CandyCustom Writeup

Comenzando con el reto, se nos proporciona un archivo llamado **candy.py** