ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UNA HERRAMIENTA DE USO DIDÁCTICO QUE PERMITA ESCONDER INFORMACIÓN ENCRIPTADA DENTRO DE UNA IMAGEN CON FORMATO JPEG EMPLEANDO ESTEGANOGRAFÍA.

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

REAL FALCONÍ FREDDY ANDRÉS
freddy.real@epn.edu.ec

DIRECTOR: ING. WILLAMS FERNANDO FLORES CIFUENTES, MSc.
fernando.flores@epn.edu.ec

Quito, Mayo 2019

AVAL

Certifico que el presente trabajo fue desarrollado por Freddy Andrés Real Falconí, bajo mi supervisión.

ING. WILLAMS FERNANDO FLORES CIFUENTES MSc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Freddy Andrés Real Falconí, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

FREDDY ANDRÉS REAL FALCONÍ

DEDICATORIA

Quiero dedicar el presente proyecto a Dios por todas las bendiciones que derramó en mi camino, a mis padres y hermana que siempre estuvieron para apoyarme, aconsejarme y guiarme en todo momento de mi carrera.

AGRADECIMIENTO

Agradezco a Dios por darme la valentía y fuerza para culminar esta importante etapa de mi vida.

A mi familia y enamorada les agradezco todo su apoyo en cada momento de mi vida porque siempre han sido el motor que me impulsa a ser mejor y cumplir todas las metas que me proponga.

Agradezco a mi tutor de tesis Ing. Fernando Flores quien siempre estuvo dispuesto a ayudarme y compartir todo el conocimiento y experiencia para que pueda finalizar el presente proyecto.

Freddy Andrés Real Falconí

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

El presente Trabajo de Titulación se centra en el desarrollo de una aplicación didáctica para el uso en la materia de Seguridad en Redes de la Carrera de Ingeniería en Electrónica y Redes de Información, dicha aplicación podrá esconder información encriptada, con el algoritmo AES (*Advanced Encryption Standard*) con una clave de 128 bits, mediante técnicas esteganográficas implementadas con el algoritmo LSB (*Least Significant Bit*).

En el primer capítulo, se presenta una perspectiva general de los fundamentos teóricos orientados a la aplicación propuesta. Se tratan algunos temas fundamentales relacionados con esteganografía, criptografía, algoritmo LSB y AES enfatizando en la clave de 128 bits, estándar JPEG, PNG y funcionalidades básicas de *Matlab* junto a su entorno de programación visual *GUIDE*.

En el segundo capítulo, se presenta un análisis del proceso seguido para diseñar, desarrollar e implementar los componentes que son requeridos para la aplicación didáctica. En primer lugar, con el uso de la metodología ágil *Scrum*, se establece los lineamientos para el algoritmo esteganográfico y criptográfico. A continuación, se obtienen algunos requisitos funcionales y no funcionales. Teniendo en cuenta los requisitos se elaboran los esquemas de los módulos con sus respectivas funcionalidades y la creación del visualizador de imágenes JPEG *lossless* mediante la implementación de *sprints* y finalmente un breve bosquejo de la visualización de las vistas de la aplicación didáctica, programación de los algoritmos LSB y AES con clave de 128 bits, codificación de los botones a usar, así como la modificación de las imágenes como cambio de zoom o brillo.

En el tercer capítulo, se presentan los resultados y las pruebas realizadas sobre la aplicación didáctica y cada uno de sus módulos, así como la robustez del algoritmo mediante la variación de zoom o brillo en la estegoimagen y el uso del visualizador creado.

Finalmente, en el cuarto capítulo, se presentan las conclusiones y recomendaciones basadas en todo el proceso llevado a cabo para obtener la aplicación didáctica.

PALABRAS CLAVE: Esteganografía, criptografía, LSB, AES

ABSTRACT

The present work focuses on the development of a didactic application for use in the subject of Network Security of the Engineering Career in Electronics and Information Networks, this application may hide encrypted information, with the AES algorithm (*Advanced Encryption Standard*) with a 128 bits key, using steganographic techniques implemented with the LSB (*Least Significant Bit*) algorithm.

In the first chapter, a general perspective of the theoretical foundations oriented to the proposed application is presented. Some fundamental topics related to steganography, cryptography, LSB and AES algorithm are discussed, emphasizing the 128 bits key, JPEG standard and basic *Matlab* functionalities together with its *GUIDE* visual programming environment.

In the second chapter, an analysis of the process followed to design, develop and implement the components that are required for the didactic application is presented. In the first place, with the use of the *Scrum* agile methodology, the guidelines for the steganographic and cryptographic algorithm are established. Next, some functional and non-functional requirements are obtained. Taking into account the requirements, the diagrams of the modules are elaborated with their respective functionalities and the creation of the *lossless* JPEG image viewer through the implementation of *sprints* and finally a brief *sketch* of the visualization of the views of the didactic application, programming of the algorithms LSB and AES with a 128-bit key, coding of the *buttons* to use, as well as the modification of images such as change of zoom or brightness.

In the third chapter, the results and the tests carried out on the didactic application and each one of its modules are presented, as well as the robustness of the algorithm by means of the variation of zoom or brightness in the estegoimagen and the use of the created visualizer.

Finally, in the fourth chapter, the conclusions and recommendations based on the whole process carried out to obtain the didactic application are presented.

KEY WORDS: Steganography, cryptography, LSB, AES

1. INTRODUCCIÓN

El objetico de este trabajo de titulación es desarrollar una aplicación didáctica para encriptar información mediante el uso del algoritmo AES con una llave de 128 bits y ocultar dicha información con el algoritmo LSB dentro de imágenes. Este prototipo cuenta con una interfaz que le permite al usuario escoger la acción que desea realizar dentro de la aplicación esta puede ser: cifrar un mensaje introducido por teclado en un cuadro de texto y ocultarlo dentro de una imagen o cifrar un archivo de texto .txt y ocultarlo dentro de una imagen para ambos casos se ocultara la información con el algoritmo LSB. Sin embargo, para métodos didácticos se ha colocado un *pop-up* menú que permitirá al usuario escoger en que bit desea ocultar la información de esta manera se podrán notar cambios más visibles cuando la información este sobrescribiendo los bits más significativos.

La aplicación también permite visualizar y comparar mediante un histograma a la imagen original con la estegoimagen, a su vez se puede modificar las imágenes en brillo o en zoom teniendo en cuenta que el mensaje oculto seguirá presente dentro de la imagen.

A parte de ello, la aplicación tiene un descifrador que permite realizar los procesos inversos para recuperar la información que se encuentre dentro de una imagen, dicha información podrá ser un mensaje ingresado por teclado o un archivo de texto .txt para lo cual se necesitará saber la contraseña con la que se encriptaron y el bit donde se ocultaron.

Además, la aplicación cuenta con un visualizador de imágenes que admite la visualización de las imágenes JPEG *lossless* ya que este formato se lo realiza sin perdidas se necesita de una aplicación específica para la apertura de dichas imágenes, sin embargo, la aplicación también entrega la imagen en formato PNG para la apertura en otros programas de visualización de imágenes.

Este visualizador permitirá abrir dos imágenes simultáneamente una alterada y la otra original y calcular su transformada discreta para apreciar cambios existentes en la imagen que no se pueda ver a simple vista, también será capaz de realizar una resta entre las dos imágenes para apreciar los pixeles que han cambiado entre ambas imágenes.

Para el desarrollo de la aplicación didáctica se utilizó *Matlab* con la versión 2018b instalada todos sus componentes para que no exista ningún error en la ejecución de funciones propietarias de *Matlab*.

1.1 Objetivos

El objetivo general de este proyecto es: Desarrollar una herramienta de uso didáctico que permita esconder información encriptada dentro de una imagen con formato JPEG empleando esteganografía.

Los objetivos específicos de este proyecto son:

- Analizar los conceptos de los algoritmos esteganográficos a programar, *Least Significant Bit* (LSB) y *Advanced Encryption Standard* (AES)
- Diseñar los módulos y submódulos de la herramienta de uso didáctico.
- Implementar la aplicación y los algoritmos diseñados para las técnicas esteganográficas en imágenes JPEG.
- Analizar los resultados finales de la herramienta de uso didáctico.

1.2 Alcance

La solución a desarrollarse tiene una aplicación demostrativa y didáctica, esta se programará en *Matlab* con un entorno amigable para el usuario, que puede esconder información dentro de una imagen JPEG en los bits menos significativos mediante la programación de los algoritmos LSB (para ocultamiento de la información) y AES (para la encriptación). La aplicación se encargará de recibir el mensaje, encriptarlo y realizar las estrategias esteganográficas para el ocultamiento de la información dentro de la imagen. En el lado del receptor de la información, se tendrá un programa que realice las funciones inversas y entregue la información original.

El programa consta de un módulo principal que será un menú que permitirá elegir mediante botones los módulos secundarios:

- Cifrar texto en imagen
- Cifrar archivos de texto en imagen
- Descifrar

A su vez dentro de los módulos secundarios: Cifrar texto en imagen y Cifrar archivos de texto en imagen se tendrá un módulo ternario que se activará con el botón Visualizar.

Para los dos primeros módulos secundarios se deberá elegir una imagen que esté en formato JPEG de cualquier tamaño y resolución, dependiendo del módulo secundario que se elija permitirá: encriptar y ocultar un mensaje ingresado por teclado o un archivo de texto con extensión .txt por una interfaz. El tamaño del archivo o el mensaje dependerá del tamaño de la imagen introducida, para esto la herramienta analizará el peso de la imagen y calculará un aproximado de bits que se puede ingresar para no distorsionarla. Se tendrá un botón cifrar que cifrará el mensaje, realizará las estrategias esteganográficas para ocultar el mensaje o archivo y creará la nueva imagen con la información oculta que se llamará: ImagenConMensaje o ImagenConMensajeDeTexto dependiendo el módulo, para esto se programará en los *scripts* de *Matlab* el algoritmo que permitirá encriptar la información con una llave de 128 bits en formato decimal, el algoritmo que vacíe los bits menos significativos de la imagen e introduzca la información encriptada en dichos bits.

Mediante el botón Visualizar se abrirá un nuevo módulo ternario, que permitirá ver las dos imágenes: la original y la alterada para ver las diferencias que existen, también se puede realizar cualquier cambio de zoom o brillo para mostrar la robustez de los algoritmos programados, estos cambios serán programados en un *script* que serán controlados por un *slider*. Las nuevas imágenes con los cambios de zoom o brillo se guardarán mediante un botón Guardar que creará una nueva imagen con el nombre ImagenCifradaVisualizada. Así mismo se podrá visualizar el histograma de la imagen con un botón Histograma de la imagen para ver las diferencias a más profundidad que será programado en dicho botón.

En el tercer módulo secundario se tiene la opción, descifrar, que permitirá recuperar el mensaje ingresado o a su vez el archivo de texto de acuerdo el botón que se elija. El archivo de texto se creará con el nombre ArchivoDeTextoDecifrado en el que estará la información original y se colocará un título que diga == Mensaje Descifrado A Continuación == y para el mensaje oculto aparecerá en un cuadro de texto una vez escogida la imagen, para esto se programará un *script* que permitirá realizar las funciones inversas, es decir leerá los bits menos significativos de la imagen y después desencriptará la información.

Existirá un producto final demostrable.

En la Figura 1.1 se muestra en bloques el proceso para introducir y extraer la información, todo este proceso será programado en *scripts* de *Matlab* y a su vez

funcionará con una interfaz gráfica. De igual manera se detallará las funciones a usar y una breve explicación del código programado.

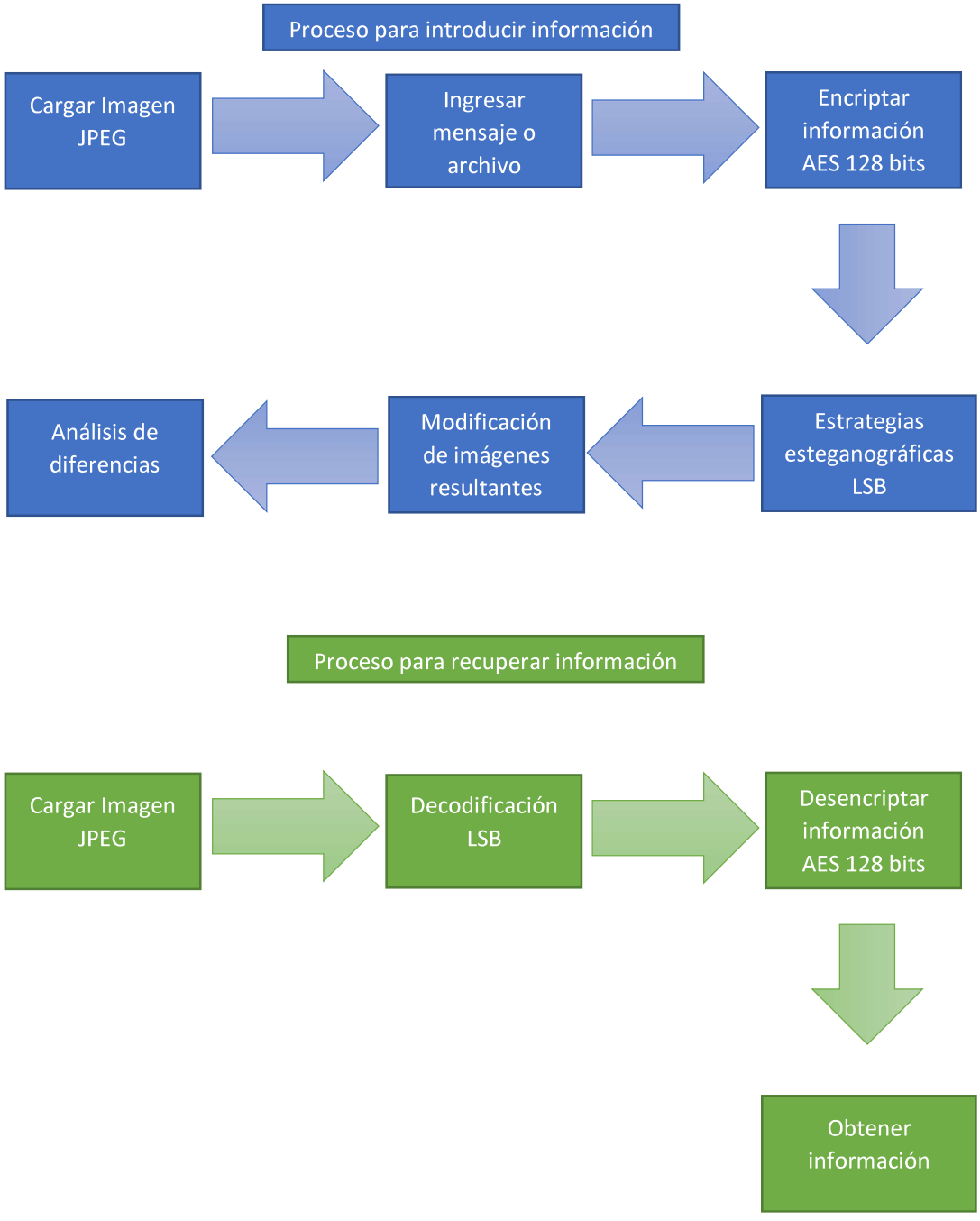

Figura 1.1. Módulos de proceso para introducir y recuperar información

1.3 Marco teórico

1.3.1 Esteganografía

Es una herramienta que permite ocultar información en un archivo multimedia, sea este una imagen, audio, video, etc., ayuda a prevenir la detección de un mensaje oculto, esta información pasa inadvertida para terceros ya que no se puede ver la diferencia del archivo original con el editado, solamente el usuario legítimo va a poder recuperar la información mediante la extracción de esta. En la Figura 1.2 se puede observar el proceso que sigue la Esteganografía [1].

Actualmente es una herramienta muy utilizada no solo para propósitos legales sino también para el terrorismo o ataques cibernéticos, ya que además de ocultar información se puede camuflar algún tipo de ejecutable que robe datos o afecte la integridad de un dispositivo. Las instituciones que emplean la esteganografía, en su mayoría, son: militares, policía, agentes de inteligencia, instituciones criminales, etc. [2].

Sin embargo, la esteganografía no es algo actual ya que las primeras técnicas esteganográficas aplicadas datan del siglo XV en la Grecia Antigua en donde las guerras eran muy comunes sean por territorios o religiones, en esa época los medios no eran digitales por lo que empleaban otras técnicas para el uso de la esteganografía ya que era de vital importancia que puedan enviar un mensaje que no pueda ser leído en caso de exposición, gracias a esto nace el concepto de esteganografía clásica.

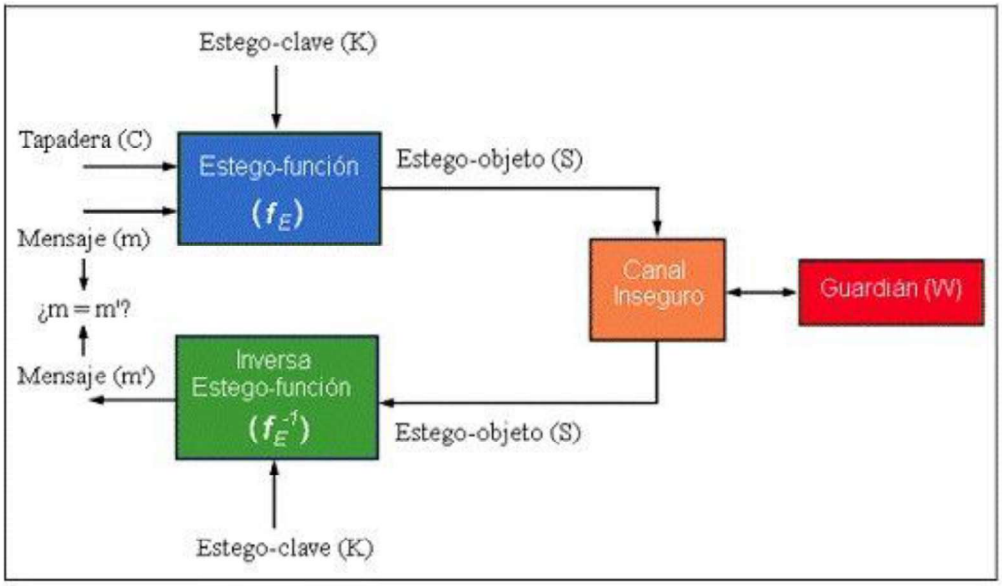

Figura 1.2. Fases de la esteganografía [2]

1.3.1.1 Esteganografía clásica

Es un conjunto de técnicas para ocultar, que se mantienen en secreto, que permiten esconder o camuflar un mensaje mediante un canal que puede o no ser seguro.
El método usado por Demarto consistía en escribir en una tabla el mensaje y cubrirlo con cera hasta que el mensaje desaparezca en su totalidad, una vez que llegue la tabla al receptor estos quitaban la cera y podían ver el mensaje sin problema, ya que el mensaje pasaba por muchas estaciones de control nadie pudo verlo, Grecia se salvó de ser conquistada por Persia mediante la utilización de esta técnica [2].

1.3.1.2 Esteganografía pura

Este tipo de esteganografía se basa en los algoritmos de ocultación y extracción de la información que solo el emisor y receptor conocen, este método es seguro siempre y cuando los usuarios que tengan acceso a este mensaje sean inexpertos en técnicas esteganográficas, sin embargo, si los usuarios que tienen acceso al mensaje tienen un alto conocimiento en técnicas esteganográficas es muy fácil de vulnerar y encontrar el mensaje oculto.

1.3.1.3 Esteganografía de clave privada

Esta técnica es una combinación entre esteganografía pura con criptosistemas simétricos, es decir, el atacante podría descubrir el algoritmo de ocultamiento y extracción del mensaje por lo cual el mensaje se cifra con métodos simétricos antes de ocultarlo, de esta manera si el atacante logra conseguir el mensaje, este estará cifrado brindándole más seguridad al emisor de este. Este método se basa en una estego-clave que es la clave para desencriptar la información una vez que se extrajo el mensaje, esta clave tiene que ser socializada entre el emisor y el receptor para que se pueda acceder a la información plana.

1.3.1.4 Esteganografía de clave pública

Este método usa dos claves sin embargo no se requiere de la socialización de las dos claves ya que la primera es secreta y es usada para encriptar la información que se va a ocultar mientras que la segunda clave es pública, esta se guarda en una base de datos pública y se usa para reconstruir el mensaje una vez que se encuentra en el receptor [3].

1.3.2 Criptología

Es la ciencia que trata los problemas teóricos relacionados con la seguridad en el intercambio de mensajes en clave entre un emisor y un receptor a través de un canal de comunicaciones.

Esta ciencia se divide en dos ramas:

1.3.2.1 Criptografía

Es una ciencia que utiliza algoritmos para convertir la información en criptogramas de tal manera que el único que pueda descifrar y entender la información sea el destinatario, para esto la criptografía emplea el uso de claves, de esta forma el emisor puede convertir los datos en criptogramas y el receptor con el uso de la clave puede realizar el proceso inverso [4].

1.3.2.2 Criptoanálisis

Es la ciencia encargada de descifrar los mensajes en clave, en otras palabras, lo contrario a la criptografía.

1.3.3 Criptografía antigua

Esta ciencia fue impulsada por guerras, religiones y comerciantes ya que todos requerían de un sistema que cifre su información para prevenir que otras personas puedan leer la información que enviaban. Se tiene como ejemplo los sacerdotes egipcios que utilizaban la escritura hierática (jeroglífica) como se puede observar en la Figura 1.3, mientras que los antiguos babilonios utilizaban la escritura cuneiforme [5].

Figura 1.3. Criptogramas Egipcios [5]

Se tiene dos tipos de criptografía de acuerdo a la clave empleada:

1.3.4 Criptografía simétrica

Los sistemas de cifrado simétrico son los que utilizan una misma clave para cifrar y descifrar el documento, por lo cual se necesita que el emisor y el receptor conozcan la clave. Se puede ver el proceso en la Figura 1.4. La fortaleza de estos algoritmos dependen de la clave generada ya que los algoritmos son de conocimiento público, lo único que separa del texto cifrado al texto plano es la clave, por lo que debe tener una

complejidad muy alta que prevenga de ataques de fuerza bruta, existen varios algoritmos simétricos con diferentes tamaños de clave como el algoritmo *Data Encryption Standard* (DES) que usa una clave de 56 bits; Triple DES que usa claves de 56, 112 o 168 bits; *Advanced Encryption Standard* con claves de 128, 196 y 256 bits.

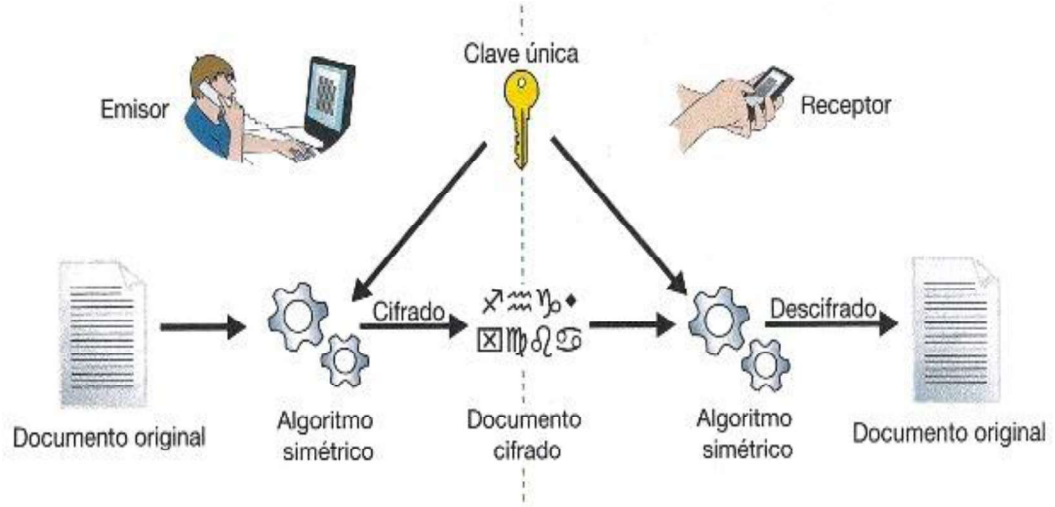

Figura 1.4. Funcionamiento de la criptografía simétrica [6]

1.3.5 Criptografía asimétrica

Estos sistemas se basan en dos claves una secreta y una pública como se observa en la Figura 1.5, el emisor codifica con la clave secreta mientras que el receptor debe desencriptar el mensaje con la clave pública.

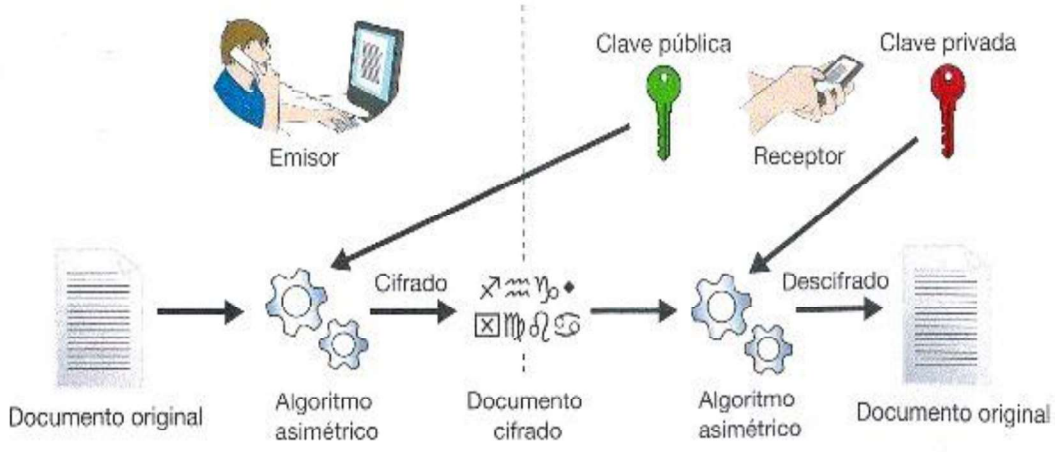

Figura 1.5. Funcionamiento de la criptografía asimétrica [6]

1.3.6 *Advanced Encryption Standard*

Para la aplicación didáctica que se va a realizar se empleará el algoritmo de criptografía simétrica AES con clave de 128 bits.

Es un algoritmo conocido como AES, o también llamado Rijndael, por sus autores Vincent Rijmen y Joan Daemen, usa cifrado simétrico, se transformó en un estándar en el año 2002 y es uno de los algoritmos más usados en la actualidad. Su característica principal es que los datos a encriptar los divide en bloques de tamaño fijo de 128 bits y cada bloque se representa como una matriz de 4x4 bytes llamado estado como se observa en la Figura 1.6.

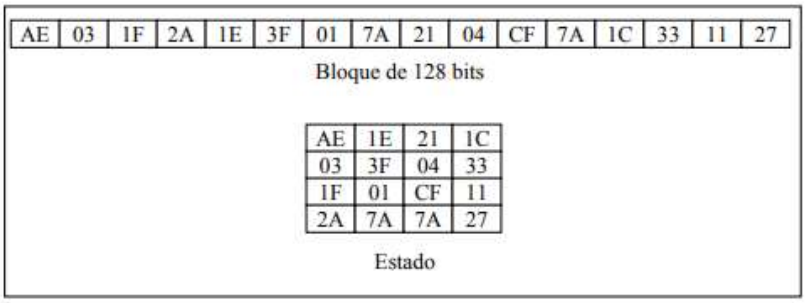

Figura 1.6. Estado AES [7]

A cada estado se le aplican once rondas que están compuestas por algunas operaciones. Las operaciones con bytes en AES como suma y multiplicación son cálculos en campos de Galois GF (2^8) o campos finitos con 8 bits.

Hay tres transformaciones distintas llamadas capas en las que se tratan los bits. Estas constan de:

- **Capa de Mezcla Lineal:** busca la difusión de los bits.
- **Capa No Lineal:** se trata de una zona que emplea cajas S.
- **Capa Clave:** operaciones con XOR de la subclave y la información de esta etapa intermedia.

La Caja S o Caja de Sustitución es un elemento fundamental empleado en algoritmos de clave simétrica los cuales realizan una sustitución, son típicamente usados para obscurecer la relación entre la clave y el texto cifrado las operaciones como se puede observar en la Figura 1.7.

Las transformaciones realizadas en cada paso del algoritmo se denominan estados. Estos estados se representan por una matriz de 4 filas y Nb = 4 columnas para el texto en claro y 4 filas y Nk = 4, 6 u 8 columnas para las claves [7].

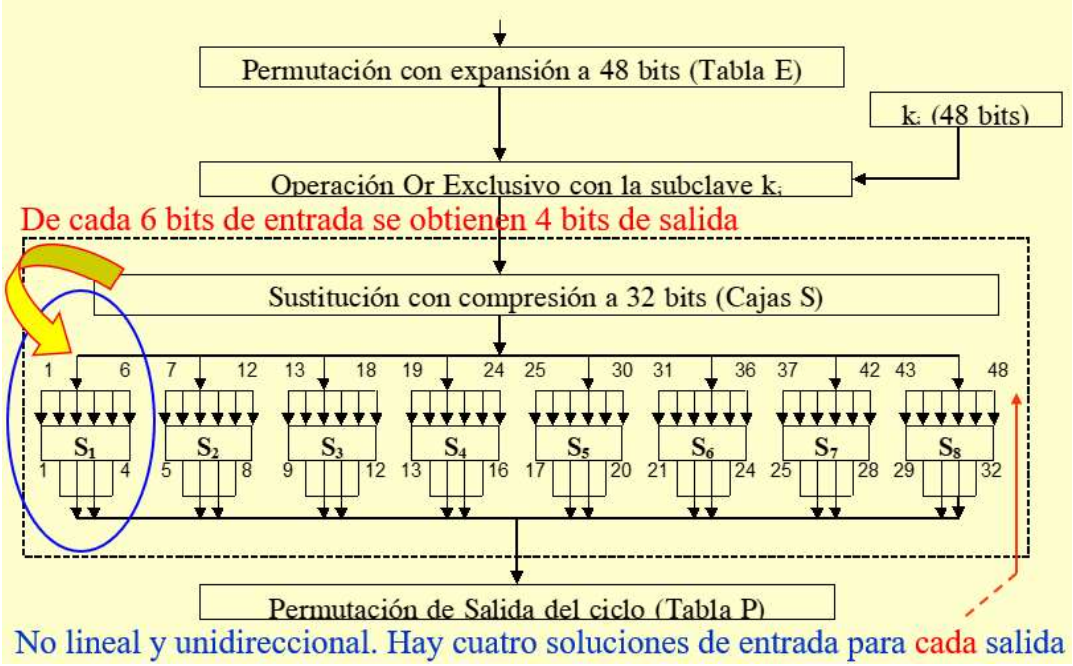

Figura 1.7. Operación de la caja S [7]

Estas once rondas se clasifican en tres tipos:

- 1 ronda Inicial
- 9 rondas estándar
- 1 ronda final

Ya que AES es un algoritmo simétrico, utiliza una única clave para cifrar y descifrar, esta puede ser de 128, 196 y 256 bits. Sin embargo, el estándar autoriza el uso de 128 bits. Esta clave es denominada estego-clave o clave inicial a partir de esta se generan diez claves más mediante un proceso matemático, estas diez claves conjuntamente con la inicial se denominan subclave y cada una se utiliza en una ronda diferente. Se debe tener en cuenta que las claves son de una vía por lo tanto no se puede realizar una comparación entre la clave usada para cifrar con la clave usada para descifrar por lo que el algoritmo realizará sus funciones a pesar de que la clave sea errónea pero el resultado no será el mensaje originalmente encriptado.

