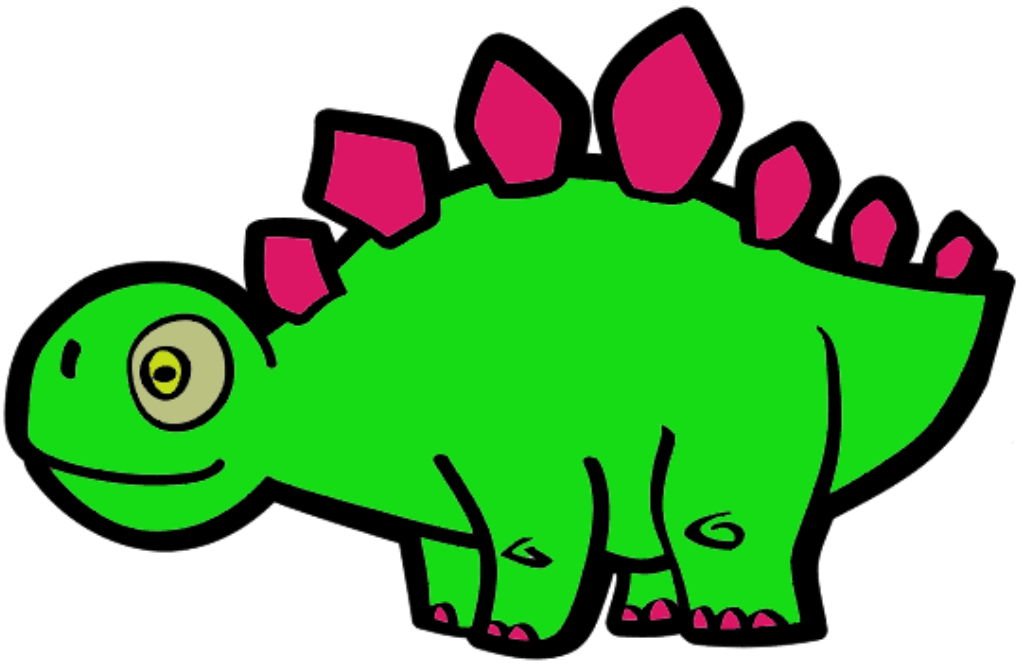
1-12-2018

DANIEL ALONSO

PROYECTO FINAL DE DESARROLLO DE APLICACIONES MULTIPLATAFORMA



Documentación del proyecto

Índice de contenido

1. Objetivos

1.1. Privacidad

Se ha desarrollado una aplicación Android cuya finalidad principal es la protección de la privacidad del usuario. En concreto, de la información que se almacena en las fotografías que se toman con su dispositivo Android, ya que pueden contener datos sensibles y que representan un problema para la privacidad.

1.2. Facilidad de uso

La aplicación que se presenta es capaz de generar una copia idéntica visualmente de la imagen recibida, sin los metadatos a la vista, pero pudiendo ser visualizados o reestablecidos con ésta aplicación.

Desde un menú principal fácil de entender y usar se pueden realizar estas acciones. También incorpora un botón de ayuda por si fuera necesaria ampliar la información.

Esto permite poder usar la foto en internet sin temor a que extraigan información sensible de ella y se ponga en compromiso tu privacidad de una forma sencilla.

1.3. Novedad

En el mercado actual no existen aplicaciones que hagan uso de la esteganografía para esconder los metadatos de fotografías hechas con tu dispositivo o seleccionadas de la galería, lo que le otorga a esta aplicación ese aspecto novedoso.

En ésta se hace uso de técnicas de actualidad en el mundo de la informática, algunas como la esteganografía existen desde la antigüedad, pero adaptada a la tecnología de nuestra época. También se usan los metadatos de las imágenes, que actualmente están desempeñando un papel muy importante debido a la BigData, y los problemas de privacidad que surgen a partir de este uso masivo de datos por parte de empresas como Google, que tiene el monopolio de la información de internet.

2. Planificación y desarrollo

2.1. Planificación

La primera fase consistió en hacer unas pruebas para ver si la idea que me rondaba la cabeza era factible de realizarse, decidí usar Java para poder portar el código después a una aplicación Android sin tener que cambiarlo mucho también use Netbeans como entorno de desarrollo.

Cuando ya lo tuve claro comencé a trabajar en el núcleo de la aplicación, más en concreto en la parte dedicada a la inserción y extracción de información mediante esteganografía en imágenes.

El siguiente paso consistió en implementar la parte de leer y escribir los metadatos y adaptar el código anterior para que la información almacenada fueran los metadatos.

