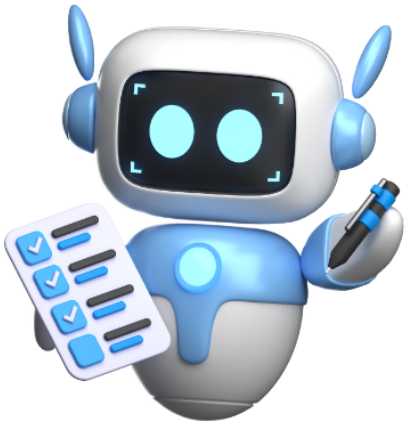
Lab de Arquitectura de Computadores y Ensambladores 1
Grupo 12

Informe Técnico

Algoritmo de Desencriptación

AES-128



Integrantes:
201610975 - Michael Cristian Itzep Ixcayau
202104727 - López Suy, Luigi Anderson
202202055 - González Espinoza, Fernando Andhré
202300652 - González Pereira, Vasti Abigail

Contents

Resumen:

La criptografía ha sido de gran importancia a lo largo de la historia, incluso decidió el curso de una guerra, específicamente en la segunda guerra mundial en donde se usó para descifrar la máquina Enigma y con ello plantear estrategias que llevaron a la ruina a los alemanes. Por lo que el aprendizaje de las bases de criptografía es significativo para los desarrolladores, por lo que se realiza el algoritmo de desencriptación simétrica AES de 128 que utiliza el lenguaje ensamblador ARM64. Este tiene como objetivo traducir un proceso de descifrado complejo y sus operaciones inversas con el uso de un lenguaje de bajo nivel, manejando bytes para recuperar el texto original a través de cadenas cifradas.

En el archivo inverse_decrypt_verbose_orderfix_fixed0.s

Se inicia declarando las variables en .data

```
5     .section .data
6         msg_last_key: .asciz "Ingrese la última clave (ronda 10): "
7             lenMsgLastKey = . - msg_last_key
8         key_err_msg: .asciz "Error: Valor de clave incorrecto\n"
9             lenKeyErr = . - key_err_msg
10        newline: .asciz "\n"
11        msg_inverse_title: .asciz " EXPANSIÓN INVERSA DE CLAVES"
12            lenMsgInvTitle = . - msg_inverse_title
13        msg_round_key: .asciz "\nClave Ronda "
14            lenMsgRoundKey = . - msg_round_key
15        msg_colon: .asciz ":\n"
16        msg_original_key: .asciz "CLAVE ORIGINAL (Ronda 0) "
17            lenMsgOriginal = . - msg_original_key
18        msg_ciphertext: .asciz "\nIngrese el texto cifrado (32 hex / 16 bytes): "
19            lenMsgCipher = . - msg_ciphertext
20        msg_cipher_state: .asciz "\nCIPHERTEXT cargado (hex 4x4):\n"
21            lenMsgCipherState = . - msg_cipher_state
22        msg_plain_title: .asciz "\nPLAINTEXT (hex 4x4):\n"
23            lenMsgPlainTitle = . - msg_plain_title
24        msg_state_title: .asciz "\nESTADO (hex 4x4):\n"
25            lenMsgStateTitle = . - msg_state_title
26        msg_round_hdr: .asciz "\n--- RONDA "
27            lenMsgRoundHdr = . - msg_round_hdr
28        msg_after_isr: .asciz " despues InvShiftRows\n"
29            lenMsgAfterISR = . - msg_after_isr
30        msg_after_isb: .asciz " despues InvSubBytes\n"
31            lenMsgAfterISB = . - msg_after_isb
32        msg_after_ark: .asciz " despues AddRoundKey\n"
33            lenMsgAfterARK = . - msg_after_ark
34        msg_after_imc: .asciz " despues InvMixColumns\n"
35            lenMsgAfterIMC = . - msg_after_imc
```

Las variables no inicializadas en .bss

Se reservan 176 bytes para expandedKeys (11 subclaves de 16 bytes cada una) y 16 bytes para el cipherState.