1.3.6.1 Ronda inicial

Denominada AddRoundKey, se realiza una operación XOR byte a byte entre el estado y la clave inicial.

1.3.6.2 Ronda estándar

Se aplican 4 operaciones:

- SubBytes: se procede a realizar el reemplazo de cada byte del estado por otro con una tabla de sustitución de bytes con valores predeterminados. El tamaño de la tabla es de 16x16 bytes y para obtener los datos como índice de fila los primeros 4 bits del byte a reemplazar y como índice de columna los 4 últimos bits.
- ShiftRows: La primera fila del estado no se modifica, los bytes de las filas restantes se rotan cíclicamente a la izquierda; una vez en la segunda fila, dos veces en la tercera y tres veces en la cuarta.
- MixColumns: a cada columna del estado se le aplica una transformación lineal y es reemplazada por el resultado de esta operación.
- AddRoundKey: igual a la operación inicial, pero con la siguiente subclave.

1.3.6.3 Ronda final

Se aplica 3 operaciones:

- SubBytes: misma forma que en la ronda estándar.
- ShiftRows: misma forma que en la ronda estándar.
- AddRoundKey: al igual que las rondas anteriores, pero con la siguiente subclave [8].

En la Figura 1.8 se puede observar las diferentes rondas empleadas por AES.

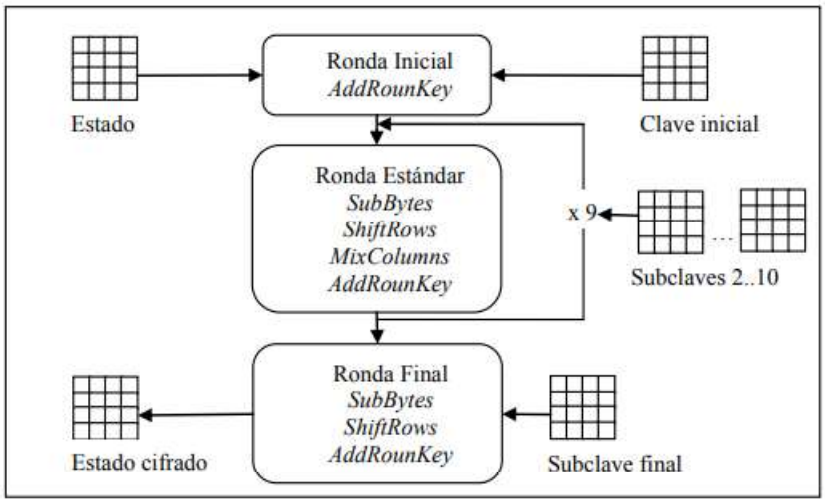

Figura 1.8. Rondas del algoritmo AES sobre un estado [8]

1.3.7 Adversario

Son todos los entes a los que se trata de ocultar la información encubierta, pueden ser pasivos o activos.

1.3.7.1 Adversario pasivo

Sospecha que se puede estar realizando una comunicación encubierta, intenta descubrir el algoritmo para extraer la información oculta, pero no la modifica.

1.3.7.2 Adversario activo

Son entes que intentan descubrir el algoritmo para extraer la información oculta del mensaje y lo modifican para corromper la información del mensaje.

Debido a los diferentes tipos de adversarios que se tiene (pasivos o activos), las técnicas que se apliquen deben ser robustas, es decir, tienen que soportar cambios o distorsiones accidentales o provocadas por un tercero (adversario activo), para lo cual se recurre a la aplicación de esteganografía y criptografía simultáneamente, de esta manera, aunque se descubra el patrón estenográfico, no sabrán el mensaje ya que se encuentra cifrado [9].

1.3.8 Métodos esteganográficos

Existen varios métodos esteganográficos que se pueden aplicar para que no se perciba un aumento excesivo en el tamaño del archivo, como, por ejemplo:

1.3.8.1 Algoritmo F5

Es un algoritmo que permite aplicar esteganografía sobre imágenes con formato JPEG usando LSB antes de la compresión de los datos.

1.3.8.2 Algoritmo LSB

Transforma la imagen en un mapa de bits y se va modificando el ultimo bit de cada byte (el menos significativo) de tal forma que permite ocultar hasta un 12,5% de la longitud del mapa de bits [10].

Este algoritmo también puede ser usado para reemplazar dos bits de un mismo byte sin embargo la alteración es más notoria dentro de la imagen. Se puede aplicar la misma técnica del algoritmo para reemplazar otros bits de cada byte con el objetivo de ver las diferencias que se produce dependiendo el bit que es reemplazado.

Algunos algoritmos LSB enceran todos los bits menos significativos y después hacen el reemplazo de cada uno, pero esto ocasiona que se produzcan cambios más notorios en los histogramas ya que esto produce que haya mayor cantidad de bits pares por lo cual el atacante podría sospechar que existe un cambio en dicha imagen.

En la Figura 1.9 se puede observar un ejemplo del algoritmo LSB en los dos últimos bits.

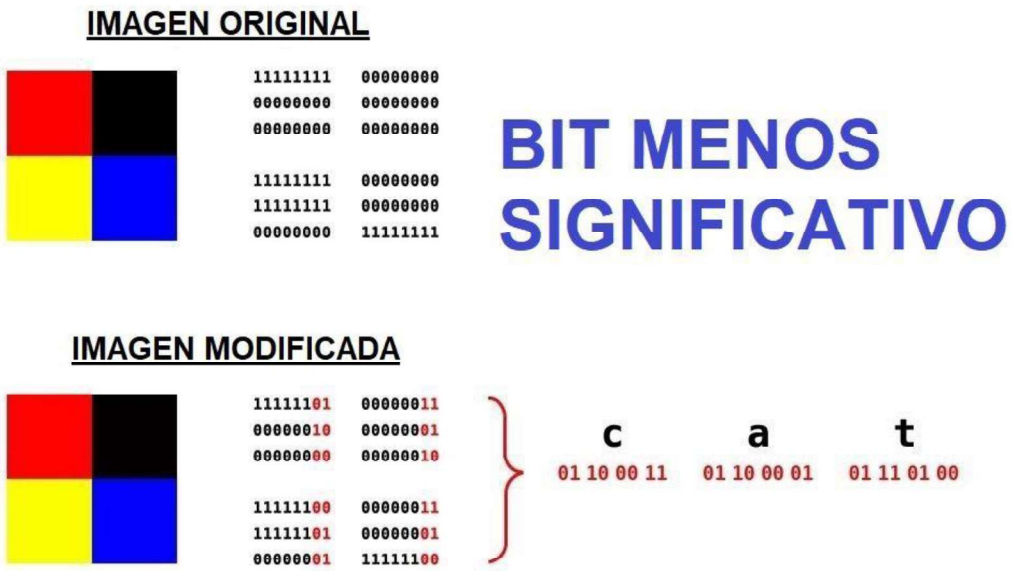

Figura 1.9. Ejemplo de uso del algoritmo LSB usando los dos últimos bits [10]

1.3.9 Análisis

Para la realización de la aplicación didáctica se hará uso del algoritmo LSB ya que es el método moderno más común usado en esteganografía; como se mencionó anteriormente, este algoritmo es muy útil para imágenes, video y audio. Para su funcionamiento en imágenes convierte el archivo multimedia en mapa de bits de tal manera que se modifica el bit menos significativo de cada byte y se coloca la información, con esto se logra que el ojo humano no pueda percibir las diferencias entre la imagen original y la estego-imagen, sin embargo ya que solo se puede usar 1 bit por cada 8 bits si el archivo o mensaje a ocultar es demasiado grande se podría usar 2 bits de cada 8 bits pero no es recomendable ya que en esos casos si se puede percibir diferencias entre las imágenes.

Si se toma un ejemplo de un mapa de bits como se muestra en la Figura 1.10.

(01011010) (11010010) (00011011)
(01101010) (00011011) (00011111)
(01011110) (01000101) (10110101)

Figura 1.10. Ejemplo de uso del algoritmo LSB usando un solo bit (parte 1)

Para introducir la letra A en representación ASCCI (10010111) usando el algoritmo LSB se reemplazaría el ultimo bit de cada byte por lo tanto se necesitaría de 3 de cada 24 bits para realizar la inserción del carácter:

(01011011) (11010010) (00011010)
(01101011) (00011010) (00011111)
(01011111) (01000101) (10110101)

Figura 1.10. Ejemplo de uso del algoritmo LSB usando un solo bit (parte 2)

Con este ejemplo se puede corroborar que el uso normal del algoritmo LSB es del 12.5% del total de bits que se ha usado, lo cual pasa desapercibido en los controles para la detección.

Adicionalmente si se usa una representación diferente como CMYK (Cyan, Magenta, Yellow y Key) se tendría un desperdicio de bits mucho menor ya que solo se necesitaría 3 pixeles para ocultar el carácter debido a que RGB tiene 24 bits por pixel mientras que CMYK 32 bits por pixel.

1.3.10 JPEG

Joint Photographic Experts Group, es un estándar creado por un grupo de expertos para la compresión y codificación de imágenes fijas de tonos continuos en escala a grises o tonos continuos. Para satisfacer las diferentes necesidades de muchas aplicaciones, el estándar JPEG incluye dos métodos básicos de compresión, cada uno con varios modos de operación. Se especifica un método basado en DCT (transformada de coseno discreta) para la compresión con pérdida, y un método predictivo para la compresión sin pérdida.

JPEG presenta una técnica sencilla de pérdida conocida como el método de línea de base, un subconjunto de los otros modos de operación basados en DCT. El método Baseline ha sido, con mucho, el método JPEG más implementado hasta la fecha, y es suficiente por sí mismo para un gran número de aplicaciones [11].

1.3.11 Reconstrucción de la imagen JPEG

El formato JPEG tiene una cabecera que da información importante sobre la imagen, algunas de ellas son: el tamaño, tablas de cuantización y codificación Huffman, el factor de submuestreo que se empleó, entre otras características que utilizan los programas encargados de abrir dicha imagen correctamente.

Se denomina marcadores a los códigos especiales que permiten identificar las componentes de la imagen comprimida, estos marcadores tienen un tamaño de 2 bytes, algunos de ellos están seguidos por una serie de parámetros, a esto se le denomina segmento.

Según el Comité Consultivo Internacional Telegráfico y Telefónico CCITT T.81 *Information technology - Digital compression and coding of continuous tone still images Requirements and guidelines* [12] los marcadores que se recomiendan para la identificación de las distintas partes de la imagen se denotan por una secuencia: 0xFF seguidos por un byte diferente a 0x00 y 0xFF. A continuación, en la Figura 1.11 se puede observar los diferentes marcadores usados en la escritura de imágenes JPEG.

Marcador	Símbolo	Descripción
0xFFD8	SI	*Start of Image Marker*
0xFFE0	APP0	*Application Specific Marker* (Es usado por JFIF)
0xFFDB	DQT	*Define Quantization Table Marker*
0xFFC0	SOF	*Start of Frame Marker*
0xFFC4	DHT	*Define Huffman Table Marker*
0xFFDA	SOS	*Start of Scan Marker*
0xFFD9	EOI	*End of Image Marker*

Figura 1.11. Marcadores para imágenes JPEG [13]

En la Figura 1.12 se muestra cómo se conforma una imagen JPEG:

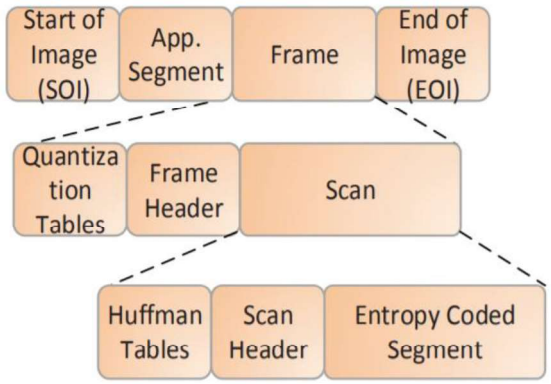

Figura 1.12. Estructura de imágenes JPEG [13]

1.3.11.1 Marcador de inicio de imagen SOI (2 BYTES)

Se encarga de marcar el inicio de la imagen comprimida.

1.3.11.2 Marcador de fin de imagen EOI (2 BYTES)

Se encarga de marcar el final de la imagen comprimida. Se debe tener en cuenta que si uno de estos dos marcadores no se encuentra en el archivo comprimido será considerado inválido y no podrá ser decodificado por ninguna herramienta.

1.3.11.3 Cabecera de datos para el campo aplicación

Este campo denominado aplicación es usado para definir que la imagen cumpla con el estándar de Formato de Intercambio de Archivos JPEG (JFIF). Esta cabecera da el bit stream de JPEG para que este sea soportado por varias plataformas y aplicaciones que puedan abrir este tipo de archivos.

1.3.11.4 Segmento para definición de las tablas de cuantificación

Este segmento es encargado de definir las tablas de cuantificación para las componentes de luminancia y crominancia, este campo solo se usa en los algoritmos de JPEG que usen la Transformada Discreta Del Coseno (DCT); en la Figura 1.13 se puede observar los diferentes campos y su tamaño.

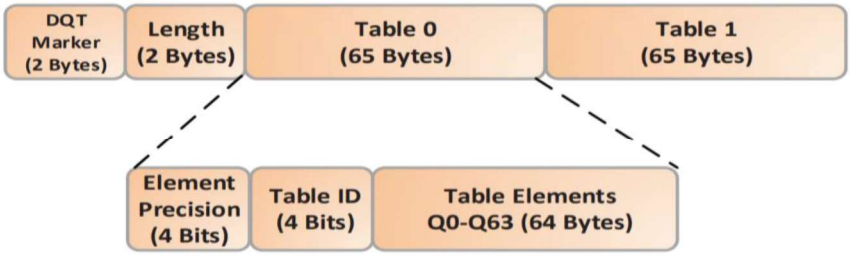

Figura 1.13. Segmento para la definición de las tablas de cuantificación [13]

El marcador DQT[1] tiene una extensión de 2 bytes, define el marcador de la tabla de cuantificación DQT e identifica de manera única el segmento de las tablas de cuantificación.

---

[1] **DQT (The Quantization table segment):** marcador de la tabla de cuantificación.

El campo longitud con una extensión de 2 bytes, define la longitud que tiene el segmento de las tablas de cuantificación incluyendo los 2 bytes del campo longitud, pero no toma en cuenta los 2 bytes del campo DQT.

El campo precisión del elemento tiene una extensión de 4 bits, un valor de 0 define una precisión de 8 bits y un valor de 1 define una precisión de 16 bits, se debe tener en cuenta que para cada tabla que esté definida, esta debe tener la precisión del elemento.

El identificador de la tabla con 4 bits de extensión, proporciona a cada tabla un identificador único que usará el decodificador para aplicar la tabla correcta de cuantificación. El elemento de la tabla de cuantificación tiene los 64 bytes restantes, dentro de este campo se colocan los 64 elementos que se definen por cada tabla [13].

1.3.11.5 Segmento para la definición de la cabecera de trama (SOF)

Este campo es el marcador que determina el inicio de la trama, su valor varía de acuerdo al tipo de implementación JPEG.

1.3.11.6 Segmento de definición de las tablas Huffman (DHT)

Es el marcador encargado de definir el inicio en la cabecera de JPEG del segmento de las tablas de codificación Huffman. Este definirá una tabla Huffman para cada una de las componentes de la imagen.

1.3.11.7 Formato de un segmento de scan

El segmento SCAN, ayuda a definir algunos parámetros como: las tablas de cuantificación o codificación Huffman que van a ser empleadas y el número de componentes que están presentes en el scan.

1.3.11.8 Segmento opcional de inicio de comentario

Este segmento se define cuando se ha incluido un comentario en la cabecera de la imagen, es opcional, sin embargo, es un campo muy utilizado para ocultar la información, el problema más grande de ocultar información aquí es que debe ir con caracteres imprimibles ya que no acepta caracteres no imprimibles como los que arroja AES, por lo que no se va a tener una fuerte seguridad en el ocultamiento del mismo. En el presente proyecto se planteó el uso del comentario para guardar la información, pero como se menciona anteriormente no aceptaba el algoritmo de encriptación AES ni tampoco se podía usar LSB ya que dentro del comentario no se podría reemplazar los

bits menos significativos porque no altera los colores de la imagen, solamente la hace crecer en tamaño. En la Figura 1.14 se puede observar la estructura del segmento comentario.

Figura 1.14. Segmento de comentario [13]

El marcador de comentario (COM) con una extensión de 2 bytes, es el encargado de indicar el inicio de un comentario.

La longitud del segmento del comentario con 2 bytes de extensión, otorga la longitud de los parámetros que se definirán en un comentario sin tomar en cuenta el marcador COM de 2 bytes.

El byte del comentario (CMI) corresponde a cada uno de los símbolos que conforman el comentario, como se mencionó anteriormente solo caracteres imprimibles.

1.3.11.9 Segmentos de datos codificados

Una vez que se realiza la codificación Huffman el resultado son los segmentos de datos codificados, estos siempre son un número entero de bytes.

Para la codificación Huffman siempre se utiliza un bit al final del segmento para que se complete y sea múltiplo de 8 bits, es decir 1 byte entero, en caso de ser necesario.

Para evitar confusiones entre marcadores y un segmento de datos, cualquier byte FF generado en el segmento de datos o por la inserción del bit que sea detectado a continuación se insertará un byte cero de relleno, de esta manera si el decodificador nota un byte cero después de un byte FF lo elimina mientras que si es diferente a cero entonces lo detecta como un marcador que debe ser interpretado.

1.3.11.10 Unidad mínima de codificación (MCU)

Si los datos en una imagen no están entrelazados, el campo MCU tendrá solo una unidad de datos o una sola componente, mientras que si los datos se encuentran entrelazados el campo MCU tendrá una o más unidades de cada componente [13].

1.3.12 PNG

Portable Network Graphics es un formato de imágenes que se basa en un algoritmo de compresión *lossless*, es decir, sin pérdidas, llamado algoritmo deflación que es una combinación del algoritmo LZ77 y la codificación Huffman, gracias a esta compresión se tiene una calidad bastante alta, pero con un peso de la imagen más alto, este formato no está patentado por lo que no necesita una licencia para su utilización [14].

Este formato usa una firma que denota que se trata de PNG, esta firma es colocada al inicio del archivo y en formato decimal: 137 80 78 71 13 10 26 10.

Se debe tener en cuenta que esta firma puede encontrarse en formato hexadecimal de igual manera al principio del archivo.

1.3.13 *Matlab*

Su nombre *Matrix Laboratory*, es una aplicación desarrollada por la empresa *The MathWorks Inc.* que combina un entorno de escritorio adaptado para el análisis iterativo y los procesos de diseño con un lenguaje de programación basado en matrices. Permite realizar *scripts* con un entorno de programación visual *GUIDE* [15].
Matlab también permite la creación de aplicaciones ejecutables, es decir con una extensión .exe mediante programación, estas aplicaciones creadas pueden ser ejecutadas sin necesidad de tener instalado todo el paquete de *Matlab* simplemente se crean instaladores para los *runtime* que necesita la aplicación para lo cual se corre el instalador y posteriormente se ejecuta la aplicación que se desea usar.

1.3.13.1 *GUI*

Las *GUI* (también conocidas como interfaces gráficas de usuario o *GUI*) proporcionan control de las aplicaciones de software al apuntar y hacer clic, eliminando la necesidad de aprender un idioma o comandos de tipo para ejecutar la aplicación. Las aplicaciones con interfaces *GUI* automatizan una tarea o cálculo mediante el uso de herramientas gráficas. La *GUI* generalmente contiene controles como menús, barras de herramientas, botones y controles deslizantes. También se puede crear aplicaciones propias personalizadas, conjuntamente con *GUIDE* (entorno de desarrollo de *GUI*) [16].

En la Figura 1.15 se puede observar un ejemplo de una interfaz gráfica.

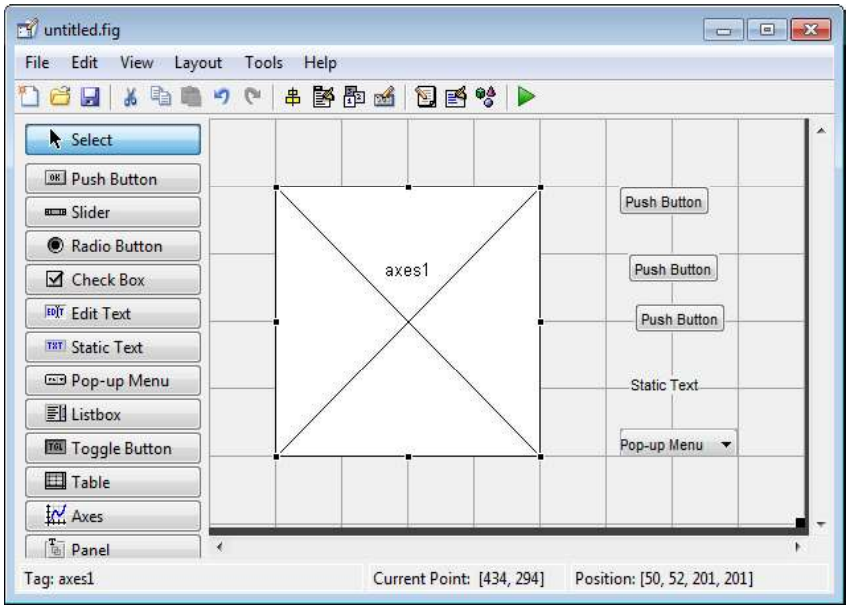

Figura 1.15. Interfaz *GUIDE*

1.3.14 *Scrum*

Es una metodología ágil de desarrollo de software que trabaja a partir de *sprints* (son bloques de tiempo en el cual se desarrolla una tarea) *Scrum* está basado en la realización de proyectos caracterizados por tener un alto grado de incertidumbre lo que provoca que los objetivos puedan ir variando en el desarrollo del proyecto y que existan varios cambios de última hora por lo cual el software se va mejorando mientras se va desarrollando [17].

Scrum tiene tres roles principales como se muestra en la Figura 1.16:

- ***Scrum Master*:** Es el líder del equipo, los guía para un correcto cumplimiento de las reglas, procesos y tiempos establecidos, trabaja directamente con el dueño del producto (*Product Owner*).
- **Dueño del producto:** También llamado *Product Owner*, es el representante de los clientes que solicitan el software, es el encargado de transmitir todos los requisitos para el proyecto.
- **Equipo *Scrum*:** Es el grupo de profesionales que llevan a cabo la realización del proyecto bajo la guía del *Scrum Master*.

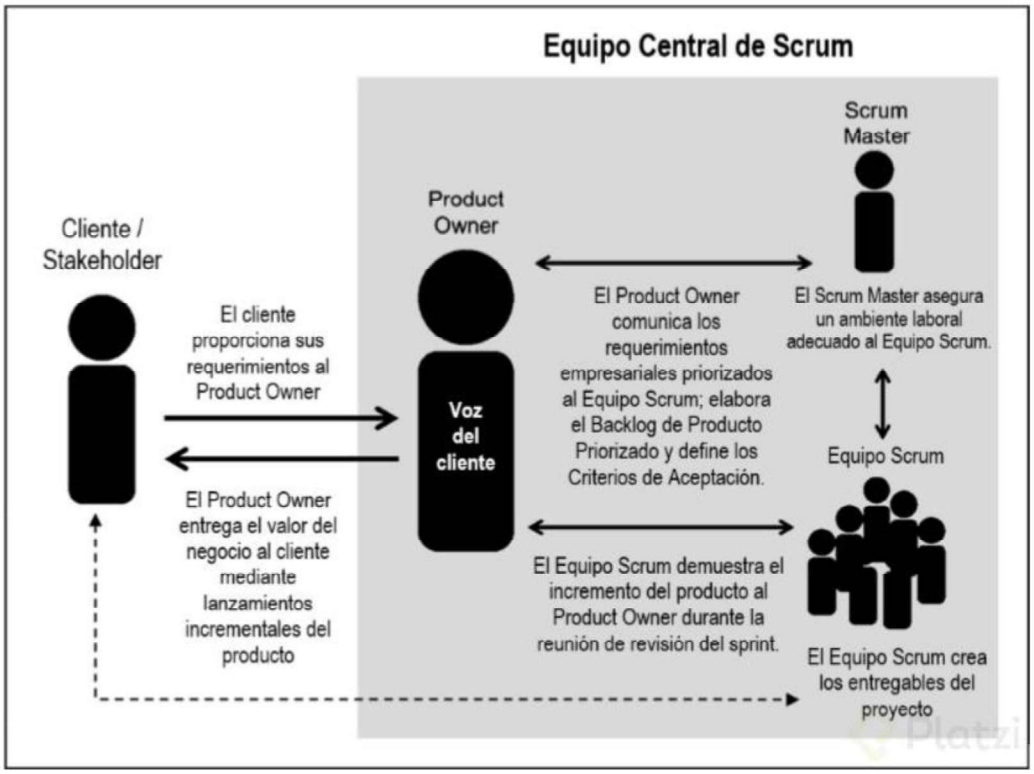

Figura 1.16. Roles de *Scrum* [17]

Scrum está desarrollado en 5 fases que definen el ciclo de desarrollo ágil como se observa en la Figura 1.17:

- **Concepto:** Se coloca en forma general todas las características que tendrá el software a desarrollar y se asignan las tareas al equipo de desarrollo.
- **Especulación:** Esta fase se repite en cada *sprint*, consiste en desarrollar y revisar los requisitos generales del software, tener las funcionalidades solicitadas y establecer las fechas de entregas.
- **Exploración:** Los requisitos del producto en la fase de especulación se incrementan y se detallan.
- **Revisión:** Se revisa los objetivos cumplidos y se compara con el objetivo general del proyecto.
- **Cierre:** Se entrega el proyecto final en la fecha establecida, sin embargo, no significa que ha finalizado el proyecto ya que puede seguir habiendo cambios

que se deben desarrollar, estos cambios serán denominados mantenimientos que se realizarán para que el producto sea el deseado por el cliente.

Figura 1.17. Fases de *Scrum* [17]

Elementos de *Scrum*:

- ***Product Backlog*:** Es un listado de todas las necesidades que tiene el cliente, es entregada al *Scrum Master*.
- ***Sprint Backlog*:** Es el listado de tareas que se realiza en un *sprint*, considerando las fechas especificadas.
- **Incremento:** Es una parte del software culminada y que se encuentra totalmente operativa [18].

1.4 Relación con trabajos afines

Anteriormente se han realizado algunos trabajos afines como:

- Implementación del algoritmo estenográfico F5 para imágenes JPEG a color [13].
- Desarrollo y análisis de una técnica esteganográfica en zonas ruidosas de la imagen mediante transformaciones de color reversibles [19].
- Desarrollo de una herramienta de uso didáctico esteganográfica para envío de texto oculto y cifrado en archivos de audio MP3 [20].

En los trabajos mencionados anteriormente y en el presente proyecto se trata el tema de esteganografía para ocultamiento de información. Sin embargo, existen algunas diferencias que se detallan a continuación:

- En el presente proyecto se utilizará el algoritmo LSB para realizar el ocultamiento en las imágenes.
- Antes de ocultar la información en la imagen se procederá a encriptarla con el algoritmo AES con llave de 128 bits.
- Se tendrá un módulo para editar las imágenes en brillo o zoom.
- Se ocultará un mensaje o un archivo de extensión .txt dentro de la imagen.

2. METODOLOGÍA

El presente trabajo de titulación se realizará empleando una investigación de tipo aplicada, se utilizará la metodología ágil de desarrollo de software *Scrum*. Se plantea una fase inicial donde se realizará la visión del proyecto, los diferentes roles de *Scrum* y los objetivos en común de las diferentes tareas a realizar, posteriormente se desarrolla la planeación, para esto se realiza los levantamientos de información y la identificación de los *sprints*, una vez culminada la parte de planeación se procede al diseño e implementación de cada una de las tareas a cumplir en los *sprint* para dar como resultado un producto final, para concluir se tiene la etapa de revisión y cierre en donde se realizarán cambios en el producto final para acoplar el software a los objetivos y metas planteadas en un principio.

Se recolectará información para los requerimientos de usuario por medio de encuestas. Con los resultados y análisis de estas encuestas se podrá determinar los requerimientos que tiene el usuario y se podrá levantar las historias de usuarios para poder detallarlos.

Para el diseño de la aplicación se lo realizará mediante el uso de diagramas de bloques.

En la implementación del proyecto se usará el software *Matlab* versión 2018b siguiendo los *backlogs* de *Scrum*.

Una vez finalizado cada *sprint* se presentará los avances de la aplicación para la revisión y mantenimiento de los mismos.

Finalmente, cuando el software esté totalmente desarrollado se realizarán todas las pruebas de funcionamiento de cada uno de los módulos y componentes y se procederá con las correcciones respectivas de ser necesario.

2.1 Consideraciones para el diseño

2.1.1 Arquitectura de la aplicación

La aplicación está compuesta por módulos en donde cada uno cumple con una actividad específica como se muestra en la Figura 2.1, se tiene:

- Módulo principal: Menú.
- Módulos secundarios: Cifrar texto en imagen, Cifrar archivos de texto en imagen y Descifrar.
- Módulo ternario: Visualizar.

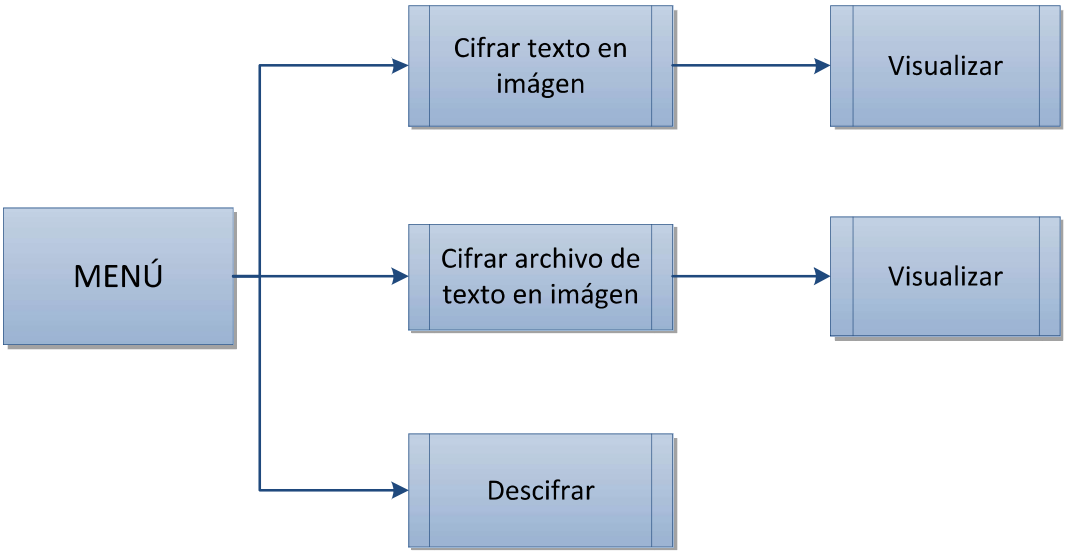

Figura 2.1. Arquitectura de la aplicación

2.1.2 Levantamiento de información

Se determinaron los roles *Scrum*:

- **Maestro *Scrum*:** Fernando Flores
- **Dueño del Producto:** Fernando Flores
- **Desarrollador:** Freddy Real

Para determinar las herramientas para que la aplicación pueda ser dinámica, de ayuda para los estudiantes y ver los formatos de imágenes que más se usan, se realizó una encuesta a los usuarios, esta se encuentra en el Anexo 1.