Seguidamente realicé la aplicación Android usando el entorno de desarrollo AndroidStudio, implementado las funcionalidades que da soporte la aplicación, para lo cual he usado el código fuente anterior teniendo que modificar de nuevo el núcleo y adaptarlo a las librerías que da soporte Android.

Y por último he diseñado la interface para que fuese sencilla y fácil de usar.

2.2. Desarrollo

La aplicación se podría dividir en varias partes:

2.2.1. Núcleo de la aplicación

Consiste en el código Java que forma la parte interna de la aplicación, la programación no es muy buena y puede optimizarse mucho más, pero es estable. Al ser la primera versión y haber tenido muy limitado el tiempo es algo normal.

En el núcleo se pueden diferenciar dos procesos.

Procesado de metadatos

Consiste en el código fuente que se encarga de manejar los metadatos de las imágenes.

La primera versión de la aplicación usaba la clase JpegImageMetadata de la librería apache/Sanselan, que posteriormente se sustituyó por la clase nativa android.media.ExifInterface, que su manejo es más sencillo y no se necesita hacer uso de librerías externas.

Para facilitar el proceso se implementó la clase ExifData que permite almacenar los metadatos e implementa la interface Serializable, lo que permite serializar el objeto en un string codificado en base64 y la clase ExifControl que permite leer y escribir el exif a partir del objeto ExifData.

El procesado de metadatos se puede dividir de dos formas:

- **Extraer metadatos**

  Consiste en extraer los metadatos de la imagen original.
  El manejo de la clase ExifInterface es sencillo, abajo se muestra como instanciar un objeto ExifInterface y extraer el metadato "TAG_ORIENTATION"

```
//Obtener el exif
ExifInterface exif = new ExifInterface(file);

//Extraer un atributo/tag
String orientString = exif.getAttribute(ExifInterface.TAG_ORIENTATION);
```

  El proceso de extracción esta implementado en los siguientes métodos

```
//Extraer el exif
ExifData exifData = ExifControl.extraerExif(rutaImagen);

//Serializar exif
String base64 = exifData.toBase64String();
```

- **Reestablecer**

  Consiste en volver a agregar los metadatos originales, el proceso se muestra en las siguientes líneas de código.

```
//Obtener datos de la Estegoimagen
String base64ExifData = estegoMaquina.getExtraccion();

//Obtener exif desde serial
ExifData exif = new ExifData(base64ExifData);

//insertar exif
ExifControl.modificarExif(rutaImagen, exif);
```

  Como en el proceso anterior se usa ExifInterface para escribir los metadatos.

```
//Instanciar objeto
ExifInterface newExif = new ExifInterface(imagePath);

//Escribir un atributo/tag
newExif.setAttribute(ExifInterface.TAG_ORIENTATION, Orientation);

//Guardar los atributos
newExif.saveAttributes();
```

Esteganografía

Consiste en el código fuente capaz de introducir y extraer información en la imagen sin modificarla visualmente demasiado (de forma imperceptible por el ojo humano).

Para este proceso se ha utilizado la clase android.graphics.Bitmap, que permite acceder a los pixeles de la imagen.

- **Inserción de datos**
  Mediante la técnica de sustitución del bit menos significativo (Posteriormente explicada), se modifican determinados pixeles de la imagen, almacenando en cada pixel parte de la información deseada.

  Este proceso esta implementado en el método *insertarCarga()* de la clase componentes.EstegoMaquina

```java
/**
 * Inserta la carga en los pixels correspondientes de la imagen
 */
private void insertarCarga() {

    //Obtener el numero de canales usados
    int numeroCanalesUsados = 3;

    //Obtener pixeles actualizados con la Carga_Tamaño
    ArrayList<Pixel> pixelesTamMod = generarPixeles(
        numeroCanalesUsados, carga.encapsularTam());

    ///Actualizo el pixel actual, cambio de filaY +1
    pixelActual.setX(0);
    pixelActual.setY(pixelActual.getY() + 1);

    //Obtener pixeles actualizados con la Carga_Carga
    ArrayList<Pixel> pixelesCargaMod = generarPixeles(
        numeroCanalesUsados, carga.getBinary());

    //Modificar los pixeles actualizados en la imagen
    imagen.actualizarImagenRGB(pixelesTamMod);          //TAMAÑO
    imagen.actualizarImagenRGB(pixelesCargaMod);        //CARGA

}
```

El proceso de insertar la carga consiste principalmente generar pixeles modificados que contienen la información de los metadatos.
Esto se hace por partes ya que primero genero los pixeles que almacenan el tamaño de la carga (Información necesaria para poder extraerla después) y por otro lado genero los pixeles que van a contener la carga.
Una vez que se tienen los pixeles se cambian por los originales.

