Miniaes PRACTICA_ Teoria numeros y criptografia teoria

Teoría De Números Y Criptografía (Universidad de Granada)

TEORÍA DE NÚMEROS Y CRIPTOGRAFÍA

Criptografía simétrica

Silvia Barroso Moreno-49616461X-silviabm98@correo.ugr.es

April 27, 2021

Contents

1 Ejercicio

Consideremos el cifrado por bloques MiniAES descrito en el ejercicio 2.1.

- 1) Calcula $E_{dni}(0x01234567)$ usando el modo CBC e IV = 0x0001.
- 2)Calcula $E_{dni}(0x01234567)$ usando el modo CFB, r = 11, y vector de inicialización IV = 0x0001.

1.1 Apartado 1

Tenemos dni=49616461, entonces clave=49616461 mod 65536=5709 y el mensaje es
0x01234567=[1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1].

Vamos a hallar el criptograma usando el modo de operación Cipher-Block-Chaining(CBC) y el cifrado por bloques MiniAes. La idea esta en calcular c_1,c_2 ya que dividiremos nuestro mensaje en dos bloques de 16 bits:
$m_1 = [1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0]$
$m_2 = [1, 1, 0, 0, 0, 1, 0, 0, 1] + [0, 0, 0, 0, 0, 0, 0] = [1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$

Finalmente, $E_{dni}(0x01234567) = c_0c_1c_2$, siendo $c_0 = 0x0001$.

Aplicando el algoritmo descrito en los apuntes, se obtiene:
$c_1 = 0x5bc7 = [1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1]$
$c_2 = 0x8f28 = [0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1]$
Por tanto, $E_{dni}(0x01234567) = c_0c_1c_2$= [1][1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1][0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1]=1 1 1 1 0 0 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1

- $c_1 = E_k(m_1 + c_0)$
  Las variables de cifrado por bloques MiniAes obtendrán los siguientes valores:
  -k=$[k_0, k_1, k_2, k_3]$=[[0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0], [0, 0, 1, 0]]
  -σ_{k_0}=[[0, 0, 0, 1], [0, 1, 1, 1], [0, 1, 1, 0], [1, 1, 1, 1]]
  -γ=[[1, 0, 0, 0], [0, 0, 0, 0], [1, 0, 1, 1], [0, 1, 0, 0]]
  -Π=[[1, 0, 0, 0], [0, 1, 0, 0], [1, 0, 1, 1], [0, 0, 0, 0]]
  -θ=[[0, 0, 1, 1], [1, 1, 1, 1], [1, 1, 1, 0], [0, 1, 0, 1]]
  -σ_{k_1}=[[1, 1, 1, 0], [0, 1, 0, 0], [0, 0, 0, 1], [0, 1, 1, 1]]
  -γ=[[0, 1, 0, 1], [0, 0, 0, 1], [1, 0, 0, 0], [0, 0, 0, 0]]
  -Π=[[0, 1, 0, 1], [0, 0, 0, 0], [1, 0, 0, 0], [0, 0, 0, 1]]
  -σ_{k_2}=[[0, 1, 0, 1], [1, 0, 1, 1], [1, 1, 0, 0], [0, 1, 1, 1]]
  Finalmente, tenemos [1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0]=0x5bc7=c_1

