UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD AZCAPOTZALCO

División de Ciencias Básicas e Ingeniería
Ingeniería en Computación

PROYECTO TECNOLÓGICO

Transmisión de archivos de texto cifrados usando esteganografía en imágenes GIF

Ángel Pérez García
204203880

Asesores:

Dr. Francisco Javier Zaragoza Martínez
No. Eco. 20197
Departamento de Sistemas

Dr. Marco Antonio Heredia Velasco
No. Eco. 37848
Departamento de Sistemas

Trimestre: 2014 Otoño
12 de Enero del 2015

Yo, Francisco Javier Zaragoza Martínez, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Dr. Francisco Javier Zaragoza Martínez
No. Eco. 20197

Yo, Marco Antonio Heredia Velasco, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Dr. Marco Antonio Heredia Velasco
No. Eco. 37848

Yo, Ángel Pérez García, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Ángel Pérez García
Matricula: 204203880

RESUMEN

El presente documento describe el desarrollo del proyecto “Transmisión de archivos de texto cifrados, usando esteganografía en imágenes GIF”. Éste proyecto tiene como objetivo principal la creación de un programa que sea capaz de recibir como entrada un archivo de texto y una imagen GIF, procesar el archivo de texto para cifrarlo y posteriormente ocultar el archivo cifrado dentro de la imagen GIF utilizando un método esteganográfico.

El algoritmo criptográfico utilizado para el cifrado del archivo de texto es Data Encryption Standard (DES) y el algoritmo esteganográfico para la ocultación el LSB, éstos algoritmos se describen en el capítulo 5 de éste documento, si en algún momento alguien más desea estudiarlos, conocerlos o utilizarlos. La implementación de los algoritmos se describe en el capítulo 6.

El lenguaje de programación en el que se escribió el código fuente para el programa es ANSI C, se utilizó una librería (GIFLIB) escrita en el mismo lenguaje para la manipulación de la imagen, y se utilizó un Sistema Operativo Linux, Todo el código fuente se encuentra en los apéndices de este documento.

El programa garantiza el proceso de criptográfico y esteganográfico siempre que se cumplan con las especificaciones que más adelante se detallan (capitulo 7 y 8) y además se describen en el manual de usuario.

El programa desarrollado está pensado para que el cualquier usuario con pocos conocimientos de informática pueda utilizarlo, basta con saber utilizar la línea de comandos, además se incluye un archivo ayuda que describe la forma de utilizarlo.

Contenido

CAPITULO 1

INTRODUCCION

Según el diccionario de la Real Academia, la palabra criptografía proviene de la unión de los términos griegos *"criptos"* (oculto, secreto) y *"grafos"* (escritura) y su definición es: "Arte de escribir con clave secreta o de un modo enigmático"[6].

El termino esteganografia proviene del griego "*steganos"* (oculto) y "*graphos"* (escritura), la esteganografía se puede definir como la ocultación de información en un canal encubierto con el propósito de prevenir la detección de un mensaje [7].

Para el ser humano siempre ha sido necesario ocultar información de cualquier índole, con las antiguas culturas se ha encontrado que han utilizado métodos criptográficos (escritura jeroglífica). De la misma forma en la antigua Grecia se ha encontrado evidencias del uso de la esteganografía.

Con la aparición de las computadoras tanto la criptografía como la esteganografía han tenido un avance espectacular, sobre todo en aquellas áreas donde se utiliza información confidencial; agencias militares y de inteligencia, información bancaria, bases de datos, etc. Aunque de estas dos ramas, la criptografía ha sido la más difundida en la mayoría de los casos.

La mayoría de los avances criptográficos y esteganográficos se intentan mantener en secreto, siendo mantenidos por aquellas agencias con alto poder. Sin embargo en los últimos años se ha logrado que estas dos áreas estén al alcance de todos, con las investigaciones llevadas a cabo en universidades de todo el mundo.

Para que la esteganografía sea de más utilidad se puede combinar con la criptografía o viceversa, de esta forma aunque se descubra el patrón esteganográfico el mensaje no podrá conocerse con facilidad. Como usuarios finales, la criptografía y la esteganografía las utilizamos en forma de programas computacionales.

En éste documento se describe una forma de cómo combinar la esteganografía y la criptografía a través de un programa informático, con la finalidad de mantener nuestra información segura.

CAPITULO 2

ANTECEDENTES.

En los últimos años el desarrollo de las comunicaciones electrónicas, unido al uso masivo y generalizado de las computadoras, hace posible la transmisión y almacenamiento de grandes flujos de información confidencial que es necesario proteger [2].

La esteganografía y la criptografía son dos grandes ramas que han tenido mucho éxito en con respecto a la seguridad informática, si se aplican éstas dos juntas obtenemos un valor más grande de seguridad.

Tomemos como ejemplo el "problema de los prisioneros"[1]. Alice y Bob son personas que se encuentran en prisión en celdas distintas, juntos están haciendo un plan para escapar de la prisión, el problema al que se enfrentan es el medio de comunicación ya que la única manera de comunicarse es a través del guardia, así que Alice y Bob necesitarán una manera de comunicarse sin levantar sospechas.

Lo anterior es un buen ejemplo en el cual la esteganografía puede ayudar para transmitir el mensaje, Bob podría dibujar una imagen de una vaca sobre pasto y decirle al guardia que le entregue la imagen a Alice, y por supuesto el guardia observará la imagen y supondrá que solo se trata de una imagen común sin saber que la imagen contiene el mensaje.

Podemos suponer que la vaca dibujada contiene manchas negras de esta manera las manchas son las que realmente contienen el mensaje, manchas grandes significa que se tienen que esperar, mientras que manchas pequeñas puede significar que es el momento de escapar.

Referencias internas:

Durante el trimestre 06-O se presentó una propuesta del proyecto terminal relacionado con la esteganografía. En dicha propuesta se menciona que el programa usará imágenes en mapa de bits sin codificación (bmp), así como archivos de texto para ocultar mensajes escritos.

Presentado por: Joel David Villegas Hernández, matrícula 204304945, asesor: Dr. Francisco Javier Zaragoza Martínez

Las principales diferencias con el proyecto presentado por Joel Villegas y el propuesto en éste documento son las siguientes:

- El proyecto de Joel como ya se mencionó, trabaja con archivos de imágenes sin codificación (bmp)[1], el proyecto propuesto trabajará con archivos de imágenes con codificación (gif)[2], lo que implica que el algoritmo para manipular tales archivos sea más complejo.
- El programa desarrollado por Joel sólo usa archivos de texto. El programa final propuesto en éste momento, utilizará archivos cifrados.

Referencias externas:

Debido al gran auge que está teniendo la seguridad informática existen varias herramientas desarrolladas basadas en esteganografía. A continuación se mencionan solo algunas de ellas:

- Sistema esteganográfico GEHEIM [3]. Se plantea un sistema robusto que enmarca medidas de seguridad para la transmisión de mensajes cifrados y aplicándoles una técnica esteganográfica.
- JPHIDE y JPSEEK [4], son herramientas basadas en Linux, que toman un archivo y lo ocultan dentro de una imagen jpg.
- WbStego[5], Programa que funciona bajo las plataformas de Windows y Linux, toma cualquier tipo de archivo y los puede ocultar en Mapas de bits, archivos de texto, archivos HTML y archivos PDF.

1. Bmp (mapa de bits). Formato de imagen estándar de Windows. En los mapa de bits existe una matriz de píxeles se le asigna una dirección asociada un código de color específico. Esto constituye una imagen. Poseen una compresión sin pérdida de calidad y suelen ocupar mucho espacio de almacenamiento.
2. Gif (GraphicsInterchangeFormat). Formato gráfico utilizado para imágenes, para la compresión usa el algoritmo de LZW (Lempel-Ziv-Welch), el cuál es un algoritmo sin pérdida, basado en la multiplicidad de aparición de secuencias de caracteres en la cadena que se debe codificar. Su principio consiste en sustituir patrones con un código de índices y construir progresivamente un diccionario. Además, funciona en bits y no en bytes, por lo tanto, no depende de la manera en que el procesador codifica información.

CAPITULO 3

JUSTIFICACIÓN.

La finalidad de la criptografía es múltiple: primeramente, mantener la confidencialidad del mensaje, así como garantizar la autenticidad tanto del criptograma como del par remitente/destinatario [2]. El principal objetivo de la esteganografía es ocultar información privada o sensible dentro de un portador que aparenta ser inocuo.

Si a los algoritmos de cifrado se le agregan algoritmos de esteganografía estamos obteniendo un nivel más alto de seguridad, con esto aseguramos que nuestros mensajes confidenciales tengan una mayor confidencialidad, por eso este tema es de actualidad.

En la elaboración de este proyecto se requirieron conocimientos acerca de algorítmica, del manejo de archivos y de conocimientos avanzados de programación, es por eso que este proyecto solo puede ser realizado por un Ingeniero en Computación.

Este proyecto puede ser continuado para que no solo maneje archivos de texto e imágenes GIF, sino se le puede agregar módulos para que funcione con video, audio o bien otros formatos de archivos de gráficos.

CAPITULO 4

OBJETIVOS

OBJETIVOS GENERALES.

Implementar un programa que permita ocultar archivos de texto dentro de una imagen GIF, aplicando un algoritmo criptográfico y un proceso esteganográfico.

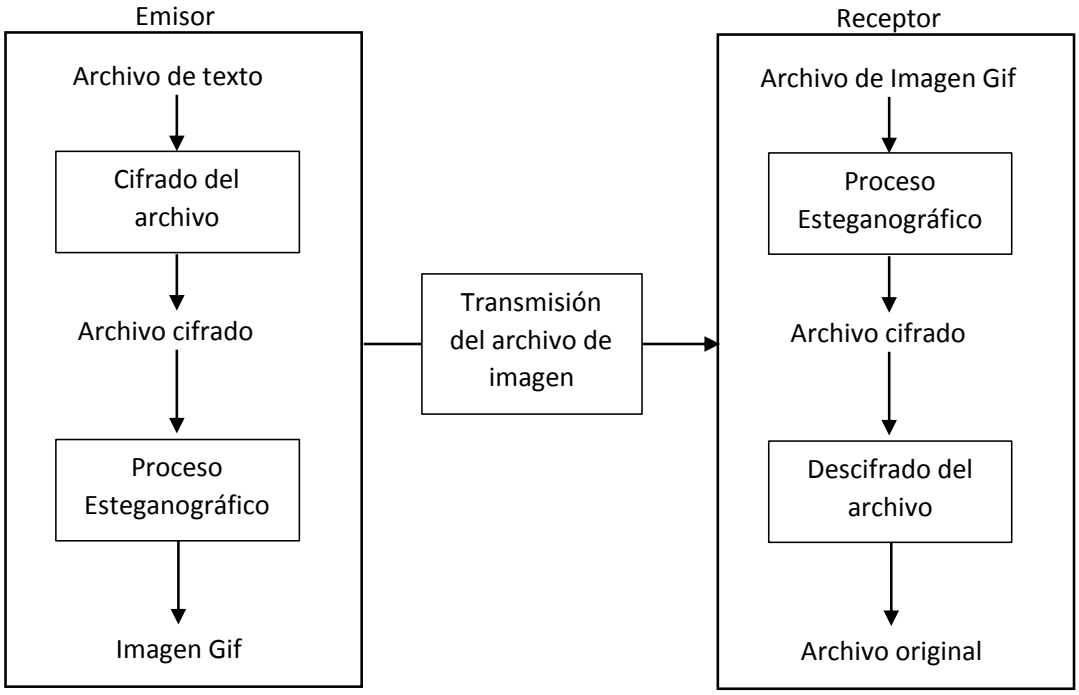
OBJETIVOS PARTICULARES.

- Seleccionar los algoritmos de cifrado y descifrado que serán aplicados a los archivos.
- Seleccionar los algoritmos esteganográficos para aplicarlos al archivo a transmitir.
- Implementar el algoritmo de cifrado seleccionado.
- Implementar el algoritmo esteganográfico seleccionado para enviar el archivo
- Implementar el algoritmo esteganográfico para obtener el archivo cifrado.
- Implementar el algoritmo de descifrado seleccionado.

CAPITULO 5

MARCO TEORICO.

A continuación se describe de manera general el funcionamiento del programa desarrollado.



CRIPTOGRAFIA

Para utilizar tanto métodos criptográficos como métodos esteganográficos, es necesario plantearse el uso de un *"algoritmo"*, el cual tiene definición como; *"Una secuencia finita y ordenada de instrucciones para llegar a la solución de un problema"*, actualmente las aplicaciones encargadas de la seguridad de la información emplean computadoras en la ejecución de los algoritmos. Existe un gran número de algoritmos encargados de cifrar y descifrar información algunos tan simples como la sustitución o transposición de la información y muchos otros de una complejidad mayor.

De la misma forma para utilizar un método criptográfico, la criptografía moderna utiliza claves que son utilizadas para obtener el mensaje cifrado y viceversa, tomando en cuenta esto, se pueden tener algoritmos simétricos que utilizas la misma clave para cifrar y para descifrar, o bien, algoritmos asimétricos donde se utilizan dos claves diferentes, una pública y otra privada. Cada uno de estos doshan sido implementados de diferentes maneras.

