1** y **grupo2** que contienen cada una de ellas, una carpeta de mensajes originales y de mensajes falsos. Se puede realizar una prueba del ataque con cualquiera de ellos, eligiendo los mensajes originales del grupo1 y los falsos del grupo1 o los originales del grupo2 y falsos del grupo2, o mezclando originales y falsos de cada grupo.

Para abrir el programa, ejecutar el archivo **hash.jar**.

2.ARCHIVOS DE LA APLICACIÓN Y SU FUNCIONAMIENTO

2.1.Visión general

Para la creación del programa se han creado las siguientes clases principales:

1. **SDES.java**: Este archivo realiza el cifrado DES Simplificado, y se utiliza en la función de compresión del proceso Hash.
2. **Hash.java**: Este archivo realiza todo el procedimiento del Hash. Primero realiza una función preliminar para procesar el mensaje original y posteriormente, con el mensaje resultante, se realiza el Hash, ayudado de la función de compresión.
3. **HashGrafico.java**: Igual que la clase anterior, salvo que muestra el proceso Hash en una ventana.
4. **HashUI.java**: Aplicación principal del programa. Es la clase principal, que ejecuta el programa en una ventana. Se detallará posteriormente.
5. **Generador.java**: Es la clase que realiza el proceso de generador de mensajes o cartas para poder realizar posteriormente el ataque de cumpleaños.
6. **SeleccionarRutas.java**: Clase que se encarga de mostrar al usuario una ventana para que seleccione la ruta donde se encuentran los mensajes originales y la ruta de los mensajes falsos para comenzar el ataque hash.
7. **AtaqueHash.java**: Clase encargada de realizar el ataque hash y mostrar el proceso en una ventana. Es posible ver posteriormente el mensaje original seleccionado y uno de los mensajes falsos asociado al original, es decir, poder ver el mensaje original y un mensaje falso que tiene el mismo hash.
8. **AbrirMensajes.java**: Clase que se utiliza para abrir el mensaje original seleccionado en el resultado del ataque hash, y el mensaje falso seleccionado asociado al mensaje original.

Ahora veamos con detalle cada clase.

2.2.SDES.java

Esta clase es una versión simplificada de la clase usada en el programa anterior SDES. La diferencia con el anterior, es que ahora sólo procesa mensajes de 12 bits, sin tener que convertir de texto a binario, ni de trocear el mensaje en bloques. También se ha eliminado el proceso de descifrado, debido a que para el hash no es necesario.

Como el funcionamiento del cifrado se explicó en el anterior programa, pasemos a la siguiente clase.

2.3.Hash.java

Esta clase es la encargada de realizar el proceso hash. Primero, recibe un mensaje de texto, lo convierte a binario, y realiza unos pasos preliminares para dejar el mensaje preparado para realizar la función Hash.

La clase consta de las siguientes variables:

- a) **private SDES sdes:** Es el objeto encargado de realizar el cifrado SDES para la función de compresión.
- b) **private String mensaje:** Variable donde se almacena el mensaje original en binario.
- c) **private String preliminar:** Variable donde se almacena el procesado del mensaje original binario, para posteriormente procesarlo en la función hash.
- d) **private int rondas:** Las rondas utilizadas en la función SDES.
- e) **private String hash:** Variable que almacena el resultado final de la función hash.

Ahora veamos el constructor y los métodos implementados en la clase:

- a) **public Hash (String mensaje, int r):** El constructor de la clase. Recibe un mensaje de texto y las rondas para el cifrado SDES utilizado en la función de compresión. El mensaje de texto lo convierte a binario con la función *texto_a_binario*, y almacena el mensaje binario en la variable *mensaje*. El valor de las rondas se almacena en la variable *rondas*.
- b) **private String compresor (String cadena):** Función de compresión. Comprime 21 bits en 12 bits. Recibe un mensaje de 21 bits, extrae los 12 primeros bits que será el mensaje para el cifrado SDES y los 9 bits restantes, será la clave para el SDES. Además, se le pasa a SDES las rondas. El método devuelve el mensaje comprimido de 12 bits.
- c) **private String funcion_preliminar (String mensaje):** Método que realiza la función preliminar para poder procesar el mensaje en la función Hash. Su funcionamiento es el siguiente:
 - Primero realiza la función módulo del mensaje con el valor de **r** (9), y se obtiene el resto. Con este dato, añadimos tantos ceros a la izquierda del mensaje para que sea divisible por **r**.
 - Lo siguiente, es añadirle **r** ceros a la derecha (2).
 - El siguiente paso es convertir la longitud del mensaje original, en binario.

