
**PITEA: Protección de Información mediante
Técnicas de Esteganografía Acústica**
**IPAST: Information Protection through
Acoustic Steganography Techniques**



**Trabajo de Fin de Grado
Curso 2024–2025**

Autores

Alberto Martín Oruña Nota: 10
Rodrigo Gallego Marín Nota: 10

Directores

José Luis Vázquez Poletti
Juan Carlos Fabero Jiménez

**Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid**

PITEA: Protección de Información mediante Técnicas de Esteganografía Acústica

IPAST: Information Protection through
Acoustic Steganography Techniques

Trabajo de Fin de Grado en Ingeniería Informática

Autores

Alberto Martín Oruña Nota: 10
Rodrigo Gallego Marín Nota: 10

Directores

José Luis Vázquez Poletti
Juan Carlos Fabero Jiménez

Convocatoria: *Junio 2025*

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Dedicatoria

*A nuestros padres, por su amor incondicional y
apoyo constante. A nuestros compañeros por
su compañía durante toda la carrera y a
nuestros amigos por hacer estos cuatro años
más amenos*

Agradecimientos

Agradecemos a nuestros tutores por su paciencia y apoyo durante el desarrollo de este trabajo. Agradecemos a todos los profesores que hemos tenido en esta etapa ya que son los que nos han formado con los conocimientos necesarios para desarrollar este proyecto.

Resumen

PITEA: Protección de Información mediante Técnicas de Esteganografía Acústica

PITEA es una herramienta en línea de comandos que permite la ocultación de archivos de texto mediante el cifrado de estos, su posterior ocultación o transformación en imágenes y, finalmente, en audio. También permite realizar el proceso inverso. La aplicación combina técnicas de criptografía y esteganografía para ocultar la información que pueda ser transmitida en la red pública, aprovechando la naturaleza digital de las imágenes y el sonido. A través de nuestra herramienta, los usuarios podrán garantizar la confidencialidad de sus datos, eligiendo entre distintos métodos de ocultación. El objetivo es desarrollar una herramienta versátil y accesible para cualquier persona con un mínimo conocimiento del uso de la línea de comandos, permitiendo proteger información sensible “a plena luz del día”. Este enfoque amplía las aplicaciones de la criptografía y la esteganografía, ofreciendo nuevas formas de ocultar y proteger información digital.

Palabras clave

Criptografía, esteganografía, línea de comandos, ocultación, cifrado, red pública, confidencialidad, información sensible, protección de datos.

Abstract

IPAST: Information Protection through Acoustic Steganography Techniques

IPAST is a command-line tool that enables the concealment of text files through encryption, followed by their embedding or transformation into images, and ultimately into audio. It also allows for the reverse process. The application combines cryptography and steganography techniques to hide information that may be transmitted over the public network, taking advantage of the digital nature of images and sound. Through our tool, users can ensure the confidentiality of their data by choosing from various concealment methods. The goal is to develop a versatile and accessible tool for anyone with basic knowledge of command-line usage, allowing them to protect sensitive information “in broad daylight”. This approach expands the applications of cryptography and steganography, offering new ways to hide and protect digital information.

Keywords

Cryptography, steganography, command-line, concealment, encryption, public network, confidentiality, sensitive information, data protection.

Índice

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Plan de trabajo	2
1.3.1. Etapa 1: Investigación previa	2
1.3.2. Etapa 2: Desarrollo de un prototipo funcional básico	2
1.3.3. Etapa 3: Modularización y ampliación de funcionalidades	3
1.3.4. Etapa 4: Creación de la CLI	3
1.3.5. Etapa 5: Pruebas de la herramienta	3
1.3.6. Etapa 6: Documentación	4
2. Estado de la Cuestión	5
2.1. Codificación y decodificación de imagen y audio	5
2.1.1. Imagen	5
2.1.2. Audio	6
2.2. Esteganografía	8
2.2.1. Orígenes	8
2.2.2. Descripción	9
2.2.3. Tipos de esteganografía	9
2.2.4. Esteganografía en contenido multimedia	10
2.3. Criptografía	12
2.3.1. Criptografía de clave secreta	12
2.3.2. Criptografía de clave pública	16
2.4. Transmisión de datos mediante SSTV	18
2.5. Comparación con otras herramientas	20
3. Arquitectura del Sistema	23
3.1. Diagramas de clases	23
3.2. Diagramas de secuencia	30
4. Pruebas experimentales	41
4.1. Transmisión vía radiofrecuencia	41

4.1.1. Generación del archivo portador	41
4.1.2. Resultado	42
4.2. Difusión en la red pública	43
4.2.1. Generación de los archivos portadores	44
4.2.2. Resultado	45
4.3. Análisis de <i>OcultadorText</i>	47
4.3.1. Análisis de <i>Tesseract-OCR</i>	48
4.3.2. Análisis de <i>EasyOCR</i>	50
5. Tecnologías descartadas	55
6. Conclusiones y Trabajo Futuro	57
6.1. Conclusiones de la herramienta	57
6.2. Trabajo futuro	57
Introduction	59
6.3. Motivation	59
6.4. Objectives	60
6.5. Work plan	60
6.5.1. Stage 1: Preliminary Research	60
6.5.2. Stage 2: Development of a Basic Functional Prototype	60
6.5.3. Stage 3: Modularization and Enhancement of Functionalities .	61
6.5.4. Stage 4: Creation of the CLI	61
6.5.5. Stage 5: Tool Testing	61
6.5.6. Stage 6: Documentation	61
Conclusions and Future Work	63
6.6. Conclusions of the tool	63
6.7. Future Work	63
Contribuciones Personales	65
Bibliografía	71
A. Guía de instalación	73
B. Manual de uso	75
B.1. Ejemplos de Ejecución	75
B.1.1. Uso de algoritmos de ocultación basados en LSB	77
B.1.2. SSTV	79
B.2. Documentación del archivo de configuración, “configuracion.toml” .	83
C. Información extra del análisis de <i>OcultadorText</i>, 4.3	85
C.1. Fuentes utilizadas	85
C.2. Frases utilizadas	85

Índice de figuras

1.1.	Uso de redes sociales por país.	1
2.1.	Representación de una imagen en formato PNG.	6
2.2.	Representación de un audio estéreo en formato WAV.	7
2.3.	Diagrama de funcionamiento de la esteganografía.	9
2.4.	Ejemplo de LSB en una imagen PNG sin canal alfa.	10
2.5.	LSB usando dominio de la transformada con la transformada de Fourier.	11
2.6.	Diagrama criptografía de clave secreta.	12
2.7.	Ejemplo cifrado César.	13
2.8.	Ciclo básico del algoritmo de Rijndael.	14
2.9.	Diagrama criptografía de clave pública.	16
2.10.	Firmado con clave pública.	17
2.11.	Onda de audio usando SSTV para imágenes en blanco y negro.	19
2.12.	Decodificación de un audio en SSTV usando QSSTV.	19
2.13.	Ejemplos de uso de <i>Steghide</i>	20
2.14.	Pantalla de inicio de <i>SilentEye</i>	21
2.15.	Menú <i>encode</i> de <i>SilentEye</i>	22
3.1.	Diagrama de clases CLI.	24
3.2.	Diagrama de clases: Main.	25
3.3.	Diagrama de clases: Cifradores.	26
3.4.	Diagrama de clases: Ocultadores en imagen.	28
3.5.	Diagrama de clases: Ocultadores en audio.	29
3.6.	Diagrama de clases: Fábricas.	29
3.7.	Diagrama de secuencia: <i>Launch</i> parte 1.	30
3.8.	Diagrama de secuencia: <i>Launch</i> parte 2.	30
3.9.	Menú de inicio.	31
3.10.	Diagrama de secuencia: Comando de ocultación.	31
3.11.	Opciones a llenar por el usuario.	32
3.12.	Diagrama de secuencia: Llamada al flujo de ocultación.	32
3.13.	Diagrama de secuencia: Creación del Cifrador.	33
3.14.	Diagrama de secuencia: Cifrado de los datos.	33

3.15. Diagrama de secuencia: Creación de los ocultadores.	33
3.16. Diagrama de secuencia: Ocultación en la imagen.	34
3.17. Imagen con ocultación por LSB transformada.	34
3.18. Imagen con ocultación por LSB transformada.	34
3.19. Diagrama de secuencia: Ocultación en un audio mediante técnica LSB.	35
3.20. Diagrama de secuencia: Modulación SSTV de la imagen.	35
3.21. Diagrama de secuencia: Comando de desocultación.	36
3.22. Diagrama de secuencia: Llamada al flujo de desocultación.	36
3.23. Diagrama de secuencia: Creación de los ocultadores.	37
3.24. Diagrama de secuencia: Desocultación de una imagen mediante LSB.	37
3.25. Diagrama de secuencia: Decodificación del audio SSTV.	38
3.26. Pasos de decodificación SSTV.	38
3.27. Diagrama de secuencia: Recuperación del mensaje oculto en la imagen.	38
3.28. Diagrama de secuencia: Creación del Cifrador.	39
3.29. Diagrama de secuencia: Descifrado de los datos.	39
4.1. Generación de audio portador, AES, <i>text</i> , SSTV.	42
4.2. Imagen generada por el <i>OcultadorText</i>	42
4.3. Interfaz del nodo de radioescucha utilizado.	43
4.4. Imagen recuperada en la prueba de campo, 4.1.	43
4.5. Generación de audio portador, sin cifrado, LSB, LSB.	44
4.6. <i>Collage</i> de las imágenes sultante de la generación del archivo 1.	44
4.7. Captura en <i>streaming</i> en Tiktok.	45
4.8. Imágenes resultantes de la prueba con el archivo 2.	45
4.9. Llamada a la herramienta para la desocultación de 1.	46
4.10. Imágenes resultantes de la desocultación del archivo 1.	46
4.11. Gráfica de resultado por tamaño de la fuente utilizada. <i>Tesseract-OCR</i>	48
4.12. Gráfica de resultado por fuente utilizada. <i>Tesseract-OCR</i>	49
4.13. Gráfica de resultado por anchura de la imagen. <i>Tesseract-OCR</i>	49
4.14. Gráfica de todos los parámetros configurables. <i>Tesseract-OCR</i>	50
4.15. Gráfica de resultado por longitud y números. <i>Tesseract-OCR</i>	50
4.16. Gráfica de resultado por tamaño de la fuente utilizada. <i>EasyOCR</i>	51
4.17. Gráfica de resultado por fuente utilizada. <i>EasyOCR</i>	51
4.18. Gráfica de resultado por anchura de la imagen utilizada. <i>EasyOCR</i>	52
4.19. Gráfica de todos los parámetros configurables. <i>EasyOCR</i>	52
4.20. Gráfica de todos los parámetros configurables. <i>EasyOCR</i>	53
6.1. Social media usage time by country	59
B.1. Ocultación LSB.	78
B.2. Descultación LSB.	78
B.3. Ocultación SSTV.	79
B.4. Marcar <i>autoslant</i>	79
B.5. Descultación SSTV iniciando con una imagen.	80

B.6. Configuración QSSTV para archivos.	80
B.7. Descultación SSTV.	81
B.8. Configuración QSSTV para <i>streaming</i>	81
B.9. Descultación SSTV en modo <i>streaming</i>	82
B.10. Descultación SSTV iniciando con un archivo que contiene base64. . .	83

Índice de tablas

2.1. Frecuencias típicas para transmisión SSTV	19
4.1. Ejemplo de <i>datasets</i> generados	47

Capítulo 1

Introducción

1.1. Motivación

La principal motivación por la que hemos decidido hacer este proyecto es que, según Kemp (2024), se mueven un gran volumen de datos en redes sociales (medible en la escala de exabytes), tienen una gran cantidad de usuarios, superando los 5 mil millones de usuarios activos y un tiempo medio de uso de 2 horas y 23 minutos al día, pero nos mantenemos 6 horas y 40 minutos en línea de media diariamente. Además, la red social más utilizada en 2024 fue TikTok, la cual se basa en videos de corta duración, admitiendo en sus orígenes videos de hasta 60 segundos y actualmente hasta 60 minutos. Esto hace que el paso de información secreta a través de vídeos con audio en la red pública pase desapercibido debido a su abusivo pero común uso en la actualidad.

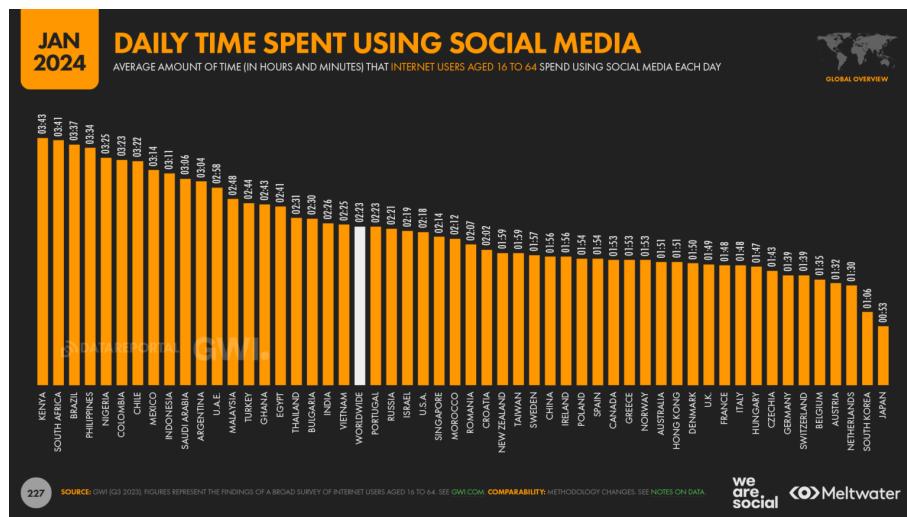


Figura 1.1: Gráfica del tiempo de uso de redes sociales por país. Kemp (2024).

Para evitar riesgo de confidencialidad en los datos, se vuelve fundamental el uso de la esteganografía, que será introducida con más detalle en este documento pero por ahora la definiremos como técnicas de ocultación de información, para garantizar que, si alguien obtiene un video de manera lícita o ilícita, no sea capaz de detectar

que contiene información valiosa, debido a que ésta está oculta y aparenta ser un video más en el mar de bytes.

1.2. Objetivos

El objetivo de este proyecto es desarrollar una herramienta capaz de utilizar métodos de transformación de texto a imágenes y de imágenes a audio con el fin de transmitir información de manera oculta y aprovechar el intenso uso de las redes sociales que hay en la actualidad.

Para ello, nos centraremos en:

- Creación de una CLI (*Command-line interface*).
- Implementación de algoritmos para la ocultación y desocultación tanto de texto en imagen como de imagen en audio.
- Tratamiento de ondas acústicas en la informática.
- Uso de criptografía de clave secreta.

1.3. Plan de trabajo

En esta sección se describe el plan de trabajo detallado con el que se espera alcanzar los objetivos del proyecto.

1.3.1. Etapa 1: Investigación previa

Objetivo: Realizar una investigación previa al desarrollo de la herramienta con el fin de situarse en los conocimientos y técnicas necesarias.

Tareas:

- Búsqueda de información sobre esteganografía y criptografía.
- Búsqueda de información sobre la codificación de archivos multimedia.
- Análisis de las herramientas existentes.

1.3.2. Etapa 2: Desarrollo de un prototipo funcional básico

Objetivo: Crear un prototipo funcional de la herramienta que realice la ocultación y desocultación mediante LSB (*Less Significant Bit*).

Tareas:

- Desarrollar un programa secuencial que realice el proceso completo de ocultar y desocultar.

- Implementar las funcionalidades básicas, como cifrar el texto mediante AES y ocultar el texto y la imagen mediante el método de ocultación LSB.
- Realizar pruebas para verificar que el sistema básico funcione correctamente y la información no se modifica en el proceso.

1.3.3. Etapa 3: Modularización y ampliación de funcionalidades

Objetivo: Modularizar el programa y añadir funcionalidades adicionales, como generar audios a partir de las imágenes para transmitirlos por SSTV.

Tareas:

- Dividir el código en módulos haciendo un uso apropiado de la POO (Programación Orientada a Objetos) para separar las responsabilidades de cifrado, esteganografía en imágenes, esteganografía en audio y manejo de entradas y salidas.
- Implementar funcionalidades adicionales, como la capacidad de elegir entre diferentes técnicas de ocultación.
- Realizar pruebas de integración para asegurarse de que los módulos funcionan bien juntos.

1.3.4. Etapa 4: Creación de la CLI

Objetivo: Crear la interfaz de línea de comandos para facilitar la interacción con el programa.

Tareas:

- Implementar la interfaz de línea de comandos.
- Integrar la interfaz con los módulos desarrollados previamente.
- Asegurar que los usuarios solo puedan elegir flujos posibles de ejecución.

1.3.5. Etapa 5: Pruebas de la herramienta

Objetivo: Realizar pruebas sobre la herramienta para asegurar la fiabilidad y descubrir las limitaciones del sistema.

Tareas:

- Realizar pruebas para comprobar la fiabilidad de la herramienta empleándola en casos reales de uso.
- Realizar pruebas para estudiar las limitaciones de la herramienta.

1.3.6. Etapa 6: Documentación

Objetivo: Documentar la arquitectura de la herramienta así como las conclusiones a las que hemos llegado.

Tareas:

- Crear documentación detallada sobre el código, las funcionalidades, el uso de la herramienta, la instalación de esta, las tecnologías usadas y también las descartadas.
- Preparar una presentación para mostrar el proyecto y sus resultados finales.

Capítulo 2

Estado de la Cuestión

En este capítulo se muestran las conclusiones obtenidas de la investigación previa a la implementación de la herramienta, es importante para situarse en los conocimientos y técnicas que existen para el fin de esta.

2.1. Codificación y decodificación de imagen y audio

En esta sección se describirán los procesos de codificación y decodificación de imagen y audio, específicamente en los formatos PNG y WAV. Estos formatos han sido seleccionados porque son los utilizados en la herramienta, ya que ambos garantizan la preservación de los datos originales al ser formatos sin pérdida.

2.1.1. Imagen

La codificación de imágenes en el formato PNG (*Portable Network Graphics*) consiste en convertir datos visuales en un archivo digital comprimido sin pérdida de calidad. Este formato, desarrollado en 1996, soporta imágenes de color indexado, en escala de grises y de color de alta calidad (hasta 16 bits por píxel), además de un canal opcional alfa para corrección de color o transparencia. Según describe Renau Torres (2011), el proceso utiliza el algoritmo Deflate, que combina la compresión LZ77 y la codificación Huffman, creado por Phil Katz en 1996 para PKZIP¹ y adoptado como base de la compresión PNG. Durante la codificación, los píxeles se comprimen con una tasa alta (superior a la de GIF [*Graphics Interchange Format*]) mediante este método sin pérdidas, lo que permite almacenar la imagen eficientemente mientras se mantiene toda su información original. PNG también ofrece visualización progresiva, similar a GIF y JPEG (*Joint Photographic Experts Group*), y destaca por su robustez ante errores de transmisión, siendo ideal para internet.

¹PKZIP es un software de compresión de archivos desarrollado por PKWARE, introdujo el popular formato de compresión de archivos ZIP.

Internamente, un archivo PNG se representa como una matriz de píxeles, donde cada píxel está compuesto por valores que representan los colores en formato RGB (*Red, Green, Blue*) o, en su caso, en escala de grises. En imágenes en color, cada píxel tiene tres valores, uno por componente RGB, y si incluye transparencia, un cuarto valor correspondiente al canal alfa. Cada uno de estos valores de color generalmente se representa con 8 bits (0-255), sumando 24 bits por píxel en imágenes RGB, y 32 bits si incluye alfa.

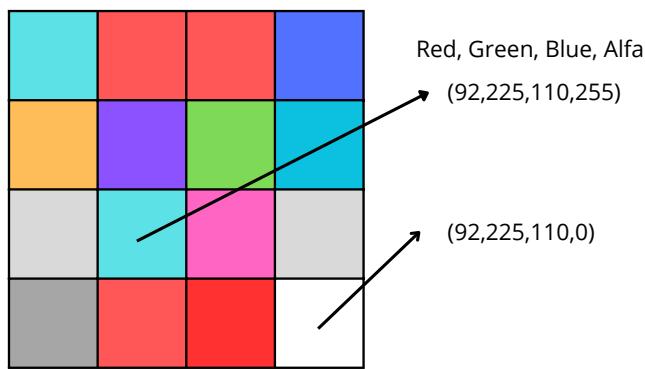


Figura 2.1: Representación de una imagen en formato PNG. Elaboración propia.

La decodificación de una imagen PNG implica revertir este proceso: el archivo comprimido se descomprime usando el algoritmo Deflate, reconstruyendo los datos de los píxeles exactamente como estaban antes de la codificación. Dado que es un formato sin pérdidas, no se descarta información, y la imagen resultante es idéntica a la original, ya sea en color, escala de grises o con canal alfa. Este formato, diseñado como estático (a diferencia de su variante animada MNG [*Multiple-image Network Graphics*]), garantiza que la visualización final preserve la calidad y las características del archivo codificado.

2.1.2. Audio

La codificación de audio en el formato WAV (*Waveform Audio File Format*) consiste en convertir una señal sonora analógica en un archivo digital sin pérdida de datos. Según Microsoft Corporation and IBM Corporation (1991), WAV es un formato que se utiliza para almacenar flujos digitales de audio y emplea principalmente LPCM como codec (*Linear Pulse Coded Modulation*, Modulación por impulsos codificados lineal, no comprimido), el cual no tiene pérdida de calidad. Este proceso digitaliza el audio mediante un muestreo de la señal a una frecuencia como 44100 Hz (calidad de CD) y cuantificando cada muestra con una profundidad de 16 bits por cada canal de audio, almacenando estas muestras sin compresión. El archivo WAV

incluye un encabezado que especifica múltiples datos de canales de audio como la frecuencia de muestreo, el número de canales (mono o estéreo) y la profundidad de bits, seguido de los datos de audio, lo que lo hace adecuado para uso profesional y para la herramienta donde se requiere máxima fidelidad.

Internamente, un archivo WAV representa el audio como una serie de muestras de amplitud tomadas en intervalos regulares (frecuencia de muestreo). Cada muestra corresponde a la amplitud de la señal en ese momento específico, y se representa como un valor binario. El formato WAV puede ser mono o estéreo, dependiendo del número de canales. En un archivo estéreo, cada muestra tiene dos valores (uno para el canal izquierdo y otro para el derecho). La frecuencia de muestreo determina cuántas muestras se toman por segundo, mientras que la profundidad de bits define la precisión de cada muestra.

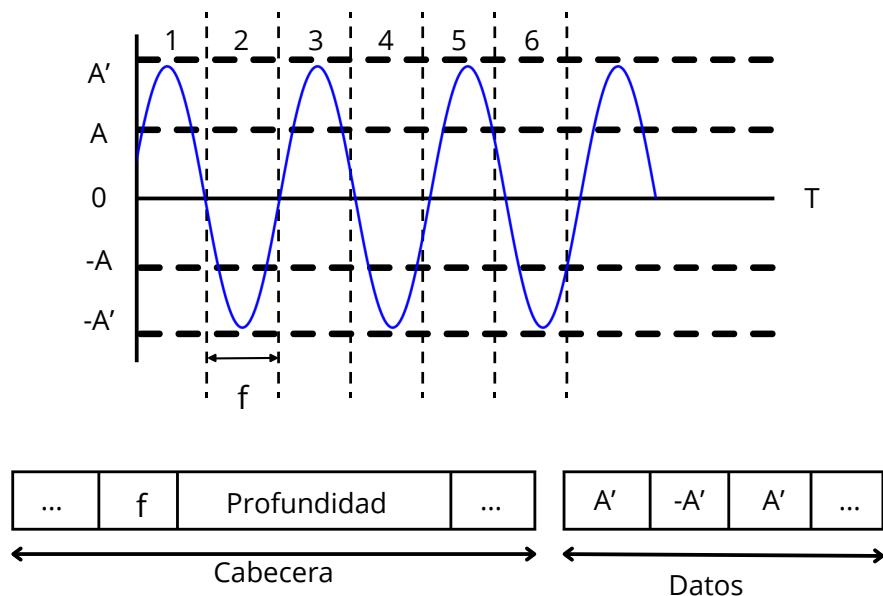


Figura 2.2: Representación de un audio estéreo en formato WAV. Elaboración propia.

