MOOC de Criptología Matemática. Advanced Encryption Standard (AES)

Leandro Marín

Módulo I. Sesión 4.
Dificultad Alta

Historia de AES

- El sistema de cifrado AES *(Advanced Encryption Standard)* es el resultado de un proceso de selección que se realizó por parte del Instituto Nacional de Normas y Tecnología (NIST).

Historia de AES

- El sistema de cifrado AES *(Advanced Encryption Standard)* es el resultado de un proceso de selección que se realizó por parte del Instituto Nacional de Normas y Tecnología (NIST).
- El objetivo del concurso era encontrar un sustituto al algoritmo DES de una forma transparente y abierta.

Historia de AES

- El sistema de cifrado AES *(Advanced Encryption Standard)* es el resultado de un proceso de selección que se realizó por parte del Instituto Nacional de Normas y Tecnología (NIST).
- El objetivo del concurso era encontrar un sustituto al algoritmo DES de una forma transparente y abierta.
- Este proceso tuvo varias fases de selección. El algoritmo finalmente elegido fue el propuesto por dos criptólogos Joan Daemen y Vincent Rijmen y que es conocido como Rijndael.

Historia de AES

- El sistema de cifrado AES *(Advanced Encryption Standard)* es el resultado de un proceso de selección que se realizó por parte del Instituto Nacional de Normas y Tecnología (NIST).
- El objetivo del concurso era encontrar un sustituto al algoritmo DES de una forma transparente y abierta.
- Este proceso tuvo varias fases de selección. El algoritmo finalmente elegido fue el propuesto por dos criptólogos Joan Daemen y Vincent Rijmen y que es conocido como Rijndael.
- Todos los detalles de este estándar se pueden consultar en el documento `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

Longitud de Claves

- El algoritmo AES puede ser utilizado con claves de 128, 192 ó 256 bits.

Longitud de Claves

- El algoritmo AES puede ser utilizado con claves de 128, 192 ó 256 bits.
- Para simplificar el número de casos explicaremos aquí únicamente el caso de 128 bits.

Longitud de Claves

- El algoritmo AES puede ser utilizado con claves de 128, 192 ó 256 bits.
- Para simplificar el número de casos explicaremos aquí únicamente el caso de 128 bits.
- Utilizaremos representación hexadecimal para las claves, por ejemplo:

```
2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
```

Longitud de Claves

- El algoritmo AES puede ser utilizado con claves de 128, 192 ó 256 bits.
- Para simplificar el número de casos explicaremos aquí únicamente el caso de 128 bits.
- Utilizaremos representación hexadecimal para las claves, por ejemplo:

  `2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c`

- Cuando utilicemos notación matricial pondremos estos elementos por columnas en una matriz cuadrada de tamaño 4,

$$\begin{bmatrix} 2b & 28 & ab & 09 \\ 7e & ae & f7 & cf \\ 15 & d2 & 15 & 4f \\ 16 & a6 & 88 & 3c \end{bmatrix}$$

El Estado

- El cifrado con AES se realzará con bloques de entrada de 128 bits.

El Estado

- El cifrado con AES se realzará con bloques de entrada de 128 bits.
- Utilizaremos también la notación hexadecimal, por ejemplo

  `32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34`

El Estado

- El cifrado con AES se realzará con bloques de entrada de 128 bits.
- Utilizaremos también la notación hexadecimal, por ejemplo

  `32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34`

- También utilizaremos notación matricial por columnas:

$$\begin{bmatrix} 32 & 88 & 31 & e0 \\ 43 & 5a & 31 & 37 \\ f6 & 30 & 98 & 07 \\ a8 & 8d & a2 & 34 \end{bmatrix}$$

El Estado

- El cifrado con AES se realzará con bloques de entrada de 128 bits.
- Utilizaremos también la notación hexadecimal, por ejemplo

  `32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34`
- También utilizaremos notación matricial por columnas:

$$\begin{bmatrix} 32 & 88 & 31 & e0 \\ 43 & 5a & 31 & 37 \\ f6 & 30 & 98 & 07 \\ a8 & 8d & a2 & 34 \end{bmatrix}$$

- Esta representación matricial la utilizaremos también para cálculos intermedios. Nos referiremos a esta matriz con los cálculos intermedios como *el estado*.

Visión General del Algoritmo

- El algoritmo AES genera dos secuencias de números, una a partir de la clave que recibirá el nombre de expansión de la clave y una segunda sucesión a partir de los 128 bits de entrada.

