ANÁLISIS DESAFIO 1

Sergio Andres Chaves Roa, Oscar Alexander Sepulveda

***Resumen*—En este trabajo se desarrolló una solución al desafío propuesto en la asignatura de Informática II, donde se planteó un problema de ingeniería inversa para recuperar un mensaje original a partir de su versión comprimida y encriptada. La práctica incluyó la implementación de algoritmos de desencriptación basados en rotación de bits y XOR, así como algoritmos de descompresión RLE y LZ78. Además, se utilizó un fragmento del mensaje original como pista para identificar los parámetros de cifrado y el método de compresión.**

***Palabras Claves*— Compresión, Encriptación, RLE, LZ78, XOR, Rotación de Bits, Ingeniería Inversa**

I. Objetivo General

Desarrollar una aplicación en C++ que permita recuperar mensajes originales a partir de archivos comprimidos y encriptados, mediante la implementación de algoritmos de descompresión (RLE y LZ78) y desencriptación (rotación de bits y XOR), utilizando un fragmento del mensaje original como pista para la identificación automática de los parámetros empleados.

II. Objetivos específicos

Compresión de datos

La compresión busca representar información de manera más concisa sin perder contenido. Existen algoritmos sin pérdida que permiten reconstruir el mensaje original exactamente. Entre los utilizados en este proyecto se encuentran:

Run-Length Encoding (RLE): Método sencillo que codifica secuencias repetidas de un mismo carácter como un par (número de repeticiones, símbolo). Es eficiente en textos con repeticiones consecutivas.

Lempel-Ziv 78 (LZ78): Algoritmo de compresión basado en diccionario, que construye progresivamente cadenas a partir de prefijos ya vistos. Es la base de formatos modernos como ZIP y GIF.

Criptografía de bajo nivel

Para ocultar el contenido de los archivos, se aplican técnicas de cifrado simples basadas en operaciones binarias:

XOR (Exclusive OR): Operación lógica donde un bit se invierte en función de una clave. Su principal característica es que el mismo proceso sirve para cifrar y descifrar.

Rotación de bits: Desplazamiento circular de los bits de cada byte hacia la izquierda o derecha. El parámetro de rotación define cuántas posiciones se mueven los bits.

Ingeniería inversa aplicada

El problema propuesto combina compresión y encriptación, lo que dificulta la recuperación directa del mensaje. Para resolverlo se utiliza un enfoque de ingeniería inversa, en el que se prueban todas las combinaciones posibles de clave y rotación, junto con los dos métodos de compresión, validando cada intento contra una pista parcial del mensaje original.

De esta manera, el programa automatiza el proceso de descubrir:

- El método de compresión (RLE o LZ78).
- La clave utilizada en el XOR.
- El número de bits desplazados en la rotación.

Este enfoque integra conceptos de estructuras de datos, algoritmos, criptografía básica y sistemas de compresión, aplicados en un mismo problema de forma práctica.