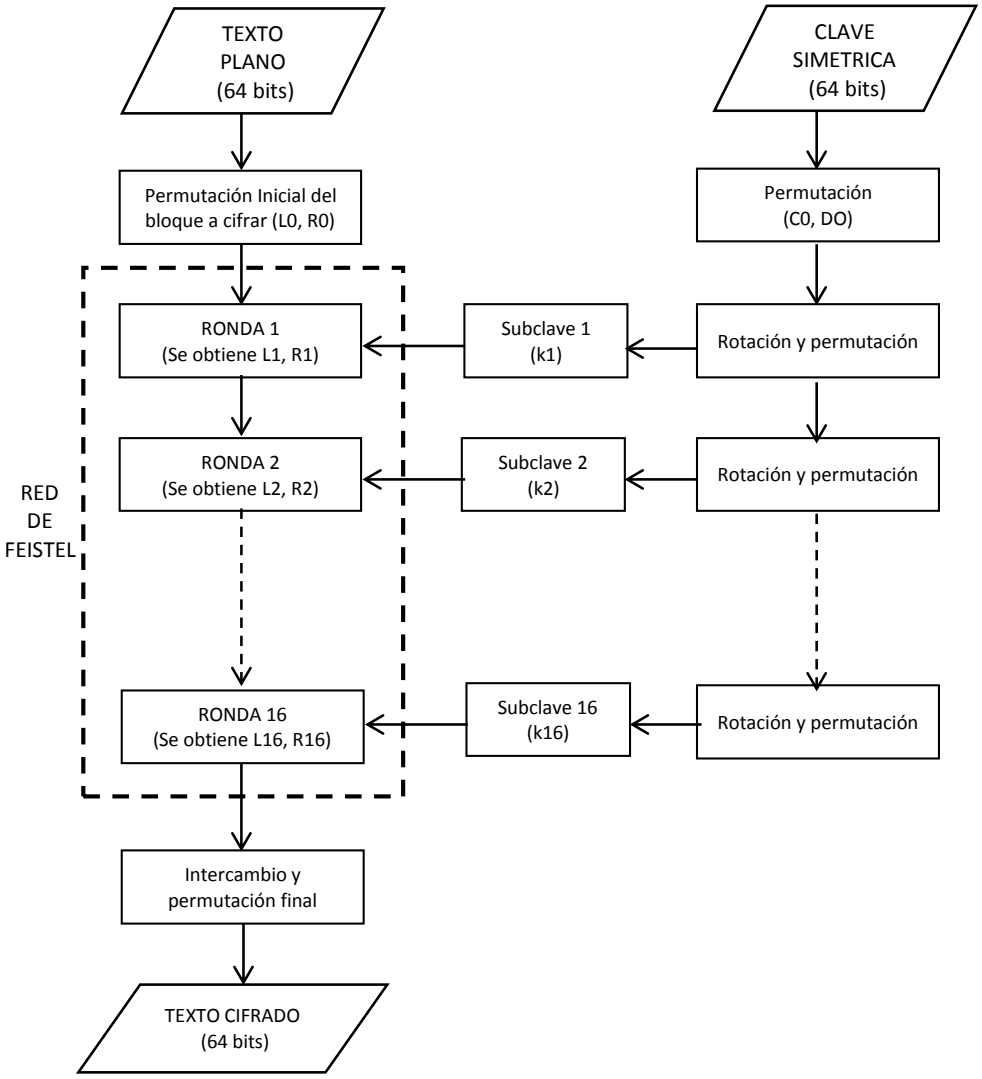
En el presente proyecto se ha implementado un algoritmo simétrico denominado DES (Data Encryption Standard)

DATA ENCRYPTION STANDARD

Es uno de los algoritmos simétricos más difundidos mundialmente. Se basa en un algoritmo desarrollado por IBM denominado *LUCIFER* a principios de la década de los setenta, y que fuera adoptado y modificado por el gobierno de los Estados Unidos utilizándolo en las comunicaciones. Actualmente el algoritmo tiene las siguientes especificaciones:

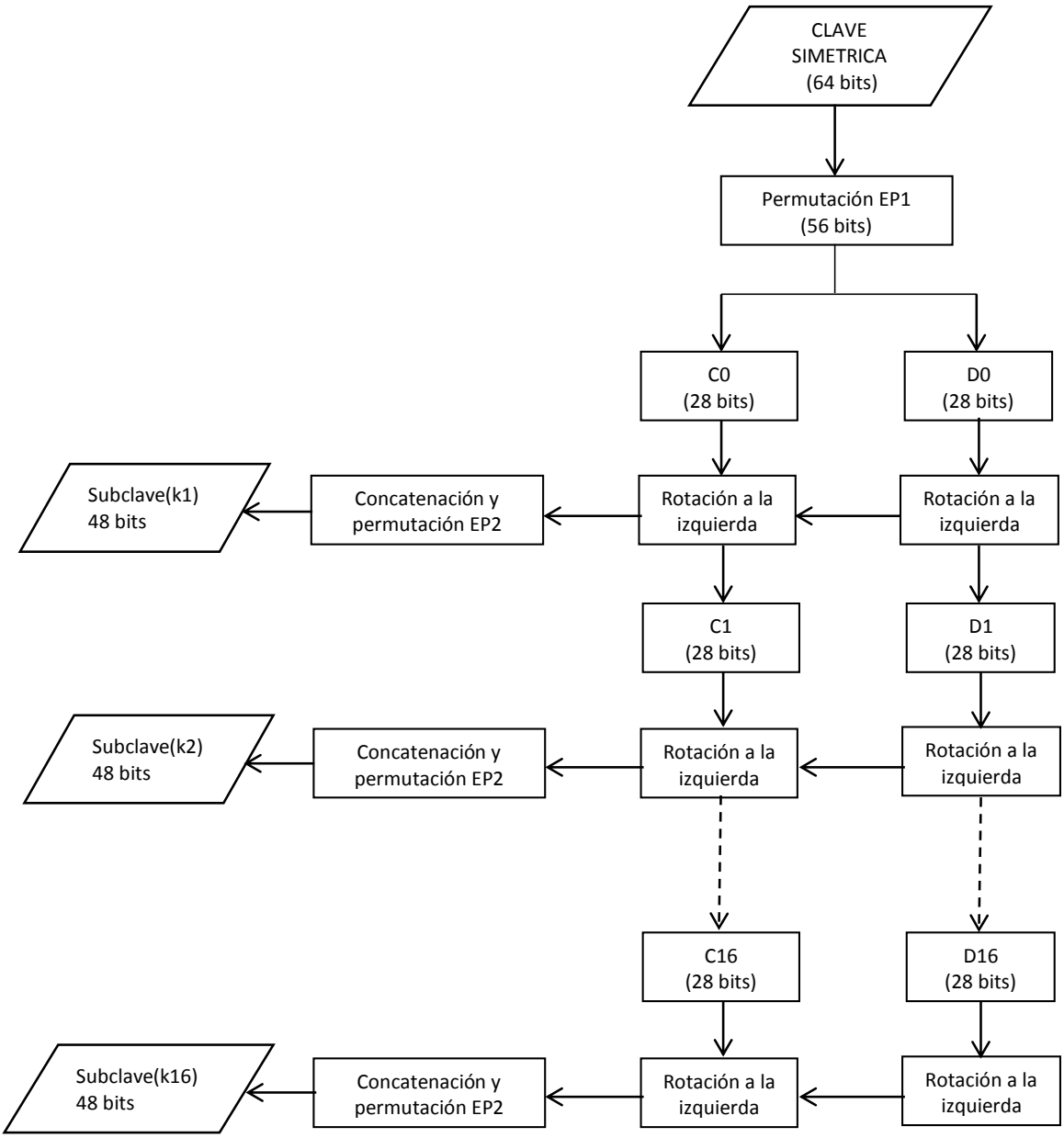
- Es un algoritmo de clave simétrica lo que significa que la clave utilizada para cifrar es la misma para descifrar, donde la clave tiene una longitud de 64 bits.
- Es un algoritmo que cifra en bloques cuya longitud es de 64 bits.
- El algoritmo utiliza: Permutaciones, concatenaciones, rotaciones, redes de Feistel y cajas de sustitución(s-boxes).
- Tiene diferentes etapas (19 en total), 16 etapas pertenecen a la red de Feistel.
- El manejo de todos estos conceptos es a nivel de bits lo que lo hace complejo de programar.

REPRESENTACION ESQUEMATICA DEL ALGORITMO DES[9].



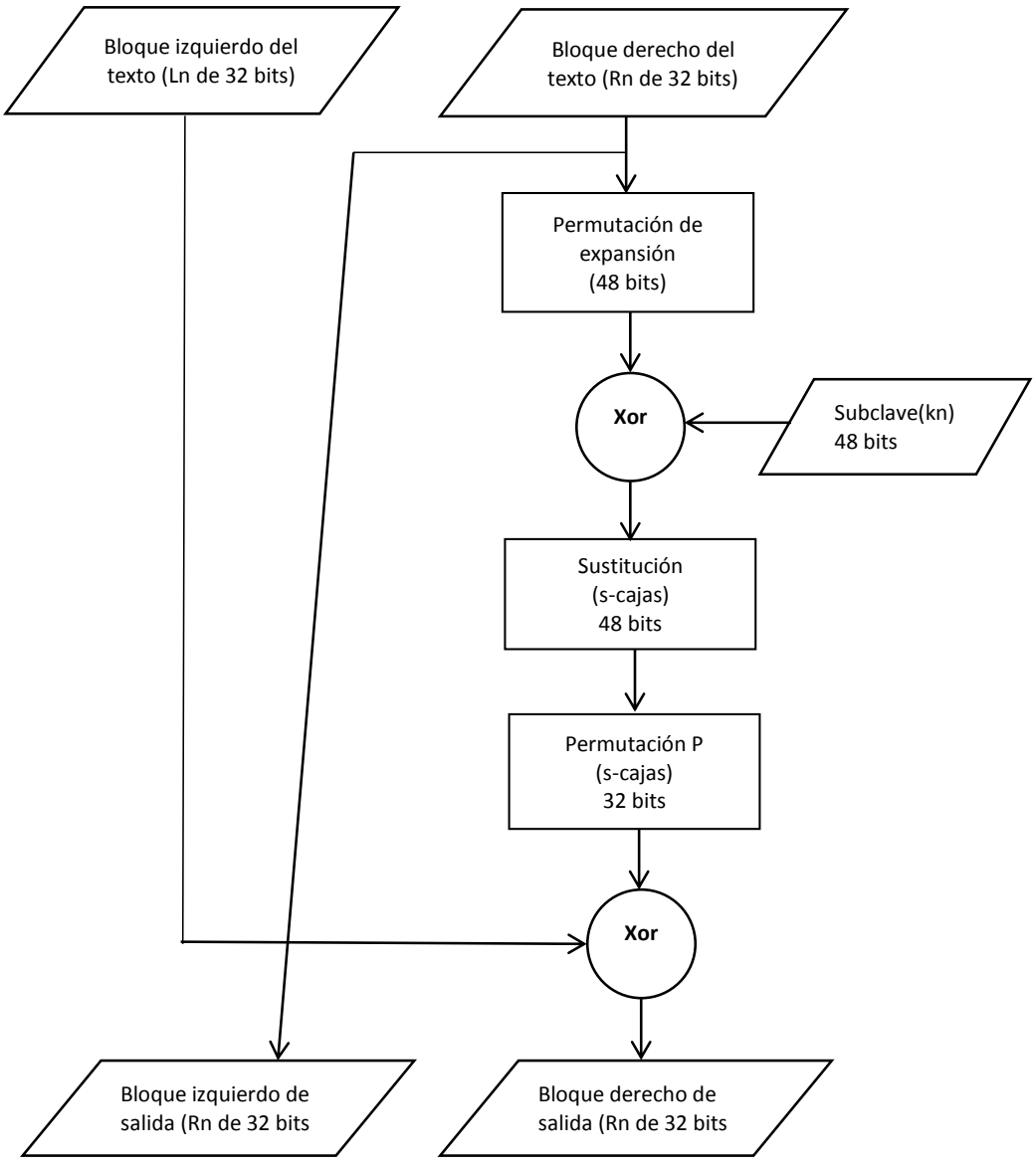
CLAVE SIMETRICA

La clave de cifrado es utilizada solamente con 54 bits de los 64 y además es dividida en 2 bloques iguales(C, D), con esta clave se generan 16 subclaves de 48 bits cada una las que realmente se utilizan para cifrar la informaciòn, a continuación se muestra el esquema del algoritmo para la el manejo de la clave.



RED DE FEISTEL

La red de Feistel del algoritmo DES consta de 16 rondas, utilizando como entrada los dos bloques de texto plano y las 16 subclaves generadas, se realizan 2 permutaciones, Cajas de sustitución(S-boxes) de 6 a 4 bits y operaciones XOR.



Para descifrar el texto se utilizan el mismo procedimiento con las mismas 16 subclaves solo que éstas deben ser en orden inverso.

PERMUTACIONES.

El algoritmo DES lleva a cabo diferentes permutaciones a nivel de bits, se toma en cuenta como el bit 1 el de la izquierda, es decir el más significativo y como bit 64 (sea el caso) el de la derecha, el menos significativo ($b_1, b_2, b_3, b_4,\ldots, b_{64}$), dichas permutaciones se describen a continuación:

Permutación inicial y permutación final del algoritmo las cuales se realizan sobre el texto.

Permutación Inicial P_i															
58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Permutación Final P_f															
40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

Para la red de Feistel se realizan las siguientes permutaciones:

La permutación E es una permutación de expansión (de 32 a 48 bits) y se realiza inicialmente, la permutación P es una permutación de reducción (de 48 a 32 bits) y se aplica al final de la ronda.

Permutación E															
32	1	2	3	4	5	4	5	6	7	8	9	8	9	10	11
12	13	12	13	14	15	16	17	16	17	18	19	20	21	20	21
22	23	24	25	24	25	26	27	28	29	28	29	30	31	32	1

Permutación P															
16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Para generar las claves se utilizan dos permutaciones. EP1 se utiliza al inicio del algoritmo, mientras que EP2, se utiliza en los 16 pasos para generar las subclaves.

Permutación EP1														
57	49	41	33	25	17	9	1	58	50	42	34	26	18	
10	2	59	51	43	35	27	19	11	3	60	52	44	36	
63	55	47	39	31	23	15	7	62	54	46	38	30	22	
14	6	61	53	45	37	29	21	13	5	28	20	12	4	

Permutación EP2														
14	17	11	24	1	5	3	28	15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2	41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32

CAJAS DE SUSTITUCION

En el caso de las cajas de sustitución, éstas se realizan dentro de la red de Feistel dividiendo el bloque de 48 bits en 8 pequeños bloques (S_n) de 6 bits cada uno (b_1, b_2, b_3, b_4, b_5, b_6) y obteniendo 8 bloques de 4 bits cada uno. El procedimiento que se debe seguir es el siguiente; se toman los bits 1 y 6 (b_1, b_6) y se forma un número que representa la fila de la caja, con los bits 2, 3, 4 y 5 (b_2, b_3, b_4, b_5) se forma otro número que representa la columna, la intersección entre la fila y la columna es el numero devuelto de 4 bits. Este procedimiento se realiza para los ocho bloques (S_1, S_2,…,S_8).