La encuesta antes mencionada permitirá justificar los requerimientos que el usuario tiene para los formatos de las imágenes y que cumpla con el objetivo de ser didáctica. Para esto se formuló una encuesta de tipo cerrada ya que los encuestados tienen que elegir una de las posibles opciones, se realizó este tipo de encuesta ya que son de fácil análisis y otorgan una manera rápida de cuantificar los resultados [21]. A continuación, se muestra los resultados de las encuestas que se realizaron a 50 personas, teniendo en cuenta que en la carrera de Ingeniería Electrónica y Redes de Información se tiene 233 personas, se consideró un porcentaje de confiabilidad del 90% y un ±11% de margen de error:

La primera pregunta permite conocer cuál es el formato de imágenes que le gustaría usar a los usuarios, en la Figura 2.2 se puede observar que el 56% de los encuestados usa más JPEG, el 44% usa más PNG y 0% usa BMP.

La segunda pregunta ayuda a conocer si el usuario ha abierto una imagen JPEG *lossless*. En la Figura 2.3 se muestra que 18% si han abierto y 82% no han abierto.

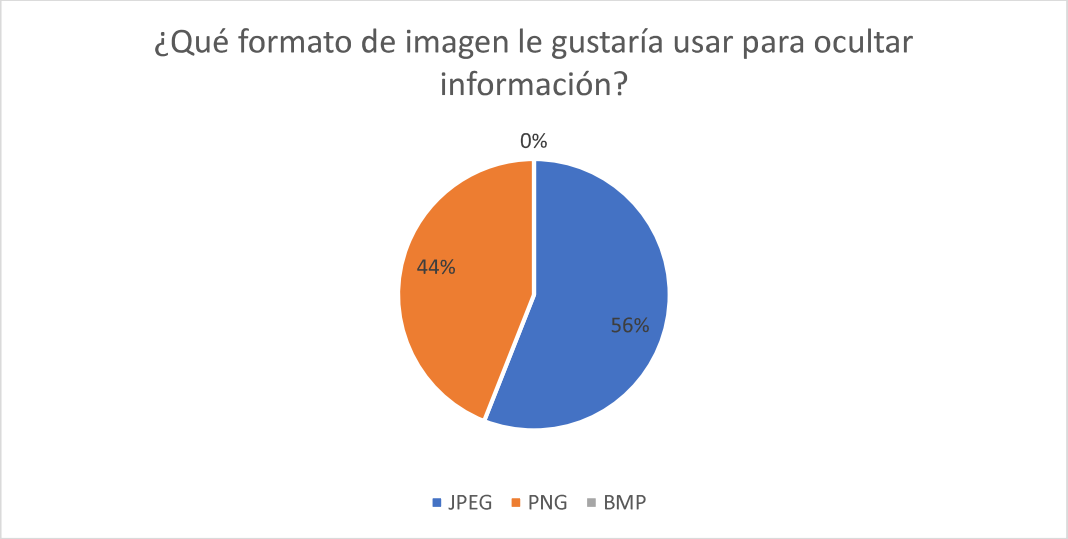

Figura 2.2. Resultados de la primera pregunta de la encuesta

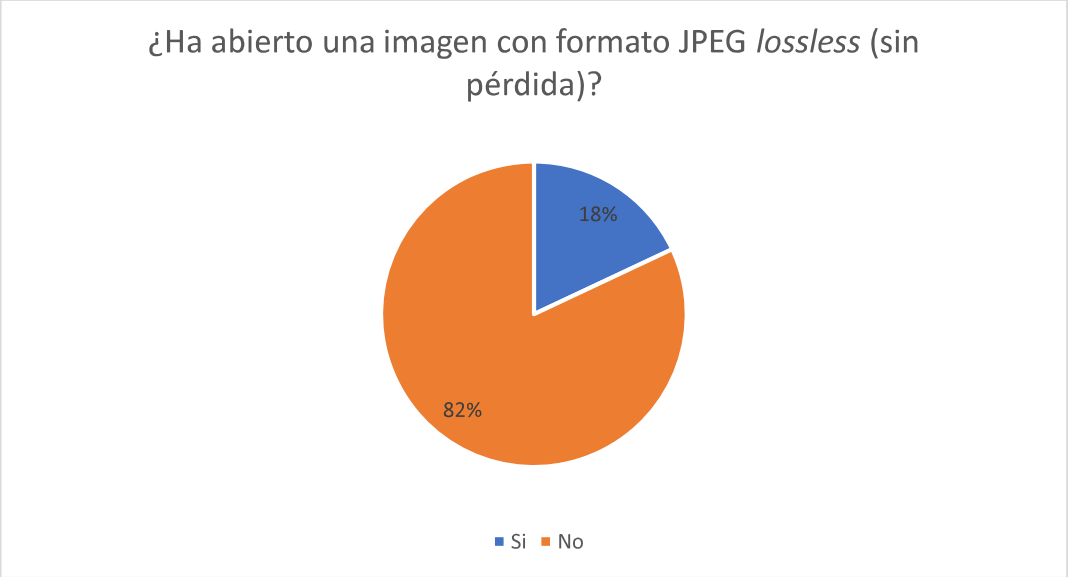

Figura 2.3. Resultados de la segunda pregunta de la encuesta

La tercera pregunta ayuda a conocer si los usuarios han intentado abrir una imagen JPEG *lossless*. En la Figura 2.4 se muestra que el 54% si han intentado mientras que el 46% no ha intentado.

En la cuarta pregunta permite conocer si al usuario le sería útil una aplicación para abrir imágenes JPEG *lossless*. En la Figura 2.5 se puede observar que al 82% si le sería útil mientras que al 18% no les sería útil.

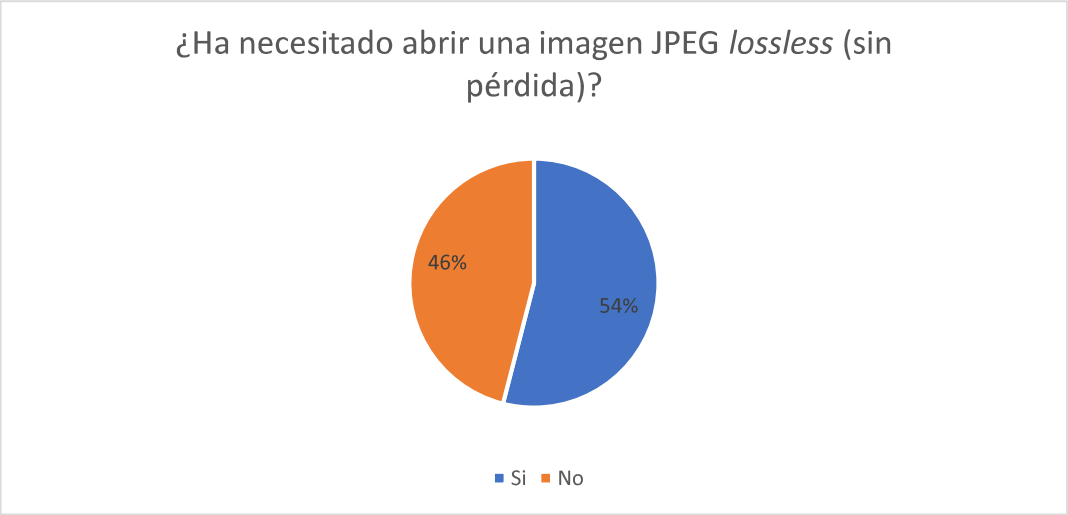

Figura 2.4. Resultados de la tercera pregunta de la encuesta

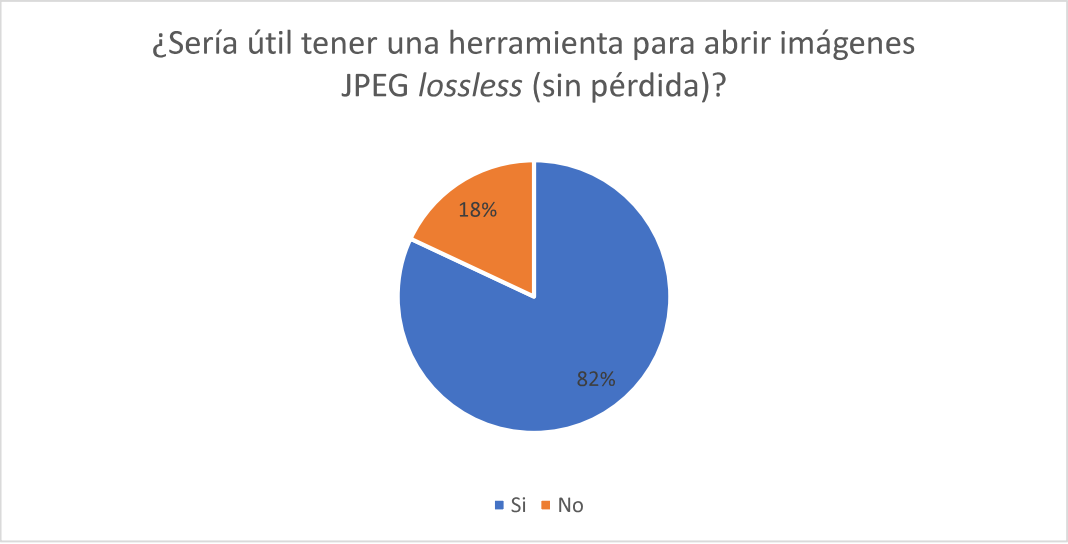

Figura 2.5. Resultados de la cuarta pregunta de la encuesta

La quinta pregunta ayuda a determinar cuál sería la elección del usuario para poder ver la diferencia entre dos imágenes. En la Figura 2.6 se puede ver que el 36% eligió histograma, el 34% Transformada Discreta de Coseno DCT y el 30% eligió una resta entre las dos imágenes.

La sexta pregunta permite determinar si el usuario considera que es una herramienta didáctica si se puede esconder la información en cualquier bit de la imagen. En la figura 2.7 se muestra que al 100% les parece una forma didáctica esconder la información en cualquier bit de la imagen.

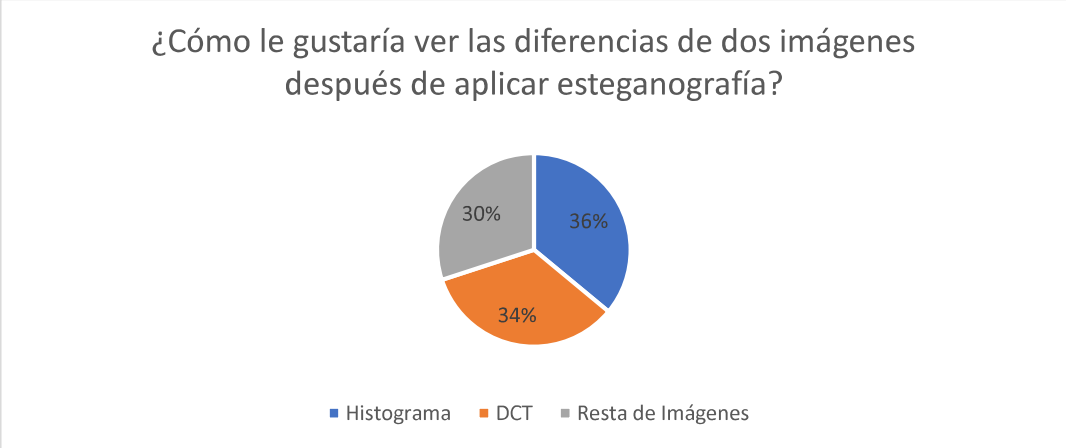

Figura 2.6. Resultados de la quinta pregunta de la encuesta

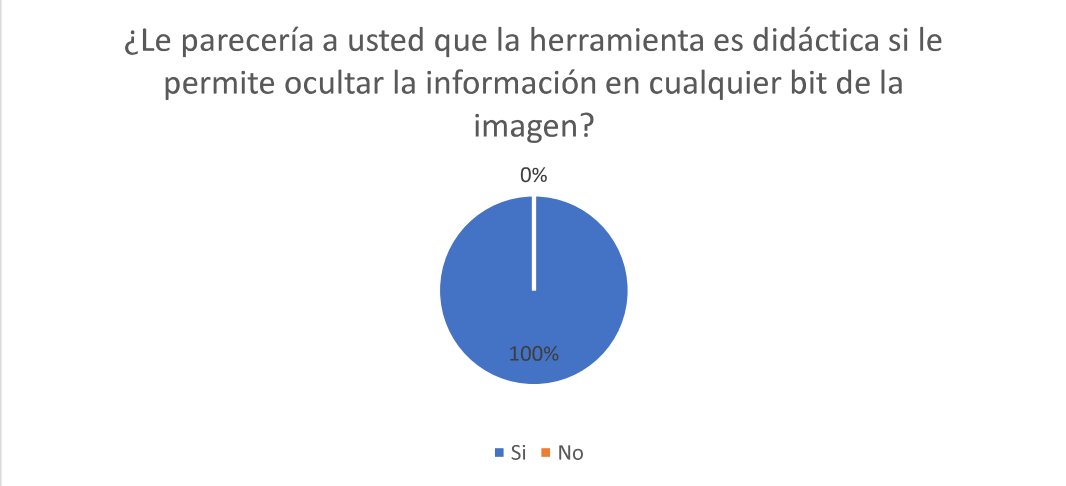

Figura 2.7. Resultados de la sexta pregunta de la encuesta

Finalmente, la séptima pregunta ayuda a determinar si al usuario le ayudaría a su enseñanza una aplicación que oculte información en imágenes usando el algoritmo LSB y muestre dónde están las diferencias. En la Figura 2.8 se muestra que al 86% si le ayudaría a su aprendizaje mientras que al 15% no le ayudaría.

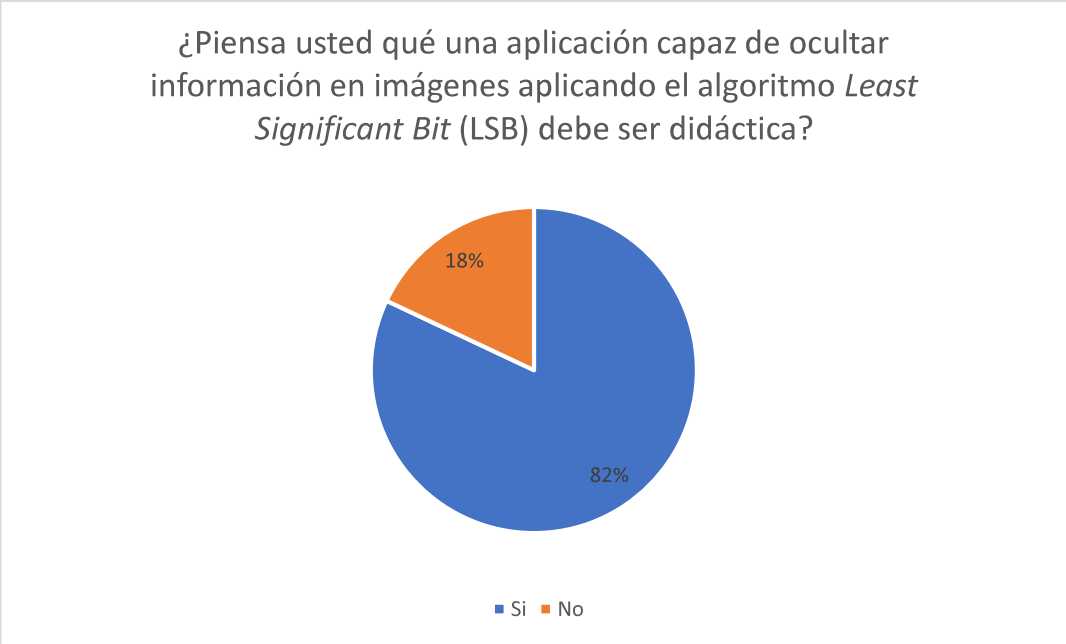

Figura 2.8. Resultados de la séptima pregunta de la encuesta

Después de analizar todos los resultados de las encuestas se puede obtener los siguientes requisitos funcionales iniciales que se los realizará en la aplicación:

- Se usarán dos formatos para las imágenes: JPEG y PNG, con esto se cumpliría el objetivo planteado y el requerimiento del usuario.
- Se creará una aplicación capaz de abrir las imágenes con formato JPEG *lossless* resultantes del algoritmo LSB.
- Para conocer las diferencias que existen entre la imagen original y la estegoimagen se implementará 3 opciones:
  - Histogramas
  - Transformada Discreta del Coseno

- Resta de imágenes

- Se implementará la opción para que el usuario pueda ocultar la información en cualquier bit de la imagen, sea este el menos significativo, el más significativo o un bit central.

- Para el ocultamiento de la información se utilizará el algoritmo *Least Significant Bit (LSB)* con las respectivas diferencias entre imágenes.

2.1.3 Historias de usuario

La aplicación didáctica estará conformada por dos actores: el usuario que realice el ocultamiento en la imagen y el usuario que extraiga la información de la imagen. La encuesta realizada en la sección anterior ayudó a establecer todos los requerimientos iniciales que tiene el usuario alineando dichos requerimientos con el alcance del presente proyecto. A continuación, se detallarán las historias de usuario en la siguiente tabla:

Tabla 2.1 Historias de usuario

ID	Título	Descripción
HU01	Formatos de imágenes JPEG y PNG	El usuario eligió usar los formatos JPEG y PNG para las imágenes donde se oculta la información
HU02	Aplicación para abrir imágenes con formato JPEG *lossless*	El usuario necesita poder abrir las imágenes con formato JPEG *lossless* desde una aplicación.
HU03	Diferencias entre imágenes	El usuario necesita ver las diferencias entre la imagen original y la estegoimagen de diferentes formas.
HU04	Ocultamiento en cualquier bit de la imagen	Para un mejor aprendizaje el usuario necesita poder esconder la información en diferentes bits de la imagen.
HU05	Algoritmo *Least Significant Bit*	El usuario necesita ocultar la información dentro de imágenes con el algoritmo LSB y ver sus diferencias

2.1.4 Requerimientos

Con las historias de usuario se pudo identificar los requerimientos de la aplicación, estos requerimientos se ven limitados por el alcance del proyecto, sin embargo, para el

presente proyecto todos los requerimientos entran dentro del alcance. A continuación, se presentan todos los requerimientos iniciales y los encontrados en el desarrollo del proyecto:

1. Realizar un menú principal que redireccione a los módulos secundarios.
2. Diferenciar los módulos cifrar texto en imagen y cifrar archivo de texto en imagen.
3. Permitir la elección de la imagen con formatos JPEG y PNG.
4. Permitir que el usuario ingrese la contraseña para la encriptación.
5. Permitir que el usuario elija en qué bit se desea guardar la información dentro de la imagen.
6. Permitir que el usuario pueda visualizar las imágenes resultantes.
7. Permitir que el usuario realice cambios en zoom o en brillo.
8. Permitir que el usuario compare los histogramas de las imágenes resultantes.
9. Permitir que el usuario recupere la información ocultada inicialmente
10. Crear una aplicación capaz de abrir los formatos de JPEG *lossless*.
11. Permitir al usuario que en la aplicación creada vea las diferencias de las imágenes con la transformada discreta de Fourier DCT y la resta de imágenes.

2.1.5 Tareas

A cada uno de los requerimientos se les ha asignado un grupo de tareas que deberán cumplirse para que el requerimiento esté finalizado. En la Tabla 2.2 se presenta el desglose de todas las tareas pertenecientes a cada requerimiento, para el estado de cada tarea se tiene algunas opciones, (Por hacer, En proceso, Probando, Realizada), sin embargo, todas estarán en un estado por hacer y conforme se avance con el cumplimiento de las mismas irán cambiando de estado. Los cambios se verán reflejados en el tablero *Scrum*.

Tabla 2.2. Tareas pertenecientes a cada requerimiento (Parte 1)

Requerimiento	Tareas	Estado
1. Realizar un menú principal que redireccione a los módulos secundarios.	1. Crear una interfaz inicial que contenga los módulos secundarios y se pueda acceder a estos mediante botones.	Por hacer
	2. Realizar la programación de cada uno de los botones para la apertura de los módulos secundarios y cierre del menú.	Por hacer

Tabla 2.2. Tareas pertenecientes a cada requerimiento (Parte 2)

Requerimiento	Tareas	Estado
2. Diferenciar los módulos cifrar texto en imagen y cifrar archivo de texto en imagen.	3. Colocar el nombre identificativo en cada botón y cada formulario secundario.	Por hacer
	4. Crear la interfaz de cada uno de los módulos secundarios.	Por hacer
	5. Crear un cuadro de texto que permita ingresar el mensaje en el módulo cifrar texto en imagen.	Por hacer
	6. Crear un botón que permita cargar el archivo .txt en el módulo cifrar archivo de texto.	Por hacer
3. Permitir la elección de la imagen con formatos JPEG y PNG.	7. Programar un botón que permita seleccionar imágenes JPEG y PNG en el módulo cifrar texto en imagen.	Por hacer
	8. Programar un botón que permita seleccionar imágenes JPEG y PNG en el módulo cifrar archivo de texto en imagen.	Por hacer
	9. Crear una variable que almacene la imagen introducida.	Por hacer
	10. Convertir la imagen introducida en bytes.	Por hacer
4. Permitir que el usuario ingrese la contraseña para la encriptación.	11. Colocar un cuadro de texto para que el usuario pueda ingresar la contraseña para el algoritmo de encriptación AES con clave de 128 bits en ambos módulos secundarios.	Por hacer
	12. Configurar el cuadro de texto de la contraseña en caso de que el usuario no introduzca una clave, ésta se genere automáticamente con 16 bytes imprimibles.	Por hacer
	13. Configurar el cuadro de texto de la contraseña si el usuario introduzca una contraseña más grande de 16 bytes, esta sea recortada o a su vez si introduce una contraseña más pequeña, esta sea rellenada.	Por hacer
	14. Mostrar la contraseña randómica generada por el programa en el cuadro de texto	Por hacer
5. Permitir que el usuario elija en que bit se desea guardar la información dentro de la imagen.	15. Colocar un *pop-up* menú en los módulos de cifrado para que el usuario elija el bit donde ocultar la información en los dos módulos de cifrar.	Por hacer
	16. Programar el botón cifrar para que tome como argumentos de entrada la imagen, contraseña y el bit donde realizar la inserción.	Por hacer
	17. Crear un cuadro de texto donde aparezca un extracto del mensaje cifrado en los módulos de cifrado.	Por hacer
	18. Crear un cuadro de texto que confirme que el cifrado fue exitoso.	Por hacer
	19. Colocar las banderas antes y después del mensaje para la recuperación del mismo.	Por hacer

Tabla 2.2. Tareas pertenecientes a cada requerimiento (Parte 3)

Requerimiento	Tareas	Estado
6. Permitir que el usuario pueda visualizar las imágenes resultantes.	20. Crear un botón para visualizar la imagen original y la estegoimagen en los dos módulos de cifrado.	Por hacer
	21. Crear cuatro formularios que permitan mostrar automáticamente las imágenes en los módulos de cifrado.	Por hacer
	22. Transformar la imagen ingresada para que el visualizador muestre 2 imágenes originales en JPEG y PNG y 2 estegoimagen en JPEG y PNG.	Por hacer
7. Permitir que el usuario realice cambios en zoom o en brillo.	23. Colocar un *slider* en cada formulario y un *radio button* para los cambios de brillo y zoom en cada imagen.	Por hacer
	24. Colocar un botón guardar en cada formulario para guardar cada imagen.	Por hacer
	25. Programar los nombres con los que se guardarán las imágenes.	Por hacer
8. Permitir que el usuario compare los histogramas de las imágenes resultante	26. Colocar un botón que permita visualizar el histograma de cada imagen.	Por hacer
	27. Configurar la gráfica para que recepte los valores de la imagen y convertirlos en un histograma.	Por hacer
9. Permitir que el usuario recupere la información ocultada inicialmente	28. Crear un tercer módulo secundario en el menú.	Por hacer
	29. Configurar el módulo de desciframiento.	Por hacer
	30. Colocar un botón para seleccionar la imagen para recuperación del mensaje.	Por hacer
	31. Colocar un cuadro de texto para colocar la contraseña de desciframiento.	Por hacer
	32. Colocar un *pop-up* menú para seleccionar de que bit se desea recuperar la información.	Por hacer
	33. Programar el algoritmo inverso para encontrar las banderas y recuperar el mensaje o archivo de texto.	Por hacer
	34. Colocar un cuadro de texto donde aparezca el mensaje introducido inicialmente o se recupere el archivo de texto .txt	Por hacer
	35. Colocar un nuevo título dentro del archivo de texto recuperado.	Por hacer
10. Crear una aplicación capaz de abrir los formatos de JPEG *lossless*.	36. Crear y programar una aplicación ejecutable en *Matlab*.	Por hacer
	37. Configurar para que la aplicación pueda mostrar dos imágenes.	Por hacer
	38. Programar la aplicación para que pueda abrir los formatos de imágenes PNG y JPEG *lossless*.	Por hacer

Tabla 2.2. Tareas pertenecientes a cada requerimiento (Parte 4)

Requerimiento	Tareas	Estado
11. Permitir al usuario que en la aplicación creada vea las diferencias de las imágenes con la transformada discreta de Fourier DCT y la resta de imágenes.	38. Configurar la aplicación para que pueda mostrar la transformada discreta de Fourier y la resta entre las dos imágenes para visualización de diferencias entre las imágenes antes introducidas.	Por hacer

2.1.6 Tablero *Scrum*

A cada tarea que se registró anteriormente se le debe dar un seguimiento mediante el tablero *Scrum*, para esto se hizo uso de una herramienta llamada Jira, en donde se puede introducir y ver el proceso en el que se encuentra dicha tarea. En la Figura 2.9 se puede observar el inicio del tablero de *Scrum*

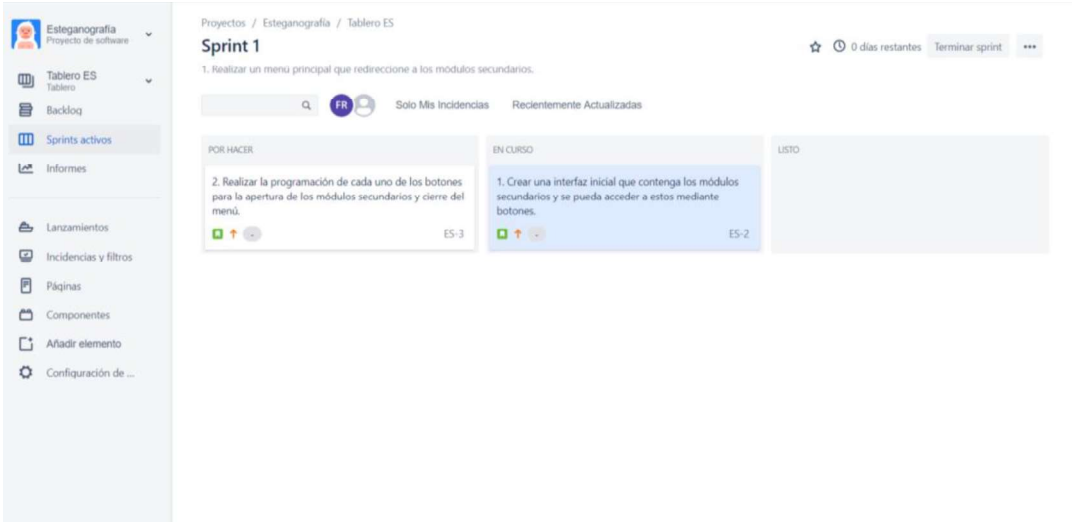

Figura 2.9. Inicio del tablero *Scrum*

2.1.7 *Sprint backlog*

Para la creación de los *sprint* se debe agrupar los requerimientos teniendo en cuenta que al final de cada *sprint* se debe generar un entregable, el conjunto de todos los *sprints* es llamado *sprint backlog*. En la Tabla 2.3 se muestra el desglose de los *sprints*.

Tabla 2.3. *Sprint backlog*

Sprint	**Nombre**	**Requerimiento**
Sprint 1	Interfaz Inicial	1. Realizar un menú principal que redireccione a los módulos secundarios.
		2. Diferenciar los módulos cifrar texto en imagen y cifrar archivo de texto en imagen.
Sprint 2	Encriptación y ocultamiento de la información.	3. Permitir la elección de la imagen con formatos JPEG y PNG.
		4. Permitir que el usuario ingrese la contraseña para la encriptación.
		5. Permitir que el usuario elija en qué bit se desea guardar la información dentro de la imagen. Permitir que el usuario pueda visualizar las imágenes resultantes.
Sprint 3	Visualización de imágenes y recuperación de información oculta.	6. Permitir que el usuario realice cambios en zoom o en brillo.
		7. Permitir que el usuario compare los histogramas de las imágenes resultantes.
		8. Permitir que el usuario recupere la información ocultada inicialmente.
Sprint 4	Aplicación para imágenes con formato JPEG *lossless* y PNG	9. Crear una aplicación capaz de abrir los formatos de JPEG *lossless*.
		10. Permitir al usuario que en la aplicación creada vea las diferencias de las imágenes con la transformada discreta de Fourier DCT y la resta de imágenes.

2.1.8 Diagrama de bloques

Para un mejor entendimiento de los algoritmos a usar se realizará algunos diagramas de bloques:

Las funciones programadas en AES seguirán el diagrama de la Figura 2.10:

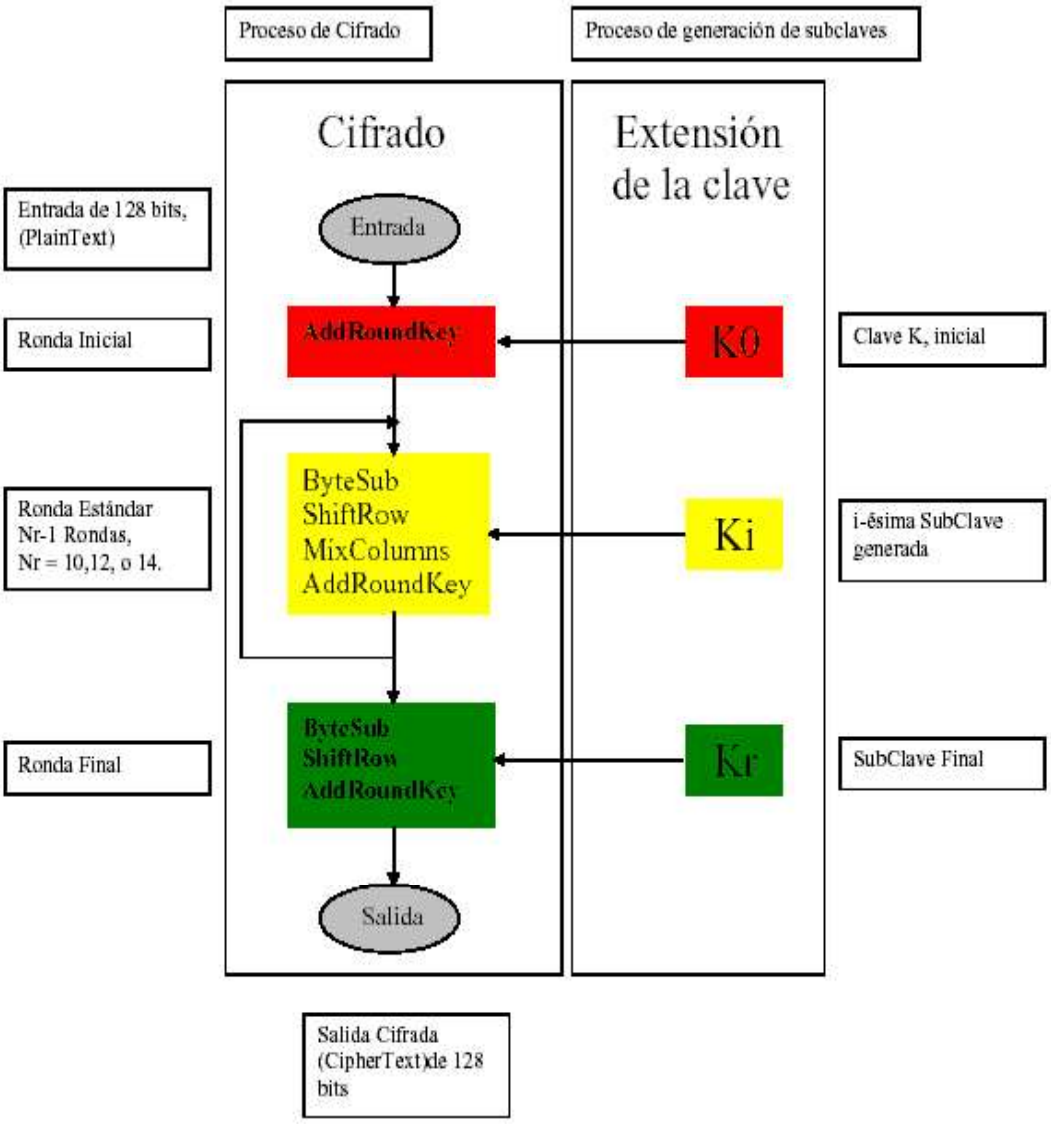

Figura 2.10. Funciones del algoritmo AES [22]

Para el algoritmo de ocultamiento LSB se puede muestra el diagrama en la Figura 2.11, para esto se debe tener en cuenta que se podrá ocultar en cualquier bit que el usuario decida para fines didácticos, este diagrama permite apreciar el proceso a seguir una vez

que se ingresa la imagen y el texto a ocultar, se debe tener en cuenta que este texto pasa por un algoritmo de encriptación antes de ocultarlo en la imagen.

