Entrega 1: Primer Reto de Informática II

1st Daniela Escobar Velandia
Dpto. de Ingeniería Electrónica y Telecomunicaciones
Universidad de Antioquia
Medellín, Colombia
daniela.escobarv@udea.edu.co

2nd Yonathan López Mejía
Dpto. de Ingeniería Electrónica y Telecomunicaciones
Universidad de Antioquia
Medellin, Colombia
harley.lopez@udea.edu.co

***Abstract*—Este documento presenta el análisis y diseño de una solución para un problema de reconstrucción de imágenes basado en transformaciones a nivel de bits. Consta en que a partir de una imagen alterada y varios archivos auxiliares, se busca identificar las operaciones aplicadas para revertirlas en el orden correcto y poder recuperar la imagen original. En esta fase se detallan las condiciones del reto, los criterios de comparación entre imágenes y la estrategia propuesta para llevar a cabo un proceso de ingeniería inversa. El informe incluyendo algoritmos, tablas y explicaciones del diseño planteado.**

***Keywords*—Operaciones a nivel de bit, rotación de bits, operación XOR, enmascaramiento, distancia de Hamming.**

I. Introducción

Se propone un desafío enfocado en el análisis, diseño e implementación de una solución para la reconstrucción de una imagen digital que ha sido modificada mediante transformaciones a nivel de bits. Estas transformaciones incluyen operaciones como *rotaciones*, *desplazamientos* y *XOR*, aplicadas en un orden desconocido.

El problema se puede plantear como un caso de ingeniería inversa, en el cual, a partir de una imagen final y varios archivos generados durante el proceso, se debe determinar qué operaciones fueron aplicadas, en qué orden, y revertirlas para recuperar la imagen original. Se requiere entender el enmascaramiento de datos a nivel de píxeles, ya que después de cada transformación se aplicó una máscara sobre una región específica de la imagen.

Se definen las tareas necesarias, se explican los algoritmos propuestos y se identifican los retos a tener en cuenta en la implementación.

II. Propuesta de solución

A. Condiciones y restricciones

Para la realización del reto se conoce de forma directa o derivada lo siguiente:

- El número de operaciones realizadas sobre la imagen original.
- La imagen resultado de la última operación: $\texttt{P}_\texttt{N}\texttt{.bmp}$.
- Los archivos de enmascaramiento: $M_{n-1}\texttt{.txt}$, que incluyen la semilla aplicada y el resultado.
- La imagen máscara: `M.bmp`.
- La imagen de entropía: $\texttt{I}_\texttt{M}\texttt{.bmp}$.
- El número máximo de operaciones aplicadas: XOR, desplazamientos o rotaciones a izquierda o derecha de hasta 8 bits.
- La operación de enmascaramiento $s(k) = I_D(k+s) + M(k)$ con s una semilla aleatoria y positiva, I_D la imagen transformada al momento del enmascaramiento y M la imagen máscara.

Se desconoce:

- El orden de las operaciones aplicadas.
- La imagen objetivo.
- Las imagenes resultado de aplicar las operaciones intermedias hasta llegar a $\texttt{P}_\texttt{N}\texttt{.bmp}$.

B. Operaciones que mantienen la integridad de los datos

En cualquier sistema de cifrado, se debe cumplir que cualquier conjunto de N datos debe ser íntegramente recuperable. En el caso del reto, no se implementan estrategias durante el cifrado que garanticen la integridad de los datos, como sumas de verificación o códigos de detección de errores. Por lo tanto, la integridad de los datos recuperados estará garantizada únicamente si las operaciones a nivel de bits son completamente reversibles, esto es: al aplicar su inversa se recuperan todos y cada uno de los bits originales.

Un ejemplo de operación reversible es la operación `XOR`. Por ejemplo:

```
1011 XOR 0011 = 1000
1000 XOR 0011 = 1011
```

Como se observa, al aplicar la misma operación `XOR` con el mismo operando (`0011`) sobre el resultado, se recuperan los 4 bits originales (`1011`).