Fila	Columna 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	S-Caja
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S_1
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S_2
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S_3
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S_4
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S_5
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S_6
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S_7
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S_8
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

ROTACIONES

En la generación de las claves se realizan 16 rotaciones de los bits a la izquierda, en algunos casos son dos bits rotados o un solo bit, la siguiente tabla muestra la cantidad de bits rotados dependiendo de la ronda.

Ronda	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits despl.	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

ESTEGANOGRAFIA

La esteganografía se basa en insertar la información dentro de un archivo, los archivos más utilizados son de tipo multimedia, imágenes, audio o video, a estos archivos generalmente se le denomina contenedor. En un proceso esteganográfico se pueden considerar los siguientes elementos:

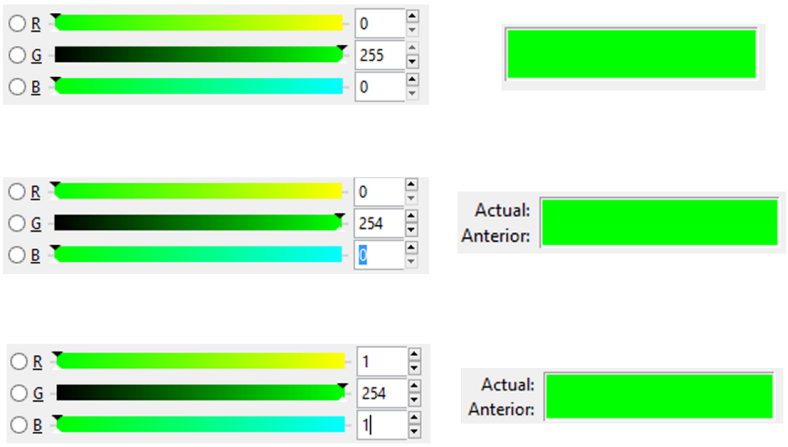
- *Objeto contenedor*: Se trata de aquel elemento que portará el mensaje oculto.
- *Estego-objeto*: Es aquel elemento que porta el mensaje oculto.
- *Adversario*: Son aquellos elementos a los cuales se les quiere ocultar la información.

Considerando estos tres elementos un proceso de ocultación efectivo, es aquel donde el adversario no logra conocer el mensaje oculto. Con la difusión de la información digital, los objetos contenedores utilizados para procesos esteganográficos son imágenes digitales, audio, video, texto plano e inclusive protocolos de comunicación. Existe una gran variedad de implementaciones donde se han utilizado éste tipo de archivos, por ejemplo, en años anteriores se encontró que se transmitía código malicioso utilizando imágenes digitales que eran enviadas a través de correo electrónico.

Sustitución de bits del objeto contenedor

Está técnica implica cambiar ciertos bits del objeto contenedor por los de la información que se desea ocultar, debido al exceso de detalle en los archivos digitales (por ejemplo las imágenes) y modificando los bits correctos del estego-objeto, éste aparenta no haber sido modificado. Por ejemplo si el objeto contenedor fuera un archivo de sonido, se pudieran modificar los bits que generen un sonido no audible para el ser humano, de esta forma esos cambios serian inapreciables para el ser humano.

En el caso de las imágenes digitales, el método más utilizado consiste en cambiar los bits menos significativos que conforman un pixel (*método LSB*), modificar solo el bit menos significativo en un pixel formado por 8 bits, solo cambia en 1%, lo que representa un cambio inapreciable para el ojo humano, considerando que hay imágenes que ocupan 3 bytes para un pixel genera más espacio para poder ocultar información. Cada bit modificado del pixel, será un bit de la información a ocultar, si lo que se quiere ocultar son caracteres, entonces se necesitarán 8 pixeles (considerando que un pixel está formado por 8 bits) para cada un solo caracter.



Modificación de los bits menos significativos en un pixel de tres bytes. Se puede modificar un bit de un componente (imagen central), o bien un bit de los tres componentes (imagen inferior) y aun así no apreciar el cambio.

Para el desarrollo de este proyecto, se utilizaron imágenes en formato GIF como objetos contenedores, lo que fue necesario entender la estructura de una imagen GIF.

IMAGEN GIF (GRAPHIC INTERCHANGE FORMAT)

Es uno de los formatos de imágenes en mapa de bits más utilizados, en los inicios de Internet el formato GIF fue de los más difundidos debido a que ofrece una de las mejores compresiones, generando así archivos de pequeño tamaño. Actualmente existen dos versiones del formato GIF; GIF87 y GIF89.

Dentro de las características de la versión GIF87 destacan las siguientes:

- Las imágenes se comprimen utilizando el algoritmo LZW (Lempel-Ziv-Weich), el cual funciona muy bien cuando se tienen imágenes con poco degradado.
- Capacidad de contener varias imágenes dentro de un único fichero.
- Posicionamiento de las imágenes dentro de un área lógica de la pantalla.

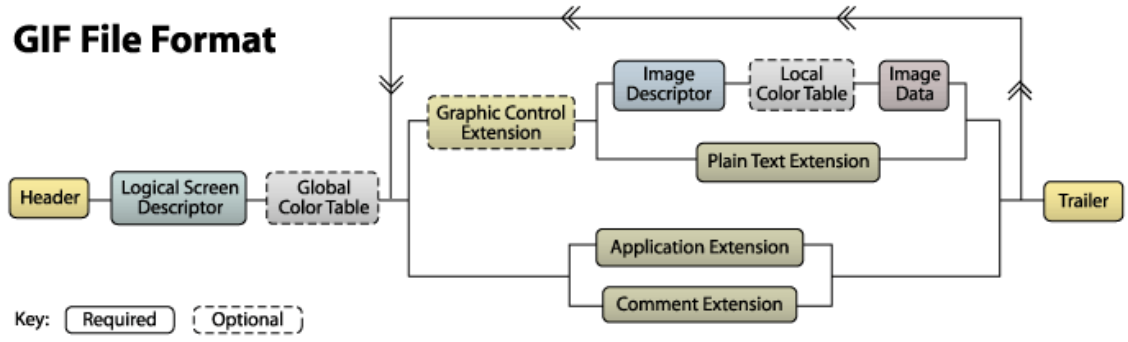
Para la versión GIF89 se añadió:

- Tiempo de espera para ir mostrando las imágenes (centésimas de segundo).
- Capacidad de especificar color transparente.
- Incluir comentarios, los cuales no se muestran.
- Capacidad de mostrar líneas de texto.
- Extensiones de aplicaciones específicas codificadas dentro del fichero.

En ambas versiones el formato GIF puede contener como máximo 256 tonos de colores, utilizando una paleta de colores que contiene desde 1 color hasta 256. En una imagen GIF los colores que hay en la paleta de colores son los únicos colores disponibles para la imagen. Como el formato GIF admite la inserción de múltiples imágenes, si es el caso cada imagen GIF dentro del formato puede tener su propia paleta de colores.

Un archivo GIF está compuesto de una secuencia de bloques de datos algunos son de longitud fija y otros de longitud variable. A continuación se muestra el diagrama de los bloques que conforman a una imagen GIF.

DESCRIPCIÓN GRÁFICA DEL FORMATO GIF[8]



BLOQUES:
Cabecera: Tiene una longitud de 6 bytes y contiene los códigos de caracteres ASCII GIF87a o GIF89a según sea la versión del archivo.

Descriptor de pantalla lógica: Tiene una longitud de 7 bytes, le indica al decodificador cuanto espacio ocupa la imagen, contiene la siguiente información:

- El ancho y el alto del *canvas* de la imagen.
- La resolución de los colores
- Un índice del color de fondo.
- El tamaño de la paleta global de colores
- La proporción de los pixeles.

Paleta Global de Colores: Consiste en una lista de colores RGB(Rojo, Verde y Azul) que forman los colores de la imagen, el decodificador los debe agrupar en tercias ya que 1 byte representa el rojo, otro byte el verde y el tercer byte el azul. Cada byte tiene un valor de 0 hasta 255 que representa la intensidad de cada color, el tamaño es variable y depende de la cantidad de colores a representar, aunque recordemos que el formato GIF solo admite hasta 256 colores diferentes.

Extensión de Control Gráficos: Se utilizan para especificar las configuraciones de las transparencias y el control de las animaciones.

Descriptor de Imagen: Contiene información que corresponde a una imagen del GIF, es necesario recodar que cada archivo GIF puede contener una o más imágenes, cada imagen tiene su propio descriptor de imagen que proporciona la siguiente información:

La posición (izquierda y arriba) en donde comienza la imagen dentro del *canvas*.

- La anchura y la altura de la imagen.
- Una bandera de entrelazado.
- Bandera de la paleta local de colores.
- El tamaño de la tabla de colores local.

Paleta Local de Colores: Tiene la misma organización que una tabla de color global y solo la utiliza el bloque de datos de imagen que le sigue, sino se especifica ninguna tabla de colores local, el decodificador debe de usar la información de la tabla de color global.
Datos de imágenes: Corresponden a los datos reales de la imagen, se leen de byte en byte para cada pixel.

Extensión de texto plano: Además de imágenes se pueden agregar texto en la pantalla, aunque la mayoría ignora éste bloque.

Extensión de aplicación: Permite que algunos programas incrusten información propia sobre éste bloque para algún propósito.

Extensión de comentario: Permite incluir texto ASCII, funciona como comentario y no se muestra en pantalla.

Trailer: Indica el final del fichero GIF, siempre es un byte de longitud.

CAPITULO 6

DESARROLLO DEL PROYECTO.

El código para el programa está escrito en lenguaje C, los archivos de texto pueden ser creados con cualquier editor de texto sin formato, se sugiere verificar que tanto el archivo de texto como la imagen GIF cumplan con las especificaciones correctas, las cuales están descritas en el manual de usuario o en el archivo de ayuda.

IMPLEMENTACION DEL ALGORITMO DES.

Se implementó una librería que contiene todas la funciones encargadas de realizar todos los pasos que conforman el algoritmos DES. A continuación se describen las operaciones más importantes. Puede consultar en el capítulo de Apéndices, el código completo del algoritmo.

Funciones que se utilizan en el algoritmo DES definidas en la cabecera cifrado.h:

```
/********FUNCION QUE SE ENCARGA DE REALIZAR LAS PERMUTACIONES*************/
unsigned long long permutacion ( unsigned char tabla[ ], int tam ,unsigned long
long bloque);

/*****FUNCION QUE GENERA LAS 16 SUBCLAVES ************/
void genera_claves( unsigned long long clave);

/****FUNCION QUE INVIERTE LAS CLAVES PARA DESCIFRAR********/
void invertir_claves( );

/****  FUNCION QUE SE ENCARGA DE CIFRAR O DESCIFRAR UN BLOQUE DE 64 BITS *****/
unsigned long long procesa_bloque( unsigned long long texto);

/*****FUNCION QUE SE ENCARGA DE LEER LOS DATOS DE UN ARCHIVO Y PROCESARLOS****/
int procesa_fichero(char *entrada, char *salida, int bandera);
```

Lo primero que observamos del algoritmo DES es que trabaja con bloques de 64 bits, a veces menos, para utilizar un bloque de 64 bits en lenguaje C se utilizó el tipo long long, esto significa que si se desea utilizar una variable de 64 bits, se declara de la siguiente manera:

```
unsigned long long x;
```

Se coloca unsigned para utilizar los 64 bits y no dejar uno para el signo.

El algoritmo DES funciona haciendo operaciones a nivel de bits y considera como bit 1 el de la izquierda es decir el más significativo y el ultimo bit el de la derecha el menos significativo ($b_1, b_2, b_3, \ldots, b_n$). Lenguaje C nos proporciona una serie de operandos que se encargan de realizar operaciones lógicas con los bits, como son AND, OR, NOT, XOR (& , | , ~, ^, respectivamente). Si se desea obtener un bit de cualquier número, se utilizan las operaciones lógicas para obtener dichos bits, por ejemplo: si se tienen los siguientes bits (1111) si desea obtener el bit 4, entonces se realiza la siguiente operación:

```
(1111) &(0001 ) = 0001
```

Escrito en C, la operación quedaría de la siguiente manera:

```
x = 0xf  &  0x1
```

Se utilizan números en hexadecimal, que son los equivalentes en binario, la variable x es la que almacenaría el bit deseado.

Como el algoritmo DES realiza varias permutaciones se implementó una función que se encargará de llevar a cabo esas permutaciones, la cual recibe como parámetros el bloque de bits a ser permutado, el tamaño de la permutación y la tabla de permutación, se obtiene como resultado el bloque permutado.

```
unsigned long long permutacion ( unsigned char tabla[ ], int tam
,unsigned long long bloque){
	int i;
	unsigned long long mask, bloque_perm;
	int ptr;
	unsigned long long bits[64]; //se utiliza para obtener el bit a ser
permutado

	bits[0]=0x01;
	for(i=1; i < 64 ; i++){
		bits[i] = (bits[i-1]) * 2;
	}
	bloque_perm = 0x0;
	for(i=0 ; i<tam ; i++){
		mask =0x0;
		ptr = tabla[i]-1;
		mask = bloque & bits[63-ptr];
		mask = mask<<(ptr);
		mask = mask>>i;
		bloque_perm = bloque_perm | mask;
	}
	return bloque_perm;
}
```

Las tablas de permutación se declararon como vectores que contienen los bits que se tienen que mover, por ejemplo la permutación inicial del algoritmo se declara de la siguiente forma:

```
unsigned char IP[ ] = {
	58, 50, 42, 34, 26, 18, 10, 2,
	60, 52, 44, 36, 28, 20, 12, 4,
	62, 54, 46, 38, 30, 22, 14, 6,
	64, 56, 48, 40, 32, 24, 16, 8,
	57, 49, 41, 33, 25, 17,  9, 1,
	59, 51, 43, 35, 27, 19, 11, 3,
	61, 53, 45, 37, 29, 21, 13, 5,
	63, 55, 47, 39, 31, 23, 15, 7
};
```

Una de las etapas del algoritmo DES es la aplicación de cajas de sustitución, esto significa recibir una cantidad de bits y regresar otra cantidad basándose en una tabla ya descrita anteriormente, para utilizar esta tabla, se declaro un vector de la siguiente forma:

```
unsigned char s_boxes[8][4][16];
```

El primer índice hace referencia a las 8 S-Cajas, el segundo a las filas de cada S-caja y el tercero a las columnas.

