En el `Advanced Encryption Standard` (AES) algunas operaciones, SubBytes(), MixColumn(), XTimes()..., interpretan los bytes como elementos del cuerpo finito (Galois Field) $GF(2^8)$.

Para construir explícitamente $GF(2^8)$ se usa el polinomio $\mathbf{m} = \mathbf{x^8} + \mathbf{x^4} + \mathbf{x^3} + \mathbf{x} + \mathbf{1}$ (representado en binario por `0b100011011`, en hexadecimal por `0x11B` o como entero por `283`).

Este polinomio es el primero de la lista de polinomios irreducibles de grado 8 con coeficientes en $\mathbb{Z}_2$: `0x11B, 0x11D, 0x12B, 0x12D, ..., 0x1F3, 0x1F5, 0x1F9.` Salvo ser el primero de la lista, no tiene nada de especial.

La práctica consistirá en implementar el AES para que pueda usar cualquier polinomio irreducible $\mathbf{m}$ de grado 8 con coeficientes en $\mathbb{Z}_2$ para construir explícitamente $GF(2^8)$.

1. El cuerpo finito $GF(2^8)$

Los elementos de este cuerpo se pueden representar por **bytes**. Los expresaremos en forma binaria, hexadecimal, entera o polinómica, según convenga.

El byte $\mathtt{b}_7\mathtt{b}_6\mathtt{b}_5\mathtt{b}_4\mathtt{b}_3\mathtt{b}_2\mathtt{b}_1\mathtt{b}_0$, $\mathtt{b}_i \in \{0,1\}$, será el polinomio $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$.

Por ejemplo, `0b01010111` = `0x57` = `87` será $x^6 + x^4 + x^2 + x + 1$.

Suma

La suma de dos elementos del cuerpo es la suma de polinomios binarios (coeficientes en $\mathbb{Z}_2$).

Por ejemplo, `0b01010111+0b10000011` será

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 = \mathtt{0b11010100}$$

Se corresponde con la operación XOR, que se denotará $\oplus$. El elemento neutro de la suma es `0b00000000=0x00=0` y el opuesto de cada elemento es el mismo.

Producto

El producto de dos elementos del cuerpo se corresponde con el producto de polinomios binarios seguido de una reducción módulo un polinomio irreducible[1] $\mathbf{m}$ de grado 8.

Por ejemplo, si $\mathbf{m} = \mathbf{x^8} + \mathbf{x^4} + \mathbf{x^3} + \mathbf{x^2} + \mathbf{1}$ (`0x11D` si lo escribimos en hexadecimal)

$$\begin{aligned}(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) &= x^{13} + x^{11} + x^9 + x^8 + x^7 + \\ &\quad x^7 + x^5 + x^3 + x^2 + x + \\ &\quad x^6 + x^4 + x^2 + x + 1 \\ &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1\end{aligned}$$

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \pmod{x^8 + x^4 + x^3 + x^2 + 1} = x^5 + x^4 + 1.$$

[1] Un polinomio $\mathbf{m}$ es irreducible si sus únicos divisores son 1 y $\mathbf{m}$.

El elemento neutro del producto es 0b00000001 = 0x01 = 1.

En $GF(2^8)$ todo elemento diferente de 0x00 tiene inverso multiplicativo. El inverso del polinomio **a** es el único polinomio **b** tal que

$$\mathbf{a}\,\mathbf{b} \equiv 1 \mod \mathbf{m}.$$

Se puede calcular usando el algoritmo extendido de Euclides.

También podemos escribir los elementos diferentes del 0x00 como potencia de un generador. Por ejemplo, si $g = x = \texttt{0b00000010} = \texttt{0x02} = 2$, entonces

$$GF(2^8) = \{g, g^2, \ldots, g^{254}, g^{255}(= g^0 = 1)\} \cup \{0\}$$

El producto de dos elementos $a = g^i$ y $b = g^j$, diferentes de 0x00, es $ab = g^i g^j = g^{i+j}$, y el inverso de a es $a^{-1} = (g^i)^{-1} = g^{-i} = g^{255-i}$. En este caso, la multiplicación y el cálculo de inversos se reducen a la búsqueda en una tabla de 255 elementos.

