UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO DE UN SISTEMA DE ESTEGANOGRAFIA: INCRUSTACION Y EXTRACCION DE INFORMACION PRIVADA EN IMAGENES DIGITALES NO COMPRIMIDAS

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A

JOSE GABRIEL OREA FLORES

DIRECTOR DE TESIS: DR. MIGUEL MOCTEZUMA FLORES

CIUDAD UNIVERSITARIA MARZO 2003

A

PAGINACION

DISCONTINUA

Diseño de un sistema de esteganografía:

Incrustación y extracción de información privada en imágenes digitales no comprimidas

por

José Gabriel Orea Flores

Universidad Nacional Autónoma de México
Facultad de Ingeniería

Tesis dirigida por: Dr. Miguel Moctezuma Flores.

Ciudad Universitaria, México, febrero de 2003.

Agradecimientos

Agradezco a mi papá el inculcarme el deleite de estudiar
y a mi mamá por insistirme en escribir esta tesis.

Contenido

Índice de Figuras

Prefacio

En esta tesis, presento el desarrollo de un sistema de esteganografía aplicado a imágenes digitales no comprimidas. La esteganografía, definida como el arte de ocultar el envío de información, ha cobrado gran importancia recientemente debido a la facilidad de comunicación electrónica que proveen las computadoras modernas y las redes internacionales de cómputo.

En el capítulo 1, presento una semblanza histórica de la esteganografía, desde los antiguos relatos griegos y chinos hasta las implementaciones modernas en medios digitales. También evalúo varias aplicaciones comerciales y las comparo contra mi propio sistema esteganográfico que he llamado ***DigiMarker***.

Tradicionalmente, se consideraba a la esteganografía como sinónimo de la criptografía. No obstante, en años recientes la diferencia se ha ido agrandando a tal grado que ahora hay quienes las conciben como disciplinas completamente separadas. También hay algunos investigadores que consideran a la esteganografía como una rama de la criptografía. Aún otros consideran que tanto la esteganografía como el encriptamiento son ramas separadas de una categoría mayor llamada *privacidad de la información*. Personalmente, comparto este último enfoque y de hecho en DigiMarker he implantado ambas tecnologías: primero, un módulo de encriptamiento y luego, la incrustación esteganográfica, de modo que queda bien clara la diferencia de propósito e independencia entre una y otra. En el capítulo 2, presento un breve panorama teórico de la criptografía y sus tecnologías más estudiadas. Si el lector está interesado en la programación, el capítulo 2 también contiene pautas para implementar el método de encriptamiento DES de 128 bits.

En la antigüedad, se usaban diversas formas de contenedores (o portadoras) de mensajes secretos. Actualmente, los contenedores más usados son los medios digitales, con particular énfasis en las fotografías digitales. En el capítulo 3, explico las diferencias entre los formatos gráficos más comunes así como sus ventajas y desventajas para servir de portadores esteganográficos. Presento con más detalle el formato no comprimido de color verdadero (24 bits) llamado mapa de bits (BMP) por ser el utilizado en DigiMarker. Nuevamente, para los lectores con interés en la programación, este capítulo contiene algunos ejemplos de programación en Visual C++ para implementar las utilerías gráficas que DigiMarker requiere.

Habiendo establecido los fundamentos teóricos de varias herramientas necesarias e incluso presentado algunos módulos de software, el capítulo 4 se concentra en el corazón de la implementación esteganográfica. Presento también análisis comparativos con otros sistemas para descubrir debilidades y fortalezas de las distintas tecnologías. Una vez más, incluyo las partes más importantes del código fuente para los lectores interesados en programación.

Siendo esta tesis un trabajo de ingeniería, es muy apropiado el medir objetivamente el rendimiento de la aplicación desarrollada. El capítulo 5 se encarga de esto, poniendo atención en dos rubros: el tamaño del mensaje a incrustar y el cambio en la portadora. Este último parámetro se aborda a su vez por dos vías: la medición aritmética (mediante cálculo de ruido) y la medición subjetiva (mediante apreciación visual). También presento conclusiones sobre los métodos de evaluación.

Finalmente, el capítulo 6 reflexiona sobre el futuro tecnológico de DigiMarker y presenta las conclusiones de la presente tesis.

Uno de mis principales propósitos al escribir este trabajo es que la lectura sea simple pero a la vez precisa e interesante de modo que el lector termine la lectura y se vea beneficiado de ella. Espero que así sea.

1

Introducción

"El medio es el mensaje"

Marshall McLuhan

1.1 Objetivo

Implementar un sistema esteganográfico basado en un algoritmo propio para la incrustación de información privada en imágenes digitales sin alterar visiblemente su apariencia. La información además será encriptada antes de ser incrustada. El sistema a desarrollar deberá poder incrustar cualquier tipo de información digital y recuperarla con 100% de integridad mediante una interfaz moderna.

1.2 Reseña del presente trabajo

Uno de mis pasatiempos favoritos es la fotografía. Con el auge de la Internet, ahora prácticamente cualquier fotógrafo tiene la posibilidad de publicar sus trabajos al mundo, privilegio que antes estaba restringido a sólo unos cuantos. Así que pronto comencé a publicar algunas de mis galerías fotográficas. Conforme pasó el tiempo, me di cuenta de que publicar las fotografías así nada más era prácticamente regalarlas a quien quisiera copiarlas, dadas las facilidades en esta era digital. De modo que me propuse encontrar una forma que proveyera algún tipo de seguridad orientada a proteger los derechos de autor.

Después de pensar durante algunas semanas, me propuse incrustar mi nombre dentro de mis fotografías digitales, de modo que en cualquier momento pudiera comprobar que esas fotografías eran trabajo original mío. Comencé una investigación en la Internet y entonces me di cuenta de que eso en que estaba pensando tenía un nombre: *Marca de Agua digital*. No sólo tenía un nombre sino que había ya una que otra aplicación comercial disponible (corría el año 2001). De todos modos, seguí adelante desarrollando mi propia idea de cómo sería un buen diseño que me proveyera de marcas de agua digitales.

Con el paso del tiempo, logré una primera solución que ocultaba mis datos personales en un marco artificial agregado a la fotografía (Fig. 1). Implementé una compresión estilo *Huffman*[1] para reducir mis datos y de esa forma era más sencillo ocultarlos dentro del marco de la foto sin que fuera perceptible al ojo humano, incluso al revisar píxel por píxel. Me

entusiasmó mucho el resultado obtenido, pero pronto me pregunté si sería posible insertar datos sin necesidad del marco artificial.

Figura 1. Fotografía original

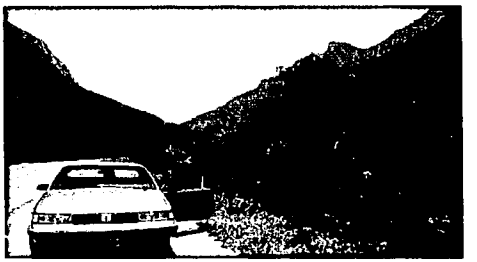


Figura 1b. Fotografía con marco artificial añadido

Figura 1c. Ampliación al marco superior izquierdo

Figura 1d. Detalle de área donde se incrustó la información

Seguí pensando al respecto y desarrollé un método simple pero bastante efectivo para incrustar mis datos en la fotografía, sin ningún marco o elemento artificial. Elaboré un primer programa de prueba y los resultados fueron muy satisfactorios también. Nuevamente me puse a investigar en la Internet qué había al respecto y por segunda ocasión descubrí que lo que ahora estaba haciendo también tenía un nombre: se llama la *técnica LSB*[2] y se refiere a usar los bits menos significativos de las fotografías digitales. Como los LSB contienen información tan detallada que el ojo humano prácticamente no puede percibir, entonces la técnica consiste grosso modo en tomar esos bits prestados para insertar ahí cualquier otra información.

Bueno, otra vez llegué tarde, pensé. Pero entonces me propuse otra idea: ¿cómo sería si procurara aumentar la cantidad de información que incrusto en la foto sin que ésta se altere? De esa idea surgieron muchas preguntas más: ¿cuántos bits puedo usar sin que haya cambios visuales notorios? ¿qué es en realidad un cambio visual notorio? ¿es acaso una medida subjetiva? ¿se puede elaborar una técnica que sirva para cualquier tipo de imagen? ¿qué clase de información se puede colocar ahí? ¿qué pasaría con imágenes que tengan áreas relativamente grandes de colores sólidos? ¿cómo sería si comprimo la información antes de incrustarla? ¿y si además la encripto? ¿se podrán usar fotografías en blanco y negro? ¿de

verdad es tan importante que no haya un cambio visual notorio? ¿qué pasa –o mejor dicho, qué quiero que pase- si alguien altera la fotografía que tiene mis datos incrustados?

La reflexión en estas y otras interrogantes me llevó a la conclusión de que debería modificar mi aplicación para conseguir los objetivos y restricciones siguientes:

1) Aumentar considerablemente la cantidad de información a incrustar en una fotografía digital.
2) La fotografía digital debería ser en formato no comprimido, color verdadero (24 bits por píxel).
3) El cambio en la fotografía podría notarse si se comparara píxel por píxel con la fotografía original, pero no debería ser tan cuantioso que si se observara la fotografía por sí misma se pudiera notar ruido extraño en ella.
4) La naturaleza de la información encriptada no debería restringirse. Debería ser posible incrustar texto, documentos ASCII, e inclusive código ejecutable o cualquier otro tipo de archivo.
5) La estructura de la fotografía debería permanecer intacta, así como su tamaño, de modo que no levantara sospecha alguna de llevar un mensaje artificial en su interior.
6) En caso de modificación de la fotografía, intencional o no, por un tercero, el mensaje secreto debería ser destruido o visiblemente modificado. Esto con el fin de detectar la intervención de un intruso.

El lector podrá adivinar qué pasó enseguida: Investigué y me di cuenta de que efectivamente todo esto tenía en esencia un nombre: *esteganografía*. Palabra extraña, pensé. En efecto, mis correctores ortográficos no reconocían esa palabra y me resultó muy difícil encontrar literatura especializada sobre el tema.[3] Para ese entonces, ya corría el primer trimestre del año 2002 y me enteré de que estaba por celebrarse en Japón apenas el Primer Taller Internacional de Esteganografía. Así que me propuse asistir para aprender de una vez por todas cuál era la situación actual en el mundo de la ingeniería en torno a esta área que tan fascinante me resultaba.

Mientras me preparaba para asistir procuré terminar la implementación de mis objetivos y criterios en una aplicación con aspecto profesional. Le puse nombre: **DigiMarker**. Qué grata sorpresa me llevé al darme cuenta de que DigiMarker tenía un desempeño notable al compararla con los prototipos que de varios países se expusieron. De ahí me convencí de que valía la pena presentar mi implementación como tema de la presente tesis.

1.3 Antecedentes Históricos de la Privacidad en la información

Existen en la literatura muchos datos históricos acerca del envío seguro de información, siendo algunos relatos realmente interesantes. A continuación resumo algunos de ellos.

Aunque la esteganografía ha estado presente por miles de años, ha sido reciente el renovado interés en ella. La idea detrás de la esteganografía es enviar en secreto un mensaje a alguien más. No es exactamente lo mismo que enviar un mensaje secreto sino más bien **ocultar el**

hecho de que se está llevando a cabo la comunicación. En el ámbito de la criptografía a menudo la existencia del mensaje es evidente, pero el mensaje es presentado de tal forma que sólo el receptor con el conocimiento apropiado (método de decodificación y llave) puede recuperarlo. En contraste, la esteganografía interviene al ocultar al mensaje mismo, sin dejar evidencia de que se está llevando a cabo comunicación alguna. Por supuesto, ambas técnicas se pueden combinar encriptando el mensaje y después ocultándolo de modo que si el mensaje fuera descubierto, su contenido todavía permanecería en secreto.

A lo largo de la historia, la comunicación secreta usando esteganografía ha jugado un papel muy importante. Quizás la mención más antigua de la esteganografía se encuentra en los escritos de Herodoto en relación al conflicto entre Grecia y Persia alrededor del año 480 a.C. En aquella época, Persia había pasado varios años levantando un gran ejército para un ataque sorpresivo contra Grecia. La preparación del ejército fue atestiguada por Demarato, un griego que vivía en Persia. Demarato quería enviar un mensaje preventivo a su tierra sin que fuera interceptado por los guardias persas, así que raspó la cera de una tableta normal de escritura de aquella época, escribió el mensaje en la tablilla base y la cubrió con cera nueva. Teniendo la apariencia de una tableta normal de escritura, la tableta pudo llegar bien a Grecia, el mensaje fue recuperado raspando de nuevo la cera y los griegos se enteraron a tiempo del plan persa y presentaron un exitoso plan de contraataque. El crédito de esta victoria se le ha dado al mensaje esteganográfico. Aquí está una traducción del relato de Herodoto:

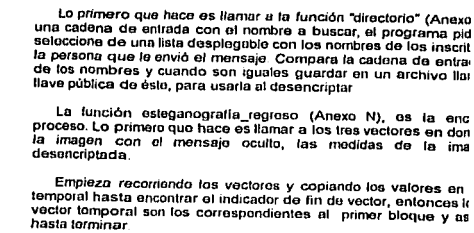
> *Como el peligro de que lo descubrieran era muy grande, sólo había una manera en que podía contribuir a que pasara el mensaje: retirar la cera de un par de tablillas de madera; escribir en la madera lo que Jerjes planeaba hacer y luego volver a cubrir el mensaje con cera. De esta forma, las tablillas, al estar aparentemente en blanco, no ocasionarían problemas con los guardas del camino. Cuando el mensaje llegó a su destino, nadie fue capaz de adivinar el secreto, hasta que, según tengo entendido, la hija de Cleomenes, Gorgo, que era la esposa de Leónidas, lo vaticinó y les dijo a los demás que si quitaban la cera encontrarían algo escrito debajo, en la madera. Se hizo así; el mensaje quedó revelado y fue leído, y después fue comunicado a los demás griegos.*[4]

Herodoto también escribió sobre otro incidente, donde en la cabeza afeitada de un mensajero se escribieron las instrucciones para un alzamiento contra el rey de Persia. Se dejó que el cabello naciera de nuevo y entonces el mensajero pudo llegar a su destino sin obstáculo alguno pues no aparentaba ser portador de ningún mensaje. Su cabeza fue rapada al llegar a su destino y así el mensaje fue leído.

Los chinos antiguos también son conocidos por haber usado una técnica en la que un ser humano era el portador del mensaje esteganográfico. El mensaje se escribía en una finísima seda y luego se insertaba en una pequeña bolita que a su vez era envuelta en cera y luego era tragada por el mensajero. De este modo se podía transportar el mensaje de forma segura.

Otra forma de esteganografía es el uso de *tintas invisibles*. Los relatos más tempranos de esta técnica se remontan al primer siglo de nuestra era, cuando se usaba la "leche" de una planta.

Esas tintas se empleaban para escribir en papel normal, pero desaparecían de la vista al secarse. Para evitar sospechas, sobre el mismo papel se escribían mensajes inocentes con procedimientos normales. Las tintas invisibles tradicionales suelen ser ricas en carbón y se hacen visibles al calentar el papel, pero algunas tintas más sofisticadas requieren algún componente químico para revelarse, algo muy parecido al proceso fotográfico de revelado. Algunas tintas simples son a base de leche, vinagre o jugo de limón. La figura 2 muestra un ejemplo a base de leche.

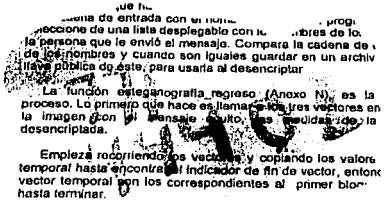


Lo primero que hace es llamar a la función "directorio" (Anexo
una cadena de entrada con el nombre a buscar, el programa pid
seleccione de una lista desplegable con los nombres de los inscrit
la persona que le envió el mensaje. Compara la cadena de entra
de los nombres y cuando son iguales guardar en un archivo llar
llave pública de éste, para usarla al desencriptar

La función esteganografía_regreso (Anexo N), es la enc
proceso. Lo primero que hace es llamar a los tres vectores en don
la imagen con el mensaje oculto, las medidas de la ima
desencriptada.

Empieza recorriendo los vectores y copiando los valores en
temporal hasta encontrar el indicador de fin de vector, entonces le
vector temporal son los correspondientes al primer bloque y as
hasta terminar

Figura 2a. Hoja con mensaje escrito con leche, invisible a simple vista.



...que le...
...cadena de entrada con el nom... ...progr...
...seccione de una lista desplegable con l... ...mbres de lo...
la persona que le envió el mensaje. Compara la cadena de
de los nombres y cuando son iguales guardar en un archiv
llave pública de éste, para usarla al desencriptar

La función esteganografía_regreso (Anexo N), es la
proceso. Lo primero que hace es llamar a los tres vectores en
la imagen con el mensaje oculto, las medidas de la
desencriptada.

Empieza recorriendo los vectores y copiando los valore
temporal hasta encontrar el indicador de fin de vector, entonc
vector temporal son los correspondientes al primer bloc
hasta terminar.

Figura 2b. La hoja después de ser calentada revela un número telefónico.

El *micro punto*, usado por los alemanes desde 1941, es una forma de esteganografía en la que una página de texto se reduce fotográficamente a un tamaño muy pequeño, por ejemplo al tamaño del punto final de una carta inocente. El punto está a plena vista, pero es tan pequeño su contenido que evita la detección del mensaje a simple vista.

Otra forma antigua de esteganografía es el *cifrado nulo*, también conocido como *código abierto*. Con cifrados nulos, un mensaje aparentemente inocente contiene un mensaje oculto, el cual es revelado viendo al mensaje portador de cierta forma. Sirva de ejemplo el siguiente párrafo:

> *Se anda evadiendo Tito. Te entero amor, ha estado atesorando cartas anónimas aquí. Luego te escribo.*

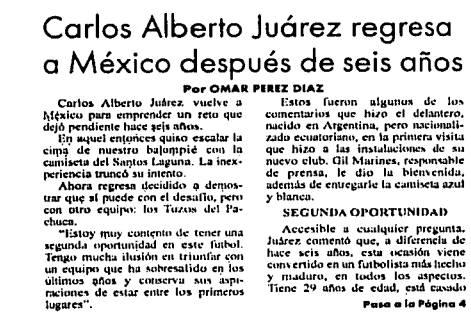
¿Puede el lector descubrir el mensaje oculto? Tome del párrafo anterior la segunda letra de cada palabra:

> *Se anda evadiendo Tito. Te entero amor, ha estado atesorando cartas anónimas aquí. Luego te escribo.*

Entonces, agregando el espaciado natural, se construye el siguiente mensaje:

> *Envíen más tanques.*

Otra técnica descrita por primera vez por los griegos, hace dos mil años, inserta un mensaje oculto en texto normal, marcando algunos caracteres selectos con un minúsculo hoyito, demasiado pequeño como para ser notado por el observador casual. El destinatario, sin embargo, examina de cerca la página y registra los caracteres marcados, agregando los espacios necesarios para dar forma a las palabras y sentido a las frases. Esta técnica se hizo muy popular entre la gente común en Inglaterra a mediados del siglo XIX. El costo de enviar correspondencia era altísimo, mientras que enviar periódico era gratuito; así que para evitar el cargo postal, la gente usaba agujas o astillas para marcar el periódico con sus mensajes y enviarlos gratuitamente a sus destinatarios. La Figura 3 muestra un ejemplo simulado de esta técnica.



Carlos Alberto Juárez regresa a México después de seis años

Por OMAR PEREZ DIAZ

Carlos Alberto Juárez vuelve a México para emprender un reto que dejó pendiente hace seis años.

En aquel entonces quiso escalar la cima de nuestro balompié con la camiseta del Santos Laguna. La inexperiencia truncó su intento.

Ahora regresa decidido a demostrar que sí puede con el desafío, pero con otro equipo: los Tuzos del Pachuca.

"Estoy muy contento de tener una segunda oportunidad en este futbol. Tengo mucha ilusión en triunfar con un equipo que ha sobresalido en los últimos años y conserva sus aspiraciones de estar entre los primeros lugares".

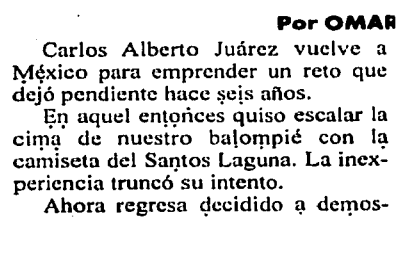
Estos fueron algunos de los comentarios que hizo el delantero, nacido en Argentina, pero nacionalizado ecuatoriano, en la primera visita que hizo a las instalaciones de su nuevo club. Gil Marines, responsable de prensa, le dio la bienvenida, además de entregarle la camiseta azul y blanca.

SEGUNDA OPORTUNIDAD

Accesible a cualquier pregunta, Juárez comentó que, a diferencia de hace seis años, esta ocasión viene convertido en un futbolista más hecho y maduro, en todos los aspectos. Tiene 29 años de edad, está casado

Pase a la Página 4

Figura 3a. Simulación de fragmento de periódico marcado



Por OMAR

Carlos Alberto Juárez vuelve a México para emprender un reto que dejó pendiente hace seis años.

En aquel entonces quiso escalar la cima de nuestro balompié con la camiseta del Santos Laguna. La inexperiencia truncó su intento.

Ahora regresa decidido a demos-

Figura 3b. Amplificación para observar las marcas que en conjunto dicen: "Me siento mal. Mándame medicina."

1.4 La esteganografía hoy en día

Para la mayoría de la gente en la actualidad, la esteganografía implica el ocultamiento de información en alguna forma de portadora digital. Las computadoras y la Internet han creado un rico entorno para implementar la esteganografía. Consecuentemente, en años recientes ha habido un interés creciente en las técnicas esteganográficas. Varios factores han contribuido a este crecimiento.

Las computadoras actuales y la Internet ciertamente proveen el medio ideal para el arte de la esteganografía. El número de bits de información que las computadoras manejan es enorme y sigue aumentando a pasos agigantados. Los tamaños de los archivos están creciendo, la capacidad de los discos y el número de archivos en una computadora están creciendo, y el número mundial de computadoras está creciendo también.

No solamente está creciendo el número de bits en uso, sino que también una gran proporción de esos bits pertenecen a ciertos tipos de archivos que proveen un amplio espacio para

ocultar información en su interior. Entre estos tipos los principales ciertamente son los archivos de imágenes, pero también son dignos de mencionar otros archivos multimedia como los archivos de audio y de video. Todos estos tipos de archivo eran raros hace apenas diez años, pero ahora son muy comunes y están constituyendo una enorme y creciente proporción del total de archivos compartidos y en uso.

Además, muchas de las computadoras en el mundo están conectadas entre sí a través de la Internet, por medio de la cual una enorme cantidad de información se está compartiendo. La Internet está desplazando rápidamente algunas formas tradicionales de comunicación proveyendo otras nuevas como: correo electrónico, video conferencia, navegación de sitios web, intercambio de archivos, conversación electrónica, etc. Cada forma de comunicación provee un intercambio de bits, en algunos casos una cantidad astronómica de bits, proveyendo amplio espacio de portadora para la comunicación esteganográfica.

Debido a la naturaleza común de estas formas de comunicación, el intercambio de las portadoras puede llevarse a cabo sin levantar ninguna sospecha particular por parte de terceros.

Otro aspecto es que las computadoras proporcionan un medio automatizado para realizar el ocultamiento y la recuperación de información. En muchos casos, esta operación puede ser transparente al usuario.

Encima de todo esto, la necesidad social de comunicación segura y legítima en nuestro mundo moderno y globalizado es ciertamente más grande que nunca. Las compañías necesitan comunicarse entre sí y con sus empleados; los abogados necesitan comunicarse con sus clientes; los bancos deben hacer transacciones a través de las redes y aún la gente común en su carácter de individuos tiene el derecho a la privacidad en su comunicación con otros.

1.5 Portadores digitales modernos para la esteganografía

Existen varias formas de realizar la esteganografía, pero ciertamente la más reconocida es cuando la incrustación de información se hace sobre un archivo individual que puede ser almacenado, copiado y transferido de una computadora a otra. Bajo esta amplia categoría, las imágenes digitales constituyen el tipo más común de portadora.

Las imágenes digitales tienen muchos bits pero aún más importante que esto, dependiendo del formato en particular, pueden tener un número muy grande de bits no significativos, que se traduce en una gran capacidad de almacenamiento esteganográfico.

Adicionalmente, las imágenes digitales están en todas partes, en contraste con diez años atrás. Ahora son parte estándar de los miles de millones de páginas web existentes y también pueden ser usadas y distribuidas de otras formas.

En el verano de 2002, Hioki Hirohisa propuso un esquema concatenado para almacenar mensajes esteganográficos de tamaño prácticamente ilimitado en la Internet[5]. La primer

porción del mensaje se almacena en una fotografía publicada en una página web. Esa misma fotografía contiene la URL[6] de la siguiente fotografía que contiene la segunda parte del mensaje y así sucesivamente, empleando tantas fotos publicadas como sea necesario. Vea la figura 4.

Figura 4a

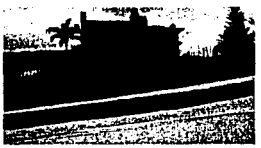


Figura 4b

Figura 4n

La figura 4a contiene la primer parte del mensaje más la URL donde está publicada la figura 4b. La figura 4b contiene la segunda parte del mensaje más la URL donde está publicada la figura siguiente. Así sucesivamente, hasta encontrar la figura 4n que contiene la última parte del mensaje y no más URLs. Esta concatenación se encuentra descrita en la figura 5.

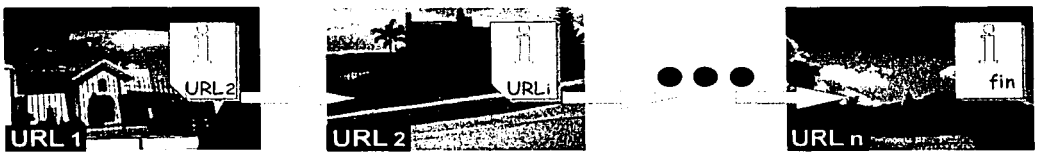


Figura 5. Descripción gráfica de la concatenación de mensajes en URLs.

Las técnicas para insertar información en imágenes digitales generalmente caen dentro de dos categorías: *dominio del espacio* y *dominio de la frecuencia*.

Una técnica dentro del dominio del espacio es insertar la información en los bits menos significativos de los píxeles de la imagen. Para aumentar la capacidad de incrustación esta técnica se puede extender a los bits superiores de cada píxel. Sin embargo, si se usan indiscriminadamente, algunas áreas planas de la imagen (en cuanto a color se refiere) se granularán conforme más información se incruste y se usen bits de más alto orden, es decir, más significativos (Ver Figura 6). Algunas técnicas muy recientes, como BPCS[7], tratan este problema limitando la incrustación de los bits superiores a las regiones más complejas de la imagen, por ejemplo, aquellas regiones con más alto nivel de ruido. No obstante, como se verá en el capítulo 5, aun esta técnica deja rastros evidentes de su incrustación, fallando así en el objetivo principal de la esteganografía.

Figura 6a. Imagen original

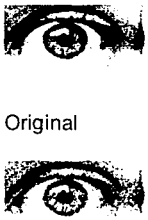


Figura 6b. Información incrustada en bits más significativos.

Adicionalmente a las técnicas en el dominio del espacio y en el dominio de la frecuencia, son posibles otros tipos de incrustación en imágenes, como usar el encabezado de la imagen y su paleta de color[8] como portadoras.

No obstante, las imágenes digitales son sólo un tipo de portadora basada en archivos para contener información esteganográfica. Otros archivos multimedia como los de audio y video también pueden contener espacio considerable donde incrustar información.

El ocultamiento de información en archivos de computadora no se limita a ocultar en los componentes ruidosos o redundantes de medios audiovisuales. El ocultamiento también puede llevarse a cabo en casi cualquier tipo de archivo, incluidos programas ejecutables, documentos, hojas de cálculo y bases de datos. Muchos de estos archivos almacenan información que es recuperada y presentada al usuario usando una aplicación asociada (por ejemplo un procesador de textos) y en muchos casos es posible incrustar la información de modo tal que sea ignorada por la aplicación asociada, pero que pueda ser extraída con una herramienta especial de software.

Además de ocultar información dentro del contenido de un archivo individual, también se pueden ocupar lugares que no son accedidos normalmente por el sistema regular de archivos. Por ejemplo, muchas computadoras almacenan los archivos en bloques y cada archivo ocupa un número entero de bloques. Esto significa que la mayoría de los archivos tienen una parte sin usar del último bloque asignado; ese espacio puede ser usado para incrustar información. Esta información normalmente es ignorada por los sistemas de manejo de archivos, por lo que deberá ser extraída mediante un software especial. La figura 7 muestra un ejemplo de este caso.

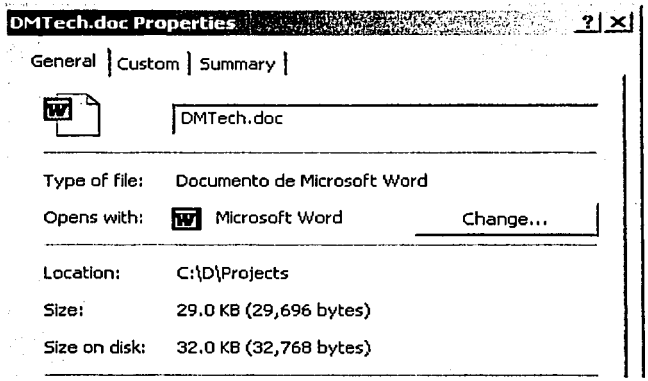


Figura 7. Diferencia entre tamaño real de archivo y espacio ocupado en disco. Este ejemplo es en el sistema operativo Windows XP.

Aunque el archivo en sí solamente ocupa 29,696 bytes, Windows XP ocupa 32,768 bytes para almacenarlo en el disco duro. La diferencia de 3,072 bytes podría ser utilizada por un programa especializado para ocultar información.

Se han diseñado algunas técnicas esteganográficas especialmente para la Internet. Por ejemplo, se puede ocultar información dentro de la porción desocupada de los encabezados de paquetes TCP/IP[9]. También se puede incrustar información en los documentos fuente de páginas web. Finalmente, otra forma interesante es usar mensajes estilo *SPAM* [10] para ocultar información.

1.6 Definición del problema

La seguridad en el envío de información digital vía correo electrónico es un tema sumamente estudiado hoy en día. Hay soluciones diversas al mismo problema, como son las firmas digitales y el encriptamiento de información mediante diversos métodos ampliamente conocidos. Estos esquemas proporcionan un método razonablemente seguro de enviar información privada vía correo electrónico.

No obstante, aunque tengamos la seguridad de que nuestra información no está siendo leída por un tercero, existe otra situación real que debe resolverse: cuando se prohíbe el envío de información encriptada en el entorno donde nos desenvolvemos. Ante esta problemática, surge la necesidad de desarrollar sistemas que aprovechen la naturaleza de la información no encriptada, tales como imágenes gráficas estándares, archivos estándares de audio, e inclusive secuencias normales de texto, para ser usados como portadoras de información privada sin levantar sospecha alguna, ya que su estructura y propiedades no serían alteradas. La esteganografía, definida como el arte y ciencia de ocultar la comunicación de información, es la solución apropiada a este problema.

El presente proyecto, denominado *DigiMarker*, incrusta cualquier tipo de información digital (documentos, hojas de cálculo, fotografías, e incluso archivos ejecutables) en un mapa de bits no comprimido con formato estándar. La imagen de mapa de bits conservará completamente su estructura original, de modo que no exista evidencia alguna que se está usando como portadora de información adicional. Visualmente tampoco aportará elementos de duda en cuanto a su contenido.

1.6.1 Problemas de un sistema encriptado de correo electrónico

Existen dos clases de sistemas criptográficos: *simétricos* y *asimétricos* (ver el capítulo 2).

Los sistemas de criptografía simétrica usan la misma llave para encriptar y desencriptar un mensaje. Los sistemas de criptografía asimétrica usan una llave (pública) para encriptar el mensaje y una llave distinta (privada) para desencriptarlo.

En un sistema simétrico, tanto el emisor como el receptor usan la misma llave. En este sistema, ambos deben acordar primero qué llave usarán para comunicarse. Tal acuerdo debe ser absolutamente secreto. A menudo se usa un canal secundario (teléfono, por ejemplo) para realizar esta comunicación. Sin embargo, estos canales secundarios suelen no ser seguros.

Un sistema asimétrico usa una llave pública y una llave privada. La llave pública está abierta a todo mundo y se usa para codificar el mensaje cuando el emisor envía un mensaje al receptor (dueño de la llave privada). Sin embargo, si la llave pública es violada, este sistema ya no funciona. Así que se debe garantizar la autenticidad de la llave pública. Por esta razón, este sistema requiere una organización especial de autenticación; debe ser una organización en la que todo mundo confíe. En realidad, este tipo de organización sólo puede existir si es sustentable comercialmente, por lo que este sistema tiene costos de mantenimiento.

Un grupo de investigación representado por Koichi Nozaki de la Universidad de Nagasaki desarrolló un sistema seguro de correo electrónico basado en esteganografía[11]. Este sistema no requiere una organización de autenticación, libera al usuario del manejo de llaves y es bastante seguro. Todas estas características superan los inconvenientes de un sistema encriptado de correo electrónico.

1.6.2 Esteganografía contra criptografía

Considerando que se desea comunicar información secreta, ¿qué beneficio tiene la esteganografía sobre la criptografía? De inmediato vienen dos ideas a la mente.

La primera es que en algunos casos es necesario prevenir que un tercero se entere de que se está realizando una comunicación privada. Si se supiera que se está llevando a cabo la comunicación, sería posible hacer uso de información adicional para intentar descubrir el contenido del mensaje o por lo menos parte de su significado. También sería posible destruir o alterar el mensaje, aún cuando no pudieran decodificar su contenido.

Una segunda ventaja es que los mensajes encriptados llaman la atención y de inmediato cobran interés ante un tercero con deseos de mirar lo ajeno. Los mensajes encriptados no tienen la apariencia de la información común, de modo que constituyen una invitación abierta para ser atacados por el criptoanálisis[12]. La historia ha demostrado vez tras vez que métodos de encriptamiento que se pensaban seguros, tarde o temprano fueron deshechos.

La esteganografía suele basarse en alguna clase de *portadora* de información. Esta portadora también se conoce como *contenedor*, *datos inútiles*, *recipiente*, etc. Durante los últimos años las imágenes han sido probablemente las portadoras de uso más común. Dentro de una imagen digital se puede insertar información secreta en forma tal que la imagen permanece sin alteración alguna ante la vista humana y quizás incluso sin cambiar su tamaño informático. El mensaje se puede recuperar con el software apropiado.

1.6.3 ¿Esteganografía o marcas de agua?

Las marcas de agua son un tema muy relacionado que también ha sido objeto de considerable interés recientemente. Mirándolo a la ligera, hay poca diferencia entre las marcas de agua y la esteganografía. Ambas técnicas buscan incrustar información dentro de otros datos, tales como una imagen digital, audio o video clips de forma que (usualmente) no produzca ningún cambio aparente en el contenedor (portadora). Sin embargo, el propósito, las técnicas y los resultados de cada uno son bastante diferentes.

Resulta instructivo comparar ambos temas. La diferencia fundamental entre marca de agua y esteganografía es el propósito para incrustar la información.

Para la marca de agua, el objetivo es típicamente proteger el *copyright* del contenedor, siendo éste en sí la información importante. La marca de agua que contiene provee una firma que identifica a la obra intelectual (fotografía, audio, video, etc.), su propietario, licencia de uso, autor, o cualquier otra información pertinente. Se dice que la información incrustada extiende o amplía la imagen, ya que le proporciona atributos adicionales.

Por el otro lado, en la esteganografía lo importante es la información incrustada y no el contenedor (aunque la portadora misma puede tener cierta importancia para algunos casos). Además, la información contenida en la esteganografía a menudo no tiene nada que ver con el significado del contenedor, mientras que con las marcas de agua suele ser lo contrario.

Como la marca de agua se usa para identificar los datos en los cuales se ha incrustado, en muchos casos se diseña a propósito de forma muy robusta, sacrificando usualmente el espacio disponible para incrustar. En imágenes digitales, por ejemplo, es deseable que la marca de agua permanezca inalterada aun después de que la portadora sea expuesta a transformaciones como filtrado, recortado, escalado, rotado o incluso compresión con pérdida[13].

Las técnicas de ocultamiento de datos para la comunicación secreta a menudo incrustan un mensaje que es más grande que el mensaje usado para la autenticación o para los derechos de autor. La técnica para la protección de derechos de autor suele llamarse *marca de agua robusta*. Incrusta una marca invisible que es muy difícil de remover del medio de almacenamiento. La remoción de la marca de agua dañaría definitivamente el medio. La técnica para la autenticación de datos se llama *marca de agua frágil*. Cualquier pequeño cambio en el medio se detecta fácilmente porque la marca de agua también se altera.

Por el otro lado, la esteganografía normalmente no requiere tanta fortaleza; de hecho a menudo es conveniente que su incrustación sea bastante frágil. Si el contenedor de datos es modificado de algún modo –incluso inocentemente- por un tercero, la información incrustada se destruye, anulando la posibilidad de ser descubierta y extraída posteriormente. Para extraer la información incrustada, se requiere un intento deliberado por parte de un usuario con conocimiento suficiente.

Debido a la naturaleza de las aplicaciones, la cantidad de información incrustada con marcas de agua suele ser bastante pequeña, mientras que la cantidad incrustada mediante esteganografía puede ser relativamente grande. Esto da a las técnicas esteganográficas bastante ventaja en aplicaciones que se pueden beneficiar de grandes capacidades de incrustamiento. Claro que si también se deseara fortaleza dentro de la esteganografía, se podría echar mano de alguna técnica de marca de agua, pero con la consecuente pérdida de capacidad de incrustación.

La información oculta por un sistema de marca de agua siempre está asociada al objeto digital que protege o a su autor, mientras que los sistemas esteganográficos ocultan cualquier tipo de información, a menudo independiente por completo del portador. También debe notarse que la comunicación esteganográfica usualmente es punto a punto (entre el emisor y el receptor) mientras que las técnicas de marca de agua suelen emplearse de uno a muchos.

