Informe desarrollo desafío uno

Integrantes:

Hawer Hernandez
Andres Felipe Henao

Docente:
Anibal Jose Guerra Soler
Informatica II

Universidad de Antioquia
Abril 2025

INTRODUCCIÓN

En la era digital, la seguridad de la información visual como imágenes médicas, documentos escaneados o fotografías sensibles se ha convertido en una necesidad crítica. La creciente facilidad para copiar, distribuir y alterar imágenes digitales plantea serios desafíos en cuanto a la confidencialidad, integridad y trazabilidad de estos datos. Este trabajo aborda el problema de la protección y recuperación de imágenes digitales mediante la aplicación de técnicas básicas de encriptación reversible a nivel de bits y enmascaramiento visual tipo esteganográfico. Estas operaciones permiten simular un esquema de cifrado simétrico, donde el conocimiento de ciertos parámetros (como una semilla aleatoria o los bits desplazados) permite revertir completamente las modificaciones y recuperar la imagen original. Además, el sistema incorpora un mecanismo heurístico que detecta y revierte automáticamente las transformaciones aplicadas, lo que lo convierte en una base experimental útil para estudiar la resiliencia de datos visuales frente a distorsiones controladas y su potencial uso en cifrado ligero, esteganografía o análisis forense digital.

ANALISIS DEL PROBLEMA

¿Cómo proteger imágenes digitales de forma reversible, simple y sin perder información, utilizando transformaciones ligeras aplicables a nivel de bits?

Esto implica varios retos técnicos:

- Preservar la integridad de los datos: Cada byte de una imagen tiene un significado visual, por lo tanto, cualquier modificación debe ser completamente reversible.
- Aumentar la confidencialidad: Si una imagen es interceptada, su contenido debe resultar ilegible o alterado sin la clave o la semilla correcta.
- Evitar pérdida de información: Operaciones como desplazamiento pueden eliminar bits; se requiere un sistema auxiliar para conservar esa información.
- No depender de librerías criptográficas: Se busca una solución educativa y de bajo nivel para explorar los principios de encriptación visual.
- Automatizar la recuperación: El sistema debe ser capaz de reconocer y revertir las transformaciones aplicadas, incluso si se desconoce el orden original.

Consideraciones sobre alternativa de solución propuesta.

Se consideraron algunas alternativas para esta solución como son:

- Simplicidad y control total ya que nos basamos en operaciones bit a bit implementadas manualmente, sin estructuras complejas ni criptografía avanzada.
- Usamos transformaciones reversibles y cada una de las transformaciones implementadas (XOR, rotación, desplazamiento) es reversible si se guarda o reproduce la información auxiliar correctamente (semilla, bits desplazados).

- Enmascaramiento como capa adicional, el enmascaramiento agrega una capa esteganográfica simple que oculta fragmentos de la imagen cifrada dentro de una máscara (otra imagen BMP).
- Seguridad básica pero frágil ya que este sistema NO es criptográficamente seguro por sí solo.

PROCESO

1 ANÁLISIS DEL PROBLEMA Y DISEÑO DEL SISTEMA

Proteger imagenes mediante transformaciones reversibles, definimos la encriptación ligera, reversibilidad e identificamos restricciones.

2 MANIPULACION DE IMAGEN BPM

Implementar funcion para cargar imagen, exportar y definir estructura como arreglo lineal RGB.

3 BITS A BITS PARA TRANSFORMACIONES

Implementar algoritmo XOR, rotacioón de bits, desplazamiento y registro de perdida, funciones inversas para cada transformación.

4 GENERACIÓN Y USO DE IMÁGENES PSEUDOALEATORIA

- Definir una función para generar una imagen aleatoria desde una semilla, garantizar la reproducibilidad de la imagen aleatoria para revertir XOR.

5 IMPLEMENTACIÓN DEL ENMASCARAMIENTO

Implementar función de enmascaramiento visual, Definir estrategia de mezcla entre la imagen y la máscara (suma de píxeles) , registrar información auxiliar en archivos .txt

PROCESO

6 IMPLEMENTACIÓN DEL DESENMASCARAMIENTO

Implementar función de recuperación de fragmentos ocultos, Leer y aplicar archivos de rastreo M*.txt para revertir el enmascaramiento.

