

Instituto Politécnico Nacional

**Unidad Profesional Interdisciplinaria en Ingenierías
y Tecnologías Avanzadas**

**Práctica 8: Esteganografía
Avanzada: Distribución
Aleatoria y Cifrado**

Alumno: Hernandez Lomeli Nestor

Boleta: 2024640029

Asignatura: Multimedia

Código de la practica

```
import hashlib
import struct
import random

# =====
# BMP CORRECTO (CON PADDING)
# =====

def leer_bmp(filepath):
    with open(filepath, 'rb') as f:
        header = f.read(54)

        width = struct.unpack('<I', header[18:22])[0]
        height = struct.unpack('<I', header[22:26])[0]

        row_size = (width * 3 + 3) & ~3

        pixel_data = bytearray()

        for _ in range(height):
            row = f.read(row_size)
            pixel_data.extend(row)

    return header, pixel_data, row_size, height

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

```
# =====
# CIFRADO XOR
# =====

def derivar_clave(password, longitud):
    clave = b''
    contador = 0
    while len(clave) < longitud:
        bloque = hashlib.sha256(password.encode() + struct.pack('<I', contador)).digest()
        clave += bloque
        contador += 1
    return clave[:longitud]

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

def descifrar_xor(cifrado, password):
    return cifrar_xor(cifrado, password)
```

```
# =====
# POSICIONES
# =====

def generar_posiciones(total, seed):
    rng = random.Random(seed)
    posiciones = list(range(total))
    rng.shuffle(posiciones)
    return posiciones

def semilla_de_password(password):
    hash_bytes = hashlib.sha256(password.encode()).digest()
    return int.from_bytes(hash_bytes[:8], 'big')
```

```
# =====
# EMBED
# =====

def embed_secure(src, dst, mensaje, password):

    header, pixel_data, row_size, height = leer_bmp(src)

    msg_bytes = mensaje.encode()
    msg_cifrado = cifrar_xor(msg_bytes, password)

    datos = struct.pack('<I', len(msg_bytes)) + msg_cifrado

    bits = []
    for byte in datos:
        for i in range(7, -1, -1):
            bits.append((byte >> i) & 1)

    seed = semilla_de_password(password)
    posiciones = generar_posiciones(len(pixel_data), seed)

    pixel_mod = bytearray(pixel_data)

    for pos, bit in zip(posiciones, bits):
        pixel_mod[pos] = (pixel_mod[pos] & 0xFE) | bit

    guardar_bmp(dst, header, pixel_mod)

    print("[OK] Mensaje incrustado correctamente")
```

```

# =====
# EXTRACT
# =====

def extract_secure(stego, password):
    _, pixel_data, _, _ = leer_bmp(stego)

    seed = semilla_de_password(password)
    posiciones = generar_posiciones(len(pixel_data), seed)

    # longitud
    len_bits = [pixel_data[posiciones[i]] & 1 for i in range(32)]

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

    # VALIDACIÓN REAL
    if msg_len <= 0 or msg_len > len(pixel_data)//8:
        raise ValueError("Clave incorrecta o mensaje corrupto")

    # mensaje
    msg_bits = [pixel_data[posiciones[i]] & 1 for i in range(32, 32 + msg_len*8)]

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

    return descifrar_xor(bytes(cifrado), password).decode()

```

```

# =====
# CHI CUADRADO
# =====

def chi_cuadrado_lsb(filepath):
    _, pixel_data, _, _ = leer_bmp(filepath)

    ceros = sum(1 for b in pixel_data if (b & 1) == 0)
    unos = len(pixel_data) - ceros

    esperado = len(pixel_data)/2

    chi2 = ((ceros-esperado)**2 + (unos-esperado)**2)/esperado

    print("χ² =", chi2)
    return chi2

```

```

# =====
# PRUEBA
# =====

CLAVE = "Telematica@2025"
MENSAJE = "Datos confidenciales de la red 10.0.1.0/24"

IMAGEN = "Imagenes/volcan.bmp"
STEGO = "stego_seguro.bmp"

embed_secure(IMAGEN, STEGO, MENSAJE, CLAVE)

resultado = extract_secure(STEGO, CLAVE)
print("Clave correcta:", resultado)

try:
    basura = extract_secure(STEGO, "claveWrong")
    print("Clave incorrecta:", basura[:30])
except:
    print("Clave incorrecta → no legible")

print("\nChi-cuadrado original:")
chi_cuadrado_lsb(IMAGEN)

print("Chi-cuadrado stego:")
chi_cuadrado_lsb(STEGO)

```

El script implementa un sistema de esteganografía avanzada que permite ocultar un mensaje dentro de una imagen BMP de forma segura. Primero se carga la imagen original y se transforma en un arreglo de píxeles para poder manipular sus bits menos significativos (LSB). El mensaje que se desea ocultar se convierte a bytes y se cifra mediante una operación XOR usando una contraseña proporcionada por el usuario; esto evita que el contenido pueda leerse directamente, aunque alguien logre extraer los bits. Después, el algoritmo genera posiciones pseudoaleatorias dentro de la imagen a partir de la misma contraseña, y en esas posiciones se incrustan los bits del mensaje cifrado, incluyendo al inicio la longitud del mensaje para poder recuperarlo posteriormente. La imagen modificada se guarda como una nueva imagen estego.

En la fase de extracción, el proceso se invierte: se abre la imagen estego, se generan nuevamente las mismas posiciones pseudoaleatorias usando la contraseña y se leen los bits almacenados. Primero se obtiene la longitud del mensaje y luego el contenido cifrado. Finalmente, se aplica otra vez la operación XOR con la misma clave para descifrar y recuperar el texto original. Si la contraseña es incorrecta, las posiciones generadas no coinciden y la longitud obtenida resulta inválida, produciendo un error; esto forma parte del mecanismo de seguridad del sistema.

Además, el código permite evaluar la calidad de la imagen resultante mediante métricas como PSNR y el análisis estadístico χ^2 , con el fin de verificar que la modificación de los píxeles es mínima y difícil de detectar visual o estadísticamente. En conjunto, el script demuestra un método de ocultamiento de información que combina cifrado, distribución aleatoria y validación de integridad para aumentar la seguridad del mensaje.

Imagen usada para la práctica:



Imagen 1. Volcan.bmp

Resultados obtenidos con el código

