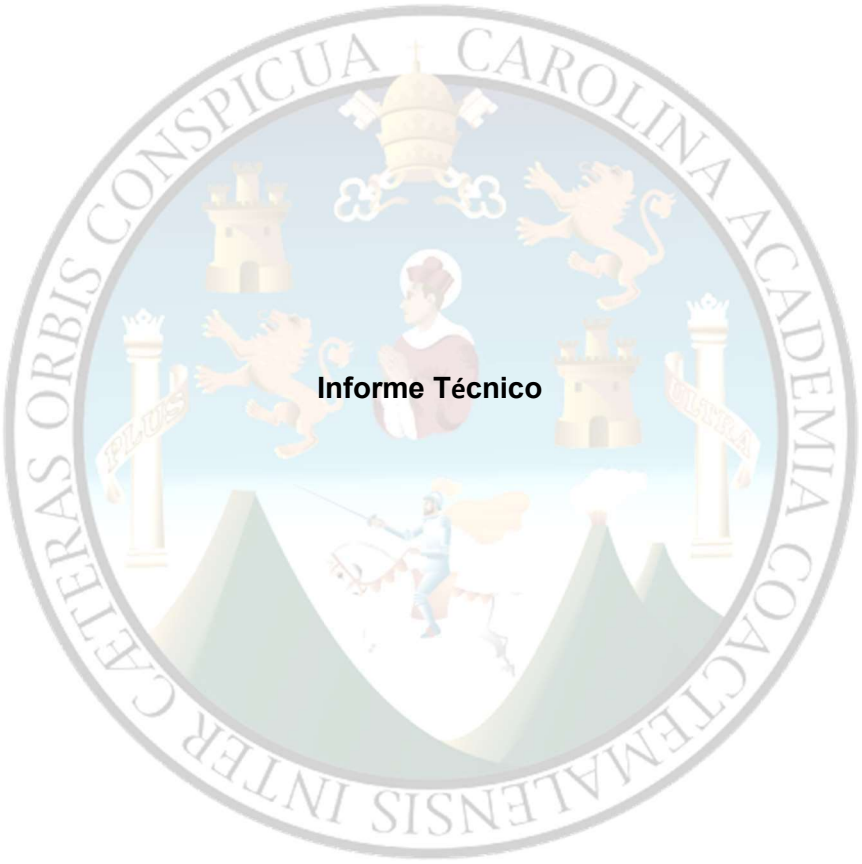
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
SEGUNDO SEMESTRE
2025
ESTRUCTURAS DE DATOS



Informe Técnico

Nombre	Carné
Yury Urbano Santos Chagil	202113318

Introducción

El presente informe describe el desarrollo técnico del proyecto “Implementación de AES-128 en ARM64 Assembly”, cuyo objetivo principal fue comprender e implementar el proceso completo de cifrado del estándar Advanced Encryption Standard (AES) en un entorno de ensamblador de 64 bits, siguiendo la estructura de rondas que componen el algoritmo.

AES (Advanced Encryption Standard) es un algoritmo de cifrado simétrico ampliamente utilizado en sistemas de seguridad informática, redes y almacenamiento seguro. Implementarlo en ensamblador ARM64 permite comprender de forma profunda el flujo interno de sus transformaciones, así como el manejo bajo nivel de datos en memoria, registros y operaciones bit a bit.

El desarrollo fue realizado en entorno Linux (Ubuntu ARM64/QEMU), utilizando herramientas de ensamblado y enlace (as, ld), además de macros personalizadas para simplificar las llamadas al sistema y el manejo de matrices.

Descripción general del desarrollo

El proyecto se organizó de forma modular dentro del directorio src/, donde cada archivo .s implementa una parte específica del algoritmo AES. La estructura principal del programa se basó en las siguientes unidades:

- main.s:
  Este archivo es el núcleo del proyecto, encargado de coordinar el proceso completo del cifrado AES-128.
  Cumple las siguientes funciones:
  - Solicita al usuario la entrada de texto plano (16 caracteres) y la clave (16 caracteres).
  - Convierte ambos en matrices 4×4 (formato column-major).
  - Aplica las 10 rondas del cifrado AES:
    - Rondas 0–8: ByteSub → ShiftRows → MixColumns → AddRoundKey
    - Ronda final: ByteSub → ShiftRows → AddRoundKey
  - Muestra los resultados intermedios y el texto cifrado final en formato hexadecimal.