Visión General del Algoritmo

- El algoritmo AES genera dos secuencias de números, una a partir de la clave que recibirá el nombre de expansión de la clave y una segunda sucesión a partir de los 128 bits de entrada.
- En esta segunda sucesión se realizarán una serie de operaciones básicas.

Visión General del Algoritmo

- El algoritmo AES genera dos secuencias de números, una a partir de la clave que recibirá el nombre de expansión de la clave y una segunda sucesión a partir de los 128 bits de entrada.
- En esta segunda sucesión se realizarán una serie de operaciones básicas.
- Algunas de estas operaciones serán de combinación de los valores generados en el paso anterior con elementos de la sucesión generada a partir de la clave.

Visión General del Algoritmo

- El algoritmo AES genera dos secuencias de números, una a partir de la clave que recibirá el nombre de expansión de la clave y una segunda sucesión a partir de los 128 bits de entrada.
- En esta segunda sucesión se realizarán una serie de operaciones básicas.
- Algunas de estas operaciones serán de combinación de los valores generados en el paso anterior con elementos de la sucesión generada a partir de la clave.
- Una vez finalizadas todas las rondas, el resultado final será el cifrado del valor de entrada.

La función SubBytes

- La función SubBytes es la S-box interna de AES, que está constituida por una permutación de 256 elementos.

La función SubBytes

- La función SubBytes es la S-box interna de AES, que está constituida por una permutación de 256 elementos.
- Permite por tanto trasformar bytes. Cuando se aplica sobre una lista de bytes, se aplicará a cada elemento de la lista.

La función SubBytes

- La función SubBytes es la S-box interna de AES, que está constituida por una permutación de 256 elementos.
- Permite por tanto trasformar bytes. Cuando se aplica sobre una lista de bytes, se aplicará a cada elemento de la lista.
- Los números no son aleatorios, tienen una justificación matemática en la que no podemos entrar.

La función SubBytes

- La función SubBytes es la S-box interna de AES, que está constituida por una permutación de 256 elementos.
- Permite por tanto trasformar bytes. Cuando se aplica sobre una lista de bytes, se aplicará a cada elemento de la lista.
- Los números no son aleatorios, tienen una justificación matemática en la que no podemos entrar.
- Nosotros los miraremos simplemente como una lista de valores y la aplicaremos como tal.

Los valores de SubBytes

```
SB = [ 0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x1, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
       0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
       0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
       0x4, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x5, 0x9a, 0x7, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
       0x9, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
       0x53, 0xd1, 0x0, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
       0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x2, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
       0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
       0xcd, 0xc, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
       0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0xb, 0xdb,
       0xe0, 0x32, 0x3a, 0xa, 0x49, 0x6, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
       0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x8,
       0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
       0x70, 0x3e, 0xb5, 0x66, 0x48, 0x3, 0xf6, 0xe, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
       0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
       0x8c, 0xa1, 0x89, 0xd, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0xf, 0xb0, 0x54, 0xbb, 0x16]

def SubBytes(s):
  return [SB[t] for t in s]
```

La Función `ShiftRows`

- Esta función se aplica a la matrix del estado produciendo una rotación en las filas de la matriz.

La Función `ShiftRows`

- Esta función se aplica a la matrix del estado produciendo una rotación en las filas de la matriz.
- La primera fila no se rota.

La Función `ShiftRows`

- Esta función se aplica a la matrix del estado produciendo una rotación en las filas de la matriz.
- La primera fila no se rota.
- La segunda fila se rota una posición hacia la izquierda.

La Función `ShiftRows`

- Esta función se aplica a la matrix del estado produciendo una rotación en las filas de la matriz.
- La primera fila no se rota.
- La segunda fila se rota una posición hacia la izquierda.
- La tercera fila se rota dos posiciones hacia la izquierda.

La Función `ShiftRows`

- Esta función se aplica a la matrix del estado produciendo una rotación en las filas de la matriz.
- La primera fila no se rota.
- La segunda fila se rota una posición hacia la izquierda.
- La tercera fila se rota dos posiciones hacia la izquierda.
- La cuarta fila se rota tres posiciones hacia la izquierda.

