Departamento de Matemática y Ciencia de la Computación

Licenciatura en Ciencia de la Computación

Laboratorio N°2

Algoritmo AES

Introducción a la Criptografía Moderna
Profesor: Rodrigo Abarzúa

Matías Daza A.
María Fernanda Maldonado P.

Índice general

1. AES: Advanced Encryption Standard

AES opera sobre una matriz de 4x4 bytes. Mediante un algoritmo se reordenan los distintos bytes de la matriz. El cifrado es de clave simétrica, por lo que la misma clave aplicada en el cibrado se aplica para el descifrado.

De manera general el algoritmo AES aplica las capas de Byte Substitution Layer y la capa de difusión. AES es rápido tanto en software como en hardware, es relativamente fácil de implementar, y requiere poca memoria.

1.1. Algoritmos

Definicón de matrices utilizadas:

Matriz constante de polinomios:

$$MatPol[4][4] = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

Matriz valor hexadecimal a binario:

$$HextoBin[4][4] = \begin{pmatrix} \text{"0000"} & \text{"0001"} & \text{"0010"} & \text{"0011"} \\ \text{"0100"} & \text{"0101"} & \text{"0110"} & \text{"0111"} \\ \text{"1000"} & \text{"1001"} & \text{"1010"} & \text{"1011"} \\ \text{"1100"} & \text{"1101"} & \text{"1110"} & \text{"1111"} \end{pmatrix}$$

Algorithm 1 Algoritmo principal

Require: time Semilla
Ensure: Matriz A[128]

```
int i, A[128],
random semilla
for (i ← 1 to 128) do
    if (rand() %2 == 0) then
        A[i] ← 0
    else
        A[i] ← 1
    end if
end for
Matriz de entrada:
i ← 1
while (i ≤ 128) do
    Mostrar A[i]
end while

Llamada a función hexadecimal
Hexadecimal(A,indice)
```

Algorithm 2 Algoritmo función Hexadecimal

Require: Matriz A[128], indice
Ensure: Matriz Baux[32]

Convertir los valores a hexadecimal
int i, j, L, R, Baux[32]
$j \leftarrow 0$
$i \leftarrow indice \times 8$
while $(i \leq 128)$ **do**
    $R = A[i] \times 1 + A[i-1] \times 2 + A[i-2] \times 4 + A[i-3] \times 8$
    $L = A[i-4] \times 1 + A[i-5] \times 2 + A[i-6] \times 4 + A[i-7] \times 8$
    Mostrar(L,R)

    $Baux[j] \leftarrow L$
    $j++$
    $Baux[j] \leftarrow R$
    $j++$
    $indice \leftarrow indice + 1$
    $aux[p] \leftarrow 0$
    $i \leftarrow indice \times 8$
end while

Llamada a función ShiftRow
ShiftRow(Baux)

Algorithm 3 Algoritmo función ShiftRow

Require: Matriz Baux[32]
Ensure: Matriz BSalida [32]

int i, j, x, BEntrada[32], BSalida[32]
Para la fila 1 de Matriz Baux
Copiar los valores en BSalida
Para la fila 2 de Matriz Baux
Mover una posición a la izquierda y copiar los valores en BSalida
Para la fila 3 de Matriz Baux
Mover dos posiciones a la izquierda y copiar los valores en BSalida
Para la fila 4 de Matriz Baux
Mover tres posiciones a la izquierda y copiar los valores en BSalida

Llamada a función MixColumn
MixColumn(BSalida);

Algorithm 4 Algoritmo función MixColumn

Require: Matriz BSalida[32]
Ensure: Matriz C[16]

```
int i, j, k, l, r, m, n, aux, inicio, fin, ind=0, B[128]
int *p1, *p2, *p3, *p4, *pres
char str[5], bin[134]
int indB[16] = {0,32,64,96,8,40,72,104,16,48,80,112,24,56,88,120}
i ← 0
j ← 0
while (i ≤ 32) do
    l ← BSalida[i]
    i++
    r ← BSalida[i]
    i++
    str ← hexToBin(l) CONCATENADO hexToBin(r)
    bin CONCATENADO str
end while

for (i ← 0; i ≤ 127; i++) do
    B[i] ← bin[i] − 48
end for
ind ← k ← aux ← m ← n ← 0
while (k < 16) do
    while (m ≤ 3) do
        while (n ≤ 3) do
            aux ← MatPol[m][n]
            n++
            inicio ← indB[k]
            fin ← inicio + 7
            k++
            p1 ← getpolinomio(B, inicio, fin, aux)
            inicio ← indB[k], fin ← inicio + 7
            p2 ← getpolinomio(B, inicio, fin, aux)
            inicio ← indB[k], fin ← inicio + 7
            p3 ← getpolinomio(B, inicio, fin, aux)
            inicio ← indB[k], fin ← inicio + 7
            p4 ← getpolinomio(B, inicio, fin, aux)
            pres ← sumpolinomio(p1, p2, p3, p4)
            final(pres)
        end while
        m++, n ← 0 , k ← ind × 4 , aux ← 0
    end while
    ind++, k ← ind × 4 , m ← 0, n ← 0
end while
```