7 DETECCIÓN AUTOMÁTICA DE TRANSFORMACIONES

- Implementar sistema heurístico para detectar la transformación aplicada en cada etapa, Automatizar la aplicación de transformaciones inversas según detección.

8 VALIDACIÓN Y EVALUACIÓN

Comparar imagen recuperada con la original, contar errores (bytes diferentes) entre ambas, mostrar resultado y transformaciones detectadas en consola.

9 PRUEBAS Y EXPERIMENTACIÓN

- Diseñar y ejecutar experimentos con distintas combinaciones de transformaciones,medir el impacto visual y lógico de cada técnica.

ALGORITMOS IMPLEMENTADOS

Algoritmos de Transformación Bit a Bit

1. Opera_xor(pixelData, generateI_m, size)

- Función: Aplica XOR entre cada byte del arreglo de píxeles y una imagen pseudoaleatoria.
- Efecto: Ofusca completamente los valores, reversible con la misma imagen generada con la misma semilla.

2. Opera_rota(pixelData, size, n)

- Función: Rota cada byte n bits a la derecha.
- Efecto: Cambia la estructura binaria sin eliminar datos, completamente reversible.

3. Opera_despla(pixelData, size, n, etapa)

- Función: Desplaza a la derecha n bits cada byte, y guarda los bits eliminados en archivo.
- Efecto: Pérdida de datos parcial si no se guarda el registro de bits.

Algoritmos inversos.

4. Opera_xor_inverse(pixelData, generateI_m, size)

- Función: Aplica XOR con la misma imagen aleatoria usada originalmente.
- Efecto: Recupera los valores originales.

5. Opera_rota_inverse(pixelData, size, n)

- Función: Rota n bits a la izquierda.
- Efecto: Deshace la rotación previa.

6. Opera_despla_inverse(pixelData, size, n, etapa)

- Función: Recupera los bits desplazados y los reinserta.
- Efecto: Reconstrucción total del byte original si se conservaron los bits perdidos.

7. generateI_m(width, height, seed)

- Función: Genera una secuencia pseudoaleatoria de bytes del mismo tamaño que la imagen.
- Uso: Para el operador XOR reversible.

8. Enmascaramiento(...)

- Función: Inserta parte de la imagen transformada en una máscara (otra imagen).
- Efecto: Oculta contenido visualmente dentro de otra imagen + registro en .txt.

9. DesEnmascaramiento(...)

- Función: Lee archivo .txt, resta la máscara y reconstruye los píxeles ocultos.
- Efecto: Recupera el segmento oculto si los datos no se han alterado.

10. loadSeedMasking(...)

- Función: Carga semilla y valores RGB desde archivo de enmascaramiento.
- Uso: Para verificar o reconstruir fragmentos.

EXPERIMENTOS

- **Experimento uno:** Sí eliminamos o comentamos la línea *"file << static_cast<int>(Bitsper) <<"";"* que se encuentra en la función *Opera_despla()* se puede analizar lo siguiente:

1. No se podrán recuperar los bits eliminados por el desplazamiento.
2. La imagen desencriptada tendrá artefactos visuales o diferencias de cientos o miles de bytes.
3. El conteo final de errores debería ser significativo.

- **Experimento dos:** En el *main()*, se reemplaza la selección aleatoria de n en la operación de rotación por un valor fijo:

Y se aplica esta transformación en cada etapa:

```
// En lugar de: int n = 2 + (rand() % 5);
int n = 7;
unsigned char* result = Opera_rota(pixelData, size, n);
```

Se puede analizar lo siguiente:

```
for (int i = 0; i < numTransformaciones; ++i) {
    // SOLO rotación de 7 bits
    int n = 7;
    unsigned char* result = Opera_rota(pixelData, size, n);
    if (pixelData != originalPixels){
        delete[] pixelData;
    }
    pixelData = result;

    Enmascaramiento(pixelData, mascaraPixels, width, height, maskWidth, maskHeight, 5000+i, i);
}
```

1. La inversión mediante *Opera_rota_inverse()* con n = 7 devuelve exactamente los datos originales.
2. el error final es 0.
3. La imagen resultante está visualmente altamente distorsionada.

