# Cifrado de flujo en Criptografia

Valentina S. Vispo

Octubre 2023

# Índice

# 1 Cifrado de flujo

## 1.1 De que se trata

Un cifrado de flujo es un cifrado de clave simetrica que genera una secuencia de simbolos pseudoaleatorios y los combina con los simbolos del texto claro (o texto plano) mediante alguna operacion binaria inversible sencilla.

$p$: texto claro o texto plano

$k$: secuencia cifrante (secuencia de bits)

$C$: texto cifrado (secuencia de bits)

Para cifrar

$$C_i = p_i \oplus k_i$$

Para obtener el texto claro, es decir, descifrar el mensaje, se realiza un or-exclusivo ($\oplus$) entre el texto cifrado $C_i$ y la misma secuencia de bits $k_i$

Para descifrar

$$p_i = C_i \oplus k_i$$

### 1.1.1 Propiedades

Para que un cifrado de flujo sea seguro debe cumplir las siguientes propiedades:

1. next bit property
2. inversible

### 1.1.2 Problemas

No debe reutilizarse o repetirse la secuencia cifrante ($k$) porque si se repite la clave, se repite la secuencia cifrante.

Maleables: una modificacion de un bit en el texto cifrado produce una modificacion del bit correspondiente en el texto claro. Un atacante puede asi modificar el texto claro sin necesidad de conocer la clave.

# 2 Desafio: Secuencia cifrante repetida

## 2.1 Enunciado

El desafío consiste en descifrar mensajes que han sido cifrados con una secuencia cifrante repetida, e identificar dicha secuencia.

El servidor provee 5 textos cifrados con un algoritmo de flujo. Tres de estos textos fueron cifrados con la misma secuencia cifrante ademas de cumplir ciertas caracteristicas:

- Están compuestos por caracteres ASCII imprimibles, sin espacios en blanco ni fines de línea. Esto implica que sus códigos están entre 33 y 126 inclusive.
- Dos textos están compuestos por repeticiones de un mismo carácter.
- Todos los textos tienen la misma longitud. Uno de los textos es la representación decimal de un número par. Por ejemplo “12345678”

## 2.2 Solucion

**Notas a tener en cuenta**

1. La respuesta será una secuencia de bytes codificada en base64
2. La respuesta contiene los cinco textos cifrados, uno por línea, y codificados individualmente en base64

**Pasos**

1. Obtener las 5 textos cifrados, los separo por cada texto (c/u de 52 caracteres) y los decodeo de base64 a bytes

```
[Decoded texts]

1st: b'p\x8cE\xf0\x10*\xd9d\x81\xbe2\xd6\x19\xc0\x8d\n\xc5\xde\xe3\x9cLu\x

2nd: b'^e5\xd2\x01\x9a\xf0\xe4\xa6\xd0DN\xf8\xf5\x1c\xb1\n\xaf\xc1\xf5\xf

3rd: b'7\x0cX\xbdc\xf6\x9c\x8a\xcb\xb8—#\x96\x99q\xdef\xc1\xab\x96\x9d`\x
fbc\xdac\xd4\xbajv'

4th: b'#\x18L\xa9w\xe2\x88\x9e\xdf\xac97\x82\x8de\xcar\xd5\xbf\x82\x89t\x

5th: b'\xc5\x14\xca\xd6b[/t1||q\x1aB\x88\x0f1\x91o\x9d\xec\x8b\xdd\x14\x0
```

