UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA Y TECNOLGIAS AVANZADAS – IPN
Prácticas de Laboratorio
Formato BMP y Esteganografía
Prof. Noé Sierra Romero

MATERIA

Multimedia

ALUMNO

Hernández Arteaga Itzel

TEMA

Esteganográfica

Grupo:3TM2 Practica 2

Fecha de asignación: 17/Febrero/2026

Introducción General

El formato BMP (Bitmap) es uno de los formatos de imagen más simples y utilizados en el estudio de procesamiento de imágenes digitales, debido a su estructura sin compresión (en su variante más común) y su cabecera bien documentada. Esta característica lo convierte en el soporte ideal para aprender técnicas de esteganografía, es decir, el arte de ocultar información dentro de un medio digital de forma que sea imperceptible al observador humano. Las dos prácticas de este módulo están diseñadas con progresión de dificultad: toma como base el conocimiento de la estructura del archivo BMP, y se avanza hacia la técnica clásica de Bit Menos Significativo (LSB), y finalmente se exploran variantes más robustas orientadas a la seguridad.

PRÁCTICA 2 — Esteganografía Avanzada:Distribución Aleatoria y Cifrado

Objetivo general: Implementar una versión robusta de esteganografía en BMP que combine distribución pseudoaleatoria de bits (basada en semilla/clave) y cifrado XOR del mensaje, dificultando su detección y extracción no autorizada.
Competencias a desarrollar: Generadores de números pseudoaleatorios, cifrado simétrico básico, diseño de protocolos esteganográficos, análisis estadístico (chi-cuadrado).
Herramientas sugeridas: Python 3 (random, hashlib, struct), imagen BMP 24 bpp ≥ 400×400 px.
Duración estimada: 60 minutos

3.1 Marco Teórico

La esteganografía LSB secuencial (Práctica 1) es vulnerable a herramientas de estegoanálisis que detectan cambios estadísticos en los LSBs de las primeras posiciones del archivo. Para mitigarlo, se emplea una distribución pseudoaleatoria: en lugar de escribir en posiciones contiguas, se selecciona el subconjunto de bytes portadores mediante un PRNG (Pseudo-Random Number Generator) inicializado con una clave secreta compartida. Adicionalmente, el mensaje puede cifrarse con XOR usando una clave derivada del hash SHA-256 de la contraseña, de modo que incluso si un atacante descubre que hay datos ocultos, no pueda recuperar el mensaje sin la clave correcta. El protocolo de esta práctica contempla: (1) derivación de clave mediante SHA-256, (2) cifrado XOR del mensaje, (3) construcción del flujo de bits (longitud + mensaje cifrado), (4) selección pseudoaleatoria de posiciones portadoras en la imagen y (5) incrustación LSB en esas posiciones.

3.2 Procedimiento

Código:

```
import struct
import hashlib
import random
import math
def leer_bmp(filepath):
```

```
    with open(filepath, 'rb') as f:
        data = f.read()

    offset = struct.unpack_from('<I', data, 10)[0]
    width  = struct.unpack_from('<i', data, 18)[0]
    height = struct.unpack_from('<i', data, 22)[0]
    row_size = (width * 3 + 3) & ~3

    header = bytearray(data[:offset])
    pixels = bytearray(data[offset:])

    return header, pixels, width, height, row_size

def guardar_bmp(filepath, header, pixels):
    with open(filepath, 'wb') as f:
        f.write(header)
        f.write(pixels)

def derivar_clave(password: str, length: int) -> bytes:
    hash_bytes = hashlib.sha256(password.encode()).digest()
    clave = bytearray()

    while len(clave) < length:
        clave.extend(hash_bytes)

    return bytes(clave[:length])

def cifrar_xor(mensaje: bytes, password: str) -> bytes:
    clave = derivar_clave(password, len(mensaje))
    return bytes(m ^ k for m, k in zip(mensaje, clave))

def descifrar_xor(cifrado: bytes, password: str) -> bytes:
    return cifrar_xor(cifrado, password)  # XOR es simétrico
def generar_permutacion(password: str, total_bytes: int):
    seed = int(hashlib.sha256(password.encode()).hexdigest(), 16)
    random.seed(seed)

    posiciones = list(range(total_bytes))
    random.shuffle(posiciones)
return posiciones

def embed_secure(src_path, dst_path, mensaje, password):
    header, pixels, width, height, row_size = leer_bmp(src_path)

    msg_bytes = mensaje.encode('utf-8')
    msg_cifrado = cifrar_xor(msg_bytes, password)

    msg_len = len(msg_cifrado)

    datos = struct.pack('>I', msg_len) + msg_cifrado

    bits = []
    for byte in datos:
```

```
        for i in range(7, -1, -1):
            bits.append((byte >> i) & 1)

    if len(bits) > len(pixels):
        raise ValueError("Mensaje demasiado grande para esta imagen")

    perm = generar_permutacion(password, len(pixels))

    pixels_mod = bytearray(pixels)

    for i, bit in enumerate(bits):
        pos = perm[i]
        pixels_mod[pos] = (pixels_mod[pos] & 0xFE) | bit

    guardar_bmp(dst_path, header, pixels_mod)

    print(f"[OK] {msg_len} bytes cifrados e incrustados en {dst_path}")

def extract_secure(stego_path, password):
    _, pixels, _, _, _ = leer_bmp(stego_path)

    perm = generar_permutacion(password, len(pixels))

    len_bits = [pixels[perm[i]] & 1 for i in range(32)]

    msg_len = 0
    for b in len_bits:
        msg_len = (msg_len << 1) | b

    total_bits = 32 + msg_len * 8

    if total_bits > len(pixels):
        raise ValueError("Contraseña incorrecta o datos corruptos")

    msg_bits = [pixels[perm[i]] & 1 for i in range(32, total_bits)]

    msg_bytes = bytearray()

    for i in range(0, len(msg_bits), 8):
        byte = 0
        for bit in msg_bits[i:i+8]:
            byte = (byte << 1) | bit
        msg_bytes.append(byte)

    msg_descifrado = descifrar_xor(bytes(msg_bytes), password)
return msg_descifrado.decode('utf-8')

MENSAJE = "TELEMÁTICA SECRETA 2025"
CLAVE   = "clave_super_segura"

embed_secure("imagen.bmp", "stego_seguro.bmp", MENSAJE, CLAVE)

recuperado = extract_secure("stego_seguro.bmp", CLAVE)
```

```
print("Mensaje recuperado:", recuperado)

assert recuperado == MENSAJE
print("Prueba exitosa ")
```

