UNIVERSIDAD AUTÓNOMA METROPOLITANA

DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

INGENIERÍA EN COMPUTACIÓN

REPORTE FINAL DE PROYECTO TERMINAL

Por: Marco Edgar Valencia Arana

Matrícula: 205206409

Asesor: Dr. Francisco Javier Zaragoza Martínez

Trimestres: 10P-11I

Esteganografía de archivos usando redundancia cíclica en imágenes JPEG

Contenido

INTRODUCCIÓN

El presente documento describe las etapas de desarrollo del proyecto terminal "Esteganografía de archivos usando redundancia cíclica en imágenes JPEG" así como la descripción de las diferentes pruebas realizadas y los resultados obtenidos.

También se describe el funcionamiento del algoritmo RSA el cual fue elegido idóneo para su implementación en este proyecto.

Es así como se irá describiendo desde cómo están estructurados los diferentes archivos de imagen usados, que técnicas esteganográficas se usaron para la inserción de los archivos en las imágenes, el diseño y desarrollo del sistema.

Además se presentará el manual de usuario en el cual se describirán los diferentes pasos a seguir para poder usar el sistema, desde la instalación hasta la implementación del mismo.

Anexo a este documento también se presenta el archivo de texto con el cual se realizaron las pruebas, así como las imágenes en las cuales se hicieron las inserciones del archivo de texto.

OBJETIVO GENERAL

Implementar un algoritmo de comprobación de redundancia cíclica (CRC) para codificar un archivo de texto en una imagen de tipo JPEG mediante la implementación de un programa en lenguaje Java.

OBJETIVOS PARTICULARES

- Determinar los algoritmos para codificar y decodificar el archivo de texto en un flujo de bytes.
- Implementar los algoritmos para la codificación y decodificación.
- Implementar el algoritmo de CRC.
- Realizar un módulo de detección de errores.
- Evaluar el funcionamiento del programa final.

ANÁLISIS

Formato de mapa de bits

Una imagen matricial, también llamada mapa de bits, es una estructura de datos que representa una rejilla rectangular de pixeles o puntos de color.

A las imágenes tipo mapa de bits se las suele caracterizar por su altura y anchura (en pixeles) y por su profundidad de color (en bits por pixel), que determina el número de colores distintos que se pueden almacenar en cada pixel, y por lo tanto, en gran medida, la calidad del color de la imagen.

Estructura del archivo

Para nuestro caso utilizaremos una imagen tipo mapa de bits con una profundidad de color de 24 bits ya que esto nos permitirá ocultar nuestra información dentro de ella. Ya que las imágenes con este formato utilizan un modelo de color llamado RGB (del inglés *Red, Green, Blue*; "rojo, verde, azul") el cual hace referencia a la composición del color en términos de la intensidad de los colores primarios con que se forma: el rojo, el verde y el azul. Es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores primarios. Es así como nuestra imagen de 24 bits utiliza 8 bits para representar una intensidad de rojo, 8 bits para una intensidad de verde y 8 bits para una intensidad azul.

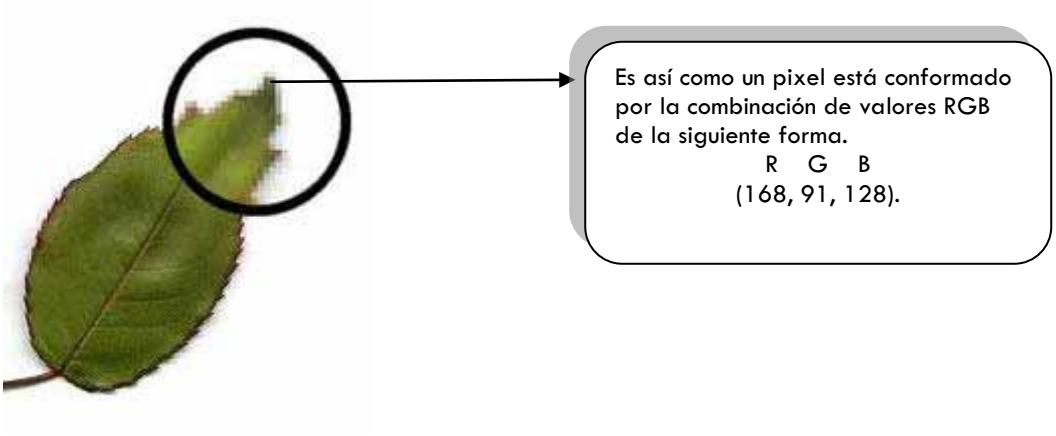


FIGURA 1. REPRESENTACION DE MAPA DE BITS 1

Formato JPEG.

JPEG (Joint Photographic Experts Group, Grupo Conjunto de Expertos en Fotografía), es el nombre de un comité de expertos que creó un estándar de compresión y codificación de archivos de imágenes fijas.

Además de ser un método de compresión, es a menudo considerado como un formato de archivo. JPEG/Exif es un formato muy usado junto con JPEG/JFIF, que también es otro formato para el almacenamiento y la transmisión de imágenes fotográficas en Internet. Estas variaciones de formatos a menudo no se distinguen, y se llaman JPEG. Los archivos de este tipo se suelen nombrar con la extensión .jpg.

Compresión

El formato JPEG utiliza habitualmente un algoritmo de compresión con pérdida para reducir el tamaño de los archivos de imágenes. Esto significa que al descomprimir o visualizar la imagen no se obtiene exactamente la misma imagen de la que se partía antes de la compresión. Existen también tres variantes del estándar JPEG que comprimen la imagen sin pérdida de datos: JPEG2000, JPEG-LS y Lossless JPEG.

ESTEGANOGRAFÍA

Las técnicas utilizadas en la realización de este proyecto para el ocultamiento del mensaje en la estructura del archivo de tipo JPEG fueron dos básicamente, se utilizo el algoritmo RSA con el que codificamos nuestro mensaje de texto en un flujo de bits.

El algoritmo consta de tres pasos: generación de claves, cifrado y descifrado.

La idea del algoritmo es la generar dos tipos de claves una pública y una privada, esto con la finalidad de que cualquier usuario pueda cifrar usando la clave pública, pero solo aquellos que conozcan la clave privada podrán descifrar correctamente

Generación de claves

- Cada usuario elige dos números primos distintos ***p*** y ***q***.
- Se calcula ***n* = *p*·*q*.**
- Se calcula φ**(n)=(p-1)(q-1).**
- Se escoge un entero positivo e menor que φ **(n),** que sea coprimo con φ **(n).**
- Se determina un ***d*** (mediante aritmética modular) que satisfaga la congruencia **d·e= 1(mod** φ **(n).**

La **clave pública** es (*n*,*e*), esto es, el módulo y el exponente de cifrado. La **clave privada** es (*n*,*d*), esto es, el módulo y el exponente de descifrado, que debe mantenerse en secreto.

Cifrado

Suponiendo que el emisor comunica su clave pública(n,e) al receptor y guarda la clave privada en secreto. Ahora el emisor quiere enviar un mensaje M a el receptor

Luego se calcularía el texto cifrado C mediante la operación:

$$C=M^e \pmod n.$$

Ahora el emisor transmite el mensaje C al receptor.

Descifrado

Ahora el receptor puede recuperar el mensaje original M a partir de C usando su exponente *d* de la clave privada mediante el siguiente cálculo:

$$M=C^d \pmod n.$$

Y así el receptor recuperaría el mensaje original con su clave privada.

Ocultamiento del archivo en la imagen

Una vez comprendido esto basta con conocer la estructura de un archivo mapa de bits el cual ya se describió con anterioridad, es así como aplicamos el segundo algoritmo basándonos en la estructura de dicho archivo, como ya sabemos una imagen está compuesta por una matriz de pixeles los cuales a su vez tienen un modelo RGB para cada pixel, para la aplicación del segundo algoritmo únicamente se hizo una conversión binaria de los valores representativos para cada color rojo, verde y azul y se tomó el tercer bit menos significativo para poder guardar los valores que representa nuestro archivo de texto, esto con la finalidad de que el cambio entre la imagen original y la modificada sea lo más pequeña posible, los detalles de la implementación de este algoritmo se describirán más adelante en el apartado "algoritmos de ocultamiento".

ALGORITMOS DE OCULTAMIENTO

Encriptamiento del texto

En este apartado describiremos los pasos de las implementaciones de los algoritmos antes descritos, así mismo para aplicar el algoritmo RSA se siguieron los siguientes pasos.

- Se generan las llaves pública y privada las cuales serán usadas para el encriptamiento y desencriptamiento sucesivamente
- Una vez creadas estas llaves se procede a abrir únicamente la llave pública (n, e) para poder comenzar el encriptamiento del texto.
- Se procede a abrir y leer el archivo de texto plano.
- El archivo de texto será leído por líneas de las cuales cada letra será convertida a su equivalente en el código ASCII y será aplicada la fórmula antes descrita con nuestra correspondiente llave pública, para ilustrar un poco más este procedimiento, se lleva a cabo el siguiente ejemplo.

Si nuestro archivo de texto plano contiene la siguiente oración **"TEXTO A ENCRIPTAR"** el proceso de encriptación será llevado a cabo de la siguiente manera.