Algorithm 5 Algoritmo función final

Require: int *resultado
Ensure: resultado en hexadecimal

```
int L, R
L = res[0] × 8 + res[1] × 4 + res[2] × 2 + res[3] × 1
R = res[4] × 8 + res[5] × 4 + res[6] × 2 + res[7] × 1
Mostrar(L,R)
```

Algorithm 6 Algoritmo función GetPolinomio

Require: int B[128],int inicio, int fin, int pol
Ensure: p

```
1: int i, n, *p
2: int Pirreducible[8] = {0,0,0,1,1,0,1,1}
3: if (pol == 01) then
4:    retornar la misma matriz
5: end if
6: if (pol == 02) then
7:    retornar la misma matriz con un deslizamiento a la izquierda
8: end if
9: if (pol == 03) then
10:   retornar la misma matriz con un deslizamiento a la izquierda + la matriz original
11: end if
12: return p
```

Algorithm 7 Algoritmo función SumPolinomio

Require: *p1, *p2, *p3, *p4
Ensure:

```
1: int i, *p
2: for (i ← 0; i < 8; i++) do
3:    if (p1[i] == p2[i]||p3[i] == p4[i]) then
4:       p[i] ← 0
5:    end if
6:    if (p1[i]! = p2[i]||p3[i]! = p4[i]) then
7:       p[i] ← 0
8:    end if
9:    if (p1[i] == p2[i] == p3[i] == p4[i]) then
10:      p[i] ← 0
11:   else
12:      p[i] ← 1
13:   end if
14: end for
15: return p
```

1.2. Formulación del experimento

Para la implementación del algoritmo se aplicaron las siguientes estrategias:

- Creación de una matriz aleatoria de 128 bits
- Función Hexadecimal: dada la matriz A[128] dividir en bloques de 8 bits. Para cada bloque se evalúan sub-bloques para obtener el valor hexadecimal. La parte izquierda (L) y la parte derecha (R) se evalúan y luego se guarda el valor decimal en una matriz Baux[32].
- Función ShiftRow: debido a que la matriz de entrada Baux[32] conserva el valor en decimal, el deslizamiento de las filas se realiza intercambiando posición a posición en una nueva matriz. Esto genera un mayor costo en memoria, pero nos asegura que el valor está en la posición correcta luego del ShiftRow.
- Función conToBin: convertir cada valor decimal en su representación binaria, de acuerdo a la tabla de conversión definida al principio del informe.
- Función MixColumn: en esta función se trabaja con punteros para la multiplicación de la matriz salida del ShiftRow con la matriz fija de polinomios, se obtiene el resultado llamando a la función GetPolinomio.
- Función GetPolinomio: función que multiplica los polinomios en el cuerpo de GF(2^8), se divide por el polinomio irreducible en caso de ser necesario..
- Función SumPolinomio: suma de los valores obtenidos con GetPolinomio.

2. Conclusiones

Con la implementación de una ronda del algoritmo AES se pudo comprobar por qué el AES es uno de los algoritmos más populares usados en criptografía simétrica, esto es debido a la gran difusión que entregan las funciones ShiftRow y MixColumn en un bloque fijo de 128 bits.

En esta oportunidad no se implementó la etapa del AddRoundKey, sin embargo, el código puede ser adaptado para incorporar nuevas funciones para el caso de agregar una clave fija o para un bloque determinado.

3. Estructuras de Datos Utilizadas

Se utilizaron arreglos y punteros para la representación de matrices y vectores.

4. Lectura de programa desde la consola

La lectura del programa se indica a continuación:
Para compilar:

- gcc -o ejecutar lab2.c

Para ejecutar:

- ./ejecutar

5. Implementación

El algoritmo está implementado en lenguaje C.

6. Plataforma Computacional

El programa fue ejecutado en un computador con las siguientes características:

- **Procesador:** Intel Core i3-350M CPU @ 2.27GHz
- **Memoria RAM:** 2 GB
- **Tipo de SO:** 64 bits
- **Sistema Operativo:** Linux - Ubuntu 14.04 LTS

6.1. Librerías

- stdio.h
- stdlib.h
- string.h
- time.h