La decodificación de un archivo WAV consiste en leer las muestras digitales almacenadas y convertirlas nuevamente en una señal sonora reproducible. Según Microsoft Corporation and IBM Corporation (1991), el encabezado proporciona la tasa de muestreo (ejemplo, 44100 Hz) y la cantidad de bits por muestra (ejemplo, 16 bits), permitiendo a los programas interpretar las muestras LPCM no comprimidas para reproducir el audio con máxima calidad y sin pérdida de datos, como se requiere en la herramienta.

2.2. Esteganografía

En esta sección se hablará de la esteganografía, definiéndola, mencionándose su empleo a lo largo de la historia, tipos y sus usos en archivos multimedia.

2.2.1. Orígenes

Uno de los primeros ejemplos conocidos del uso de la esteganografía, según Salas (2015), se remonta aproximadamente a 400 años antes de Cristo en la Antigua Grecia. Un personaje cogió dos tablillas cubiertas de cera, las cuales solían ser rayadas en la superficie como método de escritura, además grabó un mensaje en la madera de las tablillas antes de cubrirlas, quedando oculto bajo la cera y siendo solo posible leerse tras derretirla. En esta historia, el conjunto de las tablillas y la cera funcionaba como un archivo “inocente”, mientras que el mensaje real permanecía oculto bajo capas de cera.

Otro ejemplo histórico, estrechamente ligado a la literatura española, es el caso de *La Celestina*, escrita alrededor de 1485 y publicada en 1499. En esta obra, su autor, en un principio anónimo, ocultó su nombre en las primeras letras de los versos del preámbulo de la obra, dejando así su firma a plena vista². Esta misma técnica, según Salas (2015), fue utilizada en *El sueño de Polífilo (Hypnerotomachia Poliphili)* de Francesco Colonna, publicado en 1499. En este caso, la primera letra de cada capítulo forma el mensaje “*Poliam frater Franciscus Columna peramavit*”, una declaración de amor en latín que, traducida al español, significa “El hermano Francesco Colonna amó apasionadamente a Polia”.

Más recientemente, según Salas (2015) tras la Segunda Guerra Mundial y la aparición de la informática, la esteganografía se volvió más refinada. Los servicios de inteligencia de varios países (CIA, Mossad, MI6, entre otros) comenzaron a darse cuenta de que los terroristas la utilizaban para comunicarse de manera oculta a lo largo de la red. Los analistas de los servicios secretos detectaron que usuarios relacionados con Al Qaeda empleaban el tablón de Reddit³ para insertar anuncios aparentemente inocentes con caracteres hexadecimales y números primos, posiblemente utilizados como claves de cifrado para algoritmos como RSA. Sin embargo, no hay información confirmada sobre el propósito de la información oculta. Según un extracto de un libro publicado por *The New York Post*, en más de una ocasión, Reddit habría permitido rastrear a terroristas que utilizaban este tipo de cifrado, cuyos mensajes, al ser decodificados, a veces indicaban la planificación inminente de un atentado, según Russen (2015).

²Estos versos pueden encontrarse en <https://www.badosa.com/bin/obra.pl?id=n266-02>.

³<https://www.reddit.com/> Reddit es una plataforma de discusión y agregación de contenido organizada en comunidades temáticas llamadas *subreddits*.

2.2.2. Descripción

Como define Carracedo Gallardo (2004), “la esteganografía es la técnica y el arte de ocultar una información dentro de otra”. Según se menciona en su libro, esta disciplina ofrece un método simple para ocultar información mediante la aplicación de algoritmos que generan alteraciones imperceptibles en archivos que parecen inocuos, como textos, imágenes o archivos de audio. Como resultado, se obtiene un archivo que alberga datos ocultos pero conserva la misma apariencia que el original, denominado comúnmente como archivo portador o inocente. Para recuperar el mensaje oculto, se aplica un proceso inverso al utilizado en la ocultación.

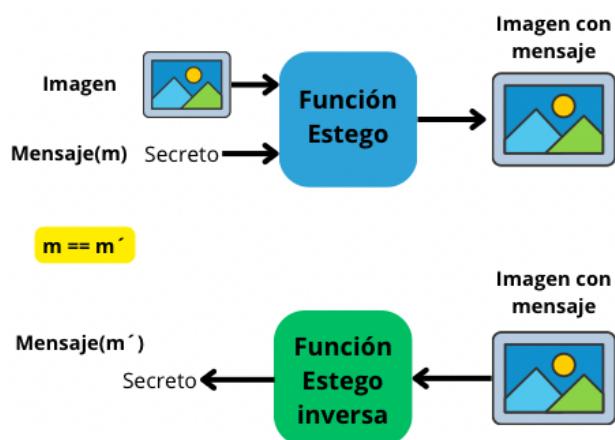


Figura 2.3: Diagrama de funcionamiento de la esteganografía. Elaboración propia.

2.2.3. Tipos de esteganografía

La esteganografía, como técnica para ocultar información, ha evolucionado a lo largo de los años, dando lugar a diversas clasificaciones y aplicaciones, tal como señala Iglesias (2015). A continuación, se presentan sus principales tipos y ámbitos de uso:

- **Esteganografía pura:** Las técnicas de esteganografía pura se caracterizan por no emplear una estego-clave, la cual es una clave especial utilizada para determinar cómo y dónde ocultar o recuperar información en el archivo portador. Al no usar esta clave, el mensaje oculto se inserta directamente en el medio elegido sin necesidad de elementos adicionales. De este modo, se asume que un receptor no deseado será incapaz de reconocer que existe información oculta en un mensaje aparentemente normal, basándose así en un modelo de

seguridad por oscuridad⁴.

- **Esteganografía de clave secreta:** Por otro lado, en la esteganografía de clave secreta, la estego-función depende del uso de una clave conocida únicamente por el emisor y el receptor. Esta clave es fundamental, ya que determina la forma en la que el mensaje se camufla y cómo recuperarlo posteriormente. Un ejemplo simple sería utilizar una estego-clave para determinar la ubicación exacta dentro del archivo portador donde se comenzará a insertar el mensaje secreto.

Iglesias (2015) señala que estas dos categorías abarcan una amplia variedad de aplicaciones, que incluyen su uso en textos, sistemas operativos, ficheros, formatos de archivo, hardware, tecnologías web, protocolos de comunicación y contenido multimedia en las cuales nos centraremos.

2.2.4. Esteganografía en contenido multimedia

Ocultar información dentro de archivos digitales, como imágenes o audios, aprovechando sus características para realizar modificaciones imperceptibles es en lo que consiste la esteganografía en contenido multimedia. Esta técnica permite almacenar datos adicionales sin alterar de forma significativa la calidad o la apariencia original del medio. Además, según Rodríguez Roca (2009), las técnicas de esteganografía pueden clasificarse en dos grandes grupos:

- **Las basadas en el dominio de la imagen:** Estas técnicas ocultan la información modificando directamente los valores de los píxeles de la imagen. Generalmente son fáciles de implementar, aunque pueden ser vulnerables ante análisis forenses avanzados. Una de las técnicas más frecuentes es la sustitución de bits menos significativos (LSB, por sus siglas en inglés), que consiste en insertar información modificando los bits menos significativos de cada píxel. Este método también puede aplicarse en archivos de audio, modificando ligeramente sus muestras digitales para ocultar datos sin afectar perceptiblemente la calidad auditiva.

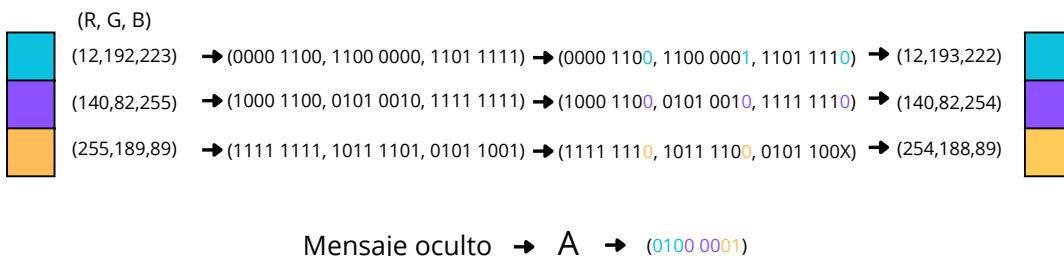


Figura 2.4: Ejemplo de LSB en una imagen PNG sin canal alfa. Elaboración propia.

⁴Es un concepto que significa proteger información simplemente manteniendo en secreto el método utilizado para ocultarla o protegerla.

- **Las basadas en el dominio de la transformación:** Estas técnicas utilizan algoritmos que modifican o transforman matemáticamente la imagen antes de insertar la información oculta, como se puede ver en la figura 2.5. Los mensajes suelen insertarse en las áreas más relevantes o significativas del archivo portador, aumentando así la resistencia ante modificaciones o análisis forenses. Además, pueden alterar propiedades específicas, como la luminosidad en imágenes, o ciertas características auditivas en el caso del audio. Por ejemplo, en imágenes se suele utilizar la transformada discreta del coseno (DCT), la cual descompone la imagen en coeficientes de frecuencias. Luego, se modifica ligeramente alguno de estos coeficientes (normalmente aquellos menos perceptibles al ojo humano) para insertar el mensaje oculto. En audio, técnicas como el encaramiento auditivo aprovechan las limitaciones auditivas del oído humano para ocultar información en frecuencias o rangos que son poco perceptibles, o la técnica del eco, que consiste en insertar datos mediante la adición de ecos muy sutiles, imperceptibles para el oyente promedio. Estos enfoques, aunque más complejos, permiten una ocultación más resistente ante análisis forenses.

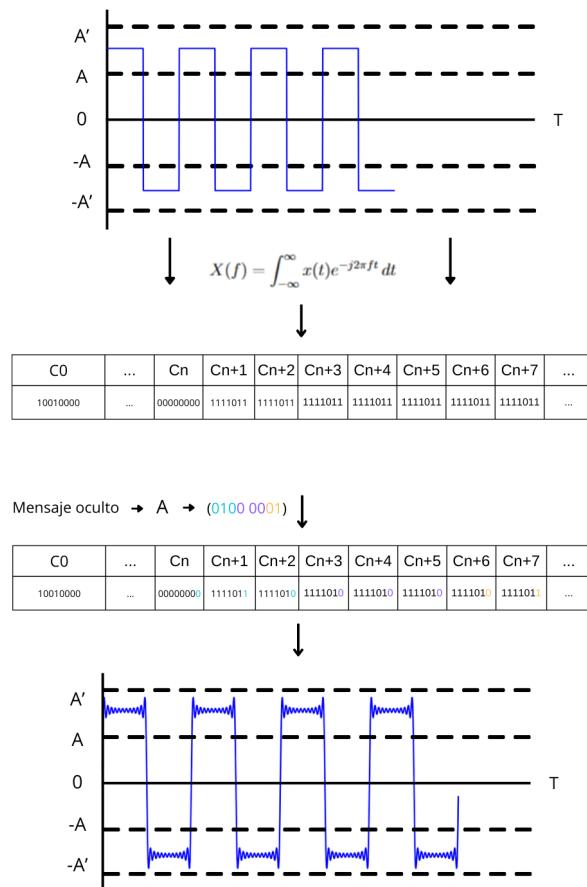


Figura 2.5: LSB usando dominio de la transformada con la transformada de Fourier. Elaboración propia.

El análisis se centrará específicamente en las técnicas del dominio de la imagen, debido a que estos métodos suelen ser más sencillos de implementar y requieren menor complejidad computacional. Además, mediante las técnicas de este dominio se pueden proporcionar ejemplos claros y prácticos que facilitan su estudio, análisis y comprensión.

2.3. Criptografía

En esta sección se hablará de la criptografía pasando por sus distintos tipos respecto a las claves. Esto sin profundizar en sus fundamentos matemáticos ya que se alejan del fin de este documento.

Prácticamente la totalidad de la información de esta sección ha sido extraída de Carracedo Gallardo (2004).

2.3.1. Criptografía de clave secreta

En este modelo de criptosistema, la clave con la cual se realiza el cifrado es la misma que se utiliza para descifrar, lo que produce que todos los integrantes de la comunicación, emisor y receptores, deban compartir la clave. Debido a esto, la criptografía de clave secreta es también llamada criptografía simétrica.

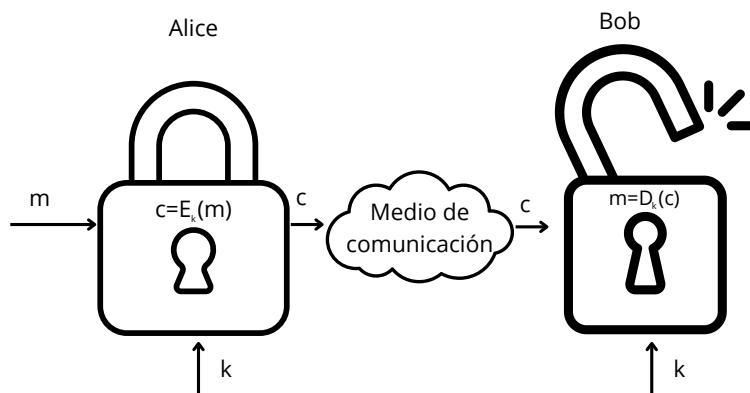


Figura 2.6: Diagrama criptografía de clave secreta. Elaboración propia.

Como se puede apreciar en la figura 2.6, denominamos “ m ” al mensaje en claro, “ c ” al mensaje cifrado y “ k ” a la clave secreta. El proceso de cifrado se define mediante la función:

$$c = E_k(m)$$

donde E_k es el algoritmo de cifrado, que a partir del mensaje en claro y una clave secreta, genera el mensaje cifrado. Ambos mensajes pueden representarse en diferentes formatos, tales como secuencias de bits, números, cadenas de texto o cualquier otro tipo de dato compatible con las operaciones internas de la función de cifrado. De manera análoga, el proceso de descifrado se describe mediante la función:

$$m = D_k(c)$$

en la que D_k representa el algoritmo de descifrado. Este recibe como entrada el mensaje cifrado y, utilizando la misma clave secreta “ k ”, recupera el mensaje original. Considerando los tipos de datos, el método de descifrado funciona de manera inversa al cifrado, intercambiando los tipos de entrada y salida.

Uno de los ejemplos más antiguos de este tipo de cifrado es el cifrado “César”, el cual debe su nombre a que fue uno de los principales métodos de cifrado del imperio romano. Se trata de un algoritmo de cifrado por sustitución, como se puede apreciar en la figura 2.7, donde cada carácter es reemplazado por otro. Su algoritmo de cifrado es descrito como:

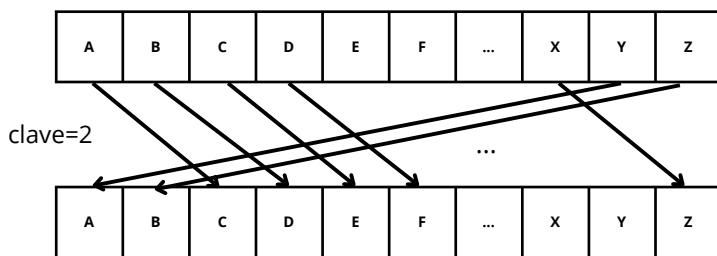
$$c[i] = (m[i] + k) \pmod{N}$$

siendo $m[i]$ el carácter en la posición i del mensaje, $c[i]$ el resultado en el mensaje cifrado en la misma posición y N el tamaño del alfabeto (Σ). Su función de descifrado es:

$$m[i] = (c[i] - k) \pmod{N}$$

cumpliéndose que todos los componentes de los mensajes en claro y cifrados pertenezcan al alfabeto :

$$\forall r \in m, r \in \Sigma \quad \text{y} \quad \forall g \in c, g \in \Sigma.$$



ZAPATO	\longrightarrow	$P=15$	\longrightarrow	$Z=26$	$A=0$	$T=20$	$O=14$	$B=1$	$C=2$	$R=16$	$W=23$	$Q=16$	\longrightarrow	BCRCWQ
--------	-------------------	--------	-------------------	--------	-------	--------	--------	-------	-------	--------	--------	--------	-------------------	--------

Figura 2.7: Ejemplo cifrado César. Elaboración propia.

Volviendo a tiempos más modernos está el cifrador AES (*Advanced Encryption Standard*)⁵, para el cual se seleccionó el algoritmo de Rijndael, de los criptógrafos belgas Joan Daemen y Vincent Rijmen, en el año 2001 tras el concurso realizado por el NIST (*National Institute of Standards and Technology*). El concurso fue lanzado en septiembre de 1997, con el fin de recibir nuevas propuestas de algoritmos de protección de información tanto civil como militar.

El algoritmo de Rijndael se lleva a cabo mediante el siguiente proceso:

- **Primera ronda:**

- Adición de la clave de ciclos (*AddRoundKey*).

- **N rondas básicas:** figura 2.8, cada una compuesta por:

- Sustitución de bytes (*ByteSub*).
- Desplazamiento de filas (*ShiftRow*).
- Mezcla de columnas (*MixColumns*).
- Adición de la clave de ciclos (*AddRoundKey*).

- **Ronda final:**

- Sustitución de bytes.
- Desplazamiento de filas.
- Adición de la clave de ciclos.

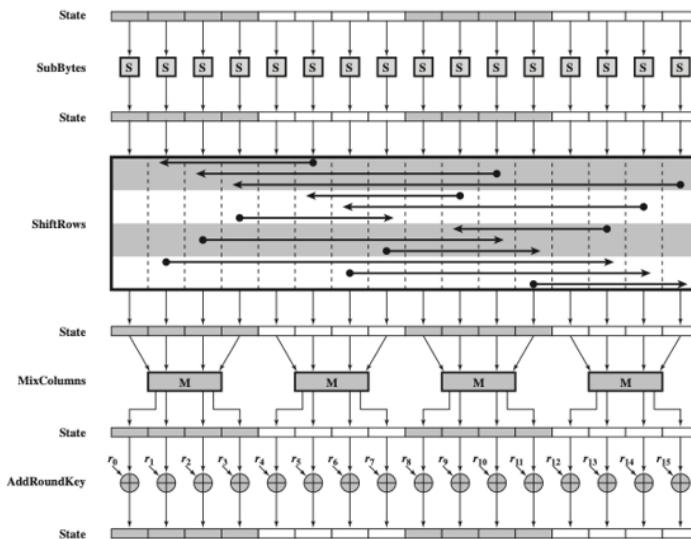


Figura 2.8: Ciclo básico del algoritmo de Rijndael, Stallings (2022).

A continuación se hablará de cada una de las funciones mencionadas en las etapas del algoritmo de Rijndael:

⁵National Institute of Standards and Technology (NIST) (2001)

1. Sustitución de bytes:

Se trata de una sustitución no lineal que se realiza sobre todos los bytes; esta sustitución se conforma de dos etapas:

- a) Cada byte se sustituye por su inverso multiplicativo (considerándose cada uno un elemento del conjunto $GF(2^8)$ ⁶) y el 00 por sí mismo.
- b) A la salida de la sustitución anterior se le aplica la siguiente operación:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

2. Desplazamiento de filas:

Se desplazan cíclicamente a la izquierda las filas; el número de veces que se desplaza cada fila depende de su posición⁷.

$$s'_{r,c} = s_{r,(c+r) \bmod 4}, \quad \text{para } 0 \leq r < 4, 0 \leq c < 4$$

siendo $S_{r,c}$ el byte en la fila r y columna c .

3. Mezcla de columnas:

Cada byte constituyente de cada columna de la salida de la función anterior es representado como polinomio sobre $GF(2^8)$ y multiplicado por un polinomio constante $c(x)$ módulo $x^4 + 1$.

$$c(x) = 03_{16}x^3 + 01_{16}x^2 + 01_{16}x + 02_{16}.$$

4. Adición de la clave de ciclos:

Se aplica la operación XOR a la salida de la función anterior y a la subclave correspondiente.

Estas subclaves son expansiones del mismo tamaño que la original. Estas expansiones consisten en añadir 4 bytes obtenidos al aplicar XOR entre la clave y partes predeterminadas de la expansión anterior. La primera vez, se realiza a la clave la sustitución de bytes, explicada anteriormente en el punto 1 de esta enumeración, y se aplica XOR con una constante definida en función del número del ciclo.

⁶[Http://www.math.huji.ac.il/~jgutierrez/GaloisFinitos.pdf](http://www.math.huji.ac.il/~jgutierrez/GaloisFinitos.pdf).

⁷(National Institute of Standards and Technology (NIST), 2001, p. 17) .

2.3.2. Criptografía de clave pública

A diferencia de la criptografía de clave secreta, en la criptografía de clave pública no se cifra y descifra con la misma clave, sino que se utiliza una clave para cifrar y otra para descifrar, por eso también es llamada criptografía asimétrica. Cuando hablamos de clave en este tipo de criptosistema nos referimos a un par de claves, la clave secreta KS , la cual se debe mantener protegida, y la clave pública KP , la cual puede viajar por la red sin protección ninguna. Este par de claves debe cumplir que la clave pública sea fácilmente derivable de la clave privada pero que la privada sea computacionalmente impracticable obtenerla a partir de la clave pública.

Como se puede ver en la figura 2.9, si se desea mandar un mensaje cifrado a un usuario o sistema **B**, se adquiere su clave pública, la cual puede viajar sin protección al no poner en riesgo la confidencialidad de los mensajes posteriormente cifrados por ella y se cifra el mensaje que se desea que únicamente pueda leer **B**. Al recibir el mensaje cifrado, **B** lo descifra con su clave secreta.

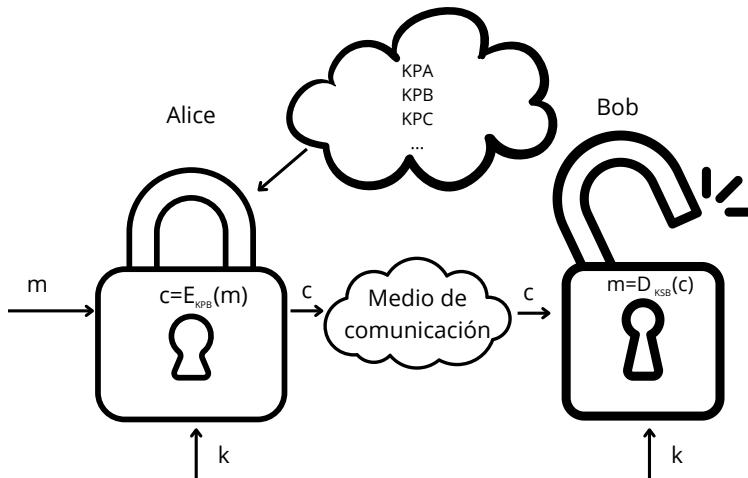


Figura 2.9: Diagrama criptografía de clave pública. Elaboración propia.