La diferencia final es que la existencia misma de la marca de agua puede ser del conocimiento público (como la marca de agua en un billete de dinero, ver Figura 8) o, al menos, será fácil de detectar y de leer. En el ámbito de las marcas de agua, el término *seguridad* quiere decir que la marca de agua será muy difícil de eliminar o de reemplazar por otra. Con la esteganografía, sin embargo, el objetivo es que un tercero no se pueda dar cuenta de la existencia del mensaje, aun cuando usen software especial para analizar la imagen (análisis esteganográfico). Así que el término *seguridad* en el ámbito de la esteganografía quiere decir que el mensaje incrustado no se pueda detectar.

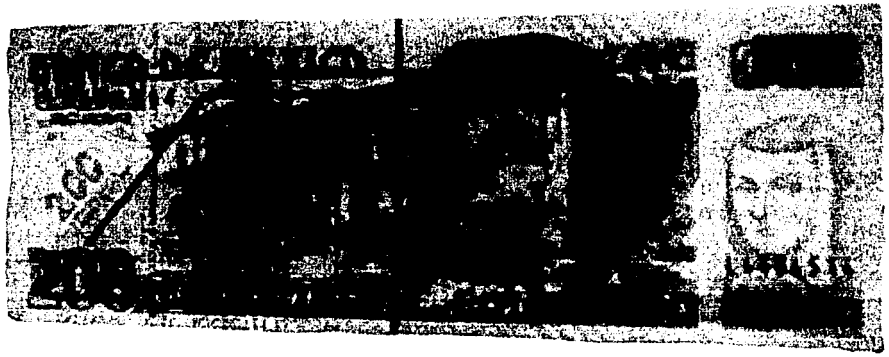


Figura 8. Marca de agua (lado derecho) en un billete mexicano.

Las comparaciones anteriores buscan esbozar las principales diferencias entre la esteganografía y las marcas de agua. Aunque muchas técnicas y aplicaciones caen íntegramente dentro de alguna de las dos, debe notarse también que existen aquellas que nublan la diferencia entre ambas.

1.7 Soluciones comerciales disponibles

El número de aplicaciones comerciales esteganográficas está creciendo rápidamente, evidencia del inmenso mercado que está esperando. Se puede encontrar un listado de las aplicaciones más importantes en:

http://www.cotse.com/tools/stega.htm

http://www.jjtc.com/Steganography/toolmatrix.htm

http://members.tripod.com/steganography/stego/software.html

Como dije anteriormente, algunas aplicaciones son 100% esteganográficas mientras que otras son de marca de agua y aún otras son más bien criptográficas o una combinación de todas las anteriores. En general, se incluyen aplicaciones que tienen que ver con la seguridad en la información.

Algunas de las aplicaciones esteganográficas comerciales más importantes tienen disponible una versión de demostración. En la tabla siguiente, resumo la revisión realizada de tales aplicaciones. Cabe aclarar que seleccioné sólo las aplicaciones que permiten usar como portadora a archivos tipo BMP, a fin de poder compararlas contra DigiMarker.

1.7.1 Evaluación de aplicaciones esteganográficas

Aplicación	Ayuda	Comprime	Encripta	Capacidad de Incrustación	Huella evidente
BlindSide	No	No	No	Muy baja	No
BPCS	en japonés	Si	No	Muy alta	A veces
Image Hide	No	No	No	Muy baja	No
In The Picture	Básica	Si	Si	Alta	Si
Steganos	Completa	Si	Si	Media	No
Third Eye	No	No	No	Muy Baja	No
BMP Secrets	**Completa**	**Si**	**Si**	**Regulable**	**No**
Courier	Si	No	No	Media	Si
DigiMarker	**Completa**	**Si**	**Si**	**Regulable**	**No**

De acuerdo con lo que me parecen criterios razonables de evaluación, observo que DigiMarker y "BMP Secrets" son las mejores implementaciones esteganográficas evaluadas.

1.8 Aplicaciones de la esteganografía

Las aplicaciones más obvias y tradicionales se relacionan con las comunicaciones secretas. La esteganografía no sólo provee un mecanismo para enviar información secreta sin que otros tengan acceso a lo enviado, sino que además oculta el hecho mismo del envío de información. Por ejemplo, una persona o compañía puede tener una página web con fotografías que contenga información secreta destinada a alguien más.

Cualquier persona en cualquier parte del mundo puede ver y bajar esas fotografías; por eso, cuando el destinatario accede a esa página no levanta sospecha alguna. Obviamente, requerirá un software especial, configurado con los parámetros correctos, para poder extraer la información privilegiada. El encriptamiento de la información incrustada proveería un grado adicional de seguridad. Este escenario es análogo a meter un objeto en una caja fuerte de alta seguridad y además esconder la caja fuerte en un sitio sumamente difícil de encontrar y que no levanta sospecha alguna.

Además de usar páginas web, los archivos contenedores de información oculta también pueden ser enviados mediante otras formas como correo electrónico, transferencias *FTP* [14], medios masivos de almacenamiento o a través de cualquier otro modo común. En contraste, el envío de información encriptada puede llamar considerablemente la atención de vigilantes o atacantes, ya que esa información es muy distinta de los mensajes ordinarios.

En algunas aplicaciones, los archivos que contienen la información oculta quizá no se envíen a ninguna parte, sino se queden en un sola computadora y sean accedidos más tarde por alguien que conozca la ubicación del archivo, los parámetros de incrustación y que tenga acceso al software de extracción. Una versión de esta aplicación es cuando alguien oculta información privada en su propia computadora y para su propio uso en una forma que otros ni siquiera descubran la existencia de la información. Aun si la información algún día fuera

encontrada, podría permanecer segura si no se tiene acceso al software de extracción y a la llave de desencriptamiento, en caso de que además hubiera sido encriptada.

En algunas aplicaciones, la presencia de información incrustada puede conocerse, pero sin el software y parámetros de extracción, la información incrustada es inseparable del contenedor. El contenedor puede ser usado como un objeto normal (por ejemplo, una imagen que puede ser vista como cualquier otra), pero su información oculta no podrá extraerse, revisarse o reemplazarse sin destruir la original y haciendo obvio que se ha hecho una sustitución.

Una variante de esto es un sistema de *tarjetas de identificación con circuito integrado* (CI). La memoria del CI contiene una imagen que empata con la imagen impresa en la tarjeta, pero contiene además información incrustada que identifica al titular de la tarjeta: nombre, dirección, estatura, huellas digitales, etc.[15]

La esteganografía digital, las marcas de agua y las huellas digitales son tres sub-disciplinas importantes del ocultamiento de información que han ganado importancia debido al frecuente uso (o mal uso) de medios digitales en computadoras, dispositivos y redes. La comunicación oculta, el control de acceso, la protección contra copia ilegal y la protección de derechos de autor han sido las áreas más populares de aplicación. Sin embargo, la lista de aplicaciones útiles también incluye, entre otras:

a) Autenticación de usuarios en redes, bancos, oficinas de alta seguridad, etc.
b) diseño de documentos infalsificables
c) monitoreo de actividades ilícitas en la Internet
d) incrustación de datos vitales en imágenes médicas
e) monitoreo automático de publicidad en los medios masivos (incluso medios no gráficos, como la radio)
f) privacidad en transacciones electrónicas
g) recuperación de información multimedia por degradación intencional o no intencional

Todas estas aplicaciones incluyen por lo menos dos componentes: los datos internos y los datos externos. Se pueden caracterizar por la importancia relativa de estos componentes. Para algunas aplicaciones, sólo los datos incrustados son importantes y la portadora es un simple contenedor de información. Para otros casos, la portadora es valiosa y los datos incrustados (externos) se usan para identificar al emisor, para proteger el contenedor (portadora) y para controlar el acceso a la misma. Existen también otras aplicaciones en las que tanto los datos internos como los externos juegan un papel importante. Cada una de estas aplicaciones tiene un conjunto de requerimientos que deben ser satisfechos por el proceso de ocultamiento de información.

El diseño de sistemas esteganográficos incluye uno o más de los siguientes elementos:

a) transparencia
b) fortaleza
c) seguridad
d) capacidad de almacenamiento

e) costo de implementación
f) costo de uso
g) posibilidad de uso en tiempo real

En los últimos meses, ha habido una fuerte demanda de aplicaciones avanzadas de seguridad debido a la ineficiencia de los métodos convencionales usados para proteger a un país y a su gente. La falsificación, el uso indebido y la duplicación indiscriminada de medios impresos y digitales han dado cabida a diversos tipos de amenazas, cuyo impacto se ha sentido recientemente en el mundo. Se han levantado algunos asuntos críticos que tienen que ver con terrorismo cibernético, biológico y químico y se han propuesto soluciones en toda la comunidad internacional. Afortunadamente, el creciente campo de la esteganografía tiene una variedad de soluciones que contribuirán a disminuir esas amenazas.

1.8.1 Imágenes con capacidad de corrección

La esteganografía también puede *auto incrustar* una imagen en sí misma, vea la Figura 9a. Esta técnica puede ayudar a reparar porciones de la imagen que hayan sido dañadas, alteradas, recortadas o sustituidas por otro contenido. La imagen puede dividirse en bloques de 8 x 8 píxeles que son procesados con la transformada del coseno[16], cuantificados e incrustados en los LSB de otros bloques de 8 x 8 situados relativamente lejos. Es posible obtener una muy buena imagen usando 2 LSB en el proceso de incrustación. El grado de reparación factible depende de la cantidad de información almacenada en los bloques redundantes. Si el interés es proteger áreas específicas de la imagen (como muy probablemente será, por ejemplo el detalle de la cara, porciones específicas de armamento, detalles distintivos de identificación, etc.), entonces es perfectamente posible incrustar esos detalles con un alto grado de resolución, de modo que la región original se pueda restaurar por completo (vea la Figura 9b). Por supuesto, la información incrustada es frágil y no es apropiada para técnicas de compresión con pérdida (ver el capítulo 3).

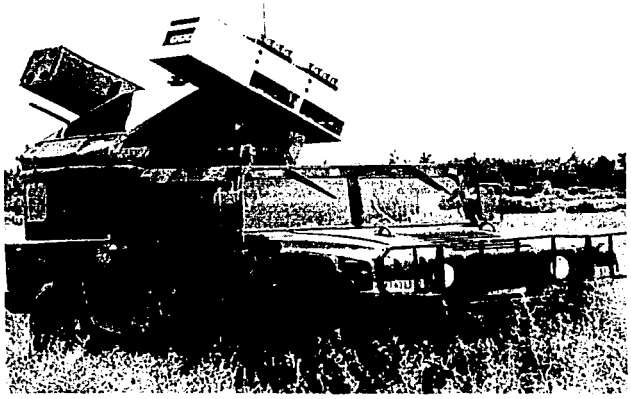


Figura 9a. Imagen original, señalando áreas que identifican la fotografía. Esas áreas deberán preservarse esteganográficamente.

Figura 9b. Áreas que se conservan a más alta resolución. Contienen el nombre de los oficiales y la placa del vehículo.

La Figura 10a muestra una ilustración del concepto: dentro o "detrás" de la fotografía original, se ocultan redundantemente los datos de interés para ser recuperados en caso de alteración de la fotografía. La Figura 10b muestra el panorama completo.

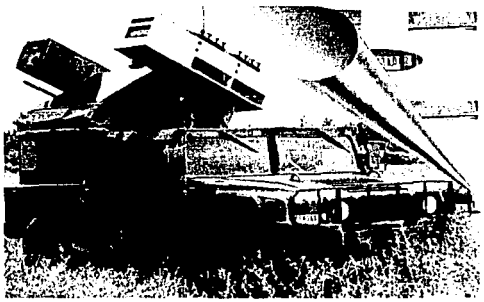


Figura 10a. Muestra parcial de la información redundante.

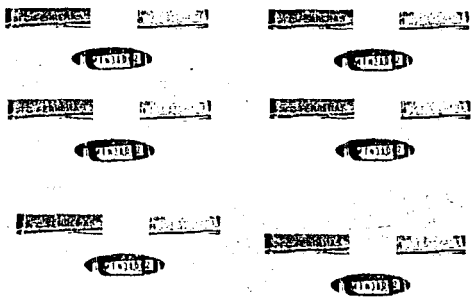


Figura 10b. Muestra total de la información redundante.

Se han propuesto diseños de técnicas para soportar la compresión JPEG pero hasta ahora no son suficientemente robustas para aplicaciones prácticas. En varias partes del mundo se está orientando la investigación en este sentido[17]. Así que se desea usar los datos libres (o redundantes) de una imagen para incorporarle algunas de las características de seguridad mencionadas anteriormente. La imagen tiene que ser dividida en regiones perceptivamente significativas e insignificativas. Las regiones insignificativas pueden ser sustituidas por la información relevante a la aplicación en particular.

1.8.2 Tarjetas plásticas inteligentes

Las *tarjetas plásticas* son ampliamente usadas hoy en día y en muchos casos nos facilitan la vida. Algunas de las aplicaciones más comunes son:

a) tarjetas de crédito y débito
b) acceso de personal a instalaciones
c) activación de servicios prepagados (telefonía, televisión, electricidad, etc.)
d) licencias de manejo
e) credenciales de elector
f) acceso a datos personales médicos (historial médico, etc.)

A menudo, las tarjetas plásticas contienen información personal, importante y confidencial que requiere altos niveles de seguridad. Normalmente, se les encuentra de dos tipos: con banda magnética y con microprocesador. Las de banda magnética son más baratas que otras tarjetas más sofisticadas con memoria y microprocesador, pero proveen un nivel de seguridad bajo que puede ser violado por un intruso profesional. Por el contrario, las tarjetas inteligentes (aquellas con microprocesador) pueden emplearse en aplicaciones actuales

satisfaciendo todos los requerimientos técnicos, incluyendo un nivel de seguridad relativamente alto. Pueden usar tecnologías universalmente aceptadas y probadas para encriptamiento, autenticación y no rechazo. Desafortunadamente, tecnologías de ingeniería de reversa y ataques invasores y no invasores han propiciado que casi todas las variantes de tarjetas hayan podido ser violadas.

Los atacantes frecuentemente apuntan hacia el algoritmo de encriptamiento y a las llaves almacenadas en el mecanismo de control de acceso de la tarjeta inteligente. Se ha observado que la gente involucrada en la copia y violación de tecnología ha tenido éxito. El diseñador no sólo debe ocuparse en proveer seguridad en la dirección usual sino también analizar el problema desde el punto de vista del intruso. Debe incluir mecanismos y usar múltiples tecnologías para hacer muy difícil las tareas de ingeniería de reversa, copiado y abuso.

Las tarjetas inteligentes se pueden hacer más seguras para poder usarse en aplicaciones críticas, empleando las siguientes ideas, extensión del trabajo realizado por Eiji Kawaguchi y su equipo al desarrollar la siguiente generación de tarjetas de crédito[18]:

a) Usar la esteganografía para almacenar datos personales o biométricos (huella digital, iris/retina, estampado de voz, etc.). La localidad exacta del almacenamiento de los bits secretos o de los bits usados para la autenticación deberán depender de una llave secreta.
b) La información confidencial no deberá estar disponible en su totalidad en una sola localidad en un solo medio en particular. Se deben emplear distintas tecnologías en distintos medios, tanto en el chip como en la tarjeta misma.
c) Incrustar un número de serie en el logotipo del emisor usando un proceso esteganográfico impreso.
d) Uso de parámetros biométricos para la generación de las llaves de encriptación y de esteganografía. De este modo, la llave no puede ser determinada ni siquiera extrayendo toda la información de la tarjeta.

La técnica esteganográfica llamada BPCS provee una amplia capacidad de almacenamiento y puede ser usada para incrustar datos biométricos, código activo y otra información secreta en una fotografía almacenada en el chip de la tarjeta.

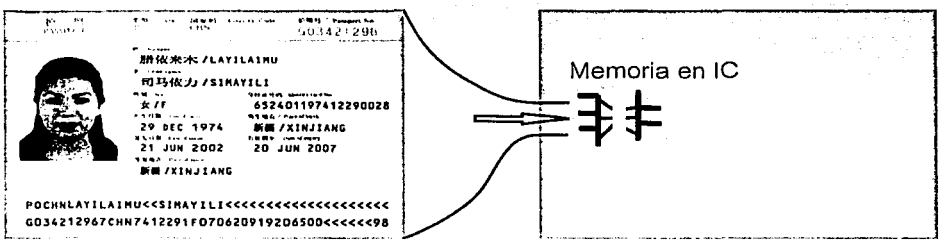


Figura 11. Las datos impresos en la parte frontal de la Tarjeta Pasaporte, están grabados esteganográficamente en el *chip* del reverso.

Habiendo descrito el panorama general de la esteganografía, en el siguiente capítulo trataré el tema particular de la criptografía. La criptografía es relevante en esta tesis pues provee un nivel adicional de seguridad al sistema propuesto.

---

[1] Método de codificación de longitud variable para cada símbolo basado en frecuencias de ocurrencia.
[2] Del inglés *Least Significant Bits*
[3] Poco a poco, la literatura sobre esteganografía ha aumentado. En esta dirección se pueden encontrar los títulos más famosos: *http://www.jjtc.com/Steganography/*
[4] Singh, Simon. *The Code Book*, Doubleday, 1999, p.18.
[5] Hirohisa, Hioki. "A Data Embedding Method using BPCS principle with new Complexity Measures", *Proceedings of STEG'02*, Japan, 2002, pp. 30-47.
[6] Del inglés *Uniform Resource Locator* (antes *Universal Resource Locator)*. Se refiere a la dirección en Internet donde se localiza el recurso.
[7] Del inglés *Bit Plane Complexity Segmentation*. Técnica desarrollada por Eiji Kawaguchi.
[8] Algunos formatos gráficos usan una "paleta de color". Se refiere a una tabla con colores indexados. La imagen no tendrá los valores de los colores sino apuntadores a la paleta de color. Normalmente la paleta de color cuando mucho tiene 256 colores disponibles.
[9] Del inglés *Transmission Control Protocol/Internet Protocol*. Se refiere al conjunto de protocolos de comunicaciones usados para conectar anfitriones ("hosts") en la Internet.
[10] El envío indiscriminado de copias de correos electrónicos a mucha gente. Frecuentemente esos mensajes son anuncios comerciales, no solicitados por los destinatarios. Se le considera una práctica de mal gusto y que debe evitarse.
[11] Nozaki, Koichi. "A Model of Anonymous Covert Internet Mailing System using Steganography", *Proceedings of STEG'02*, Japan, 2002, pp. 7-10.
[12] Criptoanálisis: el estudio formal para violar los sistemas criptográficos.
[13] Katzenbeisser, Stefan y Fabien Petitcolas. *Information Hiding Techniques for Steganography and Digital Watermarking*, Artech House, 2000, pp. 106-107.
[14] Del inglés *File Transfer Protocol*.
[15] Prototipo presentado por Sumei Guo, Takuya Ooshige y otros en el *Pacific Rim Workshop on Digital Steganography 2002*, Kitakyushu, Japón.
[16] La transformada discreta del coseno (DCT) transforma una señal o imagen del dominio espacial al dominio de la frecuencia. En otras palabras, separa la imagen en sub bandas espectrales de distinta importancia con respecto a la calidad visual de la imagen. Útil para expresar una forma de onda como una suma pesada (multiplicada por una serie de coeficientes) de cosenos. La DCT es el corazón de muchos algoritmos de procesamiento de señales y de compresión de imágenes, como el formato JPG.
[17] Saibal, Pal y Saxena Pramod. "Smart Steganographic Applications". *Proceedings of STEG'02*, Japan, 2002, p. 16.
[18] Kawaguchi, Eiji y Richard Eason. "Principles and Applications of BPCS-Steganography", *SPIE's International Symposium on Voice, Video and Data Communication*, 1998.

2

Métodos de encriptamiento

"Confusión es una palabra que hemos inventado para describir el orden que no hemos explicado"
Henry Miller

2.1 Descripción general

Encriptar es transformar un mensaje con sentido en un conjunto de datos sin significado aparente. La transformación ocurrida puede ser muy variada, pero lo importante es que siempre sea posible la recuperación íntegra del mensaje original. A este proceso inverso se le conoce como *desencriptamiento*.

Los sistemas criptográficos suelen catalogarse bajo tres criterios independientes [1], dependiendo de:

1. el tipo de transformación que usen,
2. el número de llaves empleadas y
3. la forma en que se procesa el mensaje.

Tipo de transformación. Generalmente, existen dos tipos de transformación: *sustitución* y *transposición*. La sustitución se refiere a cambiar un elemento del mensaje original, por ejemplo una letra "a", en otro elemento sin relación aparente con el elemento original. Por ejemplo se puede sustituir la "a" por un "&". La transposición se refiere a cambiar la posición de un elemento dentro del mensaje original a otra posición sin relación aparente. Los sistemas antiguos de criptografía solían basarse en una sola sustitución y/o transposición. Actualmente un sistema criptográfico seguro suele involucrar varias etapas tanto de sustitución como de transposición. Independientemente de la complejidad de la transformación empleada, debe garantizarse la existencia de la transformada inversa –desencriptamiento- que recupera íntegramente el mensaje original.

Sea el siguiente mensaje original:

En el principio creó Dios los cielos

Para fines didácticos ignoremos acentuación y letras mayúsculas. Ahora apliquemos una sustitución bajo las siguientes reglas: 1) las vocales serán sustituidas por la consonante que le siga en el alfabeto. Es decir, la "a" se reemplaza por la "b", la "e" se reemplaza por la "f", etc.

2) Las consonantes serán sustituidas por la segunda letra posterior en el alfabeto. Por ejemplo, la "n" se sustituye por la "p", la "p" por la "r", etc. 3) Los espacios en blanco se sustituyen por los siguientes caracteres, tomando uno por uno en lista circular: (#,&,%,?,/,*). El mensaje después de aplicada la sustitución queda así:

*fp#fn&rtjpejrjq%etfq?fjqu/nqu*ejfnqu*

A simple vista, parece que ha sido un buen encriptamiento. No obstante, tiene un gran problema: el desencriptamiento produce ambigüedades, ya que, por ejemplo, tanto la "e" como la "d" producen la misma salida: "f". El desencriptador no sabrá como decodificar una "f", si como "e" o como "d". Modifiquemos entonces las reglas a algo más seguro: 1) Que todas las letras se sustituyan por la siguiente tercera letra en el alfabeto, en lista circular. 2) Los espacios en blanco se sustituyen por los siguientes caracteres, tomando uno por uno en lista circular: (#,&,%,?,/,*). El mensaje después de aplicada la sustitución queda así:

*hq#ho&sulqflslr%fuhr?glrv/orv*flhorv*

Ahora apliquemos la siguiente transposición: recorrer a la derecha en lista circular 3 posiciones. Si al mensaje original aplicamos esta transposición, resulta:

losEn el principio creo Dios los cie

Evidentemente no parece un proceso suficiente para ocultar le mensaje original, así que mejor apliquemos la transposición al mensaje que ya había sufrido la sustitución:

*orvhq#ho&sulqflslr%fuhr?glrv/orv*flh*

Apliquemos una transposición más de modo que rompamos definitivamente la estructura de las palabras del mensaje original: tomemos subgrupos de 3 elementos e intercambiemos el primero con el tercero. Ejemplo: Si dice "orv", debe quedar "vro". Aplicando esta segunda transposición, el mensaje queda:

*vro#qh&ohluslfqrlsuf%?rhrlgo/v*vrhlf*

Se sugiere como ejercicio para el lector diseñar una forma automatizada que descubra el mensaje original a partir de este último mensaje encriptado, suponiendo que se desconocen las sustituciones y transposiciones aplicadas. El lector inexperto descubrirá que aun un método tan simple como el ejemplificado, por el hecho de combinar sustituciones y transposiciones, no será obvio de resolver.

Número de llaves empleadas. Aquí existen dos categorías de sistemas criptográficos: *simétricos* y *asimétricos*. Los sistemas simétricos son aquellos en los que se emplea una sola llave en ambos extremos de la comunicación, es decir, tanto el que encripta el mensaje como quien lo desencripta comparten la misma llave. Esos sistemas también suelen llamarse de llave secreta o de llave única. Por el contrario, los sistemas asimétricos requieren que quien

envía y quien lo recibe usen llaves distintas. Los sistemas asimétricos también se conocen como de doble llave o de llave pública.

¿Qué es esto de la llave? En términos simples se trata de una secuencia de elementos que de algún modo indican parámetros para llevar a cabo el proceso de encriptamiento y/o desencriptamiento. Para ilustrar un uso específico de llave, revisemos el método de encriptamiento denominado *Vigenere*.

Sean el siguiente mensaje y la siguiente llave (por simplicidad se omiten espacios):

mensaje: odiolacomidarapida
llave: usareemosestallave

Como se podrá apreciar, este método requiere que la llave sea de la misma longitud que el mensaje. Para fines prácticos esto no representa problema alguno pues la llave se puede componer de repeticiones de una llave más pequeña, por ejemplo:

mensaje: odiolacomidarapida
llave: llavellavellavella

Para realizar el encriptamiento se usa una tabla de sustitución, llamada tabla de *Vigenere*:

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*q*	*r*	*s*	*t*	*u*	*v*	*w*	*x*	*y*	*z*
a	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
c	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
d	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
e	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
f	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
g	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
h	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
i	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
j	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
k	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
l	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
m	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
n	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
o	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
p	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
r	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
s	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
t	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
u	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
v	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
w	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
x	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figura 12. Tabla de *Vigenere*, resaltando el encriptamiento del mensaje "o" con la llave "u".

Para encriptar un mensaje, se toma un elemento del mensaje y se selecciona la columna que le corresponda. Se toma un elemento de la llave y se selecciona el renglón que le corresponda.

La intersección de esa columna/renglón (mensaje/llave) da el carácter codificado. Para nuestro ejemplo (vea la Figura 12 para ver cómo se encriptó el primer elemento):

mensaje:	odiolacomidarapida
llave:	usareemosestallave
encriptado:	IVIFPEOCEMVTRLAIYE

El proceso de desencriptado es igual de simple: para cada elemento de la llave, se selecciona el renglón que le corresponde. En ese renglón se selecciona la columna a la que corresponde el elemento del mensaje cifrado. El título de la columna seleccionada da el elemento desencriptado. Ejemplo para el primer carácter: el primer elemento de la llave es "u", por lo que se selecciona el renglón "u". El primer elemento del mensaje encriptado es "I" por lo que dentro del renglón "u" se ubica la columna que tenga una "I". Esa columna resulta ser la columna titulada "o", por lo que el primer elemento del mensaje desencriptado es precisamente "o".

Fácilmente se observa que al cambiar la llave el mensaje encriptado será distinto. Y aun si usamos una repetición de caracteres, el mapeo no será evidente. Encriptemos con los siguientes datos:

mensaje:	odiolacomidarapida
llave:	llavellavellavella
encriptado:	ZOIJPLNOHMOLRVTTOA

El método *Vigenere* es un sistema simétrico puesto que una sola llave es empleada tanto para encriptar como para desencriptar.

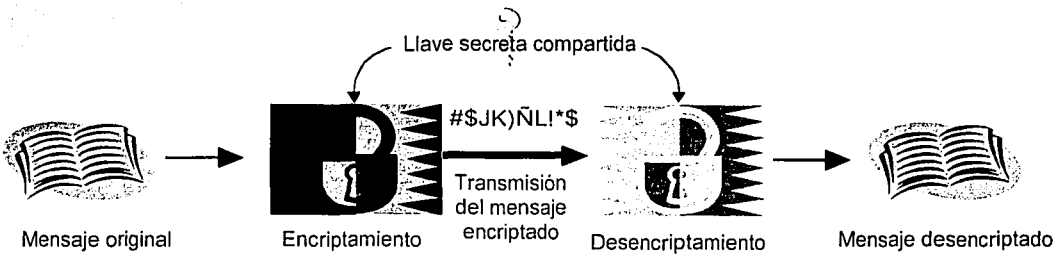


Figura 13. Modelo de encriptamiento simétrico.

Es momento oportuno para distinguir la diferencia entre el primer ejemplo que dimos donde encriptamos con sustitución/transposición y el segundo ejemplo donde usamos una llave mediante el método *Vigenere*. En el primer caso se puede desencriptar el mensaje si se conoce el procedimiento empleado en la encriptación. En el segundo caso no basta con conocer el método empleado, ya que es indispensable conocer la llave para poder desencriptar. Aunque se conozca plenamente el método *Viginere*, no será posible descifrar el

mensaje si no se tiene acceso a la llave empleada. Esta característica se conoce como el principio de Kerckhoffs[2], que dice:

> *Se asume que el oponente conoce nuestro método de encriptamiento, de modo que la seguridad radica exclusivamente en la selección secreta de la llave.*

Forma de procesar el mensaje. Las formas posibles son *en bloque* o *continua*. El proceso en bloque se refiere a tomar un número fijo de elementos del mensaje original y encriptarlos; luego se toma el siguiente bloque y así sucesivamente hasta terminar con todo el mensaje. Por su parte, el proceso continuo toma uno por uno de los elementos del mensaje original y va produciendo la salida encriptada también uno por uno.

El método *Vigenere* ilustrado anteriormente es un ejemplo de proceso continuo, ya que puede encriptar elemento por elemento.

Ahora presentaremos una introducción a una variante simplificada del método DES[3] para ilustrar un proceso en bloques.

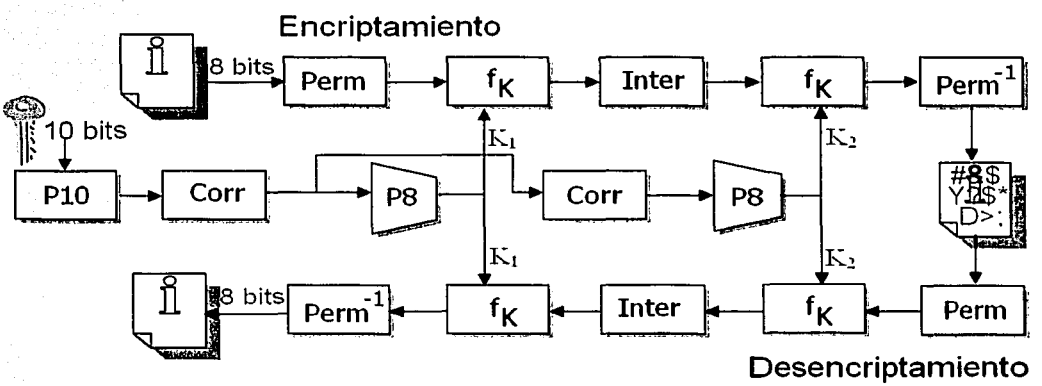


Figura 14. Diagrama de método DES simplificado.

El algoritmo simplificado DES (S-DES) toma como entrada un bloque de 8 bits del mensaje original y una llave de 10 bits, para producir como salida un bloque encriptado de 8 bits. En la figura 14, la hoja blanca que tiene una "i" representa el mensaje original; la hoja blanca con caracteres sin sentido representa el mensaje encriptado.

El algoritmo S-DES incluye cinco funciones: Una permutación inicial (**Perm**), una función compleja ($\mathbf{f_K}$) que incluye sustitución y transposición y depende de una llave K, una permutación simple (**Inter**) que intercambia las dos mitades del mensaje, nuevamente la función $\mathbf{f_K}$ y finalmente una permutación ($\mathbf{Perm^{-1}}$) que es la inversa de la permutación inicial **Perm**. Tal como mencioné anteriormente, el uso de varias etapas de transposición

(permutación) y sustitución da como resultado un algoritmo más complejo que dificulta su quebrantamiento.

La función $\mathbf{f_K}$ toma como entrada no sólo al mensaje sino también 8 bits de la llave. La llave se somete primero a una permutación (**P10**), luego a un corrimiento (**Corr**) y finalmente a una permutación (**P8**) que da como salida 8 bits, produciendo la primer subllave llamada K_1. La salida del corrimiento también alimenta a otro corrimiento y a otra instancia de **P8** para producir la segunda subllave, K_2.

El algoritmo de encriptamiento puede expresarse así:

$$\text{mensaje encriptado} = \text{Perm}^{-1}(f_{K2}(\text{Inter}(f_{K1}(\text{Perm}(\text{mensaje original})))))$$

donde

$$K1 = P8(\text{Corr}(P10(\text{llave})))$$

$$K2 = P8(\text{Corr}(\text{Corr}(P10(\text{llave}))))$$

El algoritmo de desencriptamiento es básicamente el proceso inverso:

$$\text{mensaje original} = \text{Perm}^{-1}(f_{K1}(\text{Inter}(f_{K2}(\text{Perm}(\text{mensaje encriptado})))))$$

En la figura 15 se muestra cómo se generan dos subllaves de 8 bits cada una a partir de una llave original de 10 bits.

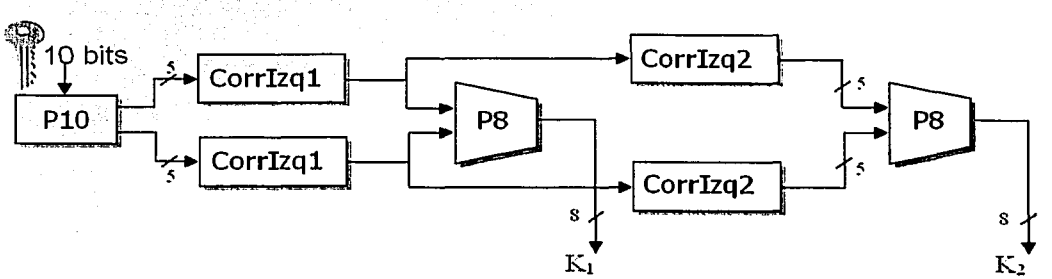


Figura 15. Generación de la llave para S-DES.

La primer permutación, **P10**, se define así:

$$P10(k_1,k_2,k_3,k_4,k_5,k_6,k_7,k_8,k_9,k_{10}) = (k_3,k_5,k_2,k_7,k_4,k_{10},k_1,k_9,k_8,k_6)$$

que también se puede expresar de la siguiente forma:

P10									
3	5	2	7	4	10	1	9	8	6

Esta tabla se lee de izquierda a derecha. Cada posición en la tabla identifica el bit de entrada que produce el bit de salida en esa posición, es decir, el primer bit de salida es el bit número 3 de la entrada, el segundo bit de salida es el bit número 5 de la entrada, etc. Por ejemplo, si aplicamos la llave (1010000010) a P10, obtenemos (1000001100).

Luego se realiza un corrimiento circular a la izquierda (**CorrIzq1**) por separado para cada 5 bits. Siguiendo el ejemplo citado, el resultado sería (00001 11000). Enseguida se aplica **P8**, que selecciona y permuta 8 de los 10 bits de acuerdo a la siguiente tabla:

P8									
6	3	7	4	8	5	10	9		

Como resultado de **P8** se obtiene la subllave K1. En nuestro ejemplo K1 valdría (10100100). Ahora regresamos al par de cadenas de 5 bits que salieron de **CorrIzq1** y realizamos una corrimiento circular a la izquierda de 2 bits para cada cadena. Para nuestro ejemplo, el valor (00001 11000) se convierte en (00100 00011). Finalmente, se aplica nuevamente **P8** para producir K2. En nuestro caso el valor obtenido para K2 = (01000011).

Regresemos a la figura 14. Ya hemos explicado cómo obtener K1 y K2, pero todavía nos faltan por desarrollar los procesos **Perm**, **Perm**$^{-1}$, $\mathbf{f_K}$ e **Inter**. A continuación se detalla cada uno. La entrada del algoritmo es un bloque de 8 bits, al cual se le aplica la permutación **Perm**:

Perm							
2	6	3	1	4	8	5	7

Perm retiene los 8 bits y simplemente los transpone. Al final del algoritmo se necesitará la permutación inversa **Perm**$^{-1}$:

Perm^{-1}							
4	1	3	5	7	2	8	6

La función $\mathbf{f_K}$ consiste de una combinación de permutaciones y sustituciones. Sean *I* y *D* los 4 bits más y menos significativos respectivamente de la entrada de 8 bits a $\mathbf{f_K}$. Sea F un mapeo de cadenas de 4 bits a cadenas de 4 bits (no necesariamente uno a uno). Se define la función así:

$$f_K(I, D) = (I \oplus F(D, SK), D)$$

donde *SK* es una subllave y $\oplus$ es la función OR-exclusivo bit por bit. Por ejemplo, supóngase que la salida de **Perm** en la figura 14 es (10111101) y que F(1101, *SK*) = (1110) para una *SK* dada. Entonces f_K (10111101) = (01011101) porque (1011) $\oplus$ (1110) = (0101).

Ahora describiremos la función de mapeo F (figura 16). La entrada es un número de 4 bits ($n_1\ n_2\ n_3\ n_4$). La primera operación es una expansión/permutación:

E/P							
4	1	2	3	2	3	4	1

Para facilitar la comprensión del procedimiento, representemos la salida de E/P de la siguiente forma:

n_4	n_1	n_2	n_3
n_2	n_3	n_4	n_1

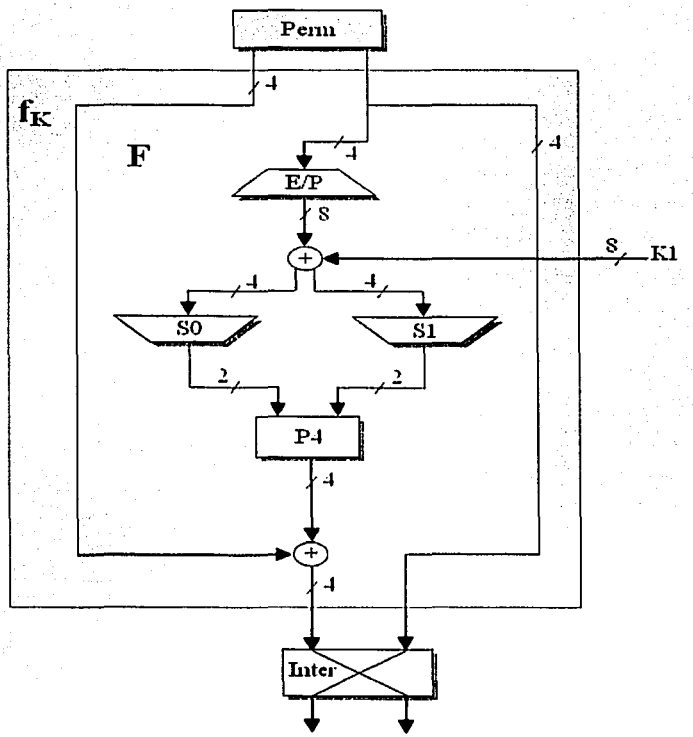


Figura 16. Diagrama de la función de mapeo F.

