**Informe De Análisis Y Diseño Del Desafió I**

Manuela Galeano Chica y Carlos Andrés Buelvas

Facultad de ingeniería, Universidad de Antioquia, Medellín

Informática II

Aníbal Guerra Soler y Augusto Salazar

12 de abril de 2025

Imagen que contiene Patrón de fondo

Descripción generada automáticamente

* **1. Introducción**

La empresa Informa2 nos solicita diseñar un algoritmo capaz de reconstruir una imagen distorsionada mediante transformaciones a nivel de bits. Para lograrlo, se nos proporcionan:

* Imágenes BMP distorsionadas.
* Archivos .txt con los resultados de enmascaramientos.
* Una imagen de ruido (IM).
* Una máscara (M) de tamaño menor.

Este informe presenta el análisis de las transformaciones aplicables, su reversibilidad y la propuesta de estructura para revertir la imagen a su estado original.

* **2. Análisis de las transformaciones**

**2.1 Desplazamiento de bits**

Cuando se desplazan los bits de un byte hacia la derecha o izquierda, sabemos que el máximo rango de movimiento puede ser de 8 bits nos dimos de cuenta que los bits "empujados" se pierden. Esta operación **no es reversible** porque no existe información para recuperar lo perdido.

**Ejemplo de desplazamiento a la derecha (4 bits):**

Original (1 byte): 10111101 → Desplazado 8 bits a la derecha: 00000000

Si se invierte 5 bits a la izquierda: 00000000 → 00000000 ❌ **No se recupera el original.**

**Conclusión:** Esta operación **pierde información**, por lo tanto, **no se usará** en la decodificación. si esta información se pudiera almacenar en algún lugar, deberían ser específicos en un archivo aparte para posteriormente incorporarlos a la imagen original, sin embargo, este no es el caso.

**2.2 Rotación de bits**

La rotación mueve los bits dentro del byte sin perder ninguno, Esta transformación se les aplica a todos los bytes RGB de la imagen. Por ejemplo, si se rota a la derecha 3 bits, los 3 últimos bits se mueven al principio.

**Ejemplo:**

Original (1 byte): 10111101

Rotado derecha (3 bits): 11101101→ Rotado izquierda (3bits): 10111101 ✔

**Conclusión:** Esta operación **es reversible** y segura para encriptación o desencriptación.

**2.3 XOR con imagen de ruido**

|  |  |  |
| --- | --- | --- |
| *A* | *B* | *A* XOR *B* |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

La operación XOR compara dos bytes (uno de la imagen original y otro de la imagen de ruido). Al aplicar XOR dos veces con el mismo patrón se recupera el original:

M/O: byte de la imagen original

M/D: byte de la imagen ruido

R1: M/O ^ M/D

R2: R1 ^ M/D = M/O

**Conclusión:** Es una operación **segura y reversible**. Se puede aplicar varias veces, siempre que se conozca el patrón (M/D).

**2.4 Conclusiones de las transformaciones**

Tras examinar detenidamente cada tipo de transformación, concluimos que únicamente dos de ellas permiten modificar la imagen sin pérdida de información: la rotación y la operación **XOR**. En cambio, el desplazamiento de bits elimina parte de la información original, lo que impide reconstruir la imagen tal como era. Por tanto, esta última puede considerarse una transformación engañosa, ya que imposibilita revertir el proceso.

* **Transformaciones válidas**: XOR y rotaciones.
* **Transformaciones descartadas**: desplazamientos de bits.
* **3. Análisis del enmascaramiento**

La máscara es un fragmento de imagen que se suma (elemento a elemento) a una sección de la imagen transformada. Esa sección se determina con una **semilla aleatoria**.

La máscara puede aplicarse en cada etapa del proceso de transformación. Si el número de pasos es reducido y la máscara es de tamaño pequeño, su efecto sobre la imagen original será leve si no se elimina. Sin embargo, cuando se realizan múltiples transformaciones sucesivas, la máscara puede acumular alteraciones que afectan significativamente la calidad de la imagen al intentar recuperarla. En otras palabras, entre más se transforme sin eliminar la máscara, mayor será la distorsión acumulada.

**Enmascaramiento:**

S(k) = ID(k + s) + M(k)

Esto genera archivos .txt con resultados que se pueden comparar en cada paso de transformación.

**Importancia:** Si se quita la máscara correctamente (restando), se puede verificar si el resultado intermedio coincide con los archivos de pista.

* **4. Estructura del algoritmo**

**Proceso de encriptación original (ejemplo):**

Paso 1: M\_O → TA → M → TXT1 → M\_D1

Paso 2: M\_D1 → TA → M → TXT2 → M\_D2

Paso 3: M\_D2 → TA → M\_D3 (final)

**Entradas disponibles:**

* Imagen final distorsionada (M\_D3)
* Archivos .txt (TXT1, TXT2)
* Imagen de ruido
* Máscaras M

**Meta:** Aplicar las transformaciones e ir comparando los resultados intermedios con los archivos TXT para descubrir el orden correcto. Además, este proceso implica realizar ingeniería inversa, ya que debemos partir del paso 3 y retroceder hasta llegar al paso 1.

* **5. Funciones necesarias en C++ (Qt)**

| **Función** | **Descripción** |
| --- | --- |
| xorImagen(...) | Aplica XOR entre dos arreglos de imágenes |
| rotarBitsDerecha(...) | Rota n bits hacia la derecha por byte (por canal RGB) |
| rotarBitsIzquierda(...) | Rota n bits hacia la izquierda (operación inversa) |
| loadPixels(...) | Carga imagen BMP como arreglo de bytes RGB |
| exportImage(...) | Guarda imagen modificada como archivo BMP |
| loadSeedMasking(...) | Carga semilla y datos RGB de un archivo .txt |

* **6. Conclusión**
* Las transformaciones **rotación** y **XOR** son las únicas reversibles.
* El desplazamiento no se usará porque pierde información.
* El algoritmo final deberá iterar combinaciones de transformaciones posibles y verificar si los resultados parciales coinciden con los datos en los archivos .txt.

Con este análisis, estamos listos para comenzar la codificación y validación del proceso de desencriptado…