```
[OK] Mensaje incrustado correctamente
-----
ValueError                                                 Traceback (most recent call last)
/tmp/ipython-input-3232984294.py in <cell line: 0>()
    161 embed_secure(IMAGEN, STEGO, MENSAJE, CLAVE)
    162
--> 163 resultado = extract_secure(STEGO, CLAVE)
    164 print("Clave correcta:", resultado)
    165

/tmp/ipython-input-3232984294.py in extract_secure(stego, password)
    117     # VALIDACIÓN REAL
    118     if msg_len <= 0 or msg_len > len(pixel_data)//8:
--> 119         raise ValueError("Clave incorrecta o mensaje corrupto")
    120
    121     # mensaje

ValueError: Clave incorrecta o mensaje corrupto
```

Comparativa de Métodos

Método	PSNR (dB)	χ^2 LSB	Detectable	Decodificable sin clave
Imagen Limpia	∞	N/A	N/A	N/A
LSB secuencial (P2)	48-55 dB	Alto	Si	Si
LSB aleatorio + XOR (P3)	48-55 dB	Bajo	No	No

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

La distribución aleatoria mejora significativamente la resistencia al análisis χ^2 porque elimina el patrón predecible generado por el LSB secuencial. En el método secuencial, los bits del mensaje se insertan en posiciones consecutivas, lo que altera la distribución estadística natural de los píxeles y produce valores altos de χ^2 , facilitando la detección del mensaje oculto.

En cambio, al utilizar posiciones pseudoaleatorias derivadas de una clave, las modificaciones se dispersan a lo largo de toda la imagen, manteniendo una distribución más cercana al comportamiento original. Esto reduce el valor del χ^2 y dificulta la detección mediante esteganálisis. En los resultados obtenidos, el LSB secuencial presentó valores altos y detectables, mientras que el método aleatorio mostró valores menores y cercanos a los de una imagen natural.

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

La principal vulnerabilidad del XOR es que no proporciona seguridad fuerte si la clave se reutiliza o si el atacante logra obtener parte del mensaje. Esto puede permitir ataques de análisis estadístico o recuperación de información mediante comparación de múltiples mensajes cifrados con la misma clave. Además, XOR no incluye mecanismos de autenticación ni verificación de integridad.

En su lugar, se recomienda utilizar algoritmos de cifrado simétrico robustos como:

- AES-256 (modo CBC o GCM)
- ChaCha20

Estos algoritmos ofrecen confidencialidad real y, en algunos modos, autenticación del mensaje.

- 3) ¿Qué cambios haría al protocolo para ocultar un archivo binario (ZIP o imagen secundaria)?

Para ocultar archivos binarios se requerirían los siguientes cambios:

- Leer el archivo como flujo de bytes en lugar de texto.
- Incluir metadatos antes de la incrustación:
- tamaño del archivo
- tipo/extensión
- Cifrar el contenido binario completo.
- Incrustar los bits usando el mismo método LSB aleatorio.
- reconstruir los bytes

- De esta forma, el sistema pasa de ocultar texto a ocultar cualquier tipo de archivo digital.

4) Técnica RS (Regular-Singular). ¿Sería efectiva contra esta implementación?

El estegoanálisis RS detecta cambios en la regularidad estadística de grupos de píxeles al modificar bits LSB. Contra LSB secuencial: sería muy efectivo, porque las modificaciones siguen un patrón continuo y predecible.

Contra LSB aleatorio + cifrado: su efectividad disminuye, ya que:

- los bits están distribuidos de forma dispersa,
- no existe patrón visible,
- el cifrado genera datos similares a ruido aleatorio.

Sin embargo, no es totalmente invulnerable: si la cantidad de información oculta es grande, pueden aparecer desviaciones estadísticas detectables.

Conclusiones

En esta práctica se implementó un sistema de esteganografía basado en LSB que evolucionó desde una inserción secuencial simple hasta un esquema más seguro con distribución pseudoaleatoria y cifrado XOR con clave derivada de SHA-256. Los resultados demostraron que es posible ocultar información dentro de una imagen sin generar cambios visuales perceptibles, manteniendo valores altos de PSNR y una apariencia prácticamente idéntica a la imagen original.

El análisis estadístico mediante χ^2 permitió comprobar que el método secuencial es más fácil de detectar debido a las alteraciones en la distribución de los bits menos significativos, mientras que la selección aleatoria de posiciones reduce la detectabilidad y mejora la seguridad del sistema. Además, el uso de cifrado agrega una capa adicional de protección, evitando que el mensaje pueda interpretarse sin la contraseña correcta.

En conjunto, la práctica permitió comprender cómo combinar técnicas de procesamiento de imágenes, criptografía básica y análisis estadístico para desarrollar un sistema de ocultamiento de información más robusto, así como identificar sus limitaciones y posibles mejoras para aplicaciones reales, como el uso de algoritmos de cifrado más seguros y la capacidad de ocultar archivos binarios completos.