Cifrando con XORs

En este ejercicio vamos a estudiar un sistema de cifrado que solo usa XOR con claves cortas, como el que vimos en el ejercicio 1. Este tipo de sistema de cifrado está completamente roto porque reutiliza la clave, y no debe usarse. En este ejercicio veremos por qué.

Nota: XOR se sigue utilizando internamente en algunos pasos de los sistemas de cifrado actuales, y es correcto porque hacen muchas más cosas además de XOR.

La función siguiente cifra un texto con una clave utilizando XOR. Cuando se acaban los caracteres de la clave, vuelve a empezar por el primero.

No es necesario entender la función, especialmente porque no es la mejor, ni más eficiente, ni tiene control de errores

In [2]:
```
from itertools import cycle

def xor(key, data):
    """
    xor de key y data. Si len(key)<len(data), reutiliza la key.

    Tanto key como data tienen que ser arrays de bytes.

    Devuelve un array de bytes
    """
    output = []
    for d, k in zip(data, cycle(key)):
        output.append(k ^ d)
    return ''.join(map(chr, output)).encode()
```

Como hemos visto, cifrar con XORs ofrece confidencialidad perfecta siempre que se cumplan las condiciones. Es decir. Es decir, que la clave:

- Sea tan larga como el mensaje
- No se reutilice nunca más para cifrar ningún otro mensaje (*one-time-pad*)

Es decir, la función de arriba ofrece confidencialidad perfecta, y se puede demostrar matemáticamente que es imposible programar nada más seguro que esa función de arriba... si se cumplen sus condiciones de uso.

Veremos qué pasa cuando no se cumplen estas condiciones.

Una persona quiere enviar el texto `Tres tristes tigres` cifrado con XOR y clave `SESAMO`

(En Python, cuando ponemos `b` al inicio de una cadena, queremos que se interprete como un array de bytes, no como un texto. Será más conveniente trabajar con arrays de bytes en nuestros ejemplos)

In [3]:
```
data = b'TRES TRISTES TIGRES'
key = b'SESAMO'
```

Ahora ciframos. Observa que la salida no es legible, son una series de bytes. En general, un XOR de un caracter visible no es imprimible y por eso no vemos nada

In [4]:
```
c = xor(key, data)
print(c)
```

```
b'\x07\x17\x16\x12m\x1b\x01\x0c\x00\x15\x08\x1cs\x11\x1a\x06\x1f\n\x00'
```

Vamos a usar base64 para poder ver algo. Recuerda: Base64 **no es un cifrado**, es una manera de codificar mensajes binarios con caracteres imprimibles. Se usa, por ejemplo, para enviar fotografías por correo electrónico (el estándar de correo electrónico solo permite enviar caracteres imprimibles).

Pero Base64 no es un cifrado: no tiene clave y siempre se puede deshacer. Solo lo usamos porque es cómodo y común tener caracteres imprimibles. Esto incluso era una de los principios de Kerckhoffs que se sigue por comodidad, aunque no sea estríctamente necesario.

In [5]:
```
from base64 import b64encode, b64decode
cb = b64encode(c)
print(cb)
```

```
b'BxcWEm0bAQwAFQgccxEaBh8KAA=='
```

Este es el mensaje que le enviamos al a otra parte, que puede deshacerlo con el mismo algoritmo si conoce la clave

In [6]:
```
print(xor(key, b64decode(cb)))
```

```
b'TRES TRISTES TIGRES'
```

Rompiendo XOR

Fíjate en el mensaje anterior: la clave es más corta que el mensaje. Cuando la clave rota, se reutiliza para cifrar el mensaje.

Eso es un error que vamos a aprovechar. **Nunca se debe reutilizar una clave**. En esa ocasión, nos pasa lo mismo que con los audiocuentos del primer día.

Vamos a ver un ejemplo sencillo: el emisor envía un texto "Envía 1000 E" a su banco usando este protocolo sencillo y clave aleatoria.

(vamos a suponer que no hay letras acentuadas. La codificación adicional que tienen los acentos nos complicaría el sistema)

Lo que envía el cliente al su banco:

In [7]:
```
k = b'1GR2f9'
m = b'Envia 1000 E'
c = xor(k, m)
print(b64encode(c))
```

```
b'dCkkWwcZAHdiAkZ8'
```

Supongamos que el atacante recibe c porque está espiando y sabe:

- que muchos mensajes al banco empiezan con "Envia "
- que las claves tienen 6 letras.