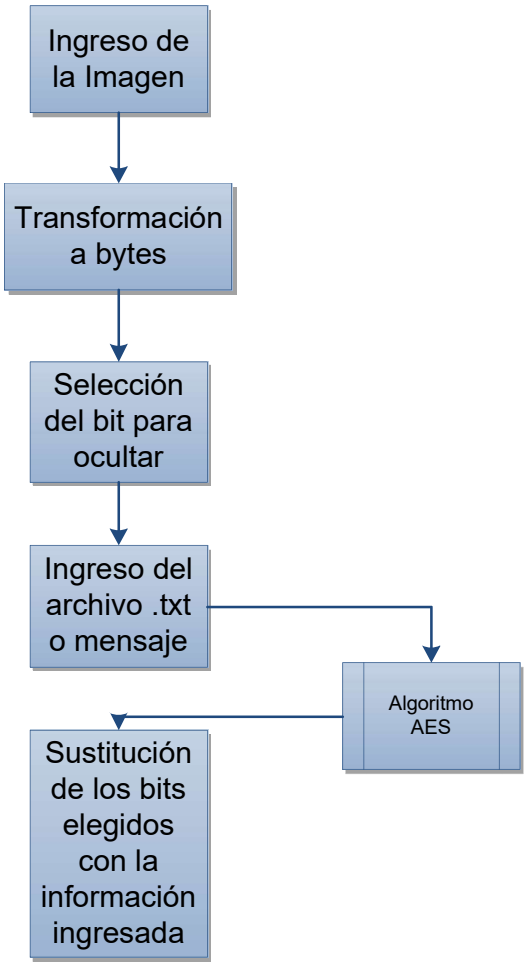

Figura 2.11. Proceso del algoritmo LSB

2.2 Ambiente de desarrollo

A continuación, se podrá observar la aplicación que se ha instalado y los pasos previos que se necesitan antes de realizar la programación de la aplicación.

2.2.1 Matlab 2018b

El lenguaje de programación que se usa es el propietario de Matlab, se utiliza sus herramientas como *GUIDE* para la realización de los formularios en donde se desplegará la información, *Matlab* cuenta con soporte para todas las funciones que admite en su página oficial o a través de la interfaz, de igual manera para la descarga

de este software se lo puede realizar desde la página oficial y con la licencia respectiva [22], se debe tener en cuenta que en la instalación se debe seleccionar todas las librerías que se muestran en la Figura 2.12 para evitar cualquier error en el uso de las funciones [23]. En la Figura 2.13 se puede observar el ambiente de trabajo de *Matlab*.

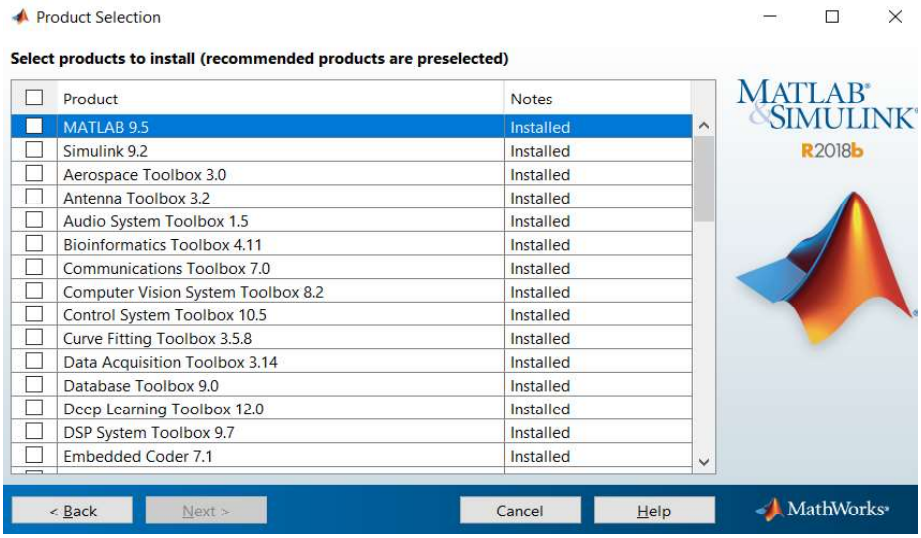

Figura 2.12. Productos a instalar

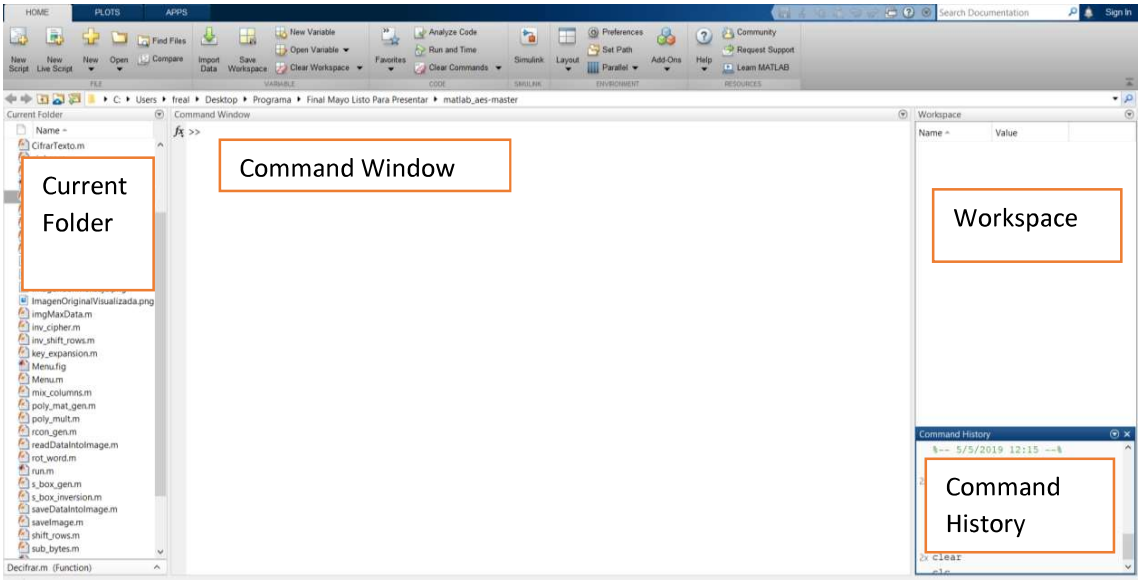

Figura 2.13. Espacio de trabajo de *Matlab*

2.2.2 Requisitos del sistema operativo

Para el presente proyecto se utilizó la plataforma de Windows, sin embargo, *Matlab* también se puede instalar en Linux o macOS [24].

- **Sistema operativo**
  - **Windows:** De Windows 7 en adelante y en Windows Server 2008 en adelante.
- **Procesador:** Intel o AMD de 64 o 32 bits con soporte de instrucciones AVX2.
- **Disco:** 3 GB solo para *Matlab* y de 5 a 8 GB para una instalación típica, se recomienda un disco de estado sólido.
- **RAM:** 4 GB mínimo, recomendado 8 GB.
- **Tarjeta Gráfica:** Soporte para *OpenGL* 3.3 recomendado con 1 GB en GPU [25].

2.2.3 Consideraciones para el código

Para la estructura del directorio del proyecto se colocarán todas las imágenes creadas en el *path* donde se encuentren las funciones y los *scripts* alojados abiertos desde *Matlab*.

En el ANEXO II se detallará los diferentes archivos y funciones que componen la interfaz para el desarrollo de la aplicación.

2.3 *Sprint* interfaz inicial

El entregable de este *sprint* será la interfaz inicial que sea capaz de redireccionar al usuario a los otros módulos secundarios, así como todo el ambiente gráfico del menú principal.

2.3.1 Menú

Para este *sprint* se consideró un *sketch* del módulo principal el cual permitirá ingresar a los diferentes módulos secundarios, así como la presentación del proyecto como se muestra en la Figura 2.14. Para esto se programó en un GUIDE de Matlab el prototipo que tendría para mostrar la presentación del producto final.

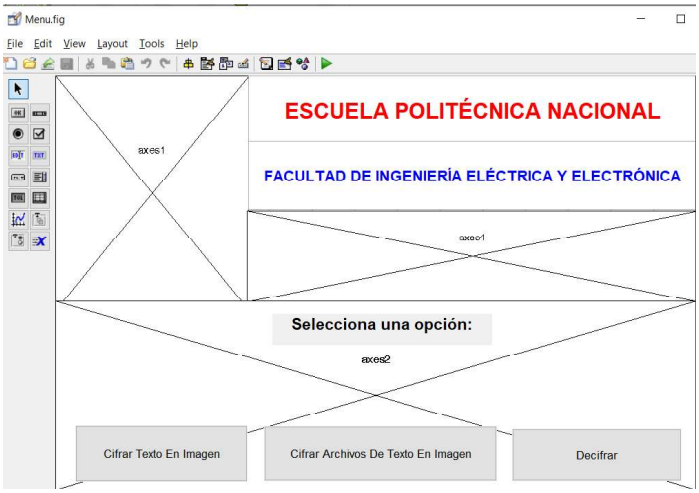

Figura 2.14. *Sketch* del módulo inicial

El módulo menú es el encargado de direccionar al usuario hacia uno de los módulos secundarios que se desea escoger, en este caso se tiene tres botones dentro del módulo que se muestra a continuación:

- `Cifrar Texto En Imagen`
- `Cifrar Archivo De Texto En Imagen`
- `Descifrar`

Este módulo fue programado como una presentación y cada uno de sus botones para abrir un módulo secundario y ocultar el menú. Para esto se ha aplicado funciones para cargar imágenes en los *axes* [26], al igual que para abrir y ocultar los formularios.

Se debe tener en cuenta que las imágenes tienen su propio *path* por lo cual se debe modificar de acuerdo al computador en el que se corra el programa.

2.3.2 Diseño

Para este *sprint* se creó la función Menu.m y el *guide* Menu.fig, en el Código 2.1 se puede observar las funciones utilizadas para llamar a las imágenes que actúan en el *guide*. Se crea la variable `path` (línea 61) que se sobrescribirá de acuerdo a la imagen que se vaya llamando. Se observa las funciones `imread` (línea 62) que usa como parámetro de entrada la variable `path` para leer la imagen desde esa ubicación [27].

La función `imshow` (línea 63) usa como parámetro de entrada el resultado de la función `imread`, es decir la imagen seleccionada y la muestra en el *axes* y se desactiva la visión

de los ejes con la función *axis off* (línea 64). Este proceso se repite para las imágenes que se cargan al iniciar el módulo principal.

```
58 -    handles.output = hObject;
59 -    axes(handles.axes1);
60      %Programación de las imagenes para la presentación
61 -    path = 'C:\Users\freal\Desktop\Programa\epn.jpg';
62 -    imag = imread(path);
63 -    imshow(imag);
64 -    axis off;
65      %set(handles.text1,'String',num2str(imag));
66 -    handles.output = hObject;
67 -    axes(handles.axes2);
68 -    path = 'C:\Users\freal\Desktop\Programa\facultad.jpg';
69 -    imag = imread(path);
70 -    imshow(imag);
71 -    axis off;
72 -    handles.output = hObject;
73 -    axes(handles.axes4);
74 -    path = 'C:\Users\freal\Desktop\Programa\titulo.jpg';
75 -    imag = imread(path);
76 -    imshow(imag);
77 -    axis off;
```

Código 2.1. Carga de imágenes

En el Código 2.2 se muestra la programación de los tres botones para abrir los nuevos formularios y cerrar el formulario actual, para lo cual se usa la función `close` con el parámetro *gcf* (línea 102) que hace referencia al formulario que está abierto y de acuerdo al botón que se presione se abre el formulario correspondiente llamándolo mediante el nombre en el `archivo .fig`.

Toda la programación se realiza en los *callbacks* de cada objeto agregado al *guide*.

```
96      % --- Executes on button press in pushbutton1.
97      function pushbutton1_Callback(hObject, eventdata, handles)
98      % hObject    handle to pushbutton1 (see GCBO)
99      % eventdata  reserved - to be defined in a future version of MATLAB
100     % handles    structure with handles and user data (see GUIDATA)
101     %Se cierra el formulario actual
102 -   close('gcf')
103     %Se abre el sigueinte formulario
104 -   CifrarTexto
105
106     % --- Executes on button press in pushbutton2.
107     function pushbutton2_Callback(hObject, eventdata, handles)
108     % hObject    handle to pushbutton2 (see GCBO)
109     % eventdata  reserved - to be defined in a future version of MATLAB
110     % handles    structure with handles and user data (see GUIDATA)
111     %Se cierra el formulario actual
112 -   close('gcf')
113     %Se abre el sigueinte formulario
114 -   CifrarArchivosDeTexto
115
116     % --- Executes on button press in pushbutton3.
117     function pushbutton3_Callback(hObject, eventdata, handles)
118     % hObject    handle to pushbutton3 (see GCBO)
119     % eventdata  reserved - to be defined in a future version of MATLAB
120     % handles    structure with handles and user data (see GUIDATA)
121     %Se cierra el formulario actual
122 -   close('gcf')
123     %Se abre el sigueinte formulario
124 -   Decifrar
```

Código 2.2. Apertura y cierre de formularios

2.3.3 Entregables

Los entregables finales para este *sprint* se muestran en la Figura 2.15

Figura 2.15. Módulo inicial entregable del *sprint* 1

2.4 *Sprint* encriptación y ocultamiento de la información

En este *sprint* se tendrá como entregable dos módulos secundarios con las funcionalidades de encriptar la información con el algoritmo AES con llave de 128 bits y el ocultamiento de la información con el algoritmo, cargar la imagen al módulo, introducir la contraseña, elegir el bit donde se desea ocultar la información e ingresar la información a ocultar.

2.4.1 Módulos Secundarios

Este *sprint* tendrá dos *sketches*, el primero será el módulo secundario para ocultar un mensaje por teclado y el segundo será un módulo secundario para ocultar un archivo de texto con extensión .txt

El *sketch* para ocultar un mensaje ingresado por teclado se muestra en la Figura 2.16 mientras que el *sketch* para ocultar un archivo de texto .txt se muestra en la Figura 2.17. Estos dos *sketches* tendrán un formulario muy parecido para el ingreso de la imagen, contraseña y elección del bit donde se desea ocultar la información.

Se debe tener en cuenta que para llegar a estos módulos se los debe seleccionar del menú principal.

Figura 2.16. *Sketch* módulo `cifrar texto en imagen`

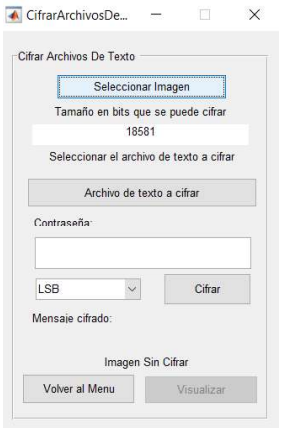

Figura 2.17. *Sketch* módulo `cifrar archivo de texto en imagen`

2.4.2 Diseño

En este *sprint* se hace uso del algoritmo de encriptación y ocultamiento por lo que se partirá desde codificación:

2.4.3 Algoritmo de encriptación

Para el algoritmo de encriptación se usó AES con clave de 128 bits, para esto se procedió a programar varias funciones independientes que se llamarán conforme se las necesite en el programa. Estas funciones ayudarán para la encriptación y para la desencriptación, ya que la clave es de 128 bits se ingresa 16 caracteres cada uno de 1

byte que estén dentro de los caracteres imprimibles *ASCII* es decir del 33 al 126. Estos caracteres se ingresan en el cuadro de texto de `Contraseña` teniendo en cuenta que el programa fabricará una contraseña si esta no es ingresada, así mismo se completará con caracteres aleatorios si no tiene 16 bytes o procederá a recortarla en caso de que exceda el tamaño requerido.

Una vez que se tenga lista la contraseña se procederá a encriptar el mensaje o archivo. Se mostrará una pequeña parte de la información encriptada en un cuadro de texto, si el mensaje fue encriptado de forma correcta, en caso de que el programa haya hecho una clave randómica se podrá ver en el cuadro de texto `Contraseña` para saber cuál fue la clave usada.

Para la desencriptación se debe colocar la misma clave ingresada, de esta manera se podrá visualizar la información en texto plano, si la contraseña no es la misma que se ingresó la información mostrada serán incongruencias que no se podrá leer. De igual manera para este proceso se hace uso de las funciones inversas ya programadas anteriormente [28].

Primero se ingresa la clave de 16 bytes que será utilizada, se tiene una función llamada `getRandomString` que se muestra en el Código 2.3, es la encargada de llenar con caracteres imprimibles (con el código ASCII del 33 al 126), en caso de que no se ingrese la contraseña o a su vez sea de menor tamaño, para ello se usa la función `randomStr` que coge los caracteres imprimibles de la variable `printableChars`.

```
1     %Se crea caracteres randómicos según ASCII del 33 al 126
2     function randomStr = getRandomString( len )
3
4 -       printableChars = [ 33 126 ];
5 -       randomStr = char( randi( printableChars, 1, len ) );
6
7 -   end
```

Código 2.3. Código para rellenar de caracteres randómicos

Después se inicializa las variables, para esto la función `aes_init` es la encargada de generar las dos tablas de sustitución `s_box` y la tabla `inv_s_box` mediante la función `s_box_gen`, define el vector constante `rcon`, la clave expandida y las dos matrices

polinomiales `poly_mat` e `inv_poly_mat` [29]. En la figura 2.18 se puede observar el diagrama de bloques de la función.

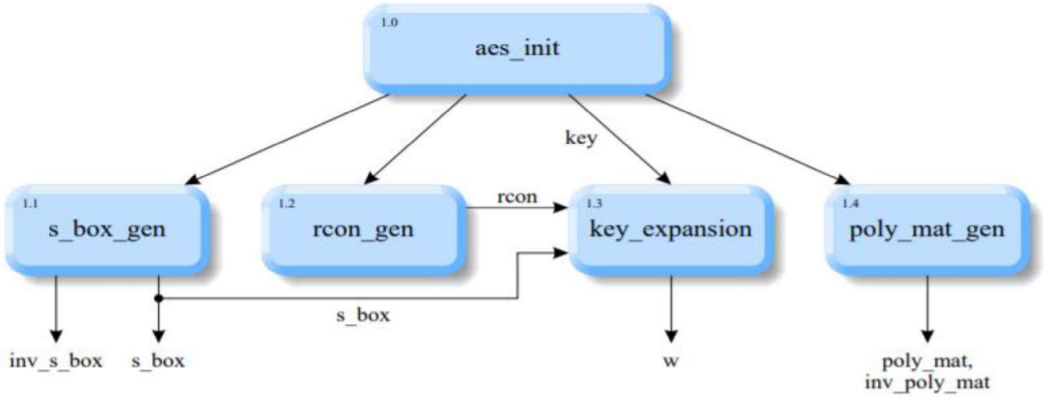

Figura 2.18. Diagrama de la función `aes_init` [29]

Esta función es usada en el *script* `encrypData_AES128` y `decrypData_AES128` como se observa en el código 2.4

```
53    %Inicialización de variables
54    function [ s_box, inv_s_box, poly_mat, inv_poly_mat, w ] = aes_init( key, round_num )
55
56 -      [s_box, inv_s_box] = s_box_gen;
57 -      rcon = rcon_gen(round_num);
58
59 -      w = key_expansion (key, s_box, rcon, round_num);
60 -      [poly_mat, inv_poly_mat]= poly_mat_gen;
61
62 -  end
```

Código 2.4. Inicialización de las variables para encriptar

Las tablas de sustitución `s_box` e `inv_s_box` son utilizadas por la clave expandida para programar la función `key_expansion` y las funciones de cifrado y descifrado: `cipher` e `inv_cipher` para sustituir directamente un byte por otro byte del mismo campo finito. La función `s_box_gen` se muestra en el Código 2.5 crea las cajas S (línea 2) buscando los inversos de todos los elementos de GF (2^8) (líneas 5 a 7) mediante el uso de `find_inverse` (línea 6) y aplicando transformaciones afines a todos los inversos (líneas 8 a 9) a través de `aff_trans` (línea 8). Finalmente, la caja S inversa

`inv_s_box`, que se utilizará en `inv_cipher`, es construida a partir de `s_box` por `s_box_inversion` (línea 11).

Se define el polinomio de reducción modular AES estándar `mod_pol` $x^8 + x^4 + x^3 + x + 1$ (línea 3) y se establece que la inversa de 0 se define como 0 (línea 4). Es importante acotar que las matrices en *Matlab* (vectores y matrices) comienzan con un índice de 1. En la Figura 2.19 se puede observar el diagrama de bloques de la función `s_box_gen`.

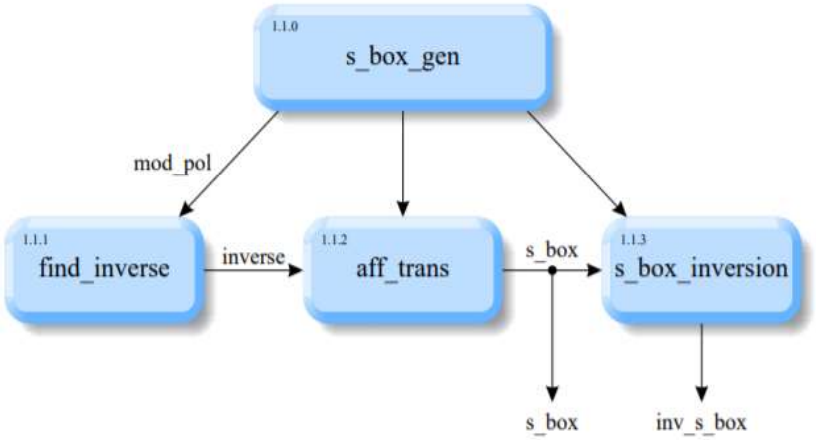

Figura 2.19. Función `s_box_gen` [29]

```
1    %Creación de la caja S
2    function [s_box, inv_s_box] = s_box_gen
3        mod_pol = bin2dec ('100011011'); %Polinomio de reducción modular
4        inverse(1) = 0;
5        for i = 1 : 255
6            inverse(i + 1) = find_inverse (i, mod_pol); %Inversos campos de galois
7        end
8        for i = 1 : 256
9            s_box(i) = aff_trans (inverse(i));
10       end
11       inv_s_box = s_box_inversion (s_box);%Creación caja S inversa
```

Código 2.5. Creación de la caja S

La función `find_inverse` es un bucle que recorre todos los valores posibles de los bytes para calcular el byte que se invertirá como se muestra en el Código 2.6.

```
1     %Bucle a traves de todos los valores de bytes posibles
2   function b_inv = find_inverse (b_in, mod_pol)
3 -     for i = 1 : 255
4 -         prod = poly_mult (b_in, i, mod_pol);
5 -         if prod == 1
6 -             b_inv = i;
7 -             break
8 -         end
9 -     end
```

Código 2.6. Función `find_inverse`

Una vez que se han encontrado los inversos de todos los bytes, el segundo paso del proceso de creación de la caja S, es una transformación en la función `aff_trans` que se muestra en el código 2.7, consiste en una multiplicación polinomial de una constante específica (31_d = 00011111_b) módulo otra constante (257_d = 10000001_b) y la adición de una tercera constante (99_d = 01100011_b):

$$b_{out} = b_{in} \bullet 31_d \bmod 257_d \oplus 99_d$$

Donde b_{in} representa el byte de entrada que se va a transformar, $\oplus$ denota la operación XOR bit a bit, y el byte de salida después de la transformación como b_{out}.

Se declara la función de *Matlab* `aff_trans` transformando la entrada byte `b_in` en el byte de salida `b_out` (línea 2). Las tres constantes se definen en forma binaria (línea 3 a 5), se hace la multiplicación de módulo (línea 6) y el XOR bit a bit (línea 7).

```
1     %Transformación con las constantes
2   function b_out = aff_trans (b_in)
3 -     mod_pol = bin2dec ('100000001');%Constantes
4 -     mult_pol = bin2dec ('00011111');
5 -     add_pol = bin2dec ('01100011');
6 -     temp = poly_mult (b_in, mult_pol, mod_pol);%Operaciones
7 -     b_out = bitxor (temp, add_pol);
```

Código 2.7. Función `aff_trans`

La caja S inversa que se crea en la función `s_box_inversion` se muestra en el Código 2.8, se utiliza en la función de descifrado `inv_cipher` para revertir la sustitución realizada a través de la caja S, toma el `s_box` como su entrada y genera la caja S inversa `inv_s_box` en un solo bucle (línea 2 a 4). El bucle recorre todos los elementos de la caja S, interpreta el valor del elemento `s_box` actual como un índice en la `s_box`

inversa e inserta los valores de 0 hasta 255 en los lugares apropiados en la s_box inversa (línea 4).

```
1     %Creación de la caja s inversa
2     function inv_s_box = s_box_inversion (s_box)
3 -       for i = 1 : 256
4 -           inv_s_box(s_box(i) + 1) = i - 1;
5 -       end
```

Código 2.8. Función s_box_inversion

La matriz de constante redonda se utiliza en el esquema de expansión de clave key_expansion. Es una matriz de ceros de 10 × 4, excepto la primera columna, que contiene potencias del 2, esto se programa en la función rcon_gen como se muestra en el Código 2.9.

Se define el polinomio mod_pol estándar de AES (línea 13) y un valor inicial constante de 1 (línea 14). Las potencias restantes de 2 son iterativamente calculadas por la multiplicación polinomial del valor anterior por 2 (línea 15 a 17). Finalmente, se agregan tres columnas cero (línea 18).

```
1     %Creación de la matriz de ceros y las potencias de 2
2     function rcon = rcon_gen(round_num)
3 -       switch round_num %Posibles casos
4 -           case 10
5 -               Rc = 10 ;
6 -           case 12
7 -               Rc = 8 ;
8 -           case 14
9 -               Rc = 7 ;
10 -          otherwise
11 -              error('Número de ronda incorrecta') ;
12        end
13 -      mod_pol = bin2dec ('100011011');
14 -      rcon(1) = 1; % genera array
15 -      for i = 2 : Rc
16 -          rcon(i) = poly_mult (rcon(i-1), 2, mod_pol);
17 -      end
18 -      rcon = [rcon(:), zeros(Rc, 3)];
```

Código 2.9. Función rcon_gen

La función de expansión de la clave (key_expansion) toma la clave de 16 bytes proporcionada por el usuario y utiliza la matriz de rondas constante creada anteriormente y la tabla de sustitución s_box para generar una clave de 176 bytes w, que se utilizará durante el proceso de encriptación y de desencriptación. El bucle

cerrado indica que las funciones `rot_word` y `sub_bytes` son llamados iterativamente por la función de expansión de clave. En la Figura 2.20 se puede observar el diagrama de bloques de la función `key_expansion`.

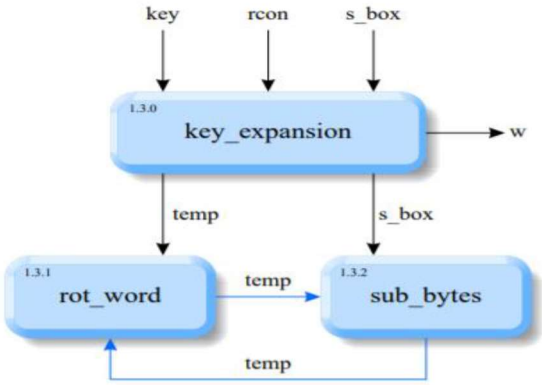

Figura 2.20. Función `key_expansion` [29]

Los 16 bytes de la clave se reorganizan en una matriz de 4 x 4, como se observa en la Figura 2.21.

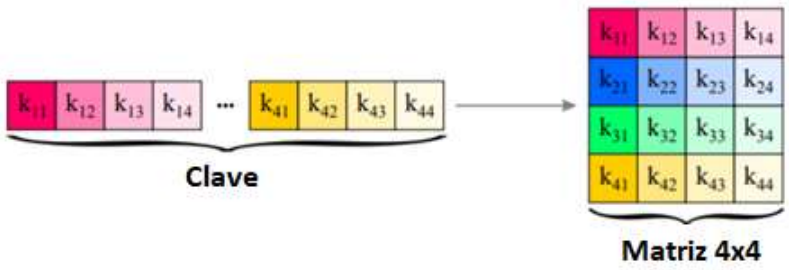

Figura 2.21. Reordenamiento de la clave en matriz 4x4 [29]

Cada fila siguiente se crea de manera diferente. Antes de aplicar la operación XOR, la fila predecesora se gira, se sustituye y se aplica la operación XOR con su constante de ronda correspondiente. Para realizar las operaciones de XOR se usa la matriz `rcon` que es una matriz de ceros de 10 x 4 exceptuando la primera columna que tiene 1, 2, 4, 8 y sus múltiplos de 2. Aquí también se utiliza la función de permutación `rot_word` para realizar una permutación cíclica (desplazamiento hacia la derecha) de la palabra de entrada. Este proceso se puede observar en la Figura 2.22.

Un bucle crea las 40 filas restantes de la programación de la clave (línea 14 a 27). Un *buffer* `temp` se llena con la fila anterior (línea 15), que es XOR-ed con la fila que se encuentra en la línea 18. La línea 5 verifica si el índice de la fila coincide con 5, 9, 13, etc., si este es el caso, la fila del *buffer* está permutada cíclicamente (línea 18), sustituida y XOR-ed con la constante de redondeo apropiada (líneas 9 y 10), antes de que finalmente sea XOR-ed (línea 26).