Además de la capacidad de cifrar mensajes, la criptografía de clave pública permite el firmado de mensajes, mediante el cifrado con la clave secreta y descifrado con la clave pública de un *hash* del mensaje; esto proporciona integridad y autenticidad de los datos. Esto se debe a que, si bien el mensaje puede ser visto por cualquiera, solo se puede hacer utilizando la clave pública de la entidad que ha cifrado el mensaje con su clave secreta, figura 2.10.

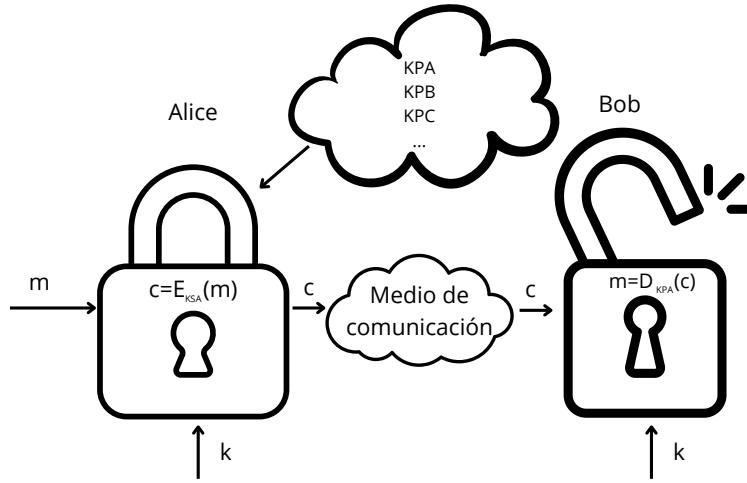


Figura 2.10: Firmado con clave pública. Elaboración propia.

Uno de los algoritmos más conocidos de este tipo de criptosistemas es el RSA⁸, cuyo nombre está compuesto de las iniciales de sus creadores: Rivest, Shamir y Adleman.

El RSA afronta el cifrado de manera diferente al AES, ya que en este caso el mensaje se divide en bloques de forma que el valor de los bloques sean números enteros que cumplan:

$$0 < \text{valor de bloque} < n - 1 \text{ y número de bits de cada bloque } \leq \log_2(n).$$

1. Generación de claves:

- a) Se eligen dos números primos p y q ⁹.
- b) Se calcula:

$$n = p * q.$$

- c) Se elige un e tal que:

$$\text{mcd}(e, \varphi(n)) = 1 \pmod{n}$$

siendo $\varphi(n)$ la función de Euler,

$$\varphi(n) = \#\{k \in \mathbb{Z} \mid 1 \leq k \leq n, \text{ mcd}(k, n) = 1\}$$

la cantidad de números enteros menores de n que son coprimos con n , además sabemos que:

$$p, q \text{ primos} \Rightarrow \varphi(p * q) = (p - 1)(q - 1).$$

⁸National Institute of Standards and Technology (NIST) (2019).

⁹Para asegurar la seguridad del algoritmo hoy en día es necesario que tengan un tamaño superior a 100 dígitos en base 10 y ambos números sean de un tamaño parecido.

d) Se calcula d de forma que:

$$d = e^{-1} \pmod{\varphi(n)}$$

siendo $KP = \{n, e\}$ y $KS = \{n, d\}$.

2. Cifrado y descifrado.

- Cifrado:

$$c = m^e \pmod{n}.$$

- Descifrado

$$m = c^d \pmod{n}.$$

La seguridad del algoritmo se basa en que el cálculo de $\varphi(n)$ sin conocer p y q es computacionalmente inviable, debido a que no conocemos ningún algoritmo con una eficiencia polinómica o menor.

2.4. Transmisión de datos mediante SSTV

En esta sección se hablará de qué es SSTV y de su funcionamiento básico, ya que para el fin de la herramienta no es necesario profundizar en detalles técnicos ni en los detalles particulares de cada uno de los 27 modos de transmisión.

Según International Amateur Radio Union (IARU) Region 1 (2021), SSTV (*Slow Scan Television* o en español, Televisión de barrido lento) es un modo de transmisión capaz de recibir y transmitir imágenes estáticas vía radio. Este modo de transmisión utiliza un ancho de banda máximo de 2,7 kHz aproximadamente, siendo representado el negro por un tono a 1500 Hz y 2300 Hz para el blanco. Además se une un pulso sincronizado (sync pulse) en 1200 Hz de manera que al ser bastante inferior al negro es invisible. Estos pulsos se envían al final de cada línea, durando 5 ms, y al final de cada imagen con una duración de 30 ms. Siendo su función mantener sincronizado el receptor con el transmisor, garantizando que la imagen se reconstruya correctamente, figura 2.11.

SSTV utiliza modulación de frecuencia, asociando una frecuencia de audio distinta a cada distinto valor de intensidad ubicable en los diferentes puntos de la imagen. Los colores son conseguidos enviando la intensidad de cada componente RGB de manera separada y consecutiva.

En la tabla 2.1 se pueden observar las frecuencias más usadas por radioaficionados para transmisión por SSTV, significando LSB (solo en esta sección), *Lower Side Band* y USB, *Upper Side Band*.

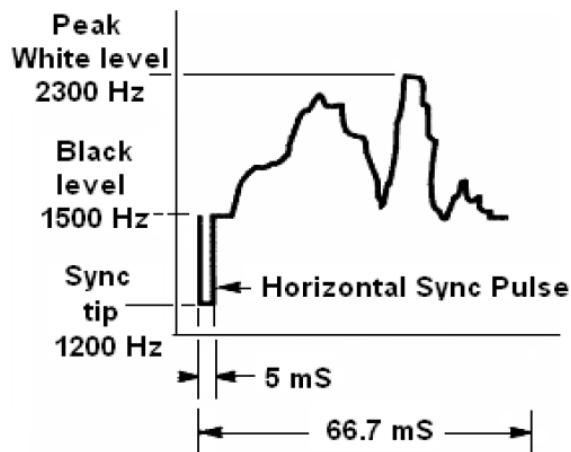


Figura 2.11: Onda de audio usando SSTV para imágenes en blanco y negro, International Amateur Radio Union (IARU) Region 1 (2021).

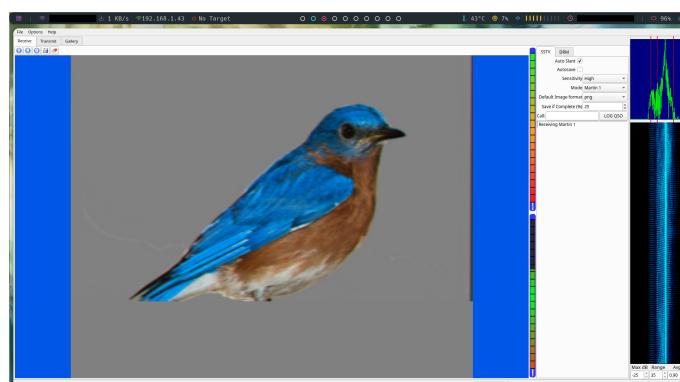


Figura 2.12: Decodificación de un audio en SSTV usando QSSTV. Elaboración propia.

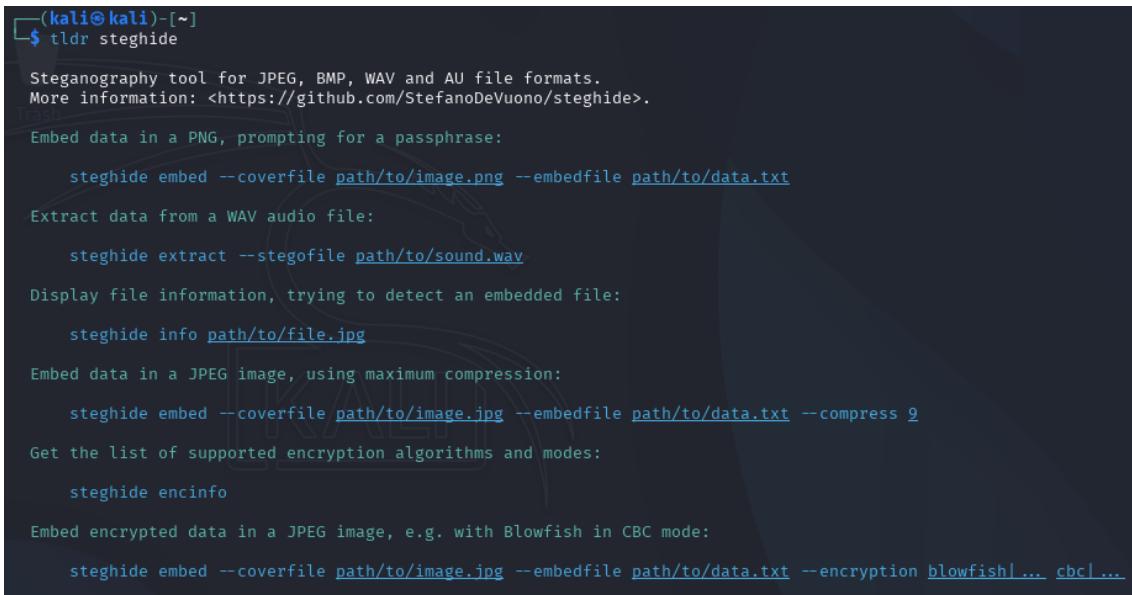
Banda	Frecuencia	Modo
80 m	(3735 ± 5) kHz	LSB
40 m	7035 kHz a 7050 kHz	LSB
20 m	14 220 kHz a 14 235 kHz	USB
15 m	21 330 kHz a 21 346 kHz	USB
10 m	28 670 kHz a 28 690 kHz	USB

Tabla 2.1: Frecuencias típicas de uso para SSTV en bandas de radioaficionado, International Amateur Radio Union (IARU) Region 1 (2021).

2.5. Comparación con otras herramientas

En esta sección se comparará la herramienta desarrollada con otras herramientas muy usadas del mundo de la esteganografía, como *Steghide*¹⁰ y *SilentEye*¹¹.

Steghide es una de las herramientas más conocidas en el mundo de la esteganografía, esta viene instalada por defecto en las distribuciones de *Kali Linux*. *Steghide* permite ocultar datos en archivos de imagen en los formatos JPEG, BMP (*Bits Maps Protocole*) y también de audio en formatos WAV y AU (*Audio file format*). La característica más relevante de *Steghide* es que oculta información sin producir cambios visibles en los tonos de color o en las propiedades de muestreo del archivo original, logrando así que sea resistente a pruebas estadísticas de primer orden, este tipo de pruebas evalúa la frecuencia de aparición de cada símbolo (por ejemplo, unos y ceros en una secuencia binaria) y la compara con la distribución teórica esperada mediante tests como el chi-cuadrado o medidas de entropía para detectar desviaciones. Además, dispone de funcionalidades como la compresión de datos antes de ocultarlos, la protección del contenido con un cifrado robusto e incorpora también un mecanismo para comprobar posteriormente que la información recuperada no ha sido modificada. A continuación, se pueden ver en la figura 2.13 unos ejemplos de uso de *Steghide*.



```
(kali㉿kali)-[~]
$ tldr steghide

Steganography tool for JPEG, BMP, WAV and AU file formats.
More information: <https://github.com/StefanoDeVuono/steghide>.

Embed data in a PNG, prompting for a passphrase:
steghide embed --coverfile path/to/image.png --embedfile path/to/data.txt

Extract data from a WAV audio file:
steghide extract --stegofile path/to/sound.wav

Display file information, trying to detect an embedded file:
steghide info path/to/file.jpg

Embed data in a JPEG image, using maximum compression:
steghide embed --coverfile path/to/image.jpg --embedfile path/to/data.txt --compress 9

Get the list of supported encryption algorithms and modes:
steghide encinfo

Embed encrypted data in a JPEG image, e.g. with Blowfish in CBC mode:
steghide embed --coverfile path/to/image.jpg --embedfile path/to/data.txt --encryption blowfish|..._cbc|...
```

Figura 2.13: Ejemplos de uso proporcionado por tldr de *Steghide*.

<Https://tldr.sh/>, tldr es una herramienta que muestra páginas de ayuda que han sido creadas por la comunidad para simplificar las páginas del manual mediante ejemplos prácticos.

Steghide es muy buena ocultando información en imágenes y audio usando LSB,

¹⁰<Https://steghide.sourceforge.net/> .

¹¹<Https://github.com/achorein/silenteye> .

pero tiene limitaciones claras. PITEA no solo hace ocultaciones mediante LSB en PNG y en WAV, sino que también añade otras opciones, como generar una imagen directamente con la información cifrada y codificada en base64, algo que *Steghide* no puede hacer. Otra ventaja de PITEA es la posibilidad de modular una señal de audio para SSTV a partir de una imagen con texto incrustado en ella. Esto permite que la imagen se pueda transmitir por radiofrecuencia y el mensaje oculto pueda ser recuperado posteriormente. Como veremos más adelante, si el secreto estuviera oculto en la imagen mediante LSB, este no podría ser recuperado.

SilentEye, también oculta mediante LSB en archivos en formato JPEG, BMP y WAV, dispone de cifrado con AES128 o AES256, dispone de compresión zlib¹² para reducir el tamaño del mensaje antes de incrustarlo y de un sistema de *plugins* que permite la integración de manera sencilla de nuevos algoritmos de esteganografía o procesos de criptografía. En las figuras 2.14 y 2.15 se puede observar la pantalla de inicio de *SilentEye* y las opciones que son desplegadas al seleccionar la opción de encode.

SilentEye, tampoco implementa la generación de una imagen directamente con el texto cifrado y codificado en base64, ni la posibilidad de transformar la imagen generada en una señal SSTV. Además, debido a que PITEA está desarrollada teniendo en cuenta los patrones de diseño GoF (*Gang of Four*) es sencillo añadir nuevos tipos de cifrado o métodos de ocultación si se tienen conocimientos básicos de POO (Programación Orientada a Objetos).



Figura 2.14: Pantalla de inicio de *SilentEye*. Captura de pantalla.

¹²zlib es una biblioteca de compresión sin pérdida basada en el algoritmo DEFLATE.

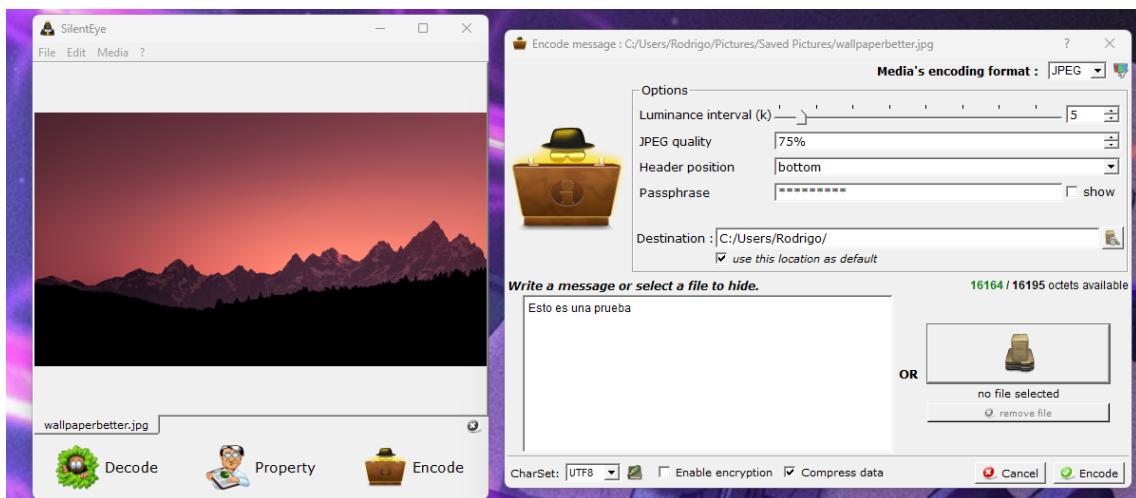


Figura 2.15: Menú *encode* de *SilentEye*. Captura de pantalla.

Capítulo 3

Arquitectura del Sistema

3.1. Diagramas de clases

Para facilitar la comprensión de la herramienta, se han realizado unos diagramas de clases que describen la arquitectura del sistema.

La interacción con el usuario se inicia en el *launch*, donde se instancian tanto el *Invoker* como el *MenuPrinter*, quienes son respectivamente los responsables de gestionar los comandos e imprimir por consola las opciones principales del programa. Los comandos son registrados y ejecutados por el *Invoker*, estos extienden de una clase abstracta denominada *Command*. Gracias a esta estructura, la creación de comandos es más sencilla en caso de que se desee ampliar la funcionalidad del sistema. Posteriormente, los comandos son ejecutados creando la instrucción correspondiente que será enviada al script de ejecución mediante *utils*, esta instrucción se genera según las opciones indicadas al comando, figura 3.1.

A continuación, figura 3.2, el método correspondiente del *main* se invoca a través del script con todas las opciones seleccionadas como argumentos. Seguidamente, las instancias concretas de cifrado y ocultación se crean mediante los *Factories* y cada instancia se llama para que ejecute su función correspondiente.

En la figura 3.3 se puede observar la clase abstracta *Cifrador* que se extiende por 2 cifradores concretos, que se instancian mediante el *Factory* correspondiente en función de las opciones que se han seleccionado de cifrado y descifrado.

Se puede elegir entre dos cifradores, *none*, el cual no cifra, y *AES*, el cual cifra los datos utilizando el algoritmo AES en modo CBC. Antes de que se cifre el mensaje, se verifica que los datos tengan un tamaño múltiplo del tamaño del bloque y se crea la clave transformando la contraseña en una clave de 16 bytes utilizando SHA256¹ y truncando el *hash* al tamaño necesario. El proceso de descifrado consiste en transformar la contraseña a clave como en el cifrado y con ella se descifraría el mensaje.

¹National Institute of Standards and Technology (NIST) (2015).

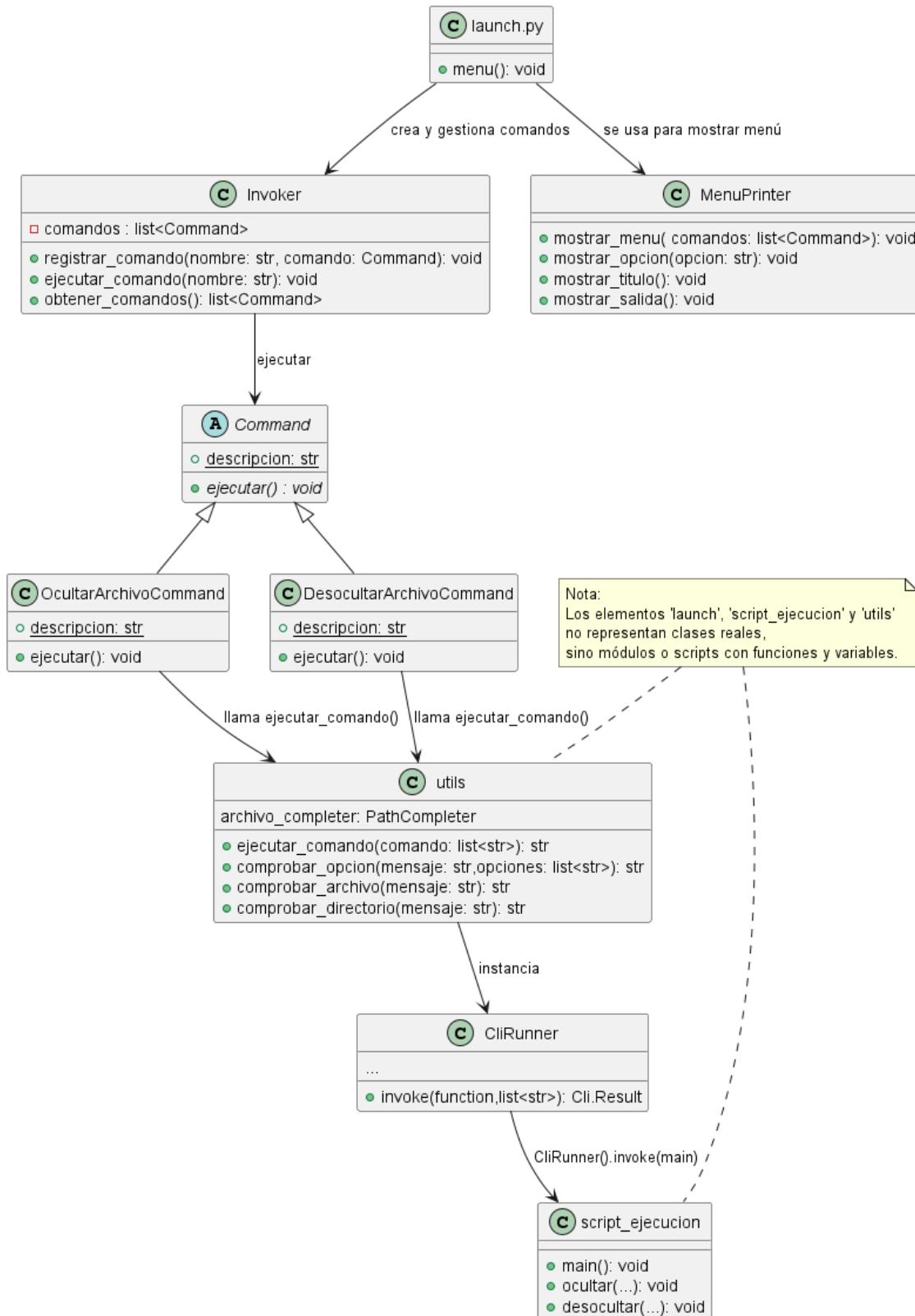


Figura 3.1: Diagrama de clases: CLI. Elaboración propia.