Otro ejemplo de operación reversible es la rotación de bits. Supongamos una rotación hacia la izquierda (`ROL`) de un valor de 4 bits:

```
1011 ROL 2 = 1110
1110 ROR 2 = 1011
```

En este caso, al rotar `1011` dos bits a la izquierda, se obtiene `1110`. Luego, al aplicar una rotación de dos bits a la derecha (`ROR`) sobre `1110`, se recupera el valor original `1011`. Por lo tanto, la operación es reversible siempre que se conozcan la dirección y la cantidad de desplazamiento.

A diferencia de las operaciones anteriores, el desplazamiento de bits (`shift`) no siempre es reversible, ya que puede provocar pérdida de información. Por ejemplo, consideremos un desplazamiento lógico a la derecha de un valor de 4 bits:

`1011 SHR 2 = 0010`

En este caso, al desplazar `1011` dos bits a la derecha, se obtiene `0010`. Sin embargo, no es posible recuperar los dos bits más significativos que fueron descartados (los bits que estaban originalmente en las posiciones altas). Por lo tanto, esta operación en general no es reversible pues usualmente al realizar desplazamientos a izquierda o derecha se perderán bits 1s no recuperables. La Tabla I resume las operaciones.

Tabla I
REVERSIBILIDAD DE LAS OPERACIONES OPERACIONES BIT A BIT

Operación	¿Es reversible?
XOR	Sí
Rotación de bits (ROL/ROR)	Sí
Desplazamiento (SHL/SHR)	No

C. Criterio de verosimilitud

1) Diferencia absoluta: Una imagen, al igual que un vector en álgebra lineal, será igual a otra si coinciden en todos sus píxeles. En el contexto del reto, es necesario definir un criterio que permita determinar si dos imágenes (o regiones de una imagen) pueden considerarse equivalentes.

Una forma común de abordar este problema consiste en realizar una resta pixel a pixel: dos imágenes se consideran iguales si la suma total de las diferencias es cero. Sin embargo, este método resulta insuficiente cuando las operaciones aplicadas durante el procesamiento no garantizan la integridad de los datos, ya que pequeñas alteraciones pueden impedir una coincidencia exacta incluso si la estructura general se conserva.

Algorithm 1: Comparación pixel a pixel entre dos porciones de imagen

Input: ImagenA, ImagenB
Output: SonIguales (booleano)

```
if dimensiones(ImagenA) ≠ dimensiones(ImagenB) then
    return falso
suma_diferencias ← 0
for cada píxel (i, j) en las imágenes do
    diferencia ← |ImagenA[i][j] − ImagenB[i][j]| ;
    suma_diferencias ← suma_diferencias + diferencia ;
if suma_diferencias = 0 then
    return verdadero
else
    return falso
```

2) Uso de la distancia de Hamming como criterio de similitud: Para afrontar el problema derivado de la pérdida de datos, se considera que dos bytes son más similares entre sí si su **distancia de Hamming** es mínima.

Dada dos secuencias de bits de igual longitud, la **distancia de Hamming** se define como el número de posiciones en las que los bits correspondientes son diferentes. Por ejemplo:

`1011` y `1100` $\Rightarrow$ distancia de Hamming = 2

En el contexto del reto, esta aproximación es más robusta frente a la pérdida de información que una simple resta, ya que tolera diferencias mínimas a nivel de bit. Sin embargo, pueden ocurrir *colisiones*, es decir, múltiples tramas con la misma distancia mínima respecto a una trama de referencia. Por ejemplo:

- `1111` vs `0111` $\Rightarrow$ distancia = 1
- `1111` vs `1110` $\Rightarrow$ distancia = 1

En este caso, ¿cuál de las dos tramas es más similar a `1111`? Este tipo de colisiones puede ocurrir en el reto. El criterio de desempate adoptado será seleccionar aquella operación que realice la mayor cantidad de desplazamientos manteniendo la distancia de Hamming mínima.

Desde el punto de vista computacional, el cálculo de la distancia de Hamming es más costoso que una resta directa, ya que requiere contar el número de bits en 1 tras una operación XOR. El algoritmo de fuerza bruta para este conteo tiene una complejidad de $O(n)$, siendo n el número de bits de la trama.