Los canales RGB usados por defecto son rojo, verde y azul en esta versión, aunque esta implementación se pensó para poder seleccionar los canales que se quieren usar, por ejemplo selecciono el rojo y el azul pero el verde no cosa que actualmente no funcionaría porque habría que implementarlo en más sitios pero que en versiones futuras se le podría hacer.

- **Extracción de datos**
  Es el proceso inverso a la Inserción de datos, consiste en obtener la información de los pixeles de la imagen de entrada.
  Este proceso se implementa en el método extraer carga de la clase componentes.EstegoMaquina.

```java
/**
 * Extrae de la imagen el la carga insertada
 */
private void extraerCarga() {

    //INICIALIZAR VARIABLES

    int contadorBitActual = 0;              //Bit actual en la extraccion
    boolean primeraInteraccion = true;      //Control de primera interaccion del bucle
    Boolean[] cargaBinariaExtraida;         //Carga binaria extraida
    boolean procesoCompletado = false;      //Control de fin de procesado

    //Obtener el numero de canales usados
    int numeroCanalesUsados = 3;

    //Obtener el tamaño de la carga
    obtenerTamCarga();

    //-----------------------EXTRACCION DE LA CARGA --------------
    //----|--
    //----v--

    cargaBinariaExtraida = new Boolean[tamCargaBin];

    //Bucle que recorre todos los pixeles implicados
    while (!procesoCompletado) {

        //Si es el primer bit de la interaccion no se actualiza el pixel.
        if (primeraInteraccion) {
            primeraInteraccion = false;
        } else {
            //Si no se ha completado el numero de bits que se deben de
            //extraer de el pixel actual
            //no se actualiza el pixel
            if (contadorBitActual % numeroCanalesUsados == 0) {
                nextPixel();
            }

        }

        //Recorre los canales activos del color
        //Extrae el valor de la carga que hay en cada canal

        int numCanal = 0;
        for (boolean canal : canalesRGB) {

            //Control de ultimo bit de carga alcanzado
            if (contadorBitActual == tamCargaBin) {
                procesoCompletado = true;
                break;
            }

           //Control de canal activo
           if (canal) {

               //Obtener el valor del canal RGB correspondiente
               float valorGamaActual = pixelActual.getColorRGB().getArrayRGB()[numCanal];

               //Obtener carga
               //Si el valor de la gama es par, su carga sera true
               cargaBinariaExtraida[contadorBitActual] = valorGamaActual % 2 == 0;
               contadorBitActual++;
           }

           numCanal++;
       }
    }

    //----^--
    //----|--
    //-----------------------------------------------------------------FIN


    //Transformar de binario a caracteres
    extraccion = Carga.binaryDecode(cargaBinariaExtraida);
    extraccion = extraccion.substring(0, extraccion.length() - 1);
}
```

Consiste en primero obtener el tamaño de la carga para poder extraer la carga binaria después, transformar ésta a una cadena de caracteres para a partir de esta reconstruir el objeto con los metadatos y poder visualizarlos o modificarlos.

2.2.2. Interface

En la interface podemos encontrar lo que es la aplicación en sí, o sea los procesos que puede realizar con ella.

Hay dos grupos diferenciados:

Generar EstegoImagen

Consiste en generar una imagen idéntica a la original, sin los metadatos en el EXIF del archivo, pero con la información de estos insertada esteganográficamente en la imagen.

En concreto lo conforman las funcionalidades de sacar foto, seleccionar imágenes y modo automático.

Obtener los metadatos

Consiste en extraer los metadatos de la imagen modificada.

Las funcionalidades de esta categoría son reestablecer imagen y visualizar metadatos.

Información de uso

En el menú principal hay un icono que abre una activity en la que se muestran el resto de iconos del menú principal junto con una pequeña descripción de la funcionalidad que realiza cada uno.

3. Recogida de información

La idea sobre cómo funcionaba la técnica de esteganografía empleada ya la tenía de información que había consultado previamente por internet y sabia más o menos como implementarlo, aun así busque más información, vi que existen más técnicas, que la que yo uso se puede mejorar o adaptar según qué necesidades se tenga, como guardar más información por píxel, etc.

