Informe del desafío 1

# Hugo Esteban Barrero García – Juan Angel Omaña Montañez

# Universidad de Antioquia

# Informática 2

# Aníbal José Guerra Soler – Augusto Salazar

# Medellín, Colombia

Abril, 2025

12 de abril de 2025

1. Objetivo General

El objetivo del Desafío 1 es implementar un sistema que permita la reconstrucción de una imagen encriptada mediante una secuencia de transformaciones aplicadas sobre sus píxeles. Estas transformaciones incluyen operaciones de rotación de bits y XOR, y deben ser revertidas utilizando información contenida en archivos de enmascaramiento (.txt).

2. Descripción del problema

Se entrega una imagen I\_D.bmp que ha sido encriptada a partir de una imagen original mediante una serie de transformaciones aplicadas paso a paso. Junto a esta imagen se entrega:

Una imagen de referencia I\_M.bmp.

Una imagen de máscara M.bmp.

Archivos de enmascaramiento M0.txt, M1.txt, ..., Mn.txt que contienen una semilla y valores esperados tras aplicar una transformación.

El sistema debe identificar la transformación aplicada en cada paso (ya sea una rotación de bits o una operación XOR) y revertirla para llegar nuevamente a la imagen original.

3. Solución propuesta

Se desarrolló un programa en C++ usando Qt para cargar y procesar las imágenes. El flujo general es:

Cargar las tres imágenes: I\_D.bmp, I\_M.bmp y M.bmp.

Leer paso a paso los archivos .txt, comenzando desde el último.

En cada paso, aplicar todas las posibles transformaciones sobre la imagen actual y validar si se obtiene el resultado esperado comparando con los valores del archivo .txt.

Si la transformación es válida, continuar con el siguiente archivo hasta llegar a M0.txt.

Exportar las imágenes intermedias (P1, P2, ...) y la final Imagen\_Original.bmp.

4. Desarrollo de la Solución

Funciones Implementadas

cargarPixeles(...): carga los píxeles de una imagen BMP y los convierte a formato RGB888. Cada píxel es representado por tres bytes (R, G, B).

exportarImagen(...): guarda una imagen en formato BMP a partir de un arreglo RGB.

cargarSemillaYEnmascaramiento(...): carga desde un archivo .txt la semilla de inicio y los valores RGB esperados tras el enmascaramiento.

rotarBitsIzquierda(...) y rotarBitsDerecha(...): realizan rotaciones circulares de bits en cada byte del arreglo que representa la imagen. La rotación se realiza por byte (canal de color), usando combinaciones de desplazamientos y el operador OR para simular el efecto circular.

aplicarXOR(...): aplica la operación XOR a nivel de bytes entre dos imágenes del mismo tamaño. Esto permite revertir una transformación previamente aplicada con XOR si se conoce la imagen usada como máscara.

validarEnmascaramiento(...): verifica si una imagen transformada más la máscara da como resultado los valores almacenados en el archivo .txt, desde una posición inicial definida por la semilla.

identificarTransformaciones(...): intenta identificar cuál fue la transformación aplicada (XOR o rotaciones de 1 a 7 bits, en ambas direcciones). Cada intento se valida mediante la función de enmascaramiento.

reconstruirImagenDesdeCaso(...): función general que coordina la desencriptación completa desde una carpeta y un número de pasos. Carga las imágenes, recorre los .txt, identifica transformaciones y guarda imágenes intermedias.

liberar(...): función de utilidad para liberar memoria dinámica de forma segura y evitar punteros colgantes.

**Algoritmo de Ingeniería Inversa**

Se intenta primero aplicar XOR con la imagen I\_M. Si el resultado cumple con el enmascaramiento, se considera válida.

Si falla, se prueban rotaciones de 1 a 7 bits a la izquierda y derecha.

La primera transformación válida se adopta y se guarda como imagen desencriptada parcial.

Este proceso se repite hasta llegar al paso 0.

**Algoritmo de Decisión – Pseudocódigo**

A continuación, se resume en pseudocódigo la lógica del sistema para identificar la transformación correcta en cada paso del proceso:

Para cada paso i desde el último .txt hacia el 0:

Cargar la imagen actual (ID)

Cargar datos del archivo Mi.txt (semilla, valores esperados)

Probar: “imagenDesencriptada” = ID XOR IM

Si la validación con la máscara y los datos del .txt es correcta:

Guardar la imagen y continuar con el siguiente paso

Sino:

Para bits desde 1 a 7:

Probar rotación a la izquierda

Si válida, guardar imagen y continuar con el siguiente paso

Probar rotación a la derecha

Si válida, guardar imagen y continuar con el siguiente paso

Si ninguna transformación fue válida:

Detener proceso con error

**Consideraciones Técnicas.**

Manipulación de archivos BMP: La lectura y escritura de imágenes se realizó mediante QImage en formato RGB888. Cada píxel está compuesto por tres bytes consecutivos (R, G y B), y no incluye canal alfa, lo que simplifica el procesamiento.