La Función ShiftRows

$$\begin{bmatrix} s[0] & s[4] & s[8] & s[12] \\ s[1] & s[5] & s[9] & s[13] \\ s[2] & s[6] & s[10] & s[14] \\ s[3] & s[7] & s[11] & s[15] \end{bmatrix} \mapsto \begin{bmatrix} s[0] & s[4] & s[8] & s[12] \\ s[5] & s[9] & s[13] & s[1] \\ s[10] & s[14] & s[2] & s[6] \\ s[15] & s[3] & s[7] & s[11] \end{bmatrix}$$

Si lo escribimos teniendo presente que la matriz del estado se lee por columnas, la función la podemos definir:

```
def ShiftRows(s):
  return [s[0],s[5],s[10],s[15],s[4],s[9],s[14],s[3],s[8],s[13],s[2],s[7],s[12],s[1],s[6],s[11]]
```

La Función `mul`

- Esta función no aparece en el estándar, pero la tenemos que introducir para poder evitar la utilización de operaciones en cuerpos binarios que no podemos explicar con la base matemática que suponemos para la realización de este curso.

La Función `mul`

- Esta función no aparece en el estándar, pero la tenemos que introducir para poder evitar la utilización de operaciones en cuerpos binarios que no podemos explicar con la base matemática que suponemos para la realización de este curso.
- Esta función hace una *especie de multiplicación* sobre bytes. Sólo necesitamos ver cómo se multiplica por 1,2 y 3 que son los únicos valores que necesitaremos para cifrar.

La Función `mul`

- Esta función no aparece en el estándar, pero la tenemos que introducir para poder evitar la utilización de operaciones en cuerpos binarios que no podemos explicar con la base matemática que suponemos para la realización de este curso.
- Esta función hace una *especie de multiplicación* sobre bytes. Sólo necesitamos ver cómo se multiplica por 1,2 y 3 que son los únicos valores que necesitaremos para cifrar.
  - Multiplicar por 1 deja el elemento igual.

La Función `mul`

- Esta función no aparece en el estándar, pero la tenemos que introducir para poder evitar la utilización de operaciones en cuerpos binarios que no podemos explicar con la base matemática que suponemos para la realización de este curso.
- Esta función hace una *especie de multiplicación* sobre bytes. Sólo necesitamos ver cómo se multiplica por 1,2 y 3 que son los únicos valores que necesitaremos para cifrar.
  - Multiplicar por 1 deja el elemento igual.
  - Para multiplicar por 2 multiplicamos el byte por 2 numéricamente, y si el resultado supera los 8 bits, es decir, si es mayor o igual a $2^8 = 256$ entonces le restamos 256 y hacemos un *o exclusivo* con el valor constante `0x1b`.

La Función `mul`

- Esta función no aparece en el estándar, pero la tenemos que introducir para poder evitar la utilización de operaciones en cuerpos binarios que no podemos explicar con la base matemática que suponemos para la realización de este curso.
- Esta función hace una *especie de multiplicación* sobre bytes. Sólo necesitamos ver cómo se multiplica por 1,2 y 3 que son los únicos valores que necesitaremos para cifrar.
  - Multiplicar por 1 deja el elemento igual.
  - Para multiplicar por 2 multiplicamos el byte por 2 numéricamente, y si el resultado supera los 8 bits, es decir, si es mayor o igual a $2^8 = 256$ entonces le restamos 256 y hacemos un *o exclusivo* con el valor constante `0x1b`.
  - Para multiplicar por 3 hacemos un *o exclusivo* entre la multiplicación por 1 y por 2.

Implementación de la Función `mul`

```
def mul(t,s):
  if t == 1:
    return s
  elif t == 2:
    s = s*2
    if s>=256:
      s = (s-256)^^0x1b
    return s
  else: # En este caso t==3
    return s^^mul(2,s)
```

Las Constantes de Ronda Rcon

- En el proceso de expansión de la clave necesitamos una lista de constantes llamadas constantes de ronda.

```
Rc = [1]
for i in range(9):
  Rc.append(mul(2,Rc[-1]))

def Rcon(i):
  return [Rc[i-1],0,0,0]
```

Las Constantes de Ronda Rcon

- En el proceso de expansión de la clave necesitamos una lista de constantes llamadas constantes de ronda.
- Esas constantes están formadas por 4 bytes, pero en realidad el único byte distinto de 0 es el primero, los otros tres son 0.

```
Rc = [1]
for i in range(9):
  Rc.append(mul(2,Rc[-1]))

def Rcon(i):
  return [Rc[i-1],0,0,0]
```