Esto es razonable, ¿no? Estas suposiciones ni siquiera son demasiado exigentes. En realidad el atacante puede probar con claves de 5 letras, o con 7, o con otros encabezados ("transfiere...") hasta que le salga un mensaje coherente.

Para descifrarlo:

- El atacante toma el mensaje cifrado y lo corta en grupos tan grandes como supone que es la clave. Es decir, de 6 letras cada uno: c1 y c2
- Hace XOR con el mensaje que ha supuesto: "Envia "

Fíjate que en ninguna de estas líneas que ejecuta el atacante está la clave, solo utiliza cosas que sabe porque están en canales inseguros: el texto cifrado c

In [8]:
```
c1 = c[:6]
c2 = c[6:]
print(xor(b'Envia ', xor(c1, c2)))
```

```
b'1000 E'
```

¡Y aparece la otra parte del mensaje! En ese momento SABE que sus suposiciones son buenas, así que puede sacar la clave con `Envia` y c1

In [9]:
```
print(xor(b'Envia ', c1))
```

```
b'1GR2f9'
```

Lo vamos a repetir muchas veces en este curso: **no se puede cifrar dos veces con la misma clave**

Veamos otro ejemplo: el usuario envía dos mensaje cifrados con la misma clave: un saludo y una orden

In [10]:
```
k = b'1GR2f9'
m1 = b'Hola Jose Antonio'
c1 = xor(k, m1)
m2 = b'Tienes que ejecutar compra de 1000 acciones de SANTACO a las 14h'
```

Supongamos que las comunicaciones usan un protocolo inventado que necesita que los mensajes tengan obligatoriamente 64 bytes, y si no los tiene rellena con ceros

In [11]:
```
m1relleno = m1 + (b'\x00' * (64 - len(m1)))
print(m1relleno)
m2relleno = m2 + (b'\x00' * (64 - len(m2)))
print(m2relleno)
```

```
b'Hola Jose Antonio\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00'
b'Tienes que ejecutar compra de 1000 acciones de SANTACO a las 14h'
```

Ahora nuestro usuario está prevenido, así que usa una clave de 64 bytes, que son 512 bits, totalmente aleatoria y ha leído que eso es muy segura por ser de 512 bits y por ser aleatoria.

In [12]:
```
k = b'1292jfmfiw8222aR2Xv3v395u5k202931292jJmfAw81L2aa2aa3Z325u5k2M292'
print(len(k) * 8)
```

```
512
```

Y cifra los dos mensajes con esa clave

In [13]:
```
c1 = xor(k, m1relleno)
print(b64encode(c1))
c2 = xor(k, m2relleno)
print(b64encode(c2))
```

```
b'eV1VU0osAhUMV3lcRl0PO11YdjN2Mzk1dTVrMjAyOTMxMjkyakptZkF3ODFMMmFhMmFhM1ozMjV1NWsyTTI5Mg
==
b'ZVtcXA8VTRccEhhXWFcCJ0Y5BBMVXFRFB1RLVlUSCAMBAhlTCSkECS8SSxEoV0Eycy81chl8ElRVWQpBbQMNWg
==
```

El atacante supongamos que no tiene forma de adivinar el primer mensaje pero sabe que es algún tipo de saludo poco interesante ("hola", "buenas", "Tengo un asunto urgente"...), y que el segundo mensaje es donde está la información aunque tampoco puede adivinar nada de ese mensaje.

Dada la diferencia en tamaño de los mensajes, y que sabe que el primero estará rellano con ceros... solo tiene que hacer XOR de los dos textos que recibe cifrados para ver la parte del mensaje que le interesa:

In [14]:
```
print(xor(c1, c2))
```

```
b'\x1c\x06\t\x0fE9O\x02\x10Ea\x0b\x1e\n\r\x1c\x1bar compra de 1000 acciones de SANTACO a
las 14h'
```

Esto no solo pasa con este cifrado simple sino con cualquier cifrado simétrico, aunque no siempre es tan evidente.

NUNCA HAY QUE REUTILIZAR LA CLAVE DE CIFRADO EN DOS MENSAJE DIFERENTES