Cada letra será llevada a su representación en código ASCII

T	E	X	T	O		A		E	N	C	R	I	P	T	A	R
84	101	120	116	111	32	97	32	101	110	99	114	105	112	116	97	114

Una vez hecho esto, a cada letra se le aplicará la fórmula de encriptamiento con la correspondiente llave pública esto se guardará en un arreglo, el cual será escrito en un nuevo archivo de texto con las claves de cada letra.

Inserción en la imagen

Los pasos a seguir para la inserción del texto en la imagen fueron:

- Se procederá a abrir la imagen verificando que sea del tipo mapa de bits.
- Se aplicara CRC-32 (código de redundancia cíclica) a cada elemento del arreglo donde se encuentra nuestro texto encriptado, esto con la finalidad de verificar algún error tras la compresión de la imagen después de la inserción del texto.

Ahora insertaremos el texto con el proceso que describimos a continuación:

Suponiendo que nuestro arreglo con los valores después del encriptado con su equivalente en binario son:

Valor decimal de cada letra	Representación Binaria
10313	10100001001001
42342	1010010101100110
23275	101101011101011
11115	10101101101011
46758	1011011010100110
20182	100111011010110
34105	1000010100111001
20182	100111011010110
42342	1010010101100110
17724	100010100111100
9145	10001110111001
35451	1000101001111011
15323	11101111011011
30992	111100100010000
11115	10101101101011
34105	1000010100111001
35451	1000101001111011

TABLA 1. REPRESENTACION DE LOS VALORES 1

Se obtiene los valores de toda la matriz de pixeles, de manera que guardamos en un objeto de tipo "pixel" los valores de cada pixel en RGB por ejemplo si para los primeros 10 pixeles los valores fueron.

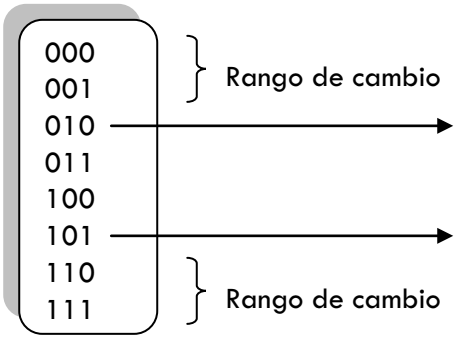
	ROJO	VERDE	AZUL
Pixel 1	10010111	00111010	10010111
Pixel 2	00111010	10010111	00111010
Pixel 3	10010111	00111010	10010111
Pixel 4	00111010	10010111	00111010

Pixel 5	10010111	00111010	10010111
Pixel 6	00111010	10010111	00111010
Pixel 7	10010111	10010111	10010111
Pixel 8	00111010	10010111	00111010
Pixel 9	10010111	00111010	10010111
Pixel 10	00111010	10010111	00111010

TABLA 2. VALOR POR PIXEL 1

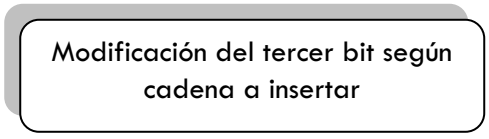
Ahora el siguiente paso será insertar cada bit de la representación binaria de nuestro texto encriptado en cada pixel, modificando los últimos tres elementos del valor con los valores '101' y '010' dependiendo si el valor a insertar es '1' o '0' esto se hace con el motivo de obtener la menor perdida de información ya que al comprimir nos queda un rango de cambio en el bit que queremos modificar que es el tercer menos significativo esto lo explicamos de la siguiente manera.

Tenemos los siguientes valores que se pueden tomar.



Si consideramos que tomaremos el tercer bit de la cadena '010' cuando se quiera representar un cero, tenemos un rango en el que los bits podrían cambiar y el tercer bit seguirá siendo cero, de la misma manera pasa con la cadena '101' para representar un uno, es por esa razón por la que se eligieron esos valores.

Si nuestra primera cadena es 10100001001001 cambiaremos el valor del primer pixel con el valor '101', el segundo pixel con el valor '010' y así sucesivamente según sea el valor del bit de la cadena a insertar en cuestión, esto lo ejemplificamos con la siguiente tabla:



	ROJO	VERDE	AZUL
Pixel 1 modificado	10010101	00111101	10010101
Pixel 2 modificado	00111010	10010010	00111010
Pixel 3 modificado	10010101	00111101	10010101
Pixel 4 modificado	00111010	10010010	00111010
Pixel 5 modificado	10010010	00111010	10010010
Pixel 6 modificado	00111010	10010010	00111010

Pixel 7 modificado	10010010	10010010	10010010
Pixel 8 modificado	00111101	10010101	00111101
Pixel 9 modificado	10010011	00111010	10010011
Pixel 10 modificado	00111010	10010010	00111010

TABLA 3. VALORES MODIFICADOS 1

Una vez hecho esto se graban los nuevos pixeles en la imagen y se procede con la compresión.

ALGORITMO DE RECUPERACIÓN

El algoritmo de recuperación es un poco más sencillo que la parte de ocultamiento, el proceso para ello se describe a continuación.

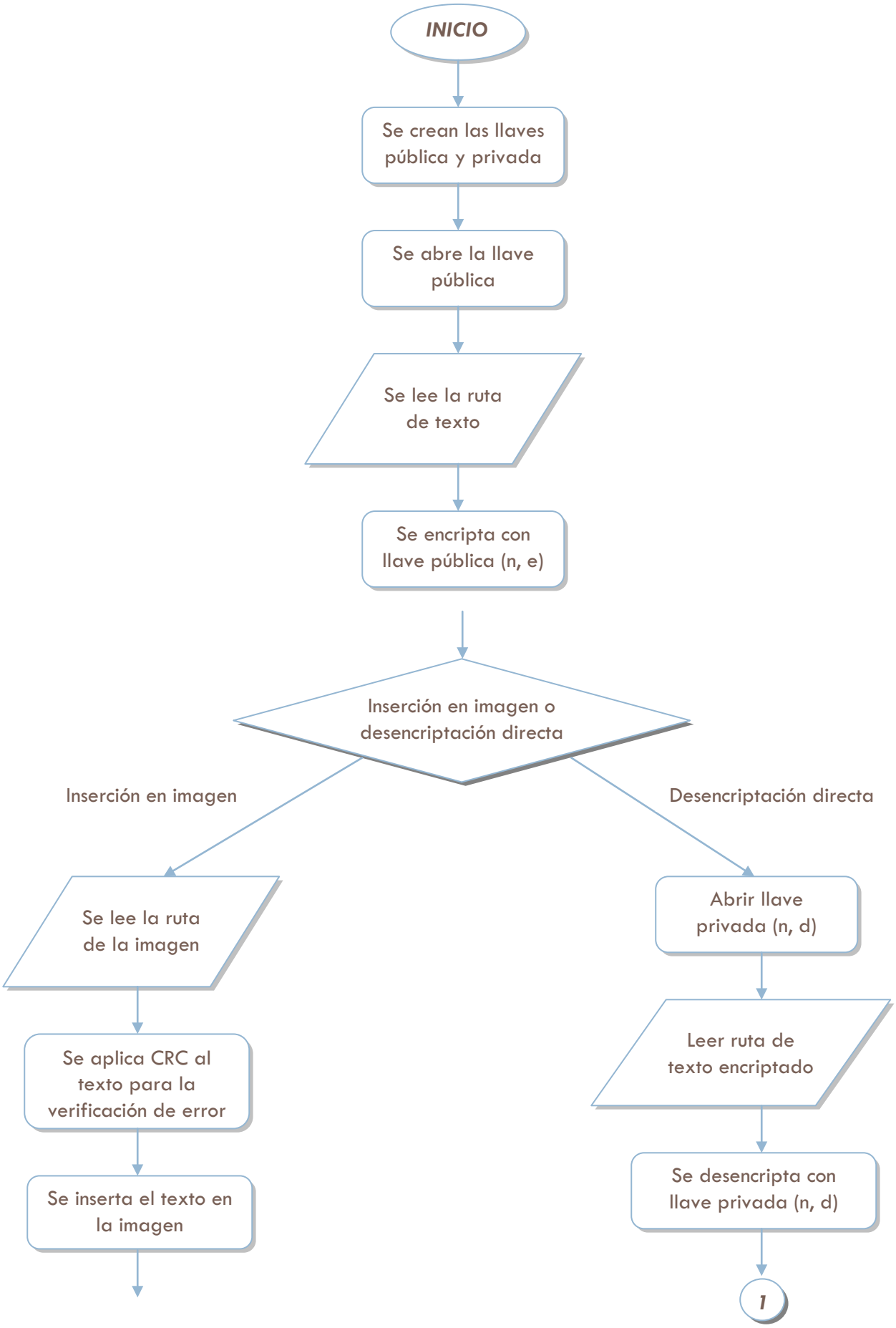
- Se abre la imagen y se obtienen los valores nuevos de los pixeles tras la compresión
- Se extrae el valor del tercer bit menos significado de cada pixel hasta llegar al número de bits establecido en una variable, la cual se guardo previamente y representa la longitud total de la cadena de bits que se insertó.
- Una vez obtenidas las cadenas de bits, se hace una conversión a su representación decimal.
- Con estos valores se obtiene la suma de control de cada uno de ellos para comprobar cualquier error tras la compresión.
- Se abre la llave privada (n, d) con la cual para finalizar se procede a la desencriptación de nuestro documento de texto.