La información sobre los formatos de imágenes también tuve que mirarla, ya que no todos los formatos se pueden usar con esta técnica de esteganografía ya que por ejemplo JPG tiene un algoritmo de compresión y el conjunto de pixeles en memoria no es el mismo una vez guardado (al guardarse se aplica la compresión y un pixel que tenía un color puede que tenga otro) ya que el algoritmo iguala los colores que son muy parecidos -aplica la compresión- para ahorrar espacio. Png no tiene compresión por lo que se puede usar perfectamente. Sería interesante estudiar otros tipos de estándares como Lossless JPEG o JPEG XT a ver qué novedades aportan y si se podrían usar.

Respecto a los metadatos ahí tuve que investigar más, ya que no sabía nada sobre conceptos básicos, como que es el Exif, que metadatos tenía una imagen, etc. Use herramientas que te permiten visualizar metadatos e imágenes para ver que se almacenaba. Una vez que ya supe de que iba el tema empecé a buscar librerías en java que me permitiesen leer y escribir metadatos en una imagen, encontré apache/Sanselan y me sirvió, también se podía usar en Android pero buscando más información descubrí la clase nativa ExifInterface que me facilitó mucho las cosas.

He buscado información sobre los permisos de Android y que versiones mínimas necesitaba mi aplicación, sobre procesos en segundo plano, como se codifican los colores en la informática, y sobre el AndroidStudio que me he tenido que actualizar a la nueva versión y mirar proyectos antiguos para recordar cómo se hacían las cosas. También tuve que investigar en internet como se hacían algunas cosas.

4. Descripción del sistema actual

Cuando se inicia la aplicación se inicia una activity que dura 3 segundos en la que se muestra el logo de la aplicación y seguido se inicia la activity del menú principal.

El menú principal consta de 6 botones, la fila de arriba son los procedimientos relacionados con crear la EstegoImagen.

El botón de sacar foto abre una activity que te permite tomar una foto, inmediatamente después, ésta es procesada y sus metadatos son insertados en los píxeles de la imagen según el algoritmo esteganográfico. Posteriormente la nueva imagen se guarda en la carpeta de la aplicación en el almacenamiento externo del dispositivo.

El botón de seleccionar foto realiza la misma acción que el de sacar foto, la diferencia es que éste abre una activity tipo FileChooser (Seleccionador de archivos) que permite seleccionar desde el almacenamiento del teléfono una o varias imágenes, que después serán procesadas.

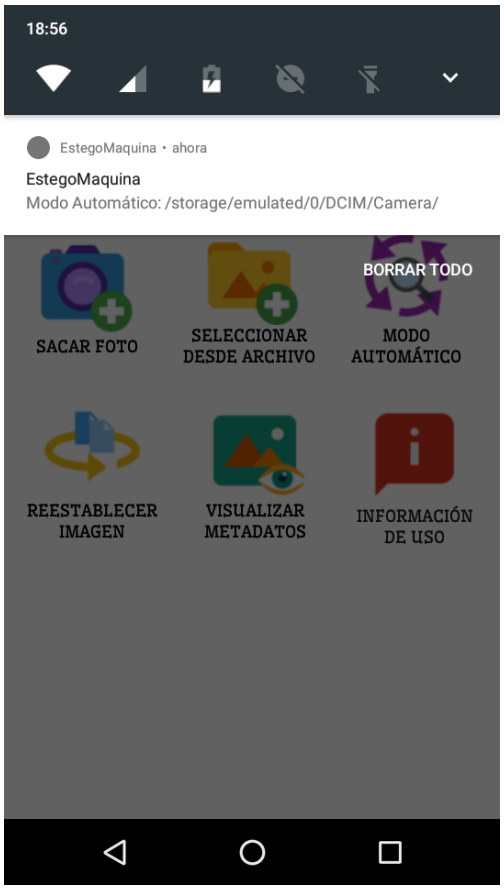
Durante el procesado de imágenes que producen 'Sacar Foto' y 'Seleccionar desde archivo' en el menú principal aparece una barra de progreso de tipo circular y un campo de texto en el que pone 'Procesando…' para hacer saber al usuario que es lo que está pasando. Una vez completado el proceso la barra de progreso desaparece y en el texto se escribe 'Completado'.

