CUERPOS DE GALOIS G(256)

```
# Cuerpo finito 256  elems (0..255)

# b_7, b_6... b_0 <-> b_7 * x^7 + ... b_1 * x + b_0
# suma: XOR bit a bit (b ^ b' en python)
# resta: lo mismo

# producto: (b_7 * x^7 + ... b_1 * x + b_0) * (b_7' * x^7 + ... b_1' * x + b_0') =
b7 * b7' * x^14 + ... b0*b0' (15 bits !!)
# necesitamos un polinomio irreducible de grado 8 m(x).
# AES: m = 0x 11B = 100010111 <->  x^8 + x^4 + x^2 + x + 1

# EN EL CONSTRUCTOR DE LA CLASE GF LE PASAMOS EL POLINOMIO IRREDUCIBLE QUE
QUEREMOS PASAR (si se deja en blanco se usa el por defecto)
# ahora con b7 * b7' * x^14 + ... b0*b0' mod m(x) (obtendremos un polinomio de
grado estrictamente menor a 8, menor a m)
# B * B' := p(x) mod m(x) = residuo(x)

# funcion para multiplicar bytes, se hace mirando rapidamente a tablas
# tablas: construidas 1 sola vez. Se crean con un algoritmo de multiplicacion
"lenta"

xTime(a):
#   a(x)*x

    if a <= 127:
        a <<1
    else:
        # se nos sale del byte si multiplicamos
        (a <<1)^m
        # hacemos XOR con m, el resultado de el residuo de dividir entre m

# producto lento::
# a(x) * b(x) = SUM:i=0:7 bi * x^i * a(x) donde x^i * a = x(x(x..a)) i veces

# generadores, tienen la longitud de ciclo de elevaciones exponenciales maxima
(que es la cardinalidad del cuerpo-1, es decir g^255 = 1)
# son diferentes g^0 = 1, g^1 = g, g^2, g^3,...,g^254
# hay que encontrar un generador (0 y 1 no valen), g=3 (0x03)

# TABLAS de g^i i de log(g^i)=i
# i | 0       1      2  ...   254
# gî| g^0=1  g^1=g    g^2  ... g^254
# log(g^i)=i
# se hacen con el producto lento: exp[i] de i=0,...,254 = g*exp[i-1]
# log[g^i] = i

# con las tablas tenemos el producto rapido y los inversos
# producto rapido: A * B, A = g^i donde i = log(A), B = g^j donde j = log(B) -->
g^i * g^j = g^(i+j)%255
# exp ((log(A) + log(B)) % 255)

# inverso rapido
# A^-1 (A!=0)
# A = g^i, (g^i)^-1 = g^(-i)%255
# exp(-log(A)%255) = A^-1
```

AES PARTE 1: OPERACIONES ELEMENTALES

```
# ByteSub: Sbox, invSbox
# ShiftRow: 0,1,2 i 3 bytes de offset
```

```
# MixColumn: Matriz circulante 0x11B (dada la primera fila, las siguientes son
la anterior + offest de 1 hacia la derecha)
#   Con la matriz circulante se multiplica por el estado (teniendo en cuenta la
multiplicacion en cuerpo finito)
#   Matriz inversa con GAUSS (invMixColumn): M^-1 donde M es la matriz
circulante

## Sbox
# ByteSub[byte] = byte'
# invByteSub[byte'] = byte  (se construye automaticamente con la otra)
# construccion ByteSub:
#   dado un byte A calculamos B=A^-1 separamos bit a bit los bits de B
construyendo un vector [b7, b6, ..., b0] (se puede hacer haciendo
desplazamientos)
#multiplica la matriz
# 0 0 0 1 1 1 1 1
# 1 0 0 0 1 1 1 1                * el vector de bs (como columna)
# 1 1 0 0 0 1 1 1
#... hasta 8 filas, matriz ciclica
#(por ejemplo primera fila, b4 + b4 + b2 + b1 + b0) se hacen las sumas como xors
exclusivas
#cuando tengamos el vector cs constuyo el byte C y le sumo 0x63

# PARA EL 0 solo le sumo 0x63
# ByteSub empieza en 0x63
#invByteSub, en la posicion 0x63 està el 0

# padding, tanto como para encript como para desencript     PKCS7
# añadir bytes para que el documento tenga longitud multiplo 16 bytes.
# si me falta n3 bytes añado 3 bytes 0x03, si me faltan 7 añado 7 bytes del tipo
0x07
# si el último es 0x01, es padding de 1 o es justo? SIEMPRE SE AÑADE PADDING, si
da justo, añadimos 16 bytes de 16

# CBC
# encriptamos con la informacion del estado anterior encriptado
# en el ficher encriptado, los 16 primeros bytes son el vector de inicializacion
# con el vi, el primer estado se hace xor y con el resultado se hace xor con el
segundo estado encriptado...

# openssl para comprobar
```

AES PARTE 2: EXPANSIÓN DE LA CLAVE

```
# Expansion de la clave
# key -> EXPANDED key
# long(bytes)               Num Rondas
# 16            176         11      (16*11 = 176)
# 24            208         13      (16*13 = 208)
# 32            240         15      (16*15 = 240)
#
#
# key = key_0
# la organizo en palabras de 4 bytes
#
# calculo las keys que necesite (todas con los mismos numeros de bytes)
# El calculo complicado es la primera palabra de cada key (la primera columna, 4
bytes)
#
# key_i[0]
# cogemos la ultima palabra de la key_i-1, llamaremos k
# 1-> Rotamos k (shift hacia arriba)
```

```
# 2-> SubByte (de la Sbox)
# 3-> Hacemos XOR con la palabra key_i-1[0]
# 4-> Hacemos XOR con la posicion del vector Rcon que toque (solo cambia el primer byte ya que el Rcon tiene vlores diferente de 0 solo en la primera componente)
#
# para key_i[j]
# 1->key_i[j-1] XOR key_i-1[j]
#
# Rcon
# 1, 2¹, 2², 2³, 2⁴... Rcon[i] = Rcon[i-1] * 0x02
# (en hexadeximal)
# Xtimes el anterior, Xtimes el anterior... Hay que calcular suficientes componentes para todas las keys, muchos valores no se usaran
```