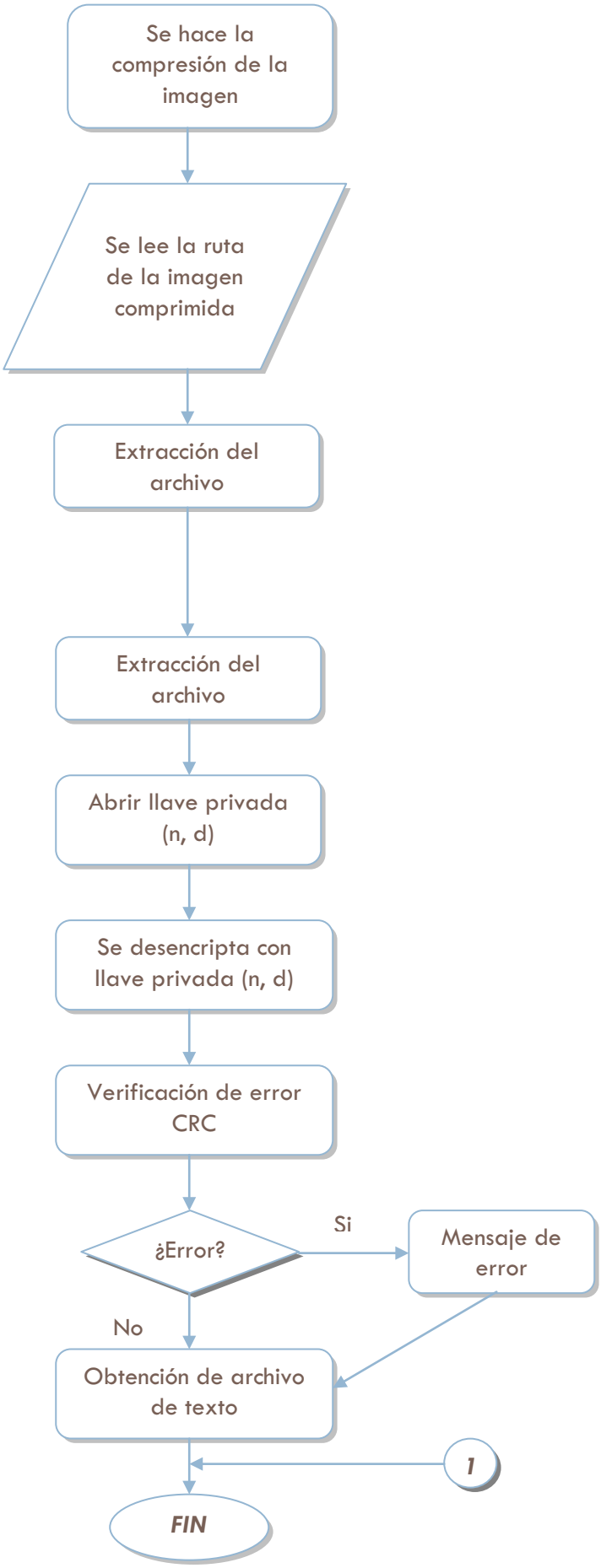
DISEÑO

A continuación se presentarán los diagramas de flujo que esquematizan el procedimiento que se llevó a cabo para la encriptación y desencriptación del mensaje.

Así mismo el usuario crea sus propias llaves pública y privada y especifica el archivo de texto plano y la imagen en la cual se ocultará.

Cabe mencionar que en el presente proyecto se estipulan dos métodos de desencriptación, el primero es la decodificación del texto directamente del archivo donde se encuentran los valores encriptados, la segunda opción es extraerlo de la imagen donde fue insertado, los dos métodos son ejemplificados en el siguiente diagrama.


INICIO
Se crean las llaves pública y privada
Se abre la llave pública
Se lee la ruta de texto
Se encripta con llave pública (n, e)
Inserción en imagen o desencriptación directa
Inserción en imagen
Desencriptación directa
Se lee la ruta de la imagen
Se aplica CRC al texto para la verificación de error
Se inserta el texto en la imagen
Abrir llave privada (n, d)
Leer ruta de texto encriptado
Se desencripta con llave privada (n, d)
1


Se hace la compresión de la imagen
Se lee la ruta de la imagen comprimida
Extracción del archivo
Extracción del archivo
Abrir llave privada (n, d)
Se desencripta con llave privada (n, d)
Verificación de error CRC
¿Error?
Si
Mensaje de error
No
Obtención de archivo de texto
1
FIN

El archivo de texto contendrá los caracteres obtenidos tras la desencriptación, en caso de haber habido error contendrá caracteres diferentes al archivo original.

DESARROLLO

Los algoritmos antes descritos fueron implementados en un sistema desarrollado bajo el lenguaje de programación Java, distribuido en varios paquetes los cuales explicaremos a continuación:

Paquetes	
interfazproyecto	Paquete que contiene el código necesario para crear nuestra interfaz del sistema
proyecto.crc	Contiene los códigos para aplicar el Código de Redundancia Cíclica
proyecto.desencriptador	Contiene los códigos para desencriptar el texto, ya sea de la imagen o directo de el archivo encriptado.
proyecto.encriptador	Aquí se encuentra el código para encriptar el archivo de texto y generar otro con los códigos después del encriptado.
proyecto.filtros	Aquí se encuentran los archivos que utiliza la clase JFileChooser para filtrar los archivos de tipo 'txt', 'jpg' y 'bmp'.
proyecto.generador.llaves	En este paquete encontraremos los códigos para generar las llaves pública y privada
proyecto.maneja.imagen	Se encuentran los archivos para realizar las tareas que tienen que ver con la imagen como son obtener los valores de los pixeles, insertar el archivo de texto, guardar la imagen modificada, además de contener una clase para la conversión a decimal y binario
proyecto.selector.imagen	Contiene los archivos de la clase JFileChooser para obtener la ruta de la imagen, también contiene las clases necesarias para hacer la compresión de la imagen y presentarla en el panel de la interfaz.

TABLA 4. PAQUETES 1

El sistema fue desarrollado bajo en entorno de desarrollo Netbeans 6.8 el cual se encuentra bajo licencia GPLv2 la cual establece la libertad de compartir y modificar software libre, para asegurarse de que el software es libre para todos sus usuarios.

Es así como los paquetes que contienen los códigos en los que aplicamos los procesos de Esteganografía se encuentran en los paquetes:

proyecto.encriptador

- ***Encriptador.java***

proyecto.desencriptador

- ***Desencriptador.java***
- ***DesencriptadorImagen.java***

proyecto.maneja.imagen

- ***Procesador_imagen.java***

En el archivo ***encriptador.java*** se aplica el algoritmo RSA antes descrito para codificar nuestro mensaje, esto los hacemos mediante la función encriptar() en la donde utilizamos nuestra llave pública y se guarda el resultado en otro archivo.

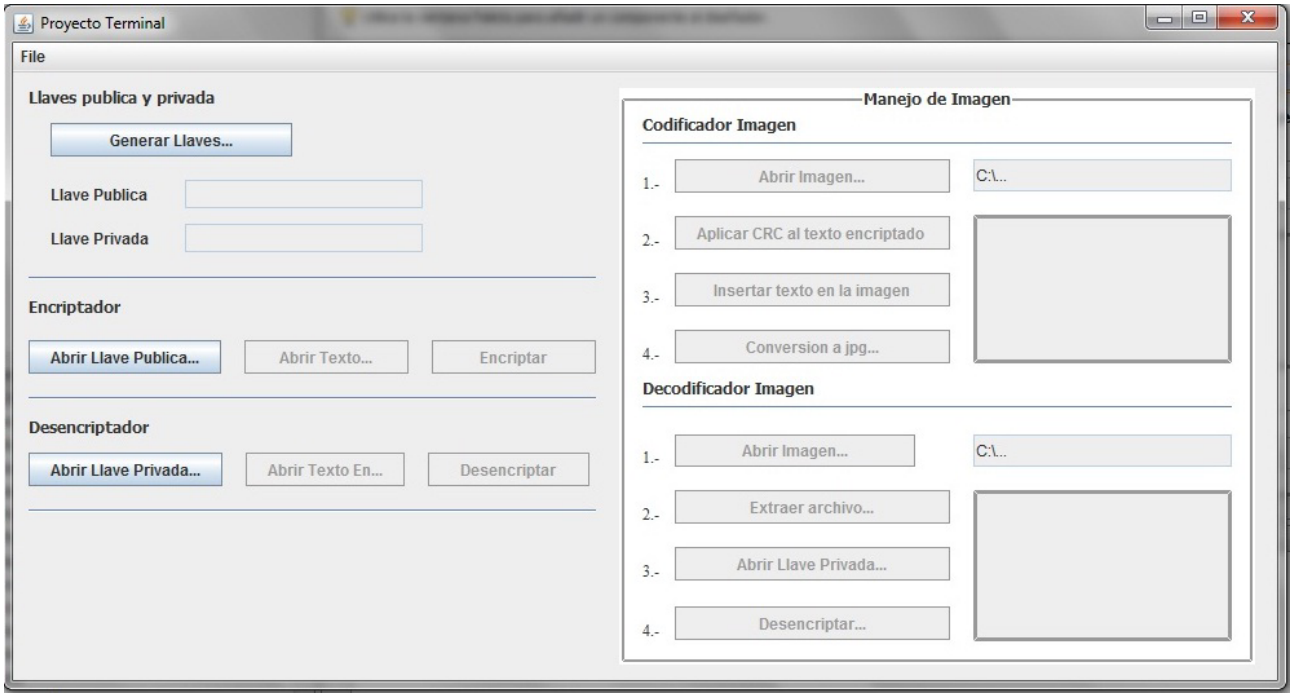
En el archivo ***Desencriptador.java y DesencriptadorImagen.java*** utilizamos los códigos resultantes de aplicar la función encriptar () para el primer código y procederemos a la desencriptación con nuestra llave privada para el segundo código será ligeramente distinto ya que obtendremos los códigos extrayéndolos de la imagen y no de un archivo, de la misma manera también utilizaremos la llave privada para proceder a la desencriptación.