- Una vez que obtenemos la longitud en binario del mensaje original, le hacemos el módulo de $(r-1)$, y con este resultado sabremos cuántos ceros hay que añadir a la izquierda para que sea divisible por $(r-1)$.
- Con el resultado anterior, añadimos un **1** cada $r-1$ bits (3).
- Y finalmente, concatenamos el resultado (2) con el resultado (3). Ahora podremos realizar el proceso hash.

d) public String funcion_hash(): Este método es el encargado de realizar la función hash. Primero llama a la *función_preliminar* con el mensaje binario original, para poder realizar el proceso hash. Para obtener el hash, se hace de forma iterativa. Primero obtendremos **H0** cuyo valor serán 12 bits a cero. A partir de H0, realizamos el siguiente proceso:

- Concatenamos **H(i-1)** con **Xi** (Xi son 9 bits del mensaje resultante de la función preliminar).
- Con el valor obtenido anteriormente, llamamos a la función de compresión, y obtenemos 12 bits que será el valor de **Hi**.

El último **Hi** resultante será el valor Hash del mensaje. En el proceso descrito para realizar el hash, tendremos un total de rondas que será el valor de la longitud del mensaje resultante de la función preliminar partido por **r**. De ese mensaje vamos extrayendo bloques de 9 bits que se concatenan con el valor del H de la ronda anterior. En la primera ronda, será **H0** concatenado con los primeros 9 bits del mensaje, en la siguiente ronda, será **H1** concatenado con los 9 siguientes bits del mensaje, y así hasta alcanzar la última ronda.

e) private String texto_a_binario (String m): Esta función convierte un mensaje de texto en un mensaje binario. Es la misma función que se programó en el SDES. Por cada carácter, lo convierte a binario, y si su longitud binaria es <8 añade los ceros a la izquierda necesarios para que su longitud sea de 8 bits.

2.4.HashGrafico.java

Esta clase realiza exactamente lo mismo que la clase anterior, pero se diferencia en que es un *Thread* (un hilo) y que tiene una clase interna, que es también un hilo, que ejecuta una ventana, donde se va mostrando el proceso Hash. Por tanto, la clase principal interactúa con la clase interna para ir mostrando el resultado de cada paso. La clase interna se llama **ProcesoHash**.

Veamos el funcionamiento global, omitiendo el proceso Hash, ya que se describió anteriormente. Primero, las variables:

a) private SDES sdes: Lo mismo que en la clase Hash.

b) private String mensaje: Lo mismo que en la clase Hash.

c) private String preliminar: Lo mismo que en la clase Hash.

d) private int rondas: Lo mismo que en la clase Hash.

e) private String hash: Lo mismo que en la clase Hash.

f) private JTextPane areaProceso: Este será el panel de texto donde se

mostrará el proceso Hash. El proceso irá añadiendo resultados a este componente, que estará a su vez dentro de la ventana creada en la clase interna **ProcesoHash**.

g) private SimpleAttributeSet attrs: Sirve para definir los atributos del texto que se añade en *areaProceso*: si es negrita, el color, etc.

h) private String cadena = "": Aquí se irá almacenando cada resultado que se volcará en *areaProceso*.

Ahora veamos el constructor, y los métodos de la clase principal:

a) public HashGrafico (String mensaje, int r): Hace lo mismo que en la clase Hash, pero además inicia el hilo de ejecución de la clase interna *ProcesoHash*, creamos la variable de atributos para el texto, e insertamos el texto del mensaje original, mensaje binario inicial.

b) public void run(): Método que inicia el hilo. Llama a la *función_hash*.

c) private String compresor (String cadena): Lo mismo que en la clase Hash.

d) private String funcion_preliminar (String mensaje): Lo mismo que en la clase Hash, pero además va añadiendo el resultado de cada paso. Para que no se muestre todo de golpe, se ha añadido la función *sleep*, con el que el proceso se duerme durante 60 ms. Antes de empezar el proceso, se duerme 500 ms.

e) public String funcion_hash (): Lo mismo que la clase Hash pero además va mostrando la salida de cada paso en el panel de texto de la ventana. También se duerme durante 60ms en cada paso como en la función descrita anteriormente.

f) private String texto_a_binario (String m): Lo mismo que en la clase Hash.

