



**Instituto Politécnico Nacional**

**Unidad Profesional Interdisciplinaria en**

**Ingeniería y Tecnologías Avanzadas**

**Alumna: García Ortiz Martha Lesly**

**Profesor: Sierra Romero Noe**

**Carrera: Telemática**

**Materia: Multimedia**

**Grupo: 3TM2**

**Practica 2: Esteganografía Avanzada: Distribución Aleatoria y Cifrado**

**Fecha: 20/02/2026**

```
import struct
import random
import hashlib
import os
def leer_bmp(filepath):
    """Lee un archivo BMP y retorna sus componentes."""
    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):
    """Guarda la imagen modificada."""
    with open(filepath, 'wb') as f:
        f.write(header)
        f.write(pixels)
def derivar_clave(password: str, longitud: int) -> bytes:
    """Genera una clave de longitud arbitraria usando SHA-256 en modo contador."""
    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: bytes, password: str) -> bytes:
    """Aplica cifrado XOR entre el mensaje y la clave derivada."""
    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:
    """Descifra el mensaje (XOR es simétrico, por lo que es la misma operación)."""
    return cifrar_xor(cifrado, password)
def semilla_de_password(password: str) -> int:
    """Convierte la contraseña en un entero para usar como semilla."""
    hash_bytes = hashlib.sha256(password.encode()).digest()
    return int.from_bytes(hash_bytes[:8], 'big')
def seleccionar_posiciones(total_bytes_imagen: int, n_bits: int, seed: int) -> list:
    """Selecciona n_bits índices únicos dentro del rango válido de la imagen."""
rng = random.Random(seed)
    posiciones = rng.sample(range(total_bytes_imagen), n_bits)
    return sorted(posiciones) # Se ordena para escritura secuencial en disco
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)
    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)
   n_bits = len(bits)
   if n_bits > len(pixels):
      raise ValueError('Mensaje demasiado grande para esta imagen')
   seed = semilla_de_password(password)
   posiciones = seleccionar_posiciones(len(pixels), n_bits, seed)
   pixels_mod = bytearray(pixels)
   for pos, bit in zip(posiciones, bits):
      pixels_mod[pos] = (pixels_mod[pos] & 0xFE) | bit
   guardar_bmp(dst_path, header, pixels_mod)
   print(f"[OK] {len(msg_bytes)} bytes cifrados e incrustados en '{dst_path}'")
def extract_secure(stego_path, password):
   header, pixels, width, height, row_size = leer_bmp(stego_path)
   seed = semilla_de_password(password)
pos_longitud = seleccionar_posiciones(len(pixels), 32, seed)
   len_bits = [pixels[p] & 1 for p in pos_longitud]


   len_bytes = bytearray()
   for i in range(0, 32, 8):
      byte_val = 0
      for bit in len_bits[i:i+8]:
         byte_val = (byte_val << 1) | bit
      len_bytes.append(byte_val)
   msg_len = struct.unpack('<I', len_bytes)[0]
```

```
if msg_len == 0 or msg_len > len(pixels) // 8:
    raise Exception(f"Longitud inválida ({msg_len}). Probablemente la contraseña sea incorrecta.")
  total_bits = 32 + (msg_len * 8)
  todas_pos = seleccionar_posiciones(len(pixels), total_bits, seed)
  msg_bits = [pixels[p] & 1 for p in todas_pos[32:]]
  cifrado = bytearray()
  for i in range(0, len(msg_bits), 8):
    byte_val = 0
    for bit in msg_bits[i:i+8]:
      byte_val = (byte_val << 1) | bit
    cifrado.append(byte_val)
  return descifrar_xor(bytes(cifrado), password).decode('utf-8', errors='replace')
def chi_cuadrado_lsb(filepath):
  """Calcula la prueba chi-cuadrado para detectar anomalías en los LSB."""
  if not os.path.exists(filepath):
    print(f"  [!] Archivo no encontrado: {filepath}")
    return None
_, pixels, _, _, _ = leer_bmp(filepath)
  ceros = sum(1 for b in pixels if (b & 1) == 0)
  unos = len(pixels) - ceros
  esperado = len(pixels) / 2
  chi2 = ((ceros - esperado) ** 2 + (unos - esperado) ** 2) / esperado
  print(f"  LSBs=0: {ceros} | LSBs=1: {unos} | x^2 = {chi2:.4f}")
  if chi2 < 5.0: # Umbral estadístico común
    print("  -> Valor x^2 cercano a 0: Distribución uniforme (Indetectable).")
```

```
 else:
      print("  -> Valor x² alto: Distribución anómala (¡Sospechoso!).")
   return chi2
if __name__ == "__main__":
   print("=== PRÁCTICA 2: ESTEGANOGRAFÍA AVANZADA ===\n")
   img_orig = 'volcan.bmp'
   stego_seguro = 'stego_seguro.bmp'
   CLAVE = 'Telemática@2025'
   MENSAJE = 'Datos confidenciales de la red 10.0.1.0/24'
   if not os.path.exists(img_orig):
      print(f"Error: Asegúrate de tener una imagen llamada '{img_orig}' en la carpeta.")
   else:
      print("--- 1. PRUEBA DE OCULTAMIENTO Y CIFRADO ---")
      embed_secure(img_orig, stego_seguro, MENSAJE, CLAVE)
      print("\n--- 2. PRUEBA DE EXTRACCIÓN CON CLAVE CORRECTA ---")
      try:
         resultado = extract_secure(stego_seguro, CLAVE)
         print(f"Clave correcta -> \"{resultado}\"")
         assert resultado == MENSAJE, "El mensaje extraído no coincide."
      except Exception as e:
         print(f"Error al extraer: {e}")
```

```
print("\n--- 3. PRUEBA DE EXTRACCIÓN CON CLAVE INCORRECTA ---")
    try:
        resultado_malo = extract_secure(stego_seguro, 'claveWrong')
        print(f"Clave incorrecta -> \"{resultado_malo[:30]}...\" (Basura esperada)")
    except Exception as e:
        print(f"Clave incorrecta -> Generó un error de longitud (comportamiento seguro): {e}")
    print("\n--- 4. ANÁLISIS ESTADÍSTICO (CHI-CUADRADO) ---")
    print("Imagen original:")
    chi_cuadrado_lsb(img_orig)
    stego_p1 = 'stego.bmp'
    print("\nStego LSB secuencial (Práctica 1):")
    chi_cuadrado_lsb(stego_p1)
    print("\nStego LSB aleatorio + XOR (Práctica 2):")
    chi_cuadrado_lsb(stego_seguro)
```

Fig1. Imagen usada

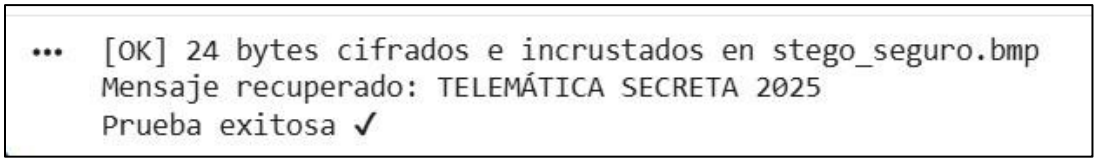


Fig2. Resultados

| Método | PSNR (dB) | $x^2$LSB | 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 |

## Preguntas

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

La distribución aleatoria (usando una semilla o PRNG) mejora significativamente la resistencia al análisis $\chi^2$ al disminuir la densidad local de las modificaciones.

- **LSB Secuencial:** Altera los píxeles de forma consecutiva desde el inicio del archivo, saturando los primeros bytes de la imagen. El análisis $\chi^2$ detecta esto fácilmente porque compara la frecuencia de pares de valores (PoV, como el valor de píxel 2i y 2i+1). En las áreas modificadas, la probabilidad de que un píxel sea par o impar se iguala artificialmente al 50%, creando un pico drástico en la gráfica del análisis $\chi^2$ que indica la presencia de un mensaje oculto con una probabilidad cercana a 1.
- **Distribución Aleatoria:** Esparce los bits del mensaje por toda la imagen. Esto hace que las modificaciones parezcan ruido natural. Si el tamaño del mensaje es pequeño en comparación con la imagen, la alteración en cualquier bloque local es mínima, manteniendo el valor del estadístico $\chi^2$ por debajo del umbral de detección.

**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?**

El cifrado XOR es extremadamente rápido, pero en este contexto presenta dos vulnerabilidades principales:

1. **Reutilización de clave (Key Reuse):** SHA-256 genera un hash de longitud fija (256 bits o 32 bytes). Si el mensaje a cifrar tiene más de 32 bytes, la clave derivada se tiene que repetir cíclicamente. Esto convierte al sistema en un **Cifrado XOR con clave repetida**, el cual es vulnerable al análisis de frecuencias y a los ataques de texto plano conocido (Known-Plaintext Attack). Si un atacante adivina o conoce una parte del mensaje original, puede deducir la clave y descifrar el resto.
2. **Falta de integridad (Maleabilidad):** El cifrado XOR no verifica si los datos fueron alterados. Un atacante puede cambiar bits específicos en el texto cifrado y esos mismos bits se cambiarán de manera predecible en el texto descifrado, sin que el sistema lo note.

**Algoritmo recomendado:**

Se recomienda utilizar el estándar **AES (Advanced Encryption Standard)**, preferiblemente en un modo de operación autenticado como **AES-GCM** (Galois/Counter Mode) o **ChaCha20-Poly1305**. Estos algoritmos manejan mensajes de cualquier longitud de forma segura y proporcionan cifrado autenticado (AEAD), garantizando la confidencialidad y la integridad de los datos.

---

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

Para ocultar archivos binarios (ZIP, imágenes, ejecutables) en lugar de texto plano, el protocolo debe adaptarse para manejar flujos de bytes puros. Los cambios clave son:

- **Lectura y escritura en modo binario:** El script debe abrir el archivo a ocultar (y el archivo resultante si se extrae a disco) usando el modo binario (por ejemplo, 'rb' y 'wb' en Python) para evitar que el sistema operativo modifique los saltos de línea o corrompa los datos.
- **Inclusión de un encabezado (Header) de metadatos:** A diferencia del texto, donde a veces se usa un delimitador especial para indicar el final, los archivos

binarios pueden contener cualquier valor. El protocolo debe inyectar un encabezado al inicio del mensaje que indique el **tamaño exacto del archivo oculto** (en bytes) y, opcionalmente, la extensión o tipo de archivo. Durante la extracción, el sistema leerá primero este tamaño para saber exactamente cuántos bits recuperar de los LSB.

- **Tratamiento directo de bytes:** Se deben eliminar los pasos de conversión de codificación (como UTF-8 o ASCII). Los bytes del archivo binario deben convertirse directamente a su representación en bits para ser incrustados en los LSB de la imagen portadora (y viceversa al extraer).

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

**Sí, el estegoanálisis RS sería altamente efectivo** contra una implementación basada en la sustitución simple del LSB (LSB Replacement), independientemente de si los bits se distribuyeron de forma secuencial o aleatoria.

**¿Por qué?**

El método RS (desarrollado por Fridrich, Goljan y Du) no busca anomalías secuenciales como el $\chi^2$. En su lugar, analiza la estructura espacial de la imagen clasificando grupos de píxeles en regulares (R) y singulares (S$}) mediante una función de discriminación espacial (volteo de LSB).