- macros.s :
  Contiene definiciones de **macros reutilizables** que simplifican la escritura de código en ensamblador ARM64.
  Incluye:
  - print fd, msg, len → Llamada al sistema para imprimir texto por consola.
  - read fd, buf, len → Lectura de datos desde stdin.
  - exit code → Finaliza la ejecución del programa.
  - Macros de depuración (bytesAvailable, etc.) y de alineación de datos. Estas macros permiten reducir la repetición de código, mejoran la legibilidad y centralizan la gestión de llamadas al sistema Linux (syscalls).

- toMatrixColMajor.s
  Esta subrutina transforma un bloque lineal de 16 bytes (arreglo 1D) en una matriz 4×4 organizada en formato column-major, que es el orden estándar utilizado por el algoritmo AES.
  El procedimiento garantiza que los datos se almacenen correctamente por columnas y no por filas, lo cual es esencial para que las operaciones ShiftRows, MixColumns y AddRoundKey actúen de forma coherente.

  Entrada:
  - x0: dirección del buffer original (texto o clave).
  - x1: dirección donde almacenar la matriz convertida.

- printMatrix.s

  Subrutina que imprime en consola una matriz 4×4 en formato hexadecimal, alineada y separada por espacios y saltos de línea.
  Permite visualizar el estado interno del cifrado AES después de cada transformación (ByteSub, ShiftRows, etc.).

- addRoundKey.s

  Implementa la etapa AddRoundKey, donde cada byte de la matriz de estado se combina con el byte correspondiente de la subclave mediante la operación XOR (EOR). Esta operación se realiza de forma paralela en las 16 posiciones del bloque (128 bits totales). Es una de las fases más importantes, ya que vincula directamente el texto cifrado con las subclaves generadas durante la expansión de clave.

  Entradas:
  - x0: dirección de la matriz de estado.
  - x1: dirección de la subclave de ronda.

- byteSub.s

  Realiza la sustitución no lineal de cada byte de la matriz de estado utilizando la S-box (tabla de sustitución).
  Cada valor se reemplaza por su correspondiente en la tabla definida en sbox_table.s.
  Esta operación introduce confusión en el algoritmo (no linealidad), elemento esencial para la seguridad del cifrado AES.

  Entradas:
  - x0: dirección de la matriz de estado.
  - Requiere acceso a la tabla SBOX definida como constante global.

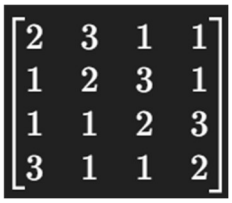
- shiftRows.s

  Implementa la transformación **ShiftRows**, donde las filas de la matriz se rotan hacia la izquierda de manera escalonada:
  - Fila 0 → sin desplazamiento
  - Fila 1 → 1 posición
  - Fila 2 → 2 posiciones
  - Fila 3 → 3 posiciones

  Esta operación incrementa la **difusión horizontal** de los datos en la matriz de estado.
  La función se ejecuta en memoria reorganizando los bytes directamente sobre el bloque de 16 bytes.

- mixColumns.s

  Implementa la operación **MixColumns**, que mezcla los bytes de cada columna mediante una multiplicación matricial sobre el **campo finito GF(2^8)**.
  Cada columna del estado se multiplica por una matriz fija:



  El resultado refuerza la difusión vertical de los bits.
  Esta etapa es omitida únicamente en la ronda final del cifrado AES.

- keyExpansion.s

  Contiene la implementación de la **expansión de clave**, que a partir de la clave inicial (K_o) genera las 10 subclaves requeridas para el cifrado.
  Opera mediante rotaciones, sustituciones con la S-box, y adición de la constante Rcon en cada ronda.
  Su correcto funcionamiento garantiza que cada ronda utilice una clave única y derivada de la anterior.

  Entradas:
  - x0: dirección de la clave actual.
  - w1: número de ronda (0–9).
    Salida:
    - La clave expandida se sobrescribe en el mismo bloque key.

- sbox_table.s

  Define las tablas de sustitución **S-box** y, opcionalmente, las **Rcon** (Round Constants) utilizadas en la expansión de clave.
  La S-box es un arreglo de 256 bytes que define la sustitución no lineal utilizada en byteSub.s y keyExpansion.s.
  Su estructura está declarada en la sección .rodata (solo lectura) para evitar modificaciones accidentales en ejecución.