En el archivo ***Procesador_imagen.java*** se realizan todas las tareas que tienen que ver con la imagen como puede ser que se obtengan los valores de los pixeles de la imagen, se modifican y se guardan en la nueva imagen.

PRUEBAS Y RESULTADOS

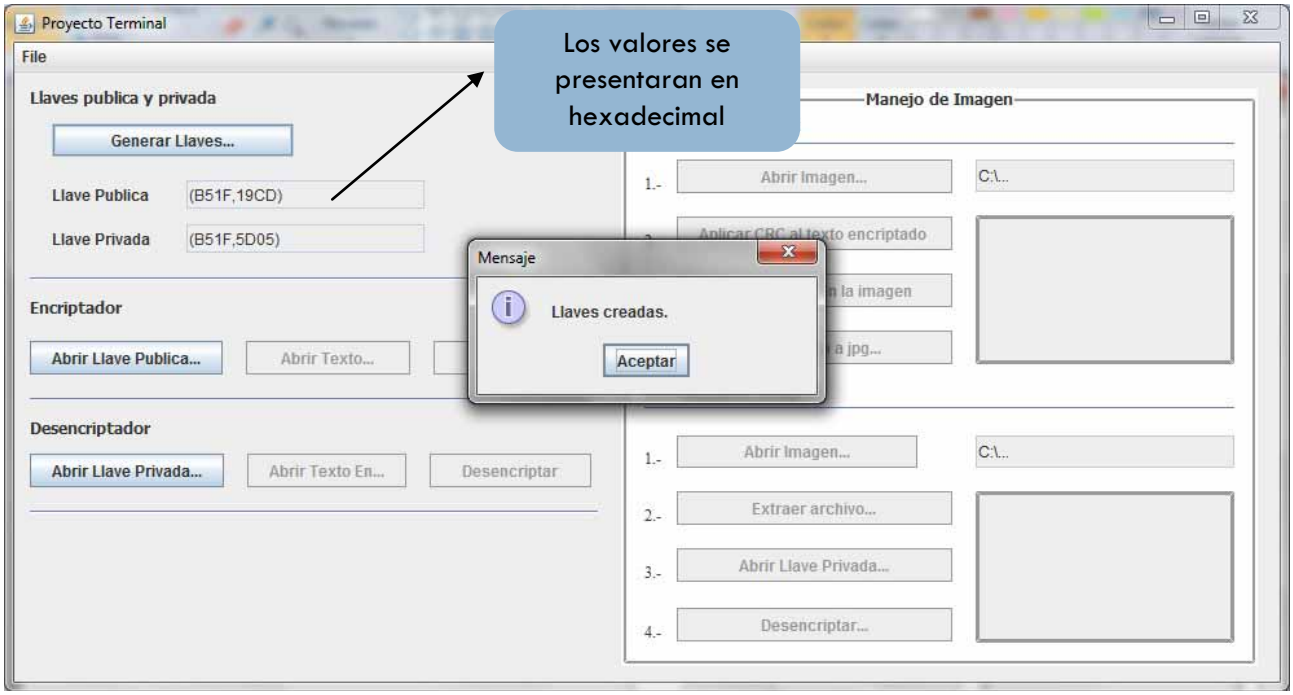
Se realizaron varios tipos de pruebas con diferentes archivos los cuales se adjuntaran al presente proyecto. Una de las primeras pruebas se realizó con el archivo de texto ***prueba.txt*** el cual contiene la cadena **"Texto a encriptar"** y con la imagen de prueba ***pruebapix.bmp*** la cual se diseñó específicamente con tres partes donde se pueden mostrar los tres colores principales que conforman el modelo RGB y así se pueda apreciar el mínimo cambio que sufrirá esta imagen después de la inserción del texto, dicha imagen se mostrará en las siguientes capturas de pantalla.

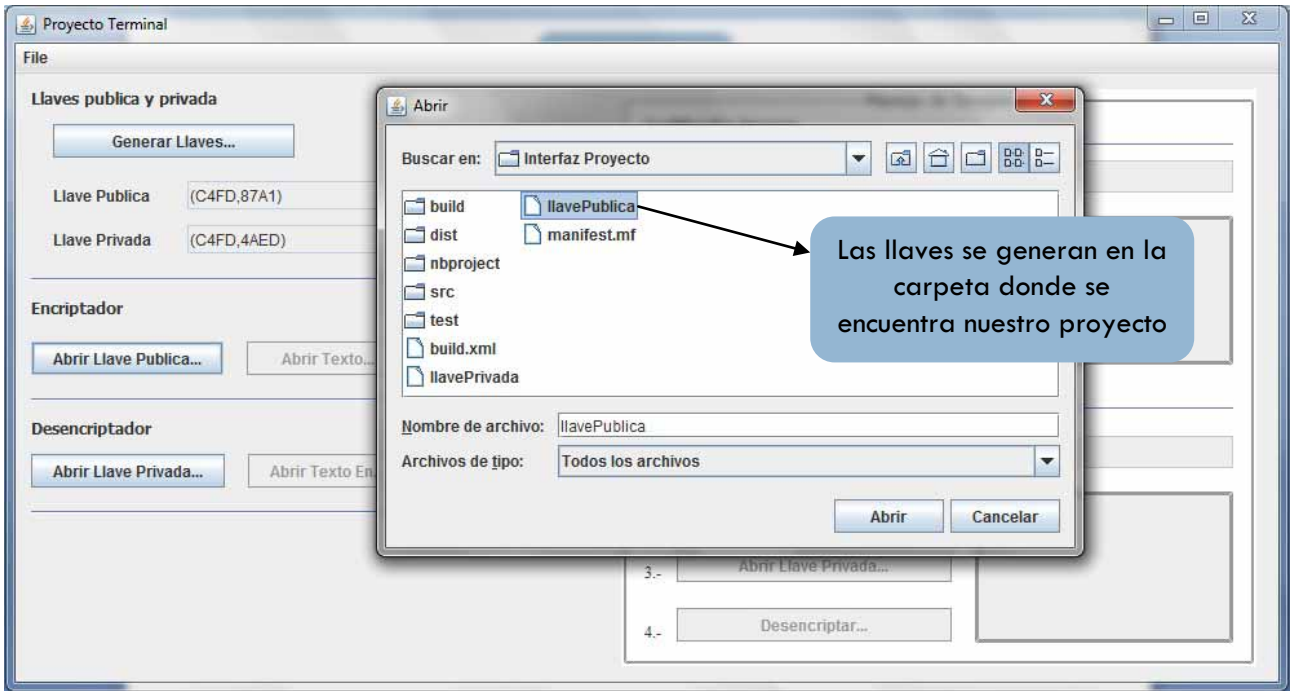
Una vez ejecutado el proyecto nuestra interfaz se presentará de la siguiente forma.



