Trabajo práctico 2: Cifradores de bloque

El trabajo práctico consiste en resolver los siguientes ejercicios de la categoría Block Ciphers. Por cada uno deberán entregar la flag encontrada y una breve explicación de cómo la encontraron. Imaginen que se lo explican a alguien que nunca lo resolvió y tiene que poder resolverlo con sólo su explicación. No hace falta entregar el código.

How AES works

De la sección AES hacer todos:

1. Keyed Permutations

La función biyectiva es el término matemático para la correspondencia uno a uno.
FLAG: crypto{bijection}

2. Resisting Brute Force

El mejor ataque contra AES es el Biclique Attack, aunque es solo levemente mejor al ataque por fuerza bruta.
FLAG: crypto{biclique}

3. Structure of AES

Para obtener el flag recorrimos la matriz de valores ASCII brindada en el enunciado y utilizando la función **chr()** convertimos los mismos a caracteres. Estos fueron concatenados, lo que dió como resultado la siguiente flag.
FLAG: crypto{inmatrix}

4. Round Keys

Para obtener el flag recorrimos ambas matrices elemento a elemento y aplicando xor entre los mismos, a este ascii resultando luego de aplicar el xor, mediante nuevamente la función **chr()** convertimos el ascii a carácter y lo concatenamos a una cadena final, la cual genero el siguiente flag.
FLAG: crypto{r0undk3y}

5. Confusion through Substitution

Para obtener esta flag recorrimos elemento a elemento de la matriz **state** y por cada elemento fuimos obteniendo su valor hexadecimal.
Para ello, utilizamos **[*hex(elem)]** que nos devuelve un array con los caracteres del hexa. Por ejemplo, si nuestro elemento de la matriz **state** era un 32, obtendremos este array de caracteres:
[‘0’, ‘x’, ‘2’, ‘0’]

Con esto, nosotros sabemos que el ‘2‘ representa la fila dentro de la SBox y que el ‘0’ representa la columna dentro del SBox.
Ya con esta información, podemos acceder al SBox y utilizando estas coordenadas reemplazamos la matriz de estado por el valor que se encuentra dentro del SBox en las coordenadas dadas.
Luego de correr todos los elementos de nuestra matriz **state**, aplicamos la función **matrix2bytes** y obtendremos la flag.

FLAG: crypto{l1n34rly}

6. Diffusion through Permutation

Para obtener esta flag lo que hicimos fue utilizar la función **shift_rows** y en base a ella construir la función **inv_shift_rows**. Lo que hicimos fue básicamente invertir los roles, ahora los elementos eran sustituidos en **shift_rows** ahora iban a reemplazar a los elementos que antes eran los que reemplazaban.

```
def shift_rows(s):
    s[0][1], s[1][1], s[2][1], s[3][1] = s[1][1], s[2][1], s[3][1], s[0][1]
    s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
    s[0][3], s[1][3], s[2][3], s[3][3] = s[3][3], s[0][3], s[1][3], s[2][3]


def inv_shift_rows(s):
    s[1][1], s[2][1], s[3][1], s[0][1] = s[0][1], s[1][1], s[2][1], s[3][1]
    s[2][2], s[3][2], s[0][2], s[1][2] = s[0][2], s[1][2], s[2][2], s[3][2]
    s[3][3], s[0][3], s[1][3], s[2][3] = s[0][3], s[1][3], s[2][3], s[3][3]
```

Lo que al principio se encontraba a la izquierda, ahora se encuentra a la derecha.
Lo que al principio se encontraba a la derecha, ahora se encuentra a la izquierda.

FLAG: crypto{d1ffUs3R}

7. Bringing It All Together

Para encontrar esta flag utilizamos todas las funciones que venimos creando en este Trabajo Práctico y ponerlas en práctica para desencriptar un texto cifrado con AES 128.
El procedimiento fue el siguiente:

1) Expandimos la clave para obtener 11 claves distintas y así poder realizar las rondas correspondientes.
2) Convertimos el texto cifrado en una matriz de bytes
3) Aplicamos la última **round key** a la matriz de bytes
4) Ejecutamos las 10 rondas
    a) Aplicamos difusión en las filas **(inv_shift_rows)**
    b) Aplicamos confusión en la matriz de bytes **(sub_bytes)**
    c) Aplicamos la clave correspondiente a su ronda **(round_key)**
    d) Aplicamos difusión en las columnas **(inv_mix_columns)**
5) Ejecutamos la ronda final
    a) Aplicamos difusión en las filas **(inv_shift_rows)**
    b) Aplicamos confusión en la matriz de bytes **(sub_bytes)**
    c) Aplicamos la clave correspondiente a su ronda **(round_key)**

Luego, a lo que nos quedó de esa matriz le aplicamos la función **matrix2bytes** e imprimimos los bytes por consola para obtener la flag.
FLAG: crypto{MYAES128}

Symmetric Starter (opcional)

La sección Symmetric Starter plantea un escenario de ataque más realista y más interesante pero también más complicado. Lo dejamos como opcional, no es requisito para aprobar el TP, ¡pero da muchos puntos!

8. Modes of Operation Starter
A partir del texto cifrado que brinda cryptohack con su Endpoint, llamamos al otro endpoint que expone permitiendo descifrar el código. El mismo es devuelto en hexadecimal, por lo que lo pasamos a bytes y encontramos el flag.
FLAG: crypto{bl0ck_c1ph3r5_4r3_f457_!}
9. Passwords as Keys
Primero que nada, nos descargamos el archivo que contenía todas las posibles **passwords**
Para encontrar la flag, primero obtuvimos el texto cifrado utilizando un **get request** hacia el endpoint que ofrece Cripto Hack.
Luego, leímos del archivo todas las posibles **passwords** y las almacenamos en una variable global (vector de passwords).
Ahora, iteramos cada una de las posibles **passwords** y en cada una de esas iteraciones fuimos haciendo los siguientes pasos:

1. Removemos los espacios en blanco de esa posible **password**
2. Obtenemos el HASH de esa posible **password**
3. Convertimos ese HASH en hexadecimal (para que la función **decrypt** de Crypto Hack pueda leerla correctamente)
4. Llamamos a la función **decrypt** (que nos provee Crypto Hack) y obtenemos la posible flag en hexadecimal
5. Convertimos esa posible flag de hexadecimal a una cadena de caracteres legible (UTF-8)
6. Finalmente nos preguntamos, si esa flag en UTF-8 empieza con la palabra **crypto** la mostramos por pantalla. De lo contrario, seguirá iterando.

FLAG: crypto{k3y5__r__n07__p455w0rdz?}