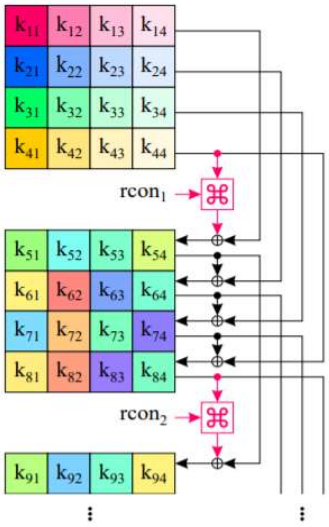

Figura 2.22. Expansión de la clave de 16 bytes [29]

```
1    %Crea la nueva clave de 176 bytes
2    function w = key_expansion (key, s_box, rcon, round_num)
3        switch round_num
4            case 10
5                Nk = 4 ;
6            case 12
7                Nk = 6 ;
8            case 14
9                Nk = 8 ;
10           otherwise
11               error('Número de ronda incorrecta') ;
12       end
13       w = (reshape (key, 4, Nk))';
14       for i = (Nk+1) : (round_num+1)*4
15           temp = w(i - 1, :);    %Buffer
16           temp_t = dec2hex(temp);
17           if mod (i, Nk) == 1
18               temp = rot_word (temp);%Operaciones
19               temp = sub_bytes (temp, s_box);
20               r = rcon ((i - 1)/Nk, :);
21               temp = bitxor (temp, r);
22           elseif mod (i, 4) == 1
23               temp = sub_bytes (temp, s_box);
24               temp_t = dec2hex(temp) ;
25           end
26           w(i, :) = bitxor (w(i - Nk, :), temp);
27       end
```

Código 2.10. Función `key_expansion`

La función de permutación `rot_word` se muestra en el Código 2.11. La palabra entrante `w_in` es un vector de fila de cuatro bytes y se permuta cíclicamente según la Figura 2.23. En *Matlab*, la permutación se puede lograr fácilmente utilizando el nuevo vector de índice (Línea 3).

Figura 2.23. Permutación cíclica [29]

```
1     %Permutación ciclica
2     function w_out = rot_word (w_in)
3 -       w_out = w_in([2 3 4 1]);
```

Código 2.11. Función `rot_word`

La función de sustitución `sub_bytes` mostrada en el Código 2.12, aplica la `s_box` a uno o más bytes de entrada `bytes_in`, la `s_box` se aplica a un byte, un vector de bytes o incluso una matriz completa de bytes. Esto hace que sea posible utilizar la misma función de sustitución simple tanto para la sustitución de un vector de fila en `key_expansion` como para la sustitución de la matriz de estado en el cifrado de la función de `cipher`.

```
1     %Aplicación de s_box
2     function bytes_out = sub_bytes (bytes_in, s_box)
3 -       bytes_out = s_box(bytes_in + 1);
```

Código 2.12. Función `sub_bytes`

Las matrices polinomiales `poly_mat` e `inv_poly_mat` que se muestran en el Código 2.13, se usan en la función `mix_columns` llamada por las funciones de cifrado y descifrado `cipher` e `inv_cipher` respectivamente. Ambas matrices tienen un tamaño de 4 × 4 y cada fila es una permutación cíclica (desplazamiento a la derecha) de la fila anterior. La función `poly_mat_gen` logra la permutación por filas de dicha matriz mediante una llamada a la función `cycle` (líneas 6 y 10). Se crea la matriz a permutar (líneas 3 a 5 y 6 a 8) definiendo su primera fila en representación hexadecimal,

convirtiendo la fila a decimal (líneas 4 y 8) y cuadruplicando la fila en una matriz de 4 × 4 (líneas 5 y 8).

```
1    %Creación de las matrices
2    function [poly_mat, inv_poly_mat] = poly_mat_gen
3        row_hex = {'02' '03' '01' '01'};
4        row = hex2dec (row_hex)';
5        rows = repmat (row, 4, 1);
6        poly_mat = cycle (rows, 'right');
7        inv_row_hex = {'0e' '0b' '0d' '09'};
8        inv_row = hex2dec (inv_row_hex)';
9        inv_rows = repmat (inv_row, 4, 1);
10       inv_poly_mat = cycle (inv_rows, 'right');
```

Código 2.13. Función `poly_mat_gen`

La función `poly_mult` mostrada en el Código 2.14, realiza la multiplicación de dos polinomios (a y b) en GF (2^8) utilizando un tercer polinomio (`mod_pol`) para la reducción modular, `poly_mult` realiza la multiplicación (líneas 4 a 9) y la reducción modular (líneas 10 a 15) en dos bucles muy similares. Para la multiplicación, se prueba cada bit (línea 4) del primer factor a (línea 5) y, si está presente, el segundo factor `b` se desplaza a la izquierda (línea 6) y XOR-ed resultado `ab` (línea 7). La reducción modular interpreta el resultado intermedio `ab` como numerador, recorre todos los bits del numerador, comenzando con el MSB (línea 10) y desplaza el módulo polinomial `mod_pol` (línea 12) detectando los bits del numerador, para realizar la operación XOR apropiada (línea 13).

```
1    %Multiplicación de polinomios
2    function ab = poly_mult(a, b, mod_pol)
3        ab = 0 ;
4        for i_bit = 1 : 8
5            if bitget(a, i_bit)
6                b_shitf = bitshift(b, i_bit - 1) ;
7                ab = bitxor(ab, b_shitf);
8            end
9        end
10
11       for i_bit = 16 : -1 : 9
12           if bitget(ab, i_bit)
13               mod_pol_shift = bitshift(mod_pol, i_bit - 9) ;
14               ab = bitxor(ab, mod_pol_shift) ;
15           end
16       end
```

Código 2.14. Función `poly_mult`

Para la función `cycle` se permuta cíclicamente las filas de la matriz de entrada. La primera fila no se desplaza mientras que los elementos de la segunda fila se desplazan

una posición a la derecha y los elementos de la tercera y cuarta fila se desplazan dos y tres posiciones a la derecha respectivamente como se observa en la Figura 2.24.

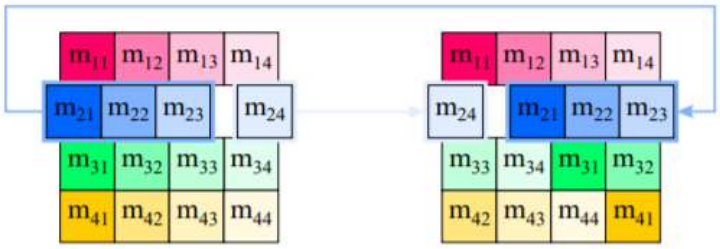

Figura 2.24. Proceso de la función `cycle` [29]

Los parámetros de entrada de la función `cycle` son la matriz (`matrix_in`) que debe permutarse y la dirección ("izquierda" o "derecha") en la que debe realizarse el cambio se puede observar en el Código 2.15. Se define un vector de columna (`col`) dependiente de la dirección de desplazamiento (líneas 2 a 6). Se define el vector de fila (`row`) (línea 8). Ambos vectores se expanden (se cuadriplican) en las matrices correspondientes (filas y columnas) (líneas 9 y 10). Se crea una matriz de índice (`ind_mat`) (línea 11) agregando ambas matrices y doblando hacia atrás a través del módulo operador (`mod`). La matriz de índice se aplica finalmente a la matriz que se va a permutar (línea 12), lo que da como resultado la matriz permutada `matrix_out`.

```
1    %Permutación de la matriz hacia la izquierda
2    function matrix_out = cycle (matrix_in, direction)
3        if strcmp (direction, 'left')
4            col = (0 : 5 : 15)';
5        else
6            col = (16 : -3 : 7)';
7        end
8        row = 0 : 4 : 12;
9        cols = repmat (col, 1, 4);
10       rows = repmat (row, 4, 1);
11       ind_mat = mod (rows + cols, 16) + 1;
12       matrix_out = matrix_in (ind_mat);
```

Código 2.15. Función `cycle`

La función `cipher` es la encargada de realizar el cifrado con la clave de 16 bytes, reordena el vector de texto plano en la matriz de estado y repite iterativamente el estado mediante las funciones mostradas en la figura 2.25.

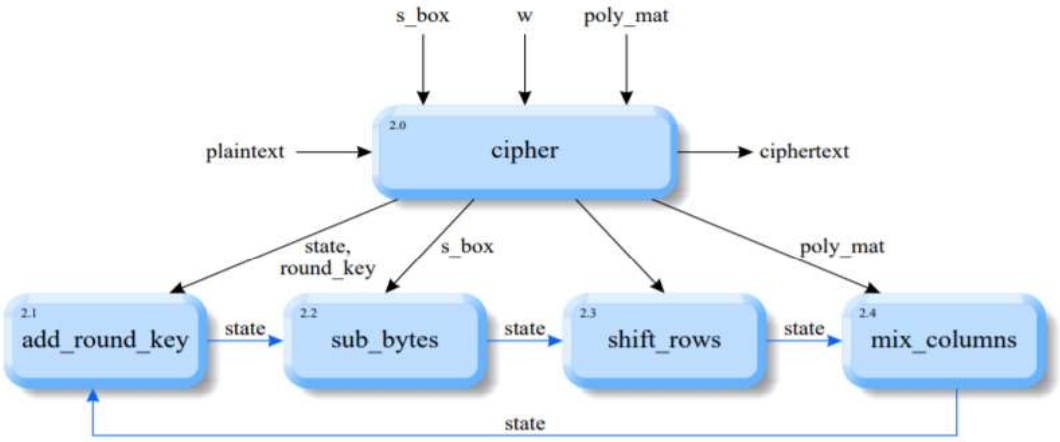

Figura 2.25. Función `cipher` [29]

La función `cipher` que se muestra en el Código 2.16 y una parte del proceso de la función en la Figura 2.26, se encarga de extraer la primera matriz 4 × 4 de la programación de la clave `w` (línea 12), para coincidir con la matriz de estado orientada a columnas. Esta primera ronda de la clave (`matrix`) debe transponerse y se agrega al estado inicial (línea 13). Las transformaciones de estado dentro del bucle (líneas 14 a 20) se repiten nueve veces. Consisten en una aplicación de la `s_box` (línea 15), una permutación cíclica de los elementos de la fila de estado (línea 16), una multiplicación de la matriz polinomial (línea 17), la extracción de la matriz de la clave de la ronda actual (línea 18) y la adición binaria de la clave de ronda (línea 19). La línea 16 remodela el estado final (es decir, el texto cifrado) de nuevo en un vector de fila larga de 16 bytes.

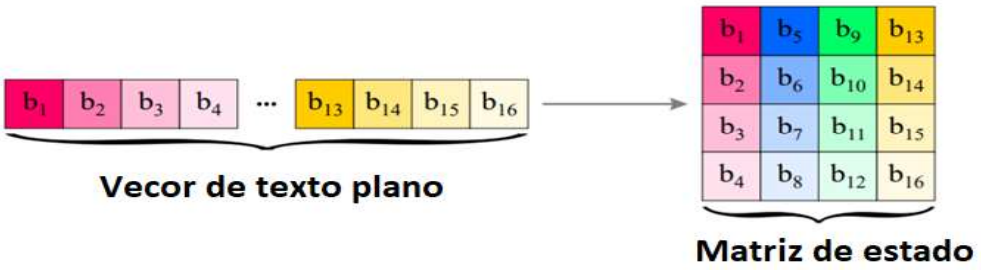

Figura 2.26. Remodelación de la columna del vector de fila de texto sin formato en la matriz de estado [29]

```
%Función de cifrado
function ciphertext = cipher (plaintext, w, s_box, poly_mat, round_num)
    switch round_num %casos
        case 10
            colomun = 4 ;
        case 12
        case 14
            colomun = 8 ;
        otherwise
            error('Número de ronda incorrecta')
    end
    state = reshape (plaintext, 4, 4);%inicio
    round_key = (w(1:4, :))';
    state = add_round_key (state, round_key);
    for i_round = 1 : round_num-1
        state = sub_bytes (state, s_box);
        state = shift_rows (state);
        state = mix_columns (state, poly_mat);
        round_key = (w((1:4) + 4*i_round, :))';
        state = add_round_key (state, round_key);
    end
    state = sub_bytes (state, s_box);
    state = shift_rows (state);
    round_key = (w((round_num+1)*4-3:(round_num+1)*4, :))';
    state = add_round_key (state, round_key);
    ciphertext = reshape (state, 1, 16);
    ciphertext = dec2hex(ciphertext);
```

Código 2.16. Función `cipher`

La función `add_round_key` declarada mostrada en el Código 2.17, realiza un XOR a nivel de bits de la matriz de estado y la matriz de clave redonda.

```
%Xor de matrices
function state_out = add_round_key (state_in, round_key)
    state_out = bitxor (state_in, round_key);
```

Código 2.17. Función `add_round_key`

La función `shift_rows` permuta cíclicamente desplazando las filas de la matriz de estado a la izquierda como muestra la Figura 2.27.

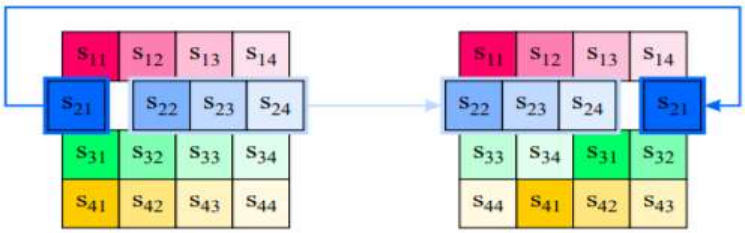

Figura 2.27. Permutación cíclica hacia la izquierda [29]

Por lo tanto, la función `shift_rows` llama a la función `cycle` con el parámetro de dirección de cambio apropiado ('izquierda') en la línea 2, como muestra el Código 2.18.

```
1     %Llamar a la función cycle con el parametro izquierda
2     function state_out = shift_rows (state_in)
3 -       state_out = cycle (state_in, 'left');
4
```

Código 2.18. Función `shift_rows`

La función `mix_columns` calcula la nueva matriz de estados `S'` multiplicando la matriz de estado actual `S` por la matriz polinomial `P` como se muestra en la Figura 2.28, teniendo en cuenta que la matriz polinomial `P` fue definida por la función `poly_mat_gen`.

$$\begin{bmatrix} s'_{11} & s'_{12} & s'_{13} & s'_{14} \\ s'_{21} & s'_{22} & s'_{23} & s'_{24} \\ s'_{31} & s'_{32} & s'_{33} & s'_{34} \\ s'_{41} & s'_{42} & s'_{43} & s'_{44} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \bullet \begin{bmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \\ s_{41} & s_{42} & s_{43} & s_{44} \end{bmatrix}$$

Figura 2.28. Cálculo de la matriz `S'` [29]

La función `mix_columns` mostrada en el Código 2.19, debe realizar una multiplicación de matriz-matriz completa que involucra los tres bucles anidados (líneas 4 a 6). Los parámetros de entrada de `mix_columns` son la matriz de estado (`state_in`) que se va a transformar y la matriz polinomial `P` (`poly_mat`), que se ha creado en `poly_mat_gen`. Se define el polinomio de módulo AES (línea 3) que se usará en la multiplicación polinomial interna (líneas 8 a 11). El doble bucle (líneas 4 y 5) aborda cada uno de los elementos de la matriz de estado transformada `S'` a través de sus índices de fila y columna. Se inicializa el estado *buffer* `temp_state` (línea 6) que contiene la suma acumulativa durante el cálculo de elementos. El bucle interno (líneas 7 a 13) recorre todos los sumandos de la matriz, almacena cada producto individual en `temp_prod` (líneas 8 a 11) y acumula (bit a bit XOR) el producto actual en el *buffer* de estado (línea 12), que finalmente se inserta en la matriz de estado transformada (`state_out`) (línea 14).

```
%Multiplicación de matrices
function state_out = mix_columns (state_in, poly_mat)
    mod_pol = bin2dec ('100011011');
    for i_col_state = 1 : 4
        for i_row_state = 1 : 4
            temp_state = 0;
            for i_inner = 1 : 4
                temp_prod = poly_mult (...
                    poly_mat(i_row_state, i_inner), ...
                    state_in(i_inner, i_col_state), ...
                    mod_pol);
                temp_state = bitxor (temp_state, temp_prod);
            end
            state_out(i_row_state, i_col_state) = temp_state;
        end
    end
```

Código 2.19. Función `mix_columns`

La función de descifrado `inv_cipher` invierte las transformaciones del proceso de cifrado. A continuación, en la Figura 2.29 se puede ver los parámetros de entrada al igual que el resultado que otorga, se debe tener en cuenta que si `w` que es la clave ingresada resultante de la función `key_expansion` es errónea se va a llevar a cabo el proceso de desencriptación, sin embargo, el resultado no será la información encriptada anteriormente.

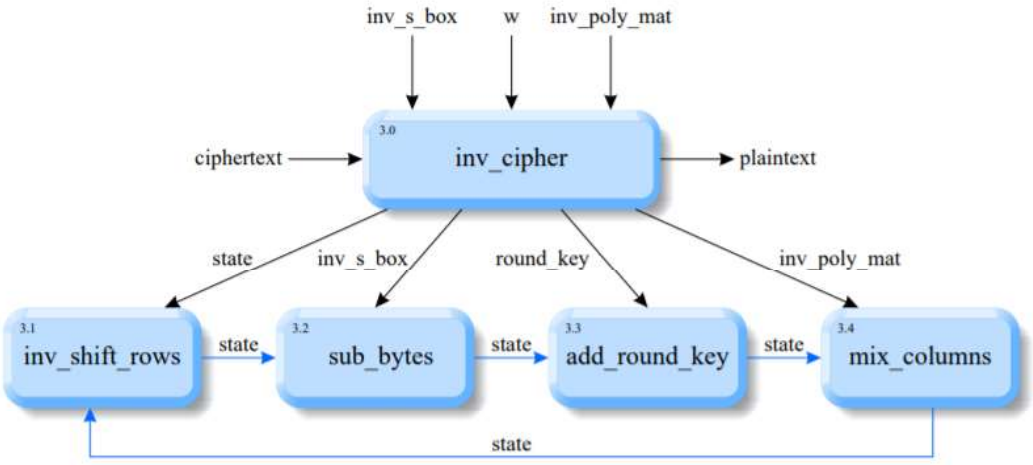

Figura 2.29. Función `inv_cipher` [29]

Se puede ver en el Código 2.20 que el texto cifrado se remodela en la matriz de estado (línea 3). La primera clave de ronda que se usará aquí (línea 5) es la última que se usó en el cifrado (`cipher`). Como consecuencia, la operación XOR (línea 6) invierte

directamente la última llamada de `add_round_key` en `cipher`. La última operación de `cipher` fue una llamada a `shift_rows`, cuyo efecto es neutralizado por la llamada a `inv_shift_rows` (línea 9). La transformación de `sub_bytes` (línea 10) utiliza la tabla de sustitución inversa `inv_s_box`, invirtiendo la sustitución en `cipher`. Lo mismo para la llamada a `mix_columns` (línea 13), que utiliza la matriz polinomial inversa `inv_poly_mat` y, por lo tanto, compensa la llamada correspondiente en `cipher`.

Como en `cipher`, pero en el orden cronológico opuesto, toma nueve rondas idénticas (línea 8) de desplazamiento de filas, sustitución de bytes y mezcla de columnas y una décima ronda final (de nuevo con una llamada de `mix_columns`) para terminar con el texto plano rearmado (línea 20).

```
2   function plaintext = inv_cipher (ciphertext, w, inv_s_box, inv_poly_mat, round_num)
3       state = reshape (ciphertext, 4, 4);
4       % Round inicial
5       round_key = (w( (round_num+1)*4-3:(round_num+1)*4, :))';
6       state = add_round_key (state, round_key);
7       % Noveno round
8       for i_round = round_num-1 : -1 : 1
9           state = inv_shift_rows (state);
10          state = sub_bytes (state, inv_s_box);
11          round_key = (w((1:4) + 4*i_round, :))';
12          state = add_round_key (state, round_key);
13          state = mix_columns (state, inv_poly_mat);
14      end
15      % Round Final
16      state = inv_shift_rows (state);
17      state = sub_bytes (state, inv_s_box);
18      round_key = (w(1:4, :))';
19      state = add_round_key (state, round_key);
20      plaintext = reshape (state, 1, 16);
```

Código 2.20. Función `inv_cipher`

La función `inv_shift_rows` se muestra en el Código 2.21, invierte el efecto de la función `shift_rows` correspondiente en el proceso de cifrado (`cipher`). Dado que `shift_rows` realiza desplazamientos a la izquierda, `inv_shift_rows` desplaza todas las filas de la matriz de estado (hacia atrás) hacia la derecha.

```
1   %Desplazamiento hacia la derecha
2   function state_out = inv_shift_rows (state_in)
3       state_out = cycle (state_in, 'right');
4
```

Código 2.21. Función `inv_shift_rows`

Para la facilidad en la programación las funciones más importantes se agruparon en la función `encryptData_AES128` que se puede ver en el Código 2.22, también se hace la validación de la contraseña para el autocompletado o recorte de la misma.

```
%Encriptar información
function [ ciphertext, strKey ] = encryptData_AES128( strData, varargin )
    aesLen = 16; %tamaño de clave
    round_num = 10;

    if nargin == 1
        strKey = '';
    else
        strKey = varargin{1};
    end
    %validación de la clave en tamaño
    [ key, strKey ] = validateKey( strKey, aesLen );
    [ s_box, ~, poly_mat, ~, w ] = aes_init( key, round_num );%inicialización de variables

    if isequal( class(strData), 'string' )%Transformación a vector de caracteres
        strData = char(strData);
    end
    data = double( strData(:) );

    N = length( data );
    m = mod( N, aesLen );
    if m > 0
        data = [ data; zeros( aesLen-m, 1 ) ];
        N = length( data );
    end

    rData = reshape( data, aesLen, [] );
    cData = zeros( size(rData) );
```

Código 2.22. Función `encryptData_AES128` (parte 1)

```
    end

    ciphertext = cData(:);

end
%Validación de clave
function [ key, strKey ] = validateKey( strKey, aesLen )

    if isempty( strKey )
        strKey = getRandomString( aesLen );
    end
    keyLen = length( strKey );
    if keyLen < aesLen
        strKey = [ strKey, getRandomString(  aesLen - keyLen ) ];
    end
    key = double( strKey(1:aesLen) )';

end
%Inicialización de variables
function [ s_box, inv_s_box, poly_mat, inv_poly_mat, w ] = aes_init( key, round_num )

    [s box, inv s box] = s box gen;
    rcon = rcon_gen(round_num);

    w = key_expansion (key, s_box, rcon, round_num);
    [poly_mat, inv_poly_mat]= poly_mat_gen;

end
```

Código 2.22. Función `encryptData_AES128` (parte 2)

En resumen, el algoritmo AES genera una matriz de sustitución `s_box` que es usada por la función `key_expansion` al igualmente que por la función de cifrado `cipher` para remplazar directamente cada byte por otro byte del mismo campo finito o campos de Galois. Se genera `w` que es la variable `output` de la función `key_espansion`, la cual toma la llave de 16 bytes suministrada por el usuario y utiliza una matriz constante anteriormente creada y la matriz de sustitución de `s_box` para generar una llave larga de 176 bytes que se denominará `w`. Y la matriz polinomial `poly_mat` que es usado en las funciones de mezclado de columnas `mix_columns` durante el cifrado.

Posteriormente, se guarda la llave larga de 176 bytes dentro de un vector de caracteres de números decimales que es pasado a la función de cifrado `cipher` junto con las variables generadas anteriormente por `aes_init`. El resultado es transformado a un vector de caracteres y guardado en la variable denominada a todas estas funciones están acopladas dentro de `decrypData_AES128` y `encryptData_AES128` [29].

Se debe tener en cuenta que el algoritmo para cifrar o descifrar es exactamente el mismo para mensajes introducidos por teclado o un archivo de texto .txt, ya que los dos serán transformados a bytes.

2.4.4 Algoritmo de ocultamiento

Para el algoritmo de ocultamiento se utilizó el algoritmo LSB, la lógica es similar para ocultar un mensaje de texto o un archivo de extensión .txt, ya que se usan las mismas funciones.

Una vez que se lee la imagen, esta es transformada en un vector de caracteres para usarla con facilidad, tanto el mensaje como el archivo .txt son transformados en bits e insertados en el vector de la imagen usando los bits escogidos por el usuario para realizar el ocultamiento, se usará banderas para tener una guía en los mensajes, estas banderas serán 64 ceros y 64 unos al inicio y final del mensaje. Para el uso de este algoritmo primero se establece una cantidad de bits máximo que se puede ingresar en la imagen, este valor dependerá del tamaño total de la imagen, es decir, la multiplicación de sus filas por sus columnas y por sus canales en caso de que sea de colores será por tres (rojo, verde y azul) mientras que si es a escala de grises será por 1, también se debe tener en cuenta que a este resultado se le resta 256 bits de las banderas colocadas para delimitar el mensaje.

Después se rearma la imagen ya con la información cifrada dentro de la misma. Así mismo para el proceso inverso se leen los bits que el usuario escoja del vector de la imagen y se extrae la información, teniendo en cuenta las banderas incluidas, los bits recuperados se convierten en caracteres, de igual manera el proceso es el similar para la extracción del mensaje o el archivo .txt.

Para que la aplicación sea más didáctica se colocó un *pop-up* menú para elegir en qué bit se desea introducir la información, de esta manera si se escoge entre los bits menos significativos no se notará una gran diferencia, pero si se coloca en los bits más significativos de la imagen se verá un cambio bastante notorio en la estegoimagen, este cambio dependerá de la cantidad de información introducida.

Para realizar este proceso de creó la función `CifraTexto` y `CifrarArchivosDeTexto` que realizará todas las operaciones de ocultamiento y encriptación.

2.4.5 Cifrar texto en una imagen

Primero se programó el botón para seleccionar una imagen, la programación del mismo se mostrará en el Código 2.23. Este botón es el encargado de permitir escoger los formatos de las imágenes (línea 88) y guardar el *path* de la misma para abrirla. También es el encargado de calcular la cantidad máxima de bits con la ayuda de la función `imgMaxData` que se muestra en el Código 2.24,

Esta función multiplica las dimensiones de la imagen y sus canales (RGB o grises), resta 256 bits (línea 5) de las banderas a ingresar y despliega el resultado en un cuadro de texto. La variable `nBits` (línea 4) es la cantidad de bits que se desea ocupar de cada byte.

Dispone de 4 botones, 1 *pop-up* menú, 5 cuadros de texto.

Botones:

- `Seleccionar Imagen`
- `Cifrar`
- `Visualizar`

- `Volver al Menú`

Cuadros de Texto:

- Para desplegar el máximo de bits
- Para introducir el mensaje a ocultar
- Para introducir la contraseña
- Para mostrar fragmento de mensaje encriptado
- Para mostrar si el proceso fue fallido o exitoso

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%Seleccionar Imagen
[ file, path ] = uigetfile({'*.jpg;*.png'},'Seleccione una Imaen');
if isequal( file, 0 )
    handles.img = [];
    set( handles.mensaje1, 'string', '...' );
    assignin('base','img',[]);
    set( handles.cifrar, 'enable', 'off' );
else
    imgPath = fullfile( path, file );
    img = imread( imgPath );
    maxBytes = imgMaxData( img );%calculo del total de bits a ingresar
    handles.img = img;
    handles.file = file;
    set( handles.mensaje1, 'string', num2str(maxBytes) );
    assignin('base','img',img);
    set( handles.cifrar, 'enable', 'on' );
end
set( handles.fin1, 'string', 'Texto sin Cifrar' )
set( handles.fin21, 'string', '' )
set( handles.buttonVisualiseMe, 'enable', 'off' );
guidata(hObject, handles);

return;
```

Código 2.23. Botón `seleccionar imagen`

```
%Calculo del total de bits
function maxBytes = imgMaxData( img )

    nBits = 1;
    maxBytes = numel( img ) * nBits / 8 - 256;

end
```

Código 2.24. Función `imgMaxData`

Una vez introducida la imagen se habilitará el botón `cifrar`, para la programación de este botón se muestra el Código 2.25, tiene como parámetros de entrada la imagen (línea 156) ingresada, el mensaje de texto ingresado por teclado (línea 157), la contraseña (línea 164) y el bit (línea 163) en cual se desea ocultar la imagen. También se hace uso de la función `saveDataIntoImage` para el algoritmo LSB (línea 166), esta función se muestra en el Código 2.26 recibe como parámetros de entrada el mensaje, la imagen y el bit donde ocultar la imagen (línea 2), se analiza el bit elegido por el usuario con la variable `bitPos` (línea 10 a 26), se transforma el mensaje ingresado a bits (línea 28), se colocan las banderas (64 unos y 64 ceros) antes y después del mensaje (línea 31), se procede con el algoritmo de ocultamiento (línea 42 a 44). Si el resultado se realizó con éxito se desplegará un mensaje en un cuadro de texto que diga Cifrado Exitoso.

Se debe tener en cuenta las condiciones que existen para la contraseña en AES. Se utiliza la función `saveImage` que se muestra en el Código 2.27, esta función permite guardar las imágenes resultantes en los diferentes formatos JPEG *lossless* y PNG (líneas 2 a 5).

```
153    % eventdata  reserved - to be defined in a future version of MATLAB
154    % handles    structure with handles and user data (see GUIDATA)
155    %Boton Cifrar
156 -  img = handles.img;
157 -  msg = get( handles.mensajepc, 'string' );
158
159 -  if isempty( msg ) || isempty(img)
160 -      return
161 -  end
162    %Tomo el resultado del bit donde se desea guardar la imagen
163 -  bitPos = get( handles.cbxBit, 'value' );
164 -  strKey = get( handles.password, 'string' );
165 -  [ ciphertext, strKey ] = encryptData_AES128( msg, strKey );
166 -  modImg = saveDataIntoImage( ciphertext, img, bitPos );
167
168 -  set( handles.password, 'string', strKey );
169 -  set( handles.fin21, 'string', char(ciphertext') );
170
171    assignin('base','modImg',modImg);
172
173 -  handles.modImg = modImg;
174 -  handles.bitPos = bitPos;
175
176 -  set( handles.fin1, 'string', 'Cifrado Exitoso' )
177 -  set( handles.buttonVisualiseMe, 'enable', 'on' );
178
```

Código 2.25. Botón `cifrar`

```
%Guardar la información dentro de la imagen
function modImg = saveDataIntoImage( data, img, varargin )

    if nargin == 3
        bitPos = varargin{1};
    else
        bitPos = 1;
    end
    %Analisis del bit donde ocultar
    if isa( bitPos, 'string' ) || isa( bitPos, 'char' )
        switch char(bitPos)
            case {'MSB','msb'}
                bitPos = 8;
            case {'LSB','lsb'}
                bitPos = 1;
            otherwise
                bitPos = str2double( bitPos );
        end
    end
    %Analisis del bit donde ocultar
    if bitPos < 0 || bitPos > 8
        error('BitPos debe tener valores enteros entre 0 y 8');
    end

    N = numel( img ) * bitPos;
    I = img(:);
    %Transformación del mensaje en bits
```

Código 2.26. Función `saveDataIntoImage` (parte 1)

```
    %Transformación del mensaje en bits
    binData = de2bi( data, 8 );
    bData = reshape( binData, [], 1 );
    %Bandera a introducir
    markSize = 64;
    mark = [ zeros( markSize, 1 ); ones( markSize, 1 )];

    bData = [ mark; bData; mark ];
    nData = length(bData);
    %Condicion para que no sobrepase la cantidad máxima
    if nData > N
        modImg = [];
        return;
    end
    %Rearmado de la imagen
    binI = de2bi( I, 8 );
    binI( 1:nData, bitPos ) = bData;
    modImg = reshape( bi2de(binI), size(img) );

end
```

Código 2.26. Función `saveDataIntoImage` (parte 2)

```
%guardar la imagen en diferentes formatos
function saveImage( fileName, img )
    imwrite( img, strcat(fileName,'.jpg'), 'mode', 'lossless' );%formato jpeg lossless
    imwrite( img, strcat(fileName,'.png') );%formato PNG
end
```

Código 2.27. Función `saveImage`

Se realiza la programación del botón `Volver al Menú` en donde se cierra el formulario actual y se abre el módulo principal, esto se muestra en el Código 2.28.