- $c_2 = E_k(m_2 + c_1)$
  k=$[k_0, k_1, k_2, k_3]$=[[0, 0, 0, 1], [1, 1, 1, 0], [1, 0, 1, 0], [0, 0, 0, 0]]
  -σ_{k_0}=[[0, 0, 0, 0], [1, 0, 0, 0], [1, 1, 1, 0], [1, 1, 0, 1]]
  -γ=[[0, 0, 1, 1], [1, 1, 0, 0], [0, 1, 0, 1], [1, 1, 0, 1]]
  -Π=[[0, 0, 1, 1], [1, 1, 0, 1], [0, 1, 0, 1], [1, 1, 0, 0]]
  -θ'=[[1, 1, 0, 0], [0, 0, 1, 0], [0, 1, 0, 0], [1, 1, 0, 1]]
  -σ_{k_1}=[[0, 0, 0, 1], [1, 0, 0, 1], [1, 0, 1, 1], [1, 1, 1, 1]]
  -γ=[[1, 0, 0, 0], [1, 1, 1, 0], [0, 1, 1, 0], [0, 1, 0, 0]]
  -Π=[[1, 0, 0, 0], [0, 1, 0, 0], [0, 1, 1, 0], [1, 1, 1, 0]]
  -σ_{k_2}=[[0, 1, 0, 1], [1, 0, 1, 1], [1, 1, 0, 0], [0, 1, 1, 1]]
  Finalmente, tenemos [0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1]=0x8f28=c_2

Para obtener c_1, c_2 he usado el siguiente algoritmo para verificar mis calculos :

```
F.<xi>=GF(2^4, modulus=GF(2)[x](x^4+x+1))
F.modulus()

def nibblesub(ll):
        xx=vector(ll)
        if xx!=0:
                xx=vector(reversed(vector(F(vector(reversed(xx)))^(-1))))
        return list(xx*matrix(GF(2),4,[0,1,1,1,1,1,1,0,1,1,0,1,1,0,1,1])+vector([0,0,1,1]))

def shiftrow(ll):
        return [ll[0],ll[3],ll[2],ll[1]]

def mixcolumn(ll):
        return [list(reversed(vector(ele))) for ele in (matrix(F,2,[xi+1,xi,xi,xi+1])*matrix(F,2,[F(vector(reversed(ll[0]))), F(vector(reversed(ll[2]))), F(vector(reversed(ll[1]))), F(vector(reversed(ll[3])))])).transpose().list()]

def addroundkey(ll,kk):
        return [list(vector(ll[0])+vector(kk[0])), list(vector(ll[1])+vector(kk[1])), list(vector(ll[2])+vector(kk[2])), list(vector(ll[3])+vector(kk[3]))]

#Calcula K0,K1,K2
def key_squedule(kk):
        ww = list(reversed(vector(GF(2),kk.bits())))
        #Si K0 es menor de 16 bit, se rellena con ceros
        while len(ww)<16:
                ww = [GF(2)(0)]+ww
        #Calculamos k1,k2 y lo metemos en la lista
        ww = [[ww[0],ww[1],ww[2],ww[3]],[ww[4],ww[5],ww[6],ww[7]],[ww[8],ww[9],ww[10],ww[11]],[ww[12],ww[13],ww[14],ww[15]]]
        ww += [list(vector(ww[0])+vector(nibblesub(ww[3]))+vector(GF(2),[0,0,0,1]))] #w4
        ww += [list(vector(ww[1]) + vector(ww[4]))] #w5
        ww += [list(vector(ww[2]) + vector(ww[5]))] #w6
        ww += [list(vector(ww[3]) + vector(ww[6]))] #w7
        ww += [list(vector(ww[4])+vector(nibblesub(ww[7]))+vector(GF(2),[0,0,1,0]))]#w8
        ww += [list(vector(ww[5]) + vector(ww[8]))]#w9
        ww += [list(vector(ww[6]) + vector(ww[9]))]#w10
        ww += [list(vector(ww[7]) + vector(ww[10]))]#w11
        return ww

def MiniAES(kk,mm):
        ww = key_squedule(kk) #w0,w1,w2,w3,.....,w11
        #m[i]
        estado = list(reversed(vector(GF(2),mm.bits())))

        while len(estado)< 16:
                estado = [GF(2)(0)]+estado
        estado = [[estado[0],estado[1],estado[2],estado[3]],[estado[4],estado[5],estado[6],estado[7]],[estado[8],estado[9],estado[10],estado[11]],[estado[12],estado[13],estado[14],estado[15]]]
        #Ya aplico la funcion de encriptar Ek()--->MiniAes
        estado = addroundkey(estado,[ww[0],ww[1],ww[2],ww[3]]) #XOR
        estado = [nibblesub(ele) for ele in estado] #GAMMA
        estado = shiftrow(estado) #PI
        estado = mixcolumn(estado) #O
        estado = addroundkey(estado,[ww[4],ww[5],ww[6],ww[7]])
        estado = [nibblesub(ele) for ele in estado]
        estado = shiftrow(estado)
        estado = addroundkey(estado,[ww[8],ww[9],ww[10],ww[11]])
        salida = list(reversed(estado[0]+estado[1]+estado[2]+estado[3]))
        return estado,Integer(salida,base=2).hex()

clave=49616461%(65536)

mensaje=list(0x01234567.bits())

#Dividimos en bloques de tamano N=16
mens2=[1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0]
```

```
mens1=[1, 1, 0, 0, 0, 1, 0, 0, 1]+[0,0,0,0,0,0,0]

c0='0x0001'
c1='0x'+MiniAES(clave, Integer(mens1, base=2)^^(Integer(c0)))[1]
c2='0x'+MiniAES(clave, Integer(mens2, base=2)^^(Integer(c1)))[1]
```

