Universidad de Antioquia
Departamento de Electrónica y Telecomunicaciones
2598521 – INFORMATICA II

Informe final

Desafío 1: Compresión y Encriptación

Rigoberto Berrio Berrio
Estudiante de Ingeniería Electrónica
Universidad de Antioquia
rigoberto.berrio1@udea.edu.co

Jeisson Stevens Martínez Arevalo
Estudiante de Ingeniería en Telecomunicaciones
Universidad de Antioquia
jeisson.martinez1@udea.edu.co

29 de septiembre de 2025

Resumen

Este informe documenta el diseño, implementación y validación de una herramienta para recuperar mensajes a partir de archivos cifrados y comprimidos, sin metadatos confiables sobre el formato original. La solución propone una búsqueda exhaustiva acotada sobre los parámetros de cifrado por byte (XOR y rotaciones circulares) y una autodetección robusta de varios esquemas de compresión (RLE en tres variantes y LZ78 en tripletas con índice de 16 bits en *endianness* LE/BE), validando el resultado mediante una pista normalizada por caso. Se priorizan la robustez, la eficiencia razonable y la trazabilidad de resultados.

Índice

1 Análisis del problema y consideraciones de la solución (a)

1.1 Contexto del desafío

Se requiere recuperar el texto plano de archivos `EncriptadoX.txt` cuyos contenidos fueron:

1. **Cifrados** por byte aplicando un **XOR** de 8 bits (clave $K \in [0, 255]$) y una **rotación circular** de bits ($n \in \{1, \ldots, 7\}$) sobre cada byte, en un **orden desconocido**: el emisor pudo usar `ROL`→`XOR` o `ROR`→`XOR`.
2. **Comprimidos** según uno de estos formatos:
   - **RLE** (Run-Length Encoding) en tres dialectos: RLE ASCII tipo `NNNs`; RLE binaria par `[len][sym]`; y RLE "tripleta" `[_][len][sym]` (ignoramos el marcador).
   - **LZ78** en tripletas (idx, c) con índice de 16 bits y **endianness LE** (byte menos significativo primero) o **BE** (byte más significativo primero).
3. Validación semántica mediante una **pista por caso** (`pistaX.txt`) que se **normaliza** (minúsculas ASCII y sin espacios ni saltos) y se busca en el texto descomprimido.

1.2 Restricciones y ambigüedades

- **Orden de cifrado desconocido**: si el origen fue `ROL`→`XOR`, el reverso correcto es `XOR`→`ROR`; si fue `ROR`→`XOR`, el reverso es `XOR`→`ROL`.
- **Endianness LZ78 desconocida**: el índice de 16 bits en las tripletas puede estar en **LE** o **BE**.
- **Formato de compresión desconocido**: como no hay cabeceras firmes (qué es y cómo se lee), se requiere *autodetección*.

1.3 Objetivos de la solución

- Determinar, para cada `EncriptadoX.txt`, la combinación (n, K, orden) que produce un flujo descomprimible bajo alguno de los formatos esperados y cuyo texto resultante contenga la pista normalizada.
- Reportar el **método de compresión** detectado, la **rotación** n y la **clave** K en hexadecimal y decimal.
- Guardar el plano en `salidaX.txt` y mostrar una vista previa segura por consola.

1.4 Estrategia general

1. **Fuerza bruta acotada** sobre (n, K) y **dos órdenes** de descifrado: `XOR`→`ROR` y `XOR`→`ROL`.

2. **Autodetección** de compresión en orden *fail-fast*: `RLE_tripleta` → `RLE_bin_LE` → `RLE_ASCII` → `LZ78_LE` → `LZ78_BE`.

3. **Validación semántica**: búsqueda de la pista normalizada dentro del plano descomprimido **durante la comparación** (conversión a minúsculas en el momento de comparar, sin modificar el búfer).