Sumar la salida de E/P con la subllave de 8 bits K1 = (k_{11},k_{12},k_{13},k_{14},k_{15},k_{16},k_{17},k_{18}) mediante OR-exclusivo:

$$\begin{array}{c|cc|c} n_4 + k_{11} & n_1 + k_{12} & n_2 + k_{13} & n_3 + k_{14} \\ n_2 + k_{15} & n_3 + k_{16} & n_4 + k_{17} & n_1 + k_{18} \end{array}$$

Ahora renombremos estos 8 bits de la siguiente forma:

$$\begin{array}{c|cc|c} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \end{array}$$

Los primeros 4 bits (primer renglón de la matriz anterior) se meten en el selector S0 para dar como salida 2 bits. Los restantes 4 bits (segundo renglón) se meten en el selector S1 para producir otros 2 bits. Las cajas selectoras S0 y S1 se definen así:

S0 =

	0	**1**	**2**	**3**
0	1	0	3	2
1	3	2	1	0
2	0	2	1	3
3	3	1	3	2

S1 =

	0	**1**	**2**	**3**
0	0	1	2	3
1	2	0	1	3
2	3	0	1	0
3	2	1	0	3

El primero y cuarto bits se toman como un número de 2 bits que especifica el renglón de la caja selectora. El segundo y tercer bits especifican la columna de la caja S. El número encontrado en la intersección de ese renglón/columna, en binario, es la salida de 2 bits. Por ejemplo, si ($p_{0,0}$ $p_{0,3}$) = (0 0) y si ($p_{0,1}$ $p_{0,2}$) = (10), entonces la salida sería el contenido de la intersección del renglón 0 con la columna 2 en la caja S0, el cual es un 3 que en binario sería (11). De igual forma ($p_{1,0}$ $p_{1,3}$) y ($p_{1,1}$ $p_{1,2}$) se usan como índices para encontrar el renglón/columna de S1 y producir así los otros dos bits de salida.

Finalmente los cuatro bits producidos en conjunto por S0 y S1 se permutan de acuerdo a P4:

P4			
2	4	3	1

La salida de P4 es la salida de la función F.

2.2 Principios y fundamentos

Ahora que ha quedado clara la diferencia entre los procesos continuo y por bloques y que se han dado ejemplos de ambos encriptadores, es posible examinar con más cuidado los fundamentos teóricos de los principios modernos de criptografía.

Virtualmente todos los algoritmos de encriptamiento por bloque están basados en una estructura conocida como el *encriptador de Feistel*. Como dije antes, el proceso continuo encripta un flujo de datos tomando elemento por elemento. Por el contrario, un proceso en bloque considera un grupo de elementos como una unidad y lo procesa para producir un bloque encriptado que tiene el mismo número de elementos. Típicamente los bloques son de 64 bits. El método S-DES ilustrado anteriormente ocupa bloques de solamente 8 bits pues fue diseñado para fines académicos.

El encriptador en bloque opera sobre un bloque de n bits de texto llano y produce un bloque encriptado de n bits. Existen 2^n posibles diferentes bloques de texto llano y, para que el encriptamiento sea reversible (es decir, que sea factible el desencriptamiento sin ambigüedad), cada uno debe producir un bloque encriptado único.

Feistel propuso ejecutar en secuencia dos o más encriptadores sencillos de forma tal que el resultado final sea criptológicamente más fuerte que cualquiera de los encriptadores utilizados. En particular, propuso el uso de un encriptador que alterna sustituciones y transposiciones (o permutaciones). Cabe mencionar que la estructura de Feistel que tiene más de 25 años de haber sido propuesta, es la que se usa prácticamente en todos los encriptadores simétricos de bloque vigentes hoy en día.

La figura 17 presenta la estructura propuesta por Feistel. Las entradas para el algoritmo de encriptamiento son un bloque de $2w$ bits y una llave K. El texto llano se divide en dos mitades, I_0 y D_0. Las dos mitades de los datos pasan a través de n rondas de procesamiento y luego se combinan para producir el bloque encriptado. Cada ronda i tiene las entradas I_{i-1} y D_{i-1}, derivadas de la ronda anterior, así como la subllave K_i, derivada de la llave completa K. Las subllaves K_i son diferentes de K y también son diferentes entre sí.

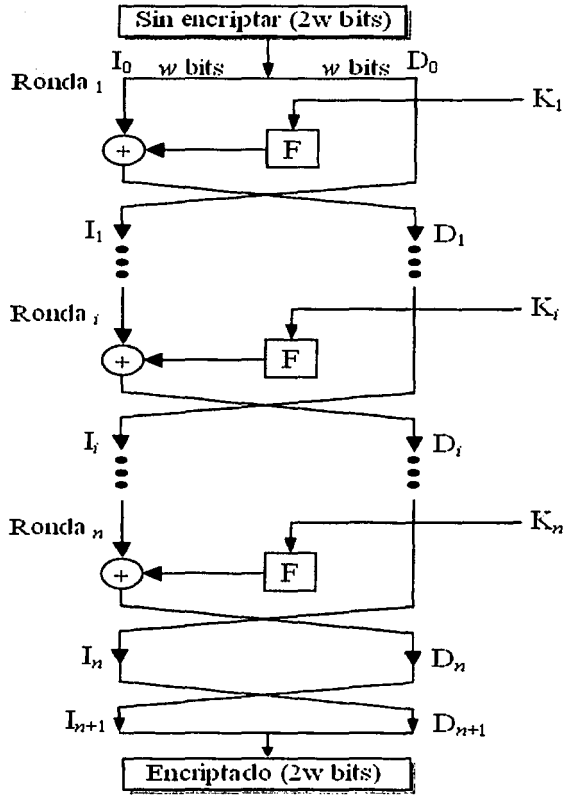


Figura 17. Diagrama de encriptamiento de Feistel

2.2.1 ¿Cómo se analiza la seguridad de un sistema de encriptamiento?

Los términos *difusión* y *confusión* fueron introducidos por Shannon para definir los dos constructores básicos de cualquier sistema criptográfico. La preocupación de Shannon era sobrevivir al ataque mediante criptoanálisis[4] basado en análisis estadístico. El contexto es el siguiente: asuma que el atacante tiene cierto conocimiento de las características estadísticas del mensaje original. Por ejemplo, en un texto llano en algún idioma determinado, normalmente se conoce la distribución de frecuencias de cada letra. También se puede predecir la aparición de determinadas palabras e incluso frases. Si estas estadísticas se reflejan de algún modo en el texto encriptado, el criptoanalista puede deducir la llave de encriptamiento, parte de la llave, o por lo menos un conjunto de llaves con alta probabilidad

de contener la llave exacta. En el encriptador ideal, en palabras de Shannon, todas las estadísticas del texto encriptado son independientes de la llave empleada.

En la *difusión*, la estructura estadística del texto llano se disipa dentro del texto encriptado. Esto se logra haciendo que cada dígito de entrada afecte el valor de muchos dígitos de salida (encriptados), lo que equivale a decir que cada dígito encriptado es afectado por muchos de los dígitos del texto original. En un encriptador binario de bloque, la *difusión* se logra aplicando repetidamente una permutación y luego una función sobre esa permutación; el efecto es que bits de diferentes posiciones en el texto original contribuyen a un solo bit del texto encriptado.

Cada bloque encriptador incluye una transformación de un bloque de texto llano en un bloque encriptado, donde la transformación depende de una llave. El mecanismo de *difusión* busca lograr que la relación estadística entre el texto llano y el encriptado sea lo más compleja posible a fin de frustrar cualquier intento de deducir la llave.

Por otro lado, la *confusión* procura que la relación entre la estadística del texto encriptado y el valor de la llave de encriptamiento sea muy compleja, con el mismo propósito de evitar la deducción de la llave. Esto se logra usando algoritmos complejos de sustitución. En contraste, una función de sustitución lineal añadiría muy poca *confusión*.

Si no se tienen suficientes elementos estadísticos para discriminar buena parte del universo de llaves posibles, la opción a seguir puede ser el uso de fuerza bruta para encontrar la posible llave solución (asumiendo de todos modos que se sabe qué se quiere encontrar o por lo menos la naturaleza de la información). Por esta razón, los encriptadores usan algoritmos con llaves suficientemente grandes, a fin de hacer impráctico el ataque por fuerza bruta. Por ejemplo, el algoritmo DES original usa una llave de 56 bits, dando un universo de llaves de 2^{56}. Pero ¿es 2^{56} llaves un número suficientemente grande?

Es buen momento para introducir entonces dos términos más: se dice que un encriptador es ***incondicionalmente seguro*** si el código encriptado que genera no contiene suficiente información para determinar sin ambigüedad el mensaje original, sin importar cuánto código encriptado se tenga disponible. En otras palabras, no importa cuánto tiempo de procesamiento se tenga disponible, será imposible descifrar el código simplemente porque la información no está allí. Con excepción de un algoritmo conocido como "cubierta de una pasada"[5], no existe algoritmo que sea incondicionalmente seguro. Así que los diseñadores de encriptadores se deben conformar con procesos que cumplan uno o ambos de los siguientes criterios:

- El costo de violar el código encriptado es bastante mayor que el valor de la información encriptada.
- El tiempo requerido para violar el código encriptado excede la vida útil de la información encriptada.

Se dice que un encriptador es ***computacionalmente seguro*** si satisface los dos criterios anteriores. El obstáculo es que es muy difícil estimar la cantidad de esfuerzo necesario para

violar con éxito un código encriptado. Consideremos por un momento el uso de fuerza bruta, que simplemente intenta llave por llave hasta encontrar una que produzca un desencriptamiento exitoso. En promedio se puede esperar recorrer la mitad del universo posible para encontrar la llave buscada.

Tamaño llave (bits)	Llaves posibles	Tiempo requerido a razón de: 1 encriptamiento/μs	10^6 encriptamiento/μs
32	2^{32}	2^{31} μs = 35.8 minutos	2.15 milisegundos
56	2^{56}	2^{55} μs = 1,142 años	10.01 horas
128	2^{128}	2^{127} μs = 5.24×10^{24} años	5.4×10^{18} años

La tabla anterior muestra cuánto tiempo se requiere para buscar en diferentes universos de llaves. El poder de cómputo de 1 encriptamiento por μs parece muy razonable para las computadoras actuales. No obstante, con la posibilidad de usar procesamiento paralelo masivo, se ha considerado también la extraordinaria velocidad de un millón de encriptamientos por microsegundo. Como se puede apreciar, DES (que originalmente usa una llave de 56 bits) podría no ser suficientemente seguro ante tal fuerza bruta de cómputo, pues en un promedio de 10 horas de búsqueda la llave podría ser descubierta. Con esto contestamos parcialmente la pregunta planteada antes acerca del nivel de seguridad que DES ofrece. Pero ahora las preguntas serían: ¿existe una red de cómputo capaz de proveer la asombrosa potencia de un millón de encriptamientos por microsegundo? ¿cuánto costaría esta infraestructura? ¿nuestra información a encriptar con DES le interesaría tanto a algún gobierno u organización como para que merezca la pena semejante esfuerzo?

La compañía RSA Security[6], especialista en seguridad de la información, de vez en cuando organiza concursos que consisten en descifrar códigos encriptados con diferentes tamaños de llaves. El método DES de 56 bits ha sido exitosamente violado por uno de los dos esfuerzos siguientes y por ambos combinados:

- A través de la organización "distributed.net"[7] que agrupa a miles de voluntarios en todo el mundo. Los voluntarios instalan en sus computadoras un programa que utiliza el tiempo ocioso de CPU para probar llaves; en caso de encontrarse, se envía por Internet a la computadora central de distributed.net.

- A través de una computadora diseñada ex profeso para violar el código DES. La computadora fue diseñada y construida por alrededor de sólo 250 mil dólares mediante el esfuerzo conjunto de tres compañías[8]. Esta computadora especial puede buscar unas 90 mil millones de llaves por segundo[9].

El 19 de enero de 1999 ambas esfuerzos fueron combinados y se logró violar el código DES de prueba ¡en sólo 22 horas! El esfuerzo combinado produjo en sus momentos más intensos, una búsqueda de 245 mil millones de llaves por segundo. En opinión de este autor, queda claro que:

1. El código DES de 56 bits, aunque es ampliamente usado en todo el mundo y oficialmente respaldado por el gobierno de los Estados Unidos de Norteamérica, no es seguro ante la capacidad actual de cómputo y organización de ataque.
2. No obstante el punto anterior, el mismo algoritmo DES puede ser empleado pero con llaves de mayor longitud, por ejemplo de 112 o de 128 bits. Usar una llave de 128 bits aumenta el universo de llaves posibles en forma exponencial, dejando sin posibilidades de éxito a cualquiera de los ataques por fuerza bruta que se pueden prever en el futuro cercano o a mediano plazo.

2.3 Implementación en Visual C

DigiMarker es una aplicación esteganográfica. La tecnología esteganográfica provee seguridad altísima sobre el mensaje enviado, tanta que ni siquiera deja ver que exista comunicación alguna. No obstante, tiene un problema: depende del conocimiento del algoritmo empleado. En otras palabras, cualquier buen programador que robara el algoritmo esteganográfico, podría extraer la información con relativa facilidad. Por esta razón, decidí implantar un nivel adicional de seguridad: encriptar la información con DES a 128 bits antes de incrustarla esteganográficamente.

Si el lector ha comprendido el funcionamiento de S-DES presentado en páginas anteriores, está listo para abordar el verdadero DES. A continuación presentamos este encriptador, que hasta la fecha es el más popular del mundo.

DES fue adoptado por el Instituto Nacional de Estándares y Tecnología de los EUA desde 1977 como el Estándar 46 de Procesamiento de Información Federal. En DES, los datos son encriptados en bloques de 64 bits, usando una llave de 56 bits. El algoritmo transforma su entrada (llana) de 64 bits, a través de una serie de pasos en una salida (encriptada) también de 64 bits. Los mismos pasos, con la misma llave, se usan para desencriptar.

Vea la figura 18. El lado izquierdo muestra que son tres las etapas por las que pasa el texto de entrada. Primero, se pasa a través de una permutación inicial que reacomoda los bits para producir una entrada permutada. A esto le sigue una etapa de 16 rondas de una misma función que incluye tanto permutaciones como sustituciones. La salida de la última ronda (16) consiste en 64 bits que son una función de el texto de entrada y la llave. Las partes izquierda y derecha de esta salida se intercambian para producir una pre-salida. Finalmente, la pre-salida se pasa a través de la permutación inicial inversa para producir el texto encriptado de 64 bits. Con excepción de las permutaciones inicial y final, DES tiene exactamente la misma estructura que el encriptador Feistel mostrado en la figura 17.

La parte derecha del diagrama muestra cómo se usa la llave de 56 bits. Inicialmente la llave se pasa a través de una función de permutación. Después, para cada una de las 16 rondas, se produce una subllave K_i mediante la combinación de un corrimiento circular izquierdo y una

permutación. Aunque la permutación es la misma en cada ronda, se produce una subllave distinta por la iteración repetida de los bits de la llave.

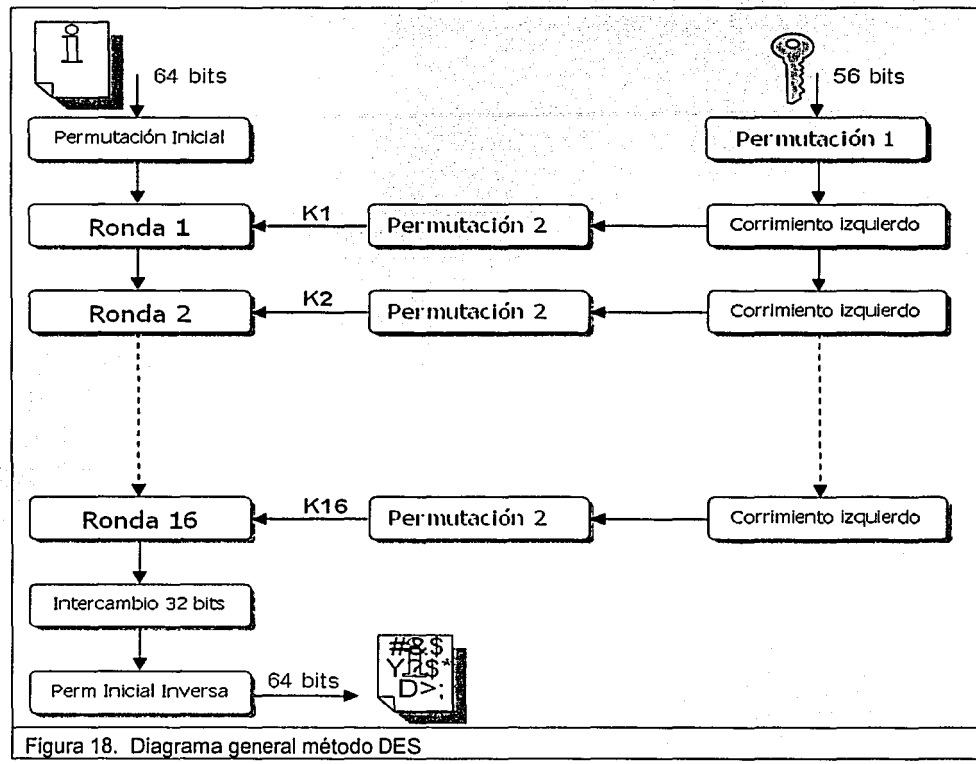


Figura 18. Diagrama general método DES

Semejante al S-DES, en DES las funciones de permutación están definidas por tablas, eso sí, bastante más grandes. Teniendo en mente S-DES se puede prever también que DES requerirá tablas para las demás funciones de expansión, reducción, cajas S e inversas de permutación. Dichas tablas no se presentan aquí explícitamente pero pueden ser consultadas en cualquier libro que detalle el funcionamiento de DES. Al final de este capítulo también se les puede encontrar implementadas en C++.

Pero, ¿cómo se puede fortalecer DES si ya vimos que su versión de 56 bits es factible de ser violada?

Debido a la vulnerabilidad potencial de DES ante un ataque de fuerza bruta, ha habido un interés considerable en encontrar una alternativa. Una posibilidad es encriptar reiteradas veces con llaves múltiples.

Por las razones anteriormente expuestas, en DigiMarker se utiliza el algoritmo DES con llave de 128 bits. En las páginas siguientes se encuentran las porciones más importantes del código en Visual C++ de esta implementación. El propósito de esta tesis no es enseñar programación; por esa razón no se explican muchos detalles, pero estoy seguro de que será de gran utilidad para los interesados en esta fascinante área de la computación. Es necesario recalcar que el algoritmo DES es de dominio público y no es difícil encontrar implementaciones en diferentes lenguajes en la Internet. También puede encontrar el algoritmo explicado paso a paso en: ftp://ripem.msu.edu/pub/crypt/docs/des-algorithm-details.txt

2.3.1 Archivo DES.CPP

```
////////////////////////////////////////////////////////////////////////////////////////
// Implementación DES - 128 bits

#include "stdafx.h"
#include "DES.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////////////////
// Datos estáticos usados por DES, D2DES y D3DES
static unsigned char Df_Key[24] = {
        0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
        0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10,
        0x89, 0xab, 0xcd, 0xef, 0x01, 0x23, 0x45, 0x67
};

static unsigned short bytebit[8] = {
        0200, 0100, 040, 020, 010, 04, 02, 01
};

static unsigned long bigbyte[24] = {
        0x800000L,      0x400000L,      0x200000L,      0x100000L,
        0x80000L,       0x40000L,       0x20000L,       0x10000L,
        0x8000L,        0x4000L,        0x2000L,        0x1000L,
        0x800L,         0x400L,         0x200L,         0x100L,
        0x80L,          0x40L,          0x20L,          0x10L,
        0x8L,           0x4L,           0x2L,           0x1L
};

/* Las tablas empleadas son de acuerdo a la especificación ANSI X3.92-1981. */
static unsigned char pc1[56] = {
        56, 48, 40, 32, 24, 16,  8,     0, 57, 49, 41, 33, 25, 17,
         9,  1, 58, 50, 42, 34, 26,    18, 10,  2, 59, 51, 43, 35,
        62, 54, 46, 38, 30, 22, 14,     6, 61, 53, 45, 37, 29, 21,
        13,  5, 60, 52, 44, 36, 28,    20, 12,  4, 27, 19, 11,  3
};

static unsigned char totrot[16] = {
        1, 2, 4, 6, 8, 10, 12, 14, 15, 17, 19, 21, 23, 25, 27, 28
};

static unsigned char pc2[48] = {
```

```
        13, 16, 10, 23,  0,  4,  2, 27, 14,  5, 20,  9,
        22, 18, 11,  3, 25,  7, 15,  6, 26, 19, 12,  1,
        40, 51, 30, 36, 46, 54, 29, 39, 50, 44, 32, 47,
        43, 48, 38, 55, 33, 52, 45, 41, 49, 35, 28, 31
};

static unsigned long SP1[64] = {
        0x01010400L, 0x00000000L, 0x00010000L, 0x01010404L,
        0x01010004L, 0x00010404L, 0x00000004L, 0x00010000L,
        0x00000400L, 0x01010400L, 0x01010404L, 0x00000400L,
        0x01000404L, 0x01010004L, 0x01000000L, 0x00000004L,
        0x00000404L, 0x01000400L, 0x01000400L, 0x00010400L,
        0x00010400L, 0x01010000L, 0x01010000L, 0x01000404L,
        0x00010004L, 0x01000004L, 0x01000004L, 0x00010004L,
        0x00000000L, 0x00000404L, 0x00010404L, 0x01000000L,
        0x00010000L, 0x01010404L, 0x00000004L, 0x01010000L,
        0x01010400L, 0x01000000L, 0x01000000L, 0x00000400L,
        0x01010004L, 0x00010000L, 0x00010400L, 0x01000004L,
        0x00000400L, 0x00000004L, 0x01000404L, 0x00010404L,
        0x01010404L, 0x00010004L, 0x01010000L, 0x01000404L,
        0x01000004L, 0x00000404L, 0x00010404L, 0x01010400L,
        0x00000404L, 0x01000400L, 0x01000400L, 0x00000000L,
        0x00010004L, 0x00010400L, 0x00000000L, 0x01010004L
};

static unsigned long SP2[64] = {
        0x80108020L, 0x80008000L, 0x00008000L, 0x00108020L,
        0x00100000L, 0x00000020L, 0x80100020L, 0x80008020L,
        0x80000020L, 0x80108020L, 0x80108000L, 0x80000000L,
        0x80008000L, 0x00100000L, 0x00000020L, 0x80100020L,
        0x00108000L, 0x00100020L, 0x80008020L, 0x00000000L,
        0x80000000L, 0x00008000L, 0x00108020L, 0x80100000L,
        0x00100020L, 0x80000020L, 0x00000000L, 0x00108000L,
        0x00008020L, 0x80108000L, 0x80100000L, 0x00008020L,
        0x00000000L, 0x00108020L, 0x80100020L, 0x00100000L,
        0x80008020L, 0x80100000L, 0x80108000L, 0x00008000L,
        0x80100000L, 0x80008000L, 0x00000020L, 0x80108020L,
        0x00108020L, 0x00000020L, 0x00008000L, 0x80000000L,
        0x00008020L, 0x80108000L, 0x00100000L, 0x80000020L,
        0x00100020L, 0x80008020L, 0x80000020L, 0x00100020L,
        0x00108000L, 0x00000000L, 0x80008000L, 0x00008020L,
        0x80000000L, 0x80100020L, 0x80108020L, 0x00108000L
};

static unsigned long SP3[64] = {
        0x00000208L, 0x08020200L, 0x00000000L, 0x08020008L,
        0x08000200L, 0x00000000L, 0x00020208L, 0x08000200L,
        0x00020008L, 0x08000008L, 0x08000008L, 0x00020000L,
        0x08020208L, 0x00020008L, 0x08020000L, 0x00000208L,
        0x08000000L, 0x00000008L, 0x08020200L, 0x00000200L,
        0x00020200L, 0x08020000L, 0x08020008L, 0x00020208L,
        0x08000208L, 0x00020200L, 0x00020000L, 0x08000208L,
        0x00000008L, 0x08020208L, 0x00000200L, 0x08000000L,
        0x08020200L, 0x08000000L, 0x00020008L, 0x00000208L,
        0x00020000L, 0x08020200L, 0x08000200L, 0x00000000L,
        0x00000200L, 0x00020008L, 0x08020208L, 0x08000200L,
        0x08000008L, 0x00000200L, 0x00000000L, 0x08020008L,
        0x08000208L, 0x00020000L, 0x08000000L, 0x08020208L,
        0x00000008L, 0x00020208L, 0x00020200L, 0x08000008L,
        0x08020000L, 0x08000208L, 0x00000208L, 0x08020000L,
        0x00020208L, 0x00000008L, 0x08020008L, 0x00020200L
};

static unsigned long SP4[64] = {
        0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
        0x00802080L, 0x00800081L, 0x00800001L, 0x00002001L,
        0x00000000L, 0x00802000L, 0x00802000L, 0x00802081L,
        0x00000081L, 0x00000000L, 0x00800080L, 0x00800001L,
        0x00000001L, 0x00002000L, 0x00800000L, 0x00802001L,
        0x00000080L, 0x00800000L, 0x00002001L, 0x00002080L,
        0x00800081L, 0x00000001L, 0x00002080L, 0x00800080L,
```

```
        0x00002000L, 0x00802080L, 0x00802081L, 0x00000081L,
        0x00800080L, 0x00800001L, 0x00802000L, 0x00802081L,
        0x00000081L, 0x00000000L, 0x00000000L, 0x00802000L,
        0x00002080L, 0x00800080L, 0x00800081L, 0x00000001L,
        0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
        0x00802081L, 0x00000081L, 0x00000001L, 0x00002000L,
        0x00800001L, 0x00002001L, 0x00802080L, 0x00800081L,
        0x00002001L, 0x00002080L, 0x00800000L, 0x00802001L,
        0x00000080L, 0x00800000L, 0x00002000L, 0x00802080L
};

static unsigned long SP5[64] = {
        0x00000100L, 0x02080100L, 0x02080000L, 0x42000100L,
        0x00080000L, 0x00000100L, 0x40000000L, 0x02080000L,
        0x40080100L, 0x00080000L, 0x02000100L, 0x40080100L,
        0x42000100L, 0x42080000L, 0x00080100L, 0x40000000L,
        0x02000000L, 0x40080000L, 0x40080000L, 0x00000000L,
        0x40000100L, 0x42080100L, 0x42080100L, 0x02000100L,
        0x42080000L, 0x40000100L, 0x00000000L, 0x42000000L,
        0x02080100L, 0x02000000L, 0x42000000L, 0x00080100L,
        0x00080000L, 0x42000100L, 0x00000100L, 0x02000000L,
        0x40000000L, 0x02080000L, 0x42000100L, 0x40080100L,
        0x02000100L, 0x40000000L, 0x42080000L, 0x02080100L,
        0x40080100L, 0x00000100L, 0x02000000L, 0x42080000L,
        0x42080100L, 0x00080100L, 0x42000000L, 0x42080100L,
        0x02080000L, 0x00000000L, 0x40080000L, 0x42000000L,
        0x00080100L, 0x02000100L, 0x40000100L, 0x00080000L,
        0x00000000L, 0x40080000L, 0x02080100L, 0x40000100L
};

static unsigned long SP6[64] = {
        0x20000010L, 0x20400000L, 0x00004000L, 0x20404010L,
        0x20400000L, 0x00000010L, 0x20404010L, 0x00400000L,
        0x20004000L, 0x00404010L, 0x00400000L, 0x20000010L,
        0x00400010L, 0x20004000L, 0x20000000L, 0x00004010L,
        0x00000000L, 0x00400010L, 0x20004010L, 0x00004000L,
        0x00404000L, 0x20004010L, 0x00000010L, 0x20400010L,
        0x20400010L, 0x00000000L, 0x00404010L, 0x20404000L,
        0x00004010L, 0x00404000L, 0x20404000L, 0x20000000L,
        0x20004000L, 0x00000010L, 0x20400010L, 0x00404000L,
        0x20404010L, 0x00400000L, 0x00004010L, 0x20000010L,
        0x00400000L, 0x20004000L, 0x20000000L, 0x00004010L,
        0x20000010L, 0x20404010L, 0x00404000L, 0x20400000L,
        0x00404010L, 0x20404000L, 0x00000000L, 0x20400010L,
        0x00000010L, 0x00004000L, 0x20400000L, 0x00404010L,
        0x00004000L, 0x00400010L, 0x20004010L, 0x00000000L,
        0x20404000L, 0x20000000L, 0x00400010L, 0x20004010L
};

static unsigned long SP7[64] = {
        0x00200000L, 0x04200002L, 0x04000802L, 0x00000000L,
        0x00000800L, 0x04000802L, 0x00200802L, 0x04200800L,
        0x04200802L, 0x00200000L, 0x00000000L, 0x04000002L,
        0x00000002L, 0x04000000L, 0x04200002L, 0x00000802L,
        0x04000800L, 0x00200802L, 0x00200002L, 0x04000800L,
        0x04000002L, 0x04200000L, 0x04200800L, 0x00200002L,
        0x04200000L, 0x00000800L, 0x00000802L, 0x04200802L,
        0x00200800L, 0x00000002L, 0x04000000L, 0x00200800L,
        0x04000000L, 0x00200800L, 0x00200000L, 0x04000802L,
        0x04000802L, 0x04200002L, 0x04200002L, 0x00000002L,
        0x00200002L, 0x04000000L, 0x04000800L, 0x00200000L,
        0x04200800L, 0x00000802L, 0x00200802L, 0x04200800L,
        0x00000802L, 0x04000002L, 0x04200802L, 0x04200000L,
        0x00200800L, 0x00000000L, 0x00000002L, 0x04200802L,
        0x00000000L, 0x00200802L, 0x04200000L, 0x00000800L,
        0x04000002L, 0x04000800L, 0x00000800L, 0x00200002L
};

static unsigned long SP8[64] = {
        0x10001040L, 0x00001000L, 0x00040000L, 0x10041040L,
        0x10000000L, 0x10001040L, 0x00000040L, 0x10000000L,
```

```
        0x00040040L, 0x10040000L, 0x10041040L, 0x00041000L,
        0x10041000L, 0x00041040L, 0x00001000L, 0x00000040L,
        0x10040000L, 0x10000040L, 0x10001000L, 0x00001040L,
        0x00041000L, 0x00040040L, 0x10040040L, 0x10041000L,
        0x00001040L, 0x00000000L, 0x00000000L, 0x10040040L,
        0x10000040L, 0x10001000L, 0x00041040L, 0x00040000L,
        0x00041040L, 0x00040000L, 0x10041000L, 0x00001000L,
        0x00000040L, 0x10040040L, 0x00001000L, 0x00041040L,
        0x10001000L, 0x00000040L, 0x10000040L, 0x10040000L,
        0x10040040L, 0x10000000L, 0x00040000L, 0x10001040L,
        0x00000000L, 0x10041040L, 0x00040040L, 0x10000040L,
        0x10040000L, 0x10001000L, 0x10001040L, 0x00000000L,
        0x10041040L, 0x00041000L, 0x00041000L, 0x00001040L,
        0x00001040L, 0x00040040L, 0x10000000L, 0x10041000L
};

// Fin de datos estáticos
////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////
// Clase CDDES
CDDES::CDDES()
{
        int i;

        m_nKeyLength = m_nUnitLength = 8;
        for ( i = 0 ; i < 32 ; i++ )
                KnL[i] = 0L;
}

CDDES::~CDDES()
{
}

////////////////////////////////////////////////////////////////////////////////////////////////
// Funciones internas
void CDDES::Scrunch(LPBYTE lpOutof, DWORD *lpInto)
{

        *lpInto  = (*lpOutof++ & 0xffL) << 24;
        *lpInto |= (*lpOutof++ & 0xffL) << 16;
        *lpInto |= (*lpOutof++ & 0xffL) << 8;
        *lpInto++ |= (*lpOutof++ & 0xffL);
        *lpInto  = (*lpOutof++ & 0xffL) << 24;
        *lpInto |= (*lpOutof++ & 0xffL) << 16;
        *lpInto |= (*lpOutof++ & 0xffL) << 8;
        *lpInto |= (*lpOutof   & 0xffL);
}

void CDDES::UnScrunch(DWORD *lpOutof, LPBYTE lpInto)
{
        *lpInto++ = (BYTE)((*lpOutof >> 24) & 0xffL);
        *lpInto++ = (BYTE)((*lpOutof >> 16) & 0xffL);
        *lpInto++ = (BYTE)((*lpOutof >>  8) & 0xffL);
        *lpInto++ = (BYTE)(*lpOutof++  & 0xffL);
        *lpInto++ = (BYTE)((*lpOutof >> 24) & 0xffL);
        *lpInto++ = (BYTE)((*lpOutof >> 16) & 0xffL);
        *lpInto++ = (BYTE)((*lpOutof >>  8) & 0xffL);
        *lpInto = (BYTE)(*lpOutof       & 0xffL);
}

void CDDES::DESFunc(DWORD *lpBlock, DWORD *lpKeys)
{
        DWORD fval, work, right, leftt;
        int round;

        leftt = lpBlock[0];
        right = lpBlock[1];
        work = ((leftt >> 4) ^ right) & 0x0f0f0f0fL;
        right ^= work;
        leftt ^= (work << 4);
```

```
        work = ((leftt >> 16) ^ right) & 0x0000ffffL;
        right ^= work;
        leftt ^= (work << 16);
        work = ((right >> 2) ^ leftt) & 0x33333333L;
        leftt ^= work;
        right ^= (work << 2);
        work = ((right >> 8) ^ leftt) & 0x00ff00ffL;
        leftt ^= work;
        right ^= (work << 8);
        right = ((right << 1) | ((right >> 31) & 1L)) & 0xffffffffL;
        work = (leftt ^ right) & 0xaaaaaaaaL;
        leftt ^= work;
        right ^= work;
        leftt = ((leftt << 1) | ((leftt >> 31) & 1L)) & 0xffffffffL;

        for( round = 0; round < 8; round++ )
        {
                work  = (right << 28) | (right >> 4);
                work ^= *lpKeys++;
                fval  = SP7[ work               & 0x3fL];
                fval |= SP5[(work >>  8) & 0x3fL];
                fval |= SP3[(work >> 16) & 0x3fL];
                fval |= SP1[(work >> 24) & 0x3fL];
                work  = right ^ *lpKeys++;
                fval |= SP8[ work               & 0x3fL];
                fval |= SP6[(work >>  8) & 0x3fL];
                fval |= SP4[(work >> 16) & 0x3fL];
                fval |= SP2[(work >> 24) & 0x3fL];
                leftt ^= fval;
                work  = (leftt << 28) | (leftt >> 4);
                work ^= *lpKeys++;
                fval  = SP7[ work               & 0x3fL];
                fval |= SP5[(work >>  8) & 0x3fL];
                fval |= SP3[(work >> 16) & 0x3fL];
                fval |= SP1[(work >> 24) & 0x3fL];
                work  = leftt ^ *lpKeys++;
                fval |= SP8[ work               & 0x3fL];
                fval |= SP6[(work >>  8) & 0x3fL];
                fval |= SP4[(work >> 16) & 0x3fL];
                fval |= SP2[(work >> 24) & 0x3fL];
                right ^= fval;
        }

        right = (right << 31) | (right >> 1);
        work = (leftt ^ right) & 0xaaaaaaaaL;
        leftt ^= work;
        right ^= work;
        leftt = (leftt << 31) | (leftt >> 1);
        work = ((leftt >> 8) ^ right) & 0x00ff00ffL;
        right ^= work;
        leftt ^= (work << 8);
        work = ((leftt >> 2) ^ right) & 0x33333333L;
        right ^= work;
        leftt ^= (work << 2);
        work = ((right >> 16) ^ leftt) & 0x0000ffffL;
        leftt ^= work;
        right ^= (work << 16);
        work = ((right >> 4) ^ leftt) & 0x0f0f0f0fL;
        leftt ^= work;
        right ^= (work << 4);
        *lpBlock++ = right;
        *lpBlock = leftt;
}

void CDDES::Cookey(DWORD *lpRaw1)
{
        DWORD *lpCook, *lpRaw0;
        DWORD  dough[32];
        int        i;

        lpCook = dough;
```

```
        for( i = 0 ; i < 16; i++, lpRaw1++ )
        {
                lpRaw0 = lpRaw1++;
                *lpCook  = (*lpRaw0 & 0x00fc0000L) << 6;
                *lpCook |= (*lpRaw0 & 0x00000fc0L) << 10;
                *lpCook |= (*lpRaw1 & 0x00fc0000L) >> 10;
                *lpCook++ |= (*lpRaw1 & 0x00000fc0L) >> 6;
                *lpCook  = (*lpRaw0 & 0x0003f000L) << 12;
                *lpCook |= (*lpRaw0 & 0x0000003fL) << 16;
                *lpCook |= (*lpRaw1 & 0x0003f000L) >> 4;
                *lpCook++ |= (*lpRaw1 & 0x0000003fL);
        }
        CDDES::LoadKey(dough);
}

// Fin de funciones internas
////////////////////////////////////////////////////////////////////////////////////////////////

// Función para cargar la llave desde un vector externo
void CDDES::LoadKey(DWORD *lpKey)
{
        int             i;

        for ( i = 0 ; i < 32 ; i++ )
                KnL[i] = lpKey[i];
}

// Función para salvar la llave actual (KnL) en un vector externo
void CDDES::SaveKey(DWORD *lpKey)
//                  DWORD[32]
{
        int             i;

        for ( i = 0 ; i < 32 ; i++ )
                lpKey[i] = KnL[i];
}

// Función para generar una llave
void CDDES::GenerateKey(LPBYTE lpKey, UINT nMode)
//                      BYTE[8],      DES_ENCRYPT/DES_DECRYPT
{
        int  i, j, l, m, n;
        BYTE pc1m[56], pcr[56];
        DWORD kn[32];

        for ( j = 0; j < 56; j++ )
        {
                l = pc1[j];
                m = l & 07;
                pc1m[j] = (lpKey[l >> 3] & bytebit[m]) ? 1 : 0;
        }
        for( i = 0; i < 16; i++ )
        {
                if( nMode == DES_DECRYPT ) m = (15 - i) << 1;
                else m = i << 1;
                n = m + 1;
                kn[m] = kn[n] = 0L;
                for( j = 0; j < 28; j++ )
                {
                        l = j + totrot[i];
                        if( l < 28 ) pcr[j] = pc1m[l];
                        else pcr[j] = pc1m[l - 28];
                }
                for( j = 28; j < 56; j++ )
                {
                    l = j + totrot[i];
                    if( l < 56 ) pcr[j] = pc1m[l];
                    else pcr[j] = pc1m[l - 28];
                }
                for( j = 0; j < 24; j++ )
                {
```

```
                        if( pcr[pc2[j]] ) kn[m] |= bigbyte[j];
                        if( pcr[pc2[j+24]] ) kn[n] |= bigbyte[j];
                }
        }
        Cookey(kn);
}

// Ejecuta el DDES
void CDDES::Go(LPBYTE lpTo, LPBYTE lpFrom)
//              BYTE[8],      BYTE[8]
{
        DWORD work[2];

        Scrunch(lpFrom, work);
        DESFunc(work, KnL);
        UnScrunch(work, lpTo);
}

// Obtiene longitud de la llave actual
UINT CDDES::GetKeyLength()
{
        return m_nKeyLength;
}

UINT CDDES::GetUnitLength()
{
        return m_nUnitLength;
}

////////////////////////////////////////////////////////////////////////////////////////////////
// Clase CD2DES
CD2DES::CD2DES()
{
        int i;

        m_nKeyLength = m_nUnitLength = 16;
        for ( i = 0 ; i < 32 ; i++ )
                KnR[i] = Kn3[i] = 0;
}


CD2DES::~CD2DES()
{
}

// Función para generar una llave D2DES
void CD2DES::GenerateKey(LPBYTE lpKey, UINT nMode)
//                       BYTE[16],     DES_ENCRYPT/DES_DECRYPT
{
        short revmod;

        revmod = (nMode == DES_ENCRYPT) ? DES_DECRYPT : DES_ENCRYPT;
        CDDES::GenerateKey(&lpKey[8], revmod);
        CDDES::SaveKey(KnR);
        CDDES::GenerateKey(lpKey, nMode);
        CDDES::SaveKey(Kn3);                                    /* Kn3 = KnL */
        return;
}

// Función para cargar la llave D2DES de un vector externo
void CD2DES::LoadKey(DWORD *lpKey)
//                   DWORD[64]
{
        int             i;

        CDDES::LoadKey(lpKey);
        lpKey = &lpKey[32];
        for ( i = 0 ; i < 32 ; i++ )
                KnR[i] = lpKey[i];
        CDDES::SaveKey(Kn3);                                    /* Kn3 = KnL */
        return;
```

```
}

// Función para salvar la llave actual D2DES key en un vector externo
void CD2DES::SaveKey(DWORD *lpKey)
//                   DWORD[64]
{
        int             i;

        CDDES::SaveKey(lpKey);
        lpKey = &lpKey[32];
        for ( i = 0 ; i < 32 ; i++ )
                lpKey[i] = KnR[i];
}

// Ejecuta el D2DES
void CD2DES::Go(LPBYTE lpTo, LPBYTE lpFrom)
//                           BYTE[16],    BYTE[16]
{
        DWORD *right, *ll, swap;
        DWORD leftt[2], bufR[2];

        right = bufR;
        ll = &leftt[1];
        Scrunch(lpFrom, leftt);
        Scrunch(&lpFrom[8], right);
        DESFunc(leftt, KnL);
        DESFunc(right, KnL);
        swap = *ll;
        *ll = *right;
        *right = swap;
        DESFunc(leftt, KnR);
        DESFunc(right, KnR);
        swap = *ll;
        *ll = *right;
        *right = swap;
        DESFunc(leftt, Kn3);
        DESFunc(right, Kn3);
        UnScrunch(leftt, lpTo);
        UnScrunch(right, &lpTo[8]);
}
```

2.3.2 Archivo DES.H

```
////////////////////////////////////////////////////////////////////////////////
// DDES, D2DES header file


#if !defined __DES_H__
#define __DES_H__

#define DES_ENCRYPT             0
#define DES_DECRYPT             1

////////////////////////////////////////////////////////////////////////////////
// Clase CDDES
class CDDES
{
// Protected data members
protected:
        UINT                    m_nKeyLength;  // Longitud de llave(en bytes) para este DES
        UINT                    m_nUnitLength; // Longitud mínima de datos (en bytes)
        DWORD                   KnL[32];       // Llave interna (primera parte)

// funciones internas
protected:
        virtual void    Scrunch(LPBYTE lpOutof, DWORD *lpInto);
        virtual void    UnScrunch(DWORD *lpOutof, LPBYTE lpInto);
        virtual void    DESFunc(DWORD *lpBlock, DWORD *lpKey);
        virtual void    Cookey(DWORD *lpRaw1);
        virtual void    LoadKey(DWORD *lpKey);
        virtual void    SaveKey(DWORD *lpKey);
```

```
// interfaz
public:
        CDDES();
        ~CDDES();
        virtual UINT    GetKeyLength();
        virtual UINT    GetUnitLength();
        virtual void    GenerateKey(LPBYTE lpKey, UINT nMode);
        virtual void    Go(LPBYTE lpTo, LPBYTE lpFrom);
};

////////////////////////////////////////////////////////////////////////////////
// Clase CD2DES
class CD2DES : public CDDES
{
protected:
        DWORD                   KnR[32];                // Llave interna (segunda parte)
        DWORD                   Kn3[32];                // Lave interna (tercera parte)

// Funciones internas
protected:
        virtual void    LoadKey(DWORD *lpKey);
        virtual void    SaveKey(DWORD *lpKey);

// interfaz
public:
        CD2DES();
        ~CD2DES();
        virtual void    GenerateKey(LPBYTE lpKey, UINT nMode);
        virtual void    Go(LPBYTE lpTo, LPBYTE lpFrom);
};

#endif
```

Las clases CDDES y CD2DES se usan para encriptar y desencriptar el mensaje oculto. Aunque finalmente sólo se usa D2DES, D2DES requiere a DDES. Por eso, se han colocado juntas. Para utilizar D2DES, se crea una instancia de CD2DES y se invocan sus funciones miembro para encriptar y desencriptar.

