Modos de cifrado AES

AES se puede utilizar en varios modos. Vamos a verlos en esta actividad.

Vamos a crear:

- un mensaje de 128 bits, el tamaño de bloque de AES.
- una clave de 128 bits

Recuerda: si no tienes la librería Crypto, tienes que instalarla con el siguiente comando:

```
python3 -m pip install pycryptodome
```

In [1]:

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from base64 import b64encode, b64decode

m = b'abcdefghabcdefgh'
k = get_random_bytes(16)
print(f'Mensaje: "{m}" Tamaño={len(m) * 8} bits')
print(f'Clave: {b64encode(k)} Tamaño={len(k) * 8} bits')
```

```
Mensaje: "b'abcdefghabcdefgh'" Tamaño=128 bits
Clave: b'FHU3F1/ysxZXdk32iybV8g==' Tamaño=128 bits
```

Modo ECB

Ciframos dos veces el mismo mensaje.

Observa que no hay memoria, y que cifrar dos veces el mismo mensaje con la misma clave produce el mismo texto cifrado.

In [2]:

```
cipher = AES.new(k, AES.MODE_ECB)
c1 = cipher.encrypt(m)
c2 = cipher.encrypt(m)
print(b64encode(c1))
print(b64encode(c2))
```

```
b'4Y1WI2m3JLAGfJRnCdIs9Q=='
b'4Y1WI2m3JLAGfJRnCdIs9Q=='
```

In [3]:

```
decipher = AES.new(k, AES.MODE_ECB)
m1 = decipher.decrypt(c1)
m2 = decipher.decrypt(c2)
print(m1)
print(m2)
```

```
b'abcdefghabcdefgh'
b'abcdefghabcdefgh'
```

Modo CBC

Ciframos dos veces el mismo mensaje. Observa que:

- tenemos que crear un IV (Vector de Inicialización), y que este IV se lo tenemos que enviar al receptor. El envío del IV puede ser en claro
- Ahora los dos cifrados son diferentes, a pesar de que estamos cifrando el mismo mensaje. ¿Por qué sucede eso?

In [4]:

```
iv = get_random_bytes(16)
cipher = AES.new(k, AES.MODE_CBC, iv=iv)
c1 = cipher.encrypt(m)
c2 = cipher.encrypt(m)
print(b64encode(c1))
print(b64encode(c2))
```

```
b'daTK3q9qgoN9V1aUduzE0A=='
b'b3zMEO7eIDQDBwCrTqi00w=='
```

Descifrado: necesita la clave y el IV. La clave es secreta y el receptor tiene que haberla recibido por un canal secreto (lo veremos) pero el IV puede recibirse sin protección al inicio de la comunicación.

In [5]:

```
decipher = AES.new(k, AES.MODE_CBC, iv=iv)
m1 = decipher.decrypt(c1)
m2 = decipher.decrypt(c2)
print(m1)
print(m2)
```

```
b'abcdefghabcdefgh'
b'abcdefghabcdefgh'
```

Ejercicios (opcional)

- ¿Puedes programar el modo CBC a partir del modo ECB? ECB es la caja AES básica, así que es posible programar (¡como ejercicio solamente!) el modo CBC como composición de ECB
- ¿Puedes programar los demás modos?

Ejemplo de solución (solo parte de cifrado) de la primera pregunta. Observa que el resultado es el mmismo de antes al cifrar m en modo CBC

In [6]:

```
from Crypto.Util.strxor import strxor

class AES_CBC():
    def __init__(self, iv):
        self.iv = iv
        self.cipher = AES.new(k, AES.MODE_ECB)
    def encrypt(self, msg):
        # primero hacemos XOR del mensaje con el IV que tenemos
        m = strxor(msg, self.iv)
        c = self.cipher.encrypt(m)
        # para la siguiente ronda, el IV es el propio texto cifrado
        self.iv = c
        return c

mycbc = AES_CBC(iv)
print(b64encode(mycbc.encrypt(m)))
print(b64encode(mycbc.encrypt(m)))
```

```
b'daTK3q9qgoN9V1aUduzE0A=='
b'b3zMEO7eIDQDBwCrTqi00w=='
```

Padding

¿Qué pasa si tenemos que enviar mensajes más cortos que la longitud de bloque de AES? Entonces tenemos que usar algún algoritmo de padding. Es decir: marcar la longitud del mensaje original.

Con Cryptodome podemos usar las funciones pad y unpad

Observa: no ponemos IV, así que la librería lo escoge aleatorio. En modo CBC solo tenemos que enviar el IV la primera vez,

In [7]:

```
from Crypto.Util.Padding import pad, unpad

# mensaje corto
m = b'1234'
cipher = AES.new(k, AES.MODE_CBC)
c = cipher.encrypt(pad(m, AES.block_size))
print({'iv':b64encode(cipher.iv), 'ciphertext':b64encode(c)})
```

```
{'iv': b'81wPaIhEX/ccoEVhVKJAsg==', 'ciphertext': b'JjLUdlylAp0gXuspMF
a/Tg==''}
```

Recepción:

In [8]:

```
decipher = AES.new(k, AES.MODE_CBC, cipher.iv)
pt = unpad(decipher.decrypt(c), AES.block_size)
print("The message was: ", pt)
```

```
The message was:  b'1234'
```

¿Qué pasa si no usamos unpad? AES es un cifrado de bloque, así que los mensajes en AES tienen

obligatoriamente un tamaño igual al bloque AES (128 bits), así que vemos el *padding* que sobra. Las función *unpad()* nos hubiese cortado esos bytes sobrantes.

(Observa: tenemos que volver a recrear el decipher, porque tiene memoria y queremos volver a descifrar el mismo mensaje)

In [9]:

```
decipher = AES.new(k, AES.MODE_CBC, cipher.iv)
pt = decipher.decrypt(c)
print(f"The message was: {pt} (longitud {len(pt) * 8} bits)")
```

```
The message was: b'1234\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0
c' (longitud 128 bits)
```