Ahora pasemos a ver la clase interna **ProcesoHash**. Esta clase es un Thread (un hilo) que crea una ventana en la que se mostrará el proceso de hash. Consta de:

a) JFrame procesoHash: Es la ventana donde se insertan los componentes.

b) JLabel etiquetaProceso: Es una etiqueta para mostrar indicar simplemente que es el proceso Hash.

c) JScrollPane scrollProceso: Es un contenedor scroll, es decir, el elemento que introduzcamos en su interior, tendrá la capacidad de poder expandir su contenido y mostrará una barra de desplazamiento para moverse dentro de la misma. En nuestro caso, es donde insertaremos el *areaProceso*, que es un JTextPane.

d) Imagen botonCerrarNormal/botonCerrarPulsado: Son las imágenes que tendrá el botón *Cerrar* de la ventana.

e) Dimension boton: Es la dimensión que tendrá el botón. Usaremos las

dimensiones de la imagen del botón creado.

- f) **JButton botonCerrar:** Es el botón Cerrar que tendrá las imágenes creadas anteriormente según se pulse el botón o no. Tendrá un ActionListener, que se utilizará para cerrar la ventana y liberarla de la memoria.
- g) **JPanel panel:** Contendrá scrollProceso y botonCerrar.

El constructor de esta clase estará vacío, y el hilo se iniciará con la llamada al método:

a) **public void run()**

Además, contiene una clase ActionListener para ejecutar la acción del botón *Cerrar* de la aplicación. Simplemente, cerrará y liberará la ventana de la memoria. Esta clase se llama *Cerrar*.

2.5.HashUI.java

Esta clase es la aplicación principal. Esta aplicación es una ventana que contiene un área para escribir un mensaje de texto, que a su vez, se utiliza en el caso de que se abra un documento de texto, volcar el texto del archivo, y también será necesario para el generado de mensajes. Existe otra sección, que se utiliza para el generador de mensajes, donde se introducen las palabras alternativas al mensaje escrito o abierto. No es necesario escribir por cada palabra una palabra alternativa, se pueden dejar en blanco las palabras de las que no se quiera tener una palabra alternativa. Está pensado para generar mensajes originales y mensajes falsos, es decir, primero escribir un mensaje válido, y utilizar palabras sustitutas que tengan el mismo significado, como si fueran versiones del mensaje, y después, una vez generados los mensajes originales, escribir un mensaje falso, extraerlo, e introducir palabras sustitutas por cada palabra que se quiera tener una alternativa.

De forma esquemática, se va a explicar el contenido de las variables de la clase, simplemente qué se observa en la ventana sin entrar en detalles de qué es cada elemento. En la ventana, se puede observar 4 áreas: un área donde escribir un mensaje de texto, a su lado otro utilizado para el generador de mensajes. Debajo de éstos, hay un panel donde se eligen las rondas del SDES y además, se puede activar o desactivar que se muestre el proceso de Hash. Y por último, existen una serie de botones: **Abrir**, para abrir un mensaje de texto; **Hash**, para realizar el hash del mensaje; **Extraer**, extrae las palabras contenidas en el área de texto y las muestra individualmente en el área del generador, aquí será donde se añadan las palabras alternativas; **Generar**, que se encarga de llamar a la clase *Generador* para generar los mensajes; **Atacar**, que abrirá una ventana para seleccionar las rutas donde se encuentran los mensajes originales y falsos, y se llamará a la clase *AtaqueHash*; y por último, **Salir**, que es el botón para salir del programa.

Esta clase contiene además unas variables necesarias para, saber si mostrar o no el proceso hash (boolean procesoHash), una variable para almacenar el texto del área de texto (String mensaje), un objeto Hash, para realizar el hash, las rondas del cifrado SDES para el proceso hash, un vector que contiene las palabras para el generador de mensajes y un array de String, que almacenará

las palabras del mensaje de texto.