Las Constantes de Ronda `Rcon`

- En el proceso de expansión de la clave necesitamos una lista de constantes llamadas constantes de ronda.
- Esas constantes están formadas por 4 bytes, pero en realidad el único byte distinto de 0 es el primero, los otros tres son 0.
- El primer byte toma el valor 1 en la primera ronda, y en cada ronda sucesiva se multiplica por 2 el último valor de la lista con la función `mul` descrita anteriormente.

```
Rc = [1]
for i in range(9):
  Rc.append(mul(2,Rc[-1]))

def Rcon(i):
  return [Rc[i-1],0,0,0]
```

La Función `MixColumns`

- Sea S el estado, que estará formado por una matriz 4×4 con bytes en las entradas.

La Función `MixColumns`

- Sea S el estado, que estará formado por una matriz 4×4 con bytes en las entradas.
- La función `MixColumns` hace la multiplicación matricial

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot S$$

en la que la suma se sustituirá por el *o exclusivo*, $\oplus$, y el producto por la operación `mul` definida anteriormente.

La Función `MixColumns`

- Sea S el estado, que estará formado por una matriz 4×4 con bytes en las entradas.
- La función `MixColumns` hace la multiplicación matricial

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot S$$

en la que la suma se sustituirá por el *o exclusivo*, $\oplus$, y el producto por la operación `mul` definida anteriormente.

- Por eso sólo hemos definido `mul` por $1, 2$ y 3, puesto que son las únicas constantes que aparecen en esta matriz.

Producto Matricial

- El producto matricial es una operación de *Álgebra Lineal* que dadas dos matrices cuadradas del mismo tamaño, nos calcula una nueva matriz cuadrada del mismo tamaño.

Producto Matricial

- El producto matricial es una operación de *Álgebra Lineal* que dadas dos matrices cuadradas del mismo tamaño, nos calcula una nueva matriz cuadrada del mismo tamaño.
- Se puede calcular columna por columna del siguiente modo (denotaremos $\bullet$ el producto dado por `mul` y $\oplus$ al operador *o exclusivo*. Tendremos en cuenta que $1 \bullet s = s$ para simplificar un poco la fórmula.)

Producto Matricial

- El producto matricial es una operación de *Álgebra Lineal* que dadas dos matrices cuadradas del mismo tamaño, nos calcula una nueva matriz cuadrada del mismo tamaño.
- Se puede calcular columna por columna del siguiente modo (denotaremos $\bullet$ el producto dado por `mul` y $\oplus$ al operador *o exclusivo*. Tendremos en cuenta que $1 \bullet s = s$ para simplificar un poco la fórmula.)
-

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} (2 \bullet s_0) \oplus (3 \bullet s_1) \oplus s_2 \oplus s_3 \\ s_0 \oplus (2 \bullet s_1) \oplus (3 \bullet s_2) \oplus s_3 \\ s_0 \oplus s_1 \oplus (2 \bullet s_2) \oplus (3 \bullet s_3) \\ (3 \bullet s_0) \oplus s_1 \oplus s_2 \oplus (2 \bullet s_3) \end{bmatrix}.$$

Implementación de MixColumns

```
def MixColumns(s):
  S = matrix(ZZ,4,4,s)
  S = S.transpose()
  T = matrix(ZZ,4,4,[2,3,1,1,1,2,3,1,1,1,2,3,3,1,1,2])
  ST = matrix(ZZ,4,4)
  for i in range(4):
    for j in range(4):
      for k in range(4):
        ST[i,j] = ST[i,j] ^^ mul(T[i,k],S[k,j])
  return [ST[i%4,i//4] for i in range(16)]
```

Valores Iniciales

- La clave está formada por 128 bits que hemos supuesto que forman una matriz de tamaño 4×4.

Valores Iniciales

- La clave está formada por 128 bits que hemos supuesto que forman una matriz de tamaño 4×4.
- Las columnas de esta matriz inicial las llamaremos w_0, w_1, w_2 y w_3.

Valores Iniciales

- La clave está formada por 128 bits que hemos supuesto que forman una matriz de tamaño 4×4.
- Las columnas de esta matriz inicial las llamaremos w_0, w_1, w_2 y w_3.
- Aunque debemos pensar en ellas como columnas, para escribirlas es más cómodo hacerlo horizontalmente, así por ejemplo si $w_0 = \begin{bmatrix} 2b \\ 7e \\ 15 \\ 16 \end{bmatrix}$ escribiremos $w_0 = 2b7e1516$.