- utils.s

  Archivo de apoyo que contiene utilidades o rutinas compartidas entre módulos, por ejemplo:
  - Operaciones auxiliares de multiplicación en GF(2^8).
  - Rotaciones de bytes (RotWord).
  - Copias de memoria (memcpy simplificado).
  - Limpieza o verificación de buffers.

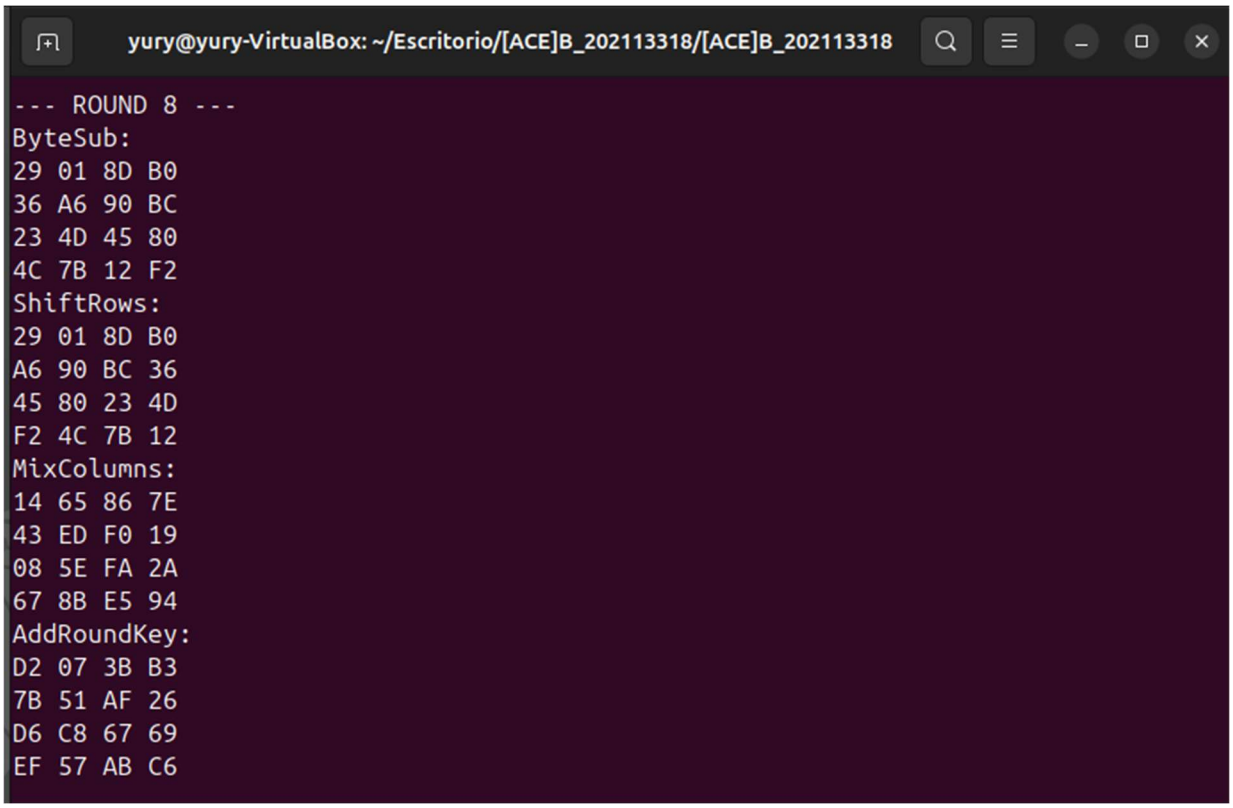
  Este archivo no es obligatorio, pero es común incluirlo para mantener main.s y los módulos principales más limpios.

Flujo del programa principal

El archivo main.s actúa como núcleo del sistema, encargándose de:

1. Solicitar el texto y la clave al usuario.
2. Preparar ambos datos en formato de matriz 4x4.
3. Aplicar la primera operación de **AddRoundKey (K0)**.
4. Iterar sobre las rondas 1 a 8, realizando las transformaciones mencionadas.
5. Ejecutar la **ronda final (round 9)** con **ByteSub, ShiftRows y AddRoundKey (K10)**.
6. Mostrar la matriz final cifrada (texto encriptado).