Existen clases internas para cada evento:

- a) **private class Salir implements ActionListener:** ActionListener para salir del programa.
- b) **private class Seleccionado implements FocusListener:** FocusListener para seleccionar el texto del área de texto.
- c) **private class Rondas implements KeyListener:** Para introducir el número de rondas del SDES.
- d) **public boolean comprobarRondas():** Para comprobar que hay un número de rondas y es mayor que cero.
- e) **private class HashMensaje implements ActionListener:** Para realizar el proceso Hash, mostrando o no el proceso según se active la casilla correspondiente.
- f) **private class Abrir implements ActionListener:** Para abrir un mensaje de texto y volcarlo al área de texto.
- g) **private class Add implements ActionListener:** Para añadir la palabra alternativa a una palabra original. Se almacenará en un array JComboBox.
- h) **private void actualizarPanel():** Para actualizar el panel del generador de mensajes. Si has cambiado tu mensaje del área de texto, y extraes, primero borra el contenido anterior y después se añade lo que corresponde al nuevo mensaje.
- i) **private class Extraer implements ActionListener:** Se encarga de extraer las palabras del área de texto y guardarlas para el generador. Añade tantos elementos necesarios (JTextField, JComboBox) por cada palabra del mensaje.
- j) **private class Generar implements ActionListener:** Se encarga de mostrar una ventana donde elegir la carpeta donde almacenarán los mensajes que se van a generar, y llama a la clase *Generador*.
- k) **private class Atacar implements ActionListener:** Se encarga de llamar a la clase *SeleccionarRutas*, para elegir las rutas de los mensajes originales y falsos, y posteriormente esa clase llamará a la clase *AtaqueHash*.
- l) **private class Mostrar implements ItemListener:** Se encarga de actualizar la variable utilizada para saber si se llamará a la clase *Hash* o a la clase *HashGrafico*. Si es ésta última, ocultará el lugar donde es muestra el Hash.
- m) **public static void main(String[] args):** Se encarga de ejecutar la aplicación.

2.6. *Generador.java*

Esta clase es un Thread y contiene un Thread (*MostrarGenerador*) para mostrar el proceso del generador de mensajes, que es también un Thread que contendrá un JFrame. Como se explicó anteriormente algunas clases comunes, como Cerrar, o el funcionamiento del método run(), se explicará el funcionamiento del generador.

- a) **private void generarMensajes():** Este es el método que se encarga de generar los mensajes. Primero prepara el formato de archivo, y se almacena el mensaje principal. Posteriormente, se recorre el vector donde almacena en cada posición las palabras alternativas. Si una posición del vector no contiene nada, significa que no hay palabras alternativas para la posición de la palabra del mensaje original. Por tanto, es un vector de tamaño el número de palabras del mensaje original, y que contendrá un vector por cada

posición con las palabras alternativas. Si una posición no contiene un vector de tamaño mayor que cero, es porque no hay palabras para generar en esa posición.

Por tanto, se recorre el Vector principal, y por cada posición que contenga un vector con elementos, se llama a la función *generar*, al que se le pasan el Vector contenido en cada posición del Vector contenedor, que tendrá las palabras alternativas, los mensajes generados anteriormente, que en el primer caso, será el mensaje original, y en las rondas posteriores, el mensaje original y los mensajes generados en la ronda anterior, y el índice de la posición donde se encuentra la palabra original, por la que se va a sustituir por cada palabra alternativa que genera un nuevo mensaje.

Además, en cada ronda, se guarda el mensaje generado, que primero hay que procesar, ya que es un Vector que contiene palabras, y habrá que formar el mensaje de texto extrayendo las palabras. También se muestra el nombre del mensaje generado y el contenido del mensaje en el área de texto de la ventana.

b) private Vector<Vector> generar (Vector<Vector> mensajes, Vector<String> palabras, int indice): Esta función crea mensajes, dadas las palabras a sustituir, los mensajes generados anteriormente, o en el primer caso, el mensaje original, y el índice del lugar donde se encuentra la palabra original. Una vez hecho, devuelve un Vector que contiene vectores de los mensajes generados.

