Universidad Santiago de Chile

Sistemas de encriptación AES

Criptología

Laboratorio 2

Author:
Katerine Muñoz Tello
Sergio Salinas Fernández

Email:
katerine.munoz@usach.cl
sergio.salinas@usach.cl

April 19, 2017

Contents

1 Algoritmo Implementado

Algorithm 1: bitGenerator. Genera números binarios aleatorios que son ingresados a un arreglo.

Data: int. Arreglo de 128-bit.
Result: int. Arreglo llenado con números binarios aleatorios.
for $i \longleftarrow 0 \ldots n$ **do** b[i] $\longleftarrow$ random number mod 2;

Algorithm 2: check. Si es que el número ingresado es distinto de cero, devuelve la suma de este con 1.

Data: int.
Result: int.
if x != 0 **then**
| x $\longleftarrow$ x + 1
return x;

Algorithm 3: hextobin. Retorna la representación binaria de un número hexadecimal

Data: char. Representación hexadecimal de número binario.
Result: char. Representación binaria del hexadecimal ingresado.

```
switch hex do
    case '0' do return'0000';
    case '1' do return'0001';
    case '2' do return'0010';
    case '3' do return'0011';
    case '4' do return'0100';
    case '5' do return'0101';
    case '6' do return'0110';
    case '7' do return'0111';
    case '8' do return'1000';
    case '9' do return'1001';
    case 'A' do return'1010';
    case 'B' do return'1011';
    case 'C' do return'1100';
    case 'D' do return'1101';
    case 'E' do return'1110';
    case 'F' do return'1111';
    case 'a' do return'1010';
    case 'b' do return'1011';
    case 'c' do return'1100';
    case 'd' do return'1101';
    case 'e' do return'1110';
    case 'f' do return'1111';
end
```

Algorithm 4: Main Function.

```
inicializar;
llamada a bitGenerator;

/* Byte Substitution                                                  */
for i ⟵ 0, 8, 16, ..., 128 do
    row ⟵ conversión de binario a decimal desde i hasta i+3 del bit generado;
    col ⟵ conversión de binario a decimal desde i+4 hasta i+7 del bit generado;
    hex ⟵ sbox[row][col];
    for j ⟵ 0 ... 4 do
        bin ⟵ hextobin(hex[0]);
        bin2 ⟵ hextobin(hex[1]);
    end
end

/* Shift Rows                                                         */
k ⟵ 0;
for i ⟵ 0 ... 16 do
    for j ⟵ 0 ... 8 do
        s[k] ⟵ guarda el binario rescatado arriba según la matriz de ShiftRow;
        k ⟵ k + 1;
    end
end

/* Mix Column                                                         */
/* Se toman los binarios de s y se transforman a grado guardados en
   aux                                                                */
raised ⟵ 7;
piv ⟵ 7;
for i ⟵ 0 ... 128 do
    if i alcanza los 8-bit then raised ⟵ 7;
    /* En la posición 7 se guarda el mismo valor de s para saber si es
    que x^0 existe.  Si se guardara el valor de su grado, siempre sería
    0.  */
     else if i = piv then
        aux[i] ⟵ s[i];
        piv ⟵ piv + 8;
        raised ⟵ raised - 1;
        continue;
    aux[i] ⟵ s[i] * raised;
    raised ⟵ raised - 1;
end
```

Algorithm 5: Main Function.

```
/* Multiplicación del arreglo s con la matriz guardado en aux2     */
row ⟵ 0;
col ⟵ 0;
piv ⟵ 0;
for int ⟵ 0, 32, 64, ..., 128 do
    k ⟵ i;
    for j ⟵ 0, 8, 16, 24, 32 do
        /* Multiplicación por 1. Los grados quedan igual salvo por x^0,
           al cual se le suma 1 y se le aplica XOR en caso de tener
           coef. mayor a 1.                                          */
        if matrix[row][col] ⟵ 01 then
            aux2[piv] ⟵ 0;
            aux2[piv+1] a aux2[piv+7] ⟵ aux[piv] a aux[piv+6] respectivamente;
            aux2[piv+8] ⟵ aux[piv+7] ⊕ 1;
        /* Multiplicación por x. Se le suma 1 a cada grado que no sea
           0. x^1 queda con el mismo valor, ya que se guardó su coef.
           no grado.                                                 */
        else if matrix[row][col] ⟵ 02 then
            aux2[piv] a aux2[piv+6] ⟵ check(aux[piv] a aux[piv+6]);
            aux2[piv+7] ⟵ aux[piv+7];
            aux2[piv+8] ⟵ 0;
        /* Multiplicación por (x + 1). Se le suma 1 a cada grado que no
           sea 0 y se le aplica XOR (por la multiplicación por 1) para
           reducir coef. mayores a 1. Para x^1 lo mismo, pero si
           sumarle 1 (porque está guardado su coef. no su grado) y x^0
           toma el valor de coef. original.                          */
        else
            aux2[piv] a aux2[piv+6] ⟵ check(aux[piv] a aux[piv+6]) ⊕ aux[piv] a
              aux[piv+6];
            aux2[piv+7] ⟵ aux[piv+7] ⊕ aux[piv+6];
            aux2[piv+8] ⟵ aux[piv+7];
        piv ⟵ piv + 9;
        col ⟵ col + 1;
    end
    row ⟵ row + 1;
    col ⟵ 0;
end
```

Algorithm 6: Main Function.

```
/* Se reducen los polinomios de grado 8 por el polinomio irreductible
   */
PI ← x^8 + x^4 + x^3 + x + 1;
for i ← 0, 9, 18 ... 144 do
    if aux2[i] == 8 then
        for j ← 0,1,2 ... 9 do
            aux2[i+j] = aux2[i+j] ⊕ PI[j];
        end
    end
end
/* Se guardan los polinomios de grado 8 como polinomios de grado 7 */
t ← 0;
for i ← 0, 9, 18 ... 144 do
    for j ← i, i+1, i+2 ... i + 9 do
        if aux2[j] == 0 then
            s[t] = 0;
        else
            s[t] = 1;
        end
        t = t+1;
    end
end
return s;
```

2 Formulación del experimento

Se elabora un programa en C que implemente la función AES que incluya:

- Byte Substitution Layer:
  - Utilizando una S-Box.
- Capa de difusión:
  - ShiftRow
  - MixColumn

La salida del programa corresponde a cada estado del arreglo de bits principal.

El algoritmo fue probado en un computador con procesador i5 de 2,9 gHz y 8GB de RAM.

Ya que el algoritmo trabaja con largos de arreglos fijos, no fue posible construir un gráfico de desempeño que no fuese lineal.

Para tratar con los polinomios en C se guardan en un arreglo de largo 9, como los polinomios no tienen coeficientes se guarda su grado, y para expresar el x^0 se guardaba un 1. Ejemplo:

$x^8 + x^4 + x^3 + x + 1$ se guarda como un arreglo [8,0,0,0,4,3,0,1,1].

Debido a la complejidad al trabajar con números hexadecimales, se decidió trabajar los elementos de la tabla S-Box como tipo *char*, transformándolos posteriormente a tipo *int*.

3 Conclusiones

El algoritmo en la partes de byte subtitution y shiftrows no parece diferenciarse mucho del algoritmos DES, pero se puede ver su potencial en la parte de MixColumn en donde se puede ver su verdadero potencial, siendo está la parte más compleja de programar y de la que se tuvo que investigar su parte matemática a fondo para poder implementarla y depurar el algoritmo.