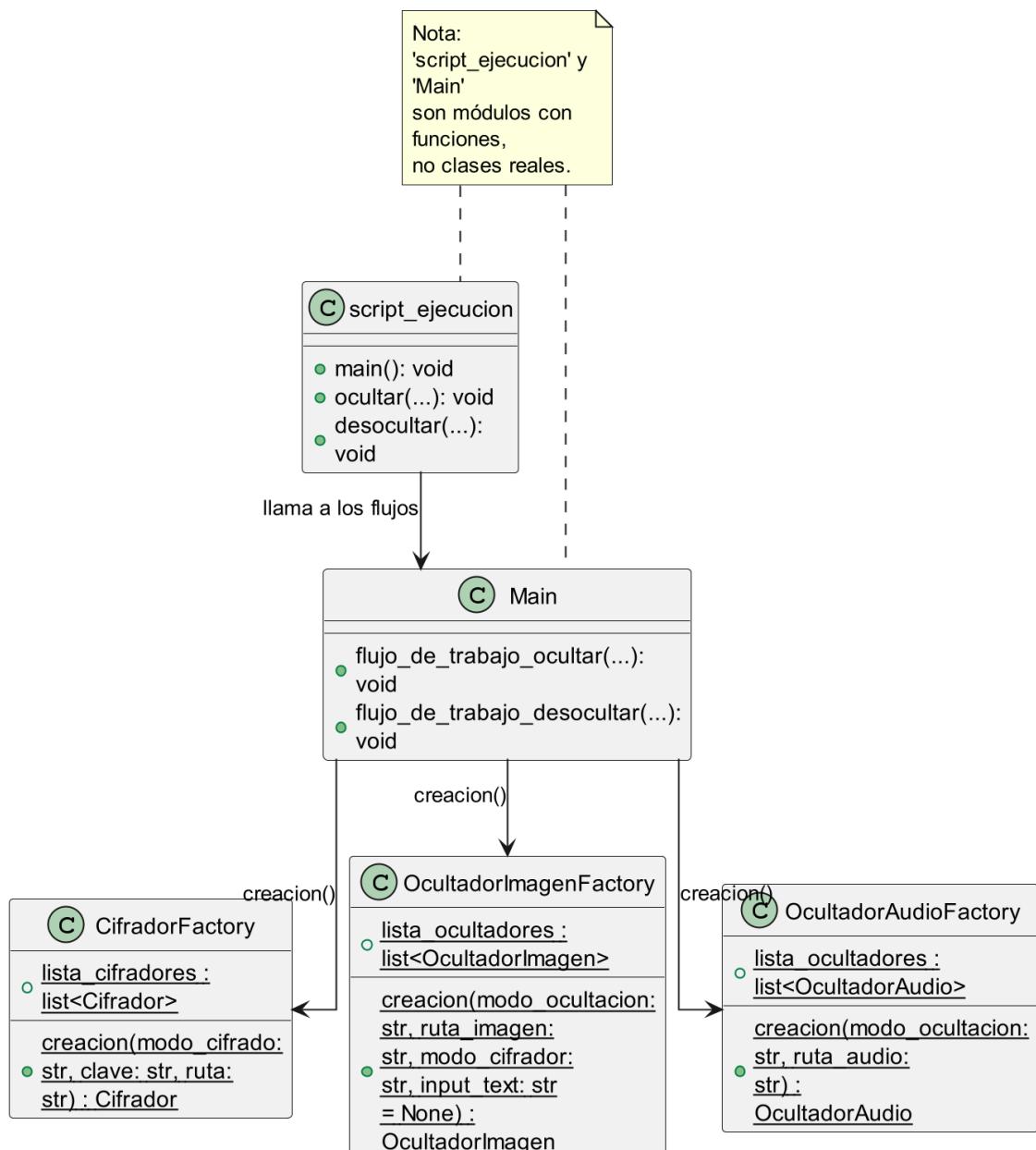


Figura 3.2: Diagrama de clases: Main. Elaboración propia.

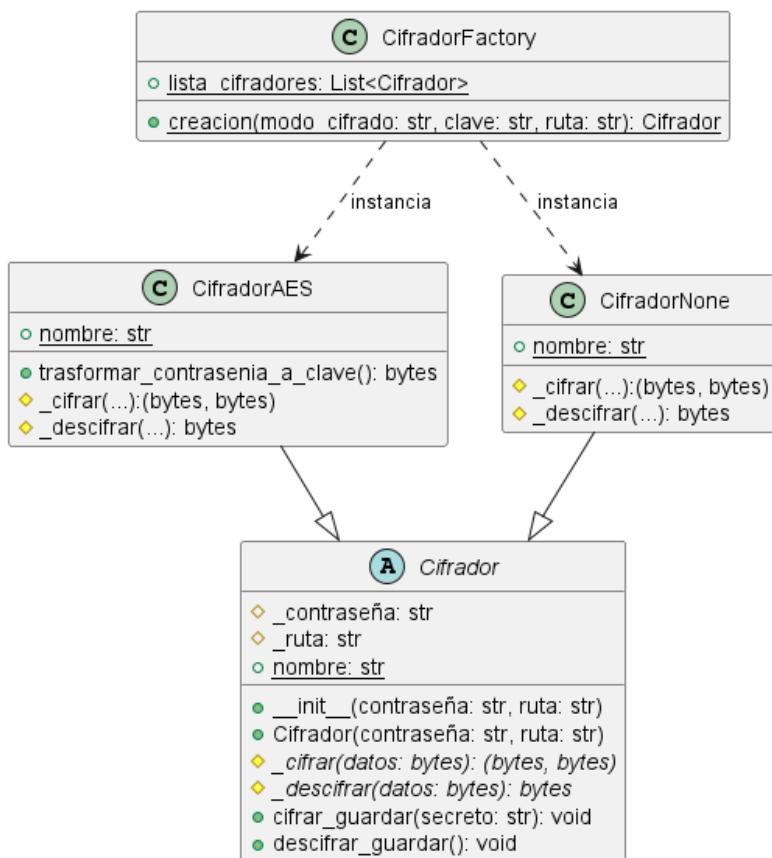


Figura 3.3: Diagrama de clases: Cifradores. Elaboración propia.

Los ocultadores tanto en imágenes como en audio siguen el mismo diseño que los cifradores, como se puede comprobar en las figuras 3.4 y 3.5. Este diseño está orientado a la escalabilidad, facilitando la incorporación de nuevas funcionalidades.

En los ocultadores en imagen se dispone de ocultación mediante LSB o *Text*. Con la ocultación LSB, primero se calcula el tamaño de los datos a ocultar en binario para indicarlo en una cabecera de 32 bits² que serán los 32 primeros datos que se ocultarán en la imagen. Después, se comprueba que la imagen tiene suficiente espacio para ocultar el mensaje completo, el cual consiste en los 32 bits de la cabecera más el número de bits de datos a ocultar, y se van recorriendo los píxeles de la imagen añadiendo en el bit menos significativo de cada píxel un bit del mensaje hasta que esté completamente oculto. El tamaño mínimo de la imagen tiene que ser:

$$\text{MB mínimos de imagen} = \frac{\left(\frac{T+M}{3}\right) \times 3}{1024^2} = \frac{T+M}{1024^2}$$

donde T = tamaño de la cabecera en bits y M = tamaño del secreto en bits.

En el proceso de desocultación, primero se leen los 32 primeros bits menos significativos para obtener el tamaño del mensaje que se va a leer y, posteriormente, se lee el mensaje completo.

Con la ocultación *Text*, si los datos vienen cifrados se codifican en base64 antes de que se escriban, si no están cifrados se escriben tal cual. Esto se debe a que al cifrar la información, los caracteres imprimibles pueden haberse convertido en no imprimibles. Primero, se carga el archivo “configuracion.toml”, archivo de configuración que, entre otras cosas, contiene la información del tamaño de la fuente, la anchura máxima y la ruta de la fuente que se va a usar. Posteriormente, se calcula la anchura máxima de las líneas y se procede a insertar el mensaje en una lista de listas de caracteres, siendo cada lista una línea del texto que se cargará en la imagen. Por último, se crea la imagen, con un tamaño fijo si se va a usar SSTV o con el tamaño necesario si no, y se dibuja el texto en ella.

Para desocultar se utilizan dos motores OCR (*Optical Character Recognition*) que extraen el texto de la imagen. Se han usado dos debido a que uno está diseñado para reconocer palabras completas, en caso de no cifrado de datos, y el otro para reconocer caracteres sueltos, caso de cifrado y codificación en base64.

En los ocultadores en audio se dispone de ocultación mediante LSB o SSTV. Con la ocultación LSB, el procedimiento sería el mismo que la ocultación en la imagen LSB, pero esta vez se ocultará en los *frames* del audio y no en píxeles. El descifrado sería también equivalente. El tamaño mínimo del audio tiene que ser:

²Se opta por usar 32 bits, que pueden representar hasta archivos de 512MB aproximadamente, en lugar de valores ligeramente menores pero más ajustados a los posibles tamaños de los archivos que se manejan, porque la diferencia en capacidad es marginal y, de este modo, se evitan posibles problemas al manejar imágenes y, sobre todo, audios de gran tamaño.

$$\text{MB mínimos de audio} = \frac{T + I + E}{1024^2}$$

donde T = tamaño de la cabecera en bits,

I = tamaño de la imagen en bits,

E = encabezado de WAV en bits.

Con la ocultación SSTV, se lee y se redimensiona la imagen a ocultar con *Pillow* y se modula la imagen en una señal SSTV usando el modo SSTV seleccionado, para los cuales es necesario que la imagen transmitida tenga un tamaño fijo. En cambio, para desocultar, se abre QSSTV³ y, dependiendo de si se ha seleccionado el modo streaming o no, se muestra por terminal la ruta del archivo que debe seleccionar para decodificar en QSSTV y dónde se debe guardar para seguir con el correcto funcionamiento de la herramienta.

Las tres *Factories* mencionadas se han implementado como subclases que extienden de una *factory* abstracta denominada *AbstractFactory*, tal como se ve en la figura 3.6. Como hemos visto antes, esto también es una facilidad en caso de que se quieran incorporar nuevos medios en los que ocultar información.

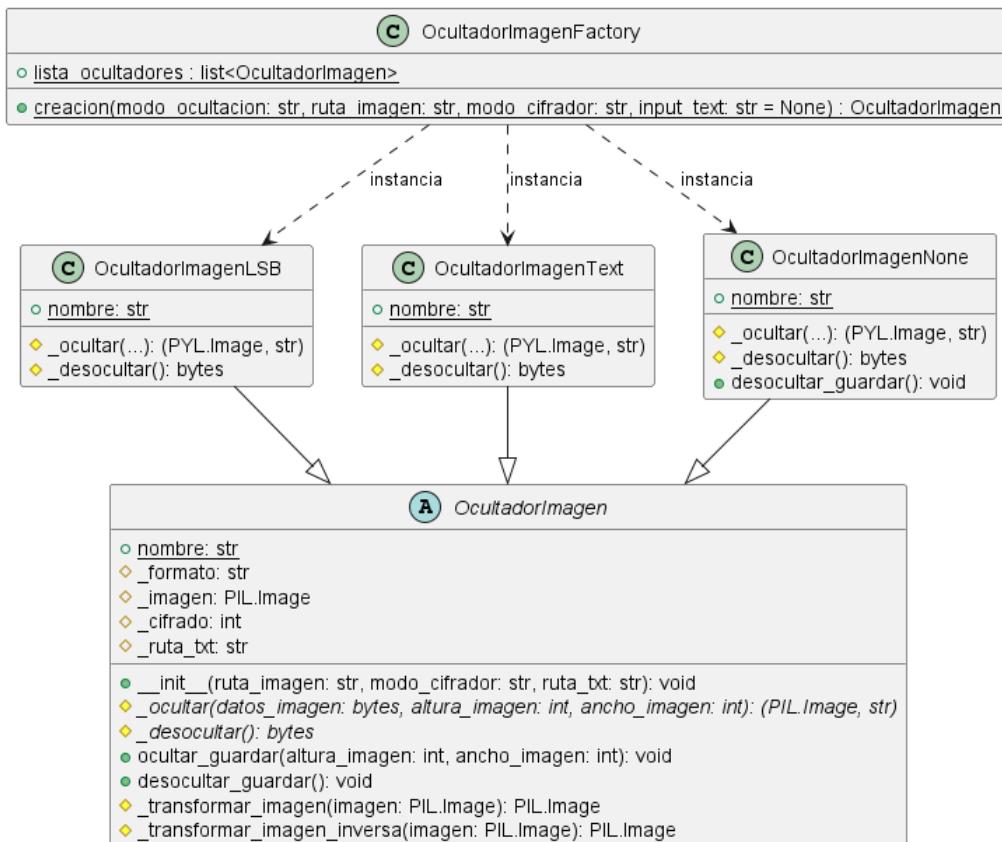


Figura 3.4: Diagrama de clases: Ocultadores en imagen. Elaboración propia.

³<https://www.cqsstv.com/>

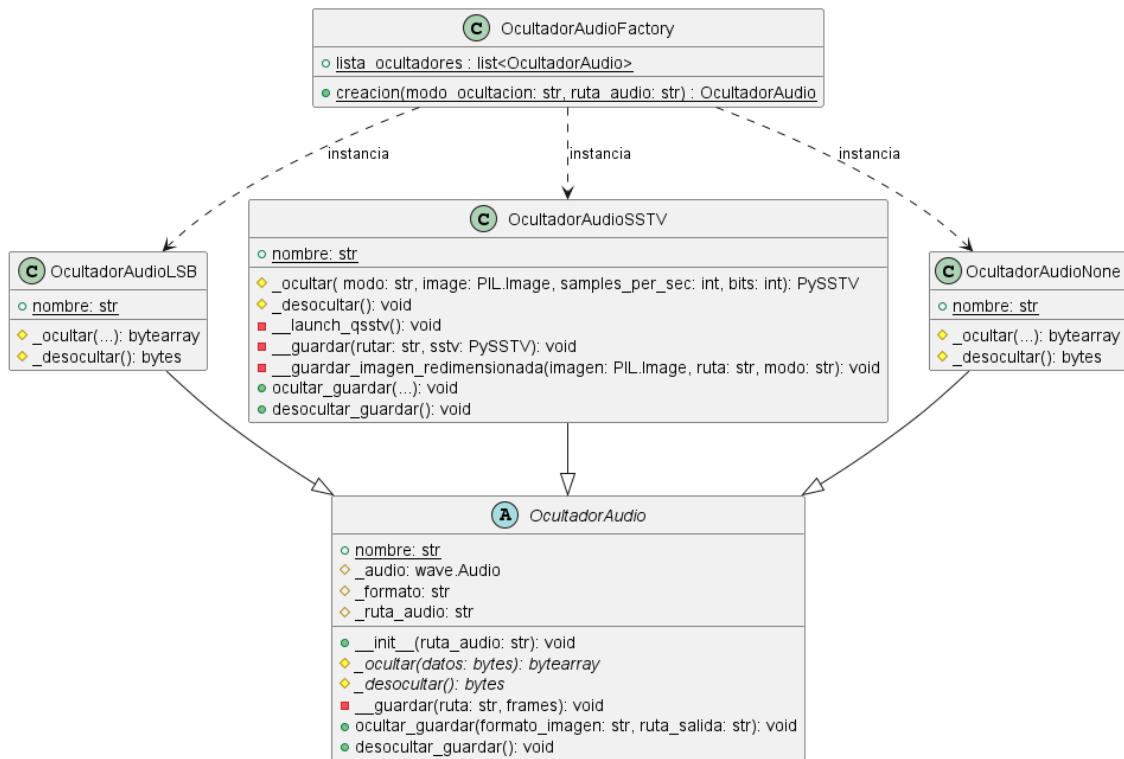


Figura 3.5: Diagrama de clases: Ocultadores en audio. Elaboración propia.

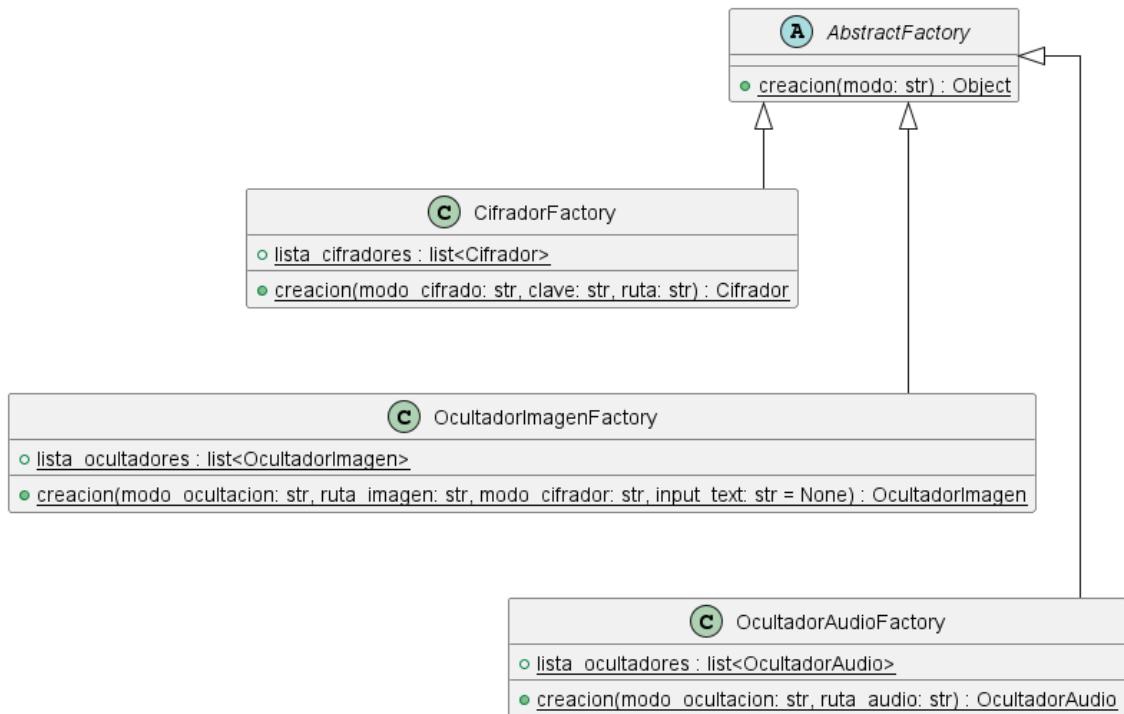


Figura 3.6: Diagrama de clases: Fábricas. Elaboración propia.

3.2. Diagramas de secuencia

A continuación, se describirán los diagramas de secuencia en los que se reflejan los flujos de ejecución del programa.

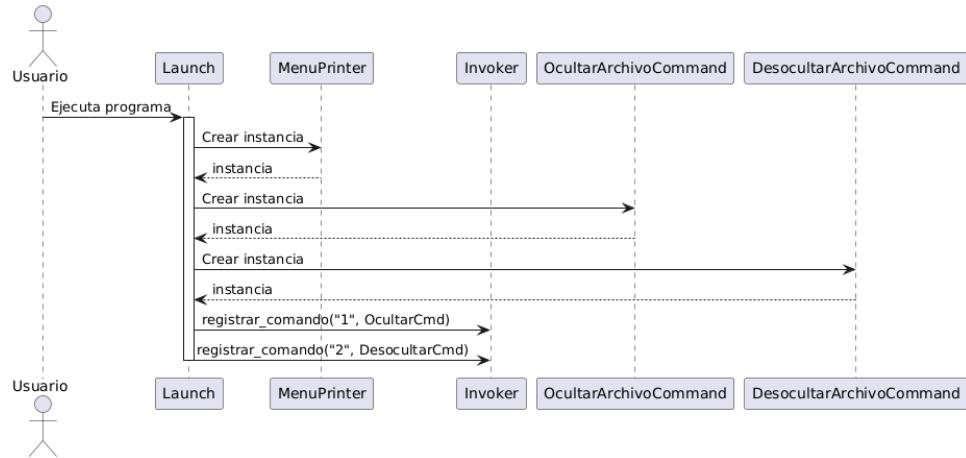


Figura 3.7: Diagrama de secuencia: *Launch* parte 1. Elaboración propia.

En primer lugar, como se ve en la figura 3.7, se crean las instancias encargadas de la impresión gráfica del menú por comandos, de la ejecución del comando seleccionado y del registro de los comandos principales necesarios. A continuación, se crean las instancias de cada comando y se registran en el invocador.

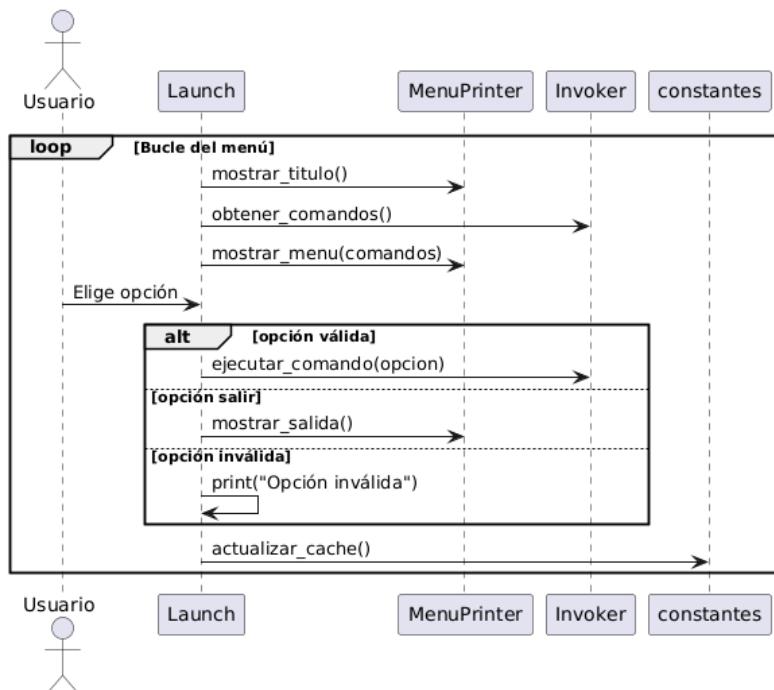


Figura 3.8: Diagrama de secuencia: *Launch* parte 2. Elaboración propia.

En la figura 3.8 se ve el bucle principal del lanzador, en el cual se muestra el título del programa y las opciones de los comandos disponibles, figura 3.9. La opción elegida se valida y, posteriormente, se ejecuta. Por último, la caché se actualiza.



Figura 3.9: Menú de inicio. Captura de pantalla de la herramienta.

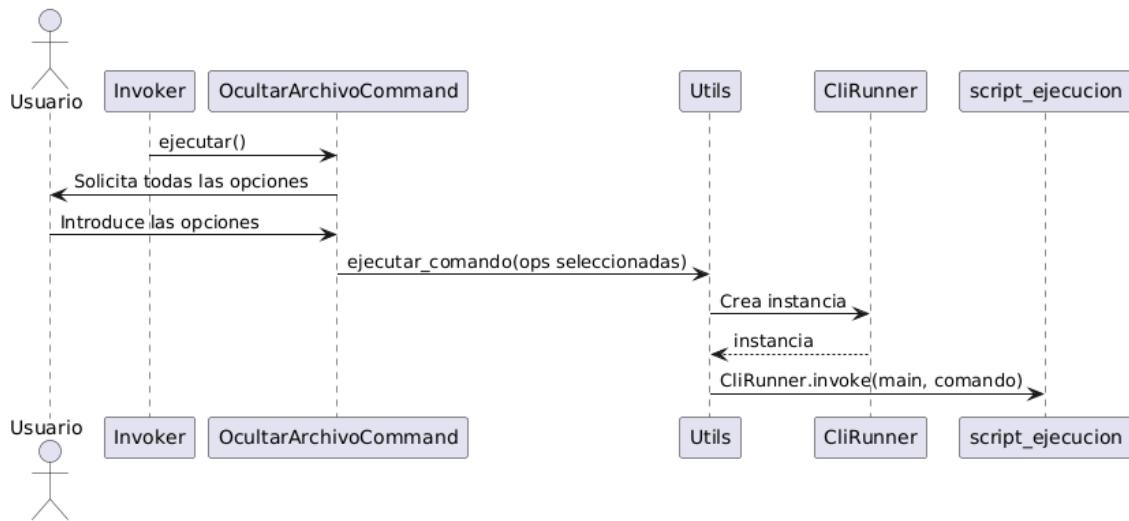


Figura 3.10: Diagrama de secuencia: Comando de ocultación. Elaboración propia.