2.7. *SeleccionarRutas.java*

Esta clase se utiliza para seleccionar las rutas de los mensajes originales y falsos y posteriormente llamar a la clase *AtaqueHash*, que realiza el ataque de cumpleaños. Lo más indicativo que se pueda reseñar de esta clase, aparte de la explicación anterior, es que contiene dos botones de examinar, uno para la ruta de mensajes originales, y otro para la ruta de mensajes falsos, y un botón de aceptar, para comenzar el ataque, y uno de cancelar, para cerrar la ventana sin realizar el ataque.

2.8. *AtaqueHash.java*

Esta clase se encarga de realizar el ataque y mostrarlo en una ventana. Es un Thread que contiene otro Thread con un JFrame utilizado para mostrar el proceso individual y global del ataque. El funcionamiento es el siguiente:

a) private boolean atacar(): Este método realiza lo siguiente: abre los mensajes tanto originales como falsos y genera el hash de todos, guardando el resultado por cada mensaje en una Hashtable, y los nombres de los mensajes en un Vector. Una vez obtenidos, mediante un bucle for, por cada hash de mensaje original, se busca un hash de un mensaje falso que coincida con el original, en cada coincidencia se produce una pequeña pausa para ver el mensaje que rompe el hash. Los nombres de los mensajes que rompen el hash, se almacenan junto al nombre original, en un Vector, para posteriormente abrir los mensajes resultantes. Este método, devolverá true si encuentra resultados satisfactorios, y false en caso contrario.

Además, muestra el proceso de los resultados de las coincidencias, en caso de tener mensajes falsos con mismo hash que el mensaje original.

Una vez realizado el proceso, en caso de éxito se tendrá la opción de abrir un mensaje original junto con su mensaje falso para leerlos. Esto se hará llamando a la clase *AbrirMensajes*.