Valores Iniciales

- La clave está formada por 128 bits que hemos supuesto que forman una matriz de tamaño 4×4.
- Las columnas de esta matriz inicial las llamaremos w_0, w_1, w_2 y w_3.
- Aunque debemos pensar en ellas como columnas, para escribirlas es más cómodo hacerlo horizontalmente, así por ejemplo si $w_0 = \begin{bmatrix} 2b \\ 7e \\ 15 \\ 16 \end{bmatrix}$ escribiremos $w_0 = 2b7e1516$.
- El proceso de expansión de la clave consiste en calcular a partir los valores iniciales, valores sucesivos $w_4, w_5, w_6, \cdots$ hasta completar los necesarios para hacer el cifrado.

Cálculo de w_4

- Vamos a suponer que nuestra clave es 2b7e151628aed2a6abf7158809cf4f3c y por lo tanto $w_0 =$ 2b7e1516, $w_1 =$ 28aed2a6, $w_2 =$ abf71588 y $w_3 =$ 09cf4f3c.

Cálculo de w_4

- Vamos a suponer que nuestra clave es 2b7e151628aed2a6abf7158809cf4f3c y por lo tanto $w_0 =$ 2b7e1516, $w_1 =$ 28aed2a6, $w_2 =$ abf71588 y $w_3 =$ 09cf4f3c.
- Estamos calculando w_i con $i = 4$ y partimos del último valor que tenemos $w_3 =$ 09cf4f3c.

Cálculo de w_4

- Vamos a suponer que nuestra clave es `2b7e151628aed2a6abf7158809cf4f3c` y por lo tanto $w_0 =$ `2b7e1516`, $w_1 =$ `28aed2a6`, $w_2 =$ `abf71588` y $w_3 =$ `09cf4f3c`.
- Estamos calculando w_i con $i = 4$ y partimos del último valor que tenemos $w_3 =$ `09cf4f3c`.
- La primera operación que hacemos es rotar un byte, es decir, poner el primer byte al final y desplazar los otros a la izquierda, en este caso `cf4f3c09`.

Cálculo de w_4

- Vamos a suponer que nuestra clave es `2b7e151628aed2a6abf7158809cf4f3c` y por lo tanto $w_0 =$ `2b7e1516`, $w_1 =$ `28aed2a6`, $w_2 =$ `abf71588` y $w_3 =$ `09cf4f3c`.
- Estamos calculando w_i con $i = 4$ y partimos del último valor que tenemos $w_3 =$ `09cf4f3c`.
- La primera operación que hacemos es rotar un byte, es decir, poner el primer byte al final y desplazar los otros a la izquierda, en este caso `cf4f3c09`.
- Luego aplicamos `SubBytes` a cada uno de los cuatro bytes de `cf4f3c09`, con lo que obtenemos `8a84eb01`.

Cálculo de w_4

- Vamos a suponer que nuestra clave es `2b7e151628aed2a6abf7158809cf4f3c` y por lo tanto $w_0 =$ `2b7e1516`, $w_1 =$ `28aed2a6`, $w_2 =$ `abf71588` y $w_3 =$ `09cf4f3c`.
- Estamos calculando w_i con $i = 4$ y partimos del último valor que tenemos $w_3 =$ `09cf4f3c`.
- La primera operación que hacemos es rotar un byte, es decir, poner el primer byte al final y desplazar los otros a la izquierda, en este caso `cf4f3c09`.
- Luego aplicamos `SubBytes` a cada uno de los cuatro bytes de `cf4f3c09`, con lo que obtenemos `8a84eb01`.
- A este número le haremos el *o exclusivo* con la constante de ronda `Rcon` que hemos descrito en la sección anterior, lo que nos da `8b84eb01` (fijémonos que es la primera constante de ronda, `01000000` y por lo tanto solo nos ha cambiado un bit).