PROBLEMAS EN EL DESARROLLO

1. Manipulación de imágenes sin estructuras STL Se optó por usar arreglos simples en lugar de estructuras como std::vector, dificultando la gestión dinámica de datos. Se solucionó mediante uso estricto de new[] y delete[].
2. Lectura y escritura de imágenes BMP sin pérdida La exportación de imágenes debía preservar los valores RGB exactos. Se utilizó el formato QImage::Format_RGB888 para evitar pérdidas por compresión o relleno.
3. Reversibilidad de las transformaciones como el desplazamiento eliminan bits. Fue necesario guardar esa información en archivos auxiliares (bits_p*.txt).
4. Detección automática de transformaciones, el orden aleatorio de transformaciones requería un sistema heurístico para identificarlas. Se validaron valores 'visualmente plausibles' (32-250) tras aplicar inversiones.
5. Dependencia de archivos auxiliares, la pérdida de archivos como M*.txt impide la recuperación de la imagen. Se manejó validando existencia de los archivos antes de usarlos.
6. Control de aleatoriedad y reproducibilidad, el uso de rand() requiere semillas fijas para obtener los mismos resultados entre ejecuciones. Se usaron semillas fijas por etapa (1234 + i).
7. Sincronización entre imágenes y máscaras, máscaras de diferente tamaño pueden causar errores de segmentación. Se implementó control del tamaño y validación del rango aleatorio. Problemas enfrentados durante el desarrollo del sistema de enmascaramiento de imagen
8. Gestión de memoria dinámica, la reasignación de punteros entre etapas podía provocar memory leaks o dobles liberaciones. Se resolvió con liberación cuidadosa usando delete[].
9. Medición de errores en la reconstrucción, se necesitaba verificar si la imagen recuperada era exacta. Se comparó byte a byte con la imagen original y se mostró el número de errores.

10. Limitaciones de seguridad real el sistema, no emplea criptografía fuerte. Se asumió esta limitación y se definió como una solución experimental y educativa.

EVOLUCIÓN DEL PROYECTO

Este programa ha evolucionado en varias etapas, cada una agregando nuevas funcionalidades para trabajar con imágenes BMP:

1. Carga y procesamiento básico

- Comienza cargando una imagen BMP sin usar estructuras o STL.
- Usa arreglos dinámicos y memoria básica de C++ para almacenar los píxeles RGB.

2. Transformaciones aleatorias

Se aplica una serie de 5 transformaciones aleatorias sobre la imagen:

- XOR con una imagen aleatoria generada a partir de una semilla.
- Rotación de bits en cada píxel.
- Desplazamiento de bits, donde se guarda el fragmento desplazado.

3. Enmascaramiento

- Después de cada transformación, se aplica un proceso de enmascaramiento con una imagen de máscara.
- Se almacena el resultado del enmascaramiento (más la semilla de posición) en archivos M0.txt, M1.txt, ..., M4.txt.

4. Desenmascaramiento y recuperación

- Se cargan los datos de enmascaramiento desde los archivos .txt.
- Se intenta revertir cada transformación en orden inverso (etapas 4 → 0), probando todas las variantes posibles hasta encontrar una coincidencia plausible.
- Se guarda la imagen recuperada y se compara con la original.

Consideraciones importantes

Lógica y diseño

- No se usan estructuras ni STL (excepto vector<string> para registrar las transformaciones detectadas).
- Todo está basado en memoria dinámica con new[] y delete[].
- Muy útil para propósitos didácticos de bajo nivel y criptografía básica.

Archivos clave involucrados

- I_O.bmp: Imagen original de entrada.
- I_D.bmp: Imagen con transformaciones y enmascaramiento aplicada.
- I_Oi.bmp: Imagen reconstruida a partir del proceso inverso.
- M0.txt a M4.txt: Datos del enmascaramiento por etapa.
- bits_p0.txt a bits_p4.txt: Bits desplazados por etapa.

Memoria

- Hay buena gestión de memoria: se libera todo lo que se reserva.
- Uso cuidadoso de punteros y control de nullptr.

Robustez del inverso

- Se considera “válido” un intento de reversión si más del 80% de los valores están en el rango [32, 250].
- Este criterio puede fallar con imágenes que no tengan una distribución típica de bytes.