El siguiente código muestra cómo se lleva a cabo esa sustitución:

```
unsigned char B[8], S[8];
int fila, columna;
        .
        .
        .
for(i=0 ; i<8 ; i++){
         fila=columna=temp = 0;
        temp = B[i] & 0x01;
        fila = fila | temp;
        temp = B[i] & 0x20;
        temp = temp>>4;
        fila = fila | temp;
        temp = (B[i] & 0x1e)>>1;
        columna = columna | temp;
        S[i] = s_boxes[i][fila][columna];
}
```

El algoritmo DES debe generar 16 subclaves a partir de una clave publica la cual también tiene una longitud de 64 bits, en uno de los pasos para generar las claves se tienen que llevar a cabo rotaciónes de los bits a la izquierda, para lo cual se utilizó la siguiente operación:

```
x >> y
```

Que se encarga de mover los bits del número ***x, y*** lugares a la izquierda. Dependiendo de la ronda en la que se encuentre el algoritmo, estas rotaciones pueden ser una o dos, por lo tanto la cantidad de rotaciones se almaceno en otra tabla:

```
int rot[ ] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};
```

Por último, el código calcula cuantas veces leerá del archivo de texto bloques de 8 bytes (64 bits) y si el último bloque no es de 8 bytes éste es completado con ceros, cada vez que se realiza una lectura de un bloque, es procesado y se regresa el bloque cifrado para ser enviado al archivo de salida se repite el proceso hasta haber cifrado todos los bloques.

Antes de proceder con el proceso de cifrado se verifica que el archivo de texto a cifrar exista, en caso contrario se finaliza el programa.

IMPLEMENTACION DEL PROCESO ESTEGANOGRÁFICO

Para la implementación del proceso esteganográfico se utilizaron imágenes GIF utilizando el método LSB, que implica introducir un bit de la información a ocultar dentro del bit menos significativo de la imagen.

Lo primero que se consideró es leer la información de la imagen GIF, como por ejemplo; la anchura, la altura, la paleta de colores y los datos de la imagen. Para poder obtener toda la información que se necesita del archivo GIF, se utilizó una librería denominada GIFLIB desarrollada por Gershon Elber, Eric S. Raymond, Toshio Kuratomi[8], escrita en lenguaje C y donde se nos proporciona el código fuente para la manipulación de las imágenes GIF. El código fuente se encuentra en el sitio SourceForge para su descarga, aunque dentro del código fuente de este proyecto ya se incluyen las librerías necesarias para que funcione.

LECTURA DE LA IMAGEN GIF.

Para leer o escribir una imagen GIF, se utiliza un descriptor de archivo GIF definido en la librería GIFLIB, el cual es un puntero a la siguiente estructura:

```
typedef struct GifFileType {
    GifWord SWidth, SHeight;
    GifWord SColorResolution;
    GifWord SBackGroundColor;
    GifByteType AspectByte;
    ColorMapObject *SColorMap;
    int ImageCount;
    GifImageDesc Image;
    SavedImage *SavedImages;
    int ExtensionBlockCount;
    ExtensionBlock *ExtensionBlocks;
    int Error;
    void *UserData;
    void *Private;
} GifFileType;
```

Los miembros de la estructura que se utilizaron en el desarrollo del proyecto son:

- **GifWord SWidth, SHeight:** son enteros que representan el ancho y el alto de todo el canvas del fichero GIF.
- **Int ImageCount:** Contiene el número de imágenes que tiene el fichero GIF.
- **ColorMapObject *ScolorMap:** Es una estructura que contiene información de la paleta de colores global, si es que existe. Si el fichero GIF no tiene paleta de colores global entonces el apuntador es NULL.

- **SavedImage *SavedImages:** Es una estructura que contiene la información de las imágenes.

La estructura ColorMapObject esta definida de la siguiente forma:

```
typedef struct ColorMapObject {
    int ColorCount;
    int BitsPerPixel;
    bool SortFlag;
    GifColorType *Colors;
} ColorMapObject;
```

Los miembros utilizados son los siguientes:

- **int ColorCount:** guarda el total de colores de la paleta de colores.
- **int BitsPerPixel:** Contiene la cantidad de bits utilizados para cada pixel.
- **GifColorType *Colors:** Es una estructura que guarda los colores que conforman la paleta de colores, está formada por tres miembros Red, Green, Blue, cada miembro es una variable de 1 byte.

La estructura SavedImage está definida de la siguiente forma:

```
typedef struct SavedImage {
    GifImageDesc ImageDesc;
    GifByteType *RasterBits;
    int ExtensionBlockCount;
    ExtensionBlock *ExtensionBlocks;
} SavedImage;
```

Los miembros de ésta estructura que se utilizaron son:

- **GifImageDesc ImageDesc:** Es una estructura que contiene la descripción de cada imagen GIF, como las dimensiones, su posición e información de su propia paleta de colores (Paleta de Colores Local)
- **GifByteType *RasterBits:** Son los bytes que componen a los datos de la imagen.

FUNCIONES:

La librería define las funciones de apertura, lectura y escritura de una imagen GIF.

Funciones para decodificar (lectura – dgif_lib.c):

```
GifFileType *DGifOpenFileName(const char *GifFileName, int *Error)
```

Abre un archivo GIF con el nombre GifFileName y regresa el puntero GifFileType con todos sus miembros, si ocurre un error se regresa NULL y la variable Error contiene el código de error.

```
Int DgifSlurp(GifFileType)
```

Lee la información de cada imagen que conforma el fichero GIF, si se produce algún error se regresa GIF_ERROR y el miembro Error de la estructura GifFileType contiene el código de error, sino se regresa GIF_OK.

```
int DGIfCloseFile(GifFileType *GifFile, int *ErrorCode )
```

Escribe el bloque de terminación del fichero GIF, cierra el fichero GIF y libera la memoria ocupada.

Funciones para codificar (escritura - egif_lib.c):

```
GifFileType *EGifOpenFileName(const char *GifFileName,
                              const bool GifTestExistence, int *Error)
```

Crea un nuevo archivo GIF con el nombre GifFileName, si el segundo parámetro (GifTestExistence) es TRUE (1) y el archivo existe, éste no es destruido y se regresa NULL, si es FALSE (0) se reescribe el archivo.

```
int EGifSpew(GifFileType *GifFile)
```

Escribe toda la información del fichero GIF que apunta la estructura GifFileType, la función escribe el bloque de terminación del fichero y libera memoria.

Funcion para mostrar errores (gif_err.c)

```
const char *GifErrorString (int ErrCode)
```

Regresa la cadena que muestra información del error producido y especificado en ErrCode, regresa NULL si no se conoce el error.

GIFLIB posee más funciones, que para el desarrollo de este proyecto no son necesarias estudiarlas a fondo, si se desea conocer más a fondo la librería GIFLIB puede consultar el sitio web del desarrollador y descargar la librería completa.

Se han elaborado una serie de funciones que se encargan de colocar la información del archivo cifrado y obtener de la imagen GIF el archivo cifrado, están definidas en el archivo cabera estego_lib.h:

Función que coloca un bit en un caracter, en el lugar indicado por índice:
```
unsigned char colocaI(unsigned char texto, unsigned short int indice, unsigned short int bit);
```

Función que extrae un bit de un caracter:
```
unsigned short int extrae(unsigned char texto_c);
```

Función que coloca un caracter en un arreglo de 8 pixeles utilizando el bit menos significativo de cada pixel:
```
void coloca_caracter(unsigned char pixeles[], unsigned char  caracter);
```

Función que obtiene un caracter de un conjunto de ocho pixeles:
```
unsigned char obtener_caracter(unsigned char pixeles[]);
```

Función que coloca el archivo de texto dentro de un GIF:
```
int coloca_archivo( char nombre_archivo[], char nombre_imagen[], char imagen_copia[] );
```

Función que obtiene un archivo de texto de una imagen GIF:
```
int obtener_archivo( char nombre_archivo[ ], char nombre_imagen[ ]);
```

Función que verifica que el tamaño del archivo de texto alcance dentro de la imagen GIF:
```
int verifica_archivos(GifFileType *Gif, FILE *archivo);
```

Función que verifica que la imagen Gif tenga las especificaciones correctas:
```
int verifica_gif(GifFileType *Gif);
```

OCULTACION DEL ARCHIVO CIFRADO EN LA IMAGEN GIF:

Antes de realizar el proceso de ocultación se verifican los siguientes aspectos:

- Que el archivo de texto alcance dentro de la imagen GIF
- Que la imagen GIF contenedora exista.
- Que la imagen GIF no supere las dimensiones de 1024 x 768 pixeles.
- Que la imagen GIF sea en escala de grises y 256 tonos de grises.
- Que sea un fichero GIF con una sola imagen.
- Que el fichero GIF de salida (objeto-estego) no exista, en caso de existir se confirma eliminación.

Una vez que se verifican las características se procede con el proceso de ocultación, el miembro GifByteType llamado RasterBits de la estructura SavedImage es el que contiene los bytes de los datos de imagen por lo tanto es ahí donde se agregarán los bytes del archivo cifrado. Se realizan los siguientes pasos:

1. Lee archivo cifrado
2. Verifica archivo cifrado
3. Lee imagen original
4. Verifica imagen original

5. Verifica imagen de salida
6. Determina tamaño de archivo cifrado
7. Repite 4 veces:
    Lee 8 pixeles de la imagen
    Enviar byte del tamaño de la imagen
7. Repite n veces hasta fin de archivo
    Lee byte del archivo cifrado
    Lee 8 pixeles de la imagen
    Coloca byte en pixeles.
    Coloca pixeles modificados en la imagen
8. Copia datos de la imagen original a la imagen copia.
9. Cierra archivos.
10. Fin

Para ingresar un bit en un byte se utiliza la función colocaI( ), que recibe el byte a cambiar, el bit que se desea ingresar y el lugar donde se desea colocar (de acuerdo al método LSB es en el bit menos significativo, es decir, el de la derecha), para guardar el bit se utilizan operaciones lógicas ya que en C son las que funcionan a nivel de bits.

```
unsigned char colocaI(unsigned char texto, unsigned short int indice,
unsigned short int bit){
	if(indice >7 || bit > 1){
		printf("error\n");
		return;
	}
	if(bit==1)
		texto |= (1<<indice);
	else
		texto &= ~(1<<indice);
	return texto;
}
```

Se tiene que ingresar el tamaño del archivo a la imagen de salida ya que se utiliza posteriormente en el proceso de obtención del archivo cifrado, para esto, se reservan los primeros 32 pixeles de la imagen.
Considerando que se permiten imágenes de hasta 1024 x 768 pixeles, en total se podrían almacenar 98304 bytes de información, de los cuales 4 bytes corresponden al tamaño del archivo, lo que resulta es que se puedan almacenar realmente hasta 98300 bytes de información, que corresponde a un número mayor del tipo int.
Para almacenar los bytes del archivo se leen grupos de ocho pixeles como se muestra en el siguiente código

Para leer un byte del archivo cifrado se utiliza el siguiente código:

```
fread(&byte, sizeof(unsigned char),1,archivo)
```

Posteriormente se leen 8 pixeles de la imagen y son almacenados en un arreglo de 8 caracteres sin signo:

```
for(j=0; j<8 ; j++){
	pixeles[j]= Imagen_Gif->RasterBits[ptr_imagen];
	ptr_imagen+=1;
}
```

Una vez que se tienen los ocho pixeles se procede a colocar el byte leído en los pixeles con la función coloca_caracter:

```
coloca_caracter(pixeles, byte);
```

Los nuevos pixeles se almacenan en otra variable, la cual se copia en el arreglo original que contiene los pixeles:

```
for (j = 0; j < 8; j++){
	Imagen_Gif->RasterBits[ptr_temp] = arreglo_pixeles[j];
	ptr_temp+=1;
}
```

Este proceso se repite hasta haber leído todos los bytes del archivo cifrado.

OBTENCION DEL ARCHIVO CIFRADO DE LA IMAGEN GIF.

Se siguen las siguientes instrucciones para la obtención del archivo cifrado.

1. Lee imagen (estego-objeto)
2 .Verifica imagen
3. Verifica y crea archivo de salida.
4. obtener el tamaño del archivo.
5. Repite n veces hasta tamaño de archivo
	Lee 8 pixeles de la imagen
	Obtener byte oculto
	Coloca byte en el archivo de salida.
6. Finaliza y cierra archivos.
7. Fin.

De la imagen que contiene el archivo oculto lo primero que se lee es el tamaño del archivo de esta manera sabremos cuantos pixeles se tendrán que leer.

```
tam_archivo = 0;
for(x=0 ; x<4 ; x++){
	for(j=0; j<8 ; j++){
		pixeles[j]= Imagen_Gif->RasterBits[ptr_imagen];
```

```
                ptr_imagen+=1;
        }
        byte = obtener_caracter(pixeles);
        tam_archivo = tam_archivo | (byte<<(x*8));

}
```

Una vez que se tiene el tamaño del archivo ya se saben cuántos pixeles leer, para obtener el byte se utiliza la funcion obtener_caracter( ), el byte obtenido es enviado al archivo de salida.

```
for(i=0 ; i<tam_archivo ; i++){
        byte = 0;
        for(j=0; j<8 ; j++){
                pixeles[j]= Imagen_Gif->RasterBits[ptr_imagen];
                ptr_imagen+=1;
        }
        byte = obtener_caracter(pixeles);
        fwrite(&byte,sizeof(unsigned char), 1, archivo );
}
```

El archivo main.c es el que se encarga de llamar las funciones correctas.