Se definen macros print y read que envuelven las llamadas al sistema (syscalls) 64 (write) y 63 (read) de Linux sobre ARM64 para facilitar la entrada/salida.

```
37      .section .bss
38          lastKey: .space 16, 0          // Última clave (ronda 10)
39          expandedKeys: .space 176, 0    // Todas las subclaves (11 claves de 16 bytes)
40          buffer: .space 256, 0
41          tempWord: .space 4, 0
42          cipherState: .space 16, 0      // Estado (ciphertext) 16 bytes
```

Conversión Hexadecimal

- convertHexKey: Procesa la cadena de entrada y llena la memoria con valores binarios.
- hex_char_to_nibble: Convierte un solo carácter ASCII a un valor de 0 a 15, manejando tanto números como letras (A-F).

```
62      // Función para convertir clave hexadecimal
63      .type convertHexKey, %function
64      .global convertHexKey
65      convertHexKey:
66          stp x29, x30, [sp, #-16]!
67          stp x19, x20, [sp, #-16]!
68          mov x29, sp
69          read 0, buffer, 33
70          ldr x1, =buffer
71          ldr x2, =lastKey
72          mov x3, #0
73          mov x11, #0
74      convert_hex_loop:
75          cmp x3, #16
76          b.ge convert_hex_done
77      skip_non_hex:
78          ldrb w4, [x1, x11]
79          cmp w4, #0
80          b.eq convert_hex_done
81          cmp w4, #10
82          b.eq convert_hex_done
83          bl is_hex_char
84          cmp w0, #1
85          b.eq process_hex_pair
86          add x11, x11, #1
87          b skip_non_hex
```

```
88      process_hex_pair:
89          ldrb w4, [x1, x11]
90          add x11, x11, #1
91          bl hex_char_to_nibble
92          lsl w5, w0, #4
93          ldrb w4, [x1, x11]
94          add x11, x11, #1
95          bl hex_char_to_nibble
96          orr w5, w5, w0
97          strb w5, [x2, x3]
98          add x3, x3, #1
99          b convert_hex_loop
100     convert_hex_done:
101         ldp x19, x20, [sp], #16
102         ldp x29, x30, [sp], #16
103         ret
104         .size convertHexKey, (. - convertHexKey)
105
106     is_hex_char:
107         cmp w4, #'0'
108         b.lt not_hex
109         cmp w4, #'9'
110         b.le is_hex
111         orr w4, w4, #0x20
112         cmp w4, #'a'
113         b.lt not_hex
114         cmp w4, #'f'
115         b.le is_hex
116     not_hex:
117         mov w0, #0
118         ret
```

Función para imprimir una clave (formato 4x4)

```
211     // Función para imprimir una clave (formato 4x4)
212     .type printKey, %function
213     printKey:
214         stp x29, x30, [sp, #-32]!
215         mov x29, sp
216         str x19, [sp, #16]
217         str x20, [sp, #24]
218         mov x19, x0
219         mov x20, #0
220     print_row_loop:
221         cmp x20, #4
222         b.ge print_key_done
223         mov x21, #0
224     print_col_loop:
225         cmp x21, #4
226         b.ge print_row_end
227         mov x2, #4
228         mul x2, x21, x2
229         add x2, x2, x20
230         ldrb w0, [x19, x2]
231         bl print_hex_byte
232         add x21, x21, #1
233         b print_col_loop
234     print_row_end:
235         print 1, newline, 1
236         add x20, x20, #1
237         b print_row_loop
238     print_key_done:
239         print 1, newline, 1
240         ldr x19, [sp, #16]
241         ldr x20, [sp, #24]
242         ldp x29, x30, [sp], #32
```

Expansión Inversa de Claves

Lógica: Utiliza las funciones rotWord (rotación de bytes) y subWord (sustitución por S-Box) junto con la constante Rcon.