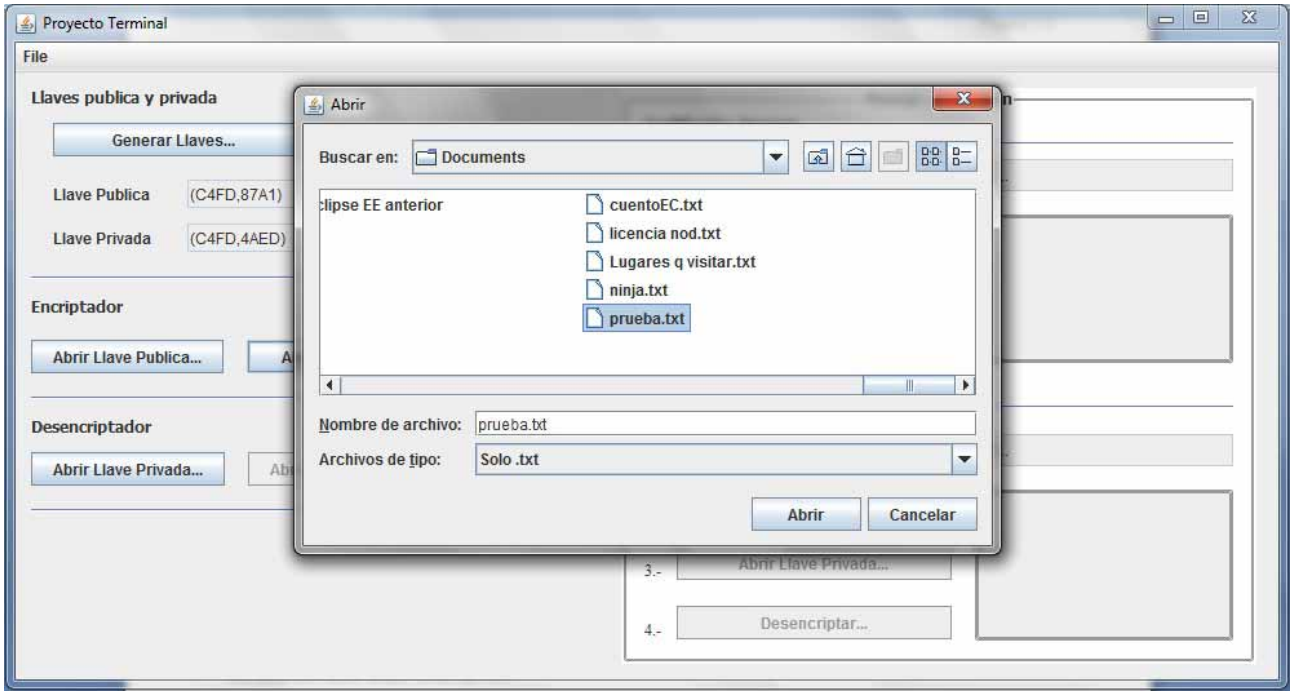
Como se mencionaba anteriormente se pueden observar dos partes fundamentales la primera es la parte para la generación de las llaves pública y privada así como el módulo para encriptar y desencriptar el texto y la segunda parte es enteramente para el manejo de la imagen cualquiera que sea el método que se elija el sistema creará un archivo con el mismo nombre con el que fue elegido pero con un carácter **'D'** concatenado al final esto para indicar que es el archivo que ha sido desencriptado.

El primer paso a seguir será crear nuestras llaves y proceder a la encriptación del mensaje esto se ejemplifica en las siguientes capturas de pantalla.

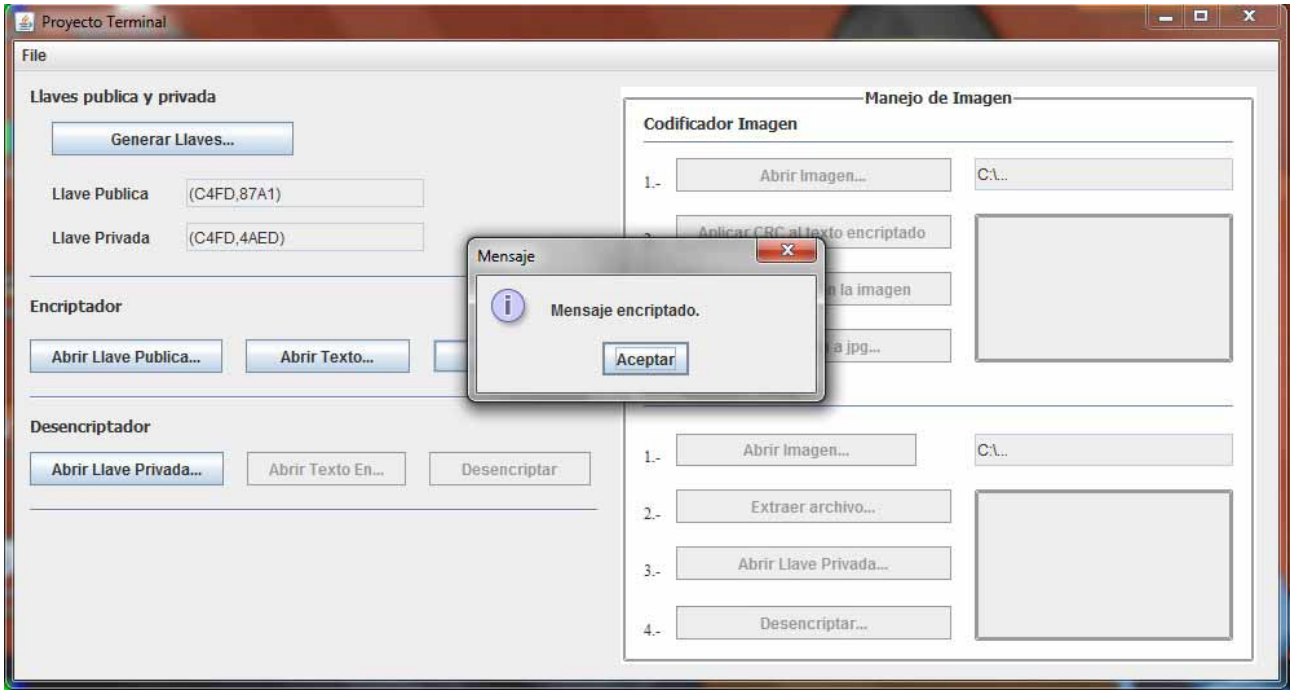




De la misma forma obtenemos la ruta del archivo que vamos a encriptar.



Y procedemos a la encriptación.



Una vez hecho esto en la misma ruta donde se encuentra el archivo original se generará otro con el carácter **'E'** concatenado al final de manera que si nuestro archivo se llama ***prueba.txt*** el archivo se llamara ***pruebaE.txt*** y contendrá nuestro mensaje cifrado.

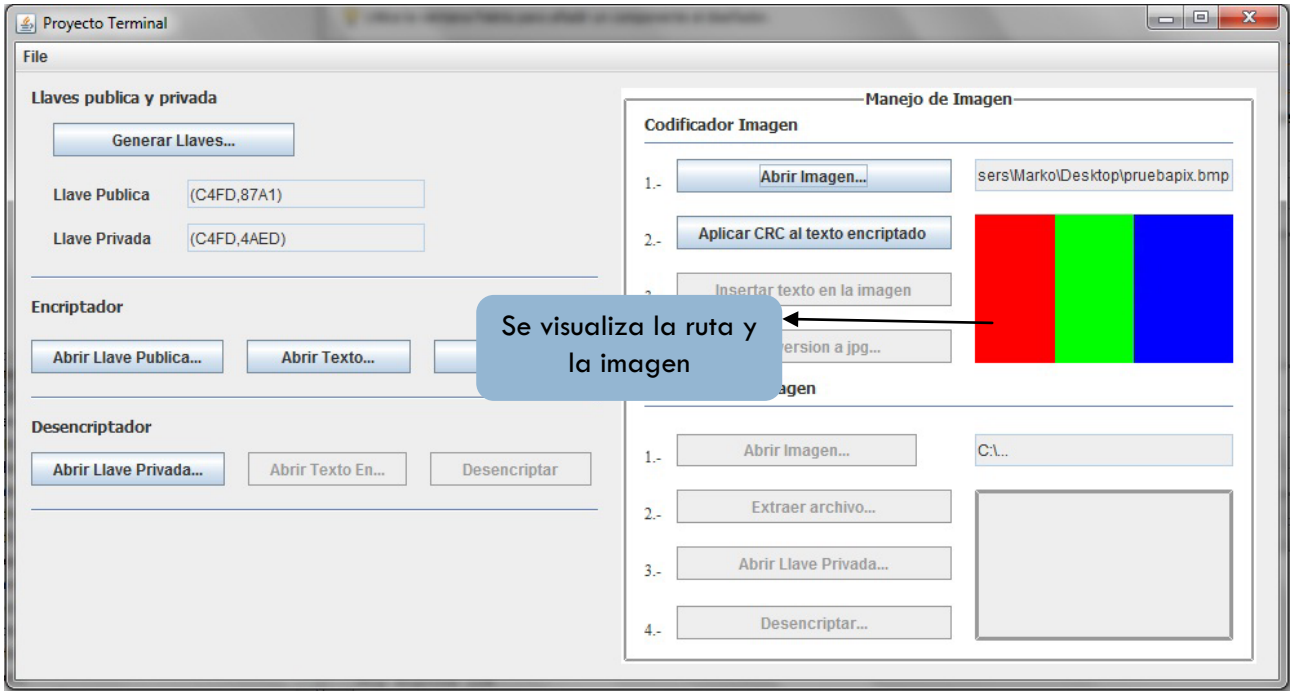
El resultado obtenido se refleja en la siguiente tabla.

Contenido del texto	Resultado tras la encriptación	Representación binaria
T	3804	111011011100
e	1650	11001110010
x	31327	111101001011111
t	48478	1011110101011110
o	23498	101101111001010
	16484	100000001100100
a	2207	100010011111
	16484	100000001100100
e	1650	11001110010
n	39073	1001100010100001
c	28816	111000010010000
r	36406	1000111000110110
i	45569	1011001000000001
p	2834	101100010010
t	48478	1011110101011110
a	2207	100010011111
r	36406	1000111000110110

TABLA 5. VALORES ENCRIPTADOS 1

Como habíamos mencionado hay dos métodos que podemos elegir para desencriptar el mensaje, uno es directamente del archivo resultante y el otro es insertarlo en la imagen, nosotros nos concentraremos en el segundo, entonces los pasos a seguir serán los siguientes:

Seleccionaremos la imagen y esta se mostrará en un panel destinado para ello.