CAPITULO 7

RESULTADOS.

Anteriormente ya se mostró que modificar un bit de un pixel formado por tres bytes no afecta visualmente en nada al pixel original. En éste capítulo observaremos como son afectados los pixeles en diferentes situaciones.

El método LSB señala que se debe modificar el bit menos significativo de los pixeles, pero ¿Qué sucedería si en vez de modificar el bit menos significativo modificamos el más significativo?

Para obtener este resultado se modificó el código del programa en la función coloca_caracter ( ), para que el bit a colocar sea el más significativo

```
void coloca_caracter(unsigned char pixeles[], unsigned char  caracter){
	int i;
	unsigned short int bit =0x00;
	for(i=0; i<8; i++){
		bit = caracter & (1<<i);
		bit = bit>>i;
		//printf("bit a colocar: %d\n", bit );
		arreglo_pixeles[i]= colocaI(pixeles[i], 7,  bit); //Se modifico
                                                                  para ocultar el bit en
                                                                  el más significativo
	}
}
```

Imagen 7-1. Imagen original

Imagen 7-2. Imagen donde se muestra el cambio al modificar el bit más significativo

Claramente logramos observar que una cantidad de pixeles han sido modificados por lo que sería demasiado sospechoso.

Si ahora modificamos un bit que se encuentre a la mitad del byte, por ejemplo el bit 4, se obtiene lo siguiente:

Imagen 7-3. Imagen en la cual se ha modificado el bit 4.

En este caso seguimos obteniendo una variación notable de los pixeles, que aunque es menor sigue siendo notorio para el ojo humano.

Por ultimo modificando el bit menos significativo obtenemos la siguiente imagen.

Imagen 7-4. Imagen en la cual se ha modificado el bit menos significativo

Logramos observar que los cambios son imperceptibles para el ojo humano, por eso es la forma más correcta de insertar la información a ocultar.

El código está escrito de tal forma que funcione en imágenes en escala de grises, ¿Qué pasa si queremos ocultar el archivo en una imagen que no es en escala de grises?

Imagen 7-5. Imagen que no está en escala de grises.

Para esto se ha modificado el código de tal forma que no se finalice si detecta que no es imagen en escala de grises y con paleta de 256 colores, es decir ya no se verifican estas opciones.

```
/*if(tam_paleta == 256){
	for(i=0 ; i<tam_paleta ; i++){
		rojo = Gif->SColorMap->Colors[i].Red;
		verde = Gif->SColorMap->Colors[i].Green;
		azul = Gif->SColorMap->Colors[i].Blue;
		if( (rojo ==verde) && (verde==azul) ){
		}
		else{
		fprintf(stderr, "La imagen no esta en escala de grises\n");
		return 0;
		}
	}
}
else{
	fprintf(stderr, "La imagen no esta en escala de grises\n");
	return 0;
}*/
```

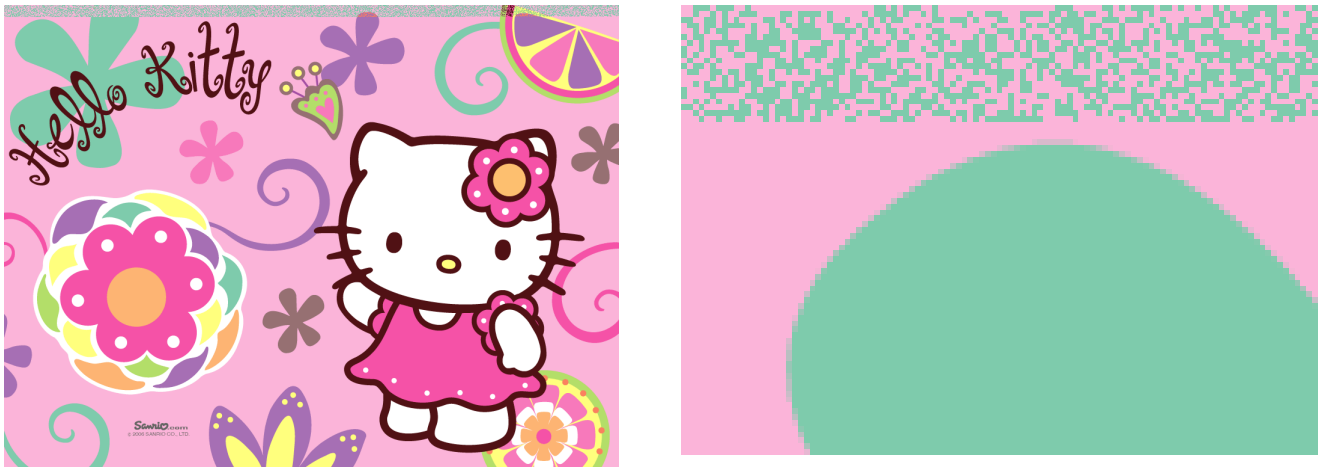


Imagen 7-6. Imagen que ha sido modificada con el algoritmo del programa

Si observamos la parte superior de la imagen observaremos una cantidad de pixeles modificados, lo que nuevamente es sospechoso.

Una vez que tenemos una imagen con información oculta, ¿Cómo verificamos que efectivamente se tiene información almacenada?

Para esto se escribió un código para que muestre los primeros 100 bytes de una imagen, el código se puede estudiar en el ***Apéndice E***, se obtienen los siguientes resultados:

Valor de los pixeles de la ***imagen 7-1***

fd fe fe fd fe fd fd fd fe fd fd fe fd fd fd fe fe fd fd fd fe fe fd fd ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff fe fd fd fd fe fe fe fd fe fe fe fd fe fe fd fd ff ff ff ff
ff ff ff ff fe fe fd fe fd fe fd fd fe fe fe fd fe fe fd fd ff ff ff fd fd fe fd fe fc fc fd fb

Si se muestran los datos de la imagen con la información oculta obtenemos el siguiente resultado:

Valor de los pixeles de la ***imagen 7-4.***

fc fe ff fd ff fc fd fd ff fc fc ff fc fc fc fe fe fc fc fc fe fe fc fc fe fe fe fe fe fe fe fe fe
ff ff fe ff fe ff ff ff fe fe ff fe fe fe fe fe fc fc fc fe fe fe fc fe fe fe fc fe fe fc fc ff ff ff
fe fe fe fe fe ff ff fc ff fc ff fd fd ff fe ff fd fe ff fc fc ff fe fe fd fc ff fc fe fc fd fd fb

Logramos observar que efectivamente los bits están siendo modificados.

CAPITULO 8

PRUEBAS:

Cuando se utiliza el programa muestra en pantalla datos de la información que ocultó, en caso de encontrar algún error o que alguna especificación no se cumpla, el programa finaliza.

Inserción de poco texto a la imagen GIF.

Archivo de texto: 105 bytes.
Dimensiones de la imagen GIF:
ancho: 576 pixeles
alto: 720 pixeles

```
/estego$ ./estego -e ejemplo.txt grises.gif 12345678 -o salida.gif
```

El programa muestra lo siguiente:

```
SE OCULTARÁ UN ARCHIVO

archivo cifrado con éxito, se procede con la ocultación
Se leyó correctamente la imagen

La cantidad de información a almacenar es de: 112 bytes
La cantidad de información que se puede almacenar es de: 51840 bytes
se modificaron 896 pixeles
PROCESO TERMINADO CON ÉXITO...
```

La cantidad de información a almacenar no coincide con el tamaño del archivo de texto debido a que el algoritmo DES agrego bytes para completar el bloque y además se agregó información del tamaño del archivo a la imagen.

El programa puede ocultar la cantidad exacta de información en la imagen GIF.

Archivo: 105 bytes.
Dimensiones de la imagen GIF:
Ancho: 30 pixeles.
Alto: 30 pixeles.

```
/estego$ ./estego -e ejemplo.txt reducido.gif 24682468 -o exacto.gif
```

Mensaje mostrado por el programa:

```
SE OCULTARÁ UN ARCHIVO

archivo cifrado con éxito, se procede con la ocultación
Se leyó correctamente la imagen

La cantidad de información a almacenar es de: 112 bytes
La cantidad de información que se puede almacenar es de: 112 bytes
se modificaron 896 pixeles
PROCESO TERMINADO CON ÉXITO...
```

En este caso se almaceno la cantidad exacta de información.
Si se agrega un byte más al archivo de texto, se obtiene lo siguiente.

Archivo: 106 bytes.
Dimensiones de la imagen GIF:
Ancho: 30 pixeles.
Alto: 30 pixeles.

```
/estego$ ./estego -e ejemplo.txt reducido.gif 24682468 -o exacto.gif
```

Mensaje mostrado por el programa:

```
SE OCULTARÁ UN ARCHIVO

archivo cifrado con éxito, se procede con la ocultación
Se leyó correctamente la imagen

La cantidad de información a almacenar es de: 120 bytes
La cantidad de información que se puede almacenar es de: 112 bytes

pixeles insuficientes para almacenar la información
```

En este caso aumentar un caracter más al archivo de texto hace que la información a ocultar aumente 8 bytes, y no es posible almacenar la información en ese tamaño de archivo GIF.

Si se desea utilizar una imagen GIF que no esté en escala de grises, el programa finalizará la ejecución, esto debido a que se intenta garantizar el proceso de ocultación.

Si se intenta utilizar la *imagen 7-5* (Color) del capítulo anterior, el programa mostrará el siguiente mensaje:

```
SE OCULTARÁ UN ARCHIVO

archivo cifrado con éxito, se procede con la ocultación
Se leyó correctamente la imagen

La imagen no esta en escala de grises
```

Para utilizar imágenes correctas se sugiere la utilización de un programa de edición gráfica como GIMP.

Es importante considerar las características de las imágenes GIF ya que visualmente pueden aparentar estar en escala de grises como lo demuestran las siguientes imágenes.

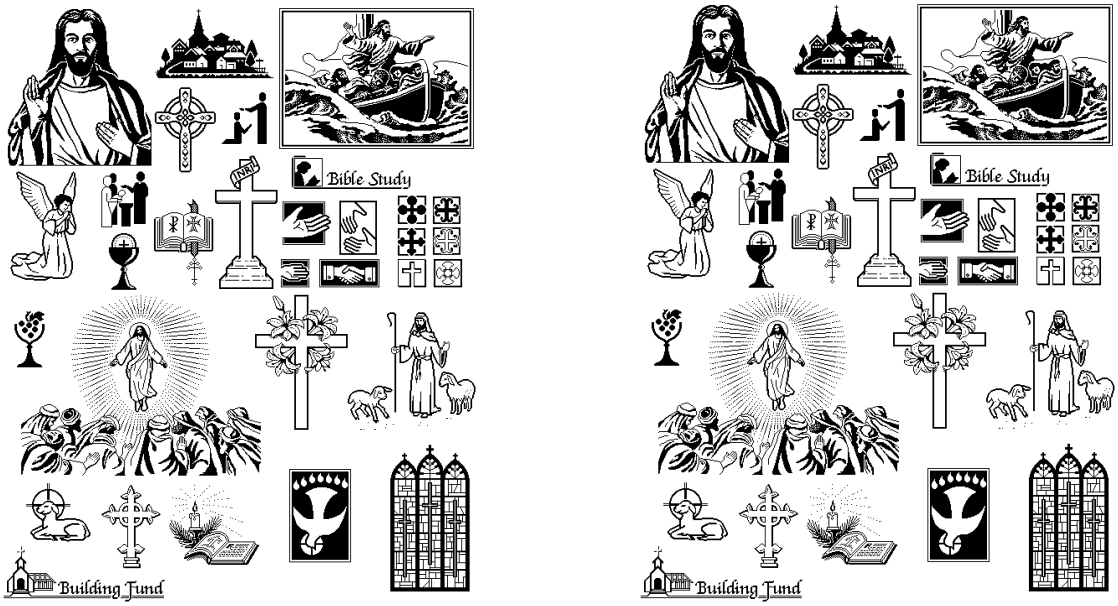


Imagen 8-1. Imágenes que aparentar tener la misma información, la de la izquierda se encuentra en escala de grises y la de la derecha no.

¿Cuál es la diferencia entre las dos imágenes GIF?
Visualmente ninguna diferencia, ambas imágenes parecer estar en escala de grises, pero lo que sucede es que tienen paletas de colores diferentes. La imagen de la izquierda tiene una paleta de colores de 256 tonos lo que garantiza que pueda estar en escala de grises y la imagen de la derecha tiene una paleta de colores de tamaño dos, lo que significa que solo hay dos colores en la imagen (blanco y negro).

CAPITULO 9

CONCLUSIONES:

Al finalizar éste proyecto se logró obtener un programa que reciba como entrada un archivo de texto y una imagen GIF, cifre el archivo de texto y lo oculte dentro de la imagen GIF.

Este proyecto pudiera ser mejorado si se agrega un algoritmo de cifrado/descifrado más robusto, lo que implicaría que la información estuviera aún más protegida.

De la misma forma se pudieran agregar módulos para que el programa no solo realice el proceso esteganográfico en imágenes GIF, sino que pudieran reconocer otro formato de archivos digitales, tales como; video, otros formatos de imágenes, audio, etc.

Además se pudieran modificar para que fuera multiplataforma.

CAPITULO 10

REFERENCIAS

[1] Investigator´s guide to Steganography
Gregory Kipper. Auerbach Publications, 2004
ISBN: 0-8493-2433-5

[2] Técnicas Criptográficas de protección de datos.
Amparo Fúster Sabater. Alfaomega, 2001
ISBN: 970-15-0602-2

[3] GERARDINO GARCÍA, María Alejandra, Martínez Marín, Andrés y Ríos Rosas, Francy, **"Sistema Esteganográfico GEHEIM"**. Revista Digital Universitaria [en línea]. Abril 2008, Vol. 9, No. 4.
http://www.revista.unam.mx/vol.9/num4/art23/int23.htm#a
ISSN: 1607-6079
Última consulta: Enero 2015.