```
198   % --- Executes on button press in Volver_Menu.
199   function Volver_Menu_Callback(hObject, eventdata, handles)
200   % hObject    handle to Volver_Menu (see GCBO)
201   % eventdata  reserved - to be defined in a future version of MATLAB
202   % handles    structure with handles and user data (see GUIDATA)
203   %Boton volver al menú
204 -  close('gcf')
205 -  openfig('Menu.fig')
206
```

Código 2.28. Botón `volver al menú`

Finalmente, se colocó el botón `visualizar` para que el usuario pueda ver las imágenes resultantes.

2.4.6 Cifrar archivo de texto en una imagen

Este formulario es muy similar a `CifrarTexto`, la diferencia es que en `CifrarTexto` el mensaje a ocultar es introducido mediante un cuadro de texto por teclado y en `CifrarArchivoDeTexto` se tiene un botón para elegir el archivo de texto .txt que se desea ocultar.

Para la programación se usa la función principal: `CifrarArchivoDeTexto` en donde se programa cada uno de los objetos requeridos mediante sus *callbacks*.

Dispone de 5 botones, 1 *pop-up* menú, 4 cuadros de texto.

Botones:

- `Seleccionar Imagen`
- `Seleccionar archivo de texto`
- `Cifrar`
- `Visualizar`
- `Volver al Menú`

Cuadros de Texto:

- Para desplegar el máximo de bits
- Para introducir la contraseña
- Para mostrar fragmento de mensaje encriptado
- Para mostrar si el proceso fue fallido o exitoso

La programación del botón `Seleccionar Imagen` se puede ver en el Código 2.29. Aquí se lee la imagen desde un *path* seleccionado y se guarda en la variable img (línea 89), se emplea nuevamente el uso de la función `imgMaxData` que calcula el tamaño máximo de bits que se pueden introducir dependiendo de la cantidad de bits deseados, es decir, este valor varía si la cantidad de bits a usar es mayor que 1, sin embargo, se recomienda ocultar la información únicamente en un bit de la imagen. Los despliega en el cuadro de texto respectivo (línea 96), se habilita el botón `cifrar` si se eligió una imagen (línea 92).

```
85    % handles    structure with handles and user data (see GUIDATA)
86    %Boton selecionar imagen
87 -  [ file, path ] = uigetfile({'*.jpg;*.png'},'Seleccione una Imagen'); %permite seleccionar los formatos establecidos
88 -  if isequal( file, 0 )
89 -      handles.img = [];%almacena la imagen en la variable img
90 -      set( handles.p1, 'string', '...' );
91 -      assignin('base','img',[]);
92 -      set( handles.cifrarArch, 'enable', 'off' );%habilita el boton cifrar
93 -  else
94 -      imgPath = fullfile( path, file );
95 -      img = imread( imgPath );
96 -      maxBytes = imgMaxData( img );%calcuula el tamaño máximo de bits a ocultar
97 -      handles.img = img;
98 -      handles.file = file;
99 -      set( handles.p1, 'string', num2str(maxBytes) );
100 -     assignin('base','img',img);
101
102 -     if ~isempty( handles.txt )
103 -         set( handles.cifrarArch, 'enable', 'on' );%si no se elige la imagen el boton cifrar permanece desactivado
104 -     end
105
106 -     set( handles.cifrarArch, 'enable', 'on' );
107 -  end
108 -  set( handles.bottonVisualiseMe2, 'enable', 'off' );
109 -  set( handles.fin25, 'string', '' );
110 -  guidata(hObject, handles);
111
112 -  return;
113
```

Código 2.29. Botón `seleccionar imagen`

Para el botón `Archivo de texto a cifrar` se muestra el Código 2.30, aquí se programa el botón para que permita elegir archivos con extensión .txt (línea 124), una vez ingresado el archivo se concatena toda la información que contiene y se lo pasa a un vector de caracteres (líneas 129 a 132), se mantiene los botones correspondientes encendidos o apagados dependiendo del caso (líneas 127 y 139).

```
117
118     % --- Executes on button press in pushbutton4.
119   function pushbutton4_Callback(hObject, eventdata, handles)
120   % hObject    handle to pushbutton4 (see GCBO)
121   % eventdata  reserved - to be defined in a future version of MATLAB
122   % handles    structure with handles and user data (see GUIDATA)
123   %Boton oclcccionar archivo dc texto
124 -  [ file, path ] = uigetfile('*.txt','Seleccione un Archivo de Texto');%formato para el archivo de texto
125 -  if isequal( file, 0 )
126 -      handles.txt = [];
127 -      set( handles.cifrarArch, 'enable', 'off' );
128 -  else
129 -      txtPath = fullfile( path, file );%selecciona todo el archivo y lo une en un vector de caracteres
130 -      txt = importdata( txtPath );
131 -      txt = strjoin( txt, '\n' );
132 -      handles.txt = txt;
133
134 -      if ~isempty( handles.img )
135 -          set( handles.cifrarArch, 'enable', 'on' );
136 -      end
137
138 -  end
139 -  set( handles.bottonvisualiseme2, 'enable', 'off' );
140 -  set( handles.fin25, 'string', '' );
141 -  guidata(hObject, handles);
142
143 -  return;
```

Código 2.30. Botón `archivo de texto a cifrar`

Para el botón `cifrar` se muestra el Código 2.31, este botón se programó para que reciba los argumentos de entrada de: la imagen introducida, el archivo .txt introducido, el bit en el que desea ingresar la información dentro de una imagen y la contraseña, se coloca una bandera adicional en la concatenación del archivo que diga `isFile==true`, con el objetivo de que al momento de descifrar se pueda saber que fue el resultado de un archivo de texto (línea 155), se analiza cuál fue el bit que eligió el usuario para ocultar la imagen (línea 161), se llaman a las funciones para cifrar el texto y para ocultar en texto (líneas 163 y 164), se muestra un mensaje de cifrado exitoso si todo los procesos salieron sin error (línea 169), finalmente se guardan las imágenes empleando el uso de la función `saveImage` (línea 176)

```
152   %botón cifrar
153 -  img = handles.img;
154 -  msg = handles.txt;
155 -  msg = strcat('|isFile==true|',msg);%se coloca una bandera adciional apra reconocer que es un archivo
156
157 -  if isempty( msg ) || isempty(img)
158 -      return
159 -  end
160
161 -  bitPos = get( handles.cbxBit, 'value' );%se analiza el bit elegido para ocultar la información en la imagen
162 -  strKey = get( handles.password, 'string' );
163 -  [ ciphertext, strKey ] = encryptData_AES128( msg, strKey );%se llama a las funciones para cifrar con AES
164 -  modImg = saveDataIntoImage( ciphertext, img, bitPos );%se llama a la funcion apra aplciar el algoritmo LSB
165
166 -  set( handles.password, 'string', strKey );
167 -  set( handles.fin25, 'string', char(ciphertext(1:16)') );
168
169 -  set( handles.fin2, 'string', 'Cifrado Exitoso' )
170 -  assignin('base','modImg',modImg);
171
172 -  handles.modImg = modImg;
173 -  handles.bitPos = bitPos;
174
175 -  set( handles.bottonVisualiseMe2, 'enable', 'on' );
176 -  saveImage( 'ImagenConMensajeDeTexto', modImg );%se llama a la función saveImage para guardar en diferentes formatos
177
178 -  guidata(hObject, handles);
179
180 -  return;
```

Código 2.31. Botón `cifrar`

Se programa el botón `volver al menú` de la misma forma que en el anterior módulo, esto se muestra en el Código 2.32.

```
207    % --- Executes on button press in Volver_Menu.
208    function Volver_Menu_Callback(hObject, eventdata, handles)
209    % hObject    handle to Volver_Menu (see GCBO)
210    % eventdata  reserved - to be defined in a future version of MATLAB
211    % handles    structure with handles and user data (see GUIDATA)
212    %Boton volver al manú
213 -  close('gcf')
214 -  openfig('Menu.fig')
215
```

Código 2.32. Botón `volver al menú`

Finalmente, se colocó el botón `visualizar` para que el usuario pueda ver las imágenes resultantes.

2.4.7 Entregables

Los entregables del segundo *sprint* son los módulos secundarios: `Cifrar texto en imagen` y `Cifrar archivo de texto en imagen` y se presentan a continuación en la Figura 2.30 y 2.31 respectivamente.

Figura 2.30. Pantalla para `cifrar texto en imagen` del entregable del segundo *sprint*

Figura 2.31. Pantalla para `cifrar archivo de texto en imagen` del entregable del segundo *sprint*

2.4.8 Revisión y retrospectiva

Los cambios que se realizaron a lo largo del desarrollo de este *sprint* fueron:

1. En un inicio no se tenía el *pop-up* menú para elegir el bit en donde se desea ingresar la información oculta, luego se decidió aumentarlo para métodos didácticos de la aplicación.
2. Inicialmente no se desplegaba un cuadro de texto con la información cifrada, sin embargo, se decidió colocar para fines didácticos de la aplicación.

2.5 *Sprint* visualización de imágenes y recuperación de información

2.5.1 Interfaz gráfica

Para este *sprint* se ha considerado un tercer módulo `visualizar` y un módulo secundario `descifrar`. El *sketch* de los formularios se muestra en las Figuras 2.32 y 2.33.

Figura 2.32. *Sketch* para descifrar la información

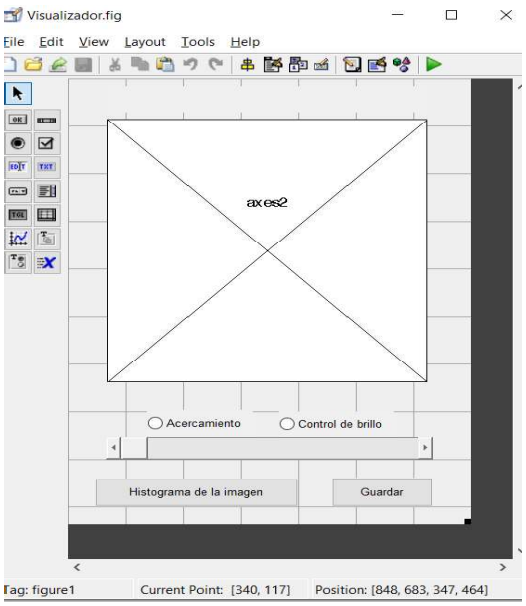

Figura 2.33. *Sketch* para visualizar las imágenes resultantes

En la Figura 2.34 se puede observar el diagrama de bloques para `descifrar`.

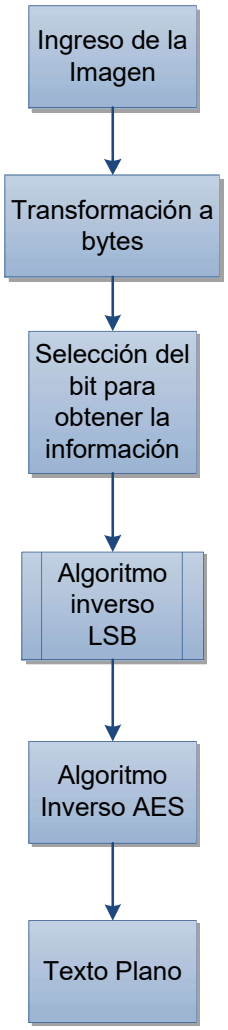

Figura 2.34. Diagrama de bloques de la función `descifrar`

2.5.2 Desarrollo

Para este *sprint* se ha programado un formulario secundario y un formulario ternario para `descifrar` y `visualizar` respectivamente.

2.5.3 Módulo secundario descifrar

Para este módulo se puede acceder desde el menú principal, consta de 3 botones, 3 cuadros de texto y 1 *pop-up* menú

Botones:

- `Seleccionar imagen a descifrar`

- `Descifrar`
- `Volver al menú`

Cuadros de texto

- Para ingresar la contraseña
- Para mostrar la información
- Para indicar si el descifrado fue exitoso

La función principal `Descifrar` contiene toda la programación de los objetos antes mencionados, teniendo en cuenta que se programaron en sus *callbacks*, para el botón `Seleccionar imagen a descifrar` se mostrará en el Código 2.33, se coloca para que lea los formatos JPEG y PNG (línea 86), se activa el botón `descifrar` (línea 89), se lee la imagen y se guarda en la variable `modImg` (líneas 97 y 98).

```
81  % hObject    handle to pushbutton9 (see GCBO)
82  % eventdata  reserved - to be defined in a future version of MATLAB
83  % handles    structure with handles and user data (see GUIDATA)
84  %Boton seleccioanr imagen para descifrar
85
86  [ file, path ] = uigetfile({'*.jpg;*.png'},'Seleccione una Imaen');%formatos para elegir la imagen
87  set( handles.fin11, 'string', '' );
88  if isequal( file, 0 )
89      set( handles.pushbutton10, 'enable', 'off' ); %activar el boton descifrar
90      set( handles.text12, 'string', 'Imagen sin Decifrar' );
91      return;
92  end
93
94  set( handles.pushbutton10, 'enable', 'on' );
95
96  imgPath = fullfile( path, file );
97  modImg = imread( imgPath );%leer la imagen
98  handles.modImg = modImg;% guardar la imagen en la variable modImg
99  handles.file = file;
100 set( handles.text12, 'string', 'Imagen sin Decifrar' );%mantener el cuadro de texto como imagen sin descifrar
101 guidata(hObject, handles);
102
103 return;
```

Código 2.33. Botón `seleccionar imagen a descifrar`

Para el botón `descifrar` se muestra el Código 2.34, este botón toma como parámetros de entrada la imagen ingresada, la contraseña ingresada y la selección del bit donde se encuentra la imagen (líneas 113 a 115), llama a la función inversa de LSB `readDataIntoImage` (línea 118), si la imagen está vacía muestra el mensaje descifrado fallido (línea 123), finalmente, llama al algoritmo inverso AES que es la función `decrypData_AES128` (línea 128)

En el código 2.35 se muestra la función inversa del algoritmo LSB llamada `readDataIntoImage`, esta función se encarga de analizar el bit en donde se encuentra la información (líneas 5 y 10), analiza las banderas ingresadas (líneas 32 y 40), recupera

la información de la imagen (líneas 25,26 y 42) y finalmente rearma la información y la imagen.

```
111  % handles    structure with handles and user data (see GUIDATA)}
112  % boton descifrar
113 - bitPos = get( handles.cbxBit, 'value' ); %analiza el bit donde se encuentra la información
114 - modImg = handles.modImg;%obtiene la imagen ingresada
115 - strKey = get( handles.password, 'string' );% obtiene la clave ingresada
116
117 - try
118 -     moddata = readDataIntoImage( modImg, bitPos );%algoritmo inverso LSB
119 - catch
120 -     moddata = [];
121 - end
122
123 - if isempty(moddata)%si la imagen esta vacia muestra decifrado fallido
124 -     set( handles.fin11, 'string', '' );
125 -     set( handles.text12, 'string', 'Decifrado Fallido' );
126 -     return;
127 - end
128 - strData = decryptData_AES128( double(moddata), strKey );% algoritmo aes inverso
129 - if isempty(strData)
130 -     set( handles.fin11, 'string', '' );
131 -     set( handles.text12, 'string', 'Decifrado Fallido' );
132 -     return;
133 - end
```

Código 2.34. Botón `descifrar`

```
1   %Función Inversa de LSB
2   function data = readDataIntoImage( modImg, varargin )
3
4 -     if nargin == 2
5 -         bitPos = varargin{1};%analisa en que bit se encuentra la imagen
6 -     else
7 -         bitPos = 1;
8 -     end
9
10 -    if isa( bitPos, 'string' ) || isa( bitPos, 'char' )%analisa en que bit se encuentra la imagen
11 -        switch char(bitPos)
12 -            case {'MSB','msb'}
13 -                bitPos = 8;
14 -            case {'LSB','lsb'}
15 -                bitPos = 1;
16 -            otherwise
17 -                bitPos = str2double( bitPos );
18 -        end
19 -    end
20
21 -    if bitPos < 0 || bitPos > 8
22 -        error('BitPos debe tener valores enteros entre 0 y 8');
23 -    end
```

Código 2.35. Función `readDataIntoImage` (parte 1)

```
27
28 -        bData = binI( :, bitPos );
29 -        bData = reshape( bData, [], 1 );
30
31 -        markSize = 64;
32 -        mark = [ zeros( markSize, 1 ); ones( markSize, 1 )]; %analisa las banderas
33
34 -        index = strfind( bData', mark' );
35 -        if length(index) < 2
36 -            data = [];
37 -            return;
38 -        end
39
40 -        bData = bData( 2*markSize + 1 : index(2) - 1 );
41
42 -        if mod( length(bData), 8 ) == 0
43 -            binData = reshape( bData, [], 8 );
44 -            data = bi2de( binData );%rearma la información
45
46 -        else
47 -            data = [];
48 -        end
49
50 -    end
```

Código 2.35. Función `readDataIntoImage` (parte 2)

La función `decryptData_AES12` se muestra en el Código 2.36, esta función es la encargada de descifrar la información obtenida por el algoritmo inverso LSB, con la contraseña indicada, cumple las siguientes condiciones:

- Si la contraseña es del tamaño de 16 bytes, pero es incorrecta descifra la información, pero entrega incongruencias.
- Si la contraseña es menor o mayor a 16 bytes entrega el mensaje descifrado fallido
- Si la contraseña es la correcta despliega la información ocultada en texto plano

En la programación se llama a las funciones inversa de AES (líneas 20 a 28), se analiza la clave ingresada (línea 30) y se realizan los algoritmos inversos.

```
19
20 -    for i = 1:nFrames
21 -        rData(:,i) = inv_cipher ( cData(:,i), w, inv_s_box, inv_poly_mat, round_num )';%funciones inversas de AES
22 -    end
23
24 -    strData = rData(:)';
25 -    strData( strData == 0 ) = [];
26 -    strData = char( strData );
27
28 - end
29
30 - function [ key, strKey ] = validateKey( strKey, aesLen )%analisis de la clave y cumplimiento de las condiciones
31
32 -    printableChars = [ 32 126 ];
33
34 -    if isempty( strKey )
35 -        strKey = char( randi( printableChars, 1, aesLen ) );
36 -    end
37 -    keyLen = length( strKey );
38 -    if keyLen < aesLen
39 -        strKey = [ strKey, char( randi( printableChars, 1, aesLen - keyLen ) ) ];
40 -    end
41 -    key = double( strKey(1:aesLen) )';
```

Código 2.36. Función `desencryptData_AES128`

2.5.4 Visualizar

El módulo ternario usa la función `visualizador` y se abre desde cualquier módulo secundario `cifrar`, se muestra el botón `visualizar` en el Código 2.37, en donde abre 4 formularios con 2 imágenes originales, una JPEG y otra PNG y 2 estegoimagenes, una JPEG y una PNG.

```
208     % --- Executes on button press in buttonVisualiseMe.
209   function buttonVisualiseMe_Callback(hObject, eventdata, handles)
210   % hObject    handle to buttonVisualiseMe (see GCBO)
211   % eventdata  reserved - to be defined in a future version of MATLAB
212   % handles    structure with handles and user data (see GUIDATA)
213
214 -   saveImage( 'ImagenOriginal', handles.img );
215 -   pause(0.2)
216 -   Visualizador( 'ImagenOriginal.png', handles.bitPos, 1 );
217 -   pause(0.2)
218 -   Visualizador( 'ImagenOriginal.jpg', handles.bitPos, 2 );
219 -   pause(0.2)
220 -   Visualizador( 'ImagenConMensaje.png', handles.bitPos, 3 );
221 -   pause(0.2)
222 -   Visualizador( 'ImagenConMensaje.jpg', handles.bitPos, 4 );
223
```

Código 2.37. Botón `visualizar`

El nuevo formulario ternario consta de: dos *push button* uno para zoom y otro para brillo, un *slider* para aumentar o disminuir y dos botones para visualizar el histograma y para guardar la imagen con los cambios. Se muestra el Código 2.38 la programación del *slider* para que aumente en zoom o brillo, para el zoom se recorta los perfiles de la matriz de la imagen para agrandarla mientras que para el brillo se cambia los valores de los bits para atenuarlos (líneas 155 a 160).

```
41      %Slider
42 -    n=get(handles.zoom2,'value');
43 -    i=get(handles.brillo2,'value');
44 -    m=get(hObject,'value');
45      % imag2 = Axes2;
46
47 -    imag2 = handles.img;
48
49 -    if(i==1)
50 -        imag2=imag2+(2*m);
51      %     imshow(imag2);
52 -        imshow( imag2, 'parent', handles.axes2 );
53 -    end
54
55 -    if(n==1)
56 -        f=size(imag2);
57 -        imag2=imag2(floor((m/100)*f(1)):f(1),floor((m/100)*f(2)):f(2),:);
58      %     imshow(imag2);
59 -        imshow( imag2, 'parent', handles.axes2 );
60 -    end
61
62 -    handles.imag2 = imag2;
63 -    guidata(hObject, handles);
```

Código 2.38. *Slider*

Para el histograma se muestra el código 2.39 en donde se llama a las funciones de *Matlab* para mostrar el histograma de la imagen actual (líneas 185 a 194).

```
179 % hObject    handle to pushbutton1 (see GCBO)
180 % eventdata  reserved - to be defined in a future version of MATLAB
181 % handles    structure with handles and user data (see GUIDATA)
182 % global Axes2
183
184 %mostrar histograma de la imagen
185 img2 = handles.imag2;
186 figure
187 hist1=imhist(img2);
188 imhist(img2);
189 title( sprintf('Histograma: %s', handles.name ) );
190 p1=hist1./(size(img2,1)*size(img2,2));
191 p1=p1(p1>0);
192 H1=-sum(p1.*(log2(p1)));
193 H1(isnan(H1))=0;
194 grid on
```

Código 2.39. Botón `mostrar histograma`

Finalmente, para el botón `guardar` se muestra el Código 2.40 y es el encargado de guardar la imagen con los cambios respectivos (líneas 207 a 225).

```
205 % imwrite(imag2,'ImagenCifradaVisualizada.png');%Se guarda la imagen actu
206
207 imag2 = handles.imag2;
208
209 if isempty(handles.data)
210     modImg = imag2;
211 else
212     modImg = saveDataIntoImage( handles.data, imag2, handles.bitPos );
213 end
214
215 strArray = strsplit( handles.name, '.' );
216 name = strArray{1};
217 ext = strArray{end};
218
219 imageName = strcat( name, '_Visualizada.', ext );
220
221 if strcmpi( ext, 'png' )
222     imwrite( modImg, imageName );
223 elseif strcmpi( ext, 'jpg' )
224     imwrite( modImg, imageName, 'mode', 'lossless' );
225 end
```

Código 2.40. Botón `guardar`

Se debe tener en cuenta que todas las imágenes guardadas en JPEG están en formato *lossless* por lo cual necesitan una aplicación especial para abrirlas.

2.5.5 Entregables

Para el tercer *sprint* se entregó el formulario secundario `descifrar` y el formulario ternario `visualizar`, que se muestran en las Figuras 2.35 y 2.36, respectivamente.

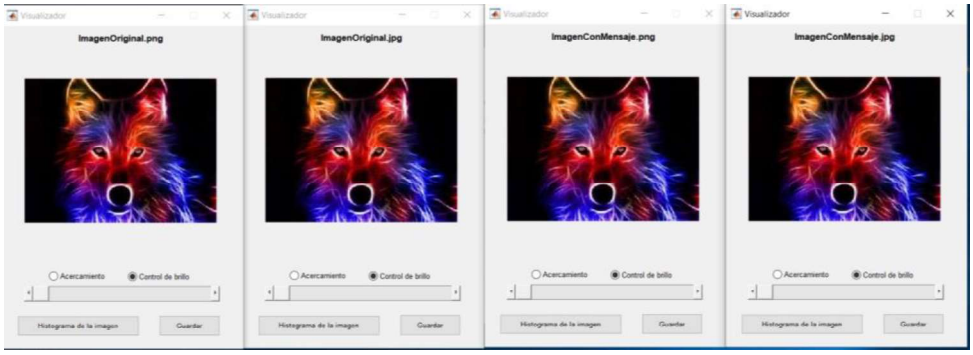

Figura 2.35. Módulo `visualizar`, entregable *sprint* 3

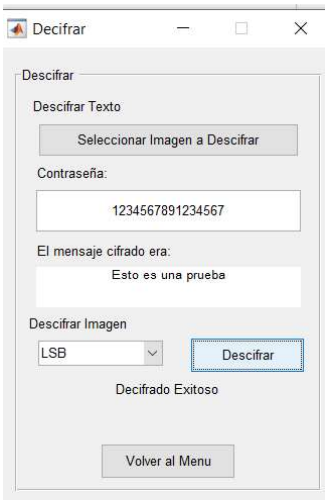

Figura 2.36. Módulo `descifrar`, entregable *sprint* 3

2.6 *Sprint* visualización de imágenes y recuperación de información

2.6.1 Interfaz gráfica

Para este *sprint* se ha considerado una aplicación que pueda abrir imágenes jpeg *lossless*, que pueda mostrar dos imágenes simultáneas para ver sus diferencias mediante la transformada DCT y la resta de las imágenes.

2.6.2 Diseño

Esta aplicación fue programada en *Matlab* como un ejecutable, en la Figura 2.37 se muestra el ambiente de la aplicación para visualizar las imágenes.

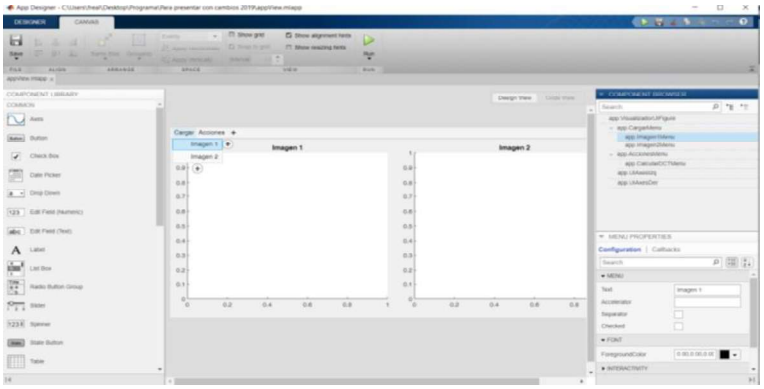

Figura 2.37. Prototipo de la aplicación para JPEG *lossless*

Se programaron los dos *axes* para que puedan leer las imágenes que el usuario desee abrir, el `menú` para abrir dichas imágenes es llamado `Cargar Menú` en donde se despliega las dos opciones y el menú acciones que me permite calcular la transformada DCT y la resta de las imágenes en formularios nuevos como se observa en el Código 2.41.

```
classdef appView < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        VisualizadorUIFigure  matlab.ui.Figure
        CargarMenu            matlab.ui.container.Menu
        Imagen1Menu           matlab.ui.container.Menu
        Imagen2Menu           matlab.ui.container.Menu
        AccionesMenu          matlab.ui.container.Menu
        CalcularDCTMenu       matlab.ui.container.Menu
        UIAxesIzq             matlab.ui.control.UIAxes
        UIAxesDer             matlab.ui.control.UIAxes
    end


    properties (Access = private)
        imgIzq % Description
        imgDer % Description
        nameIzq % Description
        nameDer % Description
    end


    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            cla( app.UIAxesDer )
            cla( app.UIAxesIzq )

            app.imgIzq = [];
```

Código 2.41. Aplicación para visualizar las imágenes con sus diferencias (parte 1)

```
        % Menu selected function: Imagen1Menu
        function Imagen1MenuSelected(app, event)
            [ file, path ] = uigetfile({'*.jpg;*.bmp;*.png'},'Seleccione una Image
            if isequal( file, 0 )
                return;
            end

            imgPath = fullfile( path, file );

            app.nameIzq = file;
            app.imgIzq = imread( imgPath );

            imshow( app.imgIzq, 'Parent', app.UIAxesIzq );
            title( app.UIAxesIzq, app.nameIzq, 'interpreter', 'none' )

            if isempty( app.imgIzq ) || isempty( app.imgDer )
                app.CalcularDCTMenu.Enable = 'off';
            else
                app.CalcularDCTMenu.Enable = 'on';
            end
        end

        % Menu selected function: Imagen2Menu
        function Imagen2MenuSelected(app, event)
            [ file, path ] = uigetfile({'*.jpg;*.bmp;*.png'},'Seleccione una Image
            if isequal( file, 0 )
                return;
            end

            imgPath = fullfile( path, file );
```

Código 2.41. Aplicación para visualizar las imágenes con sus diferencias (parte 2)

```
        % Menu selected function: CalcularDCTMenu
        function CalcularDCTMenuSelected(app, event)
            figure

            ax = subplot(122);
            J = dct2( rgb2gray( app.imgDer) );
            imshow(log(abs(J)),[])
            title( ax, app.nameDer, 'interpreter', 'none' )
            colormap(gca,jet(64))
            colorbar

            ax = subplot(121);
            J = dct2( rgb2gray( app.imgIzq) );
            imshow(log(abs(J)),[])
            title( ax, app.nameIzq, 'interpreter', 'none' )
            colormap(gca,jet(64))
            colorbar
        end
    end

    % App initialization and construction
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create VisualizadorUIFigure
            app.VisualizadorUIFigure = uifigure;
            app.VisualizadorUIFigure.Position = [100 100 842 369];
            app.VisualizadorUIFigure.Name = 'Visualizador';
```

Código 2.41. Aplicación para visualizar las imágenes con sus diferencias (parte 3)

```
            app.UIAxesDer.Position = [431 20 400 340];
        end
    end

    methods (Access = public)