Aplicamos el algoritmo CRC a cada elemento de nuestro archivo encriptado para la verificación de errores tras la compresión. El siguiente paso será insertar el texto y guardarlo como la imagen modificada, una vez hecho esto procederemos al siguiente módulo donde se podrá apreciar el cambio entre las dos imágenes.

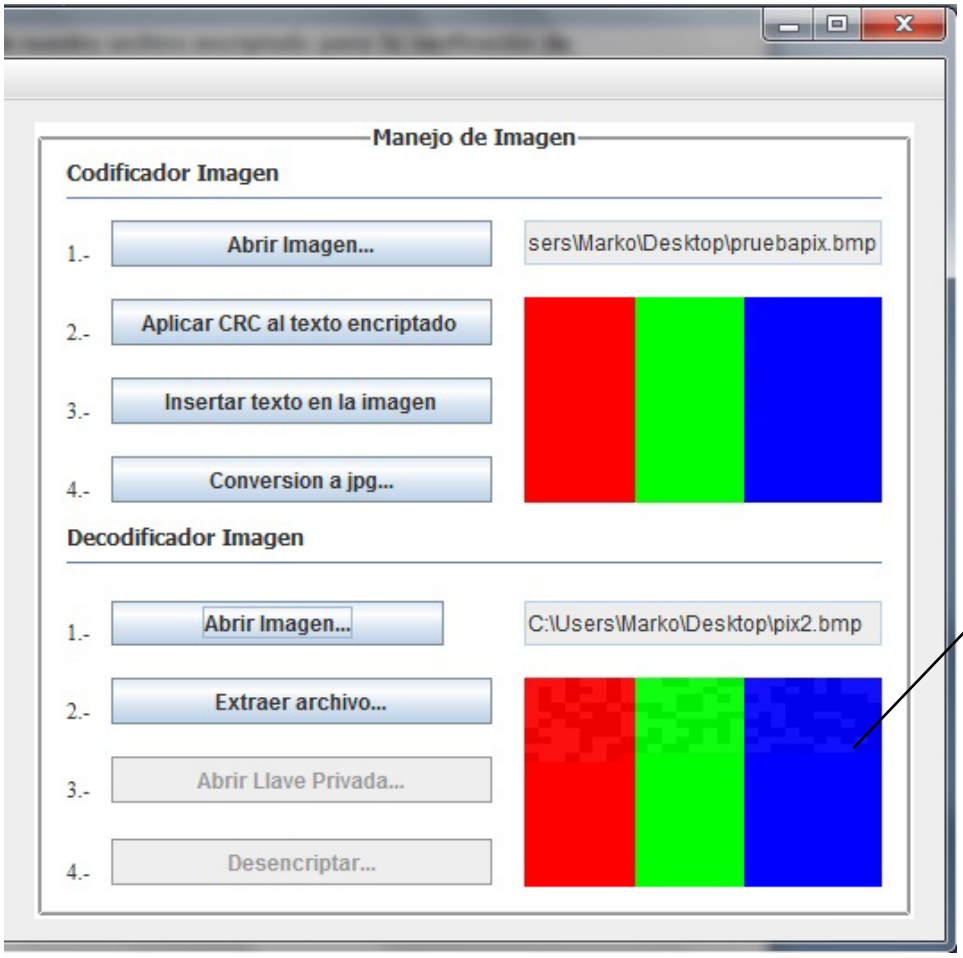
Antes de aplicar la compresión se realizarán las pruebas en la imagen sin compresión para que se pueda apreciar los cambios en ella, la captura de pantalla que continua nos muestra el resultado.

Como la matriz de pixeles es muy grande presentaremos los valores originales de los primeros diez pixeles para ejemplificar el procedimiento.

	ROJO	VERDE	AZUL
Pixel 1	11111111	00000000	00000000
Pixel 2	11111111	00000000	00000000
Pixel 3	11111111	00000000	00000000
Pixel 4	11111111	00000000	00000000
Pixel 5	11111111	00000000	00000000
Pixel 6	11111111	00000000	00000000
Pixel 7	11111111	00000000	00000000
Pixel 8	11111111	00000000	00000000
Pixel 9	00000000	11111111	00000000
Pixel 10	00000000	11111111	00000000

TABLA 6. VALORES DE LOS PIXELES 1

Después de realizar la inserción los cambios en la imagen serán poco notables.



En esta parte es donde se puede apreciar el ligero cambio entre la imagen original y la modificada cada cambio de color es porque ahí se encuentra una parte del mensaje que ocultamos.

El paso posterior será extraer el archivo de cada bit que hemos modificado en la imagen, una vez hecho esto abriremos la llave privada que ya creamos anteriormente y procederemos a desencriptar el archivo.

Entonces si la primera letra que insertamos es una 'T' como se presenta a continuación.

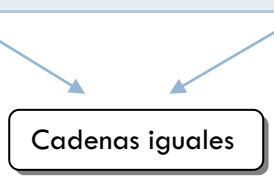
Texto	texto encriptado	Cadena que se tiene que recuperar
T	3804	111011011100

Se tendrá que extraer el tercer bit menos significativo y se hace un promedio, para ver si obtendremos un '1' o un '0', veremos los siguientes valores recuperados de la imagen modificada y veremos la cadena resultante.

	ROJO	VERDE	AZUL	Cadena Extraída
Pixel 1	11111101	00000101	00000101	1
Pixel 2	11111101	00000101	00000101	1
Pixel 3	11111101	00000101	00000101	1
Pixel 4	11111010	00000010	00000010	0
Pixel 5	11111101	00000101	00000101	1
Pixel 6	11111101	00000101	00000101	1
Pixel 7	11111010	00000010	00000010	0
Pixel 8	11111101	00000101	00000101	1
Pixel 9	00000101	11111101	00000101	1
Pixel 10	00000101	11111101	00000101	1
Pixel 11	00000010	11111010	00000010	0
Pixel 12	00000010	11111010	00000010	0

Ahora procedemos a comparar la cadena original con que obtuvimos al extraer cada bit de la tercer posición menos significativa el resultado es.

Cadena original	Cadena Extraída	Carácter que se recupera tras desencriptar
1110110111100	1110110111100	**T**

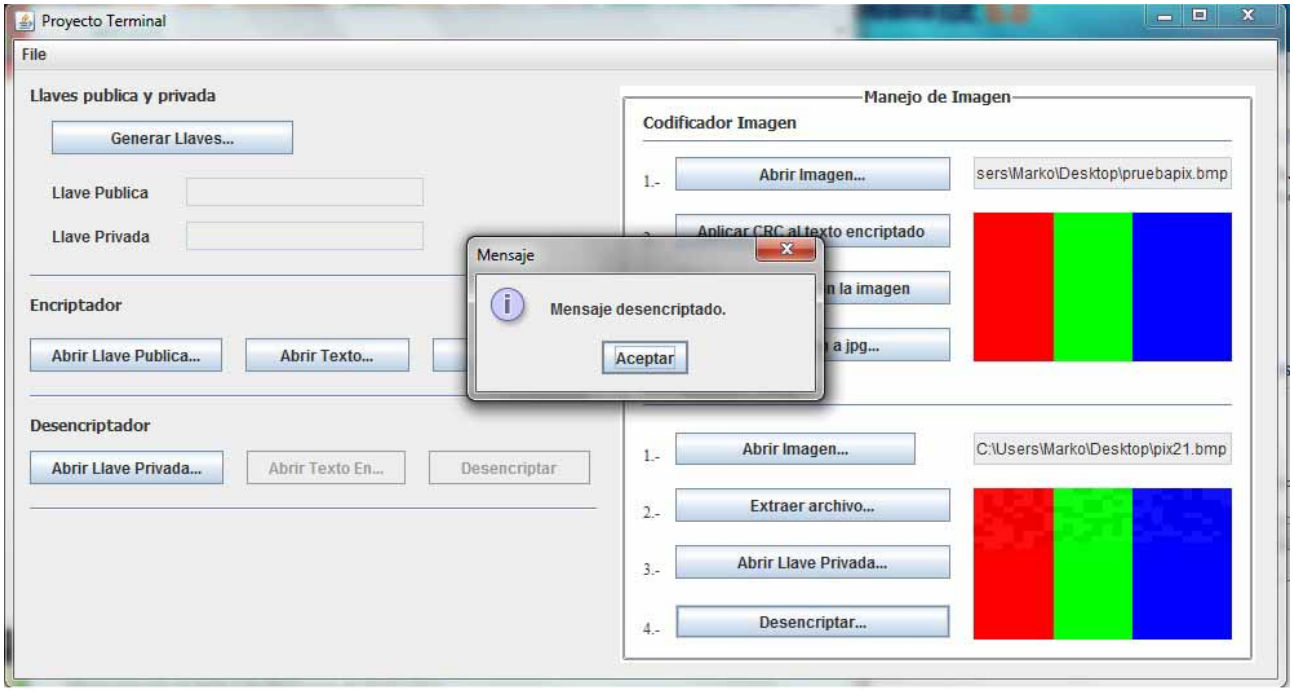


Como podemos observar se pudo recuperar nuestro primer carácter de nuestro texto original, para los caracteres subsecuentes se sigue exactamente el mismo procedimiento, hasta recuperar todo el archivo.