Variables de interés:

- m_nKeyLength
  Entero sin signo para indicar la longitud de la llave en bytes. Este valor debe ser puesto en el constructor de la clase DES. Por ejemplo, valdrá 16 para CD2DES.
- m_nUnitLength
  Entero sin signo para indicar la longitud del bloque básico en bytes. Recordemos que DES es un método de encriptamiento por bloques, por esa razón se requiere definir una longitud fija de procesamiento. También debe ser establecido en el constructor de la clase DES.
- KnL, KnR, Kn3
  Arreglo interno para guardar la llave por partes.

Funciones de interés:

- GenerateKey()
  Función para generar una llave DES interna de acuerdo con la llave provista y el tipo

de operación (encriptar/desencriptar). Se debe llamar con los parámetros correctos cada vez que se quiera encriptar o desencriptar.

- GetKeyLength()
  Función para recuperar el tamaño de la llave.
- GetUnitLength()
  Función para recuperar la longitud del bloque.
- Go()
  Función para ejecutar el encriptamiento/desencriptamiento

2.3.3 Archivo BlackBox.CPP

```
////////////////////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "BlackBox.h"
#include "DMCore.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE

static char THIS_FILE[] = __FILE__;
#endif

CBlackBox::CBlackBox()
{
        ASSERT(m_des.GetKeyLength() == KEY_LENGTH); // Longitud fija para la llave
}

CBlackBox::~CBlackBox()
{
}

////////////////////////////////////////////////////////////////////////////////////////////
// Función para obtener un tamaño de buffer correcto para este DES
DWORD CBlackBox::GetBoundedBufferSize(DWORD dwOriginalSize)
{
        DWORD           dwSize;

        if ( dwOriginalSize % m_des.GetUnitLength() == 0 )
                return dwOriginalSize;

        dwSize = ( dwOriginalSize / m_des.GetUnitLength() + 1 ) * m_des.GetUnitLength();
        return dwSize;
}

////////////////////////////////////////////////////////////////////////////////////////////
// Función para establecer la llave
void CBlackBox::SetKey(LPBYTE lpKey)
{
        ::memcpy(m_key, lpKey, KEY_LENGTH);
}

////////////////////////////////////////////////////////////////////////////////////////////
// Función para encriptar
void CBlackBox::Encrypt(LPBYTE lpBuffer, DWORD dwBufferSize, CDMCore *pCore)
{
        DWORD dwLeft;

        ASSERT(!(dwBufferSize % m_des.GetUnitLength()) && pCore);
        dwLeft = dwBufferSize;
        m_des.GenerateKey(m_key, DES_ENCRYPT);
        pCore -> SetProgress(0);
        while(dwLeft)
        {
```

```
                m_des.Go(lpBuffer, lpBuffer);
                lpBuffer = &lpBuffer[m_des.GetUnitLength()];
                dwLeft -= m_des.GetUnitLength();
                pCore -> SetProgress(( dwBufferSize - dwLeft ) * 100 / dwBufferSize);
        }
}

////////////////////////////////////////////////////////////////////////////////////////
// Función para desencriptar
void CBlackBox::Decrypt(LPBYTE lpBuffer, DWORD dwBufferSize, CDMCore *pCore)
{
        DWORD dwLeft;

        ASSERT(!(dwBufferSize % m_des.GetUnitLength()) && pCore);
        dwLeft = dwBufferSize;
        m_des.GenerateKey(m_key, DES_DECRYPT);
        pCore -> SetProgress(0);
        while(dwLeft)
        {
                m_des.Go(lpBuffer, lpBuffer);
                lpBuffer = &lpBuffer[m_des.GetUnitLength()];
                dwLeft -= m_des.GetUnitLength();
                pCore -> SetProgress(( dwBufferSize - dwLeft ) * 100 / dwBufferSize);
        }
}
```

2.3.4 Archivo BlackBox.H

```
////////////////////////////////////////////////////////////////////////////////////////
// CBlackBox class header file


#if !defined __BLACKBOX_H__
#define __BLACKBOX_H__

#include "DES.h"

#define KEY_LENGTH              16

class CDMCore;

class CBlackBox
{
// Protected data members
protected:
        CD2DES          m_des;
        BYTE            m_key[16];

// Public interface
public:
        CBlackBox();
        ~CBlackBox();
        DWORD                   GetBoundedBufferSize(DWORD dwOriginalSize);
        void                    SetKey(LPBYTE lpKey);
        void                    Encrypt(LPBYTE lpBuffer, DWORD dwBufferSize, CDMCore *pCore);
        void                    Decrypt(LPBYTE lpBuffer, DWORD dwBufferSize, CDMCore *pCore);
};

#endif
```

Debido a que DES requiere que su buffer sea de longitud fija (tamaño del bloque a procesar), se debe seccionar el bloque grande en muchas piezas pequeñas de la misma longitud. La clase CBlackBox se encarga de esto. Realiza un ciclo para procesar este número de bytes cada vuelta hasta completar el procesamiento de todo el mensaje.

Por la forma en que está estructurado, el núcleo de DigiMarker no se preocupa de qué algoritmo se usa para encriptar y lo único que tiene que hacer es enviar a esta clase el buffer que debe encriptarse/desencriptarse. Por esta razón, se llama *blackbox*, es decir, "caja negra". Para usar esta clase, simplemente hay que crear una instancia de ella y llamar a sus funciones miembro.

Variables de interés:

- m_des
  Instancia de la clase CD2DES para implementar el algoritmo DES.
- m_key
  Arreglo de bytes para guardar la llave. CBlackBox llama a GenerateKey() automáticamente desde sus funciones miembro Encrypt() y Decrypt(), así que no es necesario generar la llave cada vez.

Funciones de interés:

- Encrypt()
  Función para encriptar el buffer. Como parámetro se pasa el tamaño del buffer.
- Decrypt()
  Función para desencriptar un buffer. Como parámetro se pasa el tamaño del buffer.
- GetBoundedBufferSize()
  Función para recuperar el tamaño correcto de buffer de acuerdo al tamaño original. Se debe invocar primero esta función para poder obtener el tamaño correcto antes de crear el buffer que será utilizado para encriptar/desencriptar.
- SetKey()
  Función para establecer la llave en curso para encriptar/desencriptar.

---

[1] La mayor parte de la información presentada en este capítulo ha sido tomada de Stallings, William. *Cryptography and Network Security, Principles and Practice*, Prentice Hall, 1999.
[2] En 1883, Auguste Kerckhoffs enunció las primeras bases para la ingeniería en criptografía.
[3] Del ingles *Data Encryption Standard*. Estándar de encriptamiento de datos. Aunque muchos algoritmos de encriptamiento han surgido posteriormente, el método DES sigue siendo el más importante.
[4] El criptoanálisis es el esfuerzo analítico para descubrir la llave o el mensaje provenientes de un sistema criptográfico. Se puede categorizar de diferentes formas según los datos que tenga disponibles para su análisis.
[5] El algoritmo "cubierta de una pasada" fue propuesto por Mauborgne y es inviolable. Se basa en producir una salida aleatoria sin relación estadística alguna con el mensaje original. Como no hay relación alguna, simplemente es imposible violar el código. No obstante, es raramente empleado por requerir una extrema longitud de la llave aleatoria y que además esté en posesión tanto del encriptador como del desencriptador.
[6] Dirección en Internet: http://www.rsasecurity.com
[7] Dirección en Internet: http://www.distributed.net/
[8] Las compañías son Cryptography Research, Advanced Wireless Technologies y Electronic Frontier Foundation. Más información puede ser consultada en: http://www.eff.org/descracker.html
[9] Para más información consulte: http://www.cryptography.com/resources/whitepapers/DES.html

3

Formatos de Imágenes Digitales

"No vemos las cosas como son; vemos las cosas como somos"

Anais Nin

3.1 Introducción a las imágenes digitales

Una imagen natural comienza como una sucesión continua de variaciones de color y sombras. En el caso de una fotografía las sombras varían de claras a oscuras y los colores varían de rojos a azules, pasando por los amarillos. A una imagen de esta naturaleza se le conoce como *de tono continuo*. Eso quiere decir que las sombras y colores variantes se mezclan sin interrupción abrupta, describiendo fielmente la escena original.

3.1.1 Imágenes en tonos de gris

Una imagen digital en tonos de gris está compuesta de puntos de tonalidad gris (brillo) y no de una sucesión continua de tonos variantes. Para fabricar una imagen digital a partir de una imagen de tono continuo, la imagen debe dividirse en puntos individuales de brillo (muestreo). Adicionalmente, cada punto de brillo debe ser descrito por un valor digital (cuantificador).

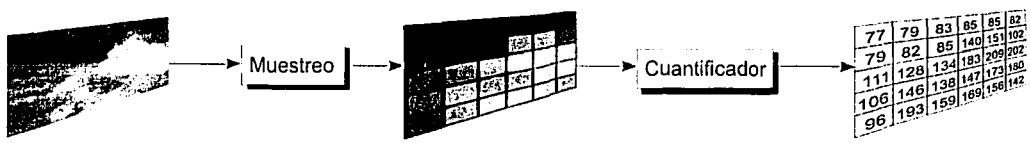


Figura 19. Conversión de imagen de tono continuo a digital.

A cada muestra obtenida se le denomina *píxel*[1] y representa un elemento discreto de la imagen digital. Una imagen generalmente se muestrea en un arreglo rectangular de píxeles. Cada píxel tiene una coordenada (x,y) que corresponde a su ubicación dentro de la imagen. En la mayoría de los casos la coordenada (0,0) se refiere a la esquina superior izquierda de la imagen, siendo la x la localidad horizontal y la y la localidad vertical.

La calidad de la imagen digital está directamente relacionada con el número de píxeles que tenga y con el rango disponible de valores de brillo. A estos aspectos se les conoce como *resolución de la imagen*. Se usa el término *resolución espacial* para describir cuántos píxeles componen una imagen. A mayor número de píxeles, mayor resolución espacial. La figura 20 muestra una misma fotografía a diferentes resoluciones espaciales.

Figura 20.a
Resolución 258 x 339 píxeles

Figura 20.b
Resolución 100 x 131 píxeles

Figura 20.c
Resolución 50 x 66 píxeles

Los requerimientos de resolución espacial para una imagen están determinados por la aplicación que se le quiera dar a la imagen. Para fines de impresión, como en esta tesis, vemos que la resolución de 258 x 339 píxeles fue adecuada para ese tamaño de impresión, pero la de 50 x 66 píxeles ya no es suficiente. Si la intención es enviar las imágenes por correo electrónico, una resolución comúnmente empleada es la denominada *VGA*[2] que consiste en 640 x 480 píxeles. Otros estándares comunes son *SVGA*[3] a 800 x 600 píxeles y *UVGA*[4] a 1024 x 768 píxeles.

Por otro lado, se tiene la *resolución de brillo*. Cada píxel representa la intensidad de brillo de la imagen original en la ubicación espacial donde fue muestreado. El concepto de resolución de brillo considera con qué precisión el brillo de un píxel puede representar la intensidad de brillo de la imagen original. Si se aumenta el rango numérico posible de brillo para cada píxel, se incrementa así mismo la resolución de brillo.

Después del proceso de muestreo (ver Figura 19), cada muestra se cuantifica. El proceso de cuantificación convierte la intensidad de tono continuo, en el punto de muestreo, a un valor digital de brillo. La precisión del valor digital depende directamente de cuántos bits se usen en el cuantificador. Si se usan 3 bits, el brillo se puede convertir en uno de ocho niveles disponibles de gris. En este caso el nivel "0" de gris representa al color negro y el nivel "7" de gris representa al color blanco; los niveles de gris "1" a "6" representan una escala descendente de tonalidades de gris, entre el negro y el blanco.

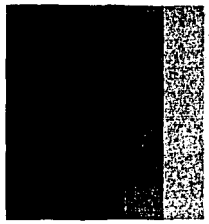


Figura 21.a
Escala de grises a 3 bits

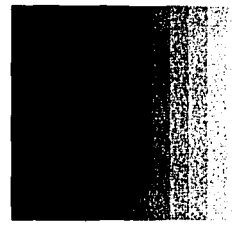


Figura 21.b
Escala de grises a 4 bits

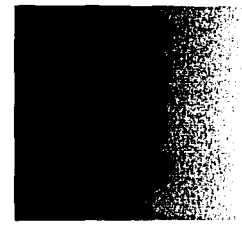


Figura 21.c
Escala de grises a 8 bits

Si se usan 4 bits para representar el brillo, cada píxel es representado por uno de 16 niveles de gris. Análogamente, un valor de brillo de 8 bits permite un rango de 256 tonalidades distintas. Vea la figura 21 y compruebe cómo la suavidad de la escala de grises mejora conforme más bits se usan en el cuantificador.

Un detalle sumamente importante como antecedente para los temas de esteganografía es el entendimiento de los llamados *planos de bits*. Cada plano de bit *i* representa cómo se vería la imagen si sólo se tomara en cuenta la información contenida en el bit *i*. Tomemos como ejemplo una fotografía en tonos de grises a 8 bits, siendo el menos significativo el bit 0 y el más significativo el bit 7. Vea las figuras 22 y 23.

Figura 22. Fotografía original de unos tulipanes. En esta imagen se usan todos los 8 planos de bits producidos por el cuantificador.

Figura 23.a
Plano de bit 7

Figura 23.b
Plano de bit 6

Figura 23.c
Plano de bit 5

Figura 23.d
Plano de bit 4

Figura 23.e
Plano de bit 3

Figura 23.f
Plano de bit 2

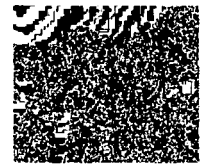


Figura 23.g
Plano de bit 1

Figura 23.h
Plano de bit 0

De la figura 23, podemos hacer observaciones muy importantes: las partes que definen con más importancia a la imagen original vienen dadas por los planos de bits más significativos, siendo el plano 7 el más importante. Por el contrario, los planos de bits menos significativos, siendo el 0 el extremo, contribuyen con detalles de sombras y pequeños artificios que en muchas ocasiones no se notan a simple vista. El plano de bit 0 pudiera interpretarse como ruido de la imagen; es decir, se podría prácticamente eliminar de la misma sin que hubiera cambios visuales notables. En otras palabras, la imagen se puede reconstruir omitiendo uno o más de los planos de bits menos significativos (por ejemplo del 0 al 2), pero no sería posible reconstruir la imagen si se eliminan los planos de bits más significativos (por ejemplo del 5 al 7).

3.1.2 Imágenes a color

Cuando trabajamos con imágenes a color, los mismos conceptos de muestreo, cuantificación, resolución espacial y resolución de brillo son válidos. Pero, en lugar de usar un solo valor para representar el brillo, las imágenes digitales a color generalmente son cuantificadas usando tres componentes de brillo. Al desplegar color, se usan tres emisores independientes de color, que emite cada uno una banda única de luz para generar en conjunto todos los colores del espectro.

Si miramos muy de cerca la pantalla de una computadora, descubriremos puntos individuales de colores sólidos. Esos puntos pueden emitir luz en los colores rojo, verde o azul. Conforme nos alejamos de la pantalla los puntos tienden a mezclarse hasta el grado en que ya no se distinguen los puntos individuales. En lugar de eso, ahora se percibe la combinación de color y se percibe como si fuera un solo color, producto de esos tres colores. Esto es debido a que todos los colores del espectro se pueden generar a partir de los llamados colores primarios: rojo, verde y azul.

A esto se le conoce como la *propiedad aditiva del color*. Si sólo se enciende el color rojo, el color que se percibe naturalmente es rojo. Pero, conforme el rojo varía de oscuro a más iluminado, se podrán observar los distintos brillos del rojo. Así mismo ocurrirá con el verde y con el azul. Ahora bien, si se encienden los componentes rojo y verde, entonces se percibirá un amarillo. Si se varía la intensidad de brillo, entonces se percibirán los colores desde café oscuro hasta amarillo brillante.

¿Cómo se almacenan las imágenes a color? Será fácil comprenderlo recordando los conceptos presentados anteriormente. Si para una imagen en tonos de gris a 8 bits se requiere un byte (ya que un byte puede almacenar uno de 2^8 valores en el rango de 0 a 2^8-1) para almacenar cada píxel, entonces en las imágenes a color que requieran la posibilidad de guardar 3 valores de intensidad (rojo, verde y azul) para cada píxel, se requerirán 3 bytes por píxel. Un byte almacenará la intensidad del rojo, otro la intensidad del verde y otro la intensidad del azul. A este arreglo se le conoce como *imágenes a color de 24 bits* o *de color verdadero.*

Cabe mencionar que también existen imágenes a color de 8 bits, siendo su aplicación más importante los dibujos estilo historieta. Esas imágenes son muy populares en los iconos usados en las páginas de Internet. Se les puede encontrar en forma de botones, separadores, animaciones (que son una secuencia de varias imágenes de 8 bits), menús, etc. Se ahorra espacio usando solamente 8 bits y aunque se tienen disponibles sólo 256 colores, eso ha demostrado ser suficiente. Pero si el interés es mostrar fotografías de escenarios naturales, entonces 8 bits a menudo no serán suficientes, por lo que se recurre a 24 bits que provee un rango aproximado de 16 millones de colores. No obstante, habrá algunos tipos de fotografías que a pesar de representar escenarios naturales puedan ser aceptablemente almacenadas con 8 bits. La figura 24a muestra un dibujo típico que se almacena con 8 bits y la figura 24b muestra un atardecer que, de manera excepcional, también está almacenado en 8 bits sin perder gran fidelidad.

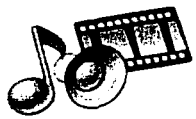


Figura 24a

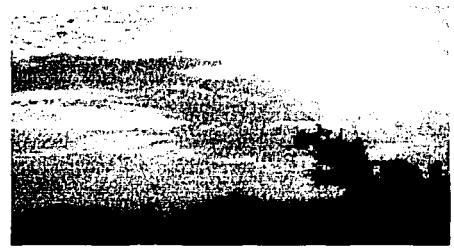


Figura 24b

Ambas imágenes a 8 bits por píxel.

3.1.3 Histogramas

Una herramienta sumamente útil en el *estegoanálisis*[5] es el *histograma.*[6] Los histogramas para una imagen a color son una versión triple del histograma de brillo de una imagen de tonos de gris. Se calculan tres histogramas, uno para cada componente de color[7]. Cada histograma nos puede ayudar a determinar la distribución de brillo, el contraste y los rangos dinámicos para cada componente de color. Otra posibilidad es primero convertir la imagen a color a una imagen de tonos de gris y entonces calcular un solo histograma que represente toda la imagen.

Sirva como ejemplo la figura 25 que muestra dos versiones de la misma fotografía. La fotografía de arriba es más oscura y por lo tanto su histograma tiene más componentes del lado izquierdo, es decir, del lado de los tonos más oscuros, tendientes al negro (valor cero). La fotografía de abajo se produjo aumentándole brillo a la imagen original. Su histograma presenta una gráfica corrida hacia la derecha, mostrando así que son más abundantes los tonos claros y más escasos los tonos negros. Se observa también una fuerte presencia de color blanco (barra vertical en el extremo derecho del histograma, valor 255).

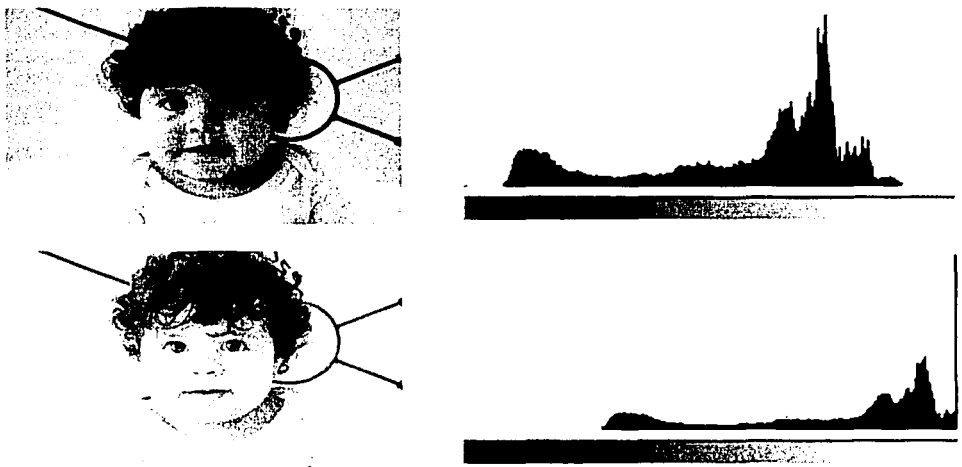


Figura 25. Fotografía con dos intensidades de brillo y sus respectivos histogramas. La escala horizontal representa los posibles valores de 0 a 255 de tonos de gris. La escala vertical representa el número de píxeles que aparecen para cada tonalidad.

3.2 Formatos gráficos

Ya hemos revisado que se pueden asignar más o menos bytes para almacenar cada píxel. Pero esto es sólo la punta del iceberg que representa la estructura de datos encargada de almacenar físicamente el archivo en una computadora. La estructura empleada puede ser tan distinta como diseñadores y programadores hay. No obstante, existe una serie de formatos gráficos que son muy populares debido a que sus aplicaciones son bastante conocidas en el mundo. En esta sección, presento una lista alfabética no exhaustiva de los formatos gráficos más comunes y sus respectivas aplicaciones. Cuando existen acrónimos en inglés sin traducción conocida, los he dejado entre comillas en el idioma original.

Formato de archivo	Descripción
BIFF	Formato XITE (3D)
BMP	Mapa de bits (MS-Windows)
BW	Formato SGI para blanco y negro
CGM	"Computer Graphics Metafile"
DRAW	Formato basado en vectores de Acorn
DWG	Archivo de AutoCAD
FAX	Estándar del Grupo 3 de faxes
DCX	Formato gráfico para fax
EPSF	Archivo encapsulado Postscript
FIG	Formato empleado por la utilería "xfig"
FITS	Sistema de transporte flexible de imagen
GIF	Formato de intercambio gráfico
GL	Formato para animaciones
ICC	Formato de impresión Kodak
IFF	Formato de intercambio de archivos
JPEG	"Joint Photographic Experts Group"
MIFF	Formato independiente de la máquina
NAP	Formato orientado a objetos
PIX	Usado por SGI
PCX	Usado por Microsoft Paintbrush
PNG	"Portable Network Graphics Specification"
PBM+	Mapa de bits portátil mejorado
PBM	Mapa de bits monocromático
PGM	Formato para escala de grises
PPM	Imágenes a todo color
PNM	Formato general de imágenes
RLE	"Run length encoded"
RAS	"Sun Raster File"
RGB	Formato de color para SGI
SLD	Transparencias de AutoDesk
SPRITE	Mapa de bits para el sistema operativo RISC de Acorn
TGA	Formato llamado "Targa"
TIFF	"Tagg Image File Format"

3.2.1 Clasificación

Una forma de clasificar los distintos formatos gráficos es en base a la compresión. La *compresión* de una imagen es una operación para reducir el espacio de almacenamiento de la misma. El objetivo es representar una imagen, con un determinado nivel de calidad requerido, pero de forma más compacta. El proceso de compresión busca extraer la información esencial de modo que la imagen pueda ser reconstruida con precisión. La información no esencial a menudo se descarta.

La compresión de imágenes generalmente se efectúa como antecedente al almacenamiento o transporte de las mismas. Esto es debido a que ambas situaciones son sensibles al tamaño de la imagen. Si se puede reducir la cantidad de datos que representan a una imagen, entonces tanto el tiempo de transferencia como el espacio de almacenamiento se reducirán. De este modo, la compresión de imágenes puede representar un gran ahorro de recursos.

Dentro de la compresión de imágenes existen dos tipos: *compresión sin pérdida* (ejemplo: formato BMP) y *compresión con pérdida* (ejemplo: formato JPG). La figura 26 muestra esta clasificación.

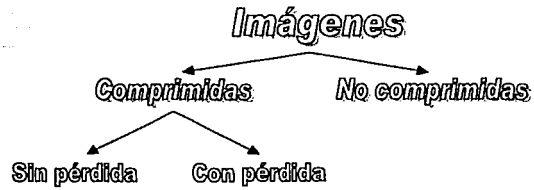


Figura 26. Clasificación de imágenes de acuerdo a la compresión

Como dije anteriormente, cada formato gráfico está determinado por una definición de estructura interna. En este capítulo, presento la descripción del formato BMP por ser éste el elegido para la aplicación esteganográfica DigiMarker. En el capítulo 4, explico las razones para haber elegido un formato de color verdadero (24 bits) no comprimido.

3.2.2 Formato BMP

El formato BMP es el formato estándar de MS-Windows. Puede contener imágenes de una amplia variedad: blanco y negro, 16 colores, 256 colores y color verdadero. La versión para 16 y 256 colores es a base de una paleta de colores que puede comprimirse con *RLE*[8]. Los archivos BMP se almacenan en un formato independiente del dispositivo. Esto quiere decir que el mapa de bits especifica el color del píxel en forma independiente al método usado para desplegar el color.

Estructura del archivo BMP. Cada archivo BMP contiene un encabezado, información del encabezado, una tabla de color (opcional) y un arreglo de bytes que definen los bits de la imagen. El *encabezado* contiene información acerca del tipo, tamaño y estructura del archivo. La *información del encabezado* especifica las dimensiones, tipo de compresión y formato de color del mapa de bits. La *tabla de color* contiene tantos elementos como colores tenga el mapa de bits. No existe tabla de color para mapas de bits de 24 bits puesto que cada píxel es representado directamente por valores rojo-verde-azul de 24 bits en el área de datos (al final del archivo). Si existe una tabla, los colores deben aparecer ordenados por grado de importancia. Esto ayuda al manejador de video a desplegar un mapa de bits en un dispositivo que no cuente con todos los colores presentes en la imagen.

La estructura, llamada BITMAPINFO se puede usar para representar una información combinada de encabezado y tabla de color. Los bits de datos, que aparecen enseguida de la tabla de color, consisten en un arreglo de bytes que representan filas consecutivas de la imagen. Cada fila (o renglón) consiste de bytes que representan los píxeles en ese renglón de la imagen, de izquierda a derecha. El número de bytes que representan un renglón depende del formato de color y el ancho de la imagen (en píxeles). En caso necesario, el renglón debe completarse con ceros para tener tamaño en múltiplos de 32 bits. Los bytes se almacenan de abajo hacia arriba, izquierda a derecha. Esto quiere decir que los primeros bytes del arreglo representan los píxeles de la esquina inferior derecha de la imagen y los últimos bytes del arreglo representan los píxeles de la esquina superior derecha.

Nombre	**Tamaño**	**Descripción**
Encabezado	14 bytes	Estructura de Windows: BITMAPFILEHEADER
Firma	2 bytes	'BM'
Tamaño archivo	4 bytes	Tamaño de archivo en bytes
Reservado	4 bytes	no usado (=0)
Offset de datos	4 bytes	Offset del archivo para inicio de datos
Información-Encabezado	40 bytes	Estructura de Windows: BITMAPINFOHEADER
Tamaño	4 bytes	Tamaño de Inf-Encabezado =40
Ancho	4 bytes	Ancho de la imagen (en píxeles)
Altura	4 bytes	Altura de la imagen (en píxeles)
Planos	2 bytes	Número de planos (=1)
Código	2 bytes	Bits por píxel 1 = paleta monocroma. NumColors = 1 4 = 4bit con paleta. NumColors = 16 8 = 8bit con paleta. NumColors = 256 16 = 16bit RGB. NumColors = 65,536 24 = 24bit RGB. NumColors = $16x10^6$
Compresión	4 bytes	Tipo de Compresión 0 = BI_RGB sin compresión 1 = BI_RLE8 compresión RLE de 8 bits 2 = BI_RLE4 compresión RLE de 4 bits
Tamaño de imagen	4 bytes	(comprimida) Tamaño de imagen Válido poner 0 si no hay compresión
Xpixels	4 bytes	resolución horizontal: píxeles/metro
Ypixels	4 bytes	resolución vertical: píxeles/metro
Colores usados	4 bytes	Número de colores usados
Colores Importantes	4 bytes	Número de colores importantes 0 = todos
Tabla de color	4 bytes * Colores usados	Presente sólo si Código <= 8 Los colores deben ordenarse por importancia
Rojo	1 byte	Intensidad de rojo
Verde	1 byte	Intensidad de verde

	Azul	1 byte	Intensidad de azul
	Reservado	1 byte	no usado (=0)
	repetir "Colores Usados" veces		
Datos de la imagen		3 bytes por pixel	Datos de cada pixel

El miembro que llamamos "Código" determina el número de bits que definen cada píxel y el número máximo de colores en la imagen. Código puede tener los siguientes valores:

Valor	Significado
1	El mapa de bits es monocromático y la tabla de colores tiene dos entradas. Cada bit en el arreglo representa un píxel. Si el bit es cero, el píxel se despliega con el color señalado en la primer entrada de la tabla de color. Si el bit es uno, el píxel adquiere el color señalado en el segundo registro de la tabla.
4	El mapa de bits tiene un máximo de 16 colores. Cada píxel en el mapa de bits se representa por un índice de 4 bits en la tabla de colores. Por ejemplo, si el primer byte en el mapa de bits es 0x1F, el byte representa dos píxeles. El primer píxel contiene el color en el segundo registro de la tabla y el segundo píxel contiene el color de la dieciseisava entrada de la tabla.
8	El mapa de bits tiene un máximo de 256 colores. Cada píxel en el mapa de bits se representa por un índice de 1 byte en la tabla de colores. Por ejemplo, si el primer byte del mapa de bits es 0x1F, el primer píxel tiene el color del registro 32 en la tabla.
24	El mapa de bits tiene un máximo de 16,777,216 colores. Cada secuencia de 3 bytes en el mapa de bits representa las intensidades relativas de rojo, verde y azul, respectivamente, para cada píxel.

Compresión.- Las versiones de Windows 3.0 y posteriores soportan compresión RLE para comprimir mapas de bits que usen 4 u 8 bits por píxel.

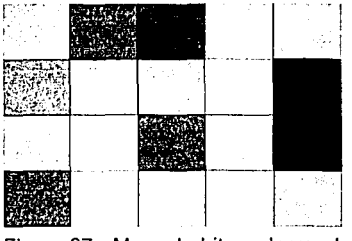
Codificación de los datos.- Dependiendo del tipo de imagen y de si hay compresión o no, se pueden dar 6 esquemas distintos de codificación de los datos que tienen los valores de los píxeles. No obstante, los 6 esquemas comparten las siguientes características:

- Los píxeles se almacenan de arriba hacia abajo y de izquierda a derecha.
- Las líneas de píxeles se completan con ceros para siempre aparecer como bloques de 32 bits.

- Para formatos no comprimidos cada línea debe tener el mismo número de bytes.
- Los índices de color comienzan en cero.
- No existe tabla de color para las imágenes de más de 256 colores.

Debido a que DigiMarker utiliza mapas de bits de color verdadero, sólo explicaré la codificación para este esquema. En la información del encabezado, Código debe valer 24 y Compresión debe valer 0. Cada 3 bytes (24 bits) se almacena un píxel. El primer byte almacena la intensidad de rojo, el segundo byte la de verde y el tercer byte la de azul. No olvidar completar con ceros los renglones en paquetes de 32 bits. Bajo este esquema no existe la tabla de colores.

Se sugiere al lector asegurarse de que ha entendido cómo se llena la estructura de un mapa de bits de color verdadero siguiendo el siguiente ejemplo:



204 153 255 **CC 99 FF**	228 4 78 **E4 4 4E**	28 21 255 **1C 15 FF**	204 153 255 **CC 99 FF**	204 153 255 **CC 99 FF**
148 157 49 **94 9D 31**	8 251 11 **8 FB B**	41 235 52 **29 EB 34**	242 228 255 **F2 E4 FF**	40 30 51 **28 1E 33**
48 192 255 **30 C0 FF**	246 237 255 **F6 ED FF**	227 7 80 **E3 7 50**	204 153 255 **CC 99 FF**	12 9 15 **C 9 F**
226 15 91 **E2 F 5B**	204 153 255 **CC 99 FF**	204 153 255 **CC 99 FF**	251 247 22 **FB F7 16**	204 153 255 **CC 99 FF**

Figura 27. Mapa de bits, color verdadero. Los números del lado derecho representan las intensidades de rojo, verde y azul, respectivamente para cada pixel. El primer renglón está en decimal; el segundo renglón (en negrillas) está en hexadecimal.

A continuación se presenta el *dump*[9] real de la imagen de la figura 27, junto con las explicaciones pertinentes.