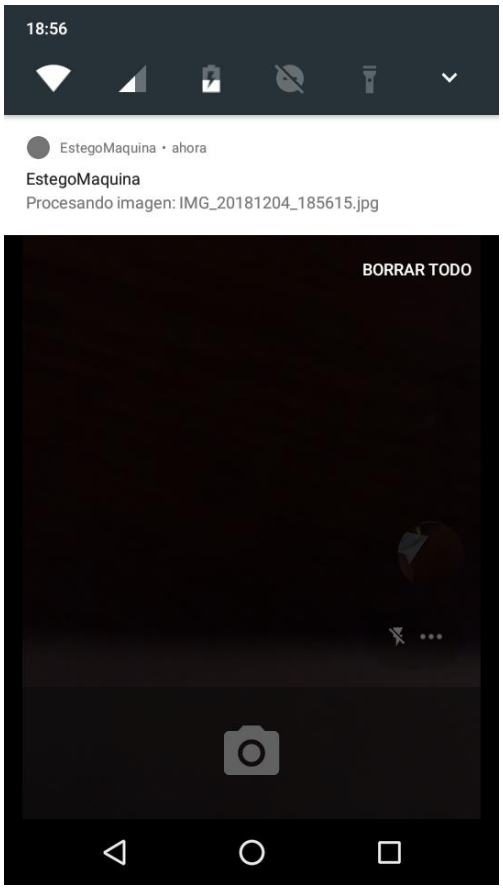
El modo automático no es estable, ya que no he sabido bien cómo hacerlo, lo he añadido a la aplicación para mostrar otra posible funcionalidad que puede incorporar en versiones posteriores.

Al pulsar sobre el botón de modo automático se inicializa un servicio que detectará cuando se ha añadido una nueva imagen al directorio de

almacenamiento de la cámara del dispositivo, de forma que el usuario puede dejar la aplicación ejecutándose en segundo plano (O incluso cerrarla) y usar su aplicación favorita para tomar fotografías, de forma que cada vez que una nueva fotografía es detectada por el servicio, automáticamente será procesada.

Al iniciarse el servicio se muestra una notificación al usuario, y cada vez que una imagen es procesada aparece otra notificación informando de que se está procesando tal imagen.





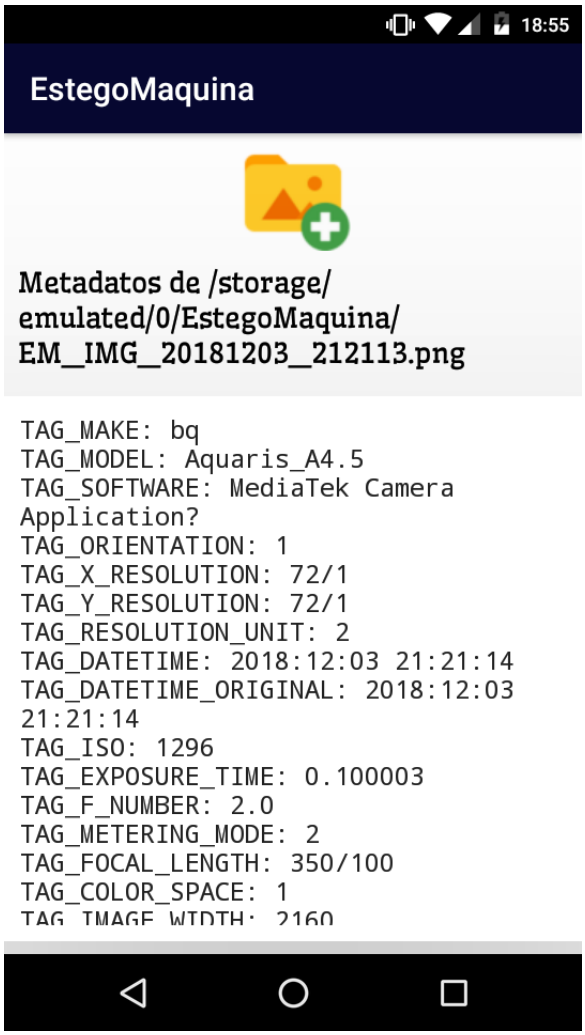
El botón de reestablecer imagen recibe una imagen anteriormente procesada por la aplicación, es decir con los metadatos insertados esteganográficamente en sus pixeles, extrayendo los mismos y generando una nueva imagen con el Exif original.

La interface es similar a la del botón ‘Seleccionar desde archivo’, se usa una activity de tipo FileChooser para seleccionar una o varias imágenes y comienzan

a reestablecerse. Como antes, también aparece el ProgressBar y el campo de texto para indicar su estado actual de procesado.

El botón de 'Visualizar Metadatos' sirve para visualizar los metadatos que lleva una EstegoImagen en sus pixeles, o sea la información esteganográfica que lleva.