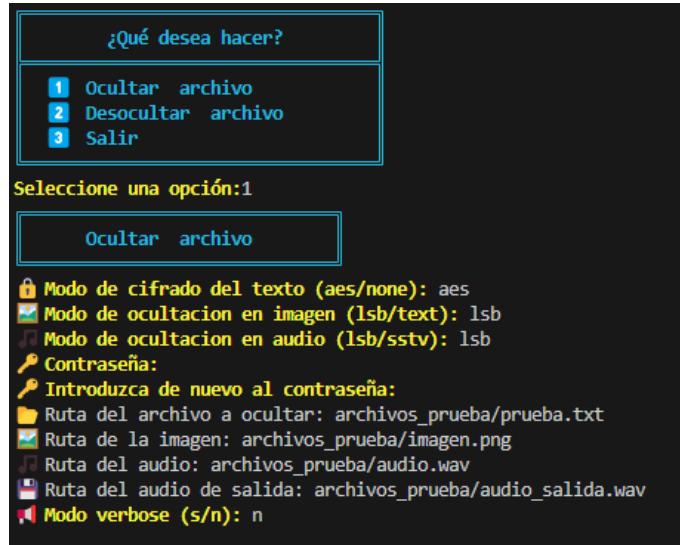


Figura 3.11: Opciones a llenar por el usuario. Captura de pantalla de la herramienta.

En caso de que se escoja ocultar (opción con índice 1) se invoca al método *ejecutar_comando(1)* del *invoker* que se encarga de que se llame al método *ejecutar()* del comando seleccionado, en este caso *OcultarArchivoComamand*. El comando correspondiente muestra al usuario las opciones posibles a la hora de cifrar y ocultar el archivo, figura 3.11. Posteriormente, las opciones seleccionadas se envían al módulo *utils*, mediante el cual se crea una instancia *CliRunner*, con la cual se invoca el script de ejecución con las instrucciones a ejecutar, tal como se observa en la figura 3.10.

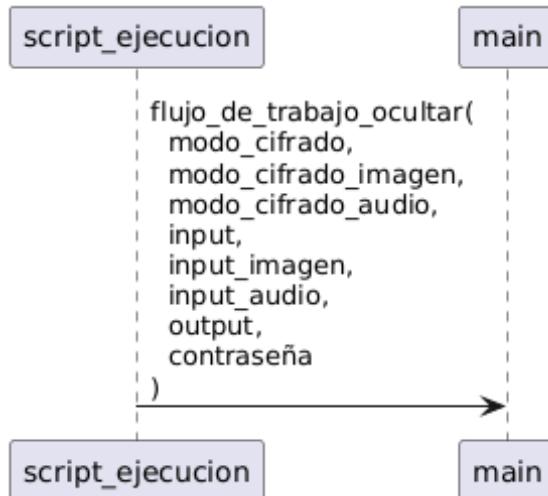


Figura 3.12: Diagrama de secuencia: Llamada al flujo de ocultación. Elaboración propia.

Como se muestra en la figura 3.12, el *main* del programa se invoca desde el script de ejecución, utilizando las opciones que se seleccionaron anteriormente como argumentos.

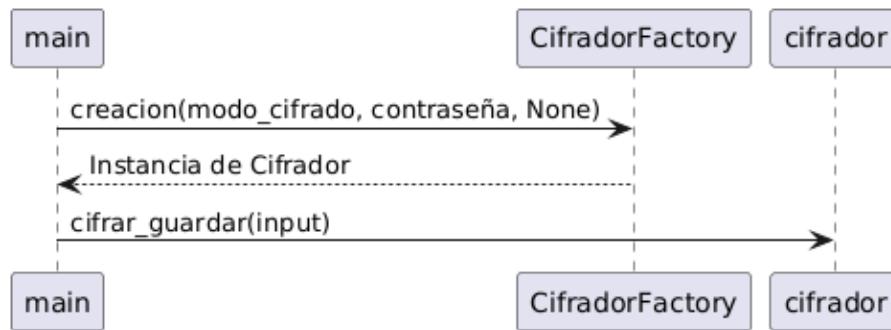


Figura 3.13: Diagrama de secuencia: Creación del Cifrador. Elaboración propia.

Ya en el main, una instancia de Cifrador se crea a través del *CifradorFactory* y luego esta se invoca para que ejecute su acción, figura 3.13.

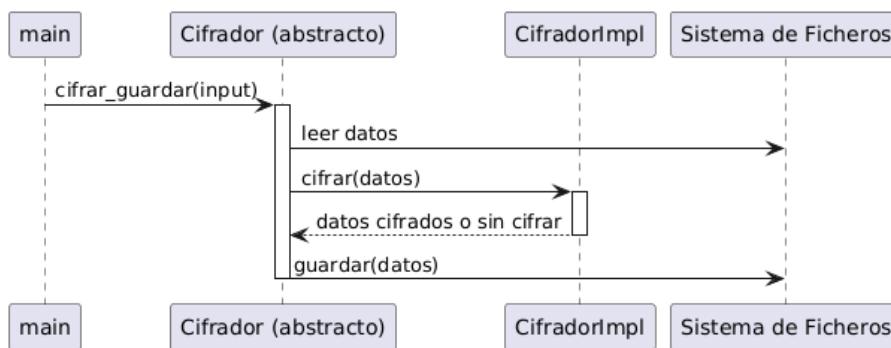


Figura 3.14: Diagrama de secuencia: Cifrado de los datos. Elaboración propia.

Primero en el Cifrador abstracto, se leen los datos desde el sistema de ficheros, luego estos se cifran en el Cifrador concreto seleccionado y, por último, se guardan de nuevo, tal como se representa en la figura 3.14.

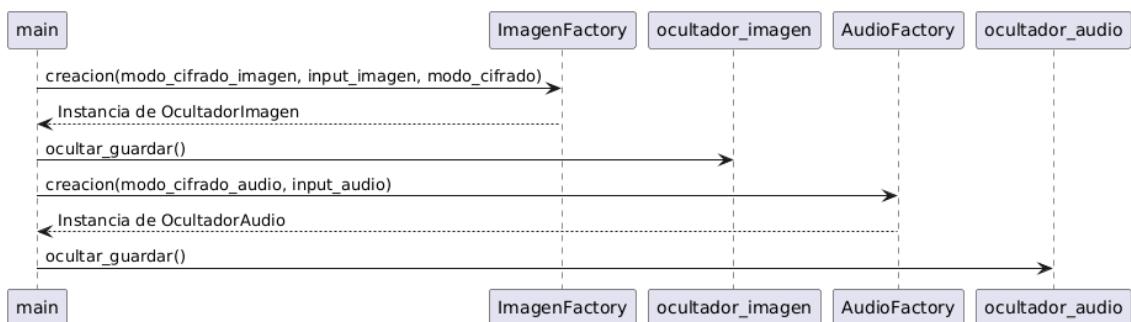


Figura 3.15: Diagrama de secuencia: Creación de los ocultadores. Elaboración propia.

A continuación, en la figura 3.15, una vez que los datos han sido cifrados, se crean instancias de los ocultadores en orden. En primer lugar, se instancia el ocultador en imágenes seguido del ocultador en audio. Tras cada instanciación, se invoca el método correspondiente de cada ocultador.

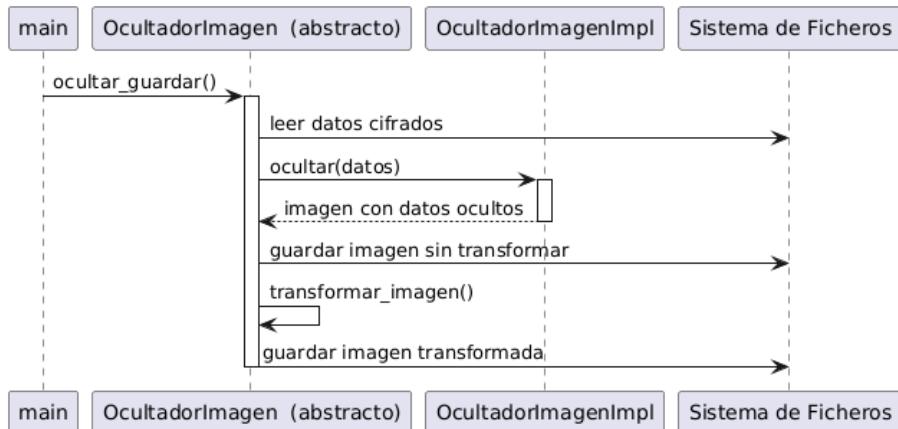


Figura 3.16: Diagrama de secuencia: Ocultación en la imagen. Elaboración propia.

Cuando se invoca el *ocultar_guardar()* del ocultador en imagen, como se puede ver en la figura 3.16, se leen los datos cifrados y se ocultan en la imagen. Posteriormente, se almacena la imagen con los datos ocultos, se le aplica una transformada para dificultar la identificación del orden de los datos ocultos y, por último, se guarda nuevamente.

En la figura 3.17 podemos ver cómo las imágenes son transformadas. La transformada que se usa consiste en dividir la imagen en 4 partes, reordenar estas 4 partes y sobre la nueva imagen se hace una negación bit a bit, lo que genera que visualmente los colores se inviertan. En el ejemplo de la figura 3.18 se puede observar que, si se escoge una imagen sin un patrón significativo, no se puede saber desde la imagen transformada cómo es la original, imposibilitando saber tanto los valores originales de los píxeles como el orden correcto de las partes de la imagen.



Figura 3.17: Imagen con ocultación por LSB antes y después de ser transformada. Elaboración propia.



Figura 3.18: Imagen con ocultación por LSB antes y después de ser transformada. Elaboración propia.

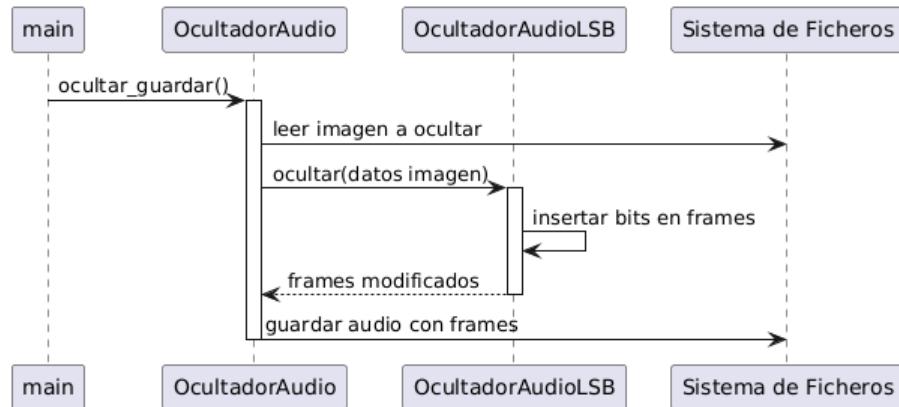


Figura 3.19: Diagrama de secuencia: Ocultación en un audio mediante técnica LSB. Elaboración propia.

En caso de querer ocultar en un audio mediante LSB, primero, la imagen con los datos ocultos que fue transformada se lee desde el sistema de ficheros, después se oculta en los frames del audio, y por último, se guarda el audio con datos ocultos, figura 3.19.

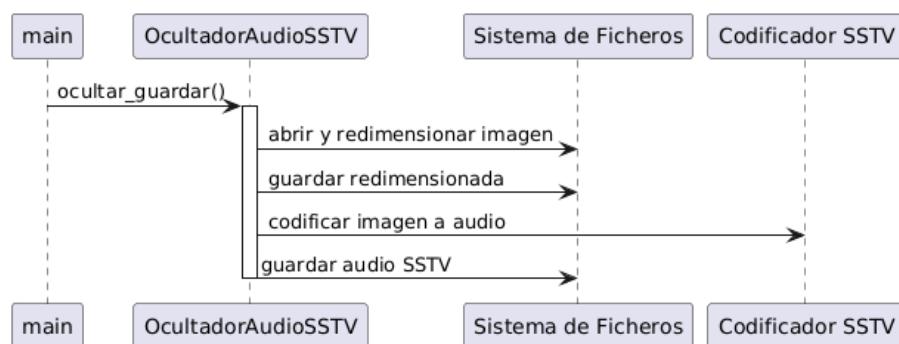


Figura 3.20: Diagrama de secuencia: Modulación SSTV de la imagen. Elaboración propia.

En caso de querer generar una señal SSTV, primero se lee la imagen desde el sistema de ficheros, se redimensiona y, posteriormente, se guarda. A continuación, esta imagen se modula en una señal de audio SSTV, la cual se guarda en el sistema de ficheros nuevamente, tal como se representa en la figura 3.20.

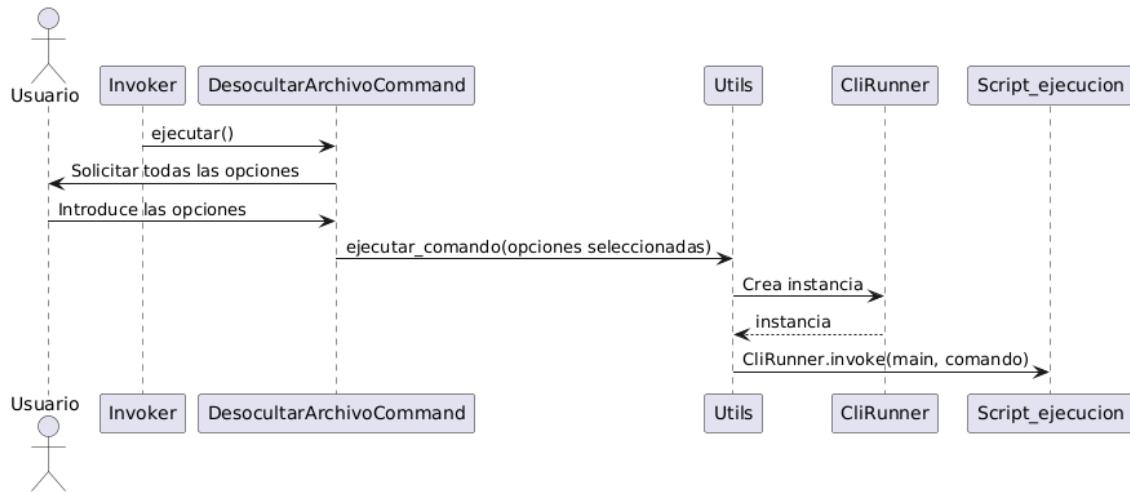


Figura 3.21: Diagrama de secuencia: Comando de desocultación. Elaboración propia.

En caso de que se escoja desocultar (opción con índice 2) se invoca al método *ejecutar_comando(2)* del *invoker* que se encarga de que se llame al método *ejecutar()* del comando seleccionado, en este caso *DesocultarArchivoCommand*, en el que se solicitan las opciones de desocultación al usuario. Posteriormente, las opciones seleccionadas se envían al módulo *utils*, mediante el cual se crea una instancia *CliRunner*, con la cual se invoca el script de ejecución con las instrucciones a ejecutar.

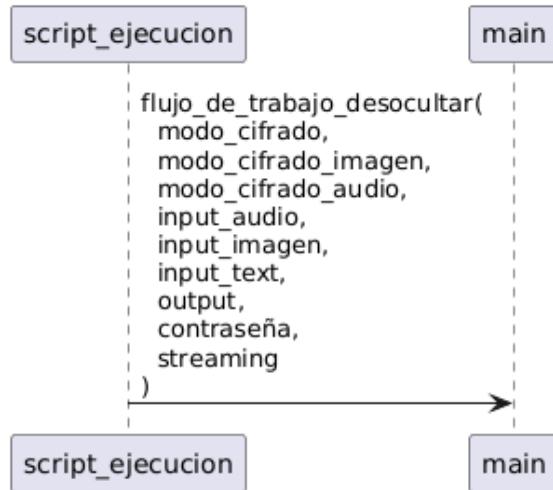


Figura 3.22: Diagrama de secuencia: Llamada al flujo de desocultación. Elaboración propia.

Como se muestra en la figura 3.22, el main del programa se invoca desde el script de ejecución, utilizando los datos de desocultación necesarios como argumentos.

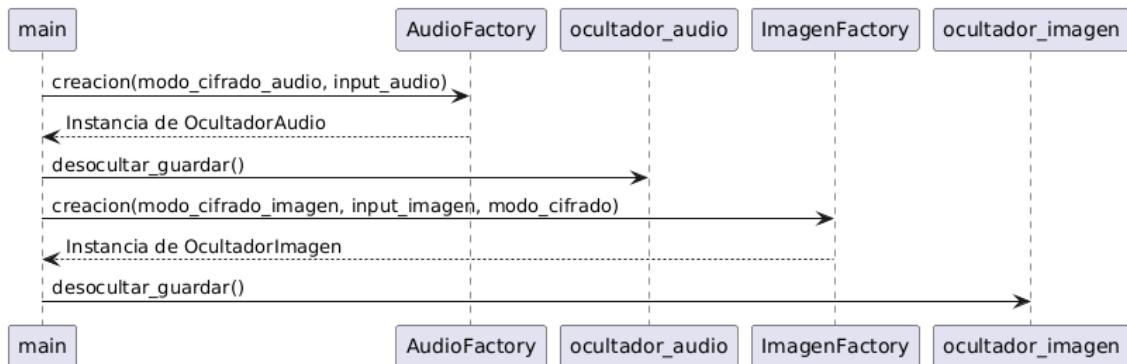


Figura 3.23: Diagrama de secuencia: Creación de los ocultadores. Elaboración propia.

En la figura 3.23 se puede ver que las instancias de los desocultadores se crean en orden, primero se crea la instancia del ocultador en audio y se extrae la imagen del audio, y a continuación, la instancia del ocultador en imagen y se extrae el mensaje de la imagen.

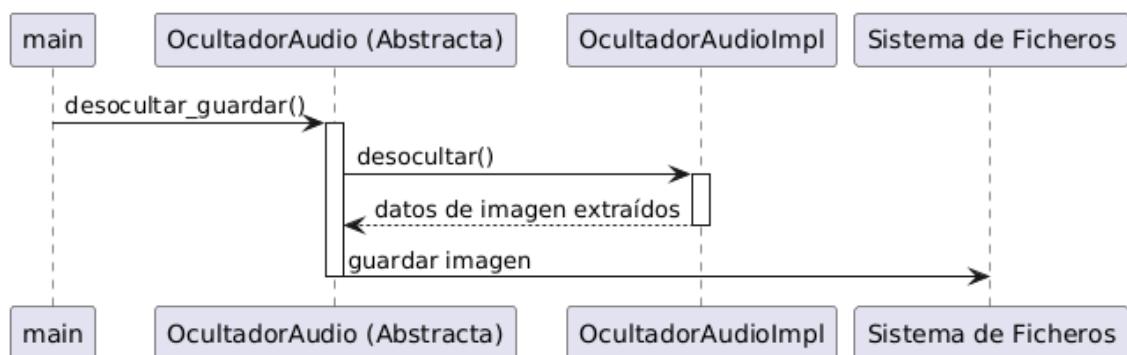


Figura 3.24: Diagrama de secuencia: Desocultación de una imagen en audio mediante técnica LSB. Elaboración propia.

Primero, se invoca al ocultador abstracto, el cual se encarga de invocar al desocultador concreto, por medio de este, el flujo de audio se lee y la imagen oculta se extrae y se almacena en el sistema de ficheros, tal como se observa en la figura 3.24.



Figura 3.25: Diagrama de secuencia: Decodificación del audio SSTV. Elaboración propia.

En caso de tratarse de un audio SSTV, se invoca el QSSTV y una serie de pasos se indican en la terminal dependiendo de si se seleccionó el modo streaming o no, figura 3.26. Posteriormente, la imagen extraída se guarda en el sistema de ficheros, figura 3.25.

```
Una vez abierto qsstv, elija el audio con ruta /home/alberto/tfg/TFG-PITEA/pitea/archivos_prueba/sstv.wav
Asegúrese de guardar la imagen como /home/alberto/tfg/TFG-PITEA/pitea/cache/cache_01-04-2025_16:52/desocultacion/imagen/imagen_contenedora_desocultacion.png
```

Figura 3.26: Pasos de decodificación SSTV. Captura de pantalla de la herramienta.

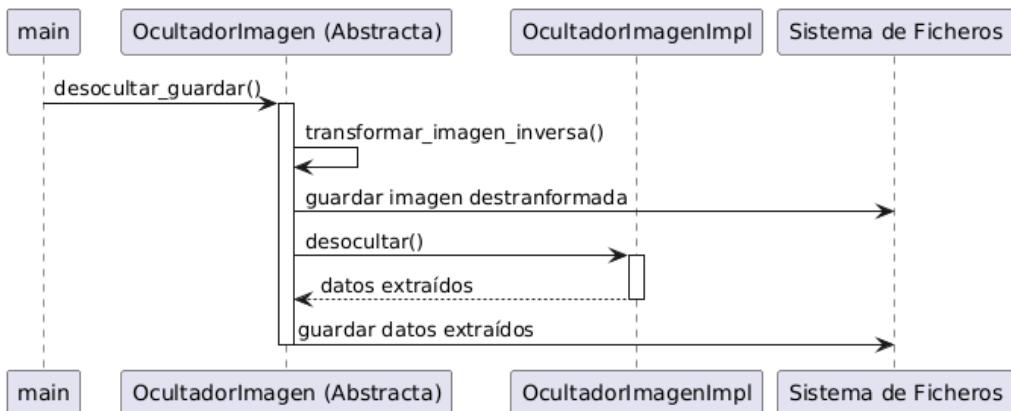


Figura 3.27: Diagrama de secuencia: Recuperación del mensaje oculto en la imagen. Elaboración propia.

A continuación, en el ocultador de imagen abstracto, se realiza la transformada inversa a la imagen para recuperar el orden original de los bytes y los valores correctos de los píxeles. Una vez realizada la transformada inversa, la imagen se guarda y el método desocultar del ocultador concreto se invoca para que los datos sean extraídos de la imagen, estos se guardan en el sistema de ficheros.

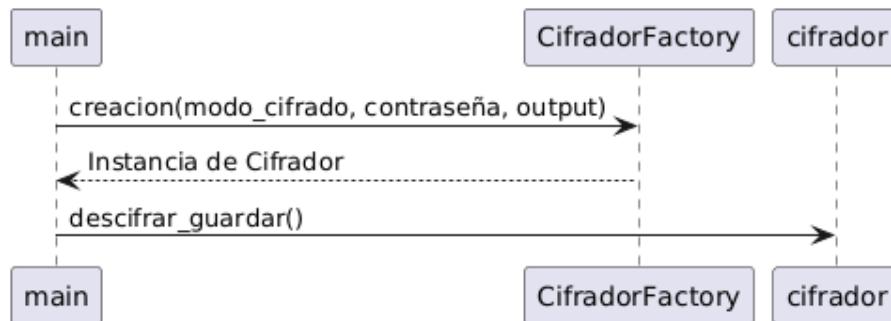


Figura 3.28: Diagrama de secuencia: Creación del Cifrador. Elaboración propia.

Después de la desocultación, se crea una instancia de Cifrador para posteriormente invocar al método de descifrado, figura 3.28.

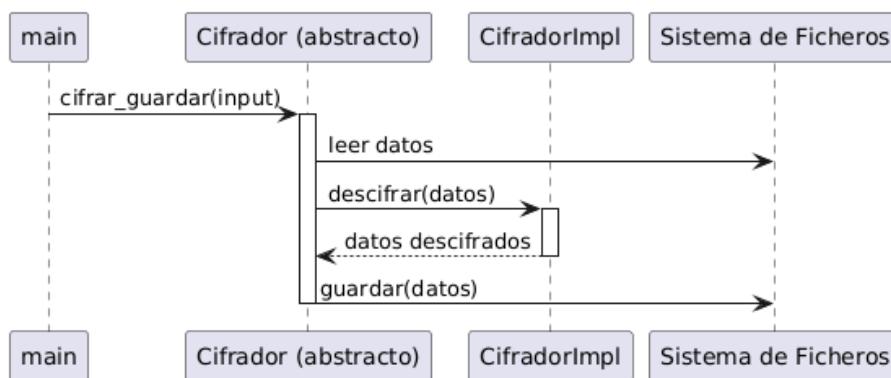


Figura 3.29: Diagrama de secuencia: Descifrado de los datos. Elaboración propia.