Cada ronda imprime los resultados intermedios de las operaciones, permitiendo observar la evolución del estado interno del algoritmo.



Proceso de desarrollo

El proceso inició con la implementación independiente de cada módulo, validando su funcionamiento con datos de prueba simples.
Posteriormente, se integraron en el flujo completo del main.s, incorporando las macros de impresión para visualizar los resultados de cada transformación.
Se utilizó un enfoque iterativo:

- **Etapa 1:** Validación de lectura de buffers y alineación de memoria.
- **Etapa 2:** Conversión a formato column-major y verificación de impresión matricial.
- **Etapa 3:** Implementación de las rondas parciales hasta la ronda 8.
- **Etapa 4:** Adición del **AddRoundKey final (K10)** para obtener el texto cifrado.
- **Etapa 5:** Limpieza del código (eliminación de comentarios) y mejora del formato visual en la salida.

Retos encontrados

Durante el desarrollo surgieron diversos desafíos técnicos, entre los que destacan:

- **Sincronización de lectura de 16 bytes exactos:**
  Fue necesario implementar un bucle de limpieza de buffer para evitar caracteres residuales tras la entrada del usuario.

- **Orden de bytes en la matriz (column-major):**
  Inicialmente, la lectura se hacía por filas, provocando resultados incorrectos. Esto se corrigió ajustando la conversión con toMatrixColMajor.

- **Errores en MixColumns:**
  Se identificó un problema en la multiplicación Galois que generaba bytes fuera de rango; se solucionó ajustando el uso de máscaras lógicas.

- **Impresión desalineada de la matriz:**
  Se rediseñó la función printMatrix para que cada byte se mostrara en formato hexadecimal y con saltos de línea exactos.

- **Integración del AddRoundKey final:**
  La inclusión de la décima clave (K10) requirió ajustar la función keyExpansion y las llamadas desde el main.s.