2 Secuencia de tareas (b) — diagrama

1. **Entrada por consola**: leer n (número de casos).
2. **Inicialización por caso** X:
   2.1. Construir rutas: `EncriptadoX.txt`, `pistaX.txt`, `salidaX.txt`.
   2.2. **Leer** `EncriptadoX.txt` a memoria (`cipher`, `clen`); si falla, saltar caso.
   2.3. **Leer y normalizar** `pistaX.txt` (`clueLower`, `clueLen`); si falla o queda vacía, saltar caso.
3. **Recuperación** (`recover`):
   3.1. Reservar búfer temporal `tmp`.
   3.2. Para $n = 1.,7$ y $K = 0.,255$:
      3.2.1. Aplicar `XOR→ROR`; `try_all_decompress`; si descomprime, buscar la pista.
      3.2.2. Si no hay éxito, aplicar `XOR→ROL`; repetir prueba de descompresores y búsqueda de pista.
      3.2.3. Si aparece la pista, **éxito**: devolver plano, método, n, K.
4. **Post-proceso por caso**:
   4.1. Si falla, informar y continuar.
   4.2. Si tiene éxito: mapear método a nombre legible, imprimir parámetros, escribir `salidaX.txt`, mostrar vista previa (1200 chars), liberar búferes.
5. **Fin**: retorno del programa.

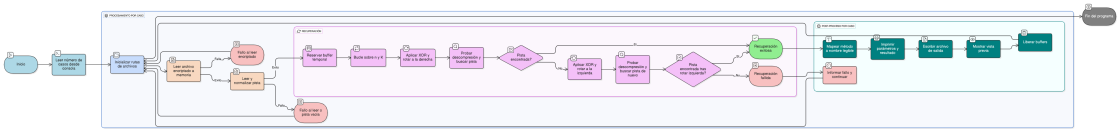

Figura 1: Diagrama de la secuencia de tareas.

3 Algoritmos implementados (c)

3.1 E/S de archivos

`readFile(path,outLen)`: abre en binario, mide con `seekg/tellg`, reserva búfer, lee y retorna puntero y longitud.
`writeFile(path,buf,len)`: abre en binario, escribe, devuelve `true/false`.
Complejidad: `readFile` tiempo $O(N)$ y memoria $O(N)$ (por el búfer); `writeFile` tiempo $O(T)$ y **memoria auxiliar** $O(1)$.

3.2 Descifrado por byte

XOR: $b \oplus K$. **Rotaciones**: `ROL8/ROR8` con $n \,\&= 7$.
`applyXorThenRotate`: para cada byte, aplica XOR y luego ROL o ROR según bandera `useROL`.
Observación: para revertir `ROL`→`XOR`, se usa `XOR`→`ROR`; para revertir `ROR`→`XOR`, se usa `XOR`→`ROL`.

3.3 Autodetección de compresión: `try_all_decompress`

Orden *fail-fast* (del más barato/probable al más costoso):

1. `RLE_tripleta`
2. `RLE_bin_LE`
3. `RLE_ASCII`
4. `LZ78_LE`
5. `LZ78_BE`

Devuelve el primer método que descomprime; también un código `which` $\in \{1.,5\}$.

3.4 RLE (tres variantes)

RLE ASCII: parsea dígitos (máx. 12) → símbolo → expansión; rechaza `num=0`; realoca capacidad cuando es necesario.
RLE binaria par: `[len][sym]` repetidos; rechaza `len=0`.
RLE tripleta: `[_][len][sym]`; ignora el primer byte; rechaza `len=0`.
Complejidad: $O(T)$ para salida de tamaño T; defensas: límites y validaciones.