Al pulsarlo se abre una activity de visualización que contiene un botón que simboliza el seleccionar desde archivo, que al pulsarle abre un FileChooser para seleccionar la imagen que se desea visualizar. Una vez seleccionada se indica el nombre del archivo en un campo de texto y en otro campo de texto deslizable se ponen los metadatos.



Por último queda el botón de información, que fue pensado para solucionar confusiones que puede tener el usuario sobre qué es lo que hace cada funcionalidad. Al pulsar sobre el botón, se abre una activity que contiene los botones que son mostrados en el menú principal y debajo de cada uno una breve descripción de la funcionalidad que realiza cada uno.

5. Especificación de requisitos

Se requieren los permisos de acceso a la cámara y al almacenamiento ya que se van a hacer fotos estas deben de ser guardadas en el dispositivo.

La versión mínima requerida es Android 7.0 Nivel de API 24 (julio 2016).

Esta optimizado para que se adapte a todas las pantallas (Responsive).

6. Glosario de términos

Esteganografía

(Del griego steganos, "cubierto" u "oculto", y graphos, "escritura") trata el estudio y aplicación de técnicas que permiten ocultar mensajes u objetos, dentro de otros, llamados portadores, de modo que no se perciba su existencia.

EstegoImagen

Se entiende como una imagen que tiene información esteganográfica en su interior, en este caso los metadatos.

Metadatos

Los metadatos (del griego meta, 'después de, más allá de' datum, 'lo que se da'), literalmente «sobre datos», son datos que describen otros datos. En general, un grupo de metadatos se refiere a un grupo de datos que describen el contenido informativo de un objeto al que se denomina recurso.

Exif

Exchangeable image file format (en español, *Formato de archivo de imagen intercambiable*) es una especificación para formatos de archivos de imagen usado por las cámaras digitales.

Exif Tag

Las etiquetas (*tags*) de metadatos definidas en el estándar Exif cubren un amplio espectro incluido:

- Información de fecha y hora.
- Configuración de la cámara.
- Información sobre localización.

- Descripción e información sobre copyright

RGB

RGB (sigla en inglés de *red, green, blue*, en español «rojo, verde y azul») es la composición del color en términos de la intensidad de los colores primarios de la luz.

Canal RGB

Un canal es cada componente del conjunto RGB, el rojo, el verde y el azul son los canales usados en esta aplicación.

Serializar

Consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como XML o JSON, entre otros. La serie de bytes o el formato pueden ser usados para crear un nuevo objeto que es idéntico en todo al original, incluido su estado interno (por tanto, el nuevo objeto es un clon del original).

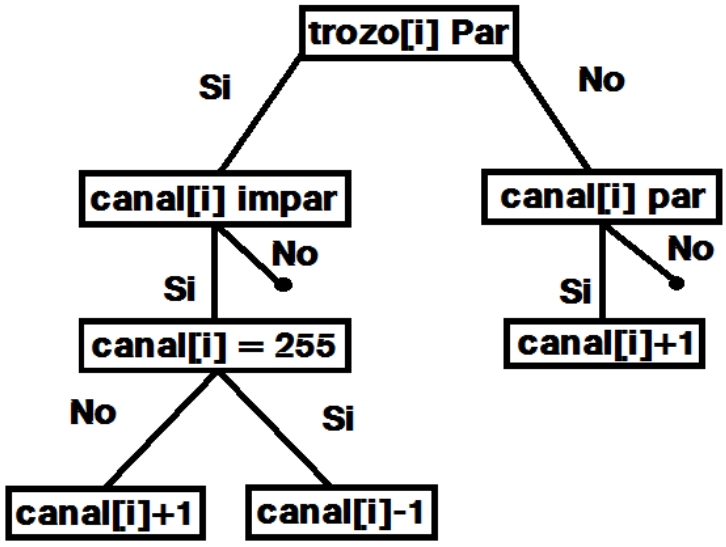
Base64

Base 64 es un sistema de numeración posicional que usa 64 como base. Es la mayor potencia de dos que puede ser representada usando únicamente los caracteres imprimibles de ASCII. Esto ha propiciado su uso para codificación de correos electrónicos, PGP y otras aplicaciones. Todas las variantes famosas que se conocen con el nombre de Base64 usan el rango de caracteres A-Z, a-z y 0-9 en este orden para los primeros 62 dígitos.

7. Descripción mediante UML

7.1. Procesado de paridad

Es la técnica usada para codificar la información dentro de un color.