1.2 Apartado 2

Dividimos el mensaje 0x01234567=[1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1] en tres bloques de 11 bit,r=11, y tenemos:
m_1=[0,0,1,0,0,0,0,0,0,0,0]
m_2=[0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1]
m_3=[1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1]

Aplicando Cipher FeedBack se obtiene:

- $x_1 = 0x0001 = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]$
- $c_1 = m_1 + msb_r(E_k(x_1))$=0x71=[1, 0, 0, 0, 1, 1, 1]
- $x_2 = lsb_{N-r}||c_1$=[1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0]
- $c_2 = m_2 + msb_r(E_k(x_2))$=0x42b=[1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1]
- $x_3 = lsb_{N-r}||c_2$=[1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1]
- $c_3 = m_3 + msb_r(E_k(x_3))$=0x268=[0, 0, 0, 1, 0, 1, 1, 0, 0, 1]

Finalmente, se obtiene el criptograma $c_1c_2c_3$=[1, 0, 0, 0, 1, 1, 1][1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1][0, 0, 0, 1, 0, 1, 1, 0, 0, 1]=1 0 0 0 1 1 1 1 1 0 1 0 1 0 0 0 0 1 0 0 0 1 0 1 1 0 0 1

```
clave=49616461%(65536)
def msb(num,r):
        lista=num.bits()
        while len(lista)<16:
                lista=lista+[0]
        return lista[-r:]
def lsb(num,r):
        lista=num.bits()
        while len(lista)<16:
                lista=lista+[0]
        return lista[:r]

mens=0x01234567.bits()
mens=mens+[0,0,0,0,0,0,0,0]
#Dividimos en bloques tamano r=11
mens1=[0,0,1,0,0,0,0,0,0,0,0]
mens2=[0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1]
mens3=[1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1]

x1='0x0001'
Integer(x1).bits()
x1=[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
c1='0x'+(Integer(mens1, base=2)^^(Integer(msb(Integer('0x'+MiniAES(clave, Integer((x1),base
    =2))[1]),11), base=2))).hex()
Integer(c1).bits()

x2=list(reversed(list(reversed(lsb(Integer(x1,base=2),5)))+list(reversed(Integer(c1).bits())
    )))
c2='0x'+(Integer(mens2, base=2)^^(Integer(msb(Integer('0x'+MiniAES(clave, Integer((x2),base
    =2))[1]),11), base=2))).hex()
```

```
Integer(c2).bits()

x3=list(reversed(list(reversed(lsb(Integer(x2,base=2),5)))+list(reversed(Integer(c2).bits())
    )))
c3='0x'+(Integer(mens3, base=2)^^(Integer(msb(Integer('0x'+MiniAES(clave, Integer((x3),base
    =2))[1]),11), base=2))).hex()
Integer(c3).bits()
```