Una vez hecho esto se generará otro archivo con el mismo nombre que el original pero con el carácter 'D' concatenado al final esto para indicar que es el archivo desencriptado.

El archivo se encontrará en la misma ruta que la del archivo original pero con la característica antes descrita.

La siguiente captura de pantalla nos ilustra este procedimiento final, en caso de haber habido error nos lo indicará en un mensaje como el que se muestra en la ventana de aviso, pero con la leyenda "CRC: Error tras desencriptado".



Como pudimos observar nuestro archivo se recupera totalmente, lamentablemente al comprimir la imagen, el mismo algoritmo JPG hace que se pierda demasiada información y no es posible recuperar toda nuestra información.

PROPUESTAS DE POSIBLES MEJORAS O AMPLIACIONES AL SISTEMA.

El presente proyecto terminal implementa el algoritmo CRC para detectar un error tras la compresión, en caso de haberlo solamente lo indica, sin embargo no lo corrige, una posible ampliación del sistema podría ser que se implementara un módulo de corrección de errores, utilizando algún algoritmo como el de código de Haming, el cual se sugiere en la propuesta del presente proyecto, o algún otro que consiga recuperar los bits perdidos tras comprimir la imagen.

REQUISITOS DEL SISTEMA

- Para poder usar el sistema bajo cualquier plataforma, primeramente se debe tener instalado el JDK de java se sugiere usar la versión jdk-6u14 para la plataforma en que se ejecute ya que fue la que se utilizo para el desarrollo del sistema.
- Algún IDE de desarrollo para visualizar el código puede ser Eclipse o Netbeans, Netbeans 6.8 fue el IDE sobre el cual se desarrolló el sistema.

MANUAL DE USUARIO

Para ocultar cualquier archivo de texto plano se requiere lo siguiente.

- El archivo de texto que contenga el mensaje a ocultar.
- Una imagen de tipo mapa de bits.

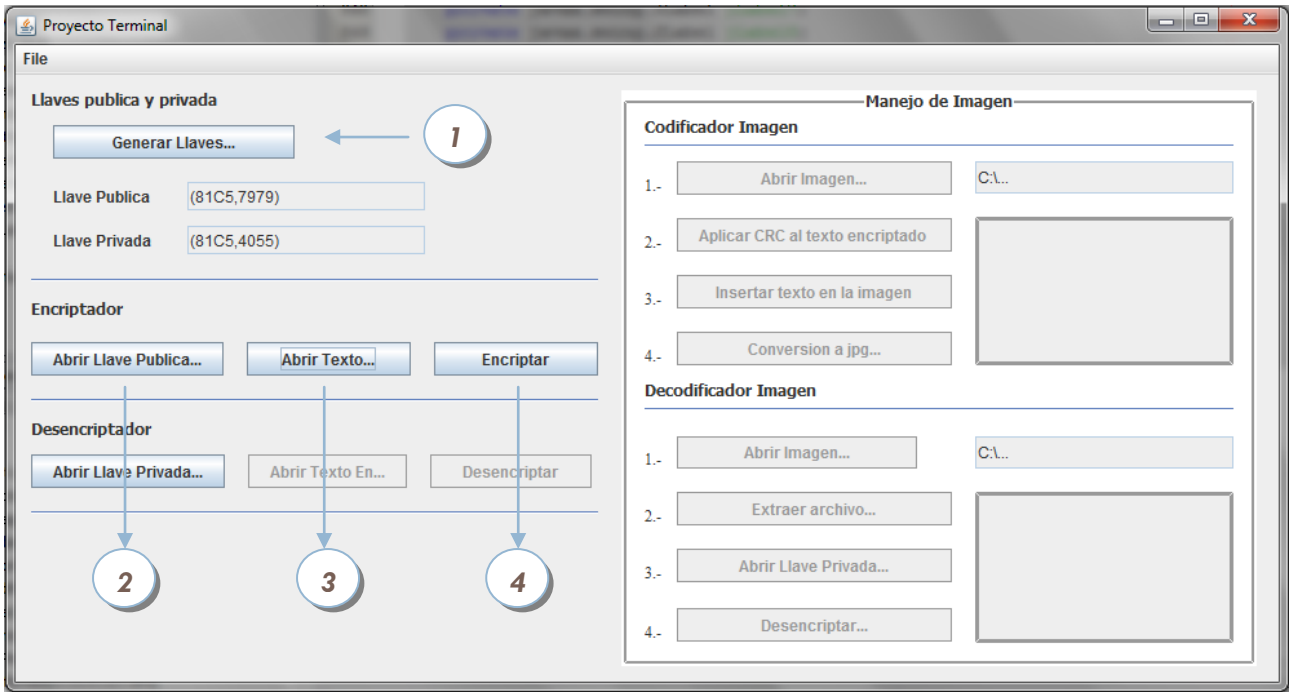
Una vez que nos aseguramos de tener estos dos archivos, necesitamos pegar la carpeta que contiene todos los archivos necesarios para que el sistema funcione, en la carpeta de trabajo de nuestro IDE en cuestión. Esta carpeta se adjunta al presente proyecto.

Posteriormente cargamos el proyecto en el IDE y lo compilamos para obtener nuestro archivo .jar que será nuestro ejecutable.

Los pasos posteriores se muestran a continuación.

1. Una vez que ejecutemos el proyecto se presentará la pantalla que mostramos anteriormente, lo primero que debe hacerse es crear las llaves pública y privada, estas se generaran en la misma carpeta del proyecto con los nombres “llavePublica” y “llavePrivada”.
2. El siguiente paso será abrir la llave pública con el botón destinado para ello.
3. Seleccionar el archivo el cual se encriptará.
4. Y finalmente encriptamos el mensaje, el cual se guardará en otro archivo con el mismo nombre y con la misma ruta en la que se encontraba el original, pero con el carácter ‘E’ al final.

Todos estos pasos los ejemplificamos a continuación.



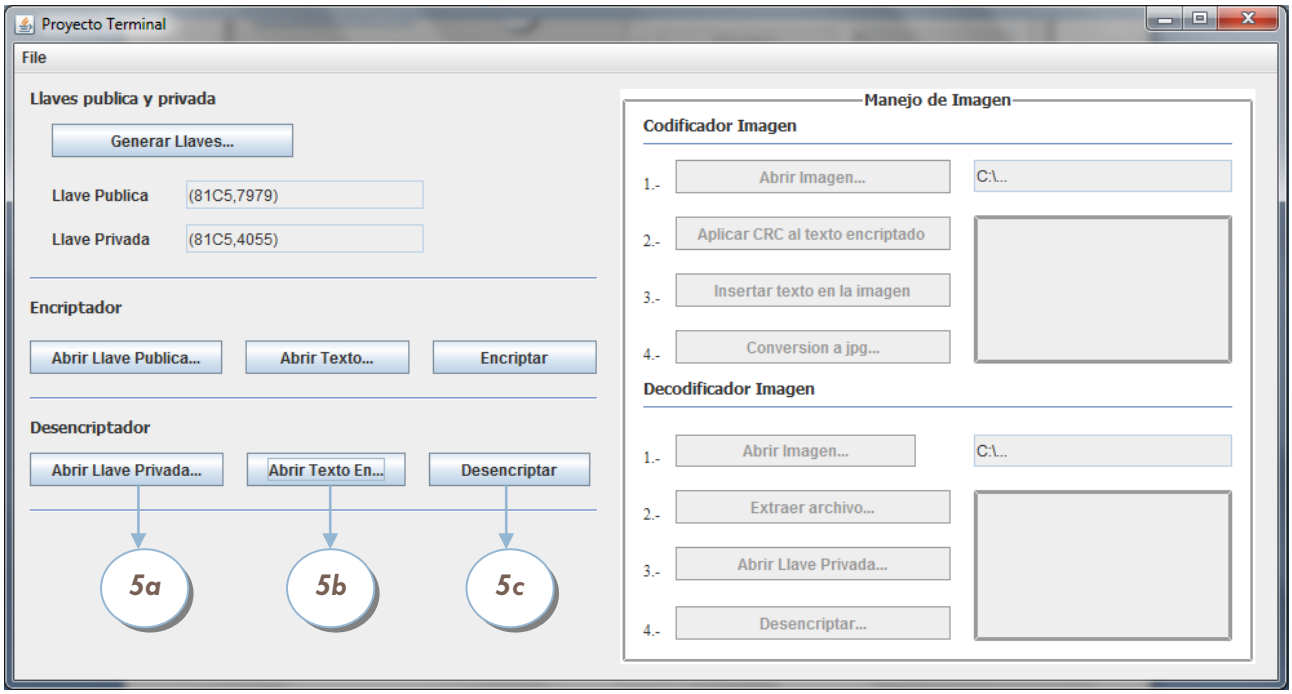
Ahora tendremos dos opciones para desencriptar el archivo, la primera será hacerlo directo del archivo encriptado.