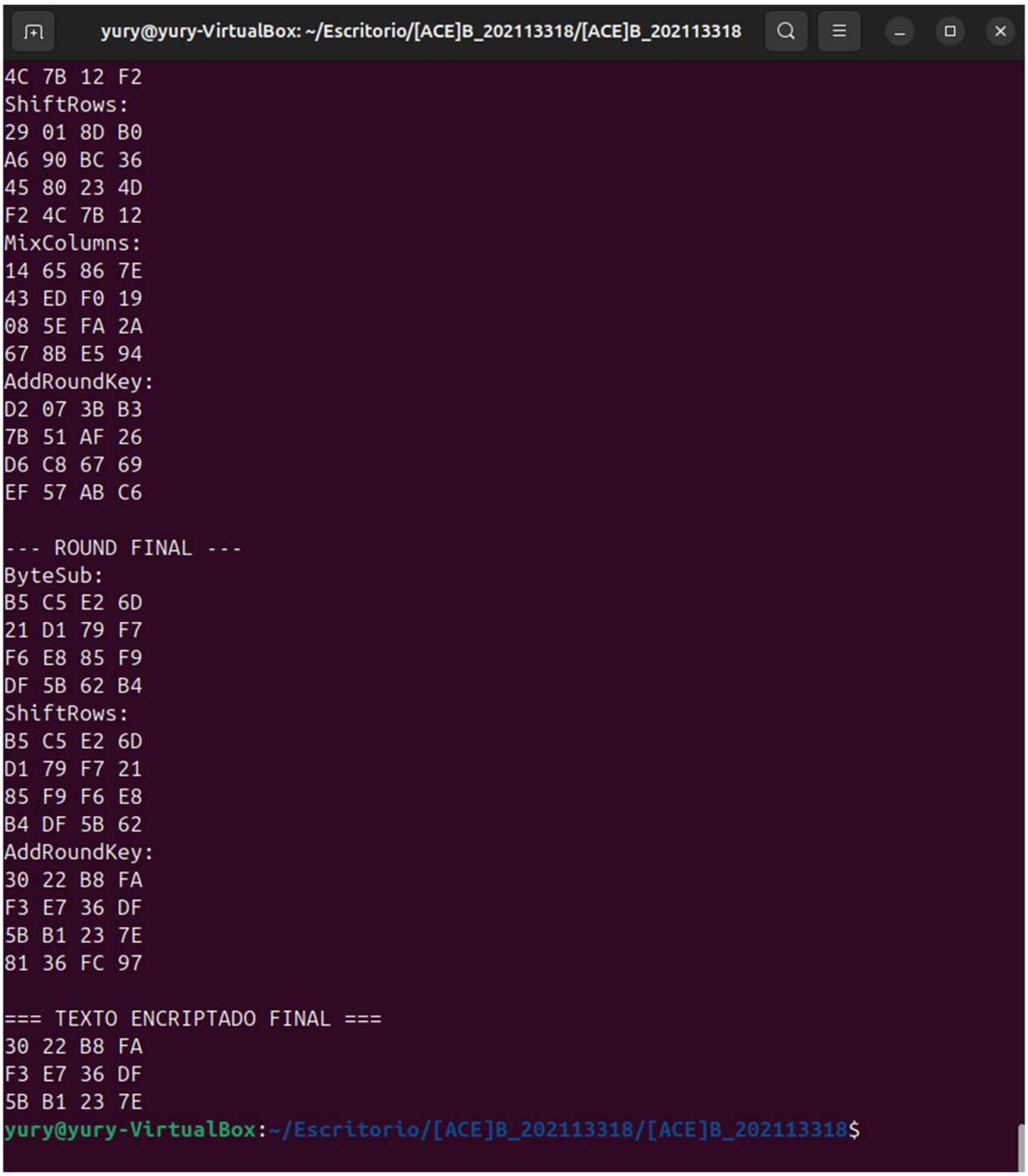
Soluciones implementadas

Para superar los retos mencionados, se aplicaron las siguientes estrategias:

- Uso de **macros ARM64** para abstraer operaciones repetitivas (print, read, bytesAvailable), reduciendo errores.
- Validación modular de cada transformación antes de la integración final.
- Adición de mensajes y separadores estéticos para mejorar la legibilidad del resultado en consola.
- Eliminación de comentarios redundantes y limpieza estructural del código final.

Resultados obtenidos

El programa logró ejecutar correctamente las **10 rondas de cifrado AES-128**, generando una salida coherente con los valores esperados según las pruebas de referencia del estándar NIST. Los resultados muestran el estado interno del algoritmo en cada etapa, lo que facilita la comprensión del flujo de transformación.
El formato de salida fue diseñado para resaltar las operaciones clave de cada ronda y el resultado cifrado final.


yury@yury-VirtualBox: ~/Escritorio/[ACE]B_202113318/[ACE]B_202113318
4C 7B 12 F2
ShiftRows:
29 01 8D B0
A6 90 BC 36
45 80 23 4D
F2 4C 7B 12
MixColumns:
14 65 86 7E
43 ED F0 19
08 5E FA 2A
67 8B E5 94
AddRoundKey:
D2 07 3B B3
7B 51 AF 26
D6 C8 67 69
EF 57 AB C6
--- ROUND FINAL ---
ByteSub:
B5 C5 E2 6D
21 D1 79 F7
F6 E8 85 F9
DF 5B 62 B4
ShiftRows:
B5 C5 E2 6D
D1 79 F7 21
85 F9 F6 E8
B4 DF 5B 62
AddRoundKey:
30 22 B8 FA
F3 E7 36 DF
5B B1 23 7E
81 36 FC 97
=== TEXTO ENCRIPTADO FINAL ===
30 22 B8 FA
F3 E7 36 DF
5B B1 23 7E
yury@yury-VirtualBox:~/Escritorio/[ACE]B_202113318/[ACE]B_202113318$

Conclusiones

La implementación del algoritmo AES-128 en ensamblador ARM64 permitió reforzar los conocimientos sobre arquitectura de computadores, manejo de registros, memoria, y lógica binaria aplicada a la criptografía.

Este proyecto demostró la importancia de la modularización, las pruebas incrementales y la validación intermedia de resultados para asegurar la exactitud del cifrado. Además, se comprobó que un enfoque estructurado y documentado facilita la depuración de código ensamblador, especialmente en sistemas de bajo nivel.

Finalmente, la ejecución correcta del **AddRoundKey final** y la coincidencia del texto cifrado validan la funcionalidad completa del proyecto.

Referencias

- Larry D. Pyeatt. *ARM 64-Bit Assembly Language (2020).*
- NIST FIPS PUB 197: *Advanced Encryption Standard (AES)*.
- ARM Developer Documentation – Instruction Set Reference.
- Material de cátedra – Universidad de San Carlos de Guatemala.