Encabezado:

Pos. Rel. (decimal)	Contenido (hexadecimal)	Explicación
[0000]	42 4D	Firma ("BM")
[0002]	76 00 00 00	Tamaño de archivo (118 bytes)
		Encabezado (14B) + Inf-Encabezado (40B) + 4 filas de (3x5)+1B cada una.
[0006]	00 00 00 00	Reservado
[0010]	36 00 00 00	Offset (54 bytes)
		Encabezado (14B) + Inf-Encabezado (40B)

Inf-Encabezado:

Pos. Rel. (decimal)	Contenido (hexadecimal)	Explicación
[0014]	28 00 00 00	Tamaño de esta sección (40 bytes)
[0018]	05 00 00 00	Ancho de la imagen (5 píxeles)
[0022]	04 00 00 00	Alto de la imagen (4 píxeles)
[0026]	01 00	Número de planos (1)
[0028]	18 00	Código (24 = color verdadero = 3 bytes por píxel)
[0030]	00 00 00 00	Compresión (0 = sin compresión)
[0034]	40 00 00 00	Tamaño de la imagen (64 bytes) 4 filas de (3x5)+1B cada una
[0038]	23 2E 00 00	Resolución horizontal (ver nota debajo de la tabla)
[0042]	23 2E 00 00	Resolución vertical (ver nota debajo de la tabla)
[0046]	00 00 00 00	Colores usados (0 = no se usa tabla de color)
[0050]	00 00 00 00	Colores importantes (0 = todos)

Nota en cuanto al cálculo de resolución.- El cuadro de colores de la figura 27 se definió como un arreglo de 5 píxeles de ancho por 4 de alto a una resolución de 300 ppp (puntos por pulgada). La resolución 300 ppp es la mínima aceptable para un estándar de impresión. Pero la tabla requiere que se especifique en píxeles por pulgada, así que al convertir 300 ppp nos da 11,811 píxeles por metro (300 ppp x 100 cm. / 2.54 cm.). El número 11,811 es precisamente 2E23 en hexadecimal. Note que aparece primero el byte menos significativo.

Datos de la imagen:

Pos. Rel. (decimal)	Contenido (hexadecimal)	Explicación
[0054]	5B 0F E2 FF 99 CC FF 99 CC 16 F7 FB FF 99 CC 00	Primer línea de píxeles más relleno 00 (comenzando desde abajo)
[0070]	FF C0 30 FF ED F6 50 07 E3 FF 99 CC 0F 09 0C 00	Segunda línea de píxeles más relleno 00
[0086]	31 9D 94 0B FB 08 34 EB 29 FF E4 F2	Tercera línea de píxeles más relleno 00

```
          33 1E 28 00
[0102]    FF 99 CC          Cuarta línea de píxeles más relleno 00
          4E 04 E4          (es decir, la línea de arriba)
          FF 15 1C
          FF 99 CC
          FF 99 CC 00
```

Es muy importante notar el orden en que vienen almacenados los valores de los píxeles. Vea la localidad [0054] donde inicia el primer píxel (esquina inferior izquierda). En esa localidad vienen los números: `5B 0F E2`. Si miramos la figura 27 encontramos **E2 F 5B**, es decir, en el orden inverso. Efectivamente Windows guarda los datos en el orden azul, verde, rojo. Este detalle se debe tener en cuenta a la hora de programar aplicaciones.

Portabilidad del formato BMP.- Aunque el formato BMP fue inventado para la plataforma Windows, muchos programas de otras plataformas también pueden leerlo y escribirlo. Debe notarse el orden de Intel en que se escriben los enteros de 2 y 4 bytes (byte menos significativo primero). Este detalle también debe tomarse en cuenta a la hora de programar, especialmente en plataformas distintas a Intel.

Derechos de autor.- Finalmente debe notarse que el formato BMP no causa cargos por su uso. Este dato es de suma importancia cuando se desea diseñar aplicaciones que produzcan archivos gráficos, a fin de no caer en una violación a los derechos de autor de alguien más.

3.3 Implementación en Visual C

A continuación, presento el código completo y probado de las funciones más importantes de DigiMarker que tienen que ver con el manejo de los archivos de mapas de bits. He incluido comentarios que aclaran qué está pasando.

```
////////////////////////////////////////////////////////////////////////////////
// Función para cargar archivos gráficos
LPDMBITMAP CDMCore::LoadBitmapFile(LPCSTR lpszFileName, UINT nIDFileType)
{
        // Invoca diferentes funciones de acuerdo al tipo de archivo
        // En nuestro caso sólo aceptamos BMP
        switch(nIDFileType)
        {
        case FT_WINDOWS_BITMAP:
                return LoadWindowsBitmap(lpszFileName);
        }

        // No hay función disponible para manejar este tipo de archivo; regresa NULL
        m -> uErrorCode = DMERR_NOT_SUPPORTED;
        return NULL;
}

////////////////////////////////////////////////////////////////////////////////
// Función para cargar bitmaps de windows
LPDMBITMAP CDMCore::LoadWindowsBitmap(LPCSTR lpszFileName)
{
```

```
        CFile                           file;
        BITMAPFILEHEADER        bfHeader;
        BITMAPINFOHEADER        biHeader;
        LPDMBITMAP                      lpDMBitmap;
        LPBYTE                          lpBuffer, lpLineBuffer;
        LPDMRGB                         lpRGB;
        LPDMBGR                         lpBGR;
        DWORD                           dwBytesPerLine;
        DWORD                           dwImageSize;
        LONG                            i, j;

        // Abre el archivo bitmap
        CFileException ex;
        if ( !file.Open(lpszFileName, CFile::modeRead|CFile::typeBinary, &ex) )
        {
                m -> uErrorCode = DMERR_FILE_EXCEPTION;
                ex.GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
                return NULL;
        }
        try
        {
                // Lee el encabezado del archivo
                file.Read(&bfHeader, sizeof(BITMAPFILEHEADER));

                // Verifica el formato
                if ( ( bfHeader.bfType & 0xFF ) != 'B' || ( ( bfHeader.bfType >> 8 ) &
0xFF ) != 'M' )
                {
                        m -> uErrorCode = DMERR_FILE_TYPE_ERROR;
                        return NULL;
                }

                // Lee la información de encabezado
                file.Read(&biHeader, sizeof(BITMAPINFOHEADER));

                // Verifica el formato del encabezado
                if ( biHeader.biSize != sizeof(BITMAPINFOHEADER) )
                {
                        m -> uErrorCode = DMERR_FILE_TYPE_ERROR;
                        return NULL;
                }

                // Verifica compresión
                // BI_RGB quiere decir sin compresión
                if ( biHeader.biCompression != BI_RGB )
                {
                        m -> uErrorCode = DMERR_NOT_SUPPORTED;
                        return NULL;
                }
        }
        catch(CFileException *pEx)
        {
                m -> uErrorCode = DMERR_FILE_EXCEPTION;
                pEx -> GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
                pEx -> Delete();
                return NULL;
        }

        // Calcula el número de bytes por línea y el tamaño de la imagen
        switch(biHeader.biBitCount)
        {
        case 24:        // Sólo se aceptan imágenes de 24 bits
                dwBytesPerLine = biHeader.biWidth * 3;
                if ( dwBytesPerLine % 4 )
                        dwBytesPerLine = ( ( dwBytesPerLine >> 2 ) + 1 ) << 2;
                dwImageSize = dwBytesPerLine * biHeader.biHeight;
                break;
        default:
                m -> uErrorCode = DMERR_NOT_SUPPORTED;
                return NULL;
        }
```

```
        // Asignación de memoria
        // Aquí se crea un buffer para indicar el avance de carga
        if ( ( lpDMBitmap = new DMBITMAP ) == NULL ||
                ( lpDMBitmap -> lpBitmap = new DMRGB[biHeader.biWidth * biHeader.biHeight] )
== NULL ||
                ( lpBuffer = new BYTE[dwImageSize] ) == NULL ||
                ( lpLineBuffer = new BYTE[dwBytesPerLine] ) == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                return NULL;
        }
        lpDMBitmap -> width = biHeader.biWidth;
        lpDMBitmap -> height = biHeader.biHeight;

        // Inicializa el indicador de avance
        m -> fProgressFactor = 0.5f;
        m -> uProgressStart = 0;
        SetProgress(0);


        // Carga los datos y cierra el archivo
        for ( i = 0 ; i < biHeader.biHeight ; i++ )
        {
                try
                {
                        file.ReadHuge(lpLineBuffer, dwBytesPerLine);
                }
                catch(CFileException *pEx)
                {
                        m -> uErrorCode = DMERR_FILE_EXCEPTION;
                        pEx -> GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
                        pEx -> Delete();
                        delete lpDMBitmap;
                        delete lpBuffer;
                        delete lpLineBuffer;
                        return NULL;
                }
                memcpy(&lpBuffer[i * dwBytesPerLine], lpLineBuffer, dwBytesPerLine);
                // Calcula el avance
                SetProgress((i + 1) * 100 / biHeader.biHeight);
        }
        file.Close();

        // Inicializa el indicador de avance
        m -> uProgressStart = 50;

        // Convierte los datos al formato del buffer interno
        switch(biHeader.biBitCount)
        {
        case 24:
                lpRGB = lpDMBitmap -> lpBitmap;
                i = biHeader.biHeight;
                while(i--)
                {
                        lpBGR = (LPDMBGR)(&lpBuffer[dwBytesPerLine * i]);
                        j = biHeader.biWidth;
                        while(j--)
                        {
                                lpRGB -> r = lpBGR -> r;
                                lpRGB -> g = lpBGR -> g;
                                lpRGB -> b = lpBGR -> b;
                                lpRGB++;
                                lpBGR++;
                        }

                        // Calcula el avance
                        SetProgress(100 - i * 100 / biHeader.biHeight);
                }
                break;
        default:
```

```
                m -> uErrorCode = DMERR_NOT_SUPPORTED;
                delete lpBuffer;
                delete lpLineBuffer;
                delete lpDMBitmap;
                return NULL;
        }

        // Libera la memoria temporal
        delete lpBuffer;
        delete lpLineBuffer;

        return lpDMBitmap;
}

////////////////////////////////////////////////////////////////////////////////////////////
// Función para generar un mapa de bits Windows a partir de nuestro buffer interno
LPWINBITMAP CDMCore::GenerateWindowsBitmap(LPDMBITMAP lpDMBitmap)
{
        LPWINBITMAP             lpWinBitmap;
        DWORD                   dwBytesPerLine;
        DWORD                   dwImageSize;
        LPDMRGB                 lpRGB;
        LPDMBGR                 lpBGR;
        LONG                    i, j;
        LPBYTE                  lpBuffer;

        ASSERT(lpDMBitmap);

        // Asignación de memoria
        if ( ( lpWinBitmap = new WINBITMAP ) == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                return NULL;
        }

        // Calcula el número de bytes por línea y el tamaño de la imagen
        dwBytesPerLine = lpDMBitmap -> width * 3;
        if ( dwBytesPerLine % 4 )
                dwBytesPerLine = ( ( dwBytesPerLine >> 2 ) + 1 ) << 2;
        dwImageSize = dwBytesPerLine * lpDMBitmap -> height;

        // Asignación de memoria para el mapa de bits
        if ( ( lpWinBitmap -> lpBuffer = new BYTE[dwImageSize] ) == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                return NULL;
        }

        // Relleno de información necesaria de encabezado
        lpWinBitmap -> biHeader.biBitCount = 24;
        lpWinBitmap -> biHeader.biClrImportant = 0;
        lpWinBitmap -> biHeader.biClrUsed = 0;
        lpWinBitmap -> biHeader.biCompression = BI_RGB;
        lpWinBitmap -> biHeader.biHeight = lpDMBitmap -> height;
        lpWinBitmap -> biHeader.biPlanes = 1;
        lpWinBitmap -> biHeader.biSize = sizeof(BITMAPINFOHEADER);
        lpWinBitmap -> biHeader.biSizeImage = dwImageSize;
        lpWinBitmap -> biHeader.biWidth = lpDMBitmap -> width;
        lpWinBitmap -> biHeader.biXPelsPerMeter = 3780;
        lpWinBitmap -> biHeader.biYPelsPerMeter = 3780;

        // Convierte el buffer interno a un mapa de bits de windows
        lpRGB = lpDMBitmap -> lpBitmap;
        lpBuffer = (LPBYTE)lpWinBitmap -> lpBuffer;
        i = lpDMBitmap -> height;
        while(i--)
        {
                lpBGR = (LPDMBGR)(&lpBuffer[dwBytesPerLine * i]);
                j = lpDMBitmap -> width;
                while(j--)
                {
```

```
                    lpBGR -> r = lpRGB -> r;
                    lpBGR -> g = lpRGB -> g;
                    lpBGR -> b = lpRGB -> b;
                    lpBGR++;
                    lpRGB++;
                }

                // Pone ceros en el resto de bytes
                memset(lpBGR, 0, dwBytesPerLine - lpDMBitmap -> width * 3);
        }

        return lpWinBitmap;
}

////////////////////////////////////////////////////////////////////////////////
// Función para liberar el bufffer interno
void CDMCore::ReleaseBitmap(LPDMBITMAP lpDMBitmap)
{
        ASSERT(lpDMBitmap);
        delete lpDMBitmap -> lpBitmap;
        delete lpDMBitmap;
}

////////////////////////////////////////////////////////////////////////////////
// Función para liberar el mapa de bits de Windows
void CDMCore::ReleaseBitmap(LPWINBITMAP lpWinBitmap)
{
        ASSERT(lpWinBitmap);
        delete lpWinBitmap -> lpBuffer;
        delete lpWinBitmap;
}
```

[1] Del inglés *picture element*.
[2] Del inglés *Video Graphics Adapter*.
[3] Del inglés *Super VGA*.
[4] Del inglés *Ultra VGA*.
[5] El estegoanálisis es el esfuerzo analítico para descubrir si un medio digital (cualquier tipo de archivo, pero muy comúnmente archivos gráficos) contiene información esteganográfica en su interior. El estegoanálisis es una disciplina muy reciente, quizá concentrándose en la India las investigaciones más notables.
[6] Un histograma se construye a partir de una tabla de frecuencias. En los histogramas usados en esta tesis, el eje horizontal va de 0 a 255 y representa la intensidad de brillo. Para cada uno de estos valores, se alza una línea vertical indicando el número de píxeles en la imagen con esa intensidad de brillo. Una imagen muy clara, con muchos píxeles en blanco, estará cargada hacia la derecha y viceversa.
[7] Baxes, Gregory. *Digital Image Processing, Principles and Applications*, Wiley, 1994, pp. 63-65
[8] Del inglés *Run Length Encoding*. Técnica de compresión que aprovecha el hecho de que, estadísticamente, muchos píxeles vecinos de una imagen tienden a tener el mismo valor de brillo. Esta forma de redundancia se puede reducir de tamaño agrupando los píxeles de brillo idéntico en códigos únicos.
[9] Vaciado literal de una porción de la memoria de la computadora. Normalmente se presenta sin ningún formato y suele aparecer en hexadecimal.

4

Método de incrustación

"Aún la más pequeña de las monedas puede ocultar la estrella más grande del universo, si la acercas lo suficiente a tu ojo"

Samuel Grafton

4.1 Temas de esteganografía

Tal como expresé en el capítulo 1, el objetivo de la esteganografía es ocultar el hecho de que se lleva a cabo una comunicación entre dos o más partes. Se trata de evitar levantar sospechas sobre la transmisión de un mensaje. Al igual que el encriptamiento, es un medio para proveer privacidad, pero a diferencia de aquel, que revuelve y modifica el mensaje hasta hacerlo ininteligible, la esteganografía ni siquiera permite ver el mensaje. Evidentemente una forma robusta de implementar un medio de comunicación sería primero encriptar el mensaje y luego usar una técnica esteganográfica para ocultar su presencia; este método provee un nivel de seguridad más alto que el usar una sola técnica.

La esteganografía se puede aplicar de diversas formas a los medios digitales. Una forma atractiva es ocultar información –disfrazada como ruido ambiental- en una pista de audio. Más popular aún sería ocultar la información en imágenes tales como fotografías o dibujos. De acuerdo a lo expuesto en el capítulo 3, es posible prescindir de los planos de bits menos significativos de una imagen digital sin alterar notoriamente su apariencia. La técnica esteganográfica basada en este principio se llama precisamente LSB.

La información puede ocultarse de formas diversas en las imágenes digitales. Considere las siguientes ideas:

- Usar los bits menos significativos (LSB)
- Seleccionar áreas ruidosas de la imagen y en su interior incrustar el mensaje
- Detectar siluetas en la imagen e incrustar el mensaje en los bordes de alto contraste
- Distribuir uniformemente el mensaje dentro de la imagen
- Distribuir aleatoriamente el mensaje dentro de la imagen
- Alterar en mayor grado los colores que sean menos perceptibles al ojo humano
- Estadísticamente procurar modificar la imagen lo menos posible (se puede recurrir a un método de prueba y error)
- Transformar la imagen al dominio de la frecuencia y ahí incrustar la información; luego regresarla al dominio del espacio.

4.1.1 Caso de estudio: Método ABCDE

Una forma interesante es combinar dos o más de las ideas anteriores a fin de producir un método mejor. De la evaluación presentada en el capítulo 1 sobre diferentes aplicaciones esteganográficas, se puede observar que BPCS tiene una capacidad de incrustación muy alta. El método BPCS utiliza los bits menos significativos, pero además selecciona las áreas más ruidosas de la imagen. Así que nos puede servir para ilustrar una combinación de técnicas. En los siguientes párrafos se explica la metodología ABCDE[1] que a su vez está basada en BPCS.

Debido a que nuestra vista no es sensible a pequeños cambios en regiones ruidosas, no será posible darnos cuenta si los píxeles de una imagen ruidosa son cambiados por otros píxeles cuando éstos también sean ruidosos. ABCDE está basado en este principio.

Esta forma de incrustación funciona bien si podemos localizar correctamente las regiones ruidosas de una portadora. La idea es dividir cada plano de bits de la imagen portadora en pequeños bloques binarios y cuadrados de píxeles y encontrar bloques que tengan patrones complejos en blanco y negro. Si el bloque binario tiene un patrón complejo en blanco y negro, podemos considerar que el bloque se encuentra en una región ruidosa.

Es necesario calcular una medida de complejidad para determinar si un bloque se puede considerar complejo o no. Esta medida se define como la longitud total de bordes blanco y negro dentro de un bloque. Si el valor de esta medida de complejidad para bordes blanco y negro es mayor que cierto umbral dado, entonces se considera que el bloque en cuestión es *complejo*.

También se puede considerar al archivo de datos como un flujo de bloques binarios. En BPCS la incrustación de datos se lleva a cabo reemplazando los bloques complejos en los planos de bits de una imagen portadora por los bloques que se obtienen del archivo del mensaje a incrustar. Cuando consideramos el archivo del mensaje como una secuencia de bloques binarios, algunos de ellos pueden ser simples (es decir, no complejos). Si incrustamos un bloque simple, tal como viene, dentro de una región ruidosa, el cambio será notorio fallando así en el propósito esteganográfico (vea la figura 28). Así que el archivo del mensaje deberá primero ser convertido a una secuencia de bloques complejos.

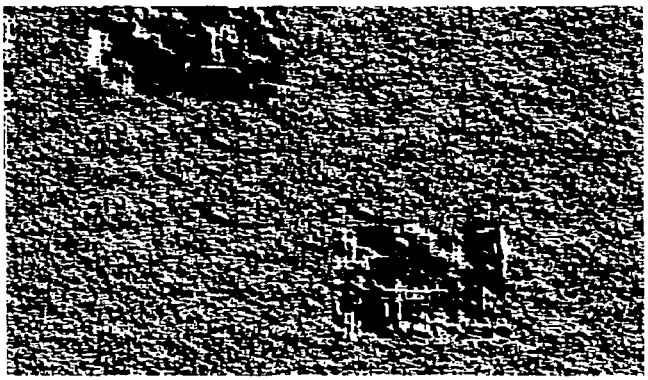


Figura 28. Incrustación de bloques simples en región ruidosa. Como los bloques incrustados no son complejos, su presencia es evidente.

Para convertir los bloques simples en bloques complejos se aplica una operación llamada *conjugación*. A fin de extraer el archivo de mensaje de la imagen portadora, debemos saber cuáles bloques del mensaje fueron conjugados y cuáles no. Esta información se almacena en un mapa de conjugación. Si el mapa de conjugación no es suficientemente complejo también debe conjugarse.

Aunque la medida de complejidad para bordes blanco y negro es apropiada para clasificar bloques como simples o complejos, no siempre es aplicable. Por ejemplo, los bloques con patrones periódicos (como tiras o tableros de ajedrez, vea la figura 29) serían clasificados como complejos por esta medida. Tales bloques, sin embargo, no son apropiados para incrustar datos ya que al reemplazarlos por bloques ruidosos el patrón periódico desaparecería, produciendo un cambio visual notable. Esta medida también puede indicar que los bloques en la frontera de una región ruidosa son complejos. Incrustar datos en tales bloques puede resultar en cambios notables a la imagen portadora.

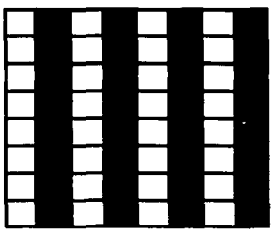


Figura 29a

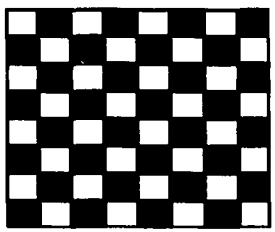


Figura 29b

Bloques que no se deben considerar complejos.

Así que es necesario introducir otras medidas de complejidad para distinguir con más precisión bloques complejos de bloques simples. Una medida es calculada a partir de la

distribución de píxeles en columnas y renglones. A esta medida se le conoce como *irregularidad longitudinal*. La otra medida se calcula de la distribución de bordes blanco y negro entre renglones y columnas adyacentes. A esta medida se le conoce como *ruido de borde*. Ambas medidas se usan simultáneamente.

La irregularidad longitudinal se usa para evaluar la no uniformidad de patrones blanco y negro dentro de un bloque. Esta medida evita que bloques de patrón uniforme sean aceptados para incrustar un mensaje. El ruido de borde se usa adicionalmente para verificar si un bloque contiene muchos bordes blanco y negro y si están uniformemente distribuidos dentro del bloque. Con esta medida se puede evitar usar bloques que no estén completamente contenidos en regiones ruidosas.

Un bloque es considerado complejo si su irregularidad longitudinal y su ruido de borde son mayores que ciertos valores de umbral dados. Los valores de umbral pueden ser especificados independientemente para cada plano de bits. Este ajuste de parámetros permite aumentar la capacidad de incrustación, sin sacrificar la calidad de la imagen portadora. A cambio, requiere más tiempo de procesamiento.

En ABCDE, la incrustación de datos se lleva a cabo de la siguiente forma: el archivo de mensaje se considera como una secuencia de bloques binarios. Todos los bloques se convierten a bloques complejos. Aunque la secuencia de bloques originales seguramente contendrá tanto bloques complejos como bloques simples, todos los bloques se convierten por igual a bloques complejos. De esta forma se puede evitar el incrustar información del mapa de conjugación mencionado anteriormente. Esto también simplifica el algoritmo empleado.

Los valores de umbral pueden ser incrustados junto con el archivo del mensaje o bien pueden ser manejados externamente. Cuando se decide no incrustar estos valores (parámetros para determinar qué bloques son complejos), se deben especificar con toda precisión en el momento de extraer el mensaje incrustado. Es prácticamente imposible extraer correctamente la información sin conocer esos valores. A esta forma se le conoce como *modo seguro de incrustación* y es muy útil en aplicaciones esteganográficas. Por otro lado, cuando los parámetros son incrustados junto con el mensaje, no se necesita conocer nada excepto el tamaño del bloque a procesar. Si se usa un tamaño de bloque por omisión (por ejemplo 64 x 64 bits), cualquiera puede extraer el archivo incrustado. A esta forma se le conoce como *modo autocontenido de incrustación* y es útil para aplicaciones de manejo de datos.

La región ruidosa de una imagen portadora se localiza en cada plano de bits como una pequeña área de píxeles con patrones ruidosos. Cada plano de bits se divide en pequeños bloques cuadrados, como se ilustra en la figura 30. Se dice que un bloque binario de píxeles está en una región ruidosa si tiene un patrón blanco y negro complejo. Sólo este tipo de bloques complejos se puede usar para incrustar información. Además deberán usarse primero los bloques complejos encontrados en el plano de bit cero.

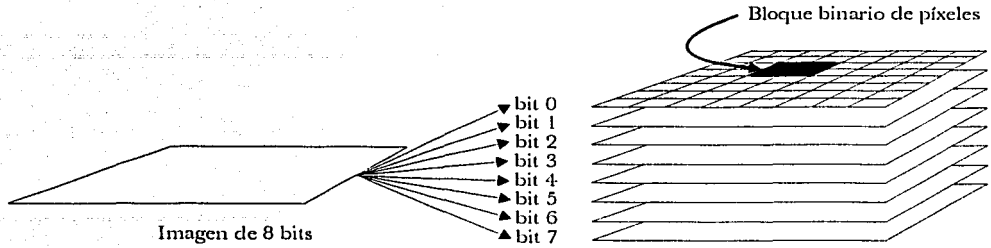


Figura 30. Bloque binario de píxeles en plano de bits extraído de una imagen de 8 bits.

Los bloques de una imagen portadora son examinados uno por uno, comenzando con el plano de bit cero, luego el uno y así sucesivamente hasta terminar en el plano más alto. El archivo de mensaje es incrustado, bloque por bloque, conforme se van encontrando bloques complejos dentro de los planos de bits.

El problema ahora consiste en determinar si un bloque es simple o complejo. Se dice que un bloque es *simple* cuando está completa o casi completamente lleno de blanco o de negro. La figura 31a muestra un bloque de 8 x 8 que se ve simple. Tiene (8-1) x 8 x 2 = 112 bordes de píxel en su interior, pero sólo 9 de ellos están entre píxeles blanco y negro. Así que se puede considerar un bloque como simple cuando tiene pocos bordes entre blanco y negro en su interior. Los otros bordes están entre píxeles del mismo color, ya sea entre píxeles blancos o entre píxeles negros.

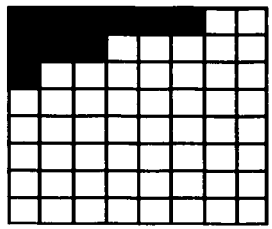


Figura 31a. Un bloque simple

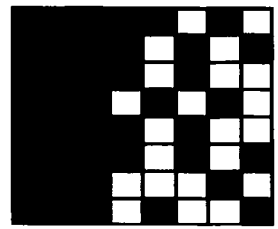


Figura 31b. Bloque en el límite de una zona ruidosa.

Supóngase que k de M bordes de píxel están entre píxeles blanco y negro dentro de un bloque. La medida de complejidad está dada por:

$$\alpha = k / M$$

Por ejemplo, $\alpha = 9/112$ para el bloque de la figura 31a. Si α es grande para un bloque determinado, el bloque tiene muchos bordes entre blanco y negro en su interior y por lo tanto se le puede considerar complejo. Por el contrario, si α es pequeño, el bloque será considerado simple. El rango de esta medida es [0,1]. Es necesario definir un valor de umbral, α_0, para

distinguir bloques simples de complejos. Un bloque B se considera complejo si $\alpha(B) \geq \alpha_0$. La figura 32 muestra un bloque simple y uno complejo respecto a α. Los valores de α son 20/112 (izquierda) y 72/112 (derecha).

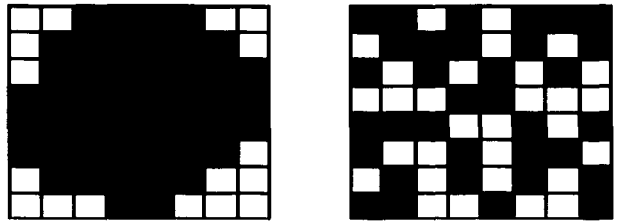


Figura 32. Un bloque simple y un bloque complejo respecto a α.

Esta medida de complejidad de bordes blancos y negros es fácil de entender y normalmente funciona bien para clasificar los bloques en simples y complejos. Sin embargo, esta medida no siempre es aplicable ya que en casos como los presentados en la figura 29, aunque esos bloques presentan una α muy grande, no necesariamente son complejos. Por ejemplo, la figura 29b que presenta el patrón de un tablero de ajedrez, tiene un $\alpha = 1$, que es el valor máximo posible para α. Sin embargo, este bloque tiene un patrón periódico regular y no puede ser considerado como complejo. Si reemplazamos ese bloque por un bloque ruidoso, el resultado será interrumpir el patrón periódico regular resultando en un cambio notable para la imagen portadora.

Una solución que viene a la mente es añadir un límite superior α_1 (además de α_0) y solamente aceptar los bloques que tengan $\alpha \in [\alpha_0, \alpha_1]$. Desafortunadamente, aun si se adopta este criterio, todavía es posible encontrar bloques como el mostrado en la figura 29a para el que $\alpha = 0.5$. Si excluimos 0.5 del rango $[\alpha_0, \alpha_1]$, entonces perderemos muchos bloques complejos.

Se presenta otro problema: debido a que los planos de bits de una imagen portadora se dividen regularmente en bloques sin importar su contenido, algunos de los bloques pueden caer en la frontera entre una región ruidosa y una región informativa. La figura 31b muestra un bloque en esta situación. La medida de complejidad de bordes blancos y negros α puede catalogar a ese bloque como complejo. No obstante, si bloques de ese tipo se usan para incrustar información, muy probablemente crecerá la región ruidosa, dejando como resultado modificaciones evidentes en la imagen portadora.

Así se concluye que α no siempre será suficiente para determinar si un bloque es complejo. Si la distribución de píxeles blancos y negros dentro de un bloque tiene periodicidad regular, ese bloque no deberá ser usado para incrustar información; es decir, dicho tipo de bloques debe ser considerado simple. Si la irregularidad longitudinal de un bloque es relativamente grande, el bloque no puede tener un patrón regular blanco y negro ni tampoco puede ser completamente blanco o negro. La irregularidad longitudinal se calcula en base al histograma de las longitudes de píxeles blancos y negros sobre un renglón o columna. Sea la secuencia binaria de píxeles mostrada en la figura 33a. Vea que consiste en:

- una secuencia de tres píxeles blancos
- una secuencia de un píxel negro
- una secuencia de dos píxeles blancos
- una secuencia de dos píxeles negros

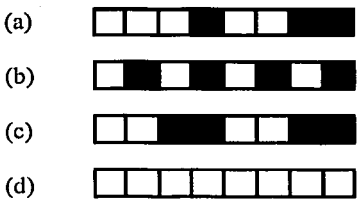


Figura 33. Secuencias binarias de píxeles.

De la figura 33a,

$$h[1] = 1,\ h[2] = 2,\ h[3] = 1$$

donde $h[i]$ es la frecuencia de secuencias de i píxeles blancos o negros. La irregularidad de una secuencia binaria de píxeles se calcula así:

$$h_s = \sum_{i=1}^{n} p_i \log_2 p_i \qquad \text{donde} \qquad p_i = \frac{h[i]}{\sum_{j=1}^{n} h[j]}$$

donde n es la secuencia más larga posible, por ejemplo, la longitud de la secuencia total de píxeles dentro del bloque. h_s evalúa la distribución longitudinal en la secuencia binaria. Si una secuencia sólo contiene corridas de la misma longitud, revelará una periodicidad regular. En tal caso h_s se hace cero. Por ejemplo, los valores de h_s son cero para todas las secuencias mostradas en las figuras 33b, 33c y 33d. Por el contrario, si una secuencia contiene corridas de varias longitudes, no tendrá una periodicidad definida y h_s se incrementará. De aquí en adelante consideremos que se normalizan los valores de h_s a [0,1].

Sea n x n el tamaño de bloque. Sean r_i y c_j respectivamente el renglón i-ésimo y la columna j-ésima de un bloque. La irregularidad longitudinal β de un bloque ahora la definimos como:

$$\beta = \min\{\overline{H_s(r)}, \overline{H_s(c)}\}$$

donde

$$H_s(r) = \{h_s(r_0), \ldots\ldots, h_s(r_{n-1})\}$$
$$H_s(c) = \{h_s(c_0), \ldots\ldots, h_s(c_{n-1})\}$$

y $\overline{X}$ es el promedio de todos los elementos de X.

Como se puede ver en esta definición, la misma función se aplica tanto a renglones como a columnas del bloque. Se calculan primero los valores de h_s para cada renglón y columna. Luego se calculan sus promedios en cada dirección. El menor de esos promedios se considera el valor de la irregularidad longitudinal β.

Puesto que el patrón blanco y negro de un bloque puede ser complejo en una sola dirección (si es complejo en las columnas no lo es en los renglones y viceversa), la dirección es significativa en esta definición. La figura 34 muestra un ejemplo de esto. Es complejo en relación a los renglones pero no en relación a las columnas. Tal bloque en su totalidad no se debe considerar complejo, es decir, no es apropiado para incrustar información. El rango de la irregularidad longitudinal β es [0,1]. Si β es suficientemente grande para un bloque dado, el bloque se considerará complejo, de lo contrario se catalogará como simple.

Supóngase ahora que la medida de complejidad de bordes blancos y negros α es más grande que un valor de umbral α_0. El valor de α para los bloques de las figuras 29a y 29b es de 0.5 y 1.0 respectivamente. Estos valores son mayores que los valores de umbral comúnmente usados en BPCS. Así que ambos bloques se consideran complejos en función de α. Por otro lado, la irregularidad longitudinal β es 0 para ambos. Así que se consideran bloques simples en función de β. Similarmente, los bloques con patrones periódicos o que son en su mayoría blancos o negros tendrán un valor β muy pequeño.

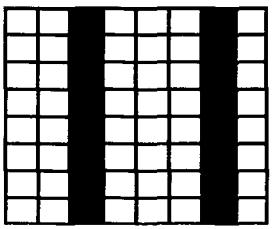
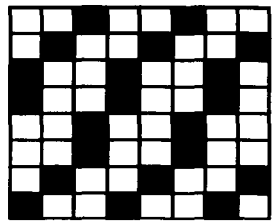
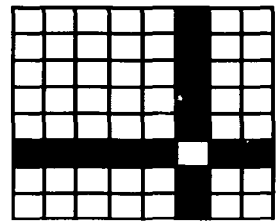


Figura 34. Bloque con patrón direccional.

La irregularidad longitudinal β por sí sola a menudo funciona bien para determinar la complejidad de un bloque. Desafortunadamente, puede fallar en rechazar algunos bloques simples. La figura 35 muestra un par de ejemplos. Aunque se ven simples, sus medidas de irregularidad longitudinal β no son pequeñas. Esta clase de excepciones puede ocurrir, ya que con esta medida de complejidad no podemos evaluar similitudes entre renglones o columnas adyacentes.



$\beta = 0.745$



$\beta = 0.694$

Figura 35. Bloques que no son complejos pero que tienen irregularidad longitudinal grande.

Esta falla se puede evitar usando una nueva medida de complejidad: *ruido en borde*. Aun si la irregularidad longitudinal de un bloque es grande, el ruido en borde se hace pequeño si renglones o columnas adyacentes son similares entre sí. De esta forma será posible rechazar ese bloque, al combinar ambas medidas de complejidad.

Si incrustamos un archivo en bloques en la frontera de regiones ruidosas con regiones informativas de una imagen portadora, la región ruidosa crecerá con la incrustación. Como consecuencia de esto, la portadora se modificará notablemente. El ruido en borde es otra medida de complejidad para verificar si hay muchos bordes de píxeles blancos y negros en un bloque y si están uniformemente distribuidos dentro del mismo. Si el ruido en borde de un bloque es suficientemente grande, entonces no es posible que esté en la frontera de una región ruidosa con una región informativa. Esto también indicaría que no hay renglones o columnas adyacentes similares entre sí.

La medida de complejidad *ruido en borde* se calcula basada en las diferencias entre secuencias adyacentes de píxeles dentro de un bloque. Sea el tamaño del bloque n x n (n>1). Sean r_i y c_j respectivamente el renglón i-ésimo y la columna j-ésima de un bloque. El ruido de borde γ de un bloque está definido:

$$\gamma = \frac{1}{n-1}\min\{E_f(P_x(r)), E_f(P_x(c))\}$$

donde

$$P_x(r) = \{\rho(r_0 \oplus r_1), \ldots\ldots, \rho(r_{n-2} \oplus r_{n-1})\}$$
$$P_x(c) = \{\rho(c_0 \oplus c_1), \ldots\ldots, \rho(c_{n-2} \oplus c_{n-1})\}$$

El operador $\oplus$ denota la operación OR-exclusiva, $\rho(x)$ es el número de unos en una secuencia binaria x y

$$E_f(X) = (1.0 - \frac{V(X)}{\max\{V(X)\}}) \cdot \overline{X}$$

donde V(X) es la varianza de X y $\overline{X}$ es el promedio de X.

Esta definición de ruido de borde γ es similar a la de β. Se cuenta el número de bordes de píxeles blancos y negros para cada par de renglones y columnas adyacentes. Se representan por el conjunto $P_x(r)$ para los renglones y $P_x(c)$ para las columnas. Luego se calcula un promedio ponderado E_f entre estos conjuntos.

El ruido de borde γ nos permite verificar si muchos bordes de píxeles blancos y negros están uniformemente distribuidos sobre un bloque tanto en las direcciones horizontales como verticales. Si el bloque tiene bordes de píxeles blancos y negros sobre una sola dirección como la mostrada en la figura 34, el ruido de borde se hace pequeño. Si el bloque se encuentra en la frontera entre una región ruidosa y una región informativa, la varianza de P_x se hace grande. Como resultado, el ruido de borde γ es pequeño para tal bloque.

Por ejemplo, γ de la figura 31b es tan pequeño como 0.212. Por otro lado, el valor de α para el mismo bloque es 0.446, que es mayor que el valor de umbral normalmente usado en BPCS. También se ha encontrado que los bloques presentados en la figura 35 tienen irregularidades longitudinales β más bien grandes, pero al calcular su ruido de borde es tan pequeño como 0.294 y 0.048 de izquierda a derecha. Así que no se consideran bloques complejos en función a su ruido de borde γ.

Resumiendo, se puede decir que es posible incrustar secretamente un archivo de mensaje en una imagen portadora si los bloques complejos se localizan apropiadamente en cada plano de bits de la imagen portadora. Los bloques complejos de la imagen portadora se identifican al calcular su irregularidad longitudinal β y su ruido de borde γ. Un bloque B es considerado complejo si cumple la siguiente condición:

$$\beta(B) \geq \beta_u \quad \text{y} \quad \gamma(B) \geq \gamma_u$$

donde β_u y γ_u son los valores de umbral. Lo ideal es determinar estos valores de umbral independientemente para cada plano de bits. Es posible agrandar la capacidad de incrustación sin sacrificar la calidad de la imagen portadora al ajustar estos valores de umbral. No obstante, al programar una aplicación que utilice este método, se deberá proveer unos valores de umbral por omisión. Estos valores deberán calcularse empíricamente.