Se calcula el offset de cada palabra (W[i]) multiplicando el índice por 4 bytes.

```
338     // Función principal: expansión inversa de claves
339     // El proceso inverso es:
340     // Para cada palabra i desde 43 hasta 4:
341     //   Si i mod Nk == 0:
342     //     W[i-Nk] = W[i] XOR SubWord(RotWord(W[i-1])) XOR Rcon[i/Nk - 1]
343     //   Sino:
344     //     W[i-Nk] = W[i] XOR W[i-1]
345     .type inverseKeyExpansion, %function
346     .global inverseKeyExpansion
347     inverseKeyExpansion:
348         stp x29, x30, [sp, #-96]!
349         mov x29, sp
350         str x19, [sp, #16]
351         str x20, [sp, #24]
352         str x21, [sp, #32]
353         str x22, [sp, #40]
354         str x23, [sp, #48]
355         str x24, [sp, #56]
356         str x25, [sp, #64]
357         str x26, [sp, #72]
358         str x27, [sp, #80]
359         str x28, [sp, #88]
360
361         ldr x19, =lastKey        // Puntero a última clave
362         ldr x20, =expandedKeys   // Puntero a claves expandidas
363         ldr x21, =Rcon           // Puntero a Rcon
```

Transformaciones del Estado

AddRoundKey:

Aplica una operación XOR bit a bit entre el estado actual y la subclave de la ronda correspondiente. Es la única función que es su propia inversa.

```
602     AddRoundKey:
603         mov x2, #0
604     1:  cmp x2, #16
605         b.ge 2f
606         ldrb w3, [x0, x2]
607         ldrb w4, [x1, x2]
608         eor w3, w3, w4
609         strb w3, [x0, x2]
610         add x2, x2, #1
611         b 1b
612     2:  ret
613         .size AddRoundKey, (. - AddRoundKey)
614
615     // InvSubBytes: state[i] = InvSbox[state[i]]
616     // x0 = state (16B)
617     .type InvSubBytes, %function
618     .global InvSubBytes
619     InvSubBytes:
620         stp x29, x30, [sp, #-16]!
621         mov x29, sp
622         ldr x1, =InvSbox
623         mov x2, #0
624     1:  cmp x2, #16
625         b.ge 2f
626         ldrb w3, [x0, x2]
627         uxtw x3, w3
628         ldrb w4, [x1, x3]
629         strb w4, [x0, x2]
630         add x2, x2, #1
631         b 1b
```

InvSubBytes:

Utiliza la tabla InvSbox (almacenada en .rodata) para realizar una sustitución no lineal de cada byte del estado.

```
615     // InvSubBytes: state[i] = InvSbox[state[i]]
616     // x0 = state (16B)
617     .type InvSubBytes, %function
618     .global InvSubBytes
619     InvSubBytes:
620         stp x29, x30, [sp, #-16]!
621         mov x29, sp
622         ldr x1, =InvSbox
623         mov x2, #0
624     1:  cmp x2, #16
625         b.ge 2f
626         ldrb w3, [x0, x2]
627         uxtw x3, w3
628         ldrb w4, [x1, x3]
629         strb w4, [x0, x2]
630         add x2, x2, #1
631         b 1b
632     2:  ldp x29, x30, [sp], #16
633         ret
634         .size InvSubBytes, (. - InvSubBytes)
635
```

InvShiftRows:

Mueve los bytes de las filas de la matriz de estado. En la desencriptación, las rotaciones son hacia la **derecha**: Fila 1 (1 posición), Fila 2 (2 posiciones), Fila 3 (3 posiciones).

```
642     .type InvShiftRows, %function
643     .global InvShiftRows
644     InvShiftRows:
645         // Row1 indices: 1,5,9,13 -> [13,1,5,9]
646         ldrb w1, [x0, #1]
647         ldrb w2, [x0, #5]
648         ldrb w3, [x0, #9]
649         ldrb w4, [x0, #13]
650         strb w4, [x0, #1]
651         strb w1, [x0, #5]
652         strb w2, [x0, #9]
653         strb w3, [x0, #13]
654
655         // Row2 indices: 2,6,10,14 -> [10,14,2,6]
656         ldrb w1, [x0, #2]
657         ldrb w2, [x0, #6]
658         ldrb w3, [x0, #10]
659         ldrb w4, [x0, #14]
660         strb w3, [x0, #2]
661         strb w4, [x0, #6]
662         strb w1, [x0, #10]
663         strb w2, [x0, #14]
664
665         // Row3 indices: 3,7,11,15 -> right3 == left1 => [7,11,15,3]
666         ldrb w1, [x0, #3]
667         ldrb w2, [x0, #7]
668         ldrb w3, [x0, #11]
669         ldrb w4, [x0, #15]
670         strb w2, [x0, #3]
671         strb w3, [x0, #7]
672         strb w4, [x0, #11]
```