        % Construct app
        function app = appView

            % Create and configure components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.VisualizadorUIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.VisualizadorUIFigure)
        end
    end
end
```

Código 2.41. Aplicación para visualizar las imágenes con sus diferencias (parte 4)

2.6.3 Entregables

Para este *sprint* se entrega la aplicación funcionando.

Como se observa en la Figura 2.38 se tiene las dos imágenes en formato JPEG y JPEG *lossless* después de pasar por la herramienta esteganográfica, se puede notar que la imagen con mensaje presenta una diferencia en zoom y en el color ya que se modificó los bits más significativos.

Figura 2.38. Visualizador imágenes PNG y JPEG *lossless* entregable del *sprint* 4

En la Figura 2.39 se puede observar la transformada discreta del coseno de las imágenes ingresadas anteriormente, como se observa hay un aumento en la tonalidad roja.

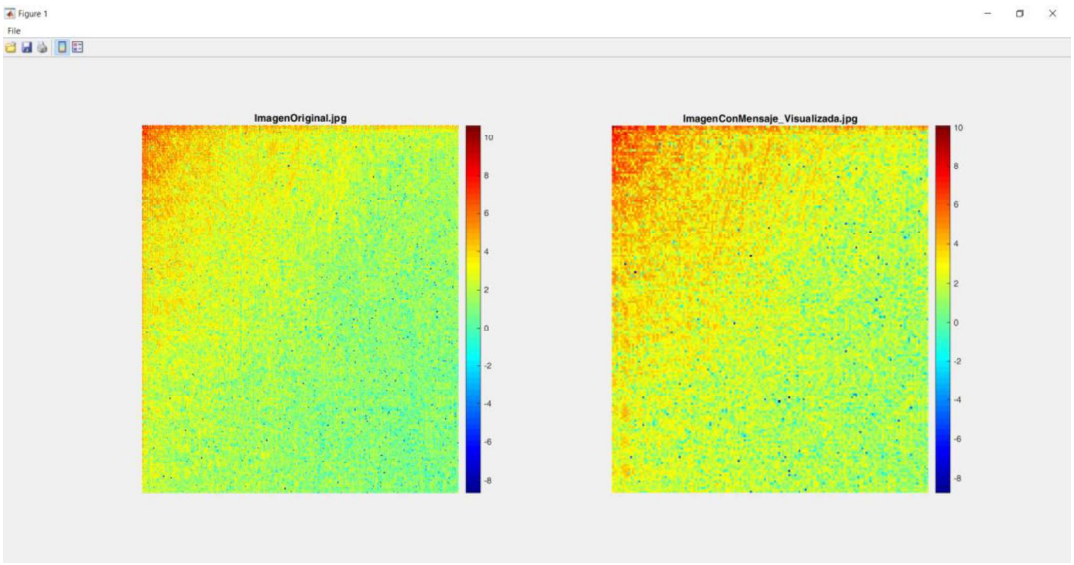

Figura 2.39. Visualizador con la DCT entregable del *sprint* 4

En la Figura 2.40 se puede observar la resta de las dos imágenes en los diferentes canales rojo, verde y azul. En la esquina superior izquierda del cuadro RGB se observa la diferencia total de las dos imágenes.

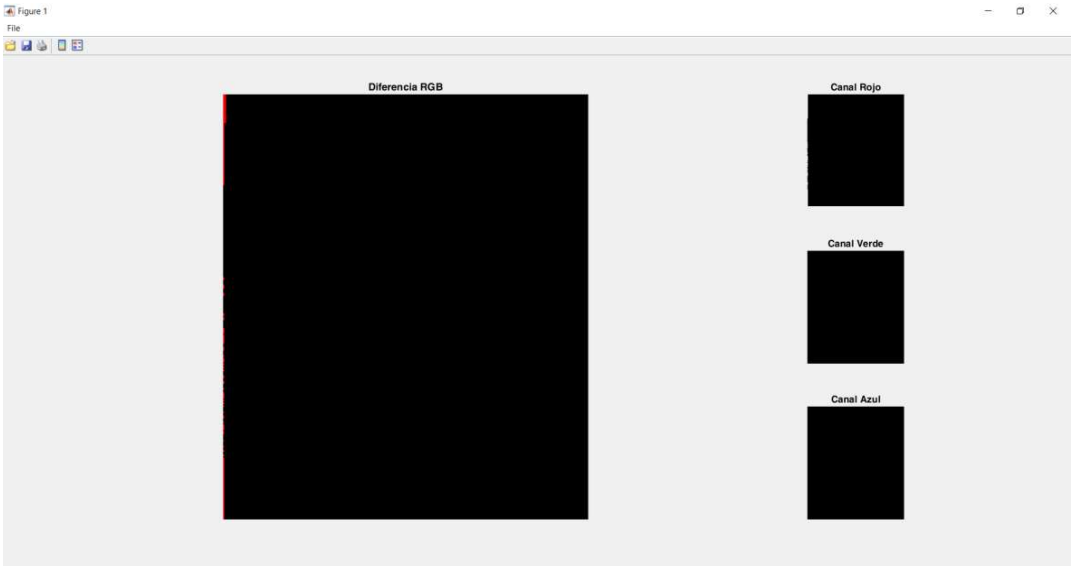

Figura 2.40. Visualizador resta de imágenes entregable del *sprint* 4

3 RESULTADOS Y DISCUSIÓN

En esta sección se presentan los resultados de las pruebas se realizaron y una demostración práctica de cada módulo para ver su funcionalidad total y que no contenga ningún error.

3.1 Módulo menú

Como se puede observar en la Figura 3.1 no existe ningún error en la ejecución del módulo principal.

Figura 3.1. Correcta ejecución del módulo Menú

3.2 Módulo cifrar texto en imagen

Para este módulo se realizó las pruebas con diferentes imágenes, generaciones de contraseñas y bits para esconder la información.

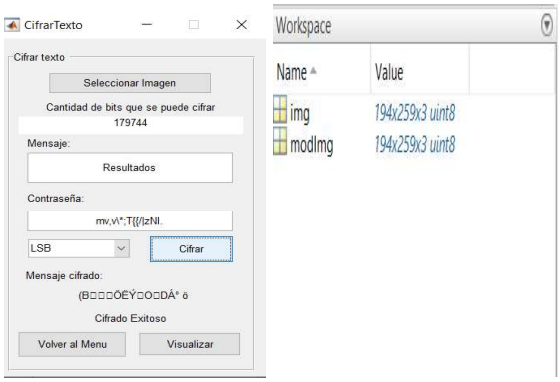

Figura 3.2. Correcta ejecución del módulo `Cifrar Texto En Imagen` con el bit LSB, generación automática de contraseña y creación de las variables `Img` y `modImg`

En la Figura 3.2 se puede observar que el cifrado fue exitoso y se despliega una parte del mensaje encriptado del algoritmo AES con llave de 128 bits.

En las Figuras 3.3, 3.4, 3.5, 3.6 y 3.7 se muestra las diferentes pruebas en los bits para ocultar la información con un resultado exitoso.

Figura 3.3. Módulo `Cifrar Texto En Imagen` con el bit número 2, ingreso de la contraseña de 16 bytes.

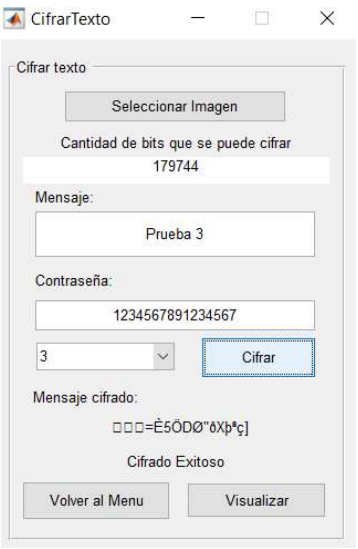

Figura 3.4. Módulo `Cifrar Texto En Imagen` con el bit número 3, ingreso de la contraseña de 18 bytes.

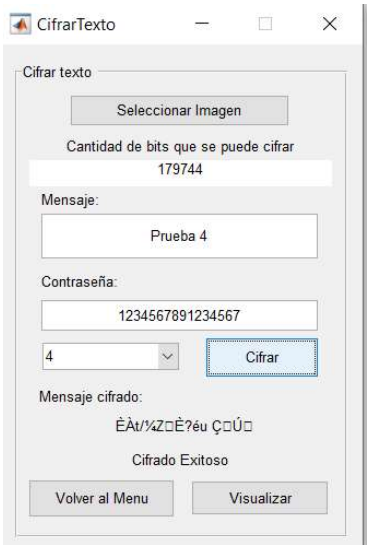

Figura 3.5. Módulo `Cifrar Texto En Imagen` con el bit número 4, ingreso de la contraseña de 16 bytes.

Como se puede observar en la Figura 3.4 y 3.5 si la contraseña es más grande se recorta y si es más pequeña se completa, por el contrario, si no se ingresa la contraseña se auto ingresa con caracteres aleatorios como se muestra en la figura 3.2.

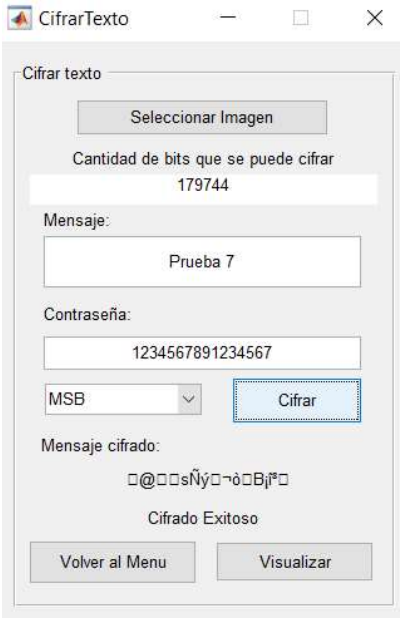

Figura 3.6. Módulo `Cifrar Texto En Imagen` con el bit MSB, ingreso de la contraseña de 7 bytes.

3.3 Visualizar en módulo cifrar texto en imagen

En el módulo `Visualizar` se muestran 4 formularios que presenta la imagen original en formato JPEG y PNG, la estegoimagen en formato JPEG y PNG y los *sliders* para realizar los cambios en brillo o zoom, como se observa en las Figuras 3.7 y 3.8.

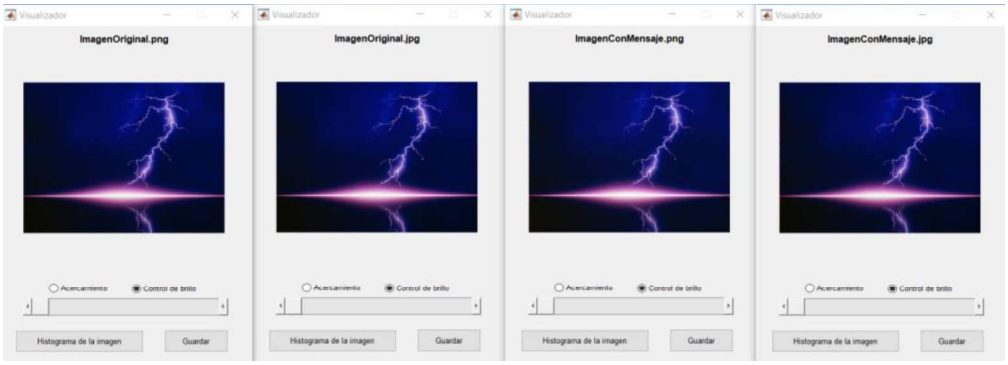

Figura 3.7. Correcta ejecución del módulo `Visualizar` en `Cifrar Texto En Imagen`

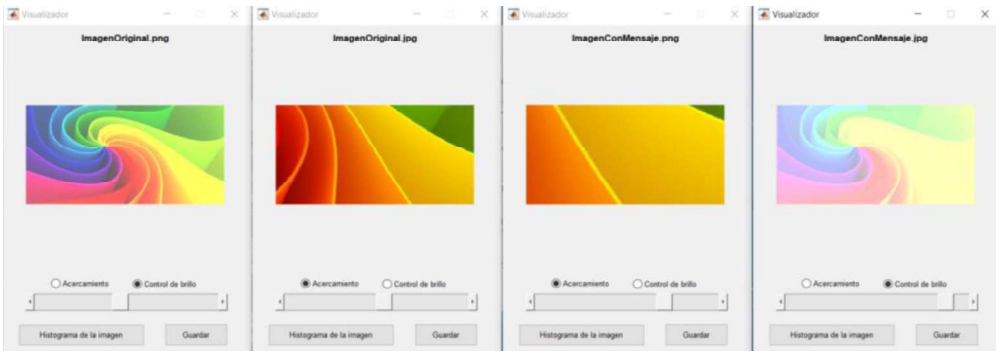

Figura 3.8. Correcta ejecución del zoom y brillo en `Cifrar Texto En Imagen`

ImagenConMensaje_Visualizada.jpg	26/5/2019 22:36	Archivo JPG	613 KB
ImagenConMensaje_Visualizada.png	26/5/2019 22:36	Archivo PNG	247 KB
ImagenOriginal_Visualizada.jpg	26/5/2019 22:36	Archivo JPG	158 KB
ImagenOriginal_Visualizada.png	26/5/2019 22:36	Archivo PNG	415 KB
ImagenOriginal.png	26/5/2019 22:36	Archivo PNG	431 KB
ImagenOriginal.jpg	26/5/2019 22:36	Archivo JPG	659 KB
ImagenConMensaje.png	26/5/2019 22:36	Archivo PNG	432 KB
ImagenConMensaje.jpg	26/5/2019 22:36	Archivo JPG	661 KB

Figura 3.9. Visualización de imágenes.

Como se puede observar en la Figura 3.9 se ha guardado la imagen original en formato JPEG y PNG, la estegoimagen en formato JPEG y PNG, la imagen original modificada en el módulo `visualizar` en formato JPEG y PNG (cada una se modificó por separado) y la estegoimagen modificada en el módulo `visualizar` en formato JPEG y PNG (cada una se modificó por separado).

En la Figura 3.10 se puede corroborar que el algoritmo LSB fue ejecutado exitosamente ya que la diferencia entre el histograma no es muy notaria, sin embargo, se puede apreciar pequeñas diferencias en la cantidad de bytes de ciertos valores como, por ejemplo, el bit 5 en la Figura 3.11, en el histograma de la izquierda se tiene un valor de 6.02 mientras que en el histograma de la derecha un valor de 5.98.

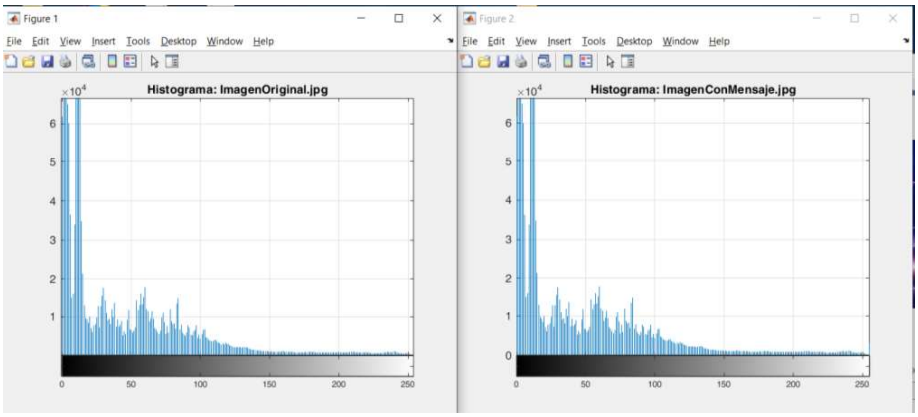

Figura 3.10. Correcta ejecución del histograma de `Cifrar Texto En Imagen`

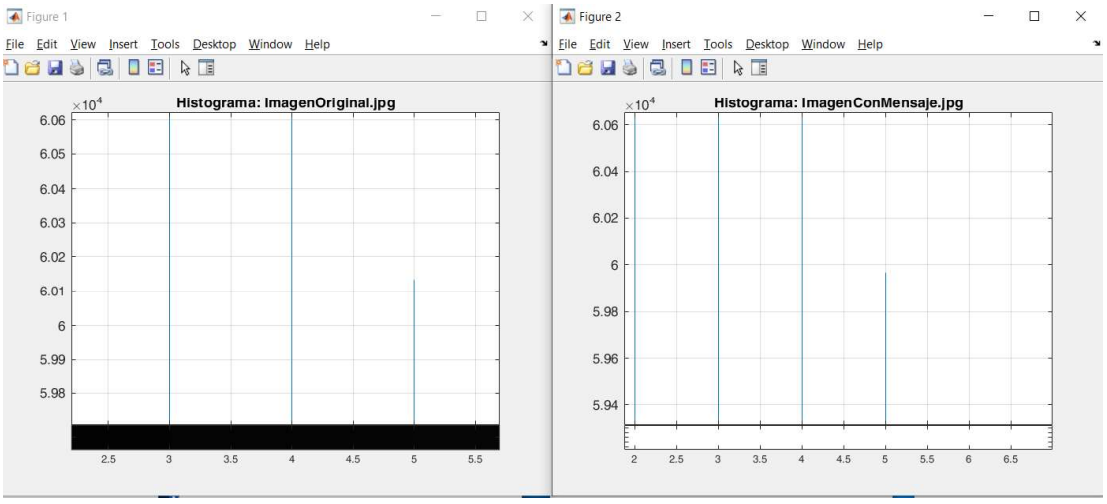

Figura 3.11. Diferencias en el histograma en el byte 5

En las Figuras 312, 3.13, 3.14 y 3.15 se puede observar las diferentes pruebas realizadas para descifrar el mensaje tanto en contraseña como con el bit a ocultar.

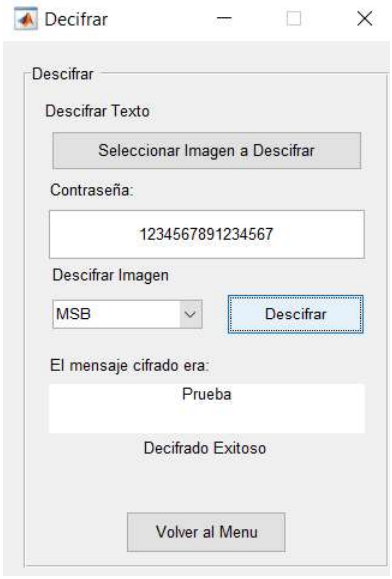

Figura 3.12. Descifrado correcto para la estegoimagen en `Cifrar Texto En Imagen`.

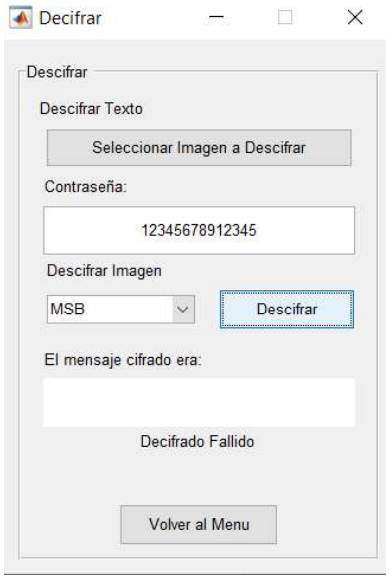

Figura 3.13. Descifrado fallido por clave de diferente tamaño para la estegoimagen en `Cifrar Texto En Imagen`.

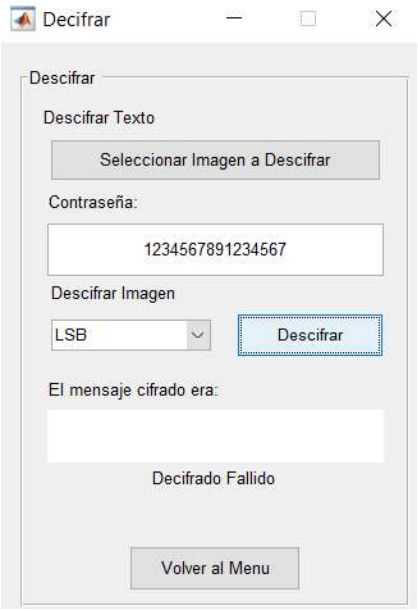

Figura 3.14. Descifrado fallido por ubicación del bit para la estegoimagen en `Cifrar Texto En Imagen`.

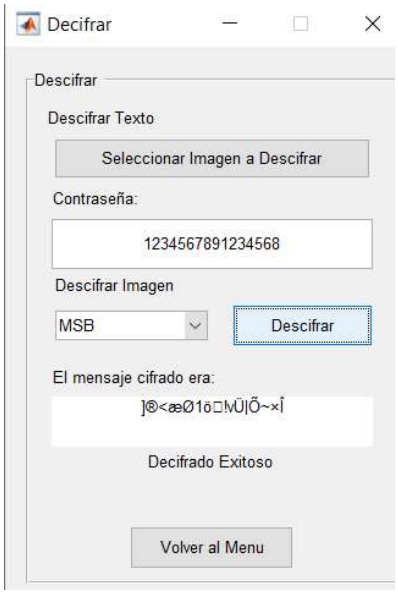

Figura 3.15. Descifrado fallido por clave incorrecta de 16 bytes para la estegoimagen en `Cifrar Texto En Imagen`.

El descifrado de la imagen será fallido cuando la clave sea de tamaño diferente a 16 bytes o el byte elegido donde se ocultó la imagen sea incorrecto, sin embargo, si la clave

es de 16 bytes, pero incorrecta el proceso se ejecutará satisfactoriamente, pero, mostrará una incongruencia en el mensaje.

3.4 Módulo cifrar archivo de texto en imagen

Este módulo funciona muy parecido al anterior (cifrar texto en imagen). De igual manera se puede observar una pequeña parte del texto cifrado con AES y su llave de 16 bytes, como se observa en las Figuras 3.16 y 3.17 se tiene una correcta ejecución de los módulos y creación de variables.

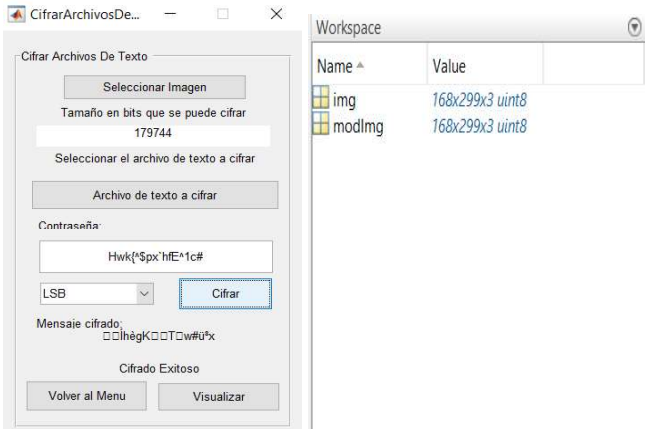

Figura 3.16. Correcta ejecución del módulo `Cifrar Archivo de Texto En Imagen`, contraseña aleatoria, bit LSB y creación de las variables

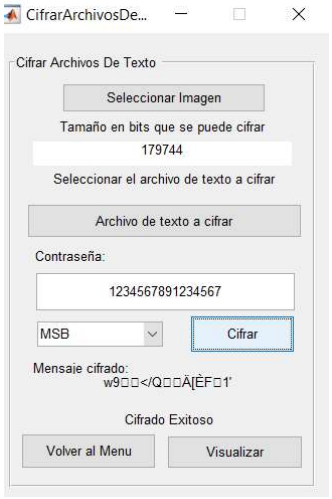

Figura 3.17. Correcta ejecución del módulo `Cifrar Archivo de Texto En Imagen`, contraseña ingresada, bit MSB

3.5 Visualizar en módulo cifrar archivo de texto en imagen

De igual manera este módulo muestra las 4 imágenes en formularios diferentes, la imagen original en JPEG y PNG y la estegoimagen en JPEG y PNG al igual que el *slider* para realizar cambios en brillo o zoom y guardar dichas imágenes como se observa en las Figuras 3.18 y 3.19. Mientras que en la Figura 3.20 se puede observar las imágenes creadas en la carpeta del *path* de trabajo.

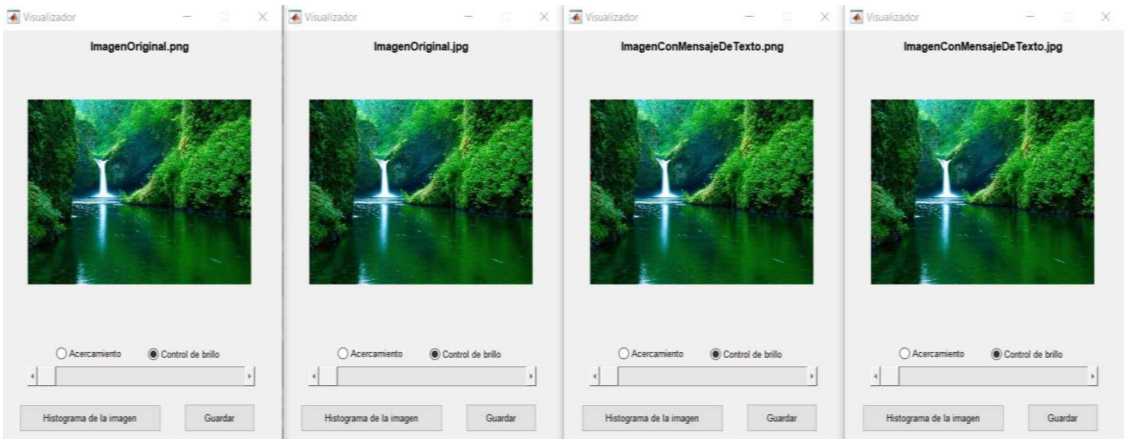

Figura 3.18. Correcta ejecución del módulo `Visualizar` en `Cifrar Archivo De Texto En Imagen`

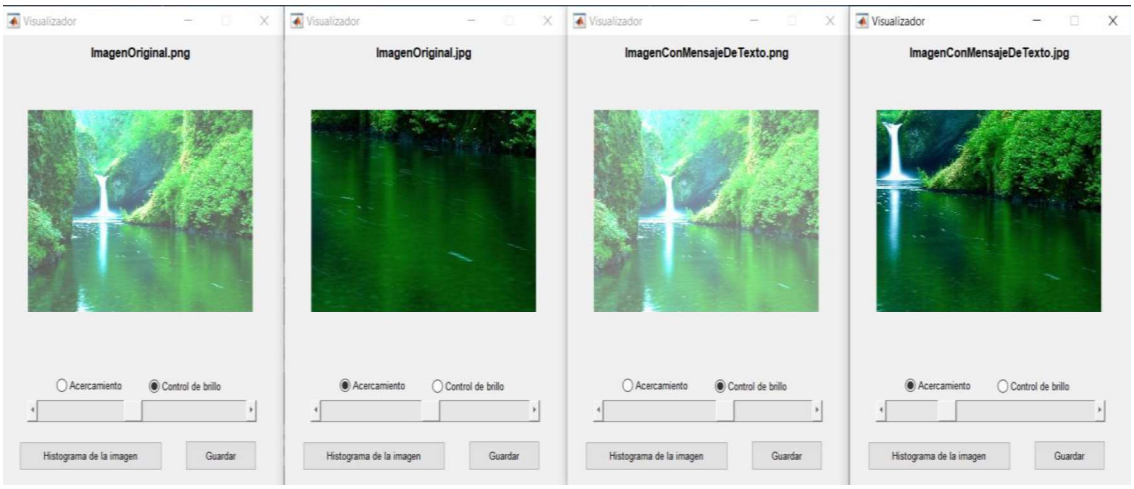

Figura 3.19. Correcta ejecución del zoom y brillo en `Cifrar Archivo De Texto En Imagen`

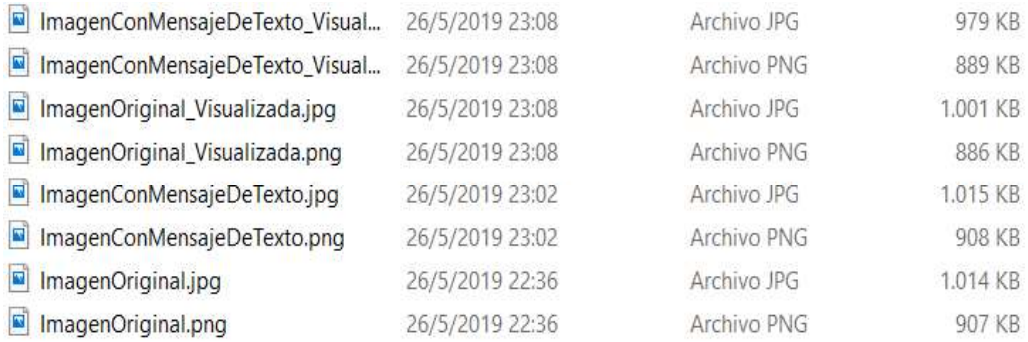

Figura 3.20. Visualización de imágenes.

Como se puede observar en la Figura 3.20 se ha guardado la imagen original en formato JPEG y PNG, la estegoimagen en formato JPEG y PNG, la imagen original modificada en el módulo visualizar en formato JPEG y PNG (cada una se modificó por separado) y la estegoimagen modificada en el módulo `visualizar` en formato JPEG y PNG (cada una se modificó por separado).

En la Figura 3.21 se puede ver que el algoritmo LSB fue exitoso ya los histogramas a simple vista no reflejan una diferencia.

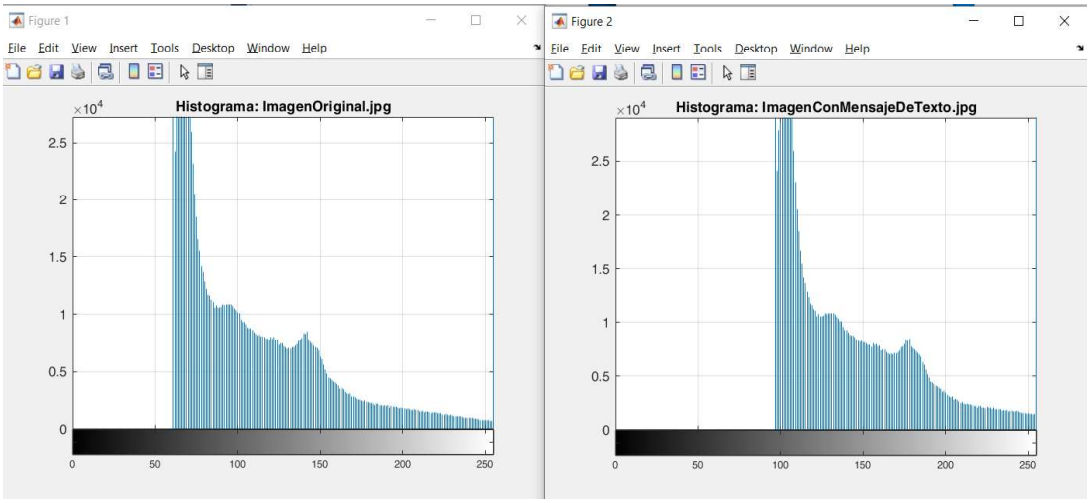

Figura 3.21. Correcta ejecución del histograma en `Cifrar Archivo De Texto En Imagen`

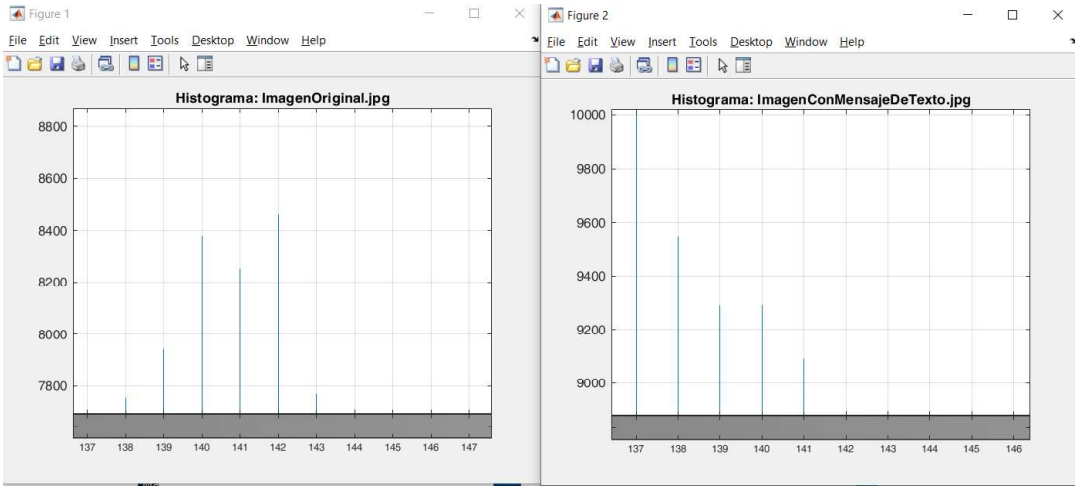

Figura 3.22. Diferencias en histograma

Como se puede observar en la figura 3.22 existe una diferencia en la cantidad de bytes con el valor 140, en el histograma de la izquierda tiene un valor aproximado de 8400 mientras que en el histograma de la derecha tiene un valor aproximado de 9300.

En la Figura 3.23 se puede ver el archivo resultante después de descifrar la estegoimagen original y la estegoimagen aplicada cambios de brillo.

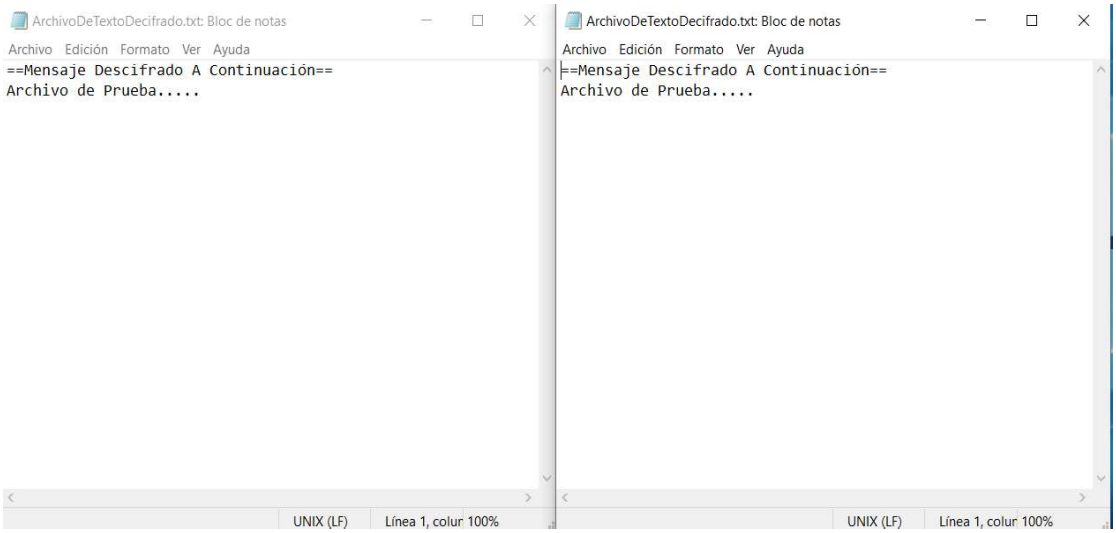

Figura 3.23. Descifrado correcto para la estegoimagen y estegoimagen con cambios en brillo para `Cifrar Archivo De Texto En Imagen`

3.6 Cifrado de un archivo mas grande que la cantidad máxima de bits disponibles para ocultar

Se realizó una prueba con un archivo de 5.75 MB que se muestra en la Figura 3.24 para ver el resultado del programa, ya que solo se puede utilizar 18581 bits es decir 2322 bytes por lo tanto el programa no realizará ninguna acción y seguirá mostrando el mensaje Imagen sin Descifrar como se muestra en la Figura 3.25.

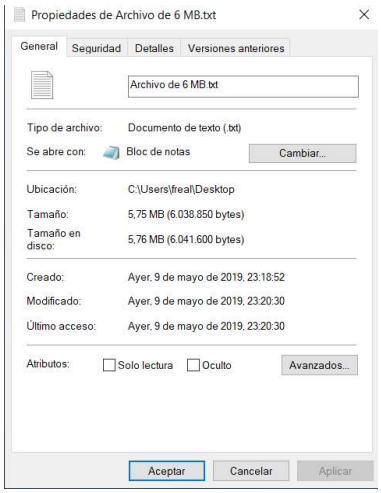

Figura 3.24. Archivo de 5.75 MB

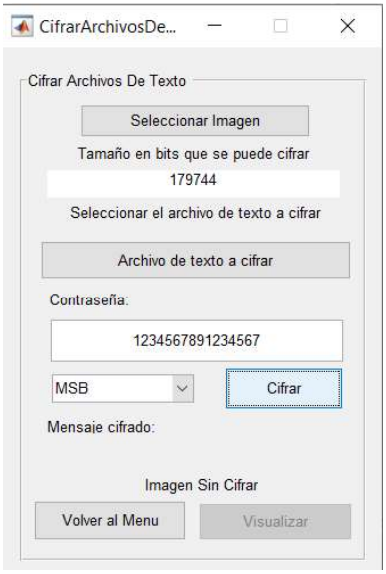

Figura 3.25. No se realiza el cifrado

3.7 Pruebas de tamaños con cambios de brillo

La Figura 3.26. muestra la estegoimagen vs la estegoimagen modificada en brillo, como se puede observar la imagen JPEG decremento 11 KB mientras que la PNG 8 KB a pesar de que el cambio fue el mismo para ambas, esto se debe a la diferencia métodos usados para formar la imagen original en JPEG y PNG.

Figura 3.26. Comparación de tamaños con cambios de brillo

3.8 Pruebas de tamaños con cambios de zoom

En la Figura 3.27. se puede apreciar la diferencia en tamaños que existe cuando la estegoimagen es sometida a cambios en zoom, para el formato JPG decremento en 53 KB mientras que la imagen en PNG decremento en 39 KB a pesar que el cambio fue el mismo para ambas, de igual manera la razón es la compresión que recibe JPG al usar el modo *lossless*.

Se puede notar que cuando se realiza zoom en las imágenes presentan una pérdida mucho más grande que cuando se realiza brillo, esto es debido a que el zoom recorta totalmente valores de la matriz mientras que el brillo solo los modifica.

Figura 3.27. Comparación de tamaños con cambios de zoom

3.9 Pruebas de tamaños con imagen original

3.9.1 JPEG

Como se puede ver en la figura 3.28 la estegoimagen tienen un tamaño mucho más grande que la imagen original esto es debido a que JPEG para ser compatible con LSB tiene que ser guardada en modo *lossless*, es decir, sin perdidas y ya que se guarda en

este modo la imagen no comprime sus bytes por lo tanto aumenta muchísimo su tamaño. Se debe tener en cuenta también que este formato no es soportado en Windows debido a que las aplicaciones buscan en la cabecera que se haya realizado la codificación Huffman para la compresión, sin embargo, como no se la realizó las aplicaciones no pueden abrirla, pero mediante *Matlab* si se puede acceder a dicha imagen.

Figura 3.28. Comparación de tamaños entre la imagen original y la estegoimagen en JPEG

3.9.2 PNG

En la figura 3.29. se puede observar que el tamaño entre la imagen original y la estegoimagen resultante en PNG no varía en nada ya que no se aumenta ni disminuye ningún bit simplemente se modifican, se puede apreciar que la imagen JPG si creció un poco más debido al modo *lossless* empleado en *Matlab*.

Figura 3.29. Comparación de tamaños entre la imagen original y la estegoimagen en PNG

3.10 Valores de histogramas

Los histogramas a simple vista no muestran un cambio muy grande como se muestra en la Figura 3.30, sin embargo, si se analiza cada uno de los bytes si se puede encontrar diferencias en el valor de cada uno.

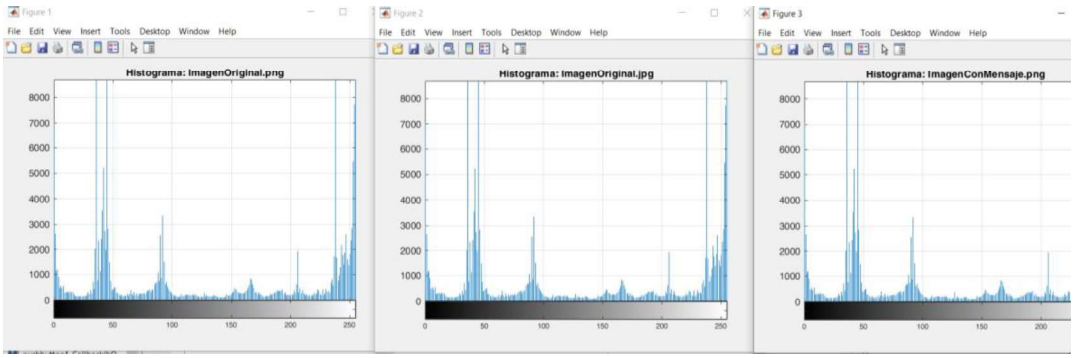

Figura 3.30. Comparación de histogramas

3.11 Comparación de archivo original y recuperado

En la Figura 3.31 se puede observar el archivo original y el desencriptado, se ve que mantienen la misma información, la diferencia radica en el título que se coloca al archivo desencriptado.

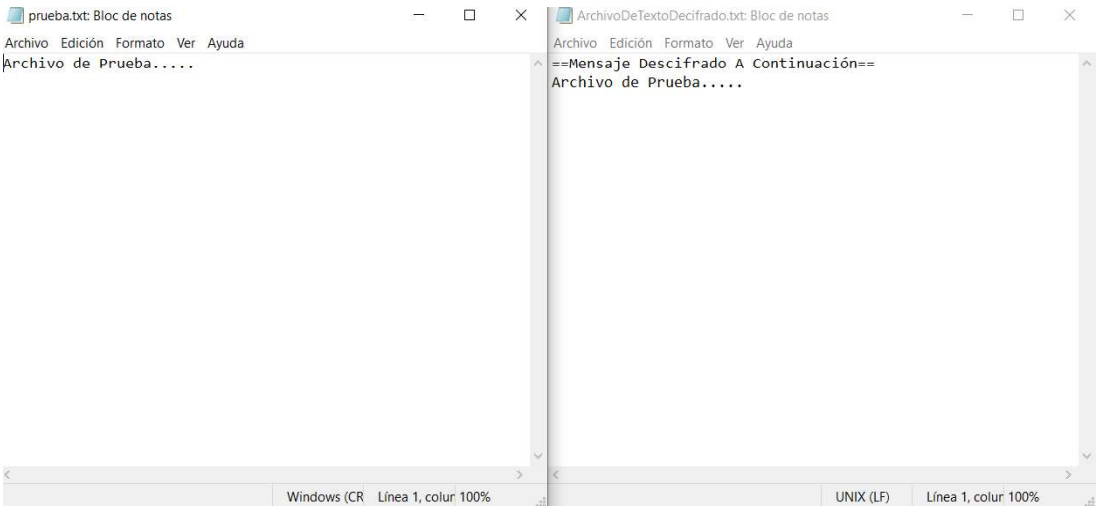

Figura 3.31. Comparación archivo original con archivo descifrado

3.12 Visualizador

En esta sección se presenta los resultados de la aplicación programada para visualizar las imágenes y sus diferencias mediante la transformada discreta del coseno y la resta de imágenes que se han implementado dentro de la aplicación.

En la Figura 3.32 se puede observar el ambiente de trabajo de la aplicación para la visualización de las imágenes y sus diferencias.

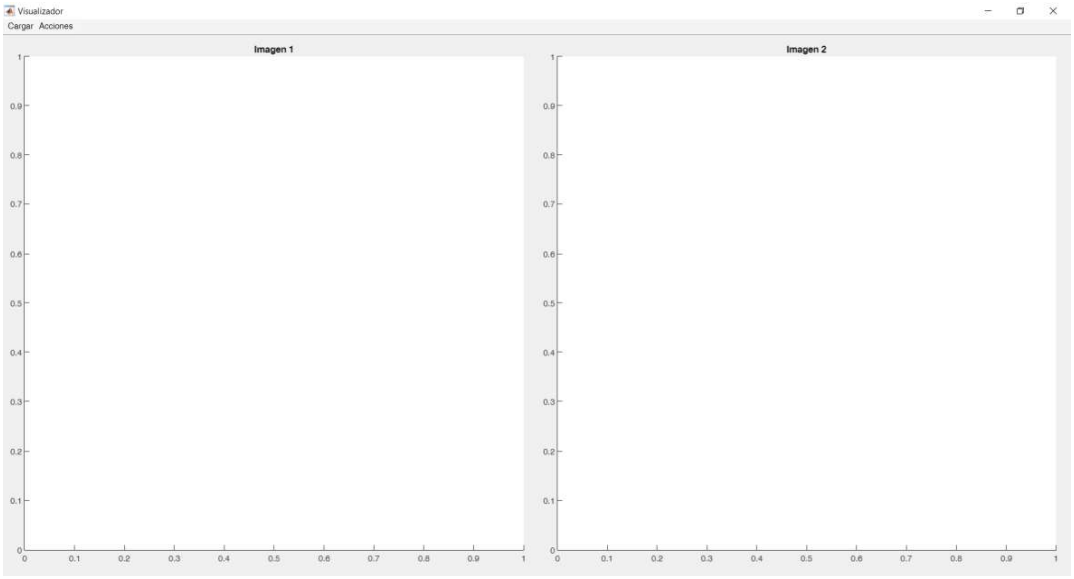

Figura 3.32. Aplicación programada visualizador

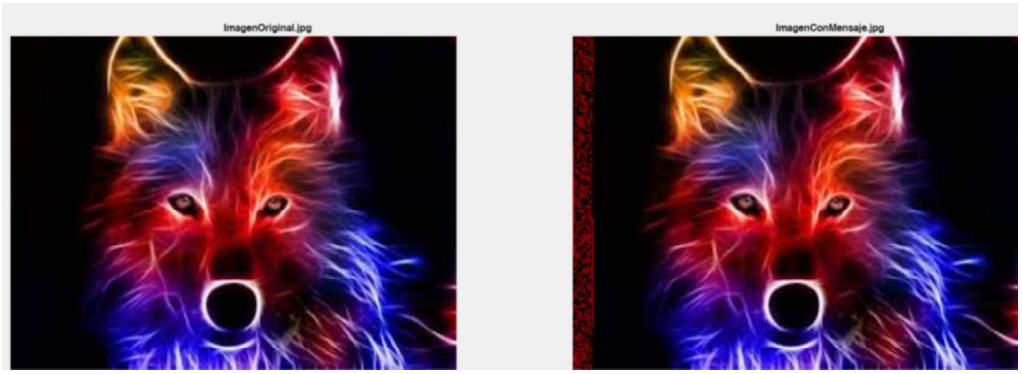

Figura 3.33. Imagen original vs estegoimagen

Como se puede observar en la Figura 3.33 existe una diferencia muy notaria en la imagen de la derecha se observa una franja de color rojo, esto es debido a que para ocultar la información se utilizó el bit más significativo MSB.

En la Figura 3.34 se puede observar la transformada discreta del coseno en donde se ve una difuminación del color azul en la parte inferior derecha debido a las diferencias incluidas en la estegoimagen.

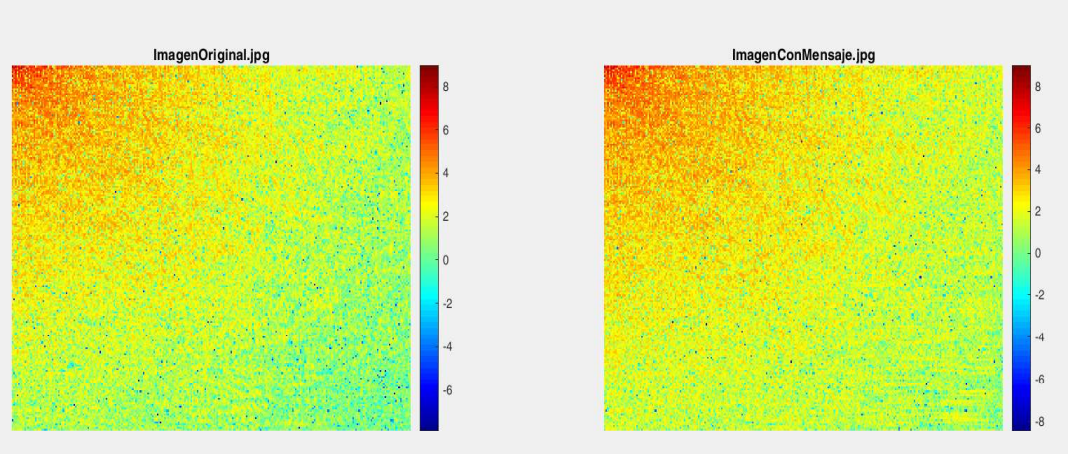

Figura 3.34. Cálculo de la DCT en el visualizador

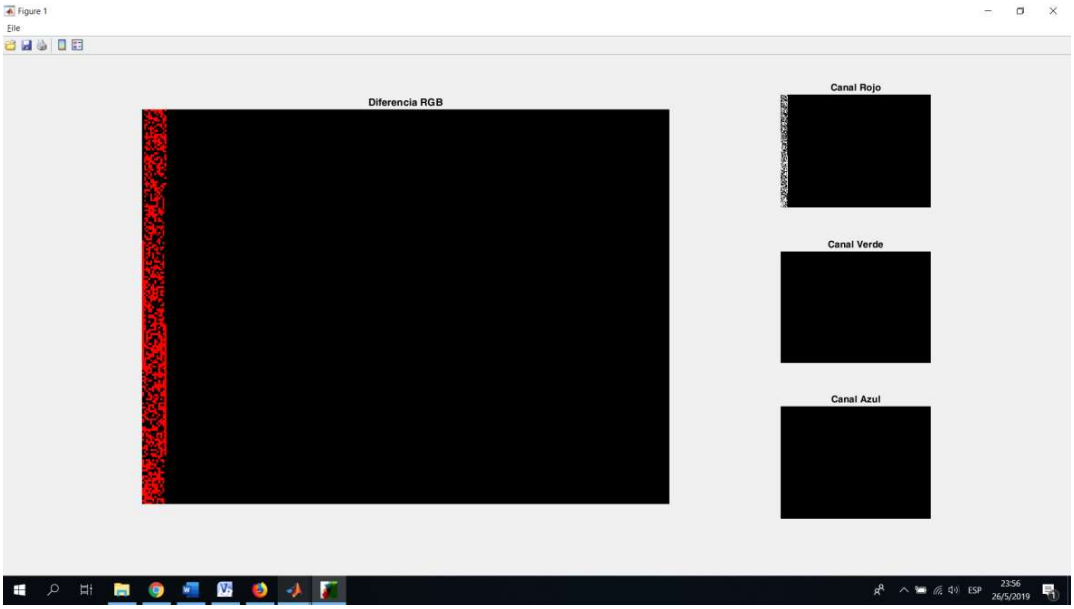

Figura 3.35. Resta de las dos imágenes

En la Figura 3.35 se muestra la resta de las dos imágenes original y estegoimagen, para lo cual se muestra la principal donde se observa los 3 canales RGB y en la parte de la derecha cada canal por separado.