Se deben especificar valores de umbral grandes para los planos de bits más altos, dado que cambios excesivos en los planos altos de bits modifican inaceptablemente la imagen. Por el otro lado, los planos de bits bajos se pueden usar casi libremente puesto que casi no alteran la imagen y sí permiten aumentar considerablemente la capacidad de incrustación. En general se puede decir que no es posible notar cambio alguno al modificar el plano LSB de una imagen. Es por esta razón que el método LSB es tan efectivo en las aplicaciones esteganográficas. Pero también es factible modificar planos de bits ligeramente superiores sin

producir grandes cambios, aunque estas modificaciones deberán hacerse con mucho mayor cuidado para no hacerlas aparentes.

4.1.2 Otros enfoques

Otra posibilidad sería distribuir el mensaje más uniformemente (en vez de bloques cuadrados) en la imagen, a fin de que las posibles alteraciones parezcan ruido natural en lugar de una alteración ajena a la imagen.

Una posibilidad más es colocar el mensaje (dividido en fragmentos muy pequeños) en localidades aleatorias de la portadora. Este mismo concepto era empleado por los chinos antiguos: el receptor y el emisor tenían copia de una máscara de papel con un cierto número de agujeros situados al azar. Cada agujero dejaba ver un solo ideograma. El emisor colocaba su máscara sobre una hoja de papel, escribía el mensaje a través de los agujeros, luego quitaba la máscara y luego redactaba un texto portador que se acoplara de manera natural al mensaje secreto inicialmente escrito. El receptor podía leer el mensaje oculto de forma inmediata simplemente colocando su copia de máscara sobre la hoja recibida. Con las herramientas computacionales disponibles actualmente, no es difícil implementar una técnica similar usando máscaras y portadoras digitales.

4.1.3 Método DigiMarker

Revisemos ahora con más detalle el método LSB. Para almacenar un mensaje en los bits menos significativos de una imagen de 24 bits, se pueden utilizar 3 bits por píxel, tomando prestado un bit para cada componente de color. De aquí se puede ver fácilmente que en una imagen VGA será posible almacenar unos

$$\frac{640 \text{ píxeles horizontal} \times 480 \text{ píxeles vertical} \times 3 \text{ Bytes por píxel}}{8 \text{ bits por Byte}} = 115{,}200 \text{ Bytes}$$

115,200 bytes representan el 12.5% del tamaño de la imagen portadora. Se dice entonces que con este método se tiene un grado de ocupación de 12.5% Evidentemente, un objetivo de las técnicas esteganográficas será incrementar el grado de ocupación lo más posible, pero sin sacrificar la seguridad; es decir, sin presentar evidencias de la incrustación de información. A menudo, esta medida será encontrada como resultado conjunto de técnicas computacionales y de apreciación subjetiva.

¿Cuánto se modifica una imagen al insertar información en los LSB? A primera vista, podríamos decir que se modifica uno de cada 8 bits. De modo que para insertar una letra (que se almacena en un byte) se modificarán 64 bits, es decir, 8 bytes, o sea poco menos de tres píxeles. Afortunadamente, en la realidad, por estadística, las modificaciones hechas

generalmente son menores. Tomemos por ejemplo la letra "B" que se ocultará en tres píxeles. Sean los tres píxeles originales los siguientes:

	Rojo	*Verde*	*Azul*
píxel 1	01011010	11011101	00011000
píxel 2	00110111	10001001	10001011
píxel 3	10001001	00100110	10101001

El valor binario para la letra "B" es 10000100. Al insertar la letra "B" en los tres píxeles (de arriba abajo e izquierda a derecha en la tabla), los píxeles quedarían así:

	Rojo	*Verde*	*Azul*
píxel 1	0101101**1**	1101110**0**	00011000
píxel 2	0011011**0**	1000100**0**	10001011
píxel 3	1000100**0**	00100110	10101001

De los 8 bits que se emplearon, sólo 5 requirieron ser modificados. Estadísticamente, se podría esperar que en promedio sólo se requiera modificar el 50% de los bits empleados. Esta característica hace atractivo el método LSB. Además de esto, también es posible almacenar información en el plano de bit 1, e incluso en el plano de bit 2. El cuidado que se debe tener en estos casos es que la apariencia de la imagen no sea sospechosa de ser portadora de información oculta. La ventaja, por el contrario, es que se aumenta enormemente la capacidad de incrustación.

Una característica del método LSB es su vulnerabilidad ante modificaciones en la imagen portadora. La aplicación de prácticamente cualquier transformación (rotación, amplificación, reducción, o cualquier clase de filtro) dañará sensiblemente el mensaje transportado; incluso es probable que sea imposible recuperarlo en absoluto. ¿Es esta una característica deseable o indeseable? Depende del punto de vista. Si se trata de seguridad en la información transportada, ésta es una buena característica pues alertaría de inmediato de un posible ataque o intento de modificación. Si se trata de fortaleza contra ataques a la portadora, entonces es una característica indeseable pero que es inherente a este método. En opinión de este autor, para fines de esteganografía, la vulnerabilidad de la portadora es deseable y para fines de marca de agua se prefiere la robustez de la portadora. Aquí se observa entonces otra distinción conceptual entre marca de agua y esteganografía.

Si se usan 4 planos de bits, ¿qué tanto se altera una imagen? Bueno, depende de muchos factores, como los píxeles vecinos, si la zona es ruidosa o no, etc. Pero si analizamos píxel por píxel tendremos una idea de qué tanto se altera el color. Vea el siguiente ejemplo donde se han cambiado 4 de los 8 bits originales:

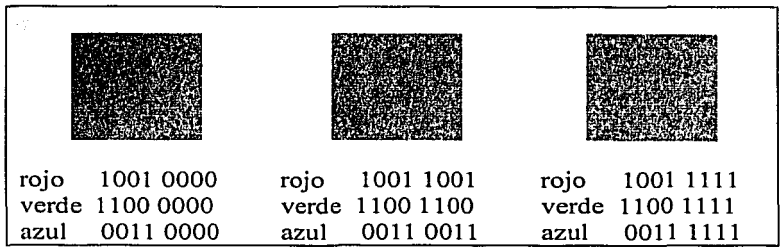


Figura 36. Alteración de color al modificar 4 de cada 8 bits.

En la figura 36, se puede tomar cualquier cuadro como el cuadro original; sea considerado, por ejemplo, el cuadro de la izquierda como la imagen original. El cuadro de la derecha representa entonces la máxima modificación posible que puede sufrir si se utilizan 4 planos de bits para incrustar información. Como podrá apreciar el lector, la diferencia visual entre el color del cuadro de la izquierda y el color del cuadro de la derecha es muy pequeña. Aunque es perfectamente posible que en ocasiones se requiera modificar todos los bits empleados (tal como se acaba de ilustrar), estadísticamente se puede prever que en general se espera una modificación de sólo el 50% de los bits. De modo que en promedio el caso encontrado sería tomar como original el cuadro izquierdo y como modificado el cuadro del centro. Evidentemente la diferencia visual entre estos dos cuadros es todavía menor, lo cual es una fortaleza del método LSB.

¿Cuánta información se puede incrustar con LSB? Dependiendo de la imagen, será posible ocupar más o menos planos de bits. Una imagen con muchas zonas de alta frecuencia podrá ser más manipulada que una imagen con áreas de baja frecuencia (como colores sólidos). La figura 37 ilustra este principio.

Figura 37a. Imagen original con zonas de distintos niveles de ruido

Figura 37b. Imagen con incrustación exagerada para notar las zonas más alteradas.

Aunque más adelante detallaré las características e implementación de DigiMarker, por ahora puedo adelantar un resumen de su principio de operación:

- Trabajar con imágenes mapa de bits no comprimidos, color verdadero (24 bits)
- Seleccionar regiones más aptas para incrustar la información secreta. Se pueden considerar aspectos como contraste, cambios de frecuencia, nivel de ruido, nivel de brillo, etc.
- Comprimir la información
- Encriptar la información
- Incrustar la información de acuerdo a los principios de LSB pero con la posibilidad dinámica de seleccionar cuántos bits emplear
- La incrustación deberá realizarse de modo que si alguna modificación es aparente, tenga la forma de ruido natural, producto de un proceso estocástico
- Proveer una interfaz que no eche a perder todo el esfuerzo técnico del método propuesto.[2]

4.1.4 Requisitos para una buena incrustación

En un sistema "perfecto", una portadora ordinaria (por ejemplo, una imagen digital) no se debe poder distinguir de una portadora esteganográfica ni por el ojo humano ni por técnica computacional alguna. Así que la seguridad de una comunicación invisible en el caso de imágenes digitales radica principalmente en la inhabilidad para distinguir una foto normal de una foto empleada como portadora esteganográfica. Por esta misma razón, en la práctica no todos los datos de la portadora se pueden emplear para portar parte del mensaje. Este hecho requiere que la portadora tenga suficiente cantidad de información redundante que en su momento pueda ser reemplazada por información confidencial. Por ejemplo, debido a errores de medición y a imperfecciones en los medios impresos, cualquier serie de datos que sean resultado de algún proceso de escaneo contendrá un componente estocástico llamado "ruido". Esos elementos aleatorios pueden ser usados para la incrustación de información secreta. De hecho ahora se sabe que los portadores ruidosos son muy convenientes para ser usados como portadores esteganográficos.

Como se puede observar, el problema con BPCS es que al procesar la imagen por bloques, un análisis minucioso, por ejemplo con un acercamiento profundo revelará alteraciones anormales a la imagen (en forma de cuadrados), despertando así la sospecha de que sea un portador esteganográfico. En términos generales, se puede decir que una buena portadora es aquella imagen con muchas zonas de alta frecuencia (que se pueden interpretar como ruido) ya que las variaciones producidas por el mensaje incrustado serán menos notorias que en un área de baja frecuencia –como por ejemplo un color sólido o con pocas variantes de tonalidad.

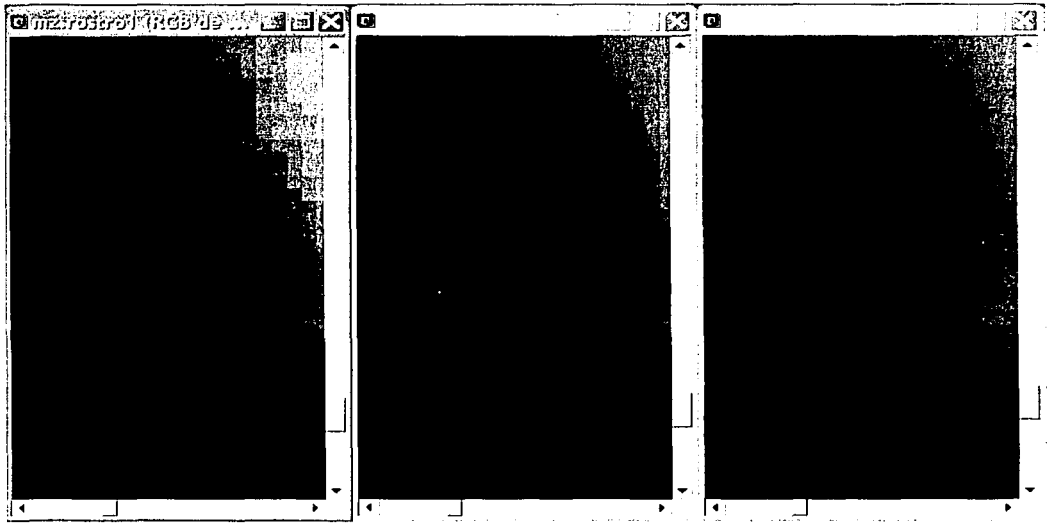


Figura 38. Izquierda: Portadora empleando el método BPCS, PSNR = 37.42. Centro: Imagen original. Derecha: Portadora empleando el programa DigiMarker, PSNR = 38.72. Todas las imágenes con acercamiento de 1400%.

La figura 38, que es un acercamiento a la figura 6 del capítulo 1, ilustra lo dicho anteriormente. En el centro aparece la imagen original, sin ninguna alteración. Observe que la ropa de la modelo es negra, presentándose una zona de baja frecuencia, poco apta para incrustar un mensaje esteganográfico. De todas formas se incrustó un mensaje utilizando dos técnicas, BPCS y DigiMarker. Se forzó a DigiMarker para que empleara 4 planos de bits a fin de evidenciar la presencia del mensaje.

Del lado izquierdo se observa el resultado de aplicar BPCS y se observa que:

a) Al descubrir una zona de baja frecuencia, BPCS discrimina los bloques no ruidosos. Sólo emplea los bloques considerados ruidosos.
b) Este método presenta dos desventajas:
    1. El área de aprovechamiento para incrustar el mensaje es menor
    2. La desventaja más importante: se evidencia la presencia de bloques que evidentemente son artificiales y no pertenecen a la imagen original, destruyendo así el propósito de la esteganografía. Dichos bloques se observan en la región negra.

Del lado derecho se observa el resultado de aplicar DigiMarker y se observa que:

a) Aprovecha por igual las zonas de alta y de baja frecuencia.

b) El ruido inducido es obvio, pero tiene un aspecto natural; es decir, ese mismo ruido se pudo presentar si la imagen se hubiera digitalizado de un periódico o de otra fuente impresa a baja resolución.
c) Una ventaja importante es que el grado de incrustación es alto.

¿Por qué se considera desventajosa la presencia evidente de los bloques (para BPCS) y no se considera desventajosa la presencia evidente de ruido (para DigiMarker)? El lector debe tener presente que en la situación real en que el análisis de las imágenes se puede dar (análisis esteganográfico), la imagen original, es decir, la portadora sin mensaje incrustado, no estará disponible para su comparación. De este modo, se busca que las alteraciones realizadas a la portadora tengan la misma apariencia que las alteraciones producidas por una fuente natural, como puede ser el ruido inducido por procesos de escaneo o de aplicación de ciertos filtros digitales. DigiMarker busca distribuir con apariencia "natural" el mensaje incrustado a fin de no levantar sospechas.

4.2 Consideración sobre la visión humana

Al ser la portadora seleccionada una imagen digital, es conveniente que analicemos con más detalle algunas características de la visión humana, en el entendido de que las imágenes digitales son producidas con el propósito de ser vistas por el ojo humano. Este entendimiento nos ayudará a mejorar el sistema esteganográfico.

El sistema visual humano está compuesto por los ojos y por una porción del cerebro que procesa las señales neuronales del ojo. Juntos el ojo y el cerebro convierten la información óptica en una percepción de una escena visual. Antes de que los impulsos visuales del ojo sean procesados por la corteza visual, ocurre un interesante procesamiento de información de intensidad en el mismo ojo.

Las principales características de interés son las formas en las que los fotorreceptores del ojo responden a diferencias en la intensidad de la luz y en las que responden a detalles finos. La relación entre la intensidad de luz que entra al ojo y su percepción de brillo no es una función lineal. Esto quiere decir que mientras cambia el brillo del objeto visto, el observador no contempla un cambio de brillo en la misma proporción. La respuesta del ojo es más logarítmica, semejante a la curva presentada en la figura 39. De hecho, está demostrado que la intensidad de brillo de un objeto debe casi duplicarse antes de que el ojo pueda percibir el cambio. De este modo, cambios pequeños de intensidad en regiones oscuras de una escena tienden a ser más perceptibles que cambios pequeños idénticos en regiones brillosas. La relación entre la intensidad de luz y el brillo percibido se conoce como la *ley de Weber*. El punto de interés en esteganografía es que la respuesta logarítmica del ojo lo hace más sensible a cambios de intensidad en las regiones oscuras que en las regiones claras de una imagen. En el proceso de incrustación de un mensaje esteganográfico, pequeños cambios en regiones iluminadas producirán, sin lugar a dudas, minúsculos cambios de intensidad que serán imperceptibles por el ojo humano.

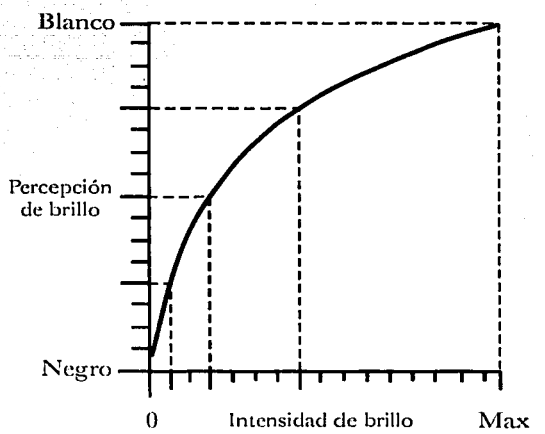


Figura 39. Respuesta logarítmica del ojo a la intensidad de luz

Las interacciones entre los fotorreceptores del ojo, conocidas como *inhibición lateral*, también producen un fenómeno importante que considerar. Hay dos efectos que ilustran el rol de la inhibición lateral en la percepción del brillo. Un efecto, llamado *contraste simultáneo*, es una ilusión óptica en la que el brillo percibido de una región depende de la intensidad del área alrededor. Este efecto se demuestra en la figura 40.

Figura 40. Ejemplo del efecto contraste simultáneo.

Los dos cuadrados en el interior tienen la misma intensidad, no obstante el cuadrado de la izquierda parece más brillante que el de la derecha. Esto es debido a que el área alrededor del cuadrado izquierdo es más oscura que el área que rodea al cuadrado derecho. Nuestro sistema visual ajusta su respuesta a la intensidad en base al promedio de intensidad en el entorno del objeto enfocado. Por esta razón, se presenta la aparente diferencia de brillo en los dos cuadrados.

Otro efecto causado por la inhibición lateral se conoce como el efecto *Mach de banda*[3]. Con este efecto, el sistema visual acentúa los cambios agudos de intensidad. La figura 41 ilustra este efecto.

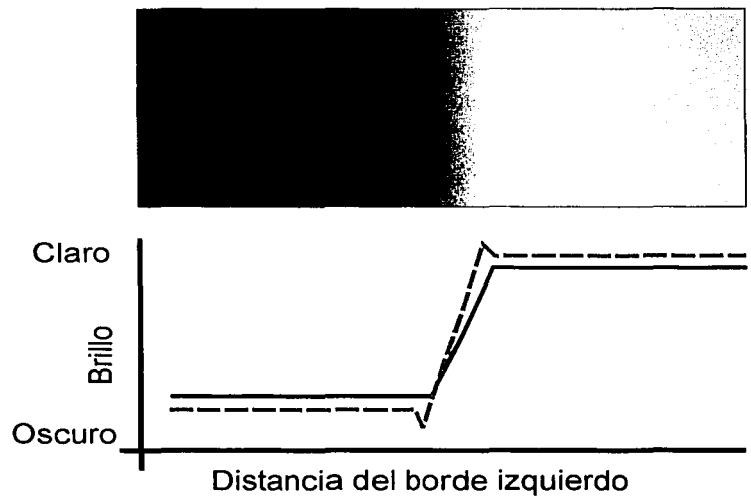


Figura 41. Efecto Mach de banda.

La curva sólida en negro representa la cantidad de luz que se refleja por la parte superior de la figura. La curva discontinua en rojo representa la brillantez de esta figura como se percibe normalmente. A la izquierda del punto donde la figura justo comienza a aclararse, las personas suelen percibir una barra oscura, que es ligeramente más oscura que el área a su izquierda. En el punto donde el brillo deja de aumentar, usualmente se percibe una barra brillante.

Por otro lado, el sistema visual tiene limitaciones fundamentales en la respuesta a la frecuencia. Como en cualquier otro sistema óptico, el ojo tiene un límite para resolver detalles o transiciones de intensidad. Los factores que lo limitan son el número y organización de fotorreceptores en la retina, la calidad de la óptica del ojo (córnea, humor acuoso, etc.) y la transmisión y procesamiento de la información visual al cerebro. Generalmente la respuesta a frecuencia cede cuando las transiciones de intensidad se vuelven más pequeñas. También es un factor el contraste, o sea, la diferencia entre niveles de gris, en la transición de intensidad. A más contraste, más detalle que el ojo puede distinguir. Si la transición es demasiado fina y/o el contraste demasiado bajo, el ojo no podrá resolver la figura; en ese punto, la vista humana sólo puede percibir un promedio de grises de esa área.

Estos fenómenos presentados ilustran el complejo proceso que ocurre dentro del sistema visual humano. Al combinar los conceptos de respuesta no lineal a la intensidad, interacción

de fotorreceptores y respuesta a la frecuencia, se pueden hacer algunas observaciones importantes:

1. La intensidad del objeto apreciado está relacionada con la intensidad promedio del entorno del objeto. El objeto parece más oscuro si su entorno es relativamente brillante. El objeto parece más claro si su entorno es relativamente oscuro.
2. Cambios modestos de intensidad son más aparentes en regiones oscuras que en regiones claras.
3. Transiciones agudas de intensidad son acentuadas aún más.
4. La respuesta al detalle de la imagen decrece conforme el detalle se hace más pequeño. Detalles en alto contraste son más fáciles de resolver que aquellos en bajo contraste.

Con estas observaciones en mente podemos afinar la acción de procesos esteganográficos a fin de optimizar los resultados.

Tomando estos criterios en consideración, se deja al lector la observación de las siguientes dos imágenes para que decida por qué motivos una de ellas es mejor portadora esteganográfica que la otra.

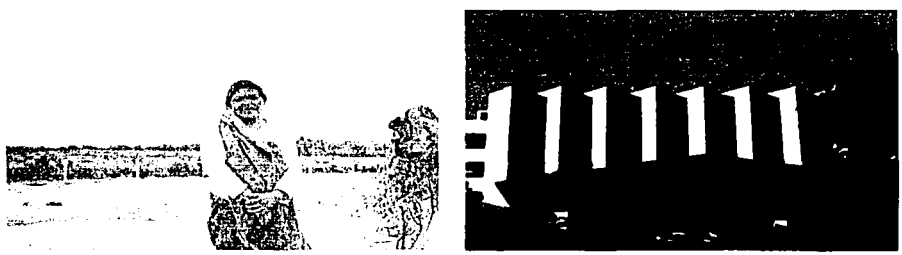


Figura 42. Dos portadoras esteganográficas de distinta calidad.