[4] http://linux01.gwdg.de/~alatham/stego.html
Última consulta: Enero 201

[5] http://wbstego.wbailer.com/
Última consulta: Enero 2015

[6] http://www.egov.ufsc.br/portal/sites/default/files/esteganografia1.pdf
Última consulta: Enero 2015.

[7] Criptografia y Seguridad en Computadoras
Cuarta Edición
Manuel J. Lucena López
http://sertel.upc.edu/tdatos/Libros/Lucena.pdf
Última consulta: Enero 2015.

[8] http://giflib.sourceforge.net/
Última consulta: Enero 2015.

[9] Descripcion del algoritmo DES.
Jorge Sanchez Arriazu.
http://www.tierradelazaro.com/public/libros/des.pdf
Última consulta: Enero 2015

CAPITULO 11

APENDICES:

Apéndice A

Biblioteca implementada para todas las funciones que se encargan de cifrar y descifrar el archivo de texto, asi como verificar los parámetros de la línea de comandos.

src/Cifrado.c

```
#include "../include/cifrado.h"

/********FUNCION QUE SE ENCARGA REALIZAR LAS PERMUTACIONES*************/
unsigned long long permutacion ( unsigned char tabla[ ], int tam ,unsigned long long
bloque){

	int i;
	unsigned long long mask, bloque_perm;
	int ptr;

	unsigned long long bits[64]; //se utiliza para obtener el bit a ser permutado

	bits[0]=0x01;
	for(i=1; i < 64 ; i++){
		 bits[i] = (bits[i-1]) * 2;
	}

	bloque_perm = 0x0;
	for(i=0 ; i<tam ; i++){
		mask =0x0;
		ptr = tabla[i]-1;
		mask = bloque & bits[63-ptr];
		mask = mask<<(ptr);
		mask = mask>>i;
		bloque_perm = bloque_perm | mask;
	}

	return bloque_perm;
}

/******* PRIMERA PERMUTACION **************/
/****** para la clave 64 - 56 *************/
unsigned char PC_1[ ] = {
	57, 49, 41, 33, 25, 17,  9, 1,
	58, 50, 42, 34, 26, 18, 10, 2,
	59, 51, 43, 35, 27, 19, 11, 3,
	60, 52, 44, 36, 63, 55, 47,39,
	31, 23, 15,  7, 62, 54, 46,38,
	30, 22, 14,  6, 61, 53, 45,37,
	29, 21, 13,  5, 28, 20, 12, 4
};
```

```
/******* SEGUNDA PERMUTACION **************/
/******* PARA LAS SUBCLAVES  **************/
unsigned char PC_2[] = {
       14, 17, 11, 24,  1,  5,  3, 28,
       15,  6, 21, 10, 23, 19, 12,  4,
       26,  8, 16,  7, 27, 20, 13,  2,
       41, 52, 31, 37, 47, 55, 30, 40,
       51, 45, 33, 48, 44, 49, 39, 56,
       34, 53, 46, 42, 50, 36, 29, 32
};

int rot[] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};//Se utiliza para realizar las 16
rotaciones de las subclaves

unsigned long long Ki[16]; //vector que almacena las 16 subclaves para cifrar

/*****FUNCION QUE GENERA LAS 16 SUBCLAVES************/
void genera_claves( unsigned long long clave)
{
       int i, j;
       unsigned long long clave_perm, CD;
       unsigned long long mask;
       unsigned long C, D;

       clave_perm = 0x0000000000000000;
 /**********PRIMERA PERMUTACION DE LA CLAVE**************/
       clave_perm = permutacion(PC_1, 56, clave);
       clave_perm = clave_perm>>8;

 /**********SE DIVIDE EN 28 BITS LA CLAVE**************/
       D = clave_perm & 0xfffffff;
       C = (clave_perm & 0xfffffff0000000)>>28;

 /**********SE REALIZAN LAS 16 ROTACIONES DE C Y D*****************/
       i=0;
       for(i=0 ; i<16 ; i++){
              mask = 0x0;
       /*********SE REALIZAN LAS ROTACIONES*********/
              if(rot[i]==1){
                     mask = (C & 0x8000000)>>27;
                     C = (C<<1)|mask;
                     C = C & 0xfffffff;

                     mask = (D & 0x8000000)>>27;
                     D = (D<<1)|mask;
                     D = D & 0xfffffff;
                     }
              else{
                     mask = (C & 0x8000000)>>27;
                     C = (C<<1)|mask;
                     C = C & 0xfffffff;
                     mask = (C & 0x8000000)>>27;
                     C = (C<<1)|mask;
                     C = C & 0xfffffff;

                     mask = (D & 0x8000000)>>27;
                     D = (D<<1)|mask;
                     D = D & 0xfffffff;
                     mask = (D & 0x8000000)>>27;
                     D = (D<<1)|mask;
                     D = D & 0xfffffff;
```

```
		}
	/********SE CONCANTENAN C Y D**********/
		CD = 0x0000000000000000;
		CD = (C<<28)|D;
		CD = CD<<8;

	/**********SE PERMUTA CD*****************/
		Ki[i]=0x0;
		Ki[i]= permutacion(PC_2, 48, CD);
	}
}

/****FUNCION QUE INVIERTE LAS CLAVES PARA DESCIFRAR********/
void invertir_claves( )
{
	int i, j;
	unsigned long long Ki_temp[16];

	for(i=0 ; i<16 ; i++){
		Ki_temp[i]= Ki[i];
	}
	for(i=0, j=15; i<16 ; i++, j--){
		Ki[i] = Ki_temp[j];
	}
}

/*********  S-CAJAS DE SUSTITUCIÓN   *********/
unsigned char s_boxes[8][4][16]={
	{{14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7}, {0, 15, 7, 4, 14, 2, 13,
1, 10, 6, 12, 11, 9, 5, 3, 8}, {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},
{15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}},
	{{15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10}, {3, 13, 4, 7, 15, 2, 8,
14, 12, 0, 1, 10, 6, 9, 11, 5}, {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
{13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}},
	{{10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8}, {13, 7, 0, 9, 3, 4, 6,
10, 2, 8, 5, 14, 12, 11, 15, 1}, {13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},
{1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12}},
	{{7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15}, {13, 8, 11, 5, 6, 15, 0,
3, 4, 7, 2, 12, 1, 10, 14, 9}, {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},
{3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}},
	{{2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9}, {14, 11, 2, 12, 4, 7, 13,
1, 5, 0, 15, 10, 3, 9, 8, 6}, {4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},
{11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3}},
	{{12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11}, {10, 15, 4, 2, 7, 12, 9,
5, 6, 1, 13, 14, 0, 11, 3, 8}, {9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},
{4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13}},
	{{4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1}, {13, 0, 11, 7, 4, 9, 1,
10, 14, 3, 5, 12, 2, 15, 8, 6}, {1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},
{6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12}},
	{{13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7}, {1, 15, 13, 8, 10, 3, 7,
4, 12, 5, 6, 11, 0, 14, 9, 2}, {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},
{2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}},
};

/*** PRIMERA PERMUTACION DEL BLOQUE DE TEXTO**********/
unsigned char IP[] = {
	58, 50, 42, 34, 26, 18, 10, 2,
	60, 52, 44, 36, 28, 20, 12, 4,
	62, 54, 46, 38, 30, 22, 14, 6,
	64, 56, 48, 40, 32, 24, 16, 8,
	57, 49, 41, 33, 25, 17,  9, 1,
```

```
	59, 51, 43, 35, 27, 19, 11, 3,
	61, 53, 45, 37, 29, 21, 13, 5,
	63, 55, 47, 39, 31, 23, 15, 7
};

/******PERMUTACION DE EXPANSION DEL BLOQUE DERECHO DE 32 A 48*********/
unsigned char EP[]={
	32,  1,  2,  3,  4,  5,  4,  5,
	 6,  7,  8,  9,  8,  9, 10, 11,
	12, 13, 12, 13, 14, 15, 16, 17,
	16, 17, 18, 19, 20, 21, 20, 21,
	22, 23, 24, 25, 24, 25, 26, 27,
	28, 29, 28, 29, 30, 31, 32,  1
};

/******  PERMUTACION POSTERIOR A LAS S-CAJAS    **********/
unsigned char PP[]={
	16, 7, 20, 21, 29, 12, 28, 17,
	1, 15, 23, 26, 5, 18, 31, 10,
	2, 8, 24, 14, 32, 27, 3, 9,
	19, 13, 30, 6, 22, 11, 4, 25,
};

/****** PERMUTACION FINAL DEL TEXTO ********/
unsigned char PF[]={
	40, 8, 48, 16, 56, 24, 64, 32,
	39, 7, 47, 15, 55, 23, 63, 31,
	38, 6, 46, 14, 54, 22, 62, 30,
	37, 5, 45, 13, 53, 21, 61, 29,
	36, 4, 44, 12, 52, 20, 60, 28,
	35, 3, 43, 11, 51, 19, 59, 27,
	34, 2, 42, 10, 50, 18, 58, 26,
	33, 1, 41,  9, 49, 17, 57, 25,
};

/*****  FUNCION QUE SE ENCARGA DE CIFRAR O DESCIFRAR UN BLOQUE DE 64 BITS  **********/
unsigned long long procesa_bloque( unsigned long long texto){

	int i, fila, columna, ronda;
	unsigned long temp;
	unsigned long long bloque_l, bloque_r, bloque_rE, s_c, s_cxor;
	unsigned long long texto_p, texto_c;
	unsigned long long mascara;
	unsigned char B[8], S[8];

	/*****PRIMERA PERMUTACION Y DIVISION DEL BLOQUE DE TEXTO******/
	texto_p = permutacion(IP, 64, texto );
	bloque_l = texto_p & 0xffffffff00000000;
	bloque_r = texto_p & 0x00000000ffffffff;
	bloque_r = bloque_r<<32;

	/*******APLICACION DE LAS 16 SUBCLAVES PARA CIFRAR*********/
	for(ronda = 0; ronda<16; ronda++){

		/*****EXPANSION DEL BLOQUE DE TEXTO DERECHO DE 32 A 48 BITS*****/
		bloque_rE = permutacion(EP, 48, bloque_r);

		/***********APLICACION DE LA SUBCLAVE-i************/
		bloque_rE = (bloque_rE>>16) ^ (Ki[ronda]>>16);
```

```
/******PARTICION DEL BLOQUE EN 8 SEGMENTOS DE 6 BITS CADA UNO********/
              mascara = 0x3f;
              for(i=0; i<8 ; i++){
                     B[7-i]= bloque_rE & mascara;
                     bloque_rE= bloque_rE>>6;
              }

              /******    SE SUSTITUYEN LOS SEGMENTOS POR LAS S-CAJAS    **********/
              for(i=0 ; i<8 ; i++){
                         fila=columna=temp = 0;
                            temp = B[i] & 0x01;
                            fila = fila | temp;
                            temp = B[i] & 0x20;
                            temp = temp>>4;
                            fila = fila | temp;

                            temp = (B[i] & 0x1e)>>1;
                            columna = columna | temp;
                            S[i] = s_boxes[i][fila][columna];
              }

              /******SE CONCATENAN LAS S-CAJAS PARA FORMAR UN BLOQUE DE 32 BITS****/
              s_c=0x0;
              for ( i = 0; i < 8; i++){
                     temp=0;
                     temp = S[i] & 0xf;
                     temp = temp<<(28-(i*4));
                     s_c=s_c|temp;
              }
              s_c = s_c<<32;

              /*******SE PERMUTA EL BLOQUE*****/
              s_c = permutacion(PP, 32, s_c);

              /******** SE REALIZA XOR CON EL BLOQUE IZQUIERDO Y SE GENERAN LOS NUEVOS
BLOQUES***********/
              s_cxor = s_c ^ bloque_l;
              bloque_l = bloque_r;
              bloque_r = s_cxor;
       }
       texto_c = 0x0;
       texto_c = bloque_r;
       texto_c = texto_c |(bloque_l>>32);

       texto_c = permutacion( PF, 64, texto_c);

       return texto_c;
}

typedef union {
       unsigned long long text_l;
       unsigned char texto[8];
}bloque_texto;

/*****FUNCION QUE SE ENCARGA DE LEER LOS DATOS DE UN ARCHIVO Y PROCESARLOS,
      SI BANDERA = 0 ENTONCES EL ARCHIVO SE CIFRA SINO ENTONCES SE DESCIFRA ****/
int procesa_fichero(char *entrada, char *salida, int bandera){

       FILE *f_entrada, *f_salida;
       int tam_archivo, lecturas, residuo;
       int i, j;
```

```
	bloque_texto texto_o, texto_procesado;

	if( (f_entrada = fopen(entrada, "rb+") )==NULL ){
		fprintf(stderr, "Error al abrir el archivo que se desea procesar\n" );
		return 0;
	}
	if( (f_salida = fopen(salida, "wb+") )==NULL ){
		fprintf(stderr, "Error al crear el archivo de salida\n" );
		return 0;
	}

	/*******segmento que se utiliza para determinar el tamaño del archivo********/
	if(bandera == 0){
		fseek (f_entrada, 0L, SEEK_END);
		tam_archivo = ftell (f_entrada);
		fwrite(&tam_archivo, sizeof(int), 1, f_salida);
		fseek(f_entrada, 0L, SEEK_SET);
	}
	else if(bandera ==1){
		fread(&tam_archivo, sizeof(int), 1, f_entrada);
	}

	lecturas = tam_archivo / 8;
	residuo = tam_archivo % 8;

	for( i=0 ; i<lecturas ; i++ ){
		fread(texto_o.texto, sizeof(unsigned char),8,f_entrada);
		texto_procesado.text_l = procesa_bloque( texto_o.text_l);
		fwrite(texto_procesado.texto,sizeof(unsigned char), 8, f_salida);
	}
	if(residuo != 0){

		for(i=0 ; i<8; i++) texto_o.texto[i]=0x0;
		if(bandera == 0 ){
			fread(texto_o.texto, sizeof(unsigned char), residuo, f_entrada);
			texto_procesado.text_l = procesa_bloque(texto_o.text_l);
			fwrite(texto_procesado.texto,sizeof(unsigned char), 8, f_salida);
		}
		else{
			fread(texto_o.texto, sizeof(unsigned char), 8, f_entrada);
			texto_procesado.text_l = procesa_bloque(texto_o.text_l);
			fwrite(texto_procesado.texto,sizeof(unsigned char), residuo,
f_salida);
		}
	}
	fclose(f_entrada);
	fclose(f_salida);

	return 1;
}

/********** Función que verifica que los parametros sean correctos**********/
/********** consulte el archivo ayuda.txt para obtener más información******/
int verifica_param(int n_param, char *parametros[ ]){

	//Se verifica para ocultar el archivo
	if(    ( (n_param == 5) || (n_param == 7) ) && ( strcmp( parametros[1], "-e" )==0
)  ){

		if(  (extension(parametros[2], 2 )==0) || (extension(parametros[3], 1
)==0) ){
```

```
                        return muestra_ayuda(3);
                }
                if(strlen(parametros[4])!=8){
                        return muestra_ayuda(2);
                }
                if( (n_param==7) && (strcmp(parametros[5],"-o")==0) ){
                        if( extension(parametros[6], 1 )==0 ) return muestra_ayuda(3);

                        return 2;
                }
                else if (n_param == 5 )return 1;
                else return muestra_ayuda(1);

                return 1;
        }
        //se verifica para obtener el archivo
        else if(      ( (n_param == 6) || (n_param == 4) ) && ( strcmp( parametros[1], "-
d" )==0  ) ){

                if(  extension(parametros[2], 1 )==0  ){
                        return muestra_ayuda(3);
                }
                if( strlen(parametros[3])!=8 ){
                        return muestra_ayuda(2);
                }
                if( n_param==6){
                        if( strcmp(parametros[4], "-o") != 0) return muestra_ayuda(1);
                        if(extension(parametros[5], 2 ) == 0) return muestra_ayuda(3);
                        return 4;
                }
                return 3;
        }
        //se muestra la ayuda
        else if( n_param == 2 ){
                if(strcmp(parametros[1], "--h")==0 ){
                        if( muestra_ayuda(0)== 0 ) return 0;
                        else return 0;
                }
                else{
                        return muestra_ayuda(1);
                }
        }
        else{
                return muestra_ayuda(1);
        }
}

/***********Funcion que revisa que el nombre tenga la extension correcta**********/
/************* Si la bandera es 1, se verifica que sea gif **********************/
/************* Si la bandera es 2, se verifica que sea txt **********************/
int extension(char fichero[ ], int bandera ){

switch(bandera){
        case 1:
                if((strstr(fichero,".gif")!=NULL)||(strstr(fichero,".GIF")!=NULL))return 1;
                else return 0;
        break;
        case 2:
                if((strstr(fichero,".txt")!=NULL)||(strstr(fichero,".TXT")!=NULL))return 1;
                else return 0;
        break;
```

```
	}
}

/********Función que muestra la ayuda al usuario*************/
/********en caso de error en los parametros***************/
int muestra_ayuda(int f ){
	FILE *f_ayuda;
	int c;
	char ayuda[ ]= "ayuda.txt";

	if(f==0){
		if( (f_ayuda = fopen(ayuda, "rb+") )==NULL ){
			fprintf(stderr, "Error al mostrar el archivo de ayuda\n" );
			return 0;
		}
		else{
			while( (c = fgetc(f_ayuda)) != EOF ){
				putchar(c);
			}
		}
		printf("\n\n");
	}
	else if(f==1){
		fprintf(stderr,"Parametros incorrectos. -----> (estego  --h) si desea
obtener ayuda\n");
		return 0;
	}
	else if(f==2){
		fprintf(stderr,"Clave incorrecta. -----> (estego  --h) si desea obtener
ayuda\n");
		return 0;
	}
	else if(f==3){
		fprintf(stderr,"Nombre o extensión incorrecta para los archivos. ----->
(estego  --h) si desea obtener ayuda\n");
		return 0;
	}
	return 1;
}
```