2. Realizo un xor entre los diferentes textos cifrados. Observo el texto obtenido y trato de identificar algun patron para asumir que ese es el texto cifrado por el generador

```
# Es indiferente el orden del xor

# 1 XOR 2
b'.\xe9p”\x11\xb0)\x80\'nv\x98\xe15\x91\xbb\xcfq”i\xba}cu\xa1\xe0\x90\xb9W\xad
    #b#\xb5=\xa1Y\xdd\x15'
```

```
# 1 XOR 3
b'G\x80\x1dMs\xdcE\xeeJ\x06\x1f\xf5\x8fY\xfc\xd4\xa3\x1fH\n\xd1\x15\x0b\x17\
    xcf\x8b\xf9\xd5>\xc0N\x00M\xdbR\xc34\xb6w'

# 1 XOR 4
b'S\x94\tYg\xc8Q\xfa^\x12\x0b\xe1\x9bM\xe8\xc0\xb7\x0b\\\x1e\xc5\x01\x1f\x03\
    xdb\x9f\xed\xc1*\xd4Z\x14Y\xcfF\xd7␣\xa2c'
1x5 b'\xb5\x98\x8f&rq\xf6\x10\xb0\xc2N\xa7\x03\x82\x05\x05\xf4O\x8c\x01\xa0\xfe
    ^\x03uPFO\xbaJ`]\x97W\xb4\xab\xc4\x02\xf6'

# 2 XOR 3 # Este interesa!
b'iimobllnmhimnlmolnjckhhbnkilimmbnnobmkb'

# 2 XOR 4
b'}}y{vxxzy|}yzxy{xz~w\x7f||vz\x7f}x}yyvzz{vy\x7fv'

# 2 XOR 5
b'\x9bq\xff\x04c\xc1\xdf\x90\x97\xac8?\xe2\xb7\x94\xbe;>\xaeh\x1a\x83=v\xd4\xb0
    \xd6\xf6\xed\xe7C?\xb4\xe2\x89\n\x9d\xdf\xe3'

# 3 XOR 4 # Este interesa! - 3 y 4 son dos textos que estan compuestos por repeticiones
    de un mismo caracter
b'\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14
    \x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\x14\
    x14\x14'

# 3 XOR 5
b'\xf2\x18\x92k\x01\xad\xb3\xfe\xfa\xc4QR\x8c\xdb\xf9\xd1WP\xc4\x0bq\xebU\x14
    \xba\xdb\xbf\x9a\x84\x8a.]\xda\x8c\xe6h\xf0\xb4\x81'

# 4 XOR 5
b'\xe6\x0c\x86\x7f\x15\xb9\xa7\xea\xee\xd0EF\x98\xcf\xed\xc5CD\xd0\x1fe\xffA\
    x00\xae\xcf\xab\x8e\x90\x9e:I\xce\x98\xf2|\xe4\xa0\x95'
```

3. Empezamos a analizar el $c_3 \oplus c_4$ (o $c_4 \oplus c_3$):

Ahora, que par de caractares generarian este $x14$?

Primero tenemos que $x14 =' 20'$, luego que para que toda una secuencia sean los mismos caracteres, debe ser el resultado de xor entre los dos textos que estan compuestos por repeticiones de un mismo caracter. Despues hay que revisar que par de caracteres (siempre en ASCII) que den como resultado b'\x14'

```
# ...
valid_key_bytes = set()

# Los que dan x\14
```

```
for a in range(33, 127):
    b_a = bytes(chr(a), 'ascii')
    for b in range(33, 127):
        b_b = bytes(chr(b), 'ascii')
        if (byte_xor(b_a, b_b) == b'\x14'):
            valid_key_bytes.add(b_a)
            valid_key_bytes.add(b_b)
# ...
```

4. Luego con la cantidad de pares de caracteres posibles, se crea una secuencia de ese caracter repetido la longitud del keystream elegido.

```
Lenght of keystream: 39
Cantidad de posibles keystream: 92
```

5. Se realiza una operacion XOR entre la palabra 3 o 4 con la secuencia generada, si el resultado de este xor da que las palabras son iguales, entonces es keystream valido.

6. Luego se itera sobre cada key de los posibles keystream, pero esta vez sobre $c_2$ que es otro de los textos que nos interesa.

7. Si alguno de estos xor, tomando solo los primeros 39 caracteres, da como resultado un numero que es par se encontro el keystream valido y se procede a realizar el envio.

```
FOUND!: 337586647237467564091228413637784458718
b'bVYC5zmsxtCR4nd5zMMrhDyb8czHOtJa7rh2FCopQqE5gDmO4DAs'
Ganaste
```

### 2.2.1 Como correr la solución

```
python3 challenges/4_Stream.py
```

## 2.3 Evitar el problema

Para evitar el problema, un concepto importante de tener en cuenta es el *nonce* (Number used only once) o IV (Initial Value).

1. No debe repetirse la secuenia cifrante: Si se repite la clave, se repite la secuencia cifrante
2. Utilizar un *nonce*: permite reutilizar la clave siempre y cuando no se repita el *nonce*.
3. Si se utiliza un *nonce*, no repetirlo.

## 2.4 Herramientas

Se utilizo la siguiente funcion para realizar las operaciones de xor:

```
1  def byte_xor(xs, ys):
2      return bytes(x ^ y for (x, y) in zip(xs, ys))
```

## 2.5 Referencias


1. Enunciado ejercicio https://ciberseguridad.diplomatura.unc.edu.ar/cripto/doc/stream.html
2. https://es.wikipedia.org/wiki/Cifrador_de_flujo
3. https://en.wikipedia.org/wiki/List_of_logic_symbols