Cálculo de w_4

- Vamos a suponer que nuestra clave es `2b7e151628aed2a6abf7158809cf4f3c` y por lo tanto $w_0 =$ `2b7e1516`, $w_1 =$ `28aed2a6`, $w_2 =$ `abf71588` y $w_3 =$ `09cf4f3c`.
- Estamos calculando w_i con $i = 4$ y partimos del último valor que tenemos $w_3 =$ `09cf4f3c`.
- La primera operación que hacemos es rotar un byte, es decir, poner el primer byte al final y desplazar los otros a la izquierda, en este caso `cf4f3c09`.
- Luego aplicamos `SubBytes` a cada uno de los cuatro bytes de `cf4f3c09`, con lo que obtenemos `8a84eb01`.
- A este número le haremos el *o exclusivo* con la constante de ronda `Rcon` que hemos descrito en la sección anterior, lo que nos da `8b84eb01` (fijémonos que es la primera constante de ronda, `01000000` y por lo tanto solo nos ha cambiado un bit).
- A este valor finalmente le haremos un *o exclusivo* con w_{i-4}, es decir, con w_0 y el resultado será w_4.

Cálculo de w_i

- En general, cuando tengamos que calcular w_i partiremos del último valor conocido w_{i-1} y dependiendo de si i es un múltiplo de 4 o no, haremos lo siguiente:

Cálculo de w_i

- En general, cuando tengamos que calcular w_i partiremos del último valor conocido w_{i-1} y dependiendo de si i es un múltiplo de 4 o no, haremos lo siguiente:
  - Si i es un múltiplo de 4, es decir, $4, 8, 12, \cdots$, haremos lo mismo que hemos visto en el caso de w_4, rotar, aplicar `SubBytes`, $\oplus$ con la constante de ronda y finalmente $\oplus$ con w_{i-4}.

Cálculo de w_i

- En general, cuando tengamos que calcular w_i partiremos del último valor conocido w_{i-1} y dependiendo de si i es un múltiplo de 4 o no, haremos lo siguiente:
  - Si i es un múltiplo de 4, es decir, $4, 8, 12, \cdots$, haremos lo mismo que hemos visto en el caso de w_4, rotar, aplicar `SubBytes`, $\oplus$ con la constante de ronda y finalmente $\oplus$ con w_{i-4}.
  - Si i no es múltiplo de 4 simplemente haremos $w_i = w_{i-1} \oplus w_{i-4}$

Cálculo de w_i

- En general, cuando tengamos que calcular w_i partiremos del último valor conocido w_{i-1} y dependiendo de si i es un múltiplo de 4 o no, haremos lo siguiente:
  - Si i es un múltiplo de 4, es decir, $4, 8, 12, \cdots$, haremos lo mismo que hemos visto en el caso de w_4, rotar, aplicar `SubBytes`, $\oplus$ con la constante de ronda y finalmente $\oplus$ con w_{i-4}.
  - Si i no es múltiplo de 4 simplemente haremos $w_i = w_{i-1} \oplus w_{i-4}$
- Para el cifrado de 128 bits tendremos que calcular hasta el w_{43}.

Implementación de la Expansión de la Clave

```
def keyExpansion(Key):
  w = [[Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]] for
                                        i in range(4)]
  for i in range(4,44):
    temp = w[-1]
    if i%4 == 0:
      temp = [temp[1],temp[2],temp[3],temp[0]]
      temp = [SB[temp[j]] for j in range(4)]
      r = Rcon(i//4)
      temp = [temp[j]^^r[j] for j in range(4)]
    temp = [temp[j]^^w[i-4][j] for j in range(4)]
    w.append(temp)
  return w
```

Primer Paso

- Dada una entrada que representaremos por una matrix de bytes tamaño 4×4 lo primero que haremos es hacer *o exclusivo* con la clave representada en columnas $[w_0, w_1, w_2, w_3]$

Primer Paso

- Dada una entrada que representaremos por una matrix de bytes tamaño 4×4 lo primero que haremos es hacer *o exclusivo* con la clave representada en columnas $[w_0, w_1, w_2, w_3]$
- Representado en el código es

```
for fil in range(4):
 for col in range(4):
  s[4*col+fil] = s[4*col+fil]^^w[4*i+col][fil]
```

Rondas desde la 1 a la 9

- En las rondas de la 1 a la 9 hay que hacer los siguientes pasos:

Rondas desde la 1 a la 9

- En las rondas de la 1 a la 9 hay que hacer los siguientes pasos:
  - Aplicar `SubBytes` a todos los bytes de la matriz de estado.

Rondas desde la 1 a la 9

- En las rondas de la 1 a la 9 hay que hacer los siguientes pasos:
  - Aplicar `SubBytes` a todos los bytes de la matriz de estado.
  - Luego se hace `ShiftRows`.