Apéndice B

Código correspondiente a la librería del proceso esteganográfico.

src/estego_lib.c

```
#include <stdio.h>
#include <fcntl.h>
#include "../include/gif_lib.h"
#include "../include/estego_lib.h"

unsigned char arreglo_pixeles[8];

/****Función que coloca un bit en un caracter, en el lugar**************/
/****indicado por indice*************************************************/
unsigned char colocaI(unsigned char texto, unsigned short int indice, unsigned short int
bit){

	if(indice >7 || bit > 1){
		printf("error\n");
		return;
	}
	if(bit==1)
		texto |= (1<<indice);
	else
		texto &= ~(1<<indice);

	return texto;
}

/****Función que extrae un bit el menos significativo de un caracter****/
/***********************************************************************/
unsigned short int extrae(unsigned char texto_c){
	unsigned short int bit;

	bit = texto_c & 0x01;

	return bit;
}

/****Función que coloca un caracter en un arreglo de 8 pixeles**********/
/******utilizando el bit menos significativo de cada pixel*************/
void coloca_caracter(unsigned char pixeles[], unsigned char  caracter){

	int i;
	unsigned short int bit =0x00;

	for(i=0; i<8; i++){
		bit = caracter & (1<<i);
		bit = bit>>i;
		arreglo_pixeles[i]= colocaI(pixeles[i], 0,  bit);
	}

}
```

```
/****Función que obtiene un caracter de un conjunto de ocho pixeles*****/
/************************************************************************/
unsigned char obtener_caracter(unsigned char pixeles[]){

	unsigned char caracter=0x00;
	int i;

	unsigned short int bit;

	for(i=0; i<8 ; i++){
		caracter = colocaI(caracter, i, extrae(pixeles[i]));
	}

	return caracter;

}

/******Función que coloca el archivo de texto dentro de un GIF**********/
/************************************************************************/
int coloca_archivo( char nombre_archivo[ ], char nombre_imagen[ ], char imagen_copia[ ]
){

	int *error=NULL;
	int x;
	int cerrar=0;
	long i;
	GifFileType *Gif, *Gif_copia;
	SavedImage *Imagen_Gif = NULL;
	unsigned  char pixeles[8];
	int bandera;
	FILE *imag_copia;
	int r;

 /*****VARIABLES QUE SE NECESITAN PARA EL ARCHIVO DE TEXTO******/
	FILE *archivo;
	unsigned char byte;
	long ptr_imagen, j, ptr_temp;
	unsigned long tam_archivo;

 /***Se verifica que exista el archivo de imagen****/
	if( (archivo = fopen(nombre_imagen, "rb+") )==NULL ){
		fprintf(stderr, "No existe el archivo de imagen\n" );
		return 0;
	}

 /**********SE LEE LA IMAGEN ORIGINAL CON TODA SU INFORMACION******************/
	if((Gif=DGifOpenFileName(nombre_imagen, error))==NULL){
		printf("Se genero el siguiente error de apertura: %s\n",
GifErrorString(Gif->Error) );
		return 0;
	}
	else{
		if(DGifSlurp(Gif)==GIF_OK)   printf("Se leyó correctamente la imagen\n\n");
		else{
			printf("Se generó el siguiente error de apertura:
%s\n",GifErrorString(Gif->Error) );
			return 0;
		}
	}
```

```
/**********SE VERIFICAN LAS CARACTERISTCAS DE LA IMAGEN ****************/
	if( verifica_gif(Gif) == 0 ){
		return 0;
	}
/*********SEGMENTO QUE LEERÁ EL ARCHIVO DE TEXTO**********/
	if( (archivo = fopen(nombre_archivo, "rb+") )==NULL ){
		fprintf(stderr, "Error al abrir el archivo que se desea procesar\n" );
		return 0;
	}
	if( verifica_archivos(Gif, archivo)== 0 ){
		return 0;
	}

/***********SE VERIFICA QUE LA IMAGEN DE SALIDA NO EXISTA***********/
	if( (imag_copia = fopen(imagen_copia, "rb+" ) ) != NULL ){
		do{
			r=0;
			printf("\nLa imagen de salida existe. ¿Desea eliminarla y/n?");
			r =  getchar();

			switch(r){
				case 121:
					fclose(imag_copia);
					remove(imagen_copia);
					break;
				case 110:
					fprintf(stderr, "Se finalizará el programa...\n");
					return 0;
					break;
			}
		}while((r!=121) && (r!=110));
	}

/***********SE DETERMINA EL TAMAÑO DEL ARCHIVO*******************/
	fseek(archivo, 0L, SEEK_END);
	tam_archivo = ftell(archivo);
	fseek(archivo, 0L, SEEK_SET);

	Imagen_Gif = Gif->SavedImages;
	ptr_imagen = 0;
	i=0;

	/************SE INGRESA EL TAMAÑO DEL ARCHIVO**************/
	for(x=0 ; x<4 ; x++){
		ptr_temp = ptr_imagen;
		byte = tam_archivo & 0x00ff;
		for(j=0; j<8 ; j++){
			pixeles[j]= Imagen_Gif->RasterBits[ptr_imagen];

			ptr_imagen+=1;
		}
		coloca_caracter(pixeles, byte);

		for (j = 0; j < 8; j++){
			Imagen_Gif->RasterBits[ptr_temp] = arreglo_pixeles[j];
			ptr_temp+=1;
		}
		tam_archivo = tam_archivo>>8;
	}

	/********SE INGRESA LA INFORMACION DEL ARCHIVO A LA IMAGEN********/
```

```
	while((fread(&byte, sizeof(unsigned char),1,archivo))>0)
	{
		ptr_temp = ptr_imagen;

		for(j=0; j<8 ; j++){
			pixeles[j]= Imagen_Gif->RasterBits[ptr_imagen];
			ptr_imagen+=1;
		}

		coloca_caracter(pixeles, byte);

		for (j = 0; j < 8; j++){
			Imagen_Gif->RasterBits[ptr_temp] = arreglo_pixeles[j];
			ptr_temp+=1;
		}
	i++;
	}
	printf("se modificaron %ld pixeles \n", ptr_imagen);

 /****Se escribe en la imagen copia, la misma información de la **/
 /*********************   imagen original    *********************/
	Gif_copia = EGifOpenFileName(imagen_copia, 1, error);

	Gif_copia->SWidth = Gif->SWidth;
	Gif_copia->SHeight = Gif->SHeight;
	Gif_copia->SColorResolution = Gif->SColorResolution;
	Gif_copia->SBackGroundColor = Gif->SBackGroundColor;
	Gif_copia->AspectByte = Gif->AspectByte;
	Gif_copia->SColorMap = Gif->SColorMap;
	Gif_copia->ImageCount = Gif->ImageCount;
	Gif_copia->Image = Gif->Image;
	Gif_copia->SavedImages = Imagen_Gif;
	Gif_copia->ExtensionBlockCount = Gif->ExtensionBlockCount;
	Gif_copia->ExtensionBlocks = Gif->ExtensionBlocks;

	if(EGifSpew(Gif_copia)==GIF_ERROR){
		printf("Se encontro el siguiente error: %s\n", GifErrorString(Gif_copia-
>Error) );
	}

	fclose(archivo);
	DGifCloseFile(Gif, error);
	//EGifCloseFile(Gif_copia, error);
	return 1;

}

/******Función que obtiene un archivo de texto de una imagen GIF********/
/***********************************************************************/
int obtener_archivo( char nombre_archivo[ ], char nombre_imagen[ ]){

	int *error=NULL;
	int cerrar=0;
	long i;
	GifFileType *Gif;
	SavedImage *Imagen_Gif = NULL;
	unsigned  char pixeles[8];
	unsigned char mascara;
	int x, r;
	FILE *fi;
```

```
        /*****VARIABLES QUE SE NECESITAN PARA EL ARCHIVO DE TEXTO******/
        FILE *archivo;
        unsigned char byte;
        long ptr_imagen, j, ptr_temp;
        unsigned long tam_archivo;

    /********* Se verifica que exista la imagen contenedora ***********/
        if( (fi = fopen(nombre_imagen, "rb+") )==NULL ){
                fprintf(stderr, "No existe el archivo de imagen\n" );
                return 0;
        }

    /**********Se lee la imagen con toda su informacion**********************/
        if((Gif=DGifOpenFileName(nombre_imagen, error))==NULL){
                printf("Se genero el siguiente error de apertura: %s\n",
GifErrorString(Gif->Error) );
                return 0;
        }
        else{
                if(DGifSlurp(Gif)==GIF_OK)   printf("Se leyó correctamente la imagen\n\n");
                else{
                printf("Se generó el siguiente error de apertura:
                        %s\n",GifErrorString(Gif->Error) );
                       return 0;
                }
        }

        /***** SE VERIFICA QUE LA IMAGEN CUMPLA CON LAS ESPECIFICACIONES*********/
        if(verifica_gif(Gif) == 0 ){
                return 0;
        }

        /***********SE VERIFICA QUE EL ARCHIVO DE SALIDA NO EXISTA ***********/
        if( (archivo = fopen(nombre_archivo, "rb+" ) ) != NULL ){
                do{
                        r=0;
                        printf("\nEl archivo de salida existe. ¿Desea eliminarlo y/n?");
                        r =  getchar();
                        switch(r){
                                case 121:
                                        fclose(archivo);
                                        remove(nombre_archivo);
                                        break;
                                case 110:
                                        fprintf(stderr, "Se finalizará el programa...\n");
                                        return 0;
                                        break;
                        }

                }while((r!=121) && (r!=110));

        }

        if( (archivo = fopen(nombre_archivo, "wb+") )==NULL ){
                fprintf(stderr, "Error al crear el archivo de salida\n" );
                return 0;
        }

        Imagen_Gif = Gif->SavedImages;
```

```
/************SE OBTIENE EL TAMAÑO DEL ARCHIVO**************/
	ptr_imagen = 0;
	tam_archivo = 0;
	for(x=0 ; x<4 ; x++){
		for(j=0; j<8 ; j++){
			pixeles[j]= Imagen_Gif->RasterBits[ptr_imagen];
			ptr_imagen+=1;
		}
		byte = obtener_caracter(pixeles);
		tam_archivo = tam_archivo | (byte<<(x*8));

	}

	/************SE ESCRIBEN LOS DATOS EN EL ARCHIVO DE TEXTO**************/
	for(i=0 ; i<tam_archivo ; i++){
		byte = 0;
		for(j=0; j<8 ; j++){
			pixeles[j]= Imagen_Gif->RasterBits[ptr_imagen];
			ptr_imagen+=1;
		}
		byte = obtener_caracter(pixeles);
		fwrite(&byte,sizeof(unsigned char), 1, archivo );
	}

	fclose(archivo);
	DGifCloseFile(Gif, error);

	return 1;
}

/******Función que verifica que el tamaño del archivo de texto ********/
/***************  alcance dentro de la imagen GIF  ********************/
int verifica_archivos(GifFileType *Gif, FILE *archivo){
	unsigned long pixeles;
	unsigned long tam_archivo;

	pixeles = Gif->SWidth * Gif->SHeight;

	fseek(archivo, 0L, SEEK_END);
	tam_archivo = ftell(archivo);
	fseek(archivo, 0L, SEEK_SET);

	printf("La cantidad de informacion a almacenar es de: %ld bytes\n",
                                                                (tam_archivo+4));
	printf("La cantidad de informacion que se puede almacenar es de: %ld bytes\n",
                                                                 (pixeles/8));

	if( (pixeles/8) >= (tam_archivo+4) ){
		//printf("el archivo si alcanza\n");
		return 1;
	}
	else{
		fprintf(stderr,"\npixeles insuficientes para almacenar la información\n");
		return 0;
	}
}
```

```
/******* Función que verifica que la imagen Gif tenga las ***********/
/****************  especificaciones correctas  *********************/
int verifica_gif( GifFileType *Gif ){
	unsigned char rojo, verde, azul;
	int ancho, alto, n_gifs, tam_paleta, i;

	ancho = Gif->SWidth;
	alto = Gif->SHeight;
	n_gifs = Gif->ImageCount;
	tam_paleta = Gif->SColorMap->ColorCount;

	if( ancho > 1024 ){
		fprintf(stderr, "La imagen supera el tamaño permitido, utilizar una imagen
				más pequeña\n");
		return 0;
	}
	if( alto > 768  ){
		fprintf(stderr, "La imagen supera el tamaño permitido, utilizar una imagen
				 más pequeña\n");
		return 0;
	}
	if( n_gifs > 1 ){
		fprintf(stderr, "Utilizar GIF de una sola imagen\n");
		return 0;
	}

	if(tam_paleta == 256){
		for(i=0 ; i<tam_paleta ; i++){
			rojo = Gif->SColorMap->Colors[i].Red;
			verde = Gif->SColorMap->Colors[i].Green;
			azul = Gif->SColorMap->Colors[i].Blue;

			if( (rojo ==verde) && (verde==azul) ){
			}
			else{
				fprintf(stderr, "La imagen no está en escala de grises\n");
				return 0;
			}
		}
	}
	else{
		fprintf(stderr, "La imagen no está en escala de grises\n");
		return 0;
	}


	return 1;
}
```

