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

**1. Introducción**

En el marco del curso de Informática II, se plantea un desafío orientado a la aplicación de conceptos avanzados de programación en C++ con Qt, manipulación de archivos binarios e ingeniería inversa sobre imágenes. El objetivo es recuperar una imagen original a partir de una versión encriptada que ha sido transformada mediante operaciones a nivel de bits y validaciones auxiliares conocidas como enmascaramientos.

Este proceso se asemeja a escenarios reales en ciberseguridad y procesamiento digital de imágenes, donde la protección de datos visuales se logra mediante transformaciones sucesivas que ocultan su estructura original. A través de la comprensión profunda de operaciones binarias, rotaciones, manejo de memoria dinámica y validaciones por sumas RGB, se construyó una solución robusta y verificable.

**2. Objetivo General**

El proyecto consiste en desarrollar un programa para reconstruir una imagen BMP original a partir de una versión alterada mediante operaciones de manipulación de bits y enmascaramiento. La imagen entregada presenta un patrón de ruido estático (ruido Gaussiano), y el desafío radica en revertir las transformaciones aplicadas utilizando archivos de rastreo (`.txt`) que contienen datos críticos para la reconstrucción.

**3. Análisis del Problema**

**3.1 Naturaleza de las Transformaciones**

Operaciones a nivel de bits: se aplican varios tipos de transformaciones para encriptar la imagen, pero teniendo en cuenta la pérdida de datos y la información brindada para poder hacer el proceso de recuperar la imagen original, únicamente se pueden hacer operaciones XOR y rotaciones a nivel de bits, si se quiere volver de forma certera realmente a una imagen. Las otras transformaciones serían irreversibles, por lo tanto, no se tendrán en cuenta para la solución del desafío.

Las transformaciones consideradas en este desafío corresponden a:

* **Operaciones XOR:** Esta operación lógica permite encriptar y desencriptar datos de manera reversible, siempre y cuando se mantenga constante el patrón de máscara (IM). Su propiedad clave es que aplicar XOR dos veces con el mismo valor revierte el resultado: A ^ B ^ B = A.
* **Rotaciones circulares de bits:** Aplicadas sobre cada byte (R, G, B de cada píxel), las rotaciones permiten mover bits hacia la izquierda o derecha, reintegrando los bits que “se caen” por un extremo al otro lado del byte. A diferencia de desplazamientos simples, las rotaciones son completamente reversibles.
* **Enmascaramiento:** Después de cada transformación, se aplicó una operación auxiliar para validar el paso. Esta consiste en sumar píxel a píxel una región de la imagen transformada con una máscara más pequeña. La suma comienza en una posición aleatoria (s) y los resultados se almacenan en un archivo .txt. Estos datos permiten verificar si una transformación es válida.

**3.2 Desafíos Clave**

* **Falta de información directa sobre la secuencia de transformaciones.**
* **Preservación de datos durante pruebas y validaciones.**
* **Reversibilidad garantizada:** No todas las operaciones a nivel de bits son reversibles, por tanto, el sistema solo puede considerar aquellas que aseguren integridad de los datos.
* **Comparación con archivos de enmascaramiento:** Es la única herramienta de validación disponible.

**4. Descripción del Problema**

Se entrega una imagen final I\_D.bmp encriptada a partir de una original desconocida. Esta fue sometida a una serie de transformaciones que incluyen XOR con una imagen aleatoria I\_M.bmp, y rotaciones circulares. A lo largo del proceso, se aplicaron validaciones mediante una máscara M.bmp y se almacenaron los resultados en archivos M0.txt a Mn.txt.

Cada archivo .txt incluye:

* Un desplazamiento s indicando la posición de inicio.
* Una lista de valores RGB resultantes de la suma de una región de la imagen transformada y la máscara.

El reto es recorrer estos archivos en sentido inverso y, en cada paso, aplicar todas las transformaciones posibles (XOR o rotaciones), validar los resultados con los datos del archivo correspondiente, y continuar hasta llegar a la imagen original.

**5. Metodología Propuesta**

**5.1. Enfoque General**

1. **Carga de datos:** Lectura de imágenes BMP (I\_D, I\_M, M) y archivos .txt.
2. **Prueba de transformaciones:** Se prueba primero XOR con I\_M. Si falla, se prueban rotaciones de 1 a 7 bits en ambas direcciones.
3. **Validación de enmascaramiento:** Se suma la imagen resultante con la máscara desde la posición s, y se compara con los datos del archivo .txt.
4. **Aprobación:** Si alguna transformación es válida, se guarda la imagen intermedia y se avanza al siguiente paso.

**5.2. Algoritmo de Decisión (Pseudocódigo)**

Para cada paso i desde el último .txt hasta M0.txt:

Cargar imagen ID

Leer semilla y valores RGB de Mi.txt

Probar: ID XOR IM

Si válida:

Guardar y continuar

Sino:

Para bits de 1 a 7:

Probar rotación izquierda y validar

Probar rotación derecha y validar

Si ninguna es válida:

Detener proceso con error

**6. Arquitectura del Sistema**

El sistema se diseñó modularmente, dividiendo la lógica de transformación, validación y flujo principal. Se utilizaron arreglos dinámicos y punteros para manipular directamente los bytes de las imágenes, maximizando eficiencia y control de memoria.

* **Uso de memcpy:** Permite trabajar sobre copias temporales al probar transformaciones, asegurando que la imagen original no se altere durante el proceso.
* **Conversión de tipos:** Se convierte unsigned char a int para evitar interpretaciones ASCII durante comparaciones.
* **Conversión de rutas Qt:** ruta.toStdString().c\_str() permite usar rutas Qt (QString) con bibliotecas estándar como ifstream.
* **Uso de operador ternario:** Para asignar nombres de salida condicionalmente:
* QString nombreSalida = (i == 0) ? carpeta + "/Imagen\_Original.bmp" : carpeta + QString("/Posible\_P%1.bmp").arg(i);  
  **QString::arg():** Facilita la creación dinámica de rutas para los archivos .txt.

**7. Desarrollo de la Solución**

**Funciones Implementadas**

|  |  |  |  |
| --- | --- | --- | --- |
| **Función** | **Parámetros** | **Salida** | **Descripción** |
| **cargarPixeles** | **QString rutaEntrada, int &ancho, int &alto** | **unsigned char\* – Buffer dinámico de bytes RGB888 – nullptr en caso de error** | **Lee un archivo BMP desde rutaEntrada, convierte internamente la imagen a formato RGB888 (24 bpp) y copia línea a línea sus píxeles en un buffer lineal de tamaño ancho × alto × 3. Devuelve el buffer o nullptr si la carga falla.** |
| **exportarImagen** | **unsigned char\* datosPixeles, int ancho, int alto, QString rutaSalida** | **bool – true si la imagen se guardó correctamente – false en caso de error** | **Construye un QImage de formato RGB888 usando el buffer datosPixeles (ancho × alto × 3 bytes) y lo exporta como BMP a la ruta rutaSalida. Imprime un mensaje de error en consola si la operación falla.** |
| **cargarSemillaYEnmascaramiento** | **const char\* rutaArchivo, int &semilla, int &num\_pixeles** | **unsigned int\* – Arreglo dinámico de valores RGB (3 × num\_pixeles) – nullptr en caso de error** | **Abre el archivo de texto rutaArchivo, lee la primera línea como semilla, luego cuenta cuántos tripletes RGB contiene el resto, vuelve a abrir el archivo y almacena dichos tripletes en un arreglo de enteros. Devuelve el arreglo y actualiza semilla y num\_pixeles.** |
| **rotarBitsDerecha** | **unsigned char\* imagen, int bits, int tamaño** | **– (modifica imagen in situ)** | **Para cada uno de los tamaño bytes en imagen, rota sus bits a la derecha en bits posiciones, combinando desplazamiento lógico derecho e izquierdo para preservar la información que "cae" por un lado.** |
| **rotarBitsIzquierda** | **unsigned char\* imagen, int bits, int tamaño** | **– (modifica imagen in situ)** | **Igual que rotarBitsDerecha, pero desplazando cada byte bits posiciones a la izquierda, y reinyectando los bits que “salen” por el otro lado.** |
| **aplicarXOR** | **unsigned char\* imgA, unsigned char\* imgB, int tamaño** | **– (modifica imgA in situ)** | **Recorre los buffers de longitud tamaño y aplica la operación bit a bit imgA[i] = imgA[i] ^ imgB[i], dejando en imgA el resultado de la combinación XOR.** |
| **validarEnmascaramiento** | **unsigned char\* imagen, unsigned char\* mascara, unsigned int\* valorestxt, int semilla, int num\_pixeles** | **bool – true si todos los valores coinciden – false al detectar la primera discrepancia** | **Para cada índice k en [0, num\_pixeles\*3), suma imagen[k + semilla] + mascara[k] y compara con valorestxt[k]. Si alguna suma difiere, retorna false; si todas coinciden, retorna true.** |
| **identificarTransformaciones** | **unsigned char\* imgEncriptada, unsigned char\* imgIM, unsigned char\* mascara, unsigned int\* datostxt, int semilla, int num\_pixeles, int tamaño** | **unsigned char\* – Buffer con imagen desencriptada – nullptr si no se identifica transformación válida** | **(1) Crea una copia de imgEncriptada y aplica XOR con imgIM; si validarEnmascaramiento aprueba, retorna esa copia. (2) Si falla, para cada bits de 1 a 7 prueba rotaciones izquierda y derecha en nuevas copias; valida cada una y retorna la primera que sea correcta.** |
| **liberar (unsigned char)** | **unsigned char\*& ptr** | **– (libera y deja ptr = nullptr)** | **Libera la memoria dinámica asociada al buffer de tipo unsigned char[] apuntado por ptr y luego pone ptr a nullptr para evitar punteros colgantes.** |
| **liberar (unsigned int)** | **unsigned int\*& ptr** | **– (libera y deja ptr = nullptr)** | **Misma funcionalidad que la versión para unsigned char, pero para arreglos de unsigned int[].** |
| **reconstruirImagenDesdeCaso** | **QString carpeta, int ultimotxt** | **– (genera en disco múltiples BMP de salida)** | **Carga las imágenes base I\_M.bmp, M.bmp e I\_D.bmp desde carpeta. A partir de ultimotxt hasta 0, para cada M<i>.txt lee semilla y máscara, aplica identificarTransformaciones, exporta el resultado a Posible\_P<i>.bmp (o Imagen\_Original.bmp si i=0) e imprime progreso.** |

**Pruebas Específicas**

* **Rotación de bits:** Se cargó I\_O.bmp, se aplicó rotación derecha de 3 bits y se guardó como I\_Rotada.bmp. Luego, se aplicó rotación inversa de 3 bits a la izquierda y se obtuvo I\_Restaurada.bmp. Comparación visual y binaria confirmó que I\_Restaurada coincide con I\_O, validando la reversibilidad de la función.
* **Operación XOR:** Se usaron I\_O.bmp y I\_M.bmp. Resultado de XOR: I\_XOR.bmp. Al aplicar XOR nuevamente con I\_M, se recuperó I\_O.bmp, validando reversibilidad.
* **Problemas corregidos:** Al aplicar XOR directamente sobre el arreglo principal, los datos se modificaban, impidiendo pruebas subsecuentes. Se implementó copia temporal copiaXOR, mejorando la precisión de validaciones.

**8. Resultados**

**8.1. Reconstrucción de Casos**

* **Caso 1:** Reconstrucción en 2 pasos. Se identificaron una operación XOR y una rotación. Las imágenes Posible\_P1.bmp y Imagen\_Original.bmp fueron generadas y validadas con éxito. La comparación RGB con los archivos M1.txt y M0.txt fue exacta.
* **Caso 2:** Reconstrucción en 6 pasos. Se probaron combinaciones de XOR y rotaciones. El sistema identificó la transformación válida en cada etapa y generó las imágenes Posible\_P6.bmp a Imagen\_Original.bmp. Todas las transformaciones fueron validadas con éxito contra los archivos .txt correspondientes.

**8.2. Validaciones Complementarias**

* **Exactitud visual:** Comparación con imágenes originales (donde se tenía referencia) mostró restauraciones exactas.
* **Consistencia del flujo:** No se presentaron errores de memoria ni incoherencias en nombres de archivos exportados.
* **Generalización:** El sistema se probó con nuevos conjuntos de imágenes estructurados bajo el mismo formato, mostrando compatibilidad sin cambios de código.

**9. Conclusiones**

* El sistema implementado cumple con los objetivos del Desafío 1.
* Las operaciones XOR y rotaciones circulares fueron suficientes para recuperar la imagen original en todos los casos.
* La validación por enmascaramiento fue una estrategia efectiva para confirmar cada transformación.
* El diseño modular, el uso de memoria dinámica y la integración con Qt permitieron una solución escalable, mantenible y eficiente.

### **10. 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).
* Repositorio del proyecto: <https://github.com/Hugo-24/Desafio-1>.