A continuación, los datos cifrados se leen en el Cifrador abstracto , desde donde se invoca el proceso de descifrado del Cifrador concreto, y los datos se convierten nuevamente a texto claro. Finalmente, los datos descifrados se guardan en el sistema de ficheros.

Capítulo **4**

Pruebas experimentales

En este capítulo se expondrán distintas pruebas realizadas a la herramienta para documentar cómo se comportan los archivos portadores en distintos entornos y analizar su eficacia.

A lo largo del capítulo se mencionará el archivo “pruebas.txt” siendo el contenido de este:

PITEA Protección de la información mediante técnicas de esteganografía acústica
Alberto Martín Oruña y Rodrigo Gallego Marín

4.1. Transmisión vía radiofrecuencia

En esta sección se hablará de cómo se comportan los archivos portadores al ser transmitidos mediante radiofrecuencia.

4.1.1. Generación del archivo portador

El audio portador ha sido generado como se indica en la figura 4.1, donde se puede apreciar que se ha generado un archivo de audio en formato WAV tras cifrar la información con el cifrado AES, ocultando el resultado del cifrado mediante el plasmado del texto en una imagen, generando la figura 4.2, y convirtiendo esa imagen en una señal de audio codificada en formato SSTV con 16 bits de profundidad y una frecuencia de 48000 Hz.



Figura 4.1: Generación de audio portador, AES, *text*, SSTV. Captura de pantalla de la herramienta.

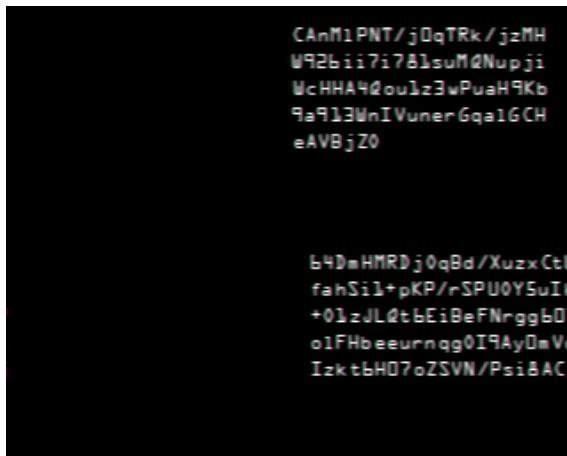


Figura 4.2: Imagen generada por el *OcultadorText*. Generada por la herramienta.

4.1.2. Resultado

El archivo portador fue transmitido en VHF (*Very High Frequency*), en la banda de radioaficionados y en la frecuencia destinada a SSTV, con un walkie-talkie¹ y fue recibido por el nodo de radio escucha² instalado en la Facultad de Informática de la UCM, como colaboración con la Unión de Radioaficionados Españoles (Sección Guadarrama), figura 4.3. La imagen recuperada en el nodo de radio escucha es la visible en la figura 4.4.

La transmisión de datos cifrados mediante VHF en las bandas de radioaficionados es ilegal, pero se ha realizado con el fin de probar la herramienta y demostrar

¹Baofeng UV-5R (<https://baofeng.es/walkie-talkie/baofeng-uv-5r-8w.html>)

²[Http://ea4urg.ucm.es:8080/](http://ea4urg.ucm.es:8080/).

su eficacia. No se debe hacer con regularidad.

Se puede apreciar que la imagen recuperada está un poco distorsionada debido a que el nodo no tiene el mejor de los sistemas de decodificación, pero con un sistema de decodificación mejor o una tipografía más grande se podría mejorar la legibilidad del texto.

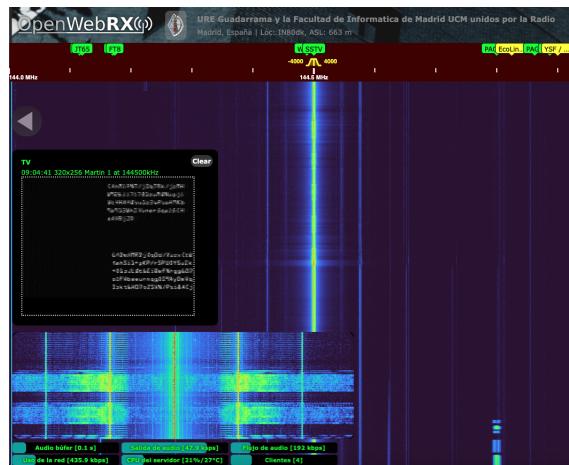


Figura 4.3: Interfaz del nodo de radioescucha utilizado tras la decodificación de la señal SSTV mandada. Captura de pantalla.

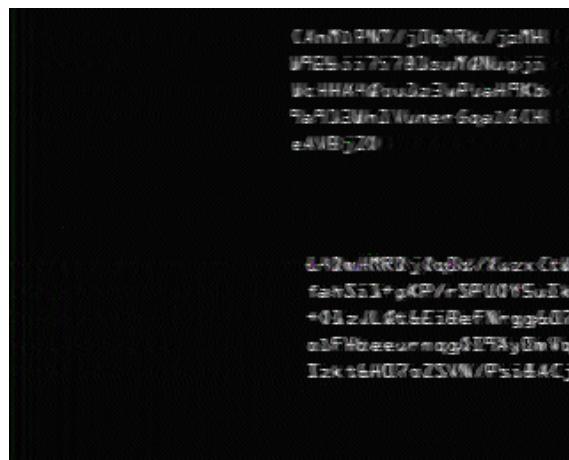


Figura 4.4: Imagen recuperada en la prueba de campo, 4.1. Generada por la herramienta.

4.2. Difusión en la red pública

En esta sección se hablará de cómo se comportan los archivos portadores al ser transmitidos por la red pública.

4.2.1. Generación de los archivos portadores

Para esta sección se han generado varios archivos portadores con el fin de comprobar todas las funcionalidades de la herramienta.

1. Audio portador sin cifrado, ocultador en imagen LSB y ocultador en audio LSB, figura 4.5, generando 4.6.
2. Audio portador con cifrado AES, ocultador en imagen mediante el plasmado de texto en la imagen y ocultador en audio mediante su paso a audio codificado con SSTV, figura 4.1.



Figura 4.5: Generación de audio portador, sin cifrado, LSB, LSB. Captura de pantalla de la herramienta.



Figura 4.6: Imágenes resultantes de la generación del archivo 1, imagen original, antes y después de la transformada respectivamente. Generadas por la herramienta.

No se ha generado un archivo con ocultador en imagen LSB y ocultador en audio con SSTV debido a que SSTV es un formato sujeto a ruido, lo que significa que no es seguro que se recupere la imagen idéntica pixel a pixel, lo que hace que no se pueda recuperar el texto de la imagen de manera segura.

El audio 2 se ha añadido a un vídeo para poder ser subido a TikTok.

4.2.2. Resultado

La primera prueba que se ha realizado es con el archivo 2 en la cual se ha aprovechado la funcionalidad de captura de SSTV en *streaming* para capturar el audio a la vez que se reproduce el video en TikTok, figura 4.7. Tras la cual se ha generado la imagen 4.8 donde se puede ver el antes y después de la transformada.

Los resultados son buenos, ya que el ojo humano puede reconocer los caracteres, haciendo que se pueda recuperar el texto original tras reconocer esos caracteres por el desocultador *text* o transcribiendo la imagen a un archivo de texto y descifrándolo posteriormente, ambas soluciones implementadas en la herramienta.

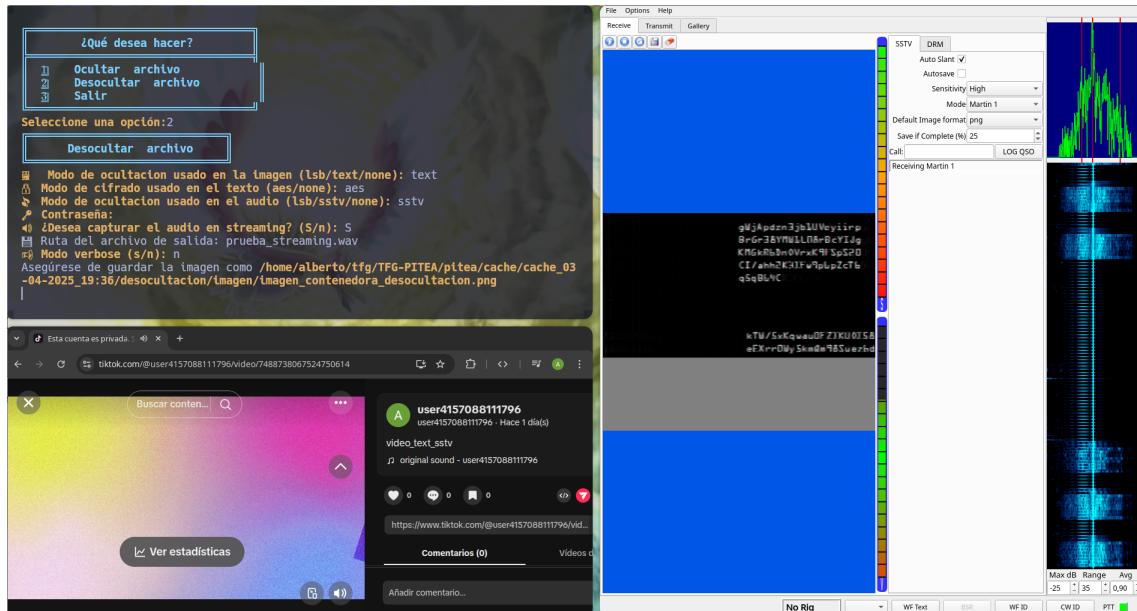


Figura 4.7: Captura en *streaming* en Tiktok, usando como decodificar QSSTV. *Collage* formado por una captura de pantalla de la herramienta, arriba izquierda; interfaz de QSSTV, derecha; e interfaz web de TikTok, abajo izquierda.



Figura 4.8: Imagenes resultantes de la prueba con el archivo 2, antes y después de la transformada respectivamente. Generadas por la herramienta.

La segunda prueba se ha realizado con el archivo 1, ha consistido en ser mandado por WhatsApp, también podría haber sido cualquier servicio de correo que no comprima archivos multimedia, y al ser descargado ha sido pasado por la herramienta, dando como resultado las siguientes figuras: 4.9 y 4.10.

El resultado es exitoso, ya que la salida de la herramienta, el archivo “salida_lsb_lsb.txt” como se puede ver en la figura 4.9, contiene:

PITEA Protección de la información mediante técnicas de esteganografía acústica
Alberto Martin Oruña y Rodrigo Gallego Marin

que es exactamente igual al archivo inicial.



Figura 4.9: Llamada a la herramienta para la desocultación de 1. Captura de pantalla de la herramienta.

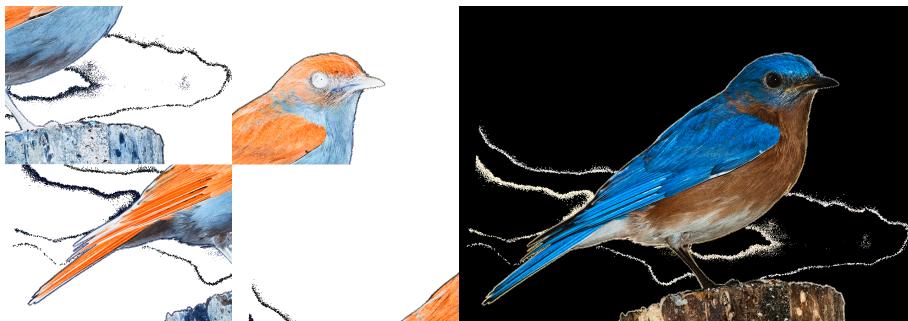


Figura 4.10: Imagenes resultantes de la desocultación del archivo 1, imagen recuperada en los LSB del audio e imagen destransformada respectivamente. Generadas por la herramienta.

4.3. Análisis de *OcultadorText*

En esta sección se hablará de los análisis realizados sobre el ocultador en imágenes que se basa en el plasmado de texto en una imagen, tratando desde la generación de los conjuntos de datos utilizados hasta las conclusiones resultantes.

El análisis se ha dividido en dos partes, una por motor OCR usado, *Tesseract-OCR* y *EasyOCR*. Sin embargo, la estructura del experimento ha sido idéntica en ambos casos, solo variando los resultados. Las gráficas de cada parámetro han sido realizadas agrupando por ese parámetro y usando como función resumen la media aritmética. Se ha elegido la media frente a otras opciones como la mediana, valor máximo o valor mínimo, ya que el fin de este análisis es ver cuánto afecta cada variable al resultado, y el uso de la media como función resumen proporciona un valor representativo del comportamiento general..

En primer lugar se generó una lista de listas de palabras y números en inglés de tamaño aleatorio, entre 1 y 25, de longitud 10. Los números aparecen con un 5 % de probabilidad. Los componentes de esta lista serán los utilizados como los secretos que se van a ocultar. Una vez generado el conjunto de secretos que vamos a procesar, se generó un conjunto de datos, *dataset*, ejecutando el ocultador de imágenes correspondiente con tamaños de fuente entre 1 y 25 con un paso de 2, 10 fuentes tipográficas³ y una anchura máxima de imagen entre 800 píxeles y 10800 con paso 1000, estos intervalos y saltos han sido los interpretados como más relevantes y óptimos teniendo en cuenta las capacidades de cómputo que se tenían en el momento del experimento, ya que el uso de motores OCR consume muchos recursos hardware. Resultan dos *datasets*⁴ similares a 4.1, siendo “resultado” la similitud entre el texto original y el texto extraído. La similitud ha sido calculada usando el algoritmo *Gestalt Pattern Matching*.

Tamaño	Fuente	Anchura	Resultado	Texto original	Texto extraido
9	OCR-a-std	800	65.9	hardpan...	harbpam...
7	OCRb	1800	72.3	quintary...	qunitary...
5	OCR-a-std	4800	88.1	depside...	deps1de...
3	OCRb	3800	91.7	unsaintlike...	unsa1nt1ike...
1	OCR-a-std	2800	50.2	cran drachm...	crarn drachm...
7	OCR-a-std	6800	95.0	isothiocyan...	isothiocvano...
3	OCR-a-std	9800	92.4	unmapped...	unmapp3d...
5	Arial	8800	87.5	diluvianism...	diluv1an1sm...
9	OCR-a-std	7800	93.2	hardpan diluvianism...	harclpan diluian1sm...
1	OCR-a-std	800	41.8	drachm...	drachm...

Tabla 4.1: Ejemplo de *datasets* generados.

³Véase el apéndice C para ver las fuentes tipográficas concretas.

⁴Se pueden encontrar en el repositorio de la herramienta <https://github.com/Alberto12x/PITEA/tree/main/experimental>.

4.3.1. Análisis de *Tesseract-OCR*

Lo primero a mencionar en este análisis es que la herramienta no detecta espacios en blanco, por lo que se ha calculado la similitud entre los textos originales y los extraídos eliminándolos.

El primer análisis que se ha realizado es ver cómo afecta el tamaño de la fuente al resultado, figura 4.11. En la gráfica se puede apreciar cómo con tamaños de fuentes inferiores a 7, los textos extraídos no tienen ninguna similitud con los originales o simplemente no se ha extraído ningún texto. Entre un tamaño de 7 a 9 la herramienta empieza a detectar mejor los textos y es a partir de un tamaño de fuente 11 cuando la herramienta alcanza resultados óptimos superando el 95 % de similitud y a partir de 17 se alcanza una similitud casi del 100 %.

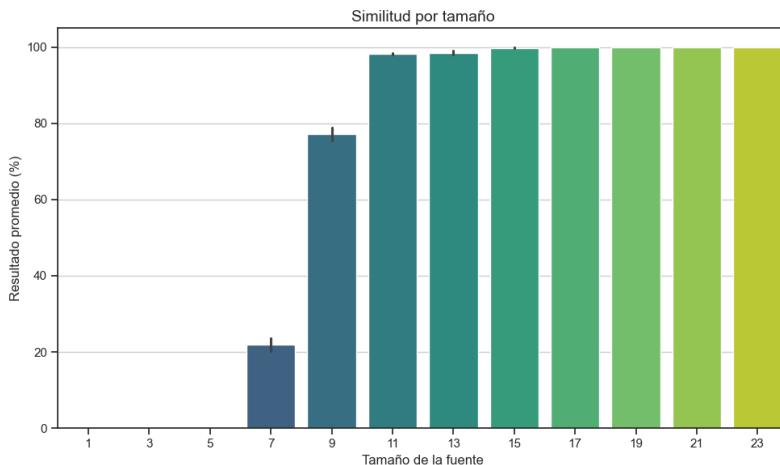


Figura 4.11: Gráfica de resultado (%) variando el tamaño de la fuente utilizada, agrupando por ello y calculando la media del resultado para cada grupo. *Tesseract-OCR*. Elaboración propia.

El siguiente análisis que se realizó fue ver cómo las tipografías utilizadas afectaban a la herramienta, figura 4.12. En la gráfica se puede ver que aunque se ven pequeñas variaciones en los resultados, estas no son superiores al 15 % por lo que la fuente tipográfica utilizada no afecta de manera significativa al resultado. Es de resaltar que las fuentes específicas para motores OCR no destacan sobre el resto, siendo posible que esto se deba a las fuentes elegidas para entrenar *tesseract-OCR*.

Posteriormente se analizó el efecto que tenía la anchura máxima de las imágenes generadas en el resultado, esto debido a que dependiendo de la anchura, la cantidad de palabras por línea varía. En su gráfica, figura 4.13, se puede ver que no afecta de manera significativa en el resultado, ya que tanto sus valores medios como márgenes de error son idénticos o muy parecidos, siendo solo necesario que la anchura máxima sea igual o mayor que el carácter más ancho del mensaje.

Se siguió graficando todos los parámetros a la vez, figura 4.14. En esta gráfica,

de 5 dimensiones, se ha utilizado el color, la forma el tamaño de los puntos como dimensiones extra. Se puede apreciar que el único parámetro relevante es el tamaño de la fuente, ya que solo se aprecian cambios al recorrer ese eje.

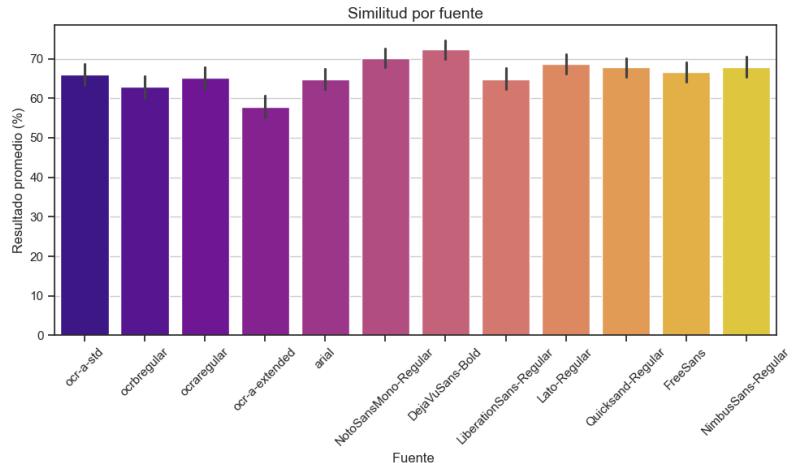


Figura 4.12: Gráfica de resultado (%) variando la fuente utilizada, agrupando por ella y calculando la media del resultado para cada grupo. *Tesseract-OCR*. Elaboración propia.

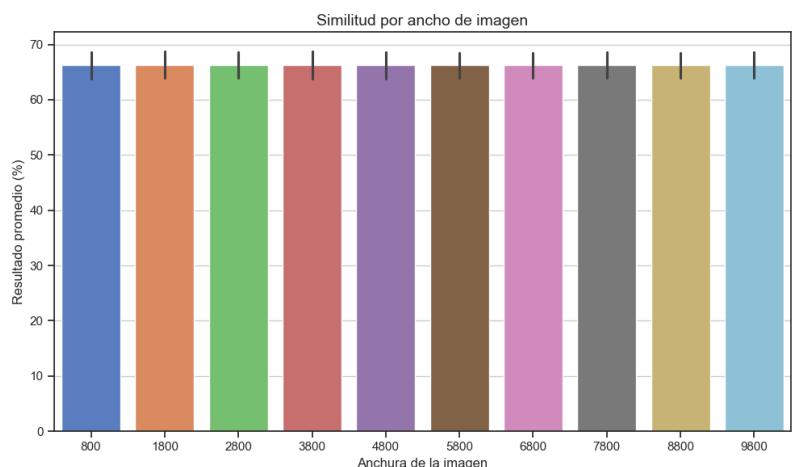


Figura 4.13: Gráfica de resultado (%) variando la anchura de la imagen utilizada, agrupando por ella y calculando la media del resultado para cada grupo. *Tesseract-OCR*. Elaboración propia.

Para finalizar los análisis sobre este motor, se analizó si la longitud del mensaje a ocultar y la existencia de números afectaban al resultado, figura 4.15. Se puede concluir que ninguna de las dos variables mencionadas afecta al resultado, ya que no se aprecian cambios al recorrer sus ejes.

Tras este análisis se puede concluir que el tamaño de la fuente tipográfica utilizado es lo que más afecta a la precisión de la herramienta, afectando el resto de

parámetros de manera casi superflua, pero siendo necesarios ya que la desocultación se puede realizar también sin motor OCR, usando el ojo humano.

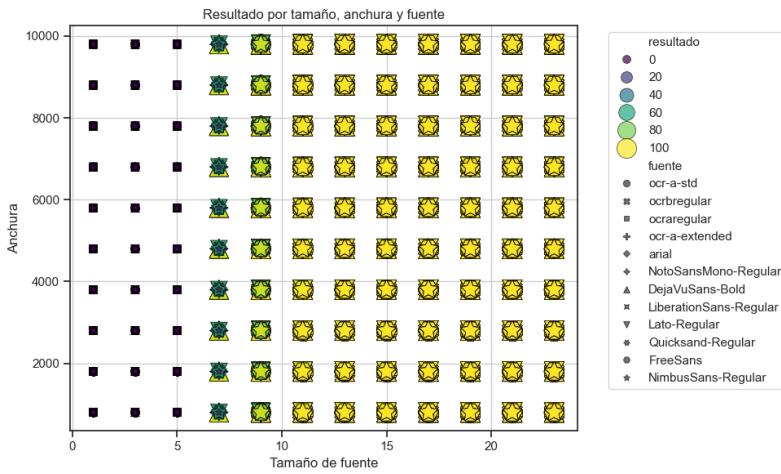


Figura 4.14: Gráfica de todos los parámetros configurables. *Tesseract-OCR*. Elaboración propia.