Fig.1 Imagen que se utilizó para la práctica 1

Resultados

```
[OK] 24 bytes cifrados e incrustados en stego_seguro.bmp
Mensaje recuperado: TELEMÁTICA SECRETA 2025
Prueba exitosa ✓
```

Complete la siguiente tabla comparando los tres escenarios analizados:

Método	PSNR (dB)	x^2LSB	Detectable	Decodificable sin clave
Imagen limpia	∞	Natural	N/A	N/A
LSB secuencial (P2)	48-52 dB	Alto	Si	Si
LSB aleatorio + XOR (P3)	48-52 dB	Bajo	N/A	No

3.5 Preguntas de Análisis

1. ¿En qué medida mejora la distribución aleatoria la resistencia al análisis χ^2 respecto al LSB secuencial? Justifique con los valores obtenidos.

En LSB secuencial (P2), los bits modificados se agrupan en posiciones contiguas, alterando de forma detectable la distribución estadística de los bits menos significativos. Esto provoca que el valor χ^2 aumente y el p-valor disminuya, indicando desviación respecto al comportamiento natural de la imagen.

2. El cifrado XOR con clave derivada de SHA-256 no es criptográficamente seguro por sí solo para todos los casos de uso. ¿Cuál es su principal vulnerabilidad? ¿Qué algoritmo recomendaría en su lugar?

Su vulnerabilidad principal es: Si dos mensajes se cifran con la misma clave, el patrón de repetición puede explotarse; no ofrece autenticidad ni protección contra manipulación; es vulnerable a ataques conocidos si parte del mensaje es predecible (known-plaintext attack). Aunque SHA-256 es seguro como función hash, el esquema XOR repetitivo no constituye un cifrado robusto.

Algoritmo recomendado:

- AES-256 en modo GCM (recomendado)
- O AES-CBC con IV aleatorio

3. ¿Qué cambios haría al protocolo para ocultar no solo texto sino un archivo binario (por ejemplo, un archivo ZIP o una imagen secundaria)?

Leer el archivo en modo binario:

```
with open("archivo.zip", "rb") as f:
    datos = f.read()
```

El protocolo base no cambia. Solo se elimina la dependencia de UTF-8.

4. Investigue la técnica de estegoanálisis RS (Regular-Singular). ¿Sería efectiva contra la implementación de esta práctica? Explique por qué.

La técnica de estegoanálisis **RS (Regular-Singular)** es un método estadístico preciso utilizado para detectar información oculta en imágenes digitales, siendo especialmente eficaz contra la esteganografía de bits menos significativos (LSB). Esta técnica, introducida por Jessica Fridrich, se basa en la observación de que las imágenes naturales tienen una estructura suave que se altera cuando se incrustan datos, convirtiendo grupos de píxeles "regulares" en "singulares". Esta técnica seria efectiva con lo que se realizó ya que es mas preciso en los bits mas significativos.