4.3 Implementación en Visual C de funciones de interés

```
////////////////////////////////////////////////////////////////////////////////
// Función para cargar archivos gráficos
LPDMBITMAP CDMCore::LoadBitmapFile(LPCSTR lpszFileName, UINT nIDFileType)
{
        // Invoca diferentes funciones de acuerdo al tipo de archivo
        // En nuestro caso sólo aceptamos BMP
        switch(nIDFileType)
        {
        case FT_WINDOWS_BITMAP:
                return LoadWindowsBitmap(lpszFileName);
        }

        // No hay función disponible para manejar este tipo de archivo; regresa NULL
        m -> uErrorCode = DMERR_NOT_SUPPORTED;
```

```
        return NULL;
}

////////////////////////////////////////////////////////////////////////////////
// Función para cargar bitmaps de windows
LPDMBITMAP CDMCore::LoadWindowsBitmap(LPCSTR lpszFileName)
{
        CFile                   file;
        BITMAPFILEHEADER        bfHeader;
        BITMAPINFOHEADER        biHeader;
        LPDMBITMAP              lpDMBitmap;
        LPBYTE                  lpBuffer, lpLineBuffer;
        LPDMRGB                 lpRGB;
        LPDMBGR                 lpBGR;
        DWORD                   dwBytesPerLine;
        DWORD                   dwImageSize;
        LONG                    i, j;

        // Abre el archivo bitmap
        CFileException ex;
        if ( !file.Open(lpszFileName, CFile::modeRead|CFile::typeBinary, &ex) )
        {
                m -> uErrorCode = DMERR_FILE_EXCEPTION;
                ex.GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
                return NULL;
        }
        try
        {
                // Lee el encabezado del archivo
                file.Read(&bfHeader, sizeof(BITMAPFILEHEADER));

                // Verifica el formato
                if ( ( bfHeader.bfType & 0xFF ) != 'B' || ( ( bfHeader.bfType >> 8 ) &
0xFF ) != 'M' )
                {
                        m -> uErrorCode = DMERR_FILE_TYPE_ERROR;
                        return NULL;
                }

                // Lee la información de encabezado
                file.Read(&biHeader, sizeof(BITMAPINFOHEADER));

                // Verifica el formato del encabezado
                if ( biHeader.biSize != sizeof(BITMAPINFOHEADER) )
                {
                        m -> uErrorCode = DMERR_FILE_TYPE_ERROR;
                        return NULL;
                }

                // Verifica compresión
                // BI_RGB quiere decir sin compresión
                if ( biHeader.biCompression != BI_RGB )
                {
                        m -> uErrorCode = DMERR_NOT_SUPPORTED;
                        return NULL;
                }
        }
        catch(CFileException *pEx)
        {
                m -> uErrorCode = DMERR_FILE_EXCEPTION;
                pEx -> GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
                pEx -> Delete();
                return NULL;
        }

        // Calcula el número de bytes por línea y el tamaño de la imagen
        switch(biHeader.biBitCount)
        {
        case 24:        // Sólo se aceptan imágenes de 24 bits
                dwBytesPerLine = biHeader.biWidth * 3;
                if ( dwBytesPerLine % 4 )
```

```
                dwBytesPerLine = ( ( dwBytesPerLine >> 2 ) + 1 ) << 2;
            dwImageSize = dwBytesPerLine * biHeader.biHeight;
            break;
        default:
            m -> uErrorCode = DMERR_NOT_SUPPORTED;
            return NULL;
        }

        // Asignación de memoria
        // Aquí se crea un buffer para indicar el avance de carga
        if ( ( lpDMBitmap = new DMBITMAP ) == NULL ||
             ( lpDMBitmap -> lpBitmap = new DMRGB[biHeader.biWidth * biHeader.biHeight] )
== NULL ||
             ( lpBuffer = new BYTE[dwImageSize] ) == NULL ||
             ( lpLineBuffer = new BYTE[dwBytesPerLine] ) == NULL )
        {
            m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
            return NULL;
        }
        lpDMBitmap -> width = biHeader.biWidth;
        lpDMBitmap -> height = biHeader.biHeight;

        // Inicializa el indicador de avance
        m -> fProgressFactor = 0.5f;
        m -> uProgressStart = 0;
        SetProgress(0);


        // Carga los datos y cierra el archivo
        for ( i = 0 ; i < biHeader.biHeight ; i++ )
        {
            try
            {
                file.ReadHuge(lpLineBuffer, dwBytesPerLine);
            }
            catch(CFileException *pEx)
            {
                m -> uErrorCode = DMERR_FILE_EXCEPTION;
                pEx -> GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
                pEx -> Delete();
                delete lpDMBitmap;
                delete lpBuffer;
                delete lpLineBuffer;
                return NULL;
            }
            memcpy(&lpBuffer[i * dwBytesPerLine], lpLineBuffer, dwBytesPerLine);
            // Calcula el avance
            SetProgress((i + 1) * 100 / biHeader.biHeight);
        }
        file.Close();

        // Inicializa el indicador de avance
        m -> uProgressStart = 50;

        // Convierte los datos al formato del buffer interno
        switch(biHeader.biBitCount)
        {
        case 24:
            lpRGB = lpDMBitmap -> lpBitmap;
            i = biHeader.biHeight;
            while(i--)
            {
                lpBGR = (LPDMBGR)(&lpBuffer[dwBytesPerLine * i]);
                j = biHeader.biWidth;
                while(j--)
                {
                    lpRGB -> r = lpBGR -> r;
                    lpRGB -> g = lpBGR -> g;
                    lpRGB -> b = lpBGR -> b;
                    lpRGB++;
                    lpBGR++;
```

```
                }

                // Calcula el avance
                SetProgress(100 - i * 100 / biHeader.biHeight);
            }
            break;
        default:
            m -> uErrorCode = DMERR_NOT_SUPPORTED;
            delete lpBuffer;
            delete lpLineBuffer;
            delete lpDMBitmap;
            return NULL;
        }

        // Libera la memoria temporal
        delete lpBuffer;
        delete lpLineBuffer;

        return lpDMBitmap;
}

////////////////////////////////////////////////////////////////////////////////////////////
// Función para generar un mapa de bits Windows a partir de nuestro buffer interno
LPWINBITMAP CDMCore::GenerateWindowsBitmap(LPDMBITMAP lpDMBitmap)
{
        LPWINBITMAP             lpWinBitmap;
        DWORD                   dwBytesPerLine;
        DWORD                   dwImageSize;
        LPDMRGB                 lpRGB;
        LPDMBGR                 lpBGR;
        LONG                    i, j;
        LPBYTE                  lpBuffer;

        ASSERT(lpDMBitmap);

        // Asignación de memoria
        if ( ( lpWinBitmap = new WINBITMAP ) == NULL )
        {
            m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
            return NULL;
        }

        // Calcula el número de bytes por línea y el tamaño de la imagen
        dwBytesPerLine = lpDMBitmap -> width * 3;
        if ( dwBytesPerLine % 4 )
            dwBytesPerLine = ( ( dwBytesPerLine >> 2 ) + 1 ) << 2;
        dwImageSize = dwBytesPerLine * lpDMBitmap -> height;

        // Asignación de memoria para el mapa de bits
        if ( ( lpWinBitmap -> lpBuffer = new BYTE[dwImageSize] ) == NULL )
        {
            m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
            return NULL;
        }

        // Relleno de información necesaria de encabezado
        lpWinBitmap -> biHeader.biBitCount = 24;
        lpWinBitmap -> biHeader.biClrImportant = 0;
        lpWinBitmap -> biHeader.biClrUsed = 0;
        lpWinBitmap -> biHeader.biCompression = BI_RGB;
        lpWinBitmap -> biHeader.biHeight = lpDMBitmap -> height;
        lpWinBitmap -> biHeader.biPlanes = 1;
        lpWinBitmap -> biHeader.biSize = sizeof(BITMAPINFOHEADER);
        lpWinBitmap -> biHeader.biSizeImage = dwImageSize;
        lpWinBitmap -> biHeader.biWidth = lpDMBitmap -> width;
        lpWinBitmap -> biHeader.biXPelsPerMeter = 3780;
        lpWinBitmap -> biHeader.biYPelsPerMeter = 3780;

        // Convierte el buffer interno a un mapa de bits de windows
        lpRGB = lpDMBitmap -> lpBitmap;
        lpBuffer = (LPBYTE)lpWinBitmap -> lpBuffer;
```

```
        i = lpDMBitmap -> height;
        while(i--)
        {
                lpBGR = (LPDMBGR)(&lpBuffer[dwBytesPerLine * i]);
                j = lpDMBitmap -> width;
                while(j--)
                {
                        lpBGR -> r = lpRGB -> r;
                        lpBGR -> g = lpRGB -> g;
                        lpBGR -> b = lpRGB -> b;
                        lpBGR++;
                        lpRGB++;
                }

                // Pone ceros en el resto de bytes
                memset(lpBGR, 0, dwBytesPerLine - lpDMBitmap -> width * 3);
        }

        return lpWinBitmap;
}

////////////////////////////////////////////////////////////////////////////////////////
// Función para incrustar el mensaje en el buffer interno
LPWATERMARK CDMCore::EmbedWatermark(LPDMBITMAP lpDMBitmap, LPCSTR lpData, UINT uType, UINT
uCompressionLevel)
{
        // lpData y lpDMBitmap no deben ser NULL
        ASSERT(lpDMBitmap);
        ASSERT(lpData);

        // Incrusta de acuerdo a distintos tipos de mensajes
        switch(uType)
        {
        case WT_TEXT:  // texto puro
                // En este caso, lpData debe apuntar directamente al string
                return EmbedPureTextWatermark(lpDMBitmap, lpData, uCompressionLevel);
        case WT_FILE:  // importa un archivo
                // En este caso lpData debe apuntar al nombre del archivo
                return EmbedFileWatermark(lpDMBitmap, lpData, uCompressionLevel);
        default:                // no habilitado
                m -> uErrorCode = DMERR_NOT_SUPPORTED;
                return NULL;
        }

        //
        m -> uErrorCode = DMERR_UNKNOWN;
        return NULL;
}

////////////////////////////////////////////////////////////////////////////////////////
// Función para duplicar el buffer interno
LPDMBITMAP CDMCore::CloneBitmap(LPDMBITMAP lpDMBitmap)
{
        LPDMBITMAP      pNewBitmap;

        if ( lpDMBitmap == NULL ) return NULL;

        // Asignación de memoria para un nuevo mapa de bits
        if ( ( pNewBitmap = new DMBITMAP ) == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                return NULL;
        }

      if ( ( pNewBitmap -> lpBitmap = new DMRGB[lpDMBitmap -> width * lpDMBitmap -> height] )
== NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                return NULL;
        }
```

```
        // Duplica el mapa de bits
        pNewBitmap -> width = lpDMBitmap -> width;;
        pNewBitmap -> height = lpDMBitmap -> height;
        memcpy(pNewBitmap -> lpBitmap, lpDMBitmap -> lpBitmap, sizeof(DMRGB) * lpDMBitmap ->
width * lpDMBitmap -> height);

        return pNewBitmap;
}

////////////////////////////////////////////////////////////////////////////////
// Función para extraer la información incrustada en el buffer
LPWATERMARK CDMCore::ExtractWatermark(LPDMBITMAP lpDMBitmap)
{
        LPWATERMARK             lpWM;
        LPWATERMARKINFO         lpInfo;
        LPCSTR                  lpBuffer;
        DWORD                   dwBoundedWMSize;
        CBlackBox               bbox;
        CZipLib                 zlib;
        UINT                    uShiftAmount;

        // Debe existir el mapa de bits fuente
        ASSERT(lpDMBitmap);
        lpBuffer = (LPCSTR)lpDMBitmap -> lpBitmap;

        // Inicializa el indicador de avance
        m -> uProgressStart = 0;
        m -> fProgressFactor = 0.3333f;
        SetProgress(0);

        // Asigna memoria
        lpInfo = new WATERMARKINFO;
        if ( lpInfo == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                return NULL;
        }

        lpWM = new WATERMARK;
        if ( lpWM == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                delete lpInfo;
                return NULL;
        }

        // Llena la estructura del mensaje
        lpWM -> lPixelBytes = lpDMBitmap -> width * lpDMBitmap -> height * sizeof(DMRGB);
        lpWM -> lpDMBitmap = lpDMBitmap;
        lpWM -> lpInfo = lpInfo;

        // Primero intenta extraer la información del mensaje
        for ( uShiftAmount = 1 ; uShiftAmount <= 3 ; uShiftAmount++ )
        {
                if ( lpWM -> lPixelBytes > (LONG)(sizeof(WATERMARKINFO) << uShiftAmount) )
                {
                        m -> bProgressEnabled = FALSE;
                        PerformExtract((LPBYTE)lpInfo,      lpBuffer,      sizeof(WATERMARKINFO),
sizeof(WATERMARKINFO) << uShiftAmount, uShiftAmount);
                        m -> bProgressEnabled = TRUE;
                        // Verifica la firma
                                // código explícitamente omitido por seguridad.
                                // Se encontró un mensaje!
                                break;
                }
                else
                {
                        // No hay suficiente espacio para el encabezado del mensaje; abortar
                        delete lpInfo;
                        delete lpWM;
```

```
                return NULL;
            }
        }

        // uShiftAmount == 4 si no se encuentra un mensaje
        if ( uShiftAmount > 3 )
        {
            delete lpInfo;
            delete lpWM;
            return NULL;
        }

        // Calcula el espacio disponible
        lpWM -> lAvailableSpace = ( lpWM -> lPixelBytes >> uShiftAmount ) -
sizeof(WATERMARKINFO);

        // Verifica la versión
        if ( lpInfo -> btVersion != WATERMARK_VERSION )
        {
            m -> uErrorCode = DMERR_VERSION_ERROR;
            delete lpInfo;
            delete lpWM;
            return NULL;
        }

        // Busca al inicio del mensaje
        lpBuffer = &lpBuffer[(sizeof(WATERMARKINFO) << uShiftAmount)];

        // Obtiene el tamaño del buffer redondeado para desencriptar el mensaje
        dwBoundedWMSize = bbox.GetBoundedBufferSize(lpInfo -> lWatermarkLength);

        // Extrae el mensaje de acuerdo a su tipo
        if ( lpInfo -> btExtension[0] == 0 && lpInfo -> btExtension[1] == 0 || lpInfo ->
btExtension[2] == 0 )
        {
            // Mensaje de texto puro

            LPBYTE lpDataBuffer;

            // Asignación de memoria para el texto
            if ( ( lpDataBuffer = new BYTE[dwBoundedWMSize] ) == NULL )
            {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                delete lpInfo;
                delete lpWM;
                return NULL;
            }

            // Extracción del mensaje
            m -> fProgressFactor = (float)dwBoundedWMSize / lpWM -> lAvailableSpace / 3.0f;
            PerformExtract(lpDataBuffer, lpBuffer, dwBoundedWMSize, ((DWORD)lpWM ->
lAvailableSpace) << uShiftAmount, uShiftAmount);

            // Desencriptamiento
            bbox.SetKey(lpInfo -> btKey);
            // Comienza el avance en 1/3
            m -> uProgressStart = dwBoundedWMSize * 100 / lpWM -> lAvailableSpace / 3;
            bbox.Decrypt(lpDataBuffer, dwBoundedWMSize, this);

            // Descompresión
            // Comienza el avance en 2/3
            m -> uProgressStart = dwBoundedWMSize * 100 / lpWM -> lAvailableSpace * 2 / 3;
            if ( !zlib.Uncompress(lpDataBuffer, lpInfo -> lWatermarkLength, this) )
            {
                m -> uErrorCode = DMERR_UNKNOWN;
                delete []lpDataBuffer;
                delete lpWM;
                delete lpInfo;
                return NULL;
            }
            delete []lpDataBuffer;
```

```
        // El mensaje ahora se encuentra en el objeto zlib
        if ( ( lpWM -> lpData = new BYTE[zlib.GetDataLength()] ) == NULL )
        {
                delete lpWM;
                delete lpInfo;
                return NULL;
        }
        strcpy((char*)lpWM -> lpData, (char*)zlib.GetData());
}
else
{
        // Importa el archivo de mensaje
        // En este caso se debe suplir una función porque deseamos
        // preguntar al usuario el nombre del archivo a almacenar
        ASSERT(m -> pMainWnd);

        CFile                   file;
        EXTRACTPROMPT  prompt;
        LPBYTE                  lpFileBuffer;
        BYTE                    ext[5];

        // Creación del buffer para extracción
        lpFileBuffer = new BYTE[dwBoundedWMSize];
        if ( lpFileBuffer == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                delete lpInfo;
                delete lpWM;
                return NULL;
        }

        // Genera una extensión y pide al usuario el nombre del archivo donde
        // salvar el archivo importado
        ext[0] = '.';
        ext[1] = lpInfo -> btExtension[0];
        ext[2] = lpInfo -> btExtension[1];
        ext[3] = lpInfo -> btExtension[2];
        ext[4] = '\0';
        prompt.lpExtension = (LPCSTR)ext;
        m -> pMainWnd -> SendMessage(DMMSG_PROMPT_FILENAME, 0, (LPARAM)&prompt);
        if ( prompt.lpszFileName == NULL )
        {
                // El usuario oprimió "Cancelar"
                m -> uErrorCode = DMERR_USER_CANCELED;
                delete lpInfo;
                delete lpWM;
                delete []lpFileBuffer;
                return NULL;
        }

        // Recordar el nombre del archivo
        lpWM -> lpData = new BYTE[strlen(prompt.lpszFileName) + 1];
        if ( lpWM -> lpData == NULL )
        {
                // No hay suficiente memoria para almacenar el nombre del archivo
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                delete lpInfo;
                delete lpWM;
                delete []lpFileBuffer;
                return NULL;
        }
        strcpy((char*)lpWM -> lpData, prompt.lpszFileName);

        // NOTA: El procedimiento completo se realiza en tres pasos:
        //
        //   1. Extracción
        //   2. Desencriptamiento
        //   3. Descompresión
        //
        // El indicador de avance funcionará en los tres pasos principales.
```

```
                // Otros procedimientos más pequeños como la extracción del encabezado,
                // son tan rápidos que no requieren involucrar al indicador de progreso.
                // Por eso se establece el factor de avance en 1/3.

                // Extraer al buffer redondeado
                m -> fProgressFactor = (float)dwBoundedWMSize / lpWM -> lAvailableSpace / 3.0f;
                PerformExtract(lpFileBuffer,   lpBuffer,   dwBoundedWMSize,   ((DWORD)lpWM   ->
lAvailableSpace) << uShiftAmount, uShiftAmount);

                // Desencriptar
                bbox.SetKey(lpInfo -> btKey);
                // Comienza en 1/3
                m -> uProgressStart = dwBoundedWMSize * 100 / lpWM -> lAvailableSpace / 3;
                bbox.Decrypt(lpFileBuffer, dwBoundedWMSize, this);

                // Descomprime
                // Comienza en 2/3
                m -> uProgressStart = dwBoundedWMSize * 100 / lpWM -> lAvailableSpace * 2 / 3;
                if ( !zlib.Uncompress(lpFileBuffer, lpInfo -> lWatermarkLength, this) )
                {
                        m -> uErrorCode = DMERR_UNKNOWN;
                        delete lpInfo;
                        delete []lpWM -> lpData;
                        delete lpWM;
                        delete []lpFileBuffer;
                        return NULL;
                }
                delete []lpFileBuffer;

                // Actualiza datos después de descomprimir
                lpFileBuffer = zlib.GetData();
                lpInfo -> lWatermarkLength = zlib.GetDataLength();

                // Crea el archivo de escritura
                CFileException ex;
                if                      (                       !file.Open(prompt.lpszFileName,
CFile::modeWrite|CFile::modeCreate|CFile::typeBinary, &ex) )
                {
                        m -> uErrorCode = DMERR_FILE_EXCEPTION;
                        ex.GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
                        delete lpInfo;
                        delete []lpWM -> lpData;
                        delete lpWM;
                        return NULL;
                }

                try
                {
                        // Escribe al archivo (longitud del archivo, no tamaño del buffer)
                        file.WriteHuge(lpFileBuffer, lpInfo -> lWatermarkLength);

                        // Cierra el archivo y libera memoria
                        file.Close();
                }
                catch(CFileException *pEx)
                {
                        m -> uErrorCode = DMERR_FILE_EXCEPTION;
                        pEx -> GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
                        pEx -> Delete();
                        delete lpInfo;
                        delete []lpWM -> lpData;
                        delete lpWM;
                        return NULL;
                }
        }

        return lpWM;
}

////////////////////////////////////////////////////////////////////////////////
// Función para incrustar mensaje de texto puro
```

```
LPWATERMARK   CDMCore::EmbedPureTextWatermark(LPDMBITMAP   lpDMBitmap,   LPCSTR   lpText,   UINT
uCompressionLevel)
{
        LPWATERMARKINFO                 lpInfo;
        LPWATERMARK                     lpWM;
        LPBYTE                          lpBuffer;
        DWORD                           dwBufferLength;
        int                                   i;
        CBlackBox                       bbox;
        CZipLib                         zlib;
        UINT                            uShiftAmount;
        LONG                            lPixelBytes, lAvailableSpace;
        LPBYTE                          lpDataBuffer;           // buffer redondeado
        DWORD                           dwBoundedWMSize;        // tamaño de buffer redondeado
        DWORD                           dwDataLength;

        // Calcula datos necesarios
        uShiftAmount = GetShiftAmount(uCompressionLevel);
        lPixelBytes = lpDMBitmap -> width * lpDMBitmap -> height * sizeof(DMRGB);
        lAvailableSpace = ( lPixelBytes >> uShiftAmount ) - sizeof(WATERMARKINFO);

        // Inicializa el indicador de avance
        m -> uProgressStart = 0;
        m -> fProgressFactor = 0.3333f;
        m -> dwProgressTotalLength = lAvailableSpace;
        SetProgress(0);

        // PASO 1: Comprimir
        if ( !zlib.Compress((LPBYTE)lpText, strlen(lpText) + 1, this) )
        {
                m -> uErrorCode = DMERR_UNKNOWN;
                return NULL;
        }
        dwDataLength = zlib.GetDataLength();

        // Obtiene tamaño de buffer redondeado
        dwBoundedWMSize = bbox.GetBoundedBufferSize(dwDataLength);

        // Verifica espacio disponible
        if ( lAvailableSpace < (LONG)dwBoundedWMSize )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_SPACE;
                return NULL;
        }

        // Asignación de memoria para la estructura del mensaje
        lpInfo = new WATERMARKINFO;
        if ( lpInfo == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                return NULL;
        }

        // Rellena la estructura del mensaje
        // Aquí se omiten algunas líneas de código por seguridad
        lpInfo -> btVersion = WATERMARK_VERSION;
        lpInfo -> btExtension[0] = 0;
        lpInfo -> btExtension[1] = 0;
        lpInfo -> btExtension[2] = 0;
        lpInfo -> lWatermarkLength = dwDataLength;

        // Asignación de memoria para la estructura del mensaje
        lpWM = new WATERMARK;
        if ( lpWM == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                delete lpInfo;
                return NULL;
        }

        // Copia los datos de la estructura del mensaje
```

```
        lpWM -> lPixelBytes = lPixelBytes;
        lpWM -> lAvailableSpace = lAvailableSpace;
        if ( ( lpWM -> lpData = new BYTE[strlen(lpText) + 1] ) == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                delete lpInfo;
                delete lpWM;
                return NULL;
        }
        strcpy((char*)lpWM -> lpData, lpText);
        lpWM -> lpDMBitmap = lpDMBitmap;
        lpWM -> lpInfo = lpInfo;

        // Crea un buffer redondeado
        if ( ( lpDataBuffer = (LPBYTE)GetBoundedBuffer(zlib.GetData(), dwDataLength) ) ==
NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                delete []lpWM -> lpData;
                delete lpInfo;
                delete lpWM;
                return NULL;
        }

        // Genera la llave de encriptamiento

        // Código explícitamente omitido por seguridad


        // Incrusta primero el encabezado del mensaje
        lpBuffer = (LPBYTE)(lpDMBitmap -> lpBitmap);
        m -> bProgressEnabled = FALSE;
        PerformEmbed(lpBuffer,    (LPCSTR)lpInfo,    sizeof(WATERMARKINFO)    <<    uShiftAmount,
sizeof(WATERMARKINFO), uShiftAmount);
        m -> bProgressEnabled = TRUE;

        // Encripta el mensaje
        bbox.SetKey(lpInfo -> btKey);
        // Comienza en 1/3
        m -> uProgressStart = dwBoundedWMSize * 100 / lAvailableSpace / 3;
        m -> fProgressFactor = (float)dwBoundedWMSize / lAvailableSpace / 3.0f;
        bbox.Encrypt(lpDataBuffer, dwBoundedWMSize, this);

        // Incrusta el mensaje
        // Comienza en 2/3
        m -> uProgressStart = dwBoundedWMSize * 100 / lAvailableSpace * 2 / 3;
        lpBuffer = &lpBuffer[(sizeof(WATERMARKINFO) << uShiftAmount)];
        PerformEmbed(lpBuffer, (LPCSTR)lpDataBuffer, ((DWORD)lAvailableSpace) << uShiftAmount,
dwBoundedWMSize, uShiftAmount);

        // Libera el buffer temporal
        delete []lpDataBuffer;

        return lpWM;
}

////////////////////////////////////////////////////////////////////////////////
// Función para incrustar un mensaje proveniente de archivo importado
LPWATERMARK CDMCore::EmbedFileWatermark(LPDMBITMAP lpDMBitmap, LPCSTR lpszFileName, UINT
uCompressionLevel)
{
        CFile                   file;
        LPBYTE                  lpBuffer, lpFileBuffer;
        LPWATERMARK             lpWM;
        LPWATERMARKINFO         lpInfo;
        CFileStatus             status;
        DWORD                   dwFileLength, dwBufferLength, dwBoundedWMSize;
        int                     i;
        CBlackBox               bbox;
        CZipLib                 zlib;
        LONG                    lPixelBytes, lAvailableSpace;
```

```
UINT                                uShiftAmount;

// NOTA: El proceso completo consiste en tres pasos principales:
//    1. Comprimir
//    2. Encriptar
//    3. Incrustar
// El indicador de progreso funcionará en los tres pasos principales.
// Otros procedimientos como incrustar el encabezado son muy rápidos
// y no requieren el indicador de avance. Por eso se establece el factor
// de avance en 1/3.

// Inicializa el indicador de avance
m -> uProgressStart = 0;
m -> fProgressFactor = 0.3333f;
SetProgress(0);

// Calcula primero los datos necesarios
uShiftAmount = GetShiftAmount(uCompressionLevel);
lPixelBytes = lpDMBitmap -> width * lpDMBitmap -> height * sizeof(DMRGB);
lAvailableSpace = ( lPixelBytes >> uShiftAmount ) - sizeof(WATERMARKINFO);

// Lee el archivo importado
CFileException ex;
if ( !file.Open(lpszFileName, CFile::modeRead|CFile::typeBinary, &ex) )
{
        m -> uErrorCode = DMERR_FILE_EXCEPTION;
        ex.GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
        return NULL;
}
try
{
        dwFileLength = file.GetLength();
        lpFileBuffer = NULL;
        if ( ( lpFileBuffer = new BYTE[dwFileLength] ) == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                return NULL;
        }
        file.ReadHuge(lpFileBuffer, dwFileLength);
        file.Close();
}
catch(CFileException *pEx)
{
        m -> uErrorCode = DMERR_FILE_EXCEPTION;
        pEx -> GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
        pEx -> Delete();
        if ( lpFileBuffer )
                delete []lpFileBuffer;
        return NULL;
}

// Establece el tamaño total para el indicador de avance
m -> dwProgressTotalLength = lAvailableSpace;

// PASO 1: Comprimir
if ( !zlib.Compress((LPBYTE)lpFileBuffer, dwFileLength, this) )
{
        m -> uErrorCode = DMERR_UNKNOWN;
        delete []lpFileBuffer;
        return NULL;
}
dwFileLength = zlib.GetDataLength();
delete []lpFileBuffer;

// Ahora los datos comprimidos del archivo se encuentran en el
// objeto zlib. Para encriptarlo se debe crear un buffer
// redondeado y luego copiar los datos al buffer.
// Observe que dwFileLength NO es la longitud real del archivo
// original importado sino de la versión comprimida.

// Obtiene el tamaño del buffer redondeado
```

```
        dwBoundedWMSize = bbox.GetBoundedBufferSize(dwFileLength);

        // Verifica disponibilidad de espacio
        if ( lAvailableSpace < (LONG)dwBoundedWMSize )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_SPACE;
                return NULL;
        }

        // Asignación de memoria para el encabezado del mensaje
        lpInfo = new WATERMARKINFO;
        if ( lpInfo == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                return NULL;
        }

        // Rellena el encabezado del mensaje
        // Aquí hay código omitido por seguridad

        lpInfo -> btVersion = WATERMARK_VERSION;
        lpInfo -> btExtension[0] = lpszFileName[strlen(lpszFileName) - 3];
        lpInfo -> btExtension[1] = lpszFileName[strlen(lpszFileName) - 2];
        lpInfo -> btExtension[2] = lpszFileName[strlen(lpszFileName) - 1];
        lpInfo -> lWatermarkLength = dwFileLength;

        // Asignación de memoria para el mensaje
        lpWM = new WATERMARK;
        if ( lpWM == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                delete lpInfo;
                return NULL;
        }

        // Rellena con el mensaje
        lpWM -> lPixelBytes = lPixelBytes;
        lpWM -> lAvailableSpace = lAvailableSpace;
        if ( ( lpWM -> lpData = new BYTE[strlen(lpszFileName) + 1] ) == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                delete lpWM;
                delete lpInfo;
                return NULL;
        }
        strcpy((char*)lpWM -> lpData, lpszFileName);
        lpWM -> lpDMBitmap = lpDMBitmap;
        lpWM -> lpInfo = lpInfo;

        // Obtiene un buffer redondeado
        if ( ( lpFileBuffer = GetBoundedBuffer(zlib.GetData(), dwFileLength) ) == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                delete []lpWM -> lpData;
                delete lpWM;
                delete lpInfo;
                return NULL;
        }

        // Genera una llave para encriptar

        // Código explícitamente omitido por seguridad

        // Primero incrusta el encabezado del mensaje
        lpBuffer = (LPBYTE)lpDMBitmap -> lpBitmap;
        // No es necesario actualizar el indicador de avance; se deshabilita temporalmente
        m -> bProgressEnabled = FALSE;
        PerformEmbed(lpBuffer,    (LPCSTR)lpInfo,    sizeof(WATERMARKINFO)    <<    uShiftAmount,
sizeof(WATERMARKINFO), uShiftAmount);
        m -> bProgressEnabled = TRUE;
```

```
        // PASO 2: Encripta
        bbox.SetKey(lpInfo -> btKey);
        // Comienza desde 1/3
        m -> uProgressStart = dwBoundedWMSize * 100 / lAvailableSpace / 3;
        m -> fProgressFactor = (float)dwBoundedWMSize / lAvailableSpace / 3.0f;
        bbox.Encrypt(lpFileBuffer, dwBoundedWMSize, this);

        // PASO 3: Incrusta la información encriptada
        // Comienza en 2/3
        m -> uProgressStart = dwBoundedWMSize * 100 / lAvailableSpace * 2 / 3;
        lpBuffer = &lpBuffer[(sizeof(WATERMARKINFO) << uShiftAmount)];
        PerformEmbed(lpBuffer, (LPCSTR)lpFileBuffer, ((DWORD)lAvailableSpace) << uShiftAmount,
dwBoundedWMSize, uShiftAmount);

        // Libera la memoria temporal
        delete []lpFileBuffer;

        return lpWM;
}

////////////////////////////////////////////////////////////////////////////////////////////
// Función para incrustar distribuidamente en un buffer
void CDMCore::PerformEmbed(LPBYTE lpBuffer, LPCSTR lpData, DWORD dwBufferLength, DWORD
dwDataLength, UINT uShiftAmount)
{
        PEMBED_PROCEDURE        pfnEmbed;
        DWORD                           dwBufferHold = dwBufferLength >> uShiftAmount;
        DWORD                           dwCounter;
        LPBYTE                          lpEntry;
        UINT                            uPercent, uLastPercent = 0;

        // Selecciona el algoritmo de acuerdo al nivel de incrustación
        switch(uShiftAmount)
        {
        case 3:
                pfnEmbed = CDMCore::PerformEmbed1BPB;
                break;
        case 2:
                pfnEmbed = CDMCore::PerformEmbed2BPB;
                break;
        case 1:
                pfnEmbed = CDMCore::PerformEmbed4BPB;
                break;
        default:
                // Debe haber un error
                ASSERT(FALSE);
        }

        // Incrustación
        for ( dwCounter = 0 ; dwCounter < dwDataLength ; dwCounter++ )
        {
                // Busca la localidad amplificada
                lpEntry = &lpBuffer[GetZoomedPosition(dwDataLength, dwBufferHold, dwCounter)
<< uShiftAmount];
                // Incrusta un byte
                (this->*pfnEmbed)(lpEntry, (BYTE)(*lpData));
                // Se prepara para el siguiente byte
                lpData++;

                // Calcula el avance
                uPercent = ( dwCounter + 1 ) * 100 / dwDataLength;
                if ( uPercent != uLastPercent )
                        SetProgress(uPercent);
                uLastPercent = uPercent;
        }
}

////////////////////////////////////////////////////////////////////////////////////////////
// Función para incrustar un byte bajo el algoritmo de 1 bit por byte
inline void CDMCore::PerformEmbed1BPB(LPBYTE lpBuffer, BYTE btData)
{
```

```
        *lpBuffer &= ~1; *lpBuffer |= (btData & 1); lpBuffer++; btData >>= 1;      // bit 0
        *lpBuffer &= ~1; *lpBuffer |= (btData & 1); lpBuffer++; btData >>= 1;      // bit 1
        *lpBuffer &= ~1; *lpBuffer |= (btData & 1); lpBuffer++; btData >>= 1;      // bit 2
        *lpBuffer &= ~1; *lpBuffer |= (btData & 1); lpBuffer++; btData >>= 1;      // bit 3
        *lpBuffer &= ~1; *lpBuffer |= (btData & 1); lpBuffer++; btData >>= 1;      // bit 4
        *lpBuffer &= ~1; *lpBuffer |= (btData & 1); lpBuffer++; btData >>= 1;      // bit 5
        *lpBuffer &= ~1; *lpBuffer |= (btData & 1); lpBuffer++; btData >>= 1;      // bit 6
        *lpBuffer &= ~1; *lpBuffer |= (btData & 1);
                // bit 7



}

////////////////////////////////////////////////////////////////////////////////////////
// Función para incrustar un byte bajo el algoritmo de 2 bits por byte
inline void CDMCore::PerformEmbed2BPB(LPBYTE lpBuffer, BYTE btData)
{

        *lpBuffer &= ~3; *lpBuffer |= (btData & 3); lpBuffer++; btData >>= 2;      // bit 0,1
        *lpBuffer &= ~3; *lpBuffer |= (btData & 3); lpBuffer++; btData >>= 2;      // bit 2,3
        *lpBuffer &= ~3; *lpBuffer |= (btData & 3); lpBuffer++; btData >>= 2;      // bit 4,5
        *lpBuffer &= ~3; *lpBuffer |= (btData & 3);
                // bit 6,7



}

////////////////////////////////////////////////////////////////////////////////////////
// Función para incrustar un byte bajo el algoritmo de 4 bits por byte
inline void CDMCore::PerformEmbed4BPB(LPBYTE lpBuffer, BYTE btData)
{

        int     original, diff, ndiff, bits;

        // Procesa los bits 0,1,2,3
        bits = btData & 15;
        original = *lpBuffer & 15;
        diff = bits - original;
        if ( diff > 8 )
        {
                ndiff = diff - 16;
                if ( *lpBuffer + ndiff >= 0 )
                        *lpBuffer += ndiff;
                else
                        *lpBuffer += diff;
        }
        else if ( diff < -8 )
        {
                ndiff = diff + 16;
                if ( *lpBuffer + ndiff <= 255 )
                        *lpBuffer += ndiff;
                else
                        *lpBuffer += diff;
        }
        else
                *lpBuffer += diff;

        // Ahora procede a los 4 bits más altos
        btData >>= 4; lpBuffer++;

        // Procesa los bits 4,5,6,7
        bits = btData & 15;
        original = *lpBuffer & 15;
        diff = bits - original;
        if ( diff > 8 )
        {
```

```
                ndiff = diff - 16;
                if ( *lpBuffer + ndiff >= 0 )
                        *lpBuffer += ndiff;
                else
                        *lpBuffer += diff;
        }
        else if ( diff < -8 )
        {
                ndiff = diff + 16;
                if ( *lpBuffer + ndiff <= 255 )
                        *lpBuffer += ndiff;
                else
                        *lpBuffer += diff;
        }
        else
                *lpBuffer += diff;



}

////////////////////////////////////////////////////////////////////////////////
// Función para extraer un string del buffer
void CDMCore::PerformExtract(LPBYTE lpData, LPCSTR lpBuffer, DWORD dwDataLength, DWORD
dwBufferLength, UINT uShiftAmount)
{
        PEXTRACT_PROCEDURE      pfnExtract;
        DWORD                           dwBufferHold = dwBufferLength >> uShiftAmount;
        DWORD                           dwCounter;
        LPCSTR                          lpEntry;
        UINT                            uPercent, uLastPercent = 0;

        // Escoge el algoritmo apropiado de acuerdo al nivel de incrustación
        switch(uShiftAmount)
        {
        case 3:
                pfnExtract = CDMCore::PerformExtract1BPB;
                break;
        case 2:
                pfnExtract = CDMCore::PerformExtract2BPB;
                break;
        case 1:
                pfnExtract = CDMCore::PerformExtract4BPB;
                break;
        default:
                ASSERT(FALSE);
        }

        // Extracción
        for ( dwCounter = 0 ; dwCounter < dwDataLength ; dwCounter++ )
        {
                // Busca en la localidad amplificada
                lpEntry = &lpBuffer[GetZoomedPosition(dwDataLength, dwBufferHold, dwCounter)
<< uShiftAmount];
                // Extrae un byte
                (this->*pfnExtract)(lpData, lpEntry);
                // Se prepara para el siguiente byte
                lpData++;

                // Calcula el avance
                uPercent = ( dwCounter + 1 ) * 100 / dwDataLength;
                if ( uPercent != uLastPercent )
                        SetProgress(uPercent);
                uLastPercent = uPercent;
        }
}

////////////////////////////////////////////////////////////////////////////////
// Función para extraer un byte en base al algoritmo de 1 bit por byte
inline void CDMCore::PerformExtract1BPB(LPBYTE lpData, LPCSTR lpBuffer)
```

```
{
        *lpData = *lpBuffer & 1; lpBuffer++;          // bit 0
        *lpData |= (*lpBuffer & 1) << 1; lpBuffer++; // bit 1
        *lpData |= (*lpBuffer & 1) << 2; lpBuffer++; // bit 2
        *lpData |= (*lpBuffer & 1) << 3; lpBuffer++; // bit 3
        *lpData |= (*lpBuffer & 1) << 4; lpBuffer++; // bit 4
        *lpData |= (*lpBuffer & 1) << 5; lpBuffer++; // bit 5
        *lpData |= (*lpBuffer & 1) << 6; lpBuffer++; // bit 6
        *lpData |= (*lpBuffer & 1) << 7;              // bit 7
}

////////////////////////////////////////////////////////////////////////////////////////
// Función para extraer un byte en base al algoritmo de 2 bits por byte
inline void CDMCore::PerformExtract2BPB(LPBYTE lpData, LPCSTR lpBuffer)
{
        *lpData = *lpBuffer & 3; lpBuffer++;          // bit 0,1
        *lpData |= (*lpBuffer & 3) << 2; lpBuffer++; // bit 2,3
        *lpData |= (*lpBuffer & 3) << 4; lpBuffer++; // bit 4,5
        *lpData |= (*lpBuffer & 3) << 6;              // bit 6,7
}

////////////////////////////////////////////////////////////////////////////////////////
// Función para extraer un byte en base al algoritmo de 4 bits por byte
inline void CDMCore::PerformExtract4BPB(LPBYTE lpData, LPCSTR lpBuffer)
{
        *lpData = *lpBuffer & 15; lpBuffer++;                 // bit 0,1,2,3
        *lpData |= (*lpBuffer & 15) << 4;                     // bit 4,5,6,7
}


////////////////////////////////////////////////////////////////////////////////////////
// Función para salvar el mapa de bits
BOOL CDMCore::SaveWindowsBitmap(LPDMBITMAP lpDMBitmap, LPCSTR lpszFileName)
{
        CFile                           file;
        LPWINBITMAP                     lpWinBitmap;
        BITMAPFILEHEADER        bfHeader;
        LONG                            i;
        DWORD                           dwBytesPerLine;
        LPBYTE                          lpLineBuffer;

        ASSERT(lpDMBitmap);

        // Genera temporalmente un mapa de bits de windows
        lpWinBitmap = GenerateWindowsBitmap(lpDMBitmap);
        if ( lpWinBitmap == NULL )
        {
                return FALSE;
        }

        // Calcula el número de bytes por línea
        dwBytesPerLine    =    lpWinBitmap    ->    biHeader.biSizeImage    /    lpWinBitmap    ->
biHeader.biHeight;

        // Asignación del buffer de línea
        lpLineBuffer = new BYTE[dwBytesPerLine];
        if ( lpLineBuffer == NULL )
        {
                m -> uErrorCode = DMERR_NOT_ENOUGH_MEMORY;
                ReleaseBitmap(lpWinBitmap);
                return FALSE;
        }

        // Crea el archivo
        CFileException ex;
        if   (   !file.Open(lpszFileName,   CFile::modeCreate|CFile::modeWrite|CFile::typeBinary,
&ex) )
        {
                m -> uErrorCode = DMERR_FILE_EXCEPTION;
                ex.GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
```

```
                ReleaseBitmap(lpWinBitmap);
                return FALSE;
        }

        // Rellena la estructura BITMAPFILEHEADER
        bfHeader.bfType = (UINT)'B' + ((UINT)'M' << 8);
        bfHeader.bfReserved1 = bfHeader.bfReserved2 = 0;
        bfHeader.bfOffBits = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);
        bfHeader.bfSize = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) + lpWinBitmap ->
biHeader.biSizeImage;

        try
        {
                // Escribe el encabezado del archivo y la información del encabezado
                file.Write(&bfHeader, sizeof(BITMAPFILEHEADER));
                file.Write(&lpWinBitmap -> biHeader, sizeof(BITMAPINFOHEADER));

                // Inicializa el indicador de avance
                m -> fProgressFactor = 1.0f;
                m -> uProgressStart = 0;
                SetProgress(0);

                // Escribe las líneas (horizontales)
                for ( i = 0 ; i < lpWinBitmap -> biHeader.biHeight ; i++ )
                {
                        // Prepara el buffer de línea
                        memcpy(lpLineBuffer,    &((LPCSTR)lpWinBitmap    ->    lpBuffer)[i    *
dwBytesPerLine], dwBytesPerLine);
                        file.WriteHuge(lpLineBuffer, dwBytesPerLine);

                        // Calcula el avance
                        SetProgress((i + 1) * 100 / lpWinBitmap -> biHeader.biHeight);
                }

                // Cierra el archivo
                file.Close();
        }
        catch(CFileException *pEx)
        {
                m -> uErrorCode = DMERR_FILE_EXCEPTION;
                pEx -> GetErrorMessage(m -> szErrorEx, MAX_ERROR_SIZE);
                pEx -> Delete();
                ReleaseBitmap(lpWinBitmap);
                delete []lpLineBuffer;
                return FALSE;
        }

        // Libera la memoria temporal
        ReleaseBitmap(lpWinBitmap);
        delete []lpLineBuffer;

        return TRUE;
}

////////////////////////////////////////////////////////////////////////////////////////////
// Función para detectar si un mapa de bits contiene un mensaje
BOOL CDMCore::DetectWatermark(LPDMBITMAP lpDMBitmap)
{
        WATERMARKINFO  wmInfo;
        DWORD                   dwPixelBytes;
        int                             i;

        // El mapa de bits ahora debe existir
        ASSERT(lpDMBitmap);

        // Calcula si el mapa de bits puede contener la información de encabezado del mensaje
        // Luego trata de extraer el encabezado
        // Enseguida prueba desde 4 bits por byte hasta 1 bit por byte
        dwPixelBytes = lpDMBitmap -> width * lpDMBitmap -> height * sizeof(DMRGB);
        for ( i = 1 ; i <= 3 ; i++ )
        {
```

```
                if ( dwPixelBytes > (sizeof(WATERMARKINFO) << i) )
                {
                        m -> bProgressEnabled = FALSE;
                        PerformExtract((LPBYTE)&wmInfo,     (LPCSTR)lpDMBitmap     ->     lpBitmap,
sizeof(WATERMARKINFO), sizeof(WATERMARKINFO) << i, i);
                        m -> bProgressEnabled = TRUE;
                        // Verifica la firma
                                // código omitido por seguridad
                                // Se encontró!
                                return TRUE;
                }
                else
                        // La imagen no tiene espacio suficiente para contener el encabezado de
un mensaje
                        return FALSE;
        }

        // Esta imagen no tiene un mensaje incrustado
        return FALSE;
}

////////////////////////////////////////////////////////////////////////////////////////////
// Función para recibir el índice de avance de los procesos de
// encriptamiento/desencriptamiento
void CDMCore::SetProgress(UINT uPercent)
{
        static UINT uLastPercent = 65535;
        UINT            uResult;

        // Aborta si no es necesario calcular el avance
        if ( !m -> bProgressEnabled )
                return;

        // Obtiene el avance absoluto
        uResult = m -> uProgressStart + (UINT)(uPercent * m -> fProgressFactor);
        if ( uResult > 100 )
                uResult = 100;

        // No hay necesidad de establecerlo si no ha cambiado
        if ( uResult == uLastPercent )
                return;

        // Lo establece
        m -> pMainWnd -> SendMessage(DMMSG_SET_PROGRESS, uResult, 0);

        uLastPercent = uResult;
}

////////////////////////////////////////////////////////////////////////////////////////////
// Función para obtener el índice de avance del proceso de compresión
BOOL CDMCore::CompressSetProgress(DWORD dwOutputed)
{
        static UINT uLastPercent = 65535;
        UINT            uResult;

        // Aborta si no es necesario calcular el avance
        if ( !m -> bProgressEnabled )
                return TRUE;

        // Mira si el número total de bytes de salida son más que el espacio disponible
        if ( dwOutputed > m -> dwProgressTotalLength )
                // Notifica al compresor que se detenga
                return FALSE;

        // Obtiene el avance absoluto
        uResult = m -> uProgressStart + (UINT)(dwOutputed * 100 / m -> dwProgressTotalLength *
m -> fProgressFactor);
        if ( uResult > 100 )
                uResult = 100;
```

```
        // No hay necesidad de establecerlo si no ha cambiado
        if ( uResult == uLastPercent )
                return TRUE;

        // Lo establece
        m -> pMainWnd -> SendMessage(DMMSG_SET_PROGRESS, uResult, 0);

        uLastPercent = uResult;
        return TRUE;
}

////////////////////////////////////////////////////////////////////////////////
// Función para recibir el índice de avance del proceso de descompresión
void CDMCore::UncompressSetProgress(UINT uPercent)
{
        static UINT uLastPercent = 65535;
        UINT        uResult;

        // Aborta si no es necesario calcular el avance
        if ( !m -> bProgressEnabled )
                return;

        // Obtiene el avance absoluto
        uResult = m -> uProgressStart + (UINT)(uPercent * m -> fProgressFactor);
        if ( uResult > 100 )
                uResult = 100;

        // No hay necesidad de establecerlo si no ha cambiado
        if ( uResult == uLastPercent )
                return;

        // Lo establece
        m -> pMainWnd -> SendMessage(DMMSG_SET_PROGRESS, uResult, 0);

        uLastPercent = uResult;
}


////////////////////////////////////////////////////////////////////////////////
// Función para recuperar el tipo de mensaje incrustado
UINT CDMCore::GetWatermarkType(LPWATERMARK lpWM)
{
        ASSERT(lpWM);
        LPBYTE pExtension = lpWM -> lpInfo -> btExtension;
        if ( pExtension[0] == 0 && pExtension[1] == 0 && pExtension[2] == 0 )
                // Si son ceros, se trata de texto puro
                return WT_TEXT;
        // De lo contrario se trata de un archivo importado
        return WT_FILE;
}

////////////////////////////////////////////////////////////////////////////////
// Función para recuperar el grado de incrustamiento
UINT CDMCore::GetShiftAmount(UINT uCompressionLevel)
{
        switch(uCompressionLevel)
        {
        case DMCOMPRESS_LOW:   // 1 bit por byte
                return 3;
        case DMCOMPRESS_NORMAL:        // 2 bits por byte
                return 2;
        case DMCOMPRESS_HIGH:  // 4 bits por byte
                return 1;
        }
        return 2;                               // Grado default: 2 bits por byte
}
```

[1] Del inglés *A Block Complexity based Data Embedding*. Incrustación de datos basada en la complejidad de bloques. Método presentado por primera vez por Hioki Hirohisa en *Pacific Rim Workshop on Digital Steganography 2002*, Kitakyushu, Japón.

[2] En su artículo "Robustness of Copyright Marking Systems", Scot Cravver hace una interesante reflexión acerca de la importancia de una interfaz apropiada en sistemas de seguridad de marca de agua. Los mismos principios son aplicables a un sistema de seguridad esteganográfico. A continuación se traduce parte de su escrito:

> "El típico usuario normalmente tiene un entendimiento limitado de los mecanismos subyacentes en la marca de agua y no desea gastar horas de entrenamiento para usar una función de su software de seguridad. La marca de agua se debe comportar como una caja negra donde el usuario introduce la imagen original y por algún proceso mágico obtiene la imagen marcada. No debiera ser necesario ningún entendimiento particular por parte del usuario. Considerando este factor humano, se puede ver que la interfaz del usuario es un componente crucial dentro de una arquitectura de seguridad. Whitten y Tygar demostraron como la interfaz defectuosa de PGP limitaba la seguridad del sistema. La interfaz debe presentarse al usuario con un modelo claro de los efectos de la marca de agua y protegerlo de un mal uso..."

[3] Este fenómeno fue descubierto por el físico Ernst Mach y es en su honor que el efecto adquiere su nombre.

5

Desempeño del sistema propuesto

"Ninguna cantidad de pruebas podrían demostrar que estoy en lo correcto; pero una sola prueba podría demostrar mi error"

Albert Einstein

5.1 Introducción

Ahora que hemos revisado qué es la esteganografía y algunas técnicas para lograrla, midamos qué tan bien se comporta el sistema propuesto DigiMarker.

Independientemente del tipo de datos que se envíe como mensaje secreto, la evaluación de un sistema esteganográfico debe incluir:

- **Cantidad de información incrustada.** Éste es un parámetro muy importante porque determinará no sólo el tamaño del mensaje sino, quizá más importante aún, el tamaño necesario de la portadora para que sea útil. Ese tamaño debe caer dentro del rango de uso normal para no levantar sospechas.

- **Fortaleza del mensaje incrustado.** ¿Qué pasa si en el transporte la imagen es atacada? De acuerdo con lo expuesto en el capítulo 4, es deseable que el mensaje sea fácilmente destruible.

- **Perceptibilidad del mensaje.** En otras palabras quiere decir qué tanto se nota que hay un mensaje oculto. Entre menos perceptibilidad, mejor. El sistema esteganográfico perfecto presenta perceptibilidad nula.

- **Seguridad del mensaje.** En el remoto caso de robo del diseño esteganográfico, ¿qué seguridad hay de preservar el contenido de los mensajes? En este punto el encriptamiento juega un papel muy importante (ver capítulo 2).

Al meditar en estos parámetros, nos damos cuenta de que para realizar una prueba justa de rendimiento, deben tomarse en cuenta muchos tipos distintos de datos. Por ejemplo, para el caso de DigiMarker, se incluye una fase de compresión. Como el lector seguramente sabe, el índice de compresión varía de acuerdo a la naturaleza de los datos a comprimir. El hecho de

que DigiMarker incluya este módulo lo hace una aplicación muy completa, pero ciertamente resulta más difícil evaluar su desempeño, ya que es necesario aumentar el abanico de pruebas.

Como es de esperarse, existe un compromiso entre el tamaño de mensaje y el grado de perceptibilidad. En este aspecto DigiMarker también presenta una gran ventaja contra otros sistemas esteganográficos: se puede configurar dinámicamente para balancear ese compromiso. No obstante, esta cualidad también contribuye a la dificultad para evaluarlo objetivamente.

5.2 Formas de evaluación

Al evaluar la perceptibilidad, se puede recurrir tanto al *análisis subjetivo* como a las *mediciones precisas* y *automatizadas*.

El análisis subjetivo se puede llevar a cabo tanto por personas completamente ajenas a esta disciplina como por personas especializadas que tengan un agudo sentido de análisis visual. Cuando se trata de personas neófitas estamos hablando del extremo de la subjetividad: aquella persona que simplemente mira con detenimiento una fotografía y entonces concluye si le parece natural o sospechosa. Por absurda que parezca esta forma, el sentido común a veces juega un papel más importante del que se le suele conceder. Cuando se trata de personas especializadas, no sólo mirarán a simple vista la imagen; también querrán mirar su histograma y algunos otros análisis realizados por computadora. ¿Por qué pues se le sigue considerando análisis subjetivo si se emplean mediciones digitales precisas? Por la razón de que el criterio que sigue normando no es un número que represente el umbral del "pasa, no pasa" sino que está operando ese agudo sentido para detectar cosas anormales en la fotografía, pero auxiliándose por herramientas computarizadas.

Cuando se usa un examen subjetivo, se debe seguir un protocolo de análisis que describa la prueba y el procedimiento de evaluación. Tales pruebas suelen ser de dos pasos. En el primer paso las imágenes a analizar se ordenan de mejores a peores (es decir, de menos a más sospechosas de contener mensajes secretos). En el segundo paso, se pide al evaluador que califique la perceptibilidad de ruido extraño en las imágenes. Esta evaluación puede ser basada, por ejemplo, en la recomendación BT.500 para medición de calidad de la ITU-R.[1] Trabajos realizados dentro del proyecto europeo OCTALIS[2] han demostrado que individuos de diferente contexto y experiencia, por ejemplo fotógrafos profesionales e investigadores, generan resultados sumamente diferentes al aplicar los análisis subjetivos a imágenes. Este hecho complica la obtención de resultados fehacientes.

El tráfico de imágenes en la red internacional es tan grande que es absolutamente imposible estar manualmente a la caza de elementos de sospecha. Las mediciones automatizadas para detectar portadoras esteganográficas pueden recurrir a medidas de proporción señal a ruido y otras medidas que pueden levantar cuestionamientos.