Aritmética de Campos de Galois

InvMixColumns:

xtime: Realiza la multiplicación por {02}. Si el bit más significativo es 1, se aplica un XOR con el polinomio irreducible 0x1B.

gf_mul_const: Calcula multiplicaciones por constantes {09, 0B, 0D, 0E} combinando llamadas a xtime y operaciones XOR.

```
677     // xtime: multiplica por 2 en GF(2^8) con polinomio 0x11B
678     // w0 = a (byte), devuelve w0
679     .type xtime, %function
680     xtime:
681         and w1, w0, #0x80
682         lsl w0, w0, #1
683         and w0, w0, #0xFF
684         cbz w1, 1f
685         mov w2, #0x1B
686         eor w0, w0, w2
687     1:  ret
688         .size xtime, (. - xtime)
```

Ciclo Principal de Desencriptación

La función AES128_DecryptBlock coordina el flujo de datos. Sigue el estándar de aplicar primero una AddRoundKey inicial, luego 9 rondas completas y una ronda final sin InvMixColumns.

```
890     round_loop:
891         cmp w21, #1
892         b.lt final_round
893         // DEBUG: inicio de ronda r
894         print 1, msg_round_hdr, lenMsgRoundHdr
895         mov w0, w21
896         bl printRoundNumber
897 
898         mov x0, x19
899         bl InvShiftRows
900         // DEBUG: estado despues InvShiftRows
901         print 1, msg_after_isr, lenMsgAfterISR
902         print 1, msg_state_title, lenMsgStateTitle
903         mov x0, x19
904         mov w22, w21
905         bl printKey
906         mov w21, w22
907         mov x0, x19
908         bl InvSubBytes
909         // DEBUG: estado despues InvSubBytes
910         print 1, msg_after_isb, lenMsgAfterISB
911         print 1, msg_state_title, lenMsgStateTitle
912         mov x0, x19
913         mov w22, w21
914         bl printKey
915         mov w21, w22
916 
917         // AddRoundKey con round r (offset r*16)
918         mov x0, x19
919         uxtw x2, w21
920         lsl x2, x2, #4       // *16
921         add x1, x20, x2
```

Tablas de Referencia

SBox

```
.global Sbox
Sbox:
    .byte 0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76
    .byte 0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0
    .byte 0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15
    .byte 0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75
    .byte 0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84
    .byte 0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf
    .byte 0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8
    .byte 0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2
    .byte 0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73
    .byte 0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb
    .byte 0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79
    .byte 0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08
    .byte 0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a
    .byte 0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e
    .byte 0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf
    .byte 0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
```

InvSBox:

Tablas de 256 bytes para sustitución.

```
.global InvSbox
InvSbox:
    .byte 0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb
    .byte 0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb
    .byte 0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e
    .byte 0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25
    .byte 0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92
    .byte 0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84
    .byte 0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06
    .byte 0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b
    .byte 0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73
    .byte 0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e
    .byte 0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b
    .byte 0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4
    .byte 0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f
    .byte 0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef
    .byte 0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61
    .byte 0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
```

InvMixMat:

Matriz de constantes para la mezcla de columnas.

```
40
41      .global InvMixMat
42      InvMixMat:
43          .byte 0x0e, 0x0b, 0x0d, 0x09, 0x09, 0x0e, 0x0b, 0x0d, 0x0d, 0x09, 0x0e, 0x0b, 0x0b, 0x0d, 0x09, 0x0e
44
```

Rcon:

Constantes de ronda para la expansión de clave.

```
.global Rcon
Rcon:
    .byte 0x01, 0x00, 0x00, 0x00
    .byte 0x02, 0x00, 0x00, 0x00
    .byte 0x04, 0x00, 0x00, 0x00
    .byte 0x08, 0x00, 0x00, 0x00
    .byte 0x10, 0x00, 0x00, 0x00
    .byte 0x20, 0x00, 0x00, 0x00
    .byte 0x40, 0x00, 0x00, 0x00
    .byte 0x80, 0x00, 0x00, 0x00
    .byte 0x1B, 0x00, 0x00, 0x00
    .byte 0x36, 0x00, 0x00, 0x00
```