Seguridad por oscuridad: introducción al criptoanális

Varias editoriales de todo el mundo han publicado una línea de cuentos infantiles en forma de coleccionable por entregas. El producto de ambas es similar: un altavoz con figuras coleccionables, el cuento suena cuando se acerca la figura al altavoz.

En algunos de estos productos, los cuentos se guardan en una tarjeta SD **en el altavoz**. El cuento no está en la figura: eso provocaría que las figuras fuesen demasiado caras. Todos los cuentos están en la memoria del altavoz, mientras que las figuras tienen etiquetas NFC o RFID pasivas con un identificador. Una de las editoriales, para evitar que se pueda acceder a los cuentos sin comprar la figura, ha decidido cifrar los archivos de audio...

...pero han confiado en la seguridad por oscuridad

(otros productos no ha protegido los archivos de audio cuentos y ha confiado en que un particular no comprará etiquetas RFID, que son la que llevan las figuras)

Vamos a analizar el sistema de cifrado que se utiliza en los audiocuentos.

La clase HexView que está a continuación es simplemente una forma sencilla de mostrar el contenido de un archivo en hexadecimal en estos notebook. Podrías también ejecutar un hexdump en línea de comandos.

In [1]:
```
import os

class HexViewer():
    def __init__(self, filename, bs=16, count=32):
        self.filename = filename
        self.bs = bs
        self.count = count
    def get_blocks(self, offset=0):
        assert self.filename
        with open(self.filename, "rb") as file:
            try:
                file.seek(offset, os.SEEK_SET)
                block = file.read(self.bs * self.count)
            except ValueError: # Empty offsetSpinbox
                return
        rows = [block[i:i + self.bs]
                for i in range(0, len(block), self.bs)]
        for row in rows:
            yield '{} - {}'.format(self.get_bytes(row), self.get_ascii(row))
    def get_bytes(self, row):
        output = " ".join(map(lambda b:"{:02X}".format(b), row))
        if len(row) < self.bs:
            output += " " * (self.bs - len(row)) * 3
        return output
    def get_ascii(self, row, not_printable='.'):
        output = []
        for char in row.decode('ascii', errors="replace"):
            if char in "\u2028\u2029\t\n\r\v\f\uFFFD":
                char = not_printable
            elif not 0x20 <= ord(char) <= 0xFF:
                char = not_printable
            output.append(char)
        return "".join(output)
    def print_blocks(self, offset=0):
        print('\n'.join(self.get_blocks(offset)))
```

Bien, en este mismo directorio hay un archivo `test.mp3` de prueba que hemos descargado de: https://file-examples.com/index.php/sample-audio-files/sample-mp3-download/ (https://file-examples.com/index.php/sample-audio-files/sample-mp3-download/)

Que el archivo cifrado en la memoria del altavoz es un MP3 y no otro formato es una suposición razonable. Podría ser también WAV, OGG, o algún otro formato de audio. En este caso son MP3. Si no lo hubiesen sido, un adversario tardaría un poco pero no demasiado más: la idea es la misma.

Vamos a ver los primeros 512 bytes (32 bloques de 16 bytes) del archivo para conocer el formato que tiene un archivo MP3 de audio:

In [3]:
```
HexViewer('test.mp3', bs=16, count=32).print_blocks()
```

```
49 44 33 03 00 00 00 00 00 66 54 43 4F 4E 00 00 - ID3......fTCON..
00 0A 00 00 00 43 69 6E 65 6D 61 74 69 63 54 41 - .....CinematicTA
4C 42 00 00 00 16 00 00 00 59 6F 75 54 75 62 65 - LB.......YouTube
20 41 75 64 69 6F 20 4C 69 62 72 61 72 79 54 49 -  Audio LibraryTI
54 32 00 00 00 10 00 00 00 49 6D 70 61 63 74 20 - T2.......Impact 
4D 6F 64 65 72 61 74 6F 54 50 45 31 00 00 00 0E - ModeratoTPE1....
00 00 00 4B 65 76 69 6E 20 4D 61 63 4C 65 6F 64 - ...Kevin MacLeod
FF FB E0 04 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 49 6E 66 6F 00 00 00 0F 00 00 07 E6 - ....Info........
00 20 41 23 00 03 06 08 0B 0D 10 12 15 17 1A 1C - . A#............
1F 21 24 26 29 2B 2E 30 33 35 38 3A 3D 40 42 45 - .!$&)+.0358:=@BE
47 4A 4C 50 52 55 57 5A 5C 5F 61 64 66 69 6B 6E - GJLPRUWZ\_adfikn
70 73 75 78 7A 7D 80 82 85 87 8A 8C 8F 91 94 96 - psuxz}..........
99 9B 9F A1 A4 A6 A9 AB AE B0 B3 B5 B8 BA BD C0 - ................
C2 C5 C7 CA CC CF D1 D4 D6 D9 DB DE E0 E3 E5 E8 - ................
EA EE F0 F3 F5 F8 FA FD 00 00 00 39 4C 41 4D 45 - ...........9LAME
33 2E 39 39 72 01 CD 00 00 00 00 00 00 00 00 34 - 3.99r..........4
FF 24 03 C2 45 00 01 40 00 20 41 23 A4 03 C6 54 - .$..E..@. A#...T
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - ................
```

Observa:

- El inicio de un archivo es conocido: `ID3` . Es muy común que los archivos empiecen con unas pocos letras que los identifican: DOCX, JPG, ZIP... tienen todos una cabera inicial que los identifica.
- Los archivos MP3 como este en ocasiones tienen "secciones de padding", que son secciones con muchos ceros

Veamos ahora el archivo `01.enc` . Este archivo está cifrado utilizando el mismo método que los audiocuentos que se pueden encontrar en el quiosco.