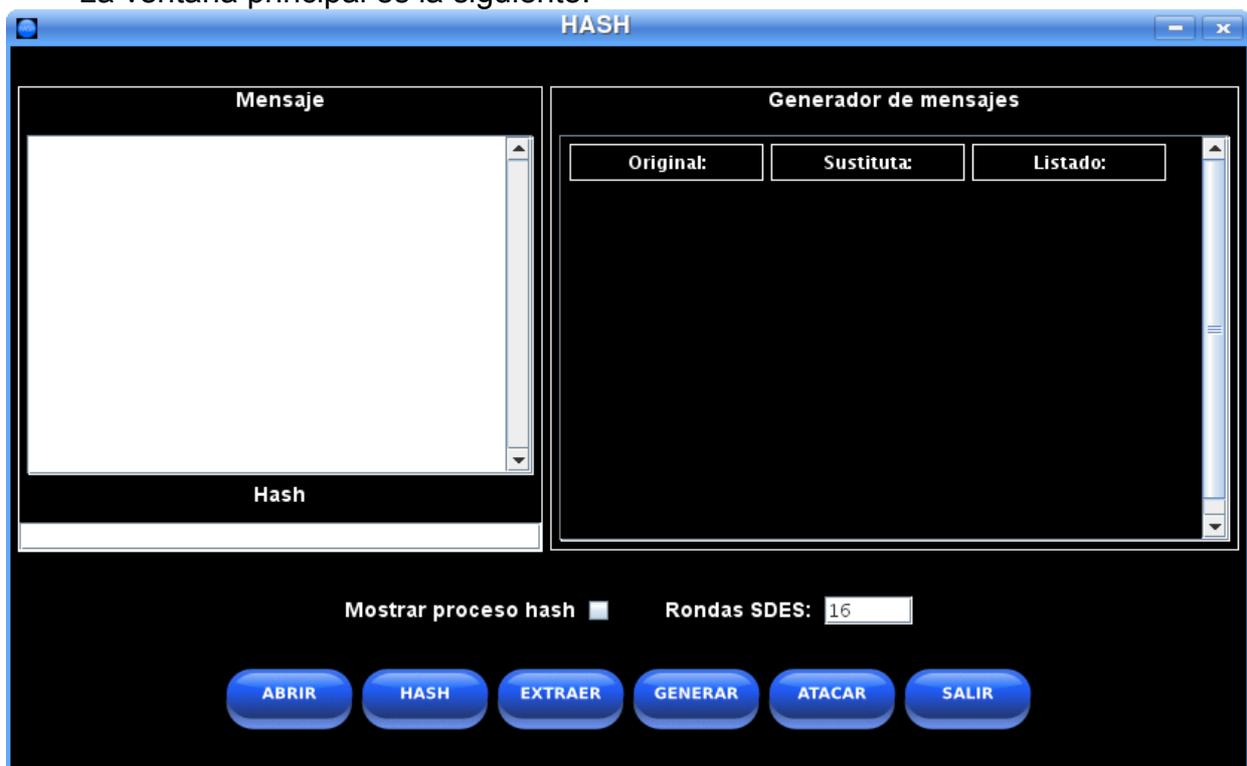
2.9. *AbrirMensajes.java*

Esta clase es utilizada para visualizar un mensaje original y el falso asociado al mensaje original. En la clase anterior, cuando se realiza el ataque, al finalizar, se crea un JComboBox que contiene los mensajes originales que durante el ataque se ha encontrado un hash igual en otro mensaje, y otro JComboBox asociado al primero, donde por cada mensaje original seleccionado en el anterior JComboBox, contendrá los mensajes falsos que tienen el mismo hash. Una vez seleccionado, el botón abrir es quien llama a esta clase para mostrar el mensaje original y el mensaje falso asociado.

3. FUNCIONAMIENTO DE LA APLICACIÓN

Ahora se explicará junto con capturas, el funcionamiento de la aplicación.

La ventana principal es la siguiente:



Como se puede observar, existe una zona para escribir un mensaje de texto, otra zona que se empleará para generar los mensajes, otra zona para elegir si ver o no el proceso hash y elegir las rondas del cifrado SDES, y por último los botones.

Para realizar el hash, se puede o bien escribir un mensaje de texto y pulsar el botón Hash, o bien abrir un archivo con el botón Abrir, seleccionar un mensaje de texto y una vez abierto pulsar el botón Hash. Además, si se activa la casilla Mostrar

proceso hash, para mostrar el proceso hash o nada más que obtener el hash del mensaje, que se mostrará en el área de texto justo debajo del mensaje, en la sección Hash.

La siguiente captura muestra un resultado de Hash sin mostrar el proceso:



Podemos observar el resultado del hash del mensaje escrito. Las siguientes capturas muestran el proceso hash, tras marcar la casilla de *Mostrar proceso hash*:





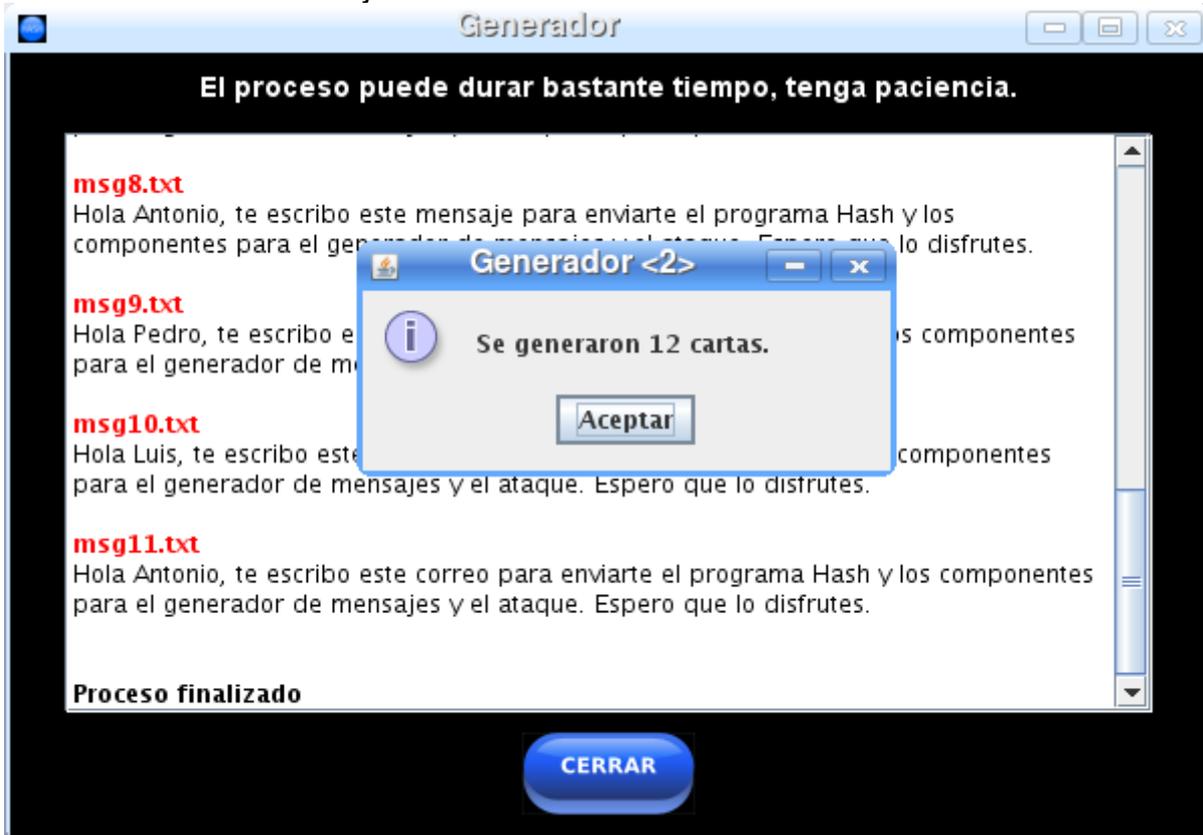
Como vemos, al final del proceso, se muestra el hash resultante.