3.5 LZ78 en tripletas (índice de 16 bits)

Interfaz: `lz78_decompress_triplet_endian(in,inLen,littleEndian,...)`
Entrada: tripletas `[b0][b1][c]` con $idx = LE?(b_0 \,|\, b_1 \ll 8) \;:\; (b_0 \ll 8 \,|\, b_1)$ y carácter c.
Diccionario: arreglos `parent[]` y `ch[]`; $nextIdx$ inicia en 1 (0 = cadena vacía).
Reconstrucción:

1. Caminar por `parent` desde idx apilando `ch[t]` hasta llegar a 0.

2. Escribir el prefijo invirtiendo la pila; añadir c.

3. Insertar nueva entrada: `parent[next]=idx; ch[next]=c; ++next`.

Validaciones: $idx < nextIdx$, límite de diccionario (65535), límite de pila (65537), realocaciones de salida, entrada múltiplo de 3.
Complejidad: $O(T)$ amortizado; memoria $O(\#\text{tripletas}) + O(T) + 64\,\text{KiB}$ (pila).

3.6 Normalización y búsqueda de pista

`read_and_normalize_pista`: quita espacios y saltos (`\r\n\t`) y pasa A–Z a a–z (ASCII).
`find_substr_ci`: comparación *case-insensitive* convirtiendo el texto descomprimido a minúscula **durante la comparación**, contra la pista ya en minúscula.
Criterio de éxito: aparición de la pista en el plano $\Rightarrow$ combinación correcta de cifrado y compresión.

4 Problemas de desarrollo afrontados (d)

1. **Ambigüedad del cifrado**: orden desconocido entre rotación y XOR.
   Mitigación: probar ambos reversos (`XOR→ROR` y `XOR→ROL`) para cada (n, K).

2. **Endianness LZ78**: índice de 16 bits en LE o BE.
   Mitigación: función con bandera `littleEndian` y validaciones que descartan la incorrecta.

3. **Formato de compresión sin firma**: no existen cabeceras confiables.
   Mitigación: autodetección ordenada con validación por pista.

4. **Robustez de memoria**: riesgo de *overflows*, entradas inválidas, longitudes inconsistentes.
   Mitigación: `new (nothrow)`, realocaciones seguras, límites (12 dígitos en RLE ASCII; diccionario/pila en LZ78), liberación en errores.

5. **Falsos positivos**: flujos aleatorios pueden descomprimir parcialmente.
   Mitigación: validaciones estructurales estrictas y **pista** normalizada como verificación semántica final.

6. **Portabilidad/compilación**: `memcpy` sin `<cstring>` puede fallar según el toolchain.
   Mitigación: inclusión explícita de `<cstring>`.

5 Evolución de la solución y consideraciones de implementación (e)

5.1 Evolución

- **Propuesta inicial**: arquitectura modular, pruebas sistemáticas de cifrado/compresión y validación por pista; idea de *pistas derivadas* (usar el plano de un caso como pista de otro).

- **Versión final**: cinco descompresores consolidados; dos órdenes de descifrado; normalización robusta de pista; reporte uniforme de resultados; defensas de memoria.

5.2 Consideraciones prácticas

- **Convención de archivos**: el programa espera `EncriptadoX.txt`, `pistaX.txt` y escribe `salidaX.txt`.
- **Complejidad**: $7 \times 256 = 1792$ combinaciones por caso; hasta 5 intentos de descompresión por combinación; corte temprano cuando aparece la pista.
- **Memoria**: búferes de entrada/temporal del tamaño del cifrado; salida crece según compresión; LZ78 usa diccionario proporcional a #tripletas y una pila de 64 KiB.
- **Consola**: vista previa limitada a 1200 caracteres; bytes no imprimibles sustituidos por punto.

6 Conclusiones

La herramienta reconstruye mensajes sin requerir metadatos, combinando un descifrado ligero (XOR + rotaciones por byte, con orden desconocido) con autodetección de compresión (RLE/LZ78 con LE/BE) y validación semántica por pista normalizada. El diseño maximiza la **robustez**, mantiene una **eficiencia razonable** y asegura **trazabilidad**. Las pruebas verifican la detección de `LZ78_TRIPLET_BE` con $n = 3$ y $K =$ `0x5A` en casos reales y sintéticos, cumpliendo los objetivos del proyecto.