Observa:

- El archivo no empieza con lo que se espera de un MP3
- En las partes que un MP3 tendría ceros, aquí aparece otra cadena

Vamos a suponer que este archivo es un MP3 cifrado. Entonces, **sabemos** que tiene que empezar con la cadena "ID3" así que tenemos que encontrar una clave que descifre el inicio a la cadeba "ID3". Además, sabemos que es muy posible que partes del archivo descifrado contengan largas secciones con "ceros". Esto de conocer al menos partes de un mensaje y aprovechar el conocimiento para descifrar partes del mismo se llama "ataque de texto en claro conocido" ([https://en.wikipedia.org/wiki/Known-plaintext_attack (https://en.wikipedia.org/wiki/Known-plaintext_attack)](https://en.wikipedia.org/wiki/Known-plaintext_attack)) y se usa en algunos ataques a sistemas criptográficos

In [4]:
```
HexViewer('01.enc', bs=16, count=32).print_blocks()
```

```
3A 21 50 71 65 74 73 65 63 14 31 37 3C 2B 63 72 - :!Pqetsec.17<+cr
65 7E 73 65 63 31 0C 1A 16 08 02 06 0C 17 27 24 - e~sec1........'$
2F 30 65 74 73 73 63 72 65 2D 1C 10 37 07 07 11 - /0etsscre-..7...
53 24 16 16 0C 1B 53 29 0A 10 17 15 01 1C 37 3B - S$....S)......7;
31 46 73 65 63 62 65 74 73 2C 0E 02 04 17 07 45 - 1Fsecbets,.....E
2E 1D 01 11 01 04 17 1D 31 24 36 54 63 72 65 7A - ........1$6Tcrez
73 65 63 39 00 02 1A 0B 43 3F 04 17 3F 00 0C 16 - sec9....C?..?...
9A 8F 93 61 63 72 65 74 73 65 63 72 65 74 73 65 - ...acretsecretse
63 72 65 74 73 65 63 72 65 74 73 65 63 72 65 74 - cretsecretsecret
73 65 63 72 2C 1A 15 0A 63 72 65 7B 73 65 64 94 - secr,...cre{sed.
65 54 32 46 63 71 63 7C 78 68 73 60 70 63 69 79 - eT2Fcqc|xhs`pciy
7C 53 41 52 5A 4E 4D 42 56 41 4B 5F 5E 32 27 31 - |SARZNMBVAK_^2'1
34 2F 2F 22 37 21 24 3F 3F 2D 04 10 15 0C 08 1C - 4//"7!$??-......
15 07 06 1D 19 0F E5 F6 F6 E2 E9 FE EA E5 E7 F3 - ................
FA E9 FA D5 D7 C3 CA D9 CB C4 C0 D0 DB C8 D8 B4 - ................
B1 A0 A4 B8 A9 BB A2 B1 B5 AB BE AA 93 86 86 9A - ................
8F 9A 83 96 96 8A 9F 89 73 65 63 4B 29 35 3E 20 - ........secK)5> 
50 5C 5C 4D 01 64 AE 72 65 74 73 65 63 72 65 40 - P\\M.d.retsecre@
8C 41 60 B0 20 74 72 25 63 52 24 57 D7 66 A5 26 - .A`. tr%cR$W.f.&
65 74 73 65 63 72 65 74 73 65 63 72 65 74 73 65 - etsecretsecretse
63 72 65 74 73 65 63 72 65 74 73 65 63 72 65 74 - cretsecretsecret
73 65 63 72 65 74 73 65 63 72 65 74 73 65 63 72 - secretsecretsecr
65 74 73 65 63 72 65 74 73 65 63 72 65 74 73 65 - etsecretsecretse
63 72 65 74 73 65 63 72 65 74 73 65 63 72 65 74 - cretsecretsecret
73 65 63 72 65 74 73 65 63 72 65 74 73 65 63 72 - secretsecretsecr
65 74 73 65 63 72 65 74 73 65 63 72 65 74 73 65 - etsecretsecretse
63 72 65 74 73 65 63 72 65 74 73 65 63 72 65 74 - cretsecretsecret
73 65 63 72 65 74 73 65 63 72 65 74 73 65 63 72 - secretsecretsecr
65 74 73 65 63 72 65 74 73 65 63 72 65 74 73 65 - etsecretsecretse
63 72 65 74 73 65 63 72 65 74 73 65 63 72 65 74 - cretsecretsecret
73 65 63 72 65 74 73 65 63 72 65 74 73 65 63 72 - secretsecretsecr
65 74 73 65 63 72 65 74 73 65 63 72 65 74 73 65 - etsecretsecretse
```

Pregunta:

- ¿Sabrías cómo se ha cifrado este archivo?
- ¿Puedes crear un código para descifrarlo?

Pistas:

- XOR rotativo:https://en.wikipedia.org/wiki/XOR_cipher (https://en.wikipedia.org/wiki/XOR_cipher)
- https://www.mikrocontroller.net/topic/503014 (https://www.mikrocontroller.net/topic/503014)
- Recomendación: utiliza la librería de Python PyCryptodome, porque la usaremos en el resto del curso: https://pycryptodome.readthedocs.io/en/latest/ (https://pycryptodome.readthedocs.io/en/latest/)

Desde línea de comandos podrías intentar algo así, pero la idea es que escribas tu propio código:

```
# pip install xortool ; hash xortool-xor
file test.mp3
hexdump -C -n 32 test.mp3
hexdump -C 01.enc | less
cat 01.enc |  xortool-xor -r "secret" -f - > 01.mp3
```

In [ ]: