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. Introducción

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.

2. Análisis del problema.

2.1. 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.

- Enmascaramiento: lo tomamos como un registro, una captura o pasos de la encriptación que se le hizo a la imagen. Se toman regiones específicas de la imagen transformada y usando la máscara, cuyos efectos quedan registrados en los archivos `.txt`.

Realmente es una operación auxiliar utilizada para verificar si una transformación aplicada a una imagen fue realizada correctamente. Consiste en tomar una imagen transformada (denominada ID) y combinarla con una máscara de color M, la cual es de menor tamaño. Para ello, se selecciona una posición de inicio aleatoria dentro de la imagen (s), conocida como desplazamiento, y a partir de esa posición se suman los valores de los canales RGB de la imagen ID con los correspondientes de la máscara M. El resultado de esas sumas se almacena en un archivo .txt, que contiene el desplazamiento utilizado y los valores RGB obtenidos. Es importante destacar que esta operación no modifica la imagen original, sino que sirve como mecanismo de validación para comprobar el orden correcto de las transformaciones.

2.2. Desafíos Clave.

- Pérdida de información: Ciertas operaciones, si no se revierten correctamente, destruyen datos de manera irreversible.

- Orden de operaciones: la secuencia de transformaciones aplicadas es desconocida, lo que requiere probar distintas combinaciones para conocer lo aplicado en la imagen.

- Validación: la ausencia de la imagen original obliga a depender de imágenes intermedias y coherencia con los archivos txt para verificar resultados.

3. Metodología Propuesta

3.1. Enfoque General

1. Carga de datos: Lectura de la imagen transformada, la máscara y los archivos de rastreo.  (Funciones brindadas por los profesores, debemos entender muy bien cómo funcionan).

2. Reversión de enmascaramientos: aplicación de operaciones aritméticas para volver a los valores de las regiones modificadas, usando los datos de los `.txt`.

3. Reversión de transformaciones: Aplicación de operaciones inversas (XOR, o rotaciones) en un orden hipotético basado en ingeniería inversa.

4. Validación: comparación de resultados intermedios con patrones esperados, como los brindados en los casos 1 y 2.

3.2. Estrategias para minimizar pérdidas

- Priorizar operaciones reversibles: Uso de rotaciones circulares en lugar de desplazamientos lógicos para preservar bits.

- Manejo de valores extremos: Aplicación del módulo de 256 en operaciones aritméticas para garantizar rangos válidos de píxeles.

4. Progreso y Logros

4.1. Avances Actuales

- Carga de recursos: Implementación básica para leer imágenes BMP y archivos de rastreo, almacenando datos en estructuras dinámicas.

- Pruebas conceptuales: Validación preliminar de operaciones de bits en entornos controlados, confirmando la reversibilidad de XOR y rotaciones.

4.2. Dificultades Relevantes

- Incertidumbre en el orden de operaciones: La falta de información sobre la secuencia original complica la reversión precisa.

- Gestión de memoria: Optimización del uso de recursos para manejar imágenes de alta resolución sin comprometer el rendimiento. Buen uso de la memoria dinámica.

5. Próximas Etapas

1. Integración de módulos: Combinar la reversión de enmascaramientos con operaciones de bits.

2. Pruebas con datos reales: Usar el Caso 1 para validar resultados intermedios y ajustar la secuencia de operaciones.

3. Documentación técnica: Elaborar un manual de procedimientos para garantizar claridad en el proceso de reconstrucción.

**Función de rotaciones de bits**

La función se encarga de realizar una **rotación circular de bits** sobre un arreglo dinámico de tipo unsigned char, el cual almacena todos los bytes de una imagen en formato BMP de 24 bits (cada píxel está compuesto por 3 bytes: RGB).

Este arreglo contiene valores entre 0 y 255 (1 byte por canal), y su tamaño es igual a ancho × alto × 3, ya que cada píxel tiene 3 componentes RGB. La función también recibe un número entero que indica **cuántos bits se deben rotar** y la **cantidad total de bytes** sobre los que debe operar (el tamaño del arreglo).

La rotación se realiza **a nivel de byte**, es decir, se toma cada valor individual (cada canal de color) y se reorganizan sus 8 bits en forma circular. Si se rota hacia la derecha, los bits que "salen" por la derecha **entran de nuevo por la izquierda**; y si se rota hacia la izquierda, los bits que "salen" por la izquierda **entran por la derecha**.

Este proceso se logra combinando dos desplazamientos:

-Uno en la dirección deseada (<< o >>), para mover los bits.

-Otro en la dirección contraria, para recuperar los bits que se salieron.

-Finalmente, se unen los dos resultados con el operador OR (|) para formar el nuevo byte rotado.

**Función de operación XOR**

Esta función aplica la operación **XOR a nivel de bits** entre dos arreglos dinámicos de tipo unsigned char. Cada arreglo contiene todos los bytes de una imagen en formato BMP de 24 bits (RGB), por lo tanto, cada píxel está compuesto por 3 bytes: uno para el canal Rojo, otro para el Verde y otro para el Azul.

La función recibe dos arreglos del mismo tamaño:

-El primero representa la imagen que se desea transformar (encriptar o desencriptar).

-El segundo representa la imagen IM que se usó como **máscara XOR**.

También recibe el tamaño total del arreglo, que corresponde a la cantidad total de bytes a operar (ancho × alto × 3).

La función recorre los arreglos y realiza la operación A[i] = A[i] ^ B[i] byte por byte. El resultado queda almacenado en el primer arreglo (la imagen original se transforma directamente).

**Pruebas de rotación de bits**

Para verificar el correcto funcionamiento de la rotación de bits, se cargó una imagen BMP en formato RGB de 24 bits (I\_O.bmp) y se aplicó una **rotación circular de 3 bits a la derecha** sobre cada uno de los bytes de la imagen. Esta rotación afecta individualmente a cada canal de color (R, G, B) de todos los píxeles.

Después de aplicar la rotación, se guardó el resultado en un nuevo archivo (I\_Rotada.bmp) para su inspección visual. Posteriormente, se aplicó una **rotación inversa (3 bits a la izquierda)** a la misma imagen rotada, con el fin de revertir el cambio. El resultado final fue guardado como I\_Restaurada.bmp.

Al comparar visualmente la imagen restaurada con la imagen original, se verificó que ambas eran idénticas, lo que confirma que la función de rotación es **reversible y no pierde información**, cumpliendo así con uno de los requisitos fundamentales del desafío.

**Pruebas de operación XOR**

Para probar la operación XOR, se cargaron dos imágenes:

-La imagen original I\_O.bmp.

-La imagen I\_M.bmp, que funciona como **máscara XOR** y tiene las mismas dimensiones que la original.

Se aplicó la operación XOR **byte por byte** entre estas dos imágenes, generando una nueva imagen I\_XOR.bmp. Esta imagen representa una versión "encriptada", donde cada canal de color ha sido modificado según la imagen IM.

A continuación, se aplicó nuevamente la operación XOR entre I\_XOR.bmp y la misma imagen I\_M.bmp. Debido a la propiedad de reversibilidad del operador XOR (A ^ B ^ B = A), el resultado obtenido fue una imagen **idéntica a la original**, la cual se guardó como I\_XORRestaurada.bmp.

La validación visual confirmó que la imagen restaurada era igual a I\_O.bmp, demostrando que la función XOR implementada es correcta, **reversible**, y adecuada para el propósito del proyecto.