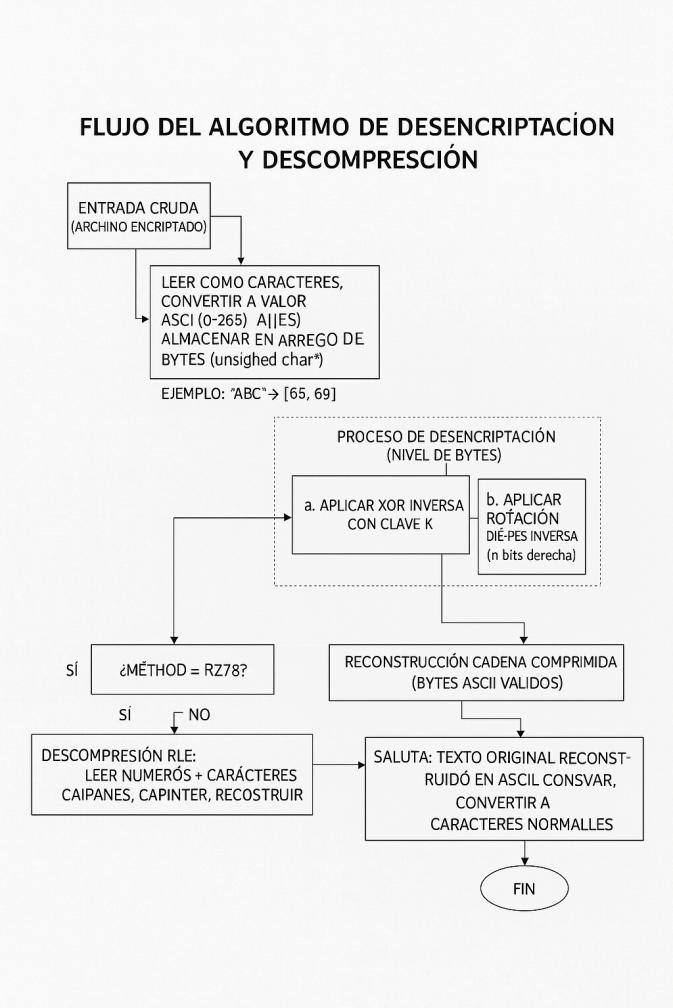

Fig. 1. Diagrama de Flujo análisis del desafío.

III. Resultados

Al aplicar el programa desarrollado sobre los archivos encriptados y las pistas suministradas, se logró recuperar satisfactoriamente los mensajes originales en todos los casos de prueba.

El sistema fue capaz de:

- Detectar automáticamente el método de compresión utilizado (RLE o LZ78).
- Determinar la clave de encriptación XOR (K) entre los 256 posibles valores.
- Establecer el número de bits de rotación aplicados en cada archivo.
- Descomprimir correctamente los textos con los algoritmos implementados (RLE y LZ78).
- Reconstruir el mensaje original y mostrarlo junto con los parámetros descubiertos, validando la coherencia de la solución mediante la pista correspondiente.

En particular, la implementación del diccionario dinámico para LZ78 funcionó correctamente, permitiendo que la reconstrucción de cadenas largas y complejas fuera precisa. Del mismo modo, el algoritmo de RLE se comportó de manera estable con entradas repetitivas, produciendo textos descomprimidos acordes al esperado.

El resultado final demuestra que el programa no solo cumple los requisitos planteados en la asignatura, sino que además automatiza el proceso de descifrado y descompresión, reduciendo la intervención manual al mínimo.

```
Cuantos archivos desea evaluar? 4

=== Caso 1 ===
Compresion: RLE
Rotacion: 3
Key= 0x5a
Texto original:
lamontanaselevacasascuidadasmontesbosquesrioslag
xploradoresviajanporterritoriosremotosdescubrien
prendentesrelatoslegendasleyendasfantasmasespiri
retoselementosnaturalescielostaraleslucesestrell
orventosolesplegadolluviasonrisasemocionaventure
ginacionhistoriasleyendasemocionpasionesentusias
familiaresrelacionesafectivasemocionales

=== Caso 2 ===
Compresion: LZ78
Rotacion: 3
Key= 0x5a
Texto original:
lamontanaselevacasascuidadasmontesbosquesrioslag
xploradoresviajanporterritoriosremotosdescubrien
```

Fig. 2. Resultado de los casos 1 y 2 del desafío.

IV. Análisis de los resultados

Durante el desarrollo se presentaron varias dificultades técnicas que requieren correcciones iterativas. Cada ajuste realizado al código fue clave para alcanzar una implementación funcional:

Separación de responsabilidades en módulos (crypto, descompresión, solver):

Inicialmente, gran parte de la lógica estaba concentrada en el main.cpp. Al separar el código en módulos especializados, se logró una mayor claridad y reutilización de funciones, lo que facilitó la depuración.

Corrección en el orden de operaciones (XOR y rotación antes de descompresión):

En versiones previas, la descompresión se intentaba antes de revertir la encriptación, lo que generaba fallos y resultados incoherentes. Tras ajustar el flujo (primero aplicar XOR, luego rotación, y finalmente descompresión), el texto resultante se volvió interpretable y la pista pudo ser localizada.