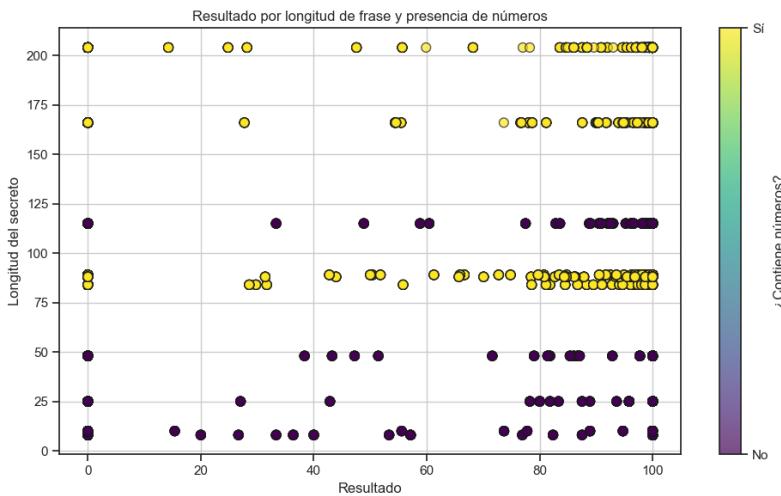


Figura 4.15: Gráfica de resultado (%) variando la longitud del secreto y el contenido numérico. *Tesseract-OCR*. Elaboración propia.

4.3.2. Análisis de *EasyOCR*

Manteniéndose la misma estructura del análisis de la sección 4.3.1, se empezó estudiando la influencia del tamaño de la fuente tipográfica en el resultado, figura 4.16. En esa gráfica se puede ver una tendencia bastante lineal a resultados mayores según se aumenta el tamaño de la fuente, además de que los mejores resultados obtenidos rondan el 60 %.

Continuando, en la figura 4.17 se puede ver que la fuente utilizada no varía demasiado el resultado, un 10 % en el mayor de los casos. Es cuanto menos curioso que

las fuentes específicas para OCR tampoco destacan en este motor.

En la figura 4.18 se puede ver cómo el ancho de la figura afecta negativamente al resultado, siendo las imágenes más anchas las que peor resultado ofrecen.

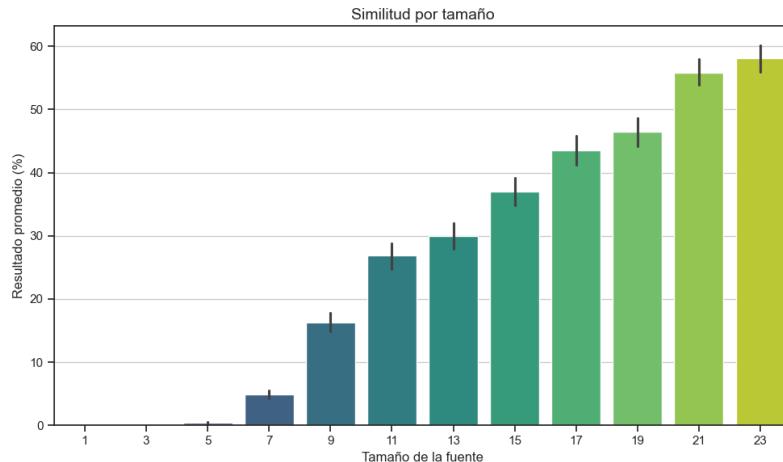


Figura 4.16: Gráfica de resultado (%) variando el tamaño de la fuente utilizada, agrupando por ello y calculando la media del resultado para cada grupo. *EasyOCR*. Elaboración propia.

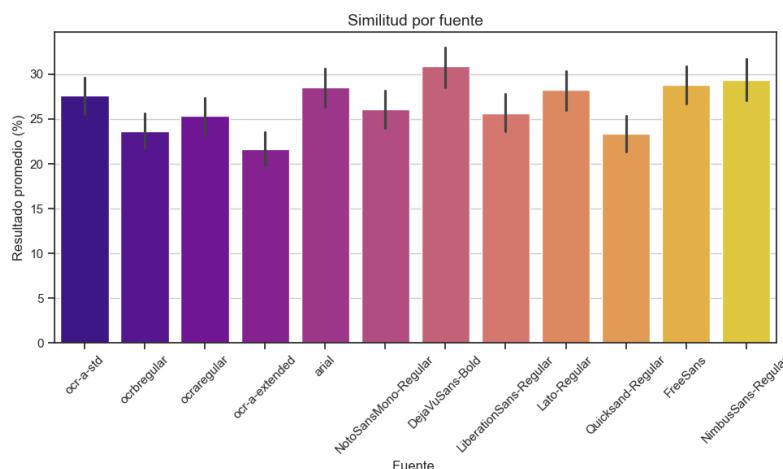


Figura 4.17: Gráfica de resultado (%) variando la fuente utilizada, agrupando por ella y calculando la media del resultado para cada grupo. *EasyOCR*. Elaboración propia.

En el caso de la figura 4.19, se volvieron a visualizar todos los parámetros utilizando formas, colores y tamaños como dimensiones extra. Se puede ver lo anteriormente mencionado que los resultados, color y tamaño, mejoran según aumenta el tamaño de la fuente y disminuye la anchura de la imagen. Además, se puede apreciar que en valores intermedios de tamaños y anchuras, ciertas fuentes son mejores que

otras, como el caso de *FreeSans* con tamaño 10 y anchura 6000 píxeles.

Dada la tendencia a mejorar resultados aumentando el tamaño de la fuente, se decidió aumentar el *dataset* para ver cómo evoluciona esa tendencia, pero esta vez solo usando una fuente y una anchura de imagen óptima para reducir la carga de trabajo al equipo encargado de generarla. El resultado es la figura 4.20 donde se puede ver que la tendencia a mejorar resultados se vuelve monótona a partir de un tamaño superior a 20 aproximadamente, experimentando cambios mínimos a partir de ese punto. Además, de manera indirecta se puede ver que el resultado asciende hasta el 80 % usando las configuraciones adecuadas.

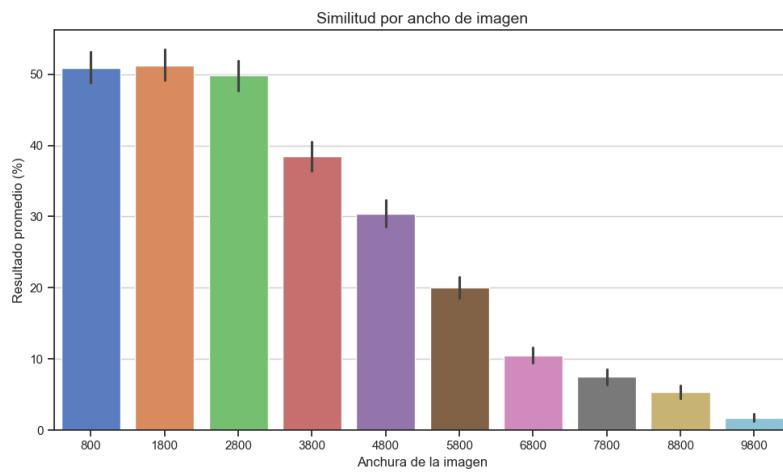


Figura 4.18: Gráfica de resultado (%) variando la anchura de la imagen utilizada, agrupando por ella y calculando la media del resultado para cada grupo. *EasyOCR*. Elaboración propia.

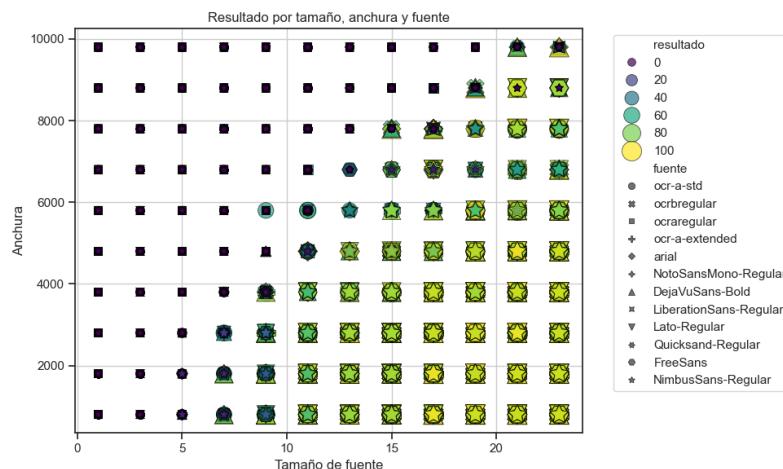


Figura 4.19: Gráfica de todos los parámetros configurables. *EasyOCR*. Elaboración propia.

Tras el análisis, podemos concluir que los parámetros introducidos varían de manera significativa el resultado, por lo que es importante tenerlos en cuenta al

ejecutar la herramienta.

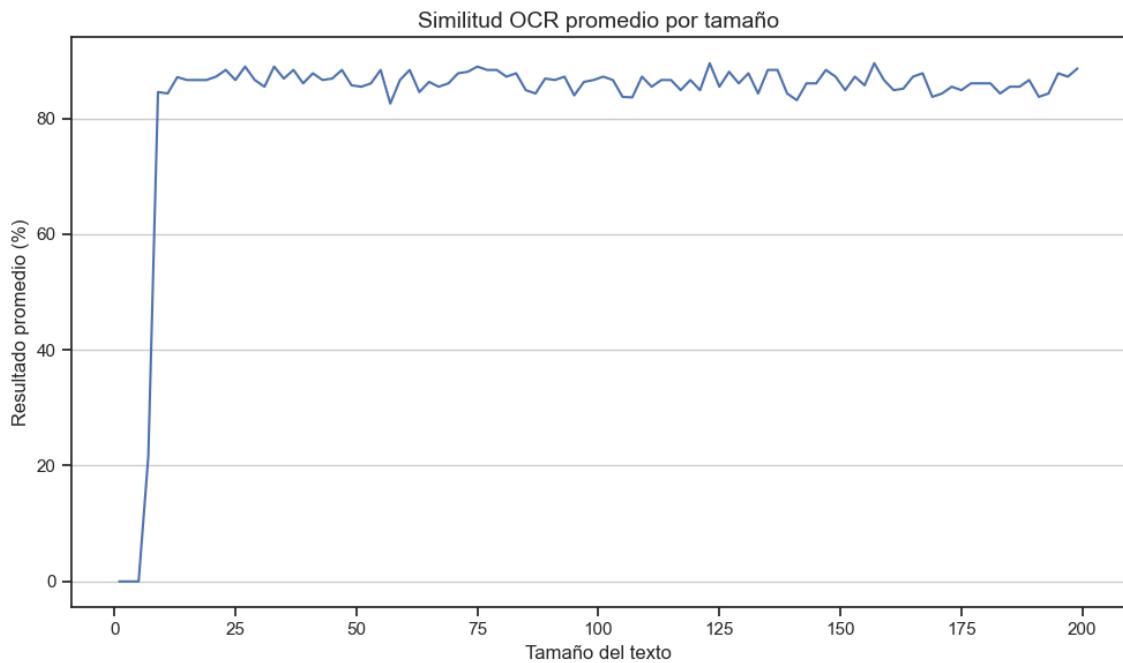


Figura 4.20: Gráfica de resultado (%) variando el tamaño de la fuente utilizada y usando el resto de parámetros con sus valores más eficaces. Fuente tipográfica: DejaVuSans-Bold y anchura de imagen 800 píxeles. *EasyOCR*. Elaboración propia.

Capítulo 5

Tecnologías descartadas

En este capítulo se describirán las tecnologías que han sido descartadas en la creación de la herramienta.

■ Marcador en método LSB

Se probó el uso de un marcador implementado mediante una secuencia de bits para determinar hasta qué píxel habría que leer en la desocultación LSB, el marcador se introducía a continuación de la secuencia de bits del secreto. Este método fue descartado ya que existía la posibilidad de que el secreto contuviera una secuencia de bits similar a los utilizados para representar el marcador, lo que llevaría a la pérdida de datos. Supongamos que utilizamos como marcador la secuencia 010111010, pues existe la posibilidad de que la secuencia de bits que componen el secreto contenga esta secuencia de bits, lo que nos llevaría a no leer el secreto completo ya que abandonaríamos la lectura sin llegar al final.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones de la herramienta

Tras finalizar la herramienta y mirar hacia atrás pasando por cada uno de los capítulos de este documento, se puede concluir que se ha creado una herramienta por línea de comandos modular y funcional cuyo propósito es combinar criptografía y esteganografía para poder comunicarse de manera secreta en la red pública. Para justificar esta conclusión, se recordarán los objetivos definidos en la sección 1.2:

- **Creación de una CLI (*Command-line interface*):**
Diagrama de clases y explicación en el capítulo 3.
- **Implementación de algoritmos para la ocultación y desocultación tanto de texto en imagen como de imagen en audio:**
Explicados en el capítulo 3 donde se puede ver que hemos implementado algoritmos basados en el uso de LSB, basado en el plasmado de texto en imagen y transformándolo para que no sea visible y basados en la transformación de archivos en ondas acústicas.
- **Tratamiento de ondas acústicas en la informática:**
Explicado en la sección 2.1.2.
- **Uso de criptografía de clave secreta:**
Explicado sus bases en la sección 2.3 y explicando su implementación en el capítulo 3.

6.2. Trabajo futuro

Mirando al futuro y con la objetividad de que todo es posible de mejorar, hay varias cosas que mejoraría sin duda la herramienta:

- Implementación de código corrector de errores, para evitar posibles ataques de intermediario y ruido en el canal de transmisión.

- Implementación de algoritmos de ocultación para más formatos de archivos. Así se dispondría de más opciones a la hora de ocultar.
- Posibilidad de añadir nuevos cifrados o ocultadores mediante *plugins* sencillos para cualquier usuario.
- Diseño de una GUI (*Graphical User Interface*) para ser más amigable con el público que no disponga de conocimientos para utilizar la herramienta mediante la CLI.
- Implementación de un decodificador para SSTV para evitar abrir QSSTV y que sea el proceso automático.
- Implementación de un motor OCR (*Optical Character Recognition*) específico para una tipografía propia y específico para nuestro contexto, letras negras en un fondo blanco.
- Análisis del *OcultadorText* más potente, generando un *dataset* mucho más grande utilizando *PySpark* y algún servicio en la nube como GCP (*Google Cloud Console*).
- Implementación de técnicas basadas en el dominio de la transformación.
- Uso de criptografía pública para el intercambio de claves secretas.

Introduction

6.3. Motivation

The main motivation for which we have decided to undertake this project is that, according to Kemp (2024), a large volume of data is moved on social networks (measurable on the scale of exabytes), they have a massive number of users, exceeding 5 billion active users, and an average usage time of 2 hours and 23 minutes per day, but we spend an average of 6 hours and 40 minutes online daily. Furthermore, the most widely used social network in 2024 was TikTok, which is based on short-duration videos, allowing videos of up to 60 seconds in its origins and currently up to 60 minutes. This makes the transmission of secret information through videos with audio on the public network go unnoticed due to its abusive but common use today.

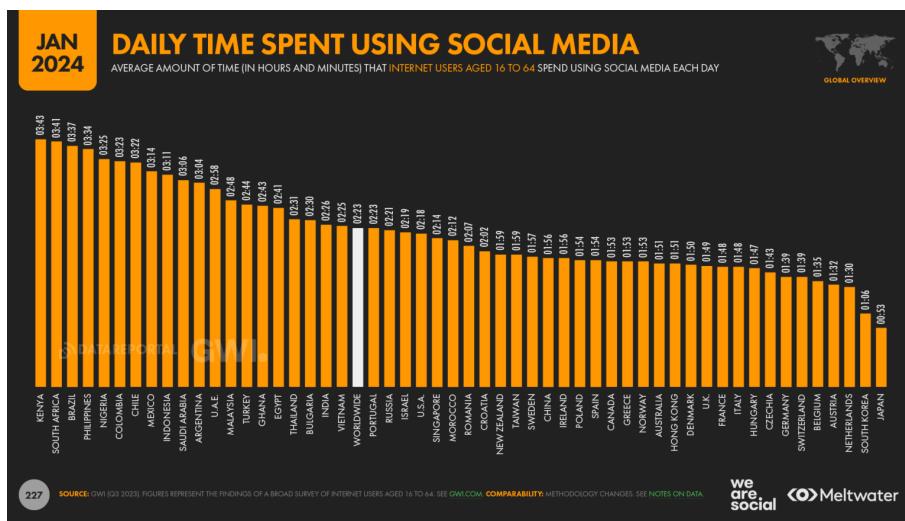


Figure 6.1: Graph of social media usage time by country. Kemp (2024).

To avoid confidentiality risks in the data, the use of steganography becomes fundamental, which will be introduced in more detail in this document, but for now, we will define it as techniques for hiding information, to ensure that if someone obtains a video legally or illegally, they are unable to detect that it contains valuable information, as this is hidden and appears to be just another video in the sea of bytes.

6.4. Objectives

The objective of this project is to develop a tool capable of using methods for transforming text into images and images into audio in order to transmit information secretly and take advantage of the intense use of social networks that exists today.

To achieve this, we will focus on:

- Creation of a CLI (Command-line interface).
- Implementation of algorithms for hiding and revealing both text in images and images in audio.
- Processing of acoustic waves in computing.
- Use of secret key cryptography.

6.5. Work plan

In this section, the detailed work plan is described with which it is expected to achieve the project's objectives.

6.5.1. Stage 1: Preliminary Research

Objective: Conduct preliminary research before the development of the tool in order to become familiar with the necessary knowledge and techniques.

Tasks:

- Search for information on steganography and cryptography.
- Search for information on multimedia file encoding.
- Analyze existing tools.

6.5.2. Stage 2: Development of a Basic Functional Prototype

Objective: Create a functional prototype of the tool that performs hiding and uncovering using LSB (Least Significant Bit).

Tasks:

- Develop a sequential program that carries out the complete process of hiding and uncovering.
- Implement basic functionalities, such as encrypting text using AES and hiding the text and image using the LSB hiding method.
- Conduct tests to verify that the basic system functions correctly and that the information is not modified in the process.

6.5.3. Stage 3: Modularization and Enhancement of Functionalities

Objective: Modularize the program and add additional functionalities, such as generating audio from images to transmit them via SSTV.

Tasks:

- Divide the code into modules by making appropriate use of OOP (Object-Oriented Programming) to separate the responsibilities of encryption, steganography in images, steganography in audio, and handling inputs and outputs.
- Implement additional functionalities, such as the ability to choose between different concealment techniques.
- Conduct integration tests to ensure that the modules work well together.

6.5.4. Stage 4: Creation of the CLI

Objective: Create the command-line interface to facilitate interaction with the program.

Tasks:

- Implement the command-line interface.
- Integrate the interface with the previously developed modules.
- Ensure that users can only choose possible execution flows.

6.5.5. Stage 5: Tool Testing

Objective: Conduct tests on the tool to ensure reliability and discover the limitations of the system.

Tasks:

- Carry out tests to check the reliability of the tool by using it in real use cases.
- Conduct tests to study the limitations of the tool.

6.5.6. Stage 6: Documentation

Objective: Document the architecture of the tool as well as the conclusions we have reached.

Tasks:

- Create detailed documentation on the code, functionalities, usage of the tool, its installation, the technologies used, and also those discarded.
- Prepare a presentation to showcase the project and its final results.

Conclusions and Future Work

6.6. Conclusions of the tool

After completing the tool and reflecting on each chapter of this document, it can be concluded that a modular and functional command-line tool has been created, which aims to combine cryptography and steganography in order to communicate secretly over the public network. To justify this conclusion, we will recall the objectives defined in section 1.2:

- **Creation of a CLI (Command-line interface):** Class diagram and explanation in chapter 3.
- **Implementation of algorithms for the concealment and revelation of text in images as well as images in audio:** Explained in chapter 3 where we can see that we have implemented algorithms based on the use of LSB (Least Significant Bit), which involves embedding text in images and transforming it to make it invisible, as well as algorithms based on the transformation of files into acoustic waves.
- **Processing of acoustic waves in computing:** Explained in section 2.1.2.
- **Use of secret key cryptography:** The fundamentals are explained in section 2.3 along with its implementation in chapter 3.

6.7. Future Work

Looking ahead and with the objectivity that everything can be improved, there are several aspects that would undoubtedly enhance the tool:

- Implementation of error-correcting code to prevent possible man-in-the-middle attacks and noise in the transmission channel.
- Implementation of hiding algorithms for more file formats. This would provide more options for hiding.

- Possibility to add new encryptions or hiders through simple plugins for any user.
- Design of a GUI (Graphical User Interface) to be more user-friendly for the general public who may not have the knowledge to use the tool via the CLI.
- Implementation of a decoder for SSTV, in order to avoid opening QSSTV and make the process automated.
- Implementation of a specific OCR (Optical Character Recognition) engine for a custom font and adapted to our context: black letters on a white background.
- A more powerful analysis of *ocultadorText*, generating a much larger dataset using PySpark and a cloud service like GCP (Google Cloud Console).
- Implementation of domain transformation-based techniques.
- Use of public cryptography for exchanging secret keys.

Contribuciones Personales

Alberto Martín Oruña

Durante el desarrollo del proyecto, realicé diversas contribuciones en todas las etapas del plan de trabajo, subsección 1.3.

En lo relativo a la etapa 1, subapartado 1.3.1, mi labor fue la búsqueda de información sobre los orígenes de la esteganografía, fundamentos y bases de la criptografía de clave secreta y de clave pública, además de la transmisión de datos mediante SSTV. Para todo ello, pasé gran parte del tiempo que le dediqué a esta etapa leyendo y entendiendo Carracedo Gallardo (2004).

Posteriormente, me ocupé de la redacción de esos temas en el capítulo 2, así como de la elaboración de las figuras y tablas de esas secciones. También redacté la motivación del proyecto, subsección 6.3.

En cuanto a la etapa 2, subapartado 1.3.2, implementé en Python un programa lineal que utilizaba los algoritmos del *ocultadorText*, ocultación mediante la incrustación del texto en la imagen y desocultación usando motores OCR; y *ocultadorSSTV*, paso de la imagen a onda acústica y su decodificación. Para ello, investigué sobre aplicaciones que implementaran decodificadores para SSTV y su integración en el código.

En la etapa 3, subapartado 1.3.3, separé en módulos todo el código existente hasta el momento, diseñando la estructura del proyecto, clases, secuencias y métodos.

Además, añadí nuevos módulos para la lectura de los argumentos y control de los flujos principales; también añadí las constantes, archivo de configuración, creación y actualización de la caché.

En esta misma etapa, tras modularizar el código, realizamos otra revisión para ver posibles mejoras, donde automaticé el uso de QSSTV en la herramienta, solucionando un problema de variables de entorno que impedía abrir ventanas emergentes,

se resolvió limpiando el entorno antes de iniciar QSSTV, también arreglé un error en la transformadas de las imágenes ya que con imágenes de dimensiones impares fallaba.

En la etapa 4, subapartado 1.3.4, ayudé a mi compañero con lo pertinente de la etapa 3, diseño de directorios de ese módulo, clases y secuencias.

En la etapa 5, subapartado 1.3.5, desarrollé pruebas utilizando las ya existentes de mi compañero para comprobar que la herramienta funcionaba correctamente con imágenes de dimensiones impares, verificando todas las combinaciones posibles de paridad, además de probar con imágenes codificadas en RGBA, ya que hasta ese momento solo habíamos probado con RGB.

En esta etapa, además, llevé a cabo todas las pruebas documentadas en el capítulo 4, así como su documentación, incluyendo generación de *datasets*, análisis de estos, generación de archivos portadores y difusión a través de la red pública.

Durante las pruebas de campo, me fijé que había un problema con las imágenes codificadas en RGBA, ya que ciertas imágenes al ser convertidas a RGB, mostraban líneas de colores en sus antiguos píxeles transparentes, lo que indicaba que la imagen no era una imagen “normal”. Lo arreglé estableciendo todos los píxeles con transparencia máxima a negro y sin transparencia antes de la ocultación.