Definid en **Python 3.x** la siguiente clase con los métodos descritos (podéis definir otros si lo consideráis necesario):

```
class G_F:
    '''
    Genera un cuerpo finito usando como polinomio irreducible el dado
    representado como un entero. Por defecto toma el polinomio del AES.
    Los elementos del cuerpo los representaremos por enteros 0<= n <= 255.
    '''

    def __init__(self, Polinomio_Irreducible = 0x11B):
        '''
        Entrada: un entero que representa el polinomio para construir el cuerpo

        Tabla_EXP y Tabla_LOG dos tablas, la primera tal que en la posición
        i-ésima tenga valor a=g**i  y la segunda tal que en la posición a-ésima
        tenga el valor i tal que a=g**i. (g generador del cuerpo finito
        representado por el menor entero entre 0 y 255.)
        '''
        self.Polinomio_Irreducible
        self.Tabla_EXP
        self.Tabla_LOG
        self.g


    def xTimes(self, n):
        '''
        Entrada: un elemento del cuerpo representado por un entero entre 0 y 255
        Salida:  un elemento del cuerpo representado por un entero entre 0 y 255
                   que es el producto en el cuerpo de 'n' y 0x02 (el polinomio X).
        '''
```

```
    def producto(self, a, b):
        '''
        Entrada: dos elementos del cuerpo representados por enteros entre 0 y 255
        Salida:  un elemento del cuerpo representado por un entero entre 0 y 255
                   que es el producto en el cuerpo de la entrada.

        Atención: Se valorará la eficiencia. No es lo mismo calcularlo
                  usando la definición en términos de polinomios o calcular
                  usando las tablas Tabla_EXP y Tabla_LOG.
        '''

    def inverso(self, n):
        '''
        Entrada: un elementos del cuerpo representado por un entero entre 0 y 255
        Salida:  0 si la entrada es 0,
                 el inverso multiplicativo de n representado por un entero entre
                    1 y 255 si n <> 0.
        Atención: Se valorará la eficiencia.
        '''
```

2. Implementación AES

La descripción de AES se puede encontrar en el documento Federal Information Processing Standards Publication (FIPS) 197: Advanced Encryption Standard (AES) `https://doi.org/10.6028/NIST.FIPS.197-upd1`

El polinomio usado para construir explícitamente $GF(2^8)$ en la especificación es **m**=`0x11B` pero cualquier otro de la lista dada anteriormente se podría usar perfectamente.

Cambiar **m** significa que:

- los resultados de las operaciones con elementos del cuerpo serán diferentes, en general.
- la tabla 4 de substitución usada en la función SubBytes() será diferente porque implica el cálculo de inversos en el cuerpo finito.
- la matriz usada en la función InvMixColumns() será diferente ya que es la inversa de la matriz usada en MixColumns()
- las constantes de la tabla 5 usada en KeyExpansion() será diferente porque implica multiplicar por `0x02` reiteradamente.

Definid en **Python 3.x** la siguiente clase con los métodos descritos (podéis definir otros si lo consideráis necesario):

```
class AES:
    '''
    Documento de referencia:
    Federal Information Processing Standards Publication (FIPS) 197: Advanced Encryption
    Standard (AES) https://doi.org/10.6028/NIST.FIPS.197-upd1

    El nombre de los métodos, tablas, etc son los mismos (salvo capitalización)
    que los empleados en el FIPS 197
    '''

    def __init__(self, key, Polinomio_Irreducible = 0x11B):
        '''
        Entrada:
            key: bytearray de 16 24 o 32 bytes
            Polinomio_Irreducible: Entero que representa el polinomio para construir
                                   el cuerpo

        SBox: equivalente a la tabla 4, pág. 14
        InvSBOX: equivalente a la tabla 6, pág. 23
        Rcon: equivalente a la tabla 5, pág. 17
        InvMixMatrix : equivalente a la matriz usada en 5.3.3, pág. 24
        '''
        self.Polinomio_Irreducible
        self.SBox
        self.InvSBox
        self.Rcon
        self.InvMixMatrix

    def SubBytes(self, State):
        '''
        5.1.1 SUBBYTES()
        FIPS 197: Advanced Encryption Standard (AES)
        '''

    def InvSubBytes(self, State):
        '''
        5.3.2 INVSUBBYTES()
        FIPS 197: Advanced Encryption Standard (AES)
        '''

    def ShiftRows(self, State):
        '''
        5.1.2 SHIFTROWS()
        FIPS 197: Advanced Encryption Standard (AES)
        '''

    def InvShiftRows(self, State):
```

```
    '''
    5.3.1 INVSHIFTROWS()
    FIPS 197: Advanced Encryption Standard (AES)
    '''


def MixColumns(self, State):
    '''
    5.1.3 MIXCOLUMNS()
    FIPS 197: Advanced Encryption Standard (AES)
    '''

def InvMixColumns(self, State):
    '''
    5.3.3 INVMIXCOLUMNS()
    FIPS 197: Advanced Encryption Standard (AES)
    '''


def AddRoundKey(self, State, roundKey):
    '''
    5.1.4 ADDROUNDKEY()
    FIPS 197: Advanced Encryption Standard (AES)
    '''


def KeyExpansion(self, key):
    '''
    5.2 KEYEXPANSION()
    FIPS 197: Advanced Encryption Standard (AES)
    '''


def Cipher(self, State, Nr, Expanded_KEY):
    '''
    5.1 Cipher(), Algorithm 1 pág. 12
    FIPS 197: Advanced Encryption Standard (AES)
    '''

def InvCipher(self, State, Nr, Expanded_KEY):
    '''
    5. InvCipher()
       Algorithm 3 pág. 20 o Algorithm 4 pág. 25. Son equivalentes
    FIPS 197: Advanced Encryption Standard (AES)
    '''
```

```
def encrypt_file(self, fichero):
    '''
    Entrada: Nombre del fichero a cifrar
    Salida:  Fichero cifrado usando la clave utilizada en el constructor
               de la clase.
             Para cifrar se usará el modo CBC, con IV correspondiente a los 16
               primeros bytes obtenidos al aplicar el sha256 a la concatenación
               de "IV" y la clave usada para cifrar. Por ejemplo:
                  Key 0x0aba289662caa5caaa0d073bd0b575f4
                      IV asociado 0xeb53bf26511a8c0b67657ccfec7a25ee
                  Key 0x46abd80bdcf88518b2bec4b7f9dee187b8c90450696d2b995f26cdf2fe058610
                      IV asociado 0x4fe68dfd67d8d269db4ad2ebac646986
             El padding usado será PKCS7.
             El nombre de fichero cifrado será el obtenido al añadir el sufijo .enc
               al nombre del fichero a cifrar: NombreFichero --> NombreFichero.enc
    '''

def decrypt_file(self, fichero):
    '''
    Entrada: Nombre del fichero a descifrar
    Salida:  Fichero descifrado usando la clave utilizada en el constructor
               de la clase.
             Para descifrar se usará el modo CBC, con el IV usado para cifrar.
             El nombre de fichero descifrado será el obtenido al añadir el sufijo .dec
               al nombre del fichero a descifrar: NombreFichero --> NombreFichero.dec
    '''
```