Por ejemplo tenemos un trozo de carga[]={True, False, False} que equivaldría a …011…

(El trozo de carga es una fracción de la carga binaria completa, de tamaño igual al número de canales usados)

Tenemos un color rojo rgb(255, 11, 22). Según el algoritmo, por cada posición del array de trozos, se obtiene el primero, en este caso es trozo[0] = true, (que se toma como 0 o par).

Como el Canal[0], o canal R tiene valor 255 o sea impar, se redondeará en 254 ya que el trozo de carga marcaba Par.

Siempre se redondea hacia arriba exceptuando el 255 a par que se hace hacia abajo por ser 255 cota superior.

Repitiendo este proceso con todos los trozos y canales se obtiene:

Carga[0] = 0 y Canal[0] = 255 => nuevo valor del Canal[0] = 254

Carga[1] = 1 y Canal[1] = 11 => nuevo valor del Canal[1] = 11

Carga[2] = 1 y Canal[2] = 22 => nuevo valor del Canal[2] = 23

El color rgb(255, 11, 22) con la carga{True, False, True} producen el color rgb(254, 11, 23) que es muy similar al original.

7.2. Técnica sustitución del bit menos significativo

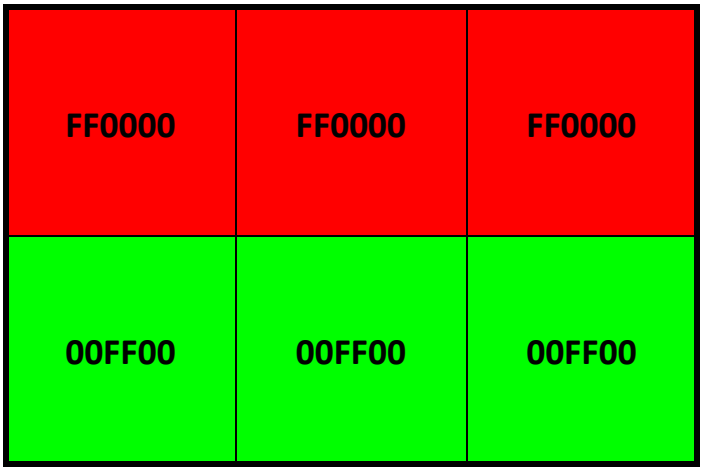
Un color RGB es representado por tres valores, cada uno correspondiente a una gama, que en decimal van del 0 al 255, aunque comúnmente se suelen representar en Hexadecimal.

rgb(255,0,0) = rgb(FF,00,00) = Rojo puro

Esta técnica consiste en modificar el valor de cada canal RGB del color un tono, con el objetivo de poder almacenar un valor **(0-1)** modificando mínimamente el color original.

Por ejemplo:

Partimos de una imagen compuesta por 6 pixeles, cada uno es del color que tiene escrito en su interior.



La carga que se va a almacenar es **“AB”**, cuya representación binaria es **0100000101000010.**

Vemos que el primer pixel es del color **FF0000**. Esto significa que:

El canal R tiene un valor de 0xFF = 255 (Valor máximo)

El canal G tiene un valor de 0x00 = 0 (Valor mínimo)

El canal B tiene un valor de 0x00 = 0

Tenemos 3 canales por pixel, por lo que podemos almacenar 3 símbolos binarios de la carga por pixel.

Suponiendo que cada carácter ocupa 8 bits (Los caracteres letra en codificación UTF-8 toman los valores ASCII de 0 a 127) se necesitan 3 pixeles para cada símbolo y sobra un bit.

Los 3 primeros dígitos de AB = 010 se van a almacenar de la siguiente forma:

- **0** => Numero par => Valor de **R va a ser par**
- **1** => Número impar => Valor de **G va a ser impar**
- **0** => Numero par => Valor de **B va a ser par**

De esta forma si el pixel 1 era FF0000 o lo que es lo mismo (255, 0, 0) ahora pasara a ser **FE0100 (254, 1, 0)**

Este proceso se repite con todos los símbolos de la carga.

Al final del proceso, a partir de la imagen original y la carga se obtiene la siguiente imagen.