En la etapa 6, subapartado 1.3.6, dediqué mucho tiempo en revisar erratas, redacté la sección de conclusiones de la herramienta, subsección 6.1, algunos puntos del trabajo futuro, 6.2 y los apéndices A, incluyendo el desarrollo y comprobación del correcto funcionamiento del código referido en ese apéndice; B y C.

Para el desarrollo del apéndice A elaboré un *script* en bash que se encarga de verificar, resolver o avisar todas las dependencias del proyecto. Además de comprobar su validez en una máquina virtual con sistema operativo Debian, esta máquina solo tenía instalados los programas propios de su instalación en el equipo.

Para el desarrollo del apéndice B, recopilé en este documento la información sobre las llamadas a la herramienta que habíamos usado para validarla a lo largo del proyecto, además de realizar capturas de pantalla de la herramienta para mostrar esas mismas funcionalidades pero utilizando la CLI.

Rodrigo Gallego Marín

Durante el desarrollo del presente proyecto, he llevado a cabo una gran variedad de tareas que abarcan desde la investigación previa y la planificación hasta la implementación, documentación y validación de los resultados. Estas contribuciones han sido muy importantes en la evolución del trabajo y en la consolidación de una herramienta funcional, modular y adaptable.

En las etapas iniciales, una de mis primeras tareas fue pasar la introducción del proyecto de un documento de Google Docs al formato LaTeX. Este paso permitió empezar a hacer la memoria en un formato más profesional y limpio.

En la fase de investigación, comencé buscando información sobre los métodos de codificación y decodificación tanto de imágenes como de audio, ya que vamos a trabajar sobre este tipo de archivos. También fui responsable de redactar esta información en la correspondiente sección del estado de la cuestión.

A continuación, me centré especialmente en comprender a fondo los principios de la esteganografía digital, desde qué medios hay hasta las técnicas existentes. Dentro de las distintas técnicas de ocultación, le puse especial énfasis al uso de bits menos significativos (LSB) y su aplicación tanto en imágenes como en audio. Paralelamente, investigué qué formatos de audio permiten ocultar imágenes, determinando sus ventajas y limitaciones en términos de compresión.

Durante la implementación, realicé pruebas iniciales del código, centrándome en comprobar la viabilidad de ocultar información utilizando LSB. Experimenté con distintas variantes: el uso de marcadores dentro del flujo de bits, que no resultó viable, y la inclusión de cabeceras que indican el tamaño del contenido oculto. Esto permitió establecer una estructura más robusta para la recuperación de datos.

Posteriormente, desarrollé y adapté los algoritmos de ocultación y recuperación basados en LSB, tanto para imágenes como para audio. Complementariamente, implementé un mecanismo de cifrado para proteger el contenido antes de ocultarlo, aumentando la seguridad ante análisis forenses o accesos no autorizados. Esto llevó a un script funcional con el cual se pudo ocultar y desocultar un mensaje de manera correcta.

Una de las mejoras que introduce fue la creación de una transformada para aplicar sobre la imagen original antes del ocultado, cuyo objetivo es desordenar los píxeles antes de aplicar la ocultación, haciendo más difícil detectar visualmente patrones que puedan delatar la presencia de información. Esta transformación se diseñó de forma reversible, de modo que tras la recuperación se puede reordenar la imagen original sin pérdida.

Para validar el sistema, realicé pruebas de transmisión mediante SSTV (Slow

Scan Television), un protocolo que convierte imágenes en sonido. En estas pruebas oculté archivos cifrados dentro de imágenes mediante LSB y los transmití en forma de audio, evaluando la pérdida de información al recibir de vuelta la imagen. Estos ensayos fueron clave para analizar que no es viable transmitir una imagen con un mensaje oculto por LSB ya que, al recuperarla, no hay forma de recuperar el mensaje debido al ruido, lo que nos llevó a plantearnos otras alternativas.

Además, desarrollé un script de tests automáticos para verificar la integridad del sistema tras cada modificación del código. Esto permitió detectar errores de forma temprana y agilizó el proceso de desarrollo ya que, tras cualquier cambio, se verificaba que se siguiera ocultando y desocultando correctamente.

Otro bloque importante de mi contribución fue la creación de una interfaz por línea de comandos (CLI). Esta interfaz, compuesta por varias clases, permite al usuario ejecutar todas las funcionalidades del programa desde la terminal, de manera sencilla e intuitiva.

Además de crear la CLI, la integré para que funcionara de manera correcta con la herramienta, lo que requería de pruebas para verificar que todo siguiera funcionando. Además, también tuve que verificar que no se pudieran introducir valores erróneos que condujeran a comportamientos extraños del programa.

También me encargué de la documentación interna del código, con comentarios explicativos de cada clase y de cada función para que quien quiera pueda entender la herramienta y qué hace cada clase. Esto permitirá en un futuro a la gente modificar el código con más facilidad.

En cuanto a la arquitectura del sistema, investigué sobre la creación de diagramas tanto de clases como de secuencia debido al gran apoyo visual que serían en la descripción de la arquitectura del sistema. Después de informarme sobre cómo hacerlos, desarrollé tanto los diagramas de clases como los diagramas de secuencia, los diagramas de clases me sirvieron para poder explicar mejor cómo está el código estructurado y las diferentes funcionalidades de los ocultadores y cifradores que hay, los diagramas de secuencia me sirvieron más para reflejar el flujo de ejecución del programa tanto a la hora de ocultar como a la de desocultar.

Después de terminar la herramienta, realice una comparación con herramientas existentes en el ámbito de la esteganografía, destacando las ventajas de nuestra solución sobre el resto.

También, realicé una revisión de las tecnologías descartadas, incluyendo las ideas que en un principio parecían viables, pero que por motivos de rendimiento, compatibilidad o dificultad de integración no fueron utilizadas. Documentar estas decisiones resultó fundamental para dejar constancia del proceso de toma de decisiones técnicas y poder dar respuesta a su no integración en la herramienta.

Como cierre, contribuí activamente a la definición de futuras líneas de trabajo, proponiendo mejoras como la implementación del código corrector de errores para más seguridad, contemplación de más formatos sobre los que ocultar datos, un sistema de plugins para que sea más sencillo añadir nuevos métodos de cifrado o ocultación y un diseño GUI para ser más amigable con los usuarios menos experimentados con la CLI.

Bibliografía

CARRACEDO GALLARDO, J. *Seguridad en redes telemáticas*. McGraw Hill, Madrid, 2004.

IGLESIAS, P. F. Esteganografía, el arte de ocultar información sensible. *Pablo y Glesias - Blog*, 2015. Disponible en <https://www.pabloyglesias.com/mundohacker-esteganografia/> (último acceso, Marzo, 2025).

INTERNATIONAL AMATEUR RADIO UNION (IARU) REGION 1. Ethics and Operating Procedures for the Radio Amateur. <https://www.iaru-r1.org/wp-content/uploads/2021/01/Eth-operating-IARU-ENGLISH-version3-2010-amended-2021.pdf>, 2021. Último acceso: 8 de junio de 2025.

KEMP, S. 5 billones de usuarios en social media. 2024. Disponible en: <https://wearesocial.com/es/blog/2024/01/digital-2024-5-billones-de-usuarios-en-social-media/>. Último acceso: 13 de marzo de 2025.

MICROSOFT CORPORATION AND IBM CORPORATION. Multimedia programming interface and data specifications 1.0. <https://www.loc.gov/preservation/digital/formats/fdd/fdd000001.shtml>, 1991. Especificación oficial del formato RIFF/WAVE desarrollada por Microsoft e IBM. Referenciado por la Library of Congress. Último acceso: 22 de abril de 2025.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST). Fips 197: Advanced encryption standard (aes). [Https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf](https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf), 2001. Último acceso: 10 de abril de 2025.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST). Fips pub 180-4: Secure hash standard (shs). <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>, 2015. Último acceso: 10 de abril de 2025.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST). Sp 800-56b revision 2: Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. [Https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br2.pdf](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br2.pdf), 2019. Último acceso: 10 de abril de 2025.

RENAU TORRES, V. *Compresión y restauración de imágenes por técnicas no lineales*. Tesis Doctoral, Universitat de València, València, 2011. Dirección: Pep Mulet Mestre y Francesc Aràndiga Llaudes.

RODRÍGUEZ ROCA, F. J. *La esteganografía como técnica para garantizar la seguridad y confidencialidad en la transmisión de datos*. Trabajo final de carrera, Universitat Oberta de Catalunya, 2009. Disponible en: <http://hdl.handle.net/10609/929>. Último acceso: 12 de marzo de 2025.

RUSSON, M.-A. Isis and al-qaeda terrorists using ebay, reddit and porn to send coded messages says mossad. *IBTimes UK*, 2015. Disponible en: <https://www.ibtimes.co.uk/isis-al-qaeda-terrorists-using-ebay-reddit-porn-send-coded-messages-says-mossad-1490130>. Último acceso: 18 de marzo de 2025.

SALAS, A. *Los hombres que susurran a las máquinas: [hackers, espías e intrusos en tu ordenador]*. Espasa Calpe, Madrid, 2015.

STALLINGS, W. *Cryptography and network security: Principles and practice*. 2022. 8^a Edición, Prentice Hall.

Apéndice A

Guía de instalación

En este apéndice se mostrará dónde descargar la aplicación y se explicará su instalación en Linux.

Se ha comprobado su funcionamiento en Debian, en la primera iso descargable en:

```
https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/
```

Para descargar la aplicación es necesario clonar el repositorio de la herramienta, <https://github.com/Alberto12x/PITEA.git>. Se puede hacer de la siguiente manera:

```
git clone https://github.com/Alberto12x/PITEA.git
```

o descargando el archivo .zip del repositorio de *GithHub*.

Una vez clonado, es necesario entrar en el repositorio y ejecutar el archivo `install.sh`, dándole permisos de ejecución si fuera necesario:

```
cd TFG-PITEA-main  
chmod +x install.sh  
sudo ./install.sh
```

Una vez instalado, se llama a la herramienta por terminal escribiendo:

```
pitea
```

Nota importante

Es de tener en cuenta que si se usa una *shell* con el AUTO_CD activado como zsh si se escribe en la terminal `pitea` desde la raíz del proyecto no se ejecutará la herramienta sino que se ejecutará `cd pitea`.

Apéndice B

Manual de uso

En este apéndice se explicará brevemente cómo usar las distintas funcionalidades de la herramienta, aportando imágenes de la terminal o de la herramienta para facilitar su entendimiento.

B.1. Ejemplos de Ejecución

El archivo “script_ejecucion.py” y “launch.py” es ejecutable de la manera estándar de Linux, *./archivo* o con *python3*.

El mensaje de ayuda generado por la herramienta al ejecutar “script_ejecucion.py” con la opción -h o –help es:

```
Usage: script_ejecucion.py [OPTIONS] COMMAND [ARGS] ...

Herramienta para la ocultacion y desocultacion de datos en imagen
→ y
audio.

Options:
-h, --help Show this message and exit.

Commands:
desocultar Ejecuta la acción de desocultación.
ocultar     Ejecuta la acción de ocultación.
```

y ejecutando cada subcomando con la misma opción:

```
> ./script_ejecucion.py ocultar -h
Usage: script_ejecucion.py ocultar [OPTIONS]

Ejecuta la acción de ocultación.
```

```

Options:
--modo-cifrado [aes|none]           Modo de cifrado a utilizar.
--modo-cifrado-imagen [lsb|text]      Modo de ocultacion a usar en la
                                         → imagen.
--modo-cifrado-audio [lsb|sstv]       Modo de ocultacion especifico
                                         → para audio.
-v, --verbose                         Modo verbose , muestra mensajes
                                         → del
                                         flujo.
-i, --input PATH                      Archivo de datos a ocultar
                                         → [required]
--input_imagen PATH                   Archivo de imagen requerido para
                                         → ciertos
                                         modo de ocultacion de imagen.
--input_audio PATH                    Archivo de audio requerido para
                                         → ciertos
                                         modo de ocultacion de audio.
-o, --output PATH                    Nombre del archivo de salida.
--contraseña TEXT                   Contraseña para cifrado o
                                         → descifrado.
-h, --help                            Show this message and exit.

```

```

> ./script_ejecucion.py desocultar -h
Usage: script_ejecucion.py desocultar [OPTIONS]

```

Ejecuta la acción de desocultación.

```

Options:
--modo-cifrado [aes|none]           Modo de cifrado a utilizar.
--modo-cifrado-imagen [lsb|text|none]
                                         Modo de ocultacion usado en la
                                         → imagen.
--modo-cifrado-audio [lsb|sstv|none]
                                         Modo de ocultacion usado en el
                                         → audio.
-v, --verbose                         Modo verbose , muestra mensajes
                                         → del
                                         flujo.
-s, --streaming                       Modo streaming, captura el audio
                                         → sstv en
                                         streaming en vez de pasarle un
                                         → audio.

```

--input_audio PATH	Archivo de audio de entrada.
--input_imagen PATH	Archivo de imagen de entrada.
-i, --input_text PATH	Archivo de texto de entrada.
-o, --output PATH	Archivo de texto de salida.
--contraseña TEXT	Contraseña para descifrado.
-h, --help	Show this message and exit.

Para ejecutar la CLI *python3 launch.py*, este comando es lo que se ejecuta tras haber ejecutado el “install.sh” y ejecutar *pitea* como se muestra en el apéndice A.

B.1.1. Uso de algoritmos de ocultación basados en LSB

B.1.1.1. Llamada por terminal

```
python3 script_ejecucion.py ocultar \
--modo-cifrado aes \
--modo-cifrado-imagen lsb \
--modo-cifrado-audio lsb \
-i archivos_prueba/prueba.txt \
--input_imagen archivos_prueba/imagen.png \
--input_audio archivos_prueba/audio.wav \
-o ./archivos_prueba/audio_salida.wav \
--contraseña qwqwqw \
-v
```

```
python3 script_ejecucion.py desocultar \
--modo-cifrado aes \
--modo-cifrado-imagen lsb \
--modo-cifrado-audio lsb \
--input_audio ./archivos_prueba/audio_salida.wav \
-o ./archivos_prueba/datos_desocultos.txt \
--contraseña qwqwqw \
-v
```

B.1.1.2. Llamada en la CLI



Figura B.1: Ejemplo de uso de la CLI para ocultación con LSB. Captura de pantalla de la herramienta.



Figura B.2: Ejemplo de uso de la CLI para desocultación con LSB. Captura de pantalla de la herramienta.

B.1.2. SSTV

```
python3 script_ejecucion.py ocultar \
--modo-cifrado aes \
--modo-cifrado-imagen text \
--modo-cifrado-audio sstv \
-i archivos_prueba/prueba.txt \
-o ./archivos_prueba/sstv.wav \
--contraseña qwqwqw \
-v
```



Figura B.3: Ejemplo de uso de la CLI para ocultación con SSTV. Captura de pantalla de la herramienta.

Para la desocultación de esta subsección es necesario configurar QSSTV. Es necesario marcar la casilla del *autoslant*.

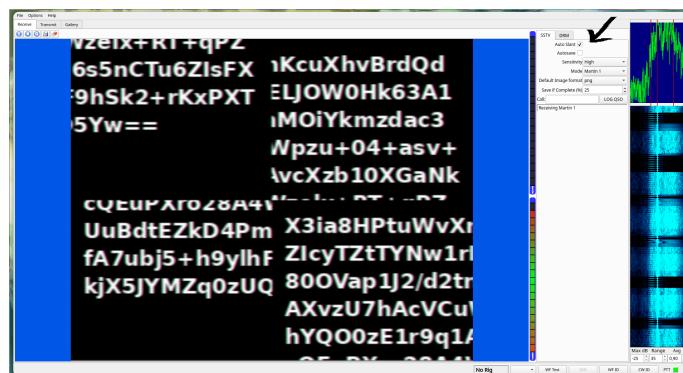


Figura B.4: Marcar *autoslant*. Captura de pantalla de QSSTV.

B.1.2.1. Desocultación desde imagen

```
python3 script_ejecucion.py desocultar \
--modo-cifrado aes \
--modo-cifrado-imagen text \
--modo-cifrado-audio none \
--input_imagen ./archivos_prueba/imagen_salida_sstv.png \
-o ./archivos_prueba/datos_desocultos_text_sstv.txt \
--contraseña qwqwqw \
-v
```



Figura B.5: Ejemplo de uso de la CLI para desocultación con SSTV usando una imagen ya decodificada de SSTV. Captura de pantalla de la herramienta.

B.1.2.2. Desocultación desde audio

Es necesario configurar QSSTV para que reciba el audio desde un archivo.

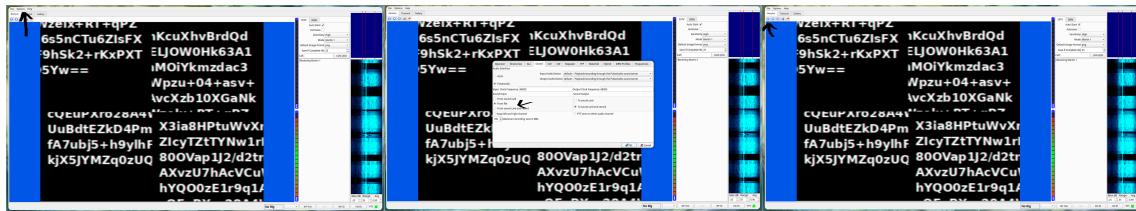


Figura B.6: Configuración QSSTV para lectura desde archivos. Capturas de pantalla de QSSTV.

```
python3 script_ejecucion.py desocultar \
--modo-cifrado aes \
--modo-cifrado-imagen text \
--modo-cifrado-audio sstv \
--input_audio ./archivos_prueba/sstv.wav \
-o ./archivos_prueba/datos_desocultos_text_sstv.txt \
--contraseña qwqwqw \
-v
```



Figura B.7: Ejemplo de uso de la CLI para desocultación con SSTV. Captura de pantalla de la herramienta.

B.1.2.3. Desocultación desde audio streaming

Es necesario configurar QSSTV para que reciba el audio desde la tarjeta de sonido del dispositivo.

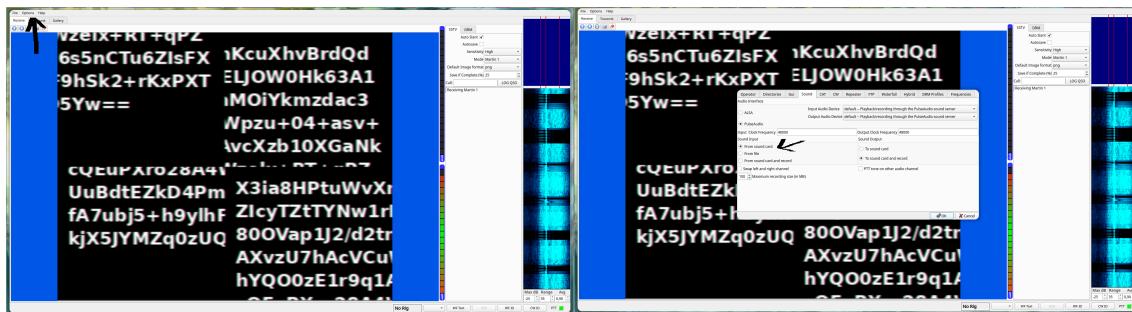


Figura B.8: Configuración QSSTV para lectura desde la tarjeta de sonido. Capturas de pantalla de QSSTV.

```
python3 script_ejecucion.py desocultar \
--modo-cifrado aes \
--modo-cifrado-imagen text \
--modo-cifrado-audio sstv \
-o ./archivos_prueba/datos_desocultos_text_sstv.txt \
--contraseña qwqwq \
-v \
-s
```



Figura B.9: Ejemplo de uso de la CLI para desocultación con SSTV en modo *streaming*. Captura de pantalla de la herramienta.

B.1.2.4. Desocultación desde Base64

```
python3 script_ejecucion.py desocultar \
--modo-cifrado aes \
--modo-cifrado-imagen none \
--modo-cifrado-audio none \
--input_text ./archivos_prueba/base64.txt \
-o ./archivos_prueba/datos_desocultos_text_sstv.txt \
--contraseña qwqwq \
-v
```



Figura B.10: Ejemplo de uso de la CLI para desocultación con SSTV iniciando con un archivo que contiene información en base64. Captura de pantalla de la herramienta.

B.2. Documentación del archivo de configuración, “configuracion.toml”

```
[persistente]
# Contador utilizado para distinguir la caché entre varias ejecuciones en
→ un mismo minuto
contador_cache = 0

# Fecha de la última ejecución, usada para saber si es necesario utilizar
→ el contador de caché
ult_fecha = "20-02-2025_21:22"

[Ajustes_sstv]
# Modo de transmisión SSTV seleccionado, especifica el tipo de imagen que
→ se usará
modo_sstv = "MartinM1"

# Muestras por segundo, define la calidad de la transmisión en términos de
→ frecuencia
samples_per_sec = 48000

# Número de bits por muestra, determina la resolución de las muestras de
→ audio
bits = 16

[Ajustes_ocultador_imagen_text]
# Tamaño de la fuente en píxeles, utilizado para ajustar el texto sobre
→ las imágenes
tamanio_fuente = 10
```

```
# Ancho máximo de las imágenes, para asegurarse de que las imágenes no
→ sean demasiado anchas
anchura_maxima = 800

# Ruta relativa a la fuente utilizada para el texto (debe estar en el
→ directorio adecuado)
ruta_fuente = "../fuentes/ocraregular.ttf"

# Configuración para usar la GPU, puede ser un valor booleano (True o
→ False)
gpu = "True"
```

Descripción de las secciones

- **persistente**: Guarda parámetros que se mantienen entre ejecuciones, como la caché y la fecha de la última ejecución.
- **Ajustes_sstv**: Define la configuración de transmisión SSTV, incluyendo modo, muestras por segundo y resolución.
- **Ajustes_ocultador_imagen_text**: Contiene opciones para superponer texto en imágenes, como tamaño de fuente, ancho máximo, fuente y uso de GPU.

Apéndice C

Información extra del análisis de *OcultadorText*, 4.3

C.1. Fuentes utilizadas

Estas fuentes han sido elegidas porque son específicas para motores OCR como *Ocr-a-std* o por ser ampliamente conocidas, como Arial.

Total de fuentes: 12

- Ocr-a-std
- Ocrbregular
- Ocraregular
- Ocr-a-extended
- Arial
- NotoSansMono-Regular
- DejaVuSans-Bold
- LiberationSans-Regular
- Lato-Regula
- Quicksand-Regula
- FreeSans
- NimbusSans-Regular

C.2. Frases utilizadas

- *hardpan diluvianism 756819 quintary unsaintlike unmapped depside isothiocyanoo cran drachm*
- *horrormongering Tetraodon cerrero warbled greaser microbiosis thickety stereomerism truthtelling porcate orthophyre*
- *gobline otogenic abdominoposterior undervaulted swivetty 943958 inframaxillary sipage matted sectoral galvanography conspectus Pholcidae thioantimonate Peristeromorphae smoodger unmuffled semiurn scarlety*
- *epicrystalline screwstock*
- *ladleful escapist malcontented effeminacy dangle*

- *hydrogel*
- *fissury tugui oscillating bahar piririgua untainted penninite 586399 Sumak Shulamite transmaterial urataemia barracan paddlewood ridgling carroterlevelman*
- *694326 vaticinal somniloquence decalvation journal plasticism slantingways comfiture*
- *Ino bicyclic baublery Taranchi edeodynna mignonne innutritious squaw familiarness 463097*
- *grogginess*