La siguiente captura muestra un mensaje de texto extraído pulsando el botón *Extraer* para generar mensajes. En este caso, se utiliza para cambiar palabras a otras “falsas”.

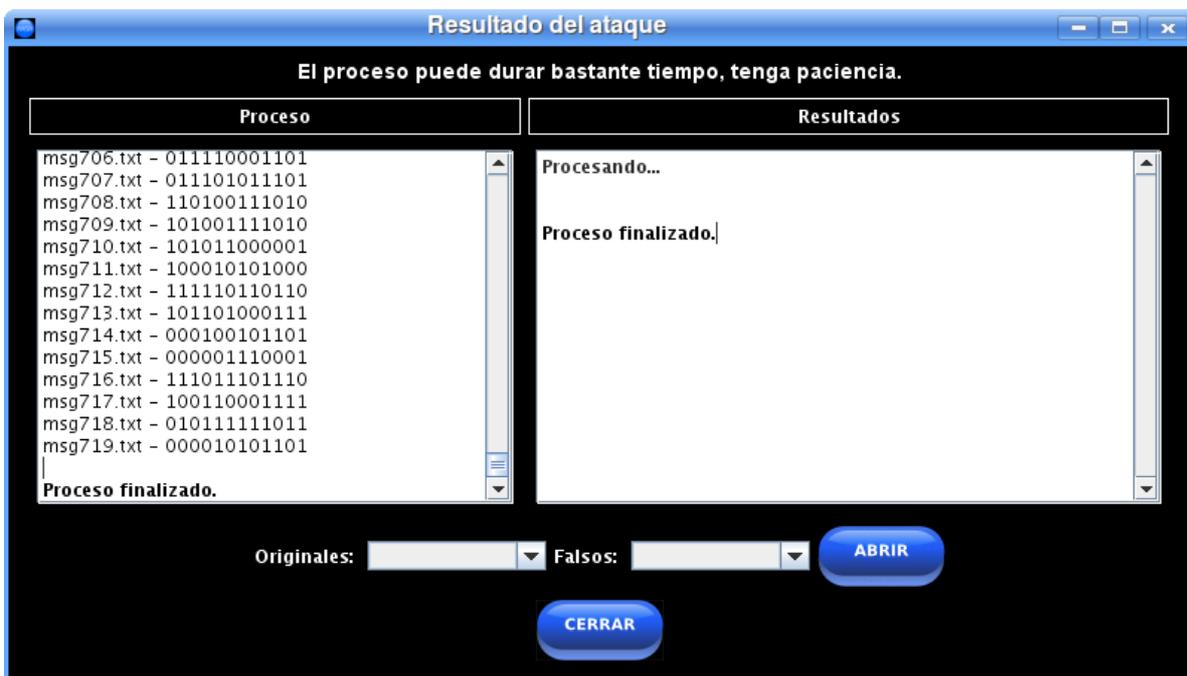


En la captura anterior, para añadir una palabra a la sección *Listado*, hay que escribirla en la sección *Sustituta*, y pulsar el botón cuadrado de color azul, a la derecha de cada combobox. Como se puede observar, no se añadieron palabras sustitutas por cada palabra original, tan solo de algunas.