Rondas desde la 1 a la 9

- En las rondas de la 1 a la 9 hay que hacer los siguientes pasos:
  - Aplicar `SubBytes` a todos los bytes de la matriz de estado.
  - Luego se hace `ShiftRows`.
  - En tercer lugar se hace la operación `MixColumns`

Rondas desde la 1 a la 9

- En las rondas de la 1 a la 9 hay que hacer los siguientes pasos:
  - Aplicar `SubBytes` a todos los bytes de la matriz de estado.
  - Luego se hace `ShiftRows`.
  - En tercer lugar se hace la operación `MixColumns`
  - Finalmente, la matriz de estado se opera mediante *o exclusivo* con la matriz de claves $[w_{4i}, w_{4i+1}, w_{4i+2}, w_{4i+3}]$

La Ronda número 10

- La ronda número 10 es similar a las rondas anteriores, salvo por el hecho de que no se hace la operación `MixColumns`

La Ronda número 10

- La ronda número 10 es similar a las rondas anteriores, salvo por el hecho de que no se hace la operación `MixColumns`
- Tras terminar la operación `ShiftRows` se hace el *o exclusivo* con la matriz de claves que corresponde y el resultado es la salida del cifrado.

Código Completo I

```
SB = [ 0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x1, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
       0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
       0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
       0x4, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x5, 0x9a, 0x7, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
       0x9, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
       0x53, 0xd1, 0x0, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
       0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x2, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
       0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
       0xcd, 0xc, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
       0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0xb, 0xdb,
       0xe0, 0x32, 0x3a, 0xa, 0x49, 0x6, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
       0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x8,
       0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
       0x70, 0x3e, 0xb5, 0x66, 0x48, 0x3, 0xf6, 0xe, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
       0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
       0x8c, 0xa1, 0x89, 0xd, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0xf, 0xb0, 0x54, 0xbb, 0x16]

def SubBytes(s):
  return [SB[t] for t in s]

def mul(t,s):
  if t == 1:
    return s
  elif t == 2:
    s = s*2
    if s>=256:
```

Código Completo II

```
      s = (s-256)^^0x1b
    return s
  else:
    return s^^mul(2,s)

Rc = [1]
for i in range(9):
  Rc.append(mul(2,Rc[-1]))

def Rcon(i):
  return [Rc[i-1],0,0,0]

def keyExpansion(Key):
  w = [[Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]] for i in range(4)]
  for i in range(4,44):
    temp = w[-1]
    if i%4 == 0:
      temp = [temp[1],temp[2],temp[3],temp[0]]
      temp = [SB[temp[j]] for j in range(4)]
      r = Rcon(i//4)
      temp = [temp[j]^^r[j] for j in range(4)]
    temp = [temp[j]^^w[i-4][j] for j in range(4)]
    w.append(temp)
  return w

def ShiftRows(s):
  return [s[0],s[5],s[10],s[15],s[4],s[9],s[14],s[3],s[8],s[13],s[2],s[7],s[12],s[1],s[6],s[11]]
```

Código Completo III

```
def MixColumns(s):
  S = matrix(ZZ,4,4,s)
  S = S.transpose()
  T = matrix(ZZ,4,4,[2,3,1,1,1,2,3,1,1,1,2,3,3,1,1,2])
  ST = matrix(ZZ,4,4)
  for i in range(4):
    for j in range(4):
      for k in range(4):
         ST[i,j] = ST[i,j] ^^ mul(T[i,k],S[k,j])
  return [ST[i%4,i//4] for i in range(16)]

def pr(s):
  for fil in range(4):
    for col in range(4):
      print hex(s[fil+4*col]),
    print

def aes(Input,Key):
  w = keyExpansion(Key)
  s = Input
  for i in range(11):
    if i!=0:
      s = SubBytes(s)
      print "Al Terminar SubBytes"
      pr(s)
      s = ShiftRows(s)
```

Código Completo IV

```
      print "Al Terminar ShiftRows"
      pr(s)
    if i!=0 and i!=10:
      s = MixColumns(s)
      print "Al Terminar MixColumns"
      pr(s)
    for fil in range(4):
      for col in range(4):
        s[4*col+fil] = s[4*col+fil]^^w[4*i+col][fil]
    print "Al Terminar Ronda ",i
    pr(s)
  return s

key = 0x2b7e151628aed2a6abf7158809cf4f3c
Key = ZZ(key).digits(256)
Key.reverse()

input = 0x3243f6a8885a308d313198a2e0370734
Input = ZZ(input).digits(256)
Input.reverse()

aes(Input,Key)
```