Apéndice C.

Código principal.

main.c

```
#include <stdio.h>
#include "./include/gif_lib.h"
#include "include/cifrado.h"
#include "include/estego_lib.h"

typedef union{
	unsigned char *clave;
	unsigned long long *clave_l;
}bloque_clave;

int main(int argc, char *argv[])
{
	char *archivo;
	char *imagen;
	char *imagen_copia;
	char archivo_copia[ ] = "./temp/temp.txt";
	char *archivo_original;
	int bandera;
	bloque_clave key;

	bandera = verifica_param(argc, argv);
	switch(bandera){
		case 0:
			return 0;
			break;
		case 1:
			printf("\t\nSE OCULTARÁ UN ARCHIVO\n\n");
			archivo = argv[2];
			imagen = argv[3];
			key.clave = argv[4];
			imagen_copia = "./salida/output.gif";
			genera_claves(*key.clave_l);

			if(procesa_fichero(archivo,archivo_copia,0)!= 0)
				printf("archivo cifrado con éxito, se procede con la
					ocultacion\n");
			else return 0;
			if( coloca_archivo(archivo_copia, imagen, imagen_copia ) != 0){
				printf("PROCESO TERMINADO CON ÉXITO... CONSULTE EL
					DIRECTORIO SALIDA\n\n");
				remove(archivo_copia);
			}
			else{
				remove(archivo_copia);
				return 0;
			}
			break;
		case 2:
			printf("\t\nSE OCULTARÁ UN ARCHIVO\n\n");
			archivo = argv[2];
			imagen = argv[3];
			key.clave = argv[4];
			imagen_copia = argv[6];
			genera_claves(*key.clave_l);
			if(procesa_fichero(archivo, archivo_copia, 0)!= 0)
```

```
                        printf("archivo cifrado con éxito, se procede con la
                                ocultacion\n");
                else return 0;
                if( coloca_archivo(archivo_copia, imagen, imagen_copia ) != 0){
                        printf("PROCESO TERMINADO CON ÉXITO...\n\n");
                        remove(archivo_copia);
                }
                else{
                        remove(archivo_copia);
                        return 0;
                }
                break;
        case 3:
                printf("\nSE PROCEDERÁ A OBTENER EL ARCHIVO\n\n");
                imagen_copia = argv[2];
                key.clave = argv[3];
                archivo_original = "./salida/output.txt";

                if(obtener_archivo(archivo_copia, imagen_copia) != 0 )
                        printf("se obtuvo el archivo con éxito, se procede a
                                descifrar\n\n");
                else return 0;
                genera_claves(*key.clave_l);
                invertir_claves( );

                if( procesa_fichero(archivo_copia, archivo_original, 1 ) != 0 ){
                        printf("PROCESO TERMINADO CON ÉXITO... CONSULTE EL
                                 DIRECTORIO SALIDA\n\n");
                        remove(archivo_copia);
                }
                else{
                        remove(archivo_copia);
                        return 0;
                }
                break;
        case 4:
                printf("\nSE PROCEDERÁ A OBTENER EL ARCHIVO\n\n");
                imagen_copia = argv[2];
                key.clave = argv[3];
                archivo_original = argv[5];

                if(obtener_archivo(archivo_copia, imagen_copia) != 0 )
                        printf("se obtuvo el archivo con éxito, se procede a
                                descifrar\n\n");
                else return 0;
                genera_claves(*key.clave_l);
                invertir_claves( );
                if( procesa_fichero(archivo_copia, archivo_original, 1 ) != 0 ){
                        printf("PROCESO TERMINADO CON ÉXITO...\n\n");
                        remove(archivo_copia);
                }
                else{
                        remove(archivo_copia);
                        return 0;
                }
                break;
        }
        return 0;
}
```

Apéndice D

Fichero Makefile que se encarga de la compilación del programa:

Make

```
all: estego

estego: main.o dgif_lib.o egif_lib.o gifalloc.o gif_err.o gif_font.o gif_hash.o quantize.o estego_lib.o cifrado.o
	gcc main.o dgif_lib.o egif_lib.o gifalloc.o gif_err.o gif_font.o gif_hash.o quantize.o estego_lib.o
cifrado.o -o estego

main.o: main.c
	gcc -c main.c

dgif_lib.o:  src/dgif_lib.c
	gcc -c src/dgif_lib.c

egif_lib.o: src/egif_lib.c
	gcc -c src/egif_lib.c

gifalloc.o: src/gifalloc.c
	gcc -c src/gifalloc.c

gif_err.o: src/gif_err.c
	gcc -c src/gif_err.c

gif_font.o: src/gif_font.c
	gcc -c src/gif_font.c

gif_hash.o: src/gif_hash.c
	gcc -c src/gif_hash.c

quantize.o: src/quantize.c
	gcc -c src/quantize.c

estego_lib.o: src/estego_lib.c
	gcc -c src/estego_lib.c

cifrado.o: src/cifrado.c
	gcc -c src/cifrado.c

clean:
	-rm *.o estego
```

Apéndice E.

Código generado para mostrar los 100 primeros bytes del bloque de datos de una imagen.

```
#include <stdio.h>
#include "./include/gif_lib.h"

int main(int argc, char *argv[])
{
	int *error=NULL;
	int cerrar=0;
	long i;
	char *imagen;
	GifFileType *Gif;
	SavedImage *Imagen_Gif = NULL;
	unsigned  char pixeles[8];
	unsigned char mascara;

	imagen = argv[1];

	 /**********Se lee la imagen con toda su informacion****************/
	if((Gif=DGifOpenFileName(imagen, error))==NULL){
		printf("Se genero el siguiente error de apertura: %s\n",
			 GifErrorString(Gif->Error) );
		return;
	}
	else{
		if(DGifSlurp(Gif)==GIF_OK)   printf("Se leyó correctamente la imagen\n\n");
		else{
			printf("Se generó el siguiente error de apertura:
				%s\n",GifErrorString(Gif->Error) );
			return;
		}
	}

	Imagen_Gif = Gif->SavedImages;

 /**************Se muestran datos de la imagen*************************/


	printf("valor de los pixeles \n");

	for(i=0 ; i < 100 ; i++){
		printf(" %x ", Gif->SavedImages->RasterBits[i]);
	}

	printf("\n\n\n");

	DGifCloseFile(Gif, error);

	return 0;
}
```

ENTREGABLES

Para la entrega del proyecto se entregan los siguientes archivos:

- Reporte Final del proyecto: Describe el desarrollo del mismo.
- Manual del programador: Se describe dentro del desarrollo del proyecto de éste documento (Capítulo 6).
- Manual de usuario: Describe los requisitos del sistema, la instalación y el uso del programa desarrollado.
- Código fuente: Se muestra en los apéndices de éste documento, o bien en un paquete comprimido.

UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD AZCAPOTZALCO

División de Ciencias Básicas e Ingeniería
Ingeniería en Computación

MANUAL DE USUARIO DEL PROGRAMA: ESTEGO V1.0

"Transmisión de archivos de texto cifrados usando esteganografía en imágenes GIF"

PROYECTO TECNOLÓGICO

Ángel Pérez García
204203880

Trimestre: 14O
Enero del 2015

ACERCA DE ESTE MANUAL

El presente manual describe cómo utilizar el programa llamado estego.

Es un programa diseñado en lenguaje C para Sistemas Operativo Linux, desarrollado como parte del proyecto de integración denominado: “Transmisión de archivos de texto cifrados, usando esteganografía en imágenes GIF”

Con el programa se puede:

- Cifrar un archivo de texto.
- Ocultar un archivo de texto (cifrado) dentro de una imagen GIF.

REQUISITOS DEL SISTEMA

- Sistema Operativo Linux.
- Compilador gcc.
- Librerías GIFLIB, aunque se incluyen dentro del código fuente del programa.

COMPILACIÓN

1. Desempaquetar el código fuente en cualquier directorio de su sistema de archivos.
2. Ejecutar en Terminal el archivo Makefile, es el que se encarga de compilar todos los códigos fuente.
3. Si no se genera algún error, el nombre del programa es ***estego.*** Listo para usarse.

EJECUCION

El programa tiene 5 formas distintas de ejecutarse a través de la línea de comandos y se describen a continuación, puede consultar la información desde el programa escribiendo ***./estego --h*** , o bien consultar el archivo ayuda.txt.

SINTAXIS

./estego -e [archivo de texto] [imagen gif] [clave]
./estego -e [archivo de texto] [imagen gif] [clave] -o [salida_imagen]
./estego -d [imagen gif] [clave]
./estego -d [imagen gif] [clave] -o [salida_texto]
./estego --h

-e
Se utiliza para señalar que se quiere ocultar el archivo de texto en la imagen gif.

-d
Se utiliza para señalarle al programa que se quiere obtener el archivo de texto de una imagen GIF.

-o
Le indica al programa que el usuario colocará el nombre del archivo de salida, ya sea un archivo de imagen o un archivo de texto. En cualquiera de los casos no se debe omitir la extensión *.gif ó *.txt según sea el caso para no generar errores.

Si no se coloca el parámetro -o, entonces el programa coloca los archivos dentro del directorio "salida" con un nombre predeterminado; para la imagen GIF de salida se le asigna el nombre de "output.gif", para el archivo de texto se le asigna "output.txt".

--h
Muestra la ayuda del programa.

ARCHIVO DE TEXTO
Nombre del archivo de texto a ser cifrado, el programa determinará si existe espacio suficiente para almacenar la información dentro del GIF.

IMAGEN GIF
Nombre de la imagen gif de entrada o de salida, debe cumplir las siguientes características, para no generar errores:

- Debe de estar en escala de grises y debe de tener una paleta de 256 colores (grises), en caso contrario se pueden obtener resultados no deseados, por lo que el programa finalizará la ejecución.
- El tamaño máximo de la imagen es de 1024 x 768 pixeles.
- Deben ser imágenes con un solo GIF.

El programa sólo trabaja con imágenes GIF, se pueden descargar las imágenes GIF desde internet, o bien poder realizarlas desde un editor gráfico, como GIMP.

CLAVE
El programa utiliza una clave pública, por lo que la clave para cifrar el archivo es la misma para descifrar. La clave debe tener una longitud de 8 caracteres, se recomienda utilizar letras mayúsculas o minúsculas así como números, no se recomienda utilizar caracteres no imprimibles.

salida_imagen:
Es el nombre de la imagen que contiene el archivo de texto, debe colocarse la extensión .gif.

salida_texto:
Es el nombre que el usuario selecciona para el archivo de texto que se encuentra dentro de la imagen gif, debe llevar la extensión .txt.

ERRORES

El programa puede detectar los siguientes errores en los cuales se finalizará la ejecución.

- No se encuentran los archivos a ser utilizados.
- Clave incorrecta.
- Parámetros incorrectos.
- Pocos pixeles para almacenar la información
- Extensiones de los archivos incorrectas.
- Tamaño de la imagen demasiado grande.
- Que la imagen no tenga paleta de 256 colores en escala de grises
- Que la imagen contenga más de un GIF.