FE0100	**FE0000**	**FE0100**
01FE00	**00FE01**	**00FE00**

La extracción de la carga es el proceso inverso:

Para el primer pixel tenemos **FE0100**, o lo que es lo **mismo (254, 1, 0)**

Esto quiere decir que:

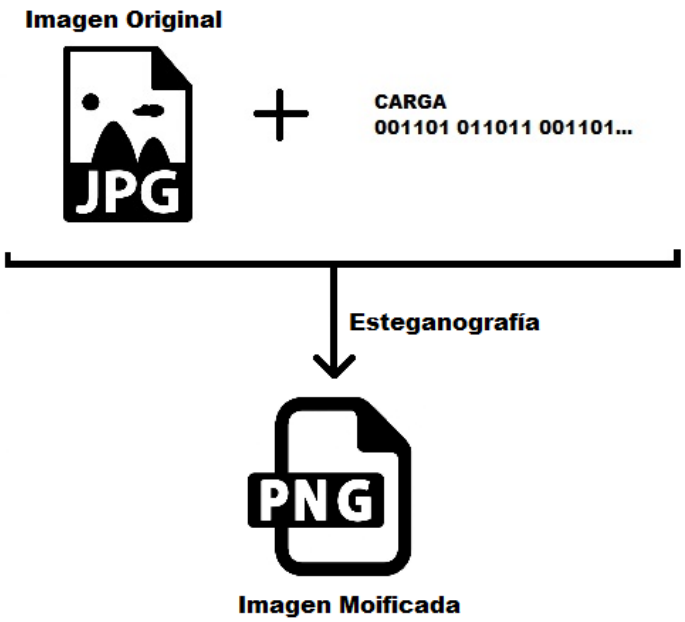
- 254 => Número par => Valor de R va a ser par => 0
- 1 => Número impar => Valor de R va a ser impar => 1
- 0 => Número par => Valor de R va a ser par => 0

Por lo tanto el pixel 1 contiene la carga **010**.

Repitiendo el proceso con todos los pixeles modificados extraemos la carga **0100000101000010, que es la representación binaria de "AB"**

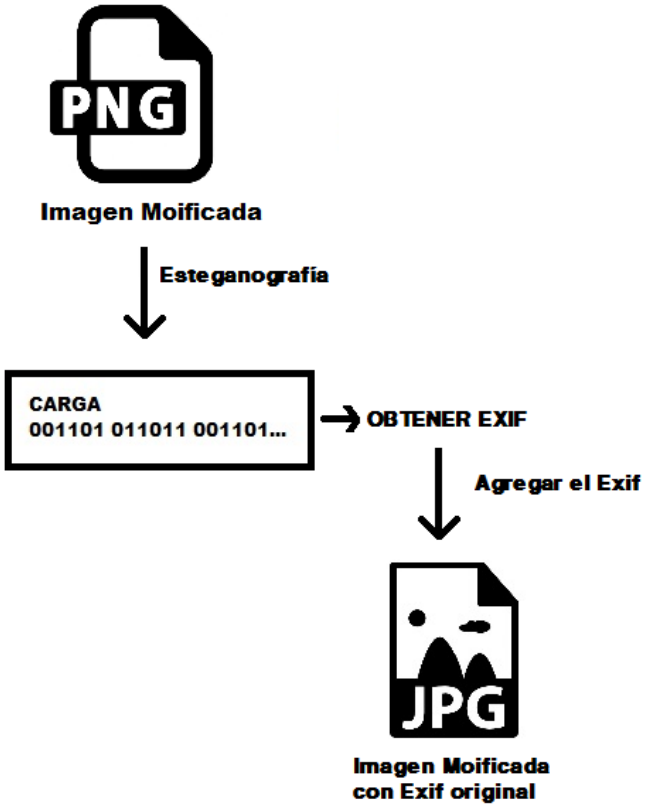
7.3. Proceso de inserción de metadatos.

De la imagen original se extrae el exif y se procesa para que sea una cadena de unos y ceros. Mediante la técnica esteganográfica de modificación del bit menos significativo se añade la información a la imagen resultante que es idéntica a la original y se guarda con formato png.



7.4. Proceso de extracción de metadatos

Partiendo de la imagen modificada, se extrae la carga mediante la técnica esteganográfica de sustitución del bit menos significativo. La información extraída se procesa y se construye un objeto ExifData que a partir de él se podrá hacer varias cosas como añadírselo a una imagen o visualizar los metadatos.



8. Referencias Bibliográficas

La información mayoritariamente ha sido sacada de internet, muchas fuentes ya no las recuerdo, también han influido alguna noticia que me llamó la atención, y algún documental.

He destacado 3 sitios web por ser las principales fuentes reconocidas.

Android Developers, de donde he sacado la mayor parte información sobre Android, como Asynctask, Colors, FileObserver, Bitmap, etc.

Stackoverflow, donde he consultado dudas similares a las mías, que habían preguntado otros usuarios y han sido resueltas por la comunidad.

Programcreek, donde he visto ejemplos de código fuente, que he podido usar.