# README

Carlos Cabrera Ramírez Sergio Medina Guzmán

Abril 2022

## Esbozo del proceso de solución

Al discutir el proyecto, pensamos principalmente en 3 módulos: uno encargado de hacer el encoding, otro para el decoding y el principal que se encarga de invocar los métodos de estos y hacer el manejo de los archivos. Adicionalmente se creó un módulo más para manejar los bits menos significativos, la conversión del mensaje por esconder a su representación como cadena en binario.
Como datos de entrada recibimos las rutas del archivo png o jpeg donde queremos esconder nuestro mensaje y del archivo de texto donde se encuentra el mensaje por esconder en la imagen, en caso de querer codificar un mensaje; y en caso de querer extraer un mensaje oculto de una imagen png, se proporciona el nombre "encoded.png" que es el archivo donde se encuentra el mensaje oculto y la ruta del archivo de texto donde se guardará el mensaje extraído.
Como salida el programa genera una imagen png llamada "encoded.png" que es donde se encuentra el mensaje oculto o bien escribe en el archivo de texto cuya ruta fue proporcionada por el usuario el mensaje que se encontraba oculto en la imagen.
Al elegir la codificación de un mensaje, el módulo steganography.py invoca la función "encode_image" que, dependiendo del modo de la imagen proporcionada, si es rgb o rgba, invoca funciones de encoder.py: "encode_rgb" o "encode_rgba", que reciben la cadena de texto leída del archivo de texto proporcionado por el usuario para convertirlo a una cadena de bits que, además, será normalizada para que su longitud sea un múltiplo del número de canales que contenga la imagen. Además, se destinan los primeros 16 bits menos significativos para almacenar la longitud de la cadena de bits que contiene nuestro mensaje oculto, de manera que al hacer la extracción del mensaje sea fácil saber en qué momento dejar de consultar bits. Una vez que se tiene la cadena de bits normalizada, se invoca la función "lsb_manager" que edita el bit menos significativo de cada canal de cada pixel de manera que coincidan con los bits que se encuentran en la cadena de bits que contiene la longitud del mensaje y el mensaje mismo.
En cuando al módulo "decoder.py", contiene una función que consulta los primeros 16 bits menos significativos de los canales de los pixeles de la imagen codificada, para saber en qué momento debe dejar de extraer dichos bits de la imagen para tener el mensaje completo como cadena de bits. Luego, el método "get_hidden_msg" va consultando los bits menos significativos a partir del 17o bit menos significativo de los canales de los pixeles de la imagen y los agrega a una cadena que posteriormente se decodifica con la función "decode_binary" que se encuentra en el módulo "binary_converter" para finalmente devolver el mensaje y almacenarlo en el archivo que el usuario especifica.
Cabe mencionar que cada caracter al convertirse a su representación binaria, no siempre es una cadena de 8 bits, por lo que la función zfill se encarga de que toda cadena de bits que representa un caracter sea de longitud 8, agregando ceros a la izquierda pues al decodificar el mensaje la cadena se separa en subcadenas de longitud 8 y cada una de estas se convierte en el caracter correspondiente.