Se debe tener en cuenta que el visualizador es capaz de abrir cualquier formato JPEG y PNG, incluyendo JPEG *lossless*.

4. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- El algoritmo LSB es una técnica esteganográfica muy eficiente ya que solo altera el bit menos significativo en la imagen presentando un 12,5% de cambios, por lo tanto, a simple vista no se puede identificar que existe una modificación en la misma, sin embargo, si se analiza la imagen con herramientas más sofisticadas, la imagen se puede dar cuenta que, algunos de esos bits fueron modificados, por lo cual es importante ocultar el mensaje encriptado para dificultar la recuperación de la información.

- La clave del algoritmo AES, sin importar el tamaño que se use, es unidireccional por lo que no se puede realizar una comparación entre la clave ingresada para encriptar y la clave usada para desencriptar, por lo tanto, si la contraseña usada para desencriptar no es la misma, pero mantiene el tamaño de la original, el algoritmo realiza todo el proceso, pero no muestra la información original, sino, caracteres sin sentido.

- De acuerdo a los resultados obtenidos se puede concluir que cuando se usa modo *lossless* con JPEG, es decir modo sin pérdidas, no realiza la codificación Huffman por lo tanto no se realiza una compresión en la imagen lo que provoca que las aplicaciones que abren imágenes no puedan abrir la imagen, sin embargo, esta imagen en modo *lossless* puede ser abierta mediante la aplicación creada en el presente proyecto.

- Con los algoritmos empleados no se pretende tener una seguridad irrompible sino una herramienta para que se pueda observar cómo se utilizan los algoritmos de encriptación y esteganográficos para ocultar información dentro de una imagen, pero se concluye que mientras más herramientas se aplique, el método será mucho más seguro.

- *Matlab* presenta una gran limitante cuando la información introducida es bastante grande, ya que al querer trabajar bit a bit demanda mucho procesamiento y tiempo para que pueda culminar con dicha operación.

- Con las imágenes obtenidas se puede concluir que el algoritmo LSB funciona de una manera más eficiente con imágenes PNG ya que cuando se trabaja con dichas imágenes no se ve ninguna diferencia en el tamaño de la imagen original con la estegoimagen haciéndola aún más imperceptible para un atacante o a su vez para una víctima.

- Se puede concluir que cada imagen tiene una diferente cantidad de bits disponibles para ocultar la información, debido a que esta cantidad depende de la filas, columnas y canales (como el RGB que serían 3 canales) que presente la imagen, siendo una relación directamente proporcional.

- Con los resultados obtenidos podemos darnos cuenta que el histograma de una imagen que pasó por el algoritmo LSB va a ser un poco diferente a la original por lo tanto sería una debilidad para descubrir que dicha imagen fue modificada mediante algún algoritmo.

4.2 Recomendaciones

- Si se quiere correr la aplicación es recomendable que se lo haga en la misma versión de *Matlab* 2018b de 64 bits para evitar que exista errores o alguna librería sea incompatible, de igual manera se recomienda instalar el paquete completo de *Matlab*.

- Se recomienda usar funciones para el algoritmo de encriptación, para llamar a dichas funciones según sea necesario y evitar la redundancia de código en los *scripts*.

- Se recomienda usar un solo bit por cada byte para el ocultamiento de la información, ya que no se notan cambios relevantes en la imagen cuando se hace la comparación entre la original y la modificada, sin embargo, al usar 2 bits por cada byte puede afectar en la resolución.

- Una recomendación importante es no intentar comparar las claves de ingreso del algoritmo de encriptación AES ya que al ser unidireccional no se conseguirá nunca comprobar que la clave para desencriptar es la misma que la ingresada para descifrar.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] L. M. Í. B. Gabriela Velásquez, "Análisis de técnicas de esteganografía aplicadas en archivos de audio e imagen," *Polo del Conocimiento,* vol. 2, no. 1, p. 67, 2017.

[2] INTECO, "Esteganografía, el arte de ocultar información," *Observatorio de la Seguridad de la Información,* p. 15, 2016.

[3] J. J. Héctor Villa, "Repositorio UTP," Universidad Tecnológica de Pereira, 2015. [Online]. Available: http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/6014/00582V712.pdf;jsessionid=B8B337B2A4B9F4468BF2331AA4E4B47F?sequence=1. [Accessed 12 Febrero 2019].

[4] J. Carillo, "Diseño e implementación de un software para ocultar información," *Sintesis Semilleros de Investigación ,* vol. 3, p. 34, 2014.

[5] S. Fernandez, "Amazonaws," Abril 2004. [Online]. Available: https://s3.amazonaws.com/academia.edu.documents/40562076/9_Criptografia_clasica.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1550163455&Signature=Q7Ja1L3geiMImBvstPLUDSy17Wg%3D&response-content-disposition=inline%3B%20filename%3DCriptografia_clasica.pd. [Accessed 12 Febrero 2019].

[6] L. Roca, "Nube Informática," 7 Julio 2016. [Online]. Available: http://minubeinformatica.com/cursos/seguridad-informatica/criptografia. [Accessed 23 junio 2019].

[7] J. R. Aguirre, Seguridad Informática y Criptografía, Madrid: Departamento de publicaciones de la Universidad Politécnica de Madrid, 2006.

[8] V. S. A. d. G. Adrian Pousa, "Sedici," 2011. [Online]. Available: http://sedici.unlp.edu.ar/bitstream/handle/10915/18646/Documento_completo.pdf?sequence=1&isAllowed=y. [Accessed 12 Febrero 2018].

[9] J. F. S. N. Eduardo Azevedo, "Esteganografía," *Exatas Tecnol,* vol. 10, no. 10, p. 35, 2015.

[10] A. C. H. V. Pablo Mendez, "Propuesta de mejora de un algoritmo criptográfico con la combinación de la esteganografía en imagenes," *Cumbres ,* vol. 3, no. 2, p. 21, 2017.

[11] J. Wallace, "IEEE," Febrero 1992. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/125072. [Accessed 12 Febrero 2019].

[12] ITU, "International Telecomunication Union," 30 Enero 2004. [Online]. Available: https://www.itu.int/rec/T-REC-T.81-200401-I!Cor1/en. [Accessed 2019 Abril 22].

[13] E. Morocho, "Repositorio Virtual EPN," Julio 2014. [Online]. Available: https://bibdigital.epn.edu.ec/handle/15000/8062. [Accessed 22 Abril 2019].

[14] J. M. B. Lopes, "Universidad Técnica de Lisboa," Diciembre 2018. [Online]. Available: http://disciplinas.ist.utl.pt/~leic-cg.daemon/textos/livro/Formatos%20de%20Imagem.pdf. [Accessed 10 04 2019].

[15] Mathworks, "Matlab," 1994. [Online]. Available: https://www.mathworks.com/products/matlab.html. [Accessed 12 Febrero 2019].

[16] Mathworks, "Matlab," 1994. [Online]. Available: https://www.mathworks.com/discovery/matlab-gui.html. [Accessed 12 Febrero 2019].

[17] M. T. Gallego, "Universidad Oberta de Catalunya," 2012. [Online]. Available: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf. [Accessed 10 Abril 2019].

[18] A. Pasini, "Universidad Nacional de la Plata," Octubre 2013. [Online]. Available: http://sedici.unlp.edu.ar/handle/10915/32421. [Accessed 18 Abril 2019].

[19] G. E. Onofre Concha, "Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE," Octubre 2016. [Online]. Available: http://repositorio.espe.edu.ec/handle/21000/12249. [Accessed 23 Junio 2019].

[20] D. A. Changoluisa Simbaña, "BIBDIGITAL," 26 Abril 2019. [Online]. Available: https://bibdigital.epn.edu.ec/handle/15000/20195. [Accessed 23 Junio 2019].

[21] E. Abascal, Análisis de Encuestas, Madrid: ESIC, 2005.

[22] T. MathWorks, "MathWorks Products," 2019. [Online]. Available: https://la.mathworks.com/store/link/products/home/new. [Accessed 23 Abril 2019].

[23] T. MatWorks, "MathWorks," 2019. [Online]. Available: https://la.mathworks.com/help/matlab/ref/commandhistory.html. [Accessed 24 Abril 2019].

[24] T. MathWorks, "MathWorks help," 2019. [Online]. Available: https://la.mathworks.com/help/matlab/ref/cd.html. [Accessed 23 Abril 2019].

[25] T. MathWorks, "MathWorks," 2019. [Online]. Available: https://la.mathworks.com/support/requirements/matlab-system-requirements.html. [Accessed 23 Abril 2019].

[26] T. MathWorks, "MathWorks Program," 2019. [Online]. Available: https://la.mathworks.com/academia/tah-support-program/eligibility.html. [Accessed 23 Abril 2019].

[27] T. MathWorks, "MathWorks Imwrite," [Online]. Available: https://la.mathworks.com/help/matlab/ref/imwrite.html. [Accessed 2019 Abril 19].

[28] J. d. J. A. Angel, "AES Advanced Encryption Standard," Abril 2005. [Online]. Available: http://www.criptored.upm.es/guiateoria/gt_m117i.htm. [Accessed 10 Abril 2019].

[29] J. J. Buchholz, "Academia," Diciembre 2001. [Online]. Available: https://www.academia.edu/26363786/Matlab_Implementation_of_the_Advanced_Encryption_Standard. [Accessed 18 Abril 2019].

6. ANEXOS

En esta sección se ha colocado la información que por su extensión no fueron incorporadas en ninguna de las secciones anteriores.

ANEXO I. Modelo de encuesta para requerimientos de usuarios.

ANEXO II. Tabla de funciones

ANEXO III. Código de la aplicación

ANEXO IV. Instalador del visualizador para imágenes JPEG *lossless*

ANEXO V. Manual de uso de la aplicación

ANEXO I

Modelo de encuesta para requerimientos de usuarios.

Marque con una X

1. **¿Qué formato de imagen le gustaría usar para ocultar información?**
    a) JPEG ( )
    b) PNG ( )
    c) BMP ( )
2. **¿Ha abierto una imagen con formato JPEG *lossless* (sin pérdida)?**
    a) Si ( )
    b) No ( )
3. **¿Ha necesitado abrir una imagen JPEG *lossless* (sin pérdida)?**
    a) Si ( )
    b) No ( )
4. **¿Sería útil tener una herramienta para abrir imágenes JPEG *lossless* (sin pérdida)?**
    a) Si ( )
    b) No ( )
5. **¿Cómo le gustaría ver las diferencias de dos imágenes después de aplicar esteganografía?**
    a) Histograma ( )
    b) Transformada discreta de Fourier DCT ( )
    c) Resta de imágenes ( )
6. **¿Le parecería a usted que la herramienta es didáctica si le permite ocultar la información en cualquier bit de la imagen?**
    a) Si ( )
    b) No ( )
7. **¿Piensa usted qué una aplicación capaz de ocultar información en imágenes aplicando el algoritmo *Least Significant Bit* (LSB) sería didáctica?**
    a) Si ( )
    b) No ( )

ANEXO II

Tabla de funciones

Función	Descripción
add_round_key.m	Realiza un XOR a nivel de bits de la matriz de estado y la matriz de clave redonda.
aff_trans.m	Consiste en una multiplicación polinomial de una constante específica con otra constante y la adición de una tercera constante.
CifrarArchivosDeTexto.fig	Interfaz gráfica para cifrar un archivo de texto dentro de una imagen.
CifrarArchivosDeTexto.m	Operaciones de ocultamiento y encriptación para archivos de texto.
CifrarTexto.fig	Interfaz gráfica para cifrar un mensaje ingresado por teclado dentro de una imagen.
CifrarTexto.m	Operaciones de ocultamiento y encriptación para mensajes ingresados por teclado.
Cipher.m	Se encarga de extraer la primera matriz 4 × 4 de la programación de la clave w.
cycle.m	Permuta cíclicamente las filas de la matriz de entrada.
Descifrar.fig	Interfaz gráfica encargada de descifrar la información oculta dentro de una imagen.
Descifrar.m	Operaciones contrarias de ocultamiento y encriptación.
decryptData_AES128.m	Función inversa de AES.
encryptData_AES128.m	Función AES.
find_inverse.m	Bucle que recorre todos los valores posibles de los bytes para calcular el byte que se invertirá.
getRandomString.m	Es la encargada de llenar con caracteres imprimibles la contraseña, con el código ASCII del 33 al 126.

imgMaxData.m	Es la encargada de calcular la cantidad máxima de bits a ingresar.
inv_cipher.m	Invierte las transformaciones del proceso de cifrado.
inv_shift_rows.m	Invierte el efecto de la función shift_rows correspondiente en el proceso de cifrado.
key_expansion.m	Toma la clave de 16 bytes proporcionada por el usuario y generar una clave de 176 bytes w.
Menu.fig	Interfaz gráfica encargada de desplegar el menú.
Menu.m	Contiene toda la programación de los botones y la presentación del archivo.
mix_columns.m	Calcula la nueva matriz de estados S'.
poly_mat_gen.m	Permutación por filas de dicha matriz mediante una llamada a la función cycle.
poly_mult.m	Realiza la multiplicación y la reducción modular.
rcon_gen.m	Creación de la matriz de ceros y las potencias de 2.
readDataIntoImage.m	Función inversa de LSB.
rot_word.m	Permuta cíclicamente.
s_box_gen.m	Es la encargada de generar las dos tablas de sustitución s_box y la tabla inv_s_box.
s_box_inversion.m	Revierte la sustitución realizada a través de la caja S.
saveDataIntoImage.m	Algoritmo LSB.
saveImage.m	Permite guardar las imágenes resultantes en los diferentes formatos JPEG *lossless* y PNG.
shift_rows.m	Permuta cíclicamente desplazando las filas de la matriz de estado a la izquierda.
sub_bytes.m	Aplica la s_box a uno o más bytes de entrada.
Visualizador.fig	Interfaz gráfica para la observación y editaje de las imágenes.

`Visualizador.m`	Contiene el código de la interfaz gráfica para la visualización de la imagen e histograma y el editaje de las mismas.
`VisualizadorCif.fig`	Interfaz gráfica para la observación y editaje de las imágenes cifradas.
`VisualizadorCif.m`	Contiene el código de la interfaz gráfica para la visualización de la imagen e histograma y el editaje de las mismas.
`VisualizadorOri.fig`	Interfaz gráfica para la observación y editaje de las imágenes originales.
`VisualizadorOri.m`	Contiene el código de la interfaz gráfica para la visualización de la imagen e histograma y el editaje de las mismas.

ANEXO III

Código de la aplicación

Por la extensión del código de la aplicación se lo colocó en el CD adjunto.

ANEXO IV

Instalador del visualizador para imágenes JPEG *lossless*

Se incluyo el ejecutable del visualizador dentro del CD adjunto

ANEXO V

Manual de uso de la aplicación

Por la extensión de este archivo se lo colocó en el CD adjunto.

ORDEN DE EMPASTADO