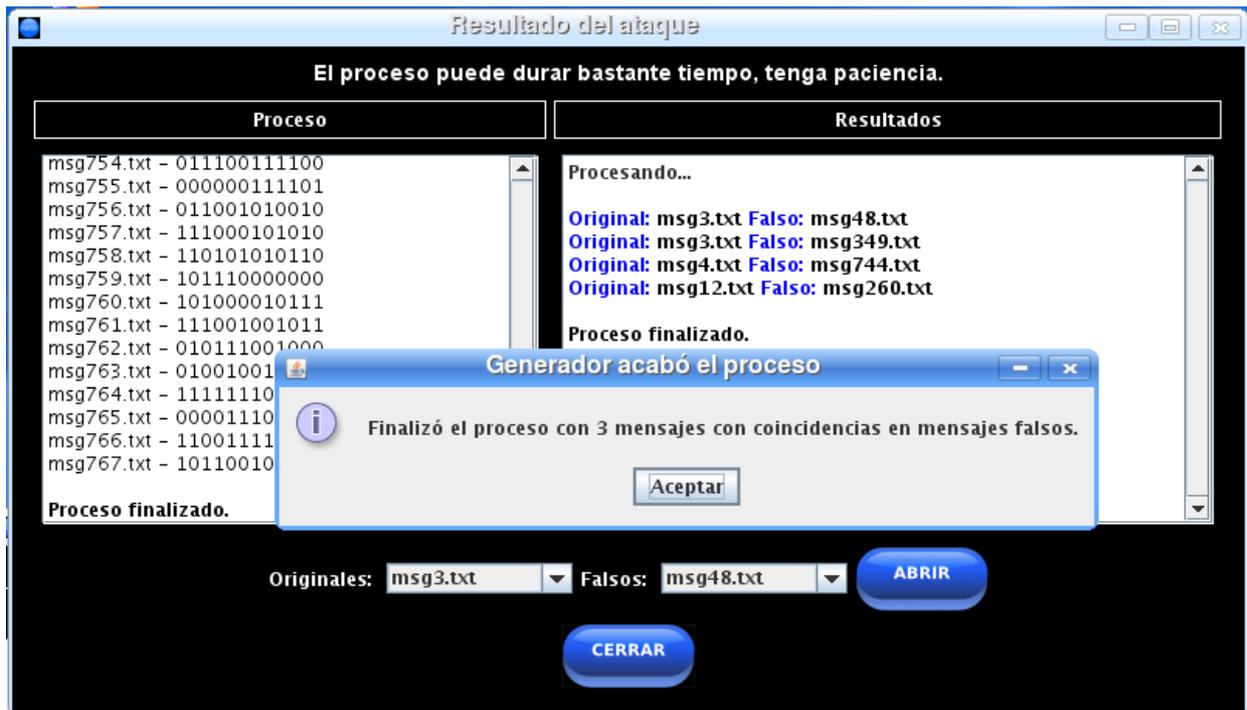
La siguiente captura muestra el resultado de la generación de mensajes, al pulsar el botón *Generar*. Cuando se pulsa este botón, se pedirá la ruta donde se quieren almacenar los mensajes.



La siguiente captura muestra un ataque sin obtener resultados:



Y por último una captura con resultados exitosos en el ataque:



En la captura anterior se muestra el resultado final, con las cartas originales y las falsas con mismo hash. No obstante, durante el proceso del ataque, si un mensaje falso tiene el mismo hash que el original se marca de color azul y se pausa el proceso durante un segundo. La siguiente captura muestra un mensaje original y un mensaje falso con el mismo hash, tras seleccionar el mensaje original y falso en sus respectivos combobox y pulsando el botón *Abrir*.