Arreglos dinámicos y memoria: Todos los datos de imágenes y archivos .txt se almacenaron en arreglos dinámicos (unsigned char\* y unsigned int\*), lo cual permite operar con imágenes de cualquier dimensión. Se implementaron funciones liberar(...) para evitar fugas de memoria y punteros colgantes.

memcpy y copias temporales: El uso de memcpy fue fundamental para operar sobre copias de imágenes al probar transformaciones. Esto permite mantener intactos los datos originales mientras se prueba si una rotación o un XOR es válida.

Reversibilidad de las operaciones:

XOR: Aprovecha la propiedad de reversibilidad (A ^ B ^ B = A) para desencriptar datos de forma exacta.

Rotaciones de bits: Se implementaron rotaciones circulares, que permiten recuperar la información original sin pérdida de datos. Ambas direcciones (izquierda y derecha) fueron consideradas, y se limitó el número de bits a un máximo de 7, ya que una rotación de 8 bits no cambia el valor.

Validación con enmascaramiento: Cada transformación se valida sumando la imagen transformada con la máscara, desde una posición dada por la semilla, y comparando con los valores almacenados en el archivo .txt. Esta verificación garantiza que solo las transformaciones correctas sean aceptadas.

Conversión de rutas: Se usó ruta.toStdString().c\_str() para abrir archivos con ifstream, convirtiendo de QString a const char\*.

Representación de datos: Para imprimir valores de tipo unsigned char, se usó conversión explícita a int, evitando así interpretaciones ASCII incorrectas.

Uso de QString::arg(): Facilitó la generación de rutas dinámicas para cargar archivos M0.txt, M1.txt, ..., automatizando la iteración sin necesidad de concatenaciones complejas.

Modularidad: el diseño permite que el mismo sistema funcione con distintos conjuntos de imágenes.

5. Resultados

Se realizaron pruebas exitosas con los casos entregados:

Caso1: se reconstruyó correctamente en 2 pasos.

Caso2: se reconstruyó correctamente en 6 pasos.

Se generaron archivos intermedios para cada paso (Posible\_P1.bmp, Posible\_P2.bmp, ...) y el archivo final Imagen\_Original.bmp. La validación se realizó con éxito en todos los pasos.

6. Conclusiones

Se logró implementar un sistema robusto capaz de identificar y revertir transformaciones en imágenes encriptadas.

El uso de pruebas por validación con valores esperados permitió determinar con certeza qué transformación fue aplicada.

El manejo seguro de memoria y la organización modular del código favorecieron la claridad y el correcto funcionamiento.

El sistema podría ser extendido para soportar nuevos tipos de transformaciones o validaciones.

Se comprendió en profundidad el funcionamiento de operaciones binarias, rotaciones de bits, y cómo se puede aplicar ingeniería inversa a nivel de bytes.

También se evidenció la importancia de comprender el tipo y estructura de los archivos tratados. Por ejemplo, el uso de archivos BMP en formato RGB888 implica trabajar directamente con arreglos dinámicos que almacenan de forma lineal los bytes que representan los canales R, G y B. Esta comprensión permitió manipular adecuadamente los datos, evitando errores como interpretaciones incorrectas de los canales o errores por lecturas excesivas de memoria.

Finalmente, se resalta la modularidad del sistema: cada componente fue diseñado de forma independiente y reutilizable, lo que permite adaptar fácilmente el programa a nuevos casos, nuevas máscaras o incluso nuevas transformaciones, con mínimos cambios al código base.

7. Bibliografía y referencias

* Documentación oficial de Qt. Disponible en: <https://doc.qt.io>
* Apuntes de clase de la asignatura Informática II.
* Archivo entregado “Desafío I v1.pdf” con instrucciones oficiales del reto.
* Explicaciones técnicas desarrolladas durante el trabajo práctico.
* Discusiones y ayudas recibidas a través del canal del curso.
* Asistencia generada con ChatGPT (OpenAI).
* Consultas complementarias realizadas con DeepSeek.
* Apoyo puntual de Gemini (Google).