5a. Abrimos la llave privada que se encuentra en el directorio del proyecto

5b. Abrimos el archivo que se creó con el mismo nombre y con terminación 'E'

5c. Se creará un nuevo archivo con el resultado del desencriptamiento el cual ahora tendrá el mismo nombre y ruta que el original pero con el carácter 'D' concatenado al final.

Veamos los pasos a seguir a continuación



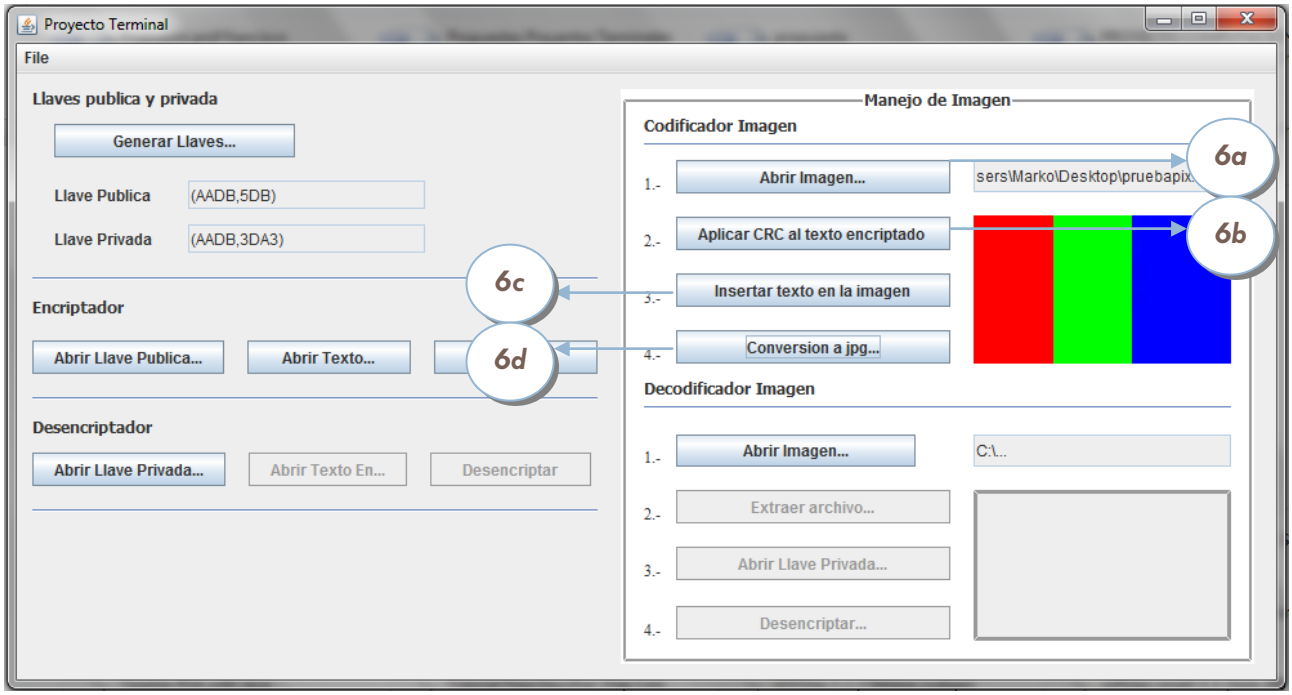
Ahora la segunda opción será ocultar el archivo encriptado en una imagen para después extraerla de la imagen modificada, los pasos a seguir en este caso son los siguientes.

6a. Abriremos la imagen en la cual pensamos ocultar nuestro archivo encriptado

6b. Aplicaremos el algoritmo CRC a nuestro archivo creado al encriptar, recordando que es aquel que tiene el mismo nombre que el original pero con el carácter 'E' concatenado al final, ahora se concatenara el carácter 'C' para identificarlo.

6c. Una vez realizado esto insertaremos el archivo en la imagen

6d. Se procede a la conversión de la imagen modificada el cual se guardara en donde el usuario elija con el nombre que establezca.



Para finalizar el proceso solo nos restará extraer nuestros códigos encriptados y obtener nuestro mensaje, los pasos se muestran a continuación.

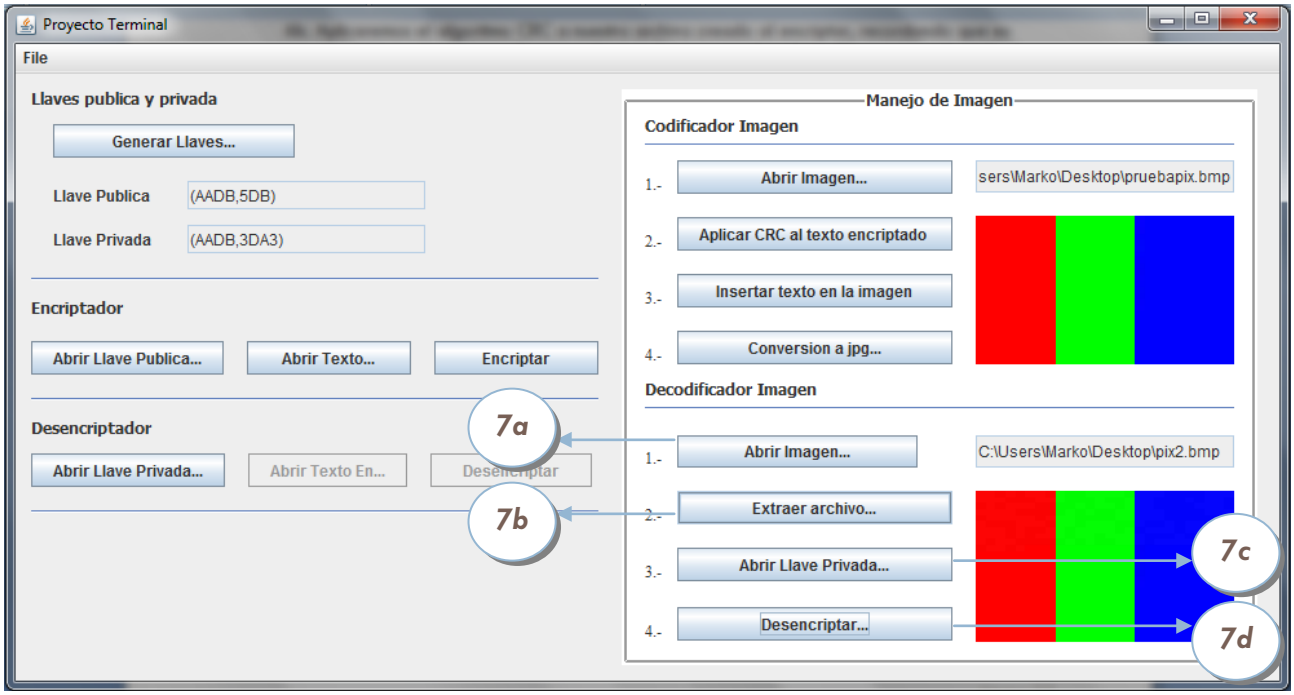
7a. Abriremos la imagen modificada con nombre y ruta establecida por el usuario

7b. Extraeremos el archivo con los códigos a desencriptar

7c. Abriremos nuestra llave privada

7d. Desencriptaremos el archivo resultante y se generará un nuevo archivo con el mismo nombre y ruta que el original pero con el carácter 'D' concatenado al final, es en este paso donde a su vez se hace una verificación con los códigos CRC que se encuentran en el archivo con terminación 'EC', en caso de haber error tras la desencriptación el sistema lo indica, y se ve el resultado en el archivo resultante.

Vemos que en realidad el sistema es muy intuitivo y simple esto con el propósito de hacerlo más fácil para el usuario final, veamos estos últimos pasos en la última captura de pantalla.



MANUAL DE INSTALACIÓN

La instalación del sistema consiste simplemente en:

- Colocar la carpeta con los paquetes del programa la cual se adjunta a este proyecto en el directorio de trabajo del IDE en cuestión.
- Ejecutar el sistema con los parámetros antes descritos en el manual de usuario.

REFERENCIAS

[1] Álvaro Gómez Vieites (2007), *Enciclopedia de la seguridad informática*

Alfa omega Ra-ma.

[2] Gregory Kipper (2004), *Investigator´s guide to Steganography*

Auerbach Publications.

[3] Joel David Villegas Hernández. *Transmisión de mensajes en archivos de imágenes y texto usando Esteganografía*. Asesor: Dr. Francisco Javier Zaragoza Martínez. Concluido en el trimestre 2009-Invierno.

[4] Ángel Pérez García. *Transmisión de archivos binarios cifrados usando Esteganografía en imágenes GIF*. Asesor: Dr. Francisco Javier Zaragoza Martínez.

[5] Claudia Maribel Adriano Rivas. *Esteganografía de mensajes de texto en párrafos justificados*. Asesor: Dr. Francisco Javier Zaragoza Martínez.

[6] Miriam Barboza García. *Esteganografía de texto en audio WAV sin compresión.*

Asesor: Dr. Francisco Javier Zaragoza Martínez.

[7] Henry S. Warren, Jr. Hacker's Delight, *Cyclic Redundancy Check: theory, practice, hardware, and software with emphasis on CRC-32.*

Link: http://www.hackersdelight.org/crc.pdf

Última consulta: 26 de febrero de 2010