Nosotros proponemos una optimización mediante el **algoritmo de Brian Kernighan**, que reduce la complejidad a $O(k)$, siendo k el número de bits en 1 del resultado, lo cual representa una mejora considerable en eficiencia cuando los valores son dispersos.

Algorithm 2: Conteo de bits en 1 usando el algoritmo de Brian Kernighan

Input: n: entero sin signo (representa la trama de bits)
Output: count: número de bits en 1

```
count ← 0 ;
while n ≠ 0 do
    n ← n AND (n − 1) ;
    count ← count + 1 ;
return count
```

D. Propuesta de ingeniería inversa

Aunque no se conocen las imágenes resultantes de las transformaciones intermedias, sí se dispone de los archivos generados durante el proceso de enmascaramiento.

Con esto en mente, se propone el siguiente procedimiento para cada imagen transformada:

Sea $\texttt{P}_{\texttt{N}}\texttt{.bmp}$ la última imagen obtenida tras aplicar una secuencia de transformaciones. A partir del archivo $\texttt{M}_{\texttt{N-1}}\texttt{.txt}$, se extraen la semilla y el resultado del enmascaramiento $s(k)$. Conociendo $s(k)$ y $m(k)$, es posible —utilizando la fórmula de enmascaramiento— obtener $I_D(k+s)$, es decir, un fragmento de la imagen $\texttt{P}_{\texttt{N-1}}\texttt{.bmp}$ de la cual se derivó $\texttt{P}_{\texttt{N}}\texttt{.bmp}$.

A continuación, se realizan operaciones de bit sobre un segmento de `P_N.bmp` que comienza en la posición $P(s)$ y tiene la misma longitud que `M.bmp`. Tras aplicar cada operación candidata, se calcula la distancia de Hamming total entre el resultado obtenido y el esperado. La operación seleccionada será aquella que produzca la menor distancia de Hamming. (Tabla II)

Una vez identificada la operación más probable, se aplica su inversa sobre `P_N.bmp` para obtener `P_N-1.bmp` en su totalidad. Este proceso se repite iterativamente hasta recuperar la imagen original.

Tabla II
OPERACIONES Y SUS INVERSAS

Operación	**Operación inversa**
XOR con clave I_M	XOR con clave I_M
Rotación a la izquierda (`ROL` n)	Rotación a la derecha (`ROR` n)
Rotación a la derecha (`ROR` n)	Rotación a la izquierda (`ROL` n)
Desplazamiento a la izquierda (`SHL` n)	`SHR` n (parcialmente reversible)
Desplazamiento a la derecha (`SHR` n)	`SHL` n (no reversible)

E. Algoritmo planteado

El diagrama del algoritmo planteado es el siguiente:

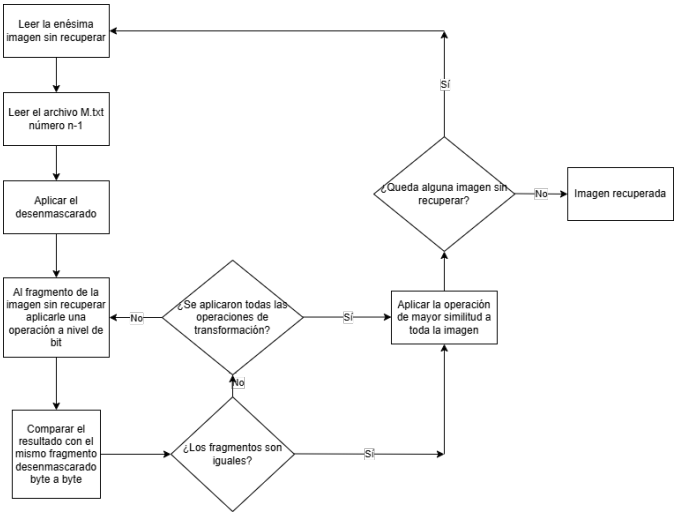

Fig. 1. Algoritmo planteado.