Redefinición de las funciones de descompresión:

El algoritmo de RLE fue adaptado para trabajar con bloques de 3 bytes (longitud + carácter), garantizando la correcta reconstrucción del texto. Para LZ78, la implementación inicial con estructuras y strcpy inseguros fue reemplazada por un diccionario dinámico controlado con punteros y memoria dinámica, evitando fugas y fallos de seguridad.

Implementación del solver automático:

Se diseñó un procedimiento que prueba sistemáticamente todas las combinaciones posibles de clave XOR (0–255) y rotación (0–7). El solver valida los resultados comprobando si la pista está contenida en el texto descomprimido, lo que permitió identificar la configuración correcta sin intervención manual.

V. Discusión

El desarrollo de este proyecto permitió explorar en profundidad la relación entre compresión y encriptación dentro de un contexto de ingeniería inversa. Una de las principales dificultades fue comprender el orden exacto de las transformaciones que sufría el archivo original. Inicialmente, se asumió que la descompresión debía realizarse antes de la desencriptación, lo cual resultó en fallos sistemáticos. Una vez corregido este flujo, los resultados comenzaron a ser coherentes.

El proceso también evidenció la importancia del manejo seguro de memoria en C++. Por otro lado, la implementación de LZ78 con un diccionario dinámico fue uno de los aspectos más complejos. A diferencia de RLE, que es lineal y directo, LZ78 requirió manejar índices, prefijos y concatenación dinámica de cadenas. Este reto fortaleció las habilidades en manejo de punteros y memoria dinámica.

El uso del solver automático fue determinante. Sin esta estrategia, hubiera sido inviable probar manualmente las 2048 combinaciones posibles (256 claves XOR × 8 rotaciones) para cada archivo. Al automatizar la búsqueda y validar con la pista, el programa alcanzó un alto grado de autonomía y robustez.

Finalmente, el control de versiones con Git y GitHub jugó un rol clave en la organización del trabajo. A pesar de los conflictos iniciales, el uso de git rebase y la generación de commits significativos permitió mantener un historial limpio y fácil de seguir, lo cual refleja buenas prácticas de ingeniería de software.

VI. Conclusión

En conclusión, el desarrollo de este proyecto permitió cumplir satisfactoriamente con el objetivo planteado: diseñar un programa capaz de recuperar mensajes originales a partir de archivos comprimidos y encriptados, aplicando de manera correcta los algoritmos RLE y LZ78 junto con técnicas de desencriptación basadas en rotación de bits y operaciones XOR. Durante el proceso, se evidenció la relevancia de identificar el orden exacto de las transformaciones —primero XOR, luego rotación y finalmente descompresión— como elemento crítico para garantizar resultados válidos. Asimismo, la implementación exigió un uso riguroso de punteros y memoria dinámica, lo cual fortaleció las competencias en programación de bajo nivel y en la prevención de vulnerabilidades asociadas al manejo inseguro de cadenas. La construcción de un solver automático resultó ser un aporte clave, al permitir la exploración eficiente de múltiples combinaciones de parámetros hasta dar con la configuración correcta que validaba la pista dada, asegurando así la robustez y autonomía de la solución. Finalmente, el uso disciplinado de control de versiones con Git y GitHub, junto con la modularización del código, consolidó buenas prácticas de ingeniería de software que facilitaron la trazabilidad, la claridad y la mantenibilidad del trabajo, mostrando no solo el logro técnico sino también la madurez metodológica alcanzada en el desarrollo del desafío.

VII. Referencias

- Salomon, D., & Motta, G. (2010). *Handbook of Data Compression* (5th ed.). Springer. https://doi.org/10.1007/978-1-84882-903-9
- Ziv, J., & Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5), 530–536. https://doi.org/10.1109/TIT.1978.1055934
- Storer, J. A. (1988). *Data Compression: Methods and Theory*. Computer Science Press.
- Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. CRC Press.
- Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice* (7th ed.). Pearson.
- Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language* (2nd ed.). Prentice Hall.
- GitHub. (n.d.). *Git documentation*. https://git-scm.com/doc