Material adicional:

- una serie de ficheros `aes-Valores_TEST_polinomio.pdf` con valores test para diversos polinomios a semejanza de los que encontraréis en el FIPS 197,
- varios ficheros `NombreFichero_polinomio_clave.enc` cifrados.

Vuestra implementación debería ser compatible con `openssl`[2] cuando usáis `0x11B` como polinomio:

- el comando
  `openssl aes-128-cbc -d -K key -iv IV -in infile -out outfile`
  debería descifralo correctamente,
- si cifráis un fichero con el comando
  `openssl aes-128-cbc -e -K key -iv IV -in infile -out outfile`
  deberíais poder descifrarlo con vuestra implementación.

[2] `openssl` está disponible en https://www.openssl.org. Se instala por defecto en la mayoría de las distribuciones de Linux, por ejemplo en la imagen Linux de la FIB.

3. Entrega

Dos ficheros:

- Un fichero que contenga todas las clases pedidas implementadas.

  El nombre del fichero será `aes_Lab_XX.py` substituyendo `XX` por el número, con exactamente dos cifras, del grupo de laboratorio elegido (ej: si el grupo elegido es el `Lab 1`, el nombre del fichero correspondiente será `aes_Lab_01.py`).

- Un fichero que contenga un script en python, `test_Lab_XX.py`, que al ser invocado desde la línea de comandos:

  `python test_Lab_XX.py -c -f file -p 0x11d -k 0xe9e03576ec312b3698c3b6f37d49d770`
  cifre el fichero `file` usando el polinomio `0x11d` y la clave `0xe9e03576ec312b3698c3b6f37d49d770` (en este caso sería una clave de 128 bits)

  `python test_Lab_XX.py -d -f file -p 0x11d -k 0xe9e03576ec312b3698c3b6f37d49d770`
  descifre el fichero `file` usando el polinomio `0x11d` y la clave `0xe9e03576ec312b3698c3b6f37d49d770`

Referencias

- Federal Information Processing Standards Publication (FIPS) 197: Advanced Encryption Standard (AES) `https://doi.org/10.6028/NIST.FIPS.197-upd1`

- NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation. `https://doi.org/10.6028/NIST.SP.800-38A`

- Padding PKCS7: section 6.3 RFC 5652. `http://tools.ietf.org/html/rfc5652#section-6.3`

Para leer

- Bruce Schneier *NSA and Bush's Illegal Eavesdropping.* (December 20, 2005)

- Schmid, Gerhard (11 July 2001). *On the existence of a global system for the interception of private and commercial communications (ECHELON interception system), (2001/2098(INI))*. European Parliament: Temporary Committee on the ECHELON Interception System.