En estas condiciones, las medidas cuantitativas de distorsión son mucho más eficientes y permiten comparaciones más justas entre diversos métodos, ya que los resultados no

dependen de evaluaciones subjetivas. Además, conceden la oportunidad de automatizar el proceso de análisis. Actualmente, las medidas de distorsión más populares en el campo de codificación y compresión de imágenes y video son la proporción señal a ruido (SNR) y el pico de la proporción señal a ruido (PSNR). A menudo se miden en decibeles (dB): SNR(db) = $10\log_{10}$(SNR). Sin embargo es bien sabido que estas medidas de diferencia en distorsión no están bien correlacionadas con el sistema visual humano. Esto puede representar un problema para su aplicación automatizada en el campo de las marcas de agua digitales o de la esteganografía. En años recientes, se ha dedicado mucha investigación para encontrar medidas adaptadas al sistema visual humano y es probable que en el futuro las pruebas de rendimiento ya puedan emplear las nuevas medidas.[3]

Al analizar ambos enfoques de análisis, se llega a la conclusión siguiente: el esquema de trabajo ideal es una combinación donde se programan robots cazadores de posibles portadoras, se discriminan las menos relevantes y luego se analiza manualmente las imágenes más potenciales.[4] Como es de esperarse, este proceso es toda una disciplina y se conoce con el nombre de *estegoanálisis*. Algunos países como la India se han destacado en esta disciplina.

5.2.1 Breve introducción al estegoanálisis

Las imágenes son el medio anfitrión más popular para la comunicación esteganográfica. Junto con el aumento en complejidad de los sistemas esteganográficos y la mejora en los estándares de codificación de imágenes, el estegoanálisis se ha vuelto sumamente difícil. Sin embargo, muchos de los sistemas esteganográficos no son perfectos y pueden dejar rastros reconocibles de una forma o de otra. Se requiere un ataque sistemático para aprovechar este hecho y poder capturar aún los más diminutos vestigios que un mensaje secreto puede dejar en una imagen portadora. Resulta muy útil tener información sobre la herramienta esteganográfica, el dominio de incrustación, características de la portadora y la capacidad de almacenamiento para la detección y extracción de mensajes secretos. En ausencia de tales datos, el estegoanalista está forzado a depender sólo de las anormalidades introducidas en la imagen debido a la incrustación de datos externos. La inspección visual detallada y el análisis estadístico pueden ayudar a sacar a la luz características inusuales de la imagen sospechosa de ser portadora esteganográfica.

En años recientes, la complejidad de los sistemas esteganográficos ha sido un asunto de importancia para distintas agencias de investigación y aplicación de la ley. Inicialmente, el objetivo de la esteganografía era simplemente ocultar un mensaje del observador casual, pero hoy en día un sistema esteganográfico serio está fuertemente equipado en términos de imperceptibilidad, fortaleza del mensaje contra ataques y alta capacidad de incrustación. Un sistema esteganográfico profesional se asegura de que su proceso de incrustación no excede los umbrales sensoriales y estadísticos para el medio digital que ocupa como portador. De este modo pasa invisible ante los robots cazadores de portadores esteganográficos.

De todos los medios digitales existentes, la esteganografía en imágenes ha recibido la mayor atención en los últimos años.[5] Las imágenes digitales pueden recibir grandes cantidades de datos externos sin crear distorsiones perceptibles. El gran número de píxeles en una imagen de alta calidad y el amplio rango de colores que el ojo humano no pude diferenciar, proveen un excelente terreno para probar distintas estrategias esteganográficas. Todos los días se publican imágenes en la red y millones se envían por correo electrónico, desde fotos de cumpleaños hasta imágenes de ultrasonido. Por esta razón, grandes esfuerzos del estegoanálisis se orientan al análisis de imágenes digitales.

El estegoanálisis es el intento por descubrir y deshabilitar mensajes ocultos en medios digitales. A pesar de ser una disciplina de difícil éxito debido a los nuevos sistemas esteganográficos, su esperanza reside en el hecho de que muchos sistemas esteganográficos no son perfectos y dejan rastros detrás de ellos. El estegoanálisis consiste de las siguientes tareas, dependiendo de la naturaleza de la aplicación:

- Detección
- Extracción
- Borrado / deshabilitado
- Modificación inteligente
- Reinserción

La detección es la primera y más importante parte del estegoanálisis. La extracción del mensaje secreto se requiere para conocer el contenido y propósito de la comunicación oculta. A menudo se usa para la recopilación de evidencia en contra de una actividad ilegal. Un adversario activo puede extraer el mensaje para deshabilitarlo o mejor aún, reemplazarlo por otro mensaje.

Dependiendo de la información disponible, se pueden prever distintos tipos de ataque. Los casos más prácticos tienen que ver con el análisis de una o varias imágenes sospechosas de las que no se conoce ningún otro dato. Este ataque se conoce como *sólo estego* y es muy difícil de realizar. Si el conjunto de imágenes provienen de una sola fuente, resulta provechoso estudiarlas para encontrar peculiaridades y similitudes. Para el caso de una imagen aislada, la tarea consiste en identificar características únicas que no se encuentran en imágenes normales.

El estegoanálisis, comparado con el criptoanálisis, es un campo joven e inmaduro.[6] El gobierno, la industria y las organizaciones de investigación alrededor del mundo han sentido la necesidad de madurar el estegoanálisis. Es relativamente poco el material de investigación publicado que está disponible actualmente. La CIA, el Pentágono, el laboratorio de investigación de la Fuerza Aérea (de los EUA) recientemente han fundado proyectos de investigación en este campo. Por eso la importancia de que investigadores mexicanos incursionemos en él.

5.2.2 Medición aritmética

Es posible definir la capacidad de incrustación y la proporción señal a ruido como indicadores de la eficiencia de un método esteganográfico en imágenes digitales. La *capacidad de incrustación* se define al evaluar la cantidad de datos que se pueden incrustar en una imagen portadora. Se define así:

$$\frac{\text{Máxima información posible de incrustar}}{\text{Tamaño de la imagen portadora}} \times 100 \quad (\%)$$

Una ventaja de DigiMarker es la capacidad de configurarlo para variar la capacidad de incrustación. La capacidad de incrustación es proporcional al rastro dejado en la portadora. A mayor capacidad de incrustación, mayor alteración en la imagen. A menor capacidad de incrustación, menor alteración en la imagen. Ese rastro se puede medir evaluando la calidad de la portadora después de haberse incrustado el mensaje. Esa calidad suele medirse con el pico de la proporción pico señal a ruido en la forma siguiente:[7]

$$PSNR = 20\log_{10}\frac{255}{\sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(I_i' - I_i)^2}}$$

donde I_i e I_i' son respectivamente los valores de un píxel en la imagen original y del mismo píxel después de la incrustación y n es el número de píxeles en la imagen. Entre más grande sea esta medida, menos alterada está la portadora, acercándose más al propósito esteganográfico. En la figura 38 también se indica que el PSNR para DigiMarker es mejor que para el método BPCS.

A continuación se incluye una secuencia de comandos en el intérprete del producto *MatLab* para calcular el PSNR de una imagen portadora esteganográfica. Para este ejemplo se requiere que el archivo llamado "original.bmp" sea la imagen original antes de ser modificada y el archivo llamado "modificado.bmp" sea el portador esteganográfico con el mensaje ya incrustado. Nótese que las imágenes se convierten previamente a tonos de grises.

```
original=imread('original.bmp');
original=rgb2gray(original);
original=double(original);

modificado=imread('modificado.bmp');
modificado=rgb2gray(modificado);
modificado=double(modificado);

tamano=size(original);
n=tamano(1)*tamano(2);

dif=modificado-original;
cuadrado=dif.^2;
suma=sum(sum(cuadrado));
denominador=sqrt(suma/n);
```

```
PSNRdb=20*log10(255/denominador)
```

5.2.3 Medición subjetiva

Como mencioné anteriormente, la inspección visual de las imágenes juega un papel importante dentro del estegoanálisis. En la siguiente sección, presento varias imágenes portadoras antes y después de servir de recipientes para los mensajes secretos. El lector podrá observar que algunas de ellas son visualmente indistinguibles, mientras que en otras se nota el ruido inducido.

5.3 Evaluación de DigiMarker

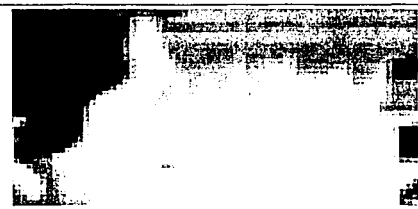
He seleccionado siete fotografías con muy diversas características para evaluar DigiMarker. Algunas de ellas son fotos viejas que han sido escaneadas en equipo casero. Otras provienen de fotografías digitales. Las hay en alto y bajo contraste, a baja y mediana resolución. En fin, la idea es proveer una gama amplia de posibilidades.

Cabe mencionar que las fotos presentadas aquí son necesariamente grandes a fin de que puedan ser impresas con cierta calidad. En el uso real de DigiMarker las fotografías a emplear pueden ser substancialmente pequeñas y obtendremos el mismo resultado. La diferencia está en el número de puntos por pulgada que una imagen debe tener para verse bien. Si la imagen se verá en un monitor de computadora –como es el caso de DigiMarker-, 72 ppp serán suficientes. Pero si la imagen ha de ser impresa –como en esta tesis- entonces debemos pensar en aumentar el número de puntos por pulgada para que las imágenes sean nítidas. Evidentemente este cambio de resolución afecta notablemente el tamaño de la imagen.

DigiMarker tiene 3 configuraciones posibles, llamadas *nivel bajo*, *medio* y *alto*. El nivel bajo incrusta los mensajes de menor tamaño pero consigue que los cambios realizados a la portadora sean completamente invisibles al ojo humano. El nivel bajo (de incrustación) es el que provee mayor seguridad. El nivel medio es un equilibrio entre capacidad de incrustación e invisibilidad del mensaje. El nivel medio es apropiado para portadoras de gran calidad esteganográfica (bajo contraste, tonos claros, etc.- ver capítulo 4) que requieran transportar mensajes grandes. El nivel alto se usa en situaciones donde la seguridad no es el factor más importante sino el tamaño del mensaje a enviar. El nivel alto, combinado con el módulo de compresión, a menudo logra incrustar mensajes cuyo tamaño original es mayor que el tamaño de la portadora misma.

Se invita al lector a que examine con detenimiento las imágenes y datos numéricos presentados en las páginas siguientes, a fin de evaluar por sí mismo esta propuesta de solución esteganográfica.

Imagen Original. Nombre: *Bahia*
Píxeles: 747 x 320, 96 ppp Tamaño en disco: 701 kB

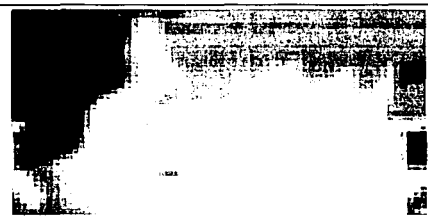


Acercamiento a 1500%

Bahia en escala de grises

Histograma

Imagen esteganográfica
Tamaño de mensaje: 299.5 kB
Tipo : Documento Word, 58 páginas

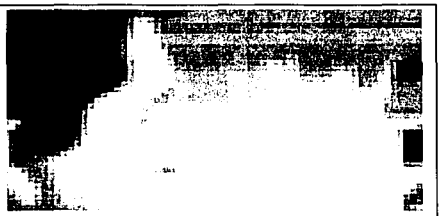


Acercamiento a 1500%
PSNR: 51.78 dB
Incrustación: 42.72% Grado: Bajo

Bahia más mensaje (nivel bajo) en escala de grises

Histograma

Imagen esteganográfica
Tamaño de mensaje: 709 kB
Tipo : Documento Word, 129 páginas

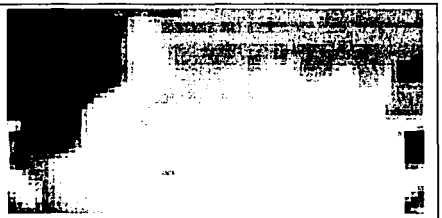


Acercamiento a 1500%
PSNR: 46.67 dB
Incrustación: 101.14% Grado: Medio

Bahía más mensaje (nivel medio) en escala de grises

Histograma

Imagen esteganográfica
Tamaño de mensaje: 1,871 kB
Tipo : Documento Word, 296 páginas

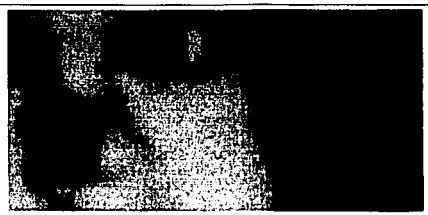


Acercamiento a 1500%
PSNR: 38.16 dB
Incrustación: 266.90% Grado: Alto

Bahía más mensaje (nivel alto) en escala de grises

Histograma

Imagen Original. Nombre: *Bebé*
Píxeles: 600 x 400, 72 ppp Tamaño en disco: 703 kB

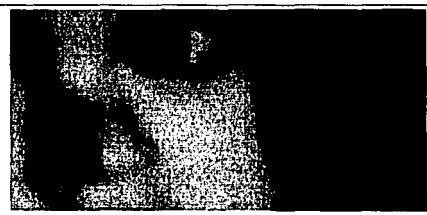


Acercamiento a 1500%

Bebé en escala de grises

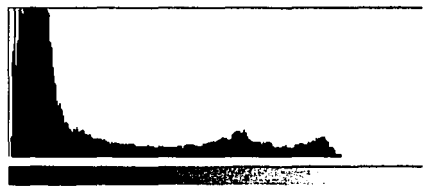
Histograma

Imagen esteganográfica
Tamaño de mensaje: 227 kB
Tipo : Archivo ejecutable



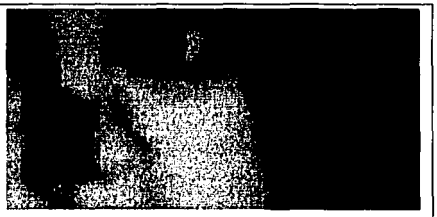
Acercamiento a 1500%
PSNR: 52.25 dB
Incrustación: 32.29% Grado: Bajo

Bebé más mensaje (nivel bajo) en escala de grises



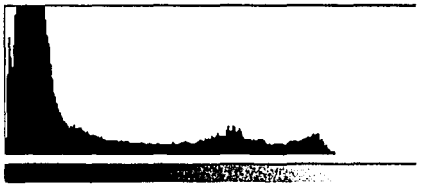
Histograma

Imagen esteganográfica
Tamaño de mensaje: 405 kB
Tipo : Archivo ejecutable



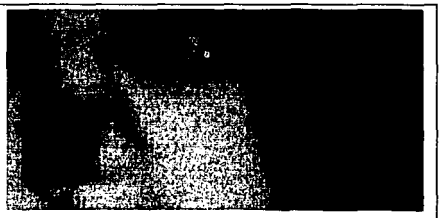
Acercamiento a 1500%
PSNR: 47.09 dB
Incrustación: 57.61% Grado: Medio

Bebé más mensaje (nivel medio) en escala de grises



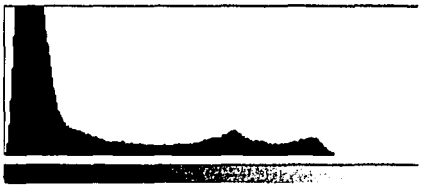
Histograma

Imagen esteganográfica
Tamaño de mensaje: 756 kB
Tipo : Archivo ejecutable



Acercamiento a 1500%
PSNR: 37.70 dB
Incrustación 107.54% Grado: Alto

Bebé más mensaje (nivel alto) en escala de grises



Histograma

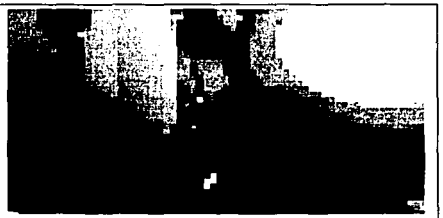
Imagen Original. Nombre: Cena
Píxeles: 307 x 231, 144 ppp Tamaño en disco: 209 kB

Acercamiento a 1500%

Cena en escala de grises

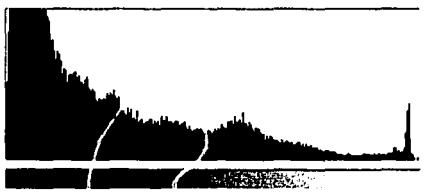
Histograma

Imagen esteganográfica
Tamaño de mensaje: 21 kB
Tipo : Fotografía 24 bits JPG comprimida



Acercamiento a 1500%
PSNR: 53.20 dB
Incrustación: 10.05% Grado: Bajo

Cena más mensaje (nivel bajo) en escala de grises



Histograma

Imagen esteganográfica
Tamaño de mensaje: 51 kB
Tipo : Fotografía 24 bits JPG comprimida

Acercamiento a 1500%
PSNR: 47.24 dB
Incrustación: 24.40% Grado: Medio

Cena más mensaje (nivel medio) en escala de grises

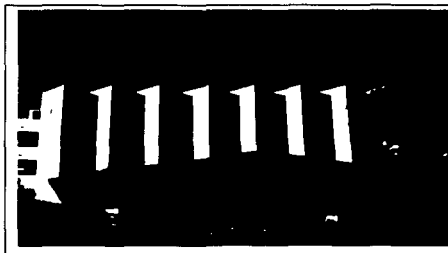
Histograma

Imagen esteganográfica
Tamaño de mensaje: 104 kB
Tipo : Fotografía 24 bits JPG comprimida

Acercamiento a 1500%
PSNR: 37.26 dB
Incrustación: 49.76% Grado: Alto

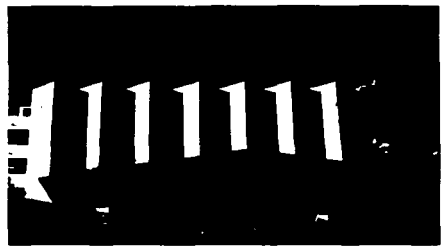
Cena más mensaje (nivel alto) en escala de grises

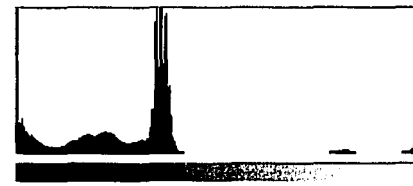
Histograma



Acercamiento a 1200%

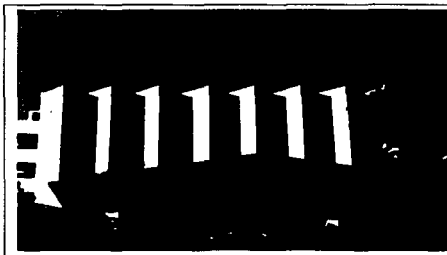
Imagen Original. Nombre: *Hotel*
Pixeles: 780 x 520, 300 ppp Tamaño en disco: 1,189 kB

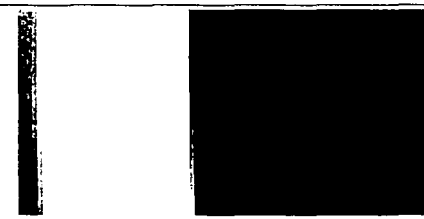




Histograma

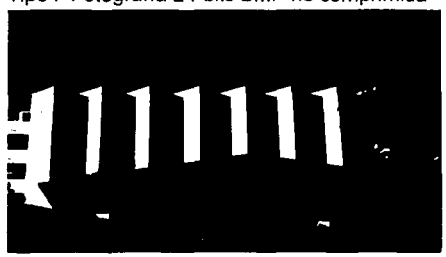
Hotel en escala de grises

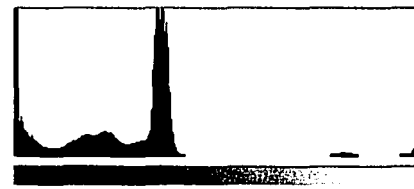




Acercamiento a 1200%
PSNR: 52.85 dB
Incrustación: 18% Grado: Bajo

Imagen esteganográfica
Tamaño de mensaje: 214 kB
Tipo : Fotografía 24 bits BMP no comprimida





Histograma

Hotel más mensaje (nivel bajo) en escala de grises

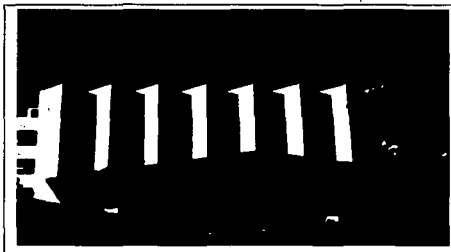
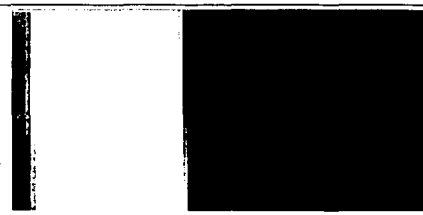
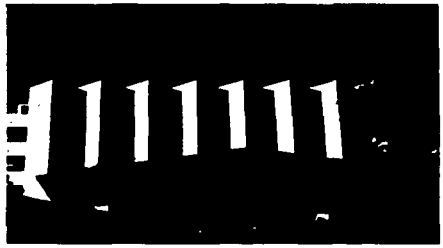


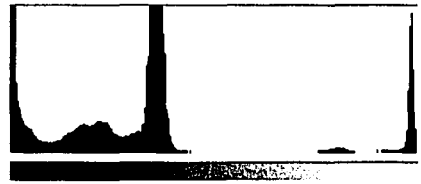
Imagen esteganográfica
Tamaño de mensaje: 676 kB
Tipo : Fotografía 24 bits BMP no comprimida



Acercamiento a 1200%
PSNR: 46.47 dB
Incrustación: 56.85% Grado: Medio



Hotel más mensaje (nivel medio) en escala de grises



Histograma

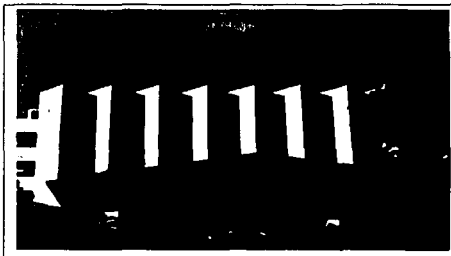
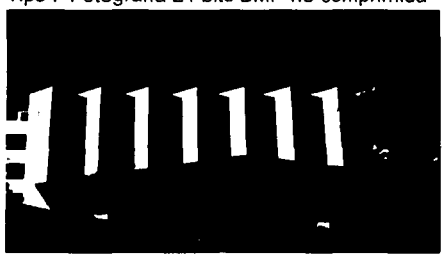
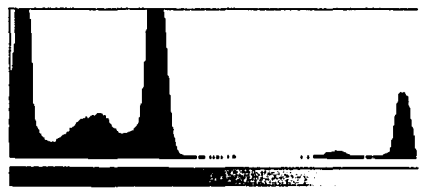


Imagen esteganográfica
Tamaño de mensaje: 1,067 kB
Tipo : Fotografía 24 bits BMP no comprimida

Acercamiento a 1200%
PSNR: 34.26 dB
Incrustación: 89.74% Grado: Alto



Hotel más mensaje (nivel alto) en escala de grises

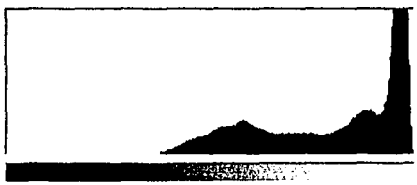


Histograma

Imagen Original. Nombre: *Mujer SLP*
Píxeles: 780 x 520, 300 ppp Tamaño en disco: 1,189 kB

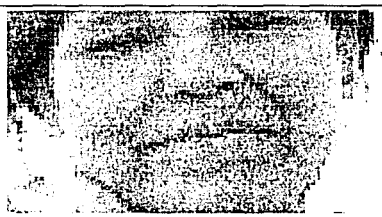
Acercamiento a 1200%

Mujer SLP en escala de grises



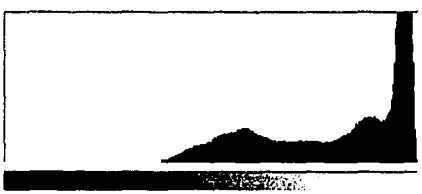
Histograma

Imagen esteganográfica
Tamaño de mensaje: 156 kB
Tipo : Video, 32 cuadros (240 x 128) y pista de audio



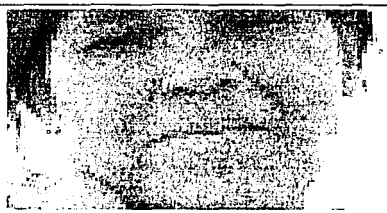
Acercamiento a 1200%
PSNR: 52.45 dB
Incrustación: 13.12% Grado: Bajo

Mujer SLP más mensaje (nivel bajo) en grises



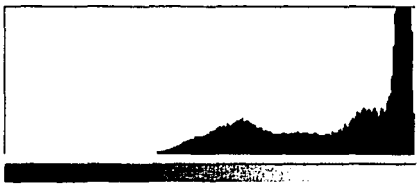
Histograma

Imagen esteganográfica
Tamaño de mensaje: 306 kB
Tipo : Video, 351 cuadros (240 x 180) y pista de audio



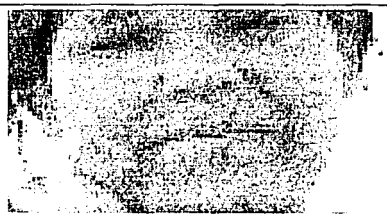
Acercamiento a 1200%
PSNR: 47.05 dB
Incrustación: 25.74% Grado: Medio

Mujer SLP más mensaje (nivel medio) en grises



Histograma

Imagen esteganográfica
Tamaño de mensaje: 506 kB
Tipo : Video, 339 cuadros (320 x 240) y pista de audio



Acercamiento a 1200%
PSNR: 37.66 dB
Incrustación: 42.56% Grado: Alto

Mujer SLP más mensaje (nivel alto) en grises

Histograma

Las pruebas reflejadas en las tablas anteriores muestran que se probó DigiMarker con distintos tipos de mensajes:

- Documentos de texto con formato (Word)
- Programas ejecutables
- Fotografías comprimidas, color verdadero
- Fotografías no comprimidas, color verdadero
- Video a distintas resoluciones y cuadros por segundo

Resulta casi paradójico pensar en transportar secretamente (¡PSNR > 50db!) toda una secuencia de video dentro de una imagen estática de tamaño modesto. Afortunadamente esto es posible con la técnica empleada por DigiMarker.

---

[1] Del inglés *International Telecommunication Union*. ITU–R se refiere en particular al sector de radiocomunicaciones. Este organismo, entre otras cosas, publica una enorme cantidad de recomendaciones. La recomendación referida, actualizada en junio de 2002, se titula *Methodology for the subjective assessment of the quality of television pictures*. Se puede consultar en http://www.itu.int/ITU-R/.

[2] Del inglés *Offer of Content through Trusted Access Links*. Es un proyecto europeo que integra un enfoque global para un acceso equitativo y protegido por derechos de autor en la distribución de señales de video e imágenes estáticas. Es de nuestro interés por la amplia investigación que han hecho de marcas de agua digitales. Se puede consultar en http://ltswww.epfl.ch/research/description/octalis.html.

[3] Katzenbeisser, Stefan y Fabien Petitcolas. *Information Hiding Techniques for Steganography and Digital Watermarking*, Artech House, 2000, pp. 111.

[4] Provos y Honeyman. "Detecting Steganographic Content on the Internet", *Technical Report CITI 01-11*, University of Michigan, 2001.

[5] Bender, Gruhl, Morimoto y Lu. "Techniques for Data Hiding", *IBM Systems Journal*, Vol. 35, 1996, pp. 313-336.

[6] Pal, Saibal, P.K. Saxena y S.K. Muttoo. "A Systematic Approach to Steganalysis of Images", *Proceedings of Pacific Rim Workshop on Digital Steganography 2002*, Kitakyushu, Japón, p. 186.

[7] Hioki, Hirohisa. "A Data Embedding Method using BPCS Principle with new Complexity Measures", *Proceedings of Pacific Rim Workshop on Digital Steganography 2002*, Kitakyushu, Japón, p. 44.

6

Conclusiones

"Lo importante es nunca dejar de preguntar"

Albert Einstein

6.1 Resumen teórico

Hemos visto que aunque la esteganografía, las marcas de agua y la criptografía, todas son disciplinas de la seguridad de la información, tienen características bien definidas que las distinguen unas de otras. La presente investigación se centra fundamentalmente en la esteganografía pero también revisa ligeramente la criptografía y las marcas de agua.

La esteganografía es el hecho de ocultar la comunicación entre dos o más partes sirviéndose de una portadora o contenedor del mensaje secreto. Aprendimos que es posible echar mano de técnicas criptográficas para que al combinarlas con el proceso esteganográfico, aumente el nivel total de seguridad del sistema de comunicación secreta. También se puede mejorar el desempeño del sistema si se agrega un módulo de compresión de datos. DigiMarker, el sistema propuesto en esta tesis, incorpora todas estas tecnologías comportándose conforme al diagrama de flujo de datos de la figura 43.

DigiMarker se basa en una mejora a la tecnología llamada LSB (bits menos significativos) para la incrustación de datos. El módulo de encriptamiento es simétrico con una llave de 128 bits, basado en el método DES (estándar de encriptamiento de datos). El módulo compresor está basado en algunas utilerías de dominio público que han demostrado ser sumamente eficientes, en algunas ocasiones mejores que programas comerciales como WinZip.

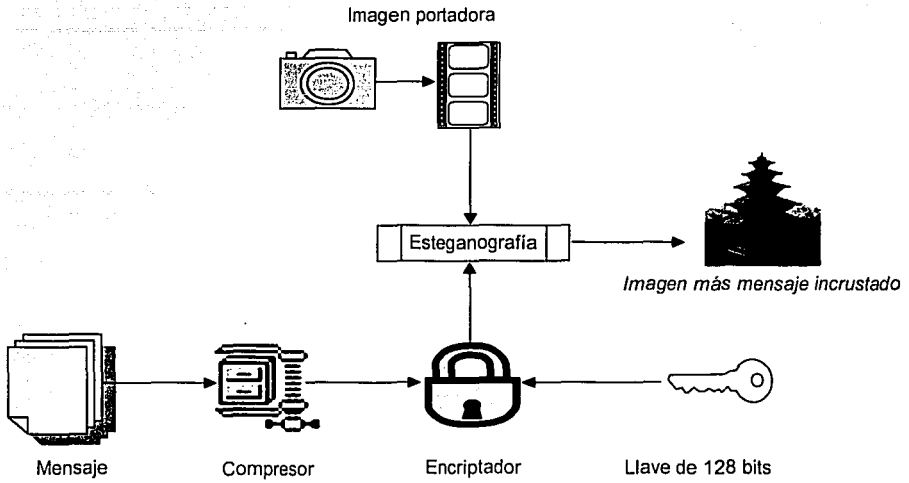


Figura 43. Flujo de datos en DigiMarker, proceso de incrustación.

6.2 Limitaciones de DigiMarker

La imagen portadora usada por DigiMarker debe ser una imagen de color verdadero, 24 bits, en formato BMP no comprimido. Hay varias razones para haber elegido este formato, siendo las más importantes:

- El formato BMP no cambia su estructura conforme pasa el tiempo, a diferencia de otras especificaciones como JPEG. Esto hace que DigiMarker no requiera mantenimiento posterior para mantenerse al día con los cambios de especificaciones.

- El formato no comprimido es el que proporciona mayor información redundante, proveyendo así amplio espacio para la incrustación de mensajes esteganográficos, a diferencia de formatos comprimidos como JPEG que limitan fuertemente el espacio disponible.

- El método LSB, sobre el cual gira la tecnología de DigiMarker, se acopla perfectamente a la estructura del formato BMP, sin necesidad de hacer transformaciones al dominio de la frecuencia. Esto produce que el tiempo de proceso para incrustar y extraer mensajes esteganográficos sea sumamente reducido.

- El formato BMP ha sido de uso muy común prácticamente desde el inicio de las fotografías digitales; hoy en día todavía se usa extensamente, especialmente en imágenes de alta resolución y se puede prever que se seguirá usando durante mucho tiempo más. Prácticamente todas las aplicaciones gráficas existentes soportan este formato.

No obstante que hay suficientes y sólidas razones para haber elegido el formato BMP, esto constituye también un límite en el alcance de DigiMarker. Algunas aplicaciones comerciales esteganográficas permiten que la portadora sea GIF o JPEG, ampliando el margen de uso (aunque sacrificando, por supuesto, el tamaño del mensaje a incrustar).

Otro límite digno de mencionar es su restricción para trabajar exclusivamente con fotografías de color verdadero, a 8 bits por color, sin paleta de colores. Igual que en el punto anterior, existen razones técnicas para haber decidido trabajar con estas restricciones, pero de cualquier manera esto es un límite actual de DigiMarker: el no poder trabajar con formatos gráficos basados en paletas de colores, siendo uno de los más comunes el formato GIF.

6.3 Trabajos futuros

Después de muchas horas de usar DigiMarker, no sólo en las pruebas controladas de laboratorio sino en la rutina de la aplicación real, he llegado a la conclusión de que sería útil e interesante trabajar en las siguientes mejoras:

1) En este momento, DigiMarker permite que el mensaje sea teclado por el usuario o bien que el usuario señale un archivo que contenga el mensaje a incrustar. Aunque el tipo de archivo no está restringido, el usuario sólo puede seleccionar un archivo a la vez. Una mejora inmediata sería permitir la inclusión de dos o más archivos para ser incrustados en una misma fotografía. Actualmente, esto se puede lograr si el usuario previamente comprime y empaqueta en un solo archivo todos los archivos que desea incrustar (usando un programa como WinZip), pero lo ideal sería que DigiMarker le ahorrara ese paso manual.

Evidentemente, la implementación de esta característica tiene un costo que sería necesario evaluar para determinar si el beneficio amerita el sacrificio tecnológico. El lector debe recordar que un objetivo central de la tecnología esteganográfica es no dejar rastro de la presencia del mensaje. El estegoanálisis busca con mucho detalle cualquier semejanza entre secuencias de fotografías sospechosas. Cualquier aplicación esteganográfica –y DigiMarker no es la excepción- altera en cierto modo algunos bits de la imagen portadora plasmando datos indispensables para los procesos de incrustación y desincrustación. A estas alteraciones consistentes se les llama la *huella del programa*. Entre más características cómodas para el usuario se incluyan en la aplicación esteganográfica, la huella plasmada será más grande y por consecuencia será más fácil de detectar. En este momento la huella de DigiMarker no sólo es sumamente pequeña (ordinariamente no rebasa el 0.006% del tamaño de la portadora),

sino que se encripta y luego se dispersa la huella mediante un algoritmo especial para hacerla aún más difícil de encontrar. Parte de esta seguridad se sacrificaría si se aumentara el tamaño de la huella. No obstante, es una mejora que merece evaluarse con cuidado e implementarla si así lo amerita.

2) El video digital es cada vez más popular. Un área digna de ser investigada es la esteganografía usando video digital como portadora. Existen básicamente dos formatos de video: el estándar NTSC[1] que se usa en América (29.9 fps[2] a 720 x 480 píxeles) y el formato PAL[3] que se usa en Asia y Europa (25 fps a 720 x 576 píxeles). En el formato PAL, que presenta menos cuadros por segundo, tendremos una exposición de 25 x 720 x 576 = 10.3 millones de píxeles por segundo. Suponiendo una incrustación de tamaño muy moderado para lograr un nivel altísimo de seguridad esteganográfica, podríamos pensar conservadoramente en transportar mensajes secretos a razón de 475kB por segundo. Esta tasa de transferencia es sumamente atractiva y merece que se le dedique investigación.

¿En qué escenario es factible este enfoque? Pues hay muchos pero imaginemos uno en particular: una cámara de video digital que entre sus microprocesadores incluya un chip esteganográfico. Se inserta en la cámara un cartucho de video portador con cualquier tipo de video previamente grabado. Posteriormente se realiza la filmación discreta en lugares de acceso restringido. La cámara, a través del chip esteganográfico, registrará las imágenes secretas montándolas o incrustándolas sobre el video portador. En caso de revisión del equipo, lo único que se verá será el video inocente, sin ningún rastro de haber registrado las imágenes restringidas.

3) Otra área interesante y de fácil acceso es modificar DigiMarker para que permita el uso de otros formatos convenientes, como puede ser el formato TIFF[4]. La mayoría de las cámaras fotográficas digitales profesionales proveen este formato para su opción de más calidad, sin compresión alguna o bien con compresión sin pérdida. Por lo tanto el formato TIFF es buen candidato para ser portador esteganográfico. Quizá resulte sorprendente para el lector ajeno a la fotografía profesional que cada una de estas fotografías digitales RGB puede medir unos 96 MB[5], proveyendo así un inmenso contenedor para mensajes secretos. Sería pues muy interesante y productivo ampliar la tecnología DigiMarker al formato TIFF o a otros formatos nativos de cámaras fotográficas digitales.

4) Aunque DigiMarker toma en cuenta algunas características de la visión humana, todavía hay mucho que investigar y desarrollar en esta dirección. Tomemos el siguiente caso:

El sistema visual humano es más sensible a cambios en la luminancia de un color. De los tres colores primarios, el verde tiene la contribución mayor para la luminancia. Basados en estos principios, Michiharu Niimi del Instituto de Tecnología de Kyushu y Richard Eason de la Universidad de Maine, propusieron un método esteganográfico usando la preservación lumínica en la cuantificación del color[6]. El método propuesto sólo funciona para imágenes de paleta de color. Sería interesante estudiar el comportamiento de la luminosidad y averiguar si es aplicable a la tecnología de DigiMarker (que no se basa en paletas de colores).

6.4 Pensamiento final

El presente trabajo utiliza técnicas de criptografía pero no es un sistema criptográfico; utiliza utilerías de compresión de datos, pero no es un sistema de compresión; utiliza programación en C, pero no es una tesis de programación. Es más, aunque el pensamiento se centra en la novedosa tecnología esteganográfica, aun así el autor se siente tentado a pensar que no es una tesis de esteganografía, sino un ensayo que reflexiona sobre lo que mexicanos más talentosos que él podrían hacer si su investigación se dirigiera al punto correcto.

---

[1] Del inglés *National Television System Committee.*

[2] Del inglés *frames per second.* Quiere decir cuántos cuadros por segundo se requieren para presentar la secuencia de video.

[3] Del inglés *Phase Alternate Line.*

[4] Del inglés *Tag Image File Format.*

[5] Por ejemplo, el modelo *ProBack Plus* de Kodak, que tiene un sensor de 16 mega píxeles y produce imágenes de 4,080 x 4,080 píxeles a 36 bits (12 bits por cada color primario). Se puede consultar más información en: http://www.kodak.com/global/en/professional/products/cameras/dcsProBack/proBackIndex.jhtml

[6] Michiharu, Niimi y Richard Eason. "A BPCS Based Steganographic Method for Palette-Based Images Using Luminance Quasi-Preserving Color Quantization", *Proceedings of Pacific Rim Workshop on Digital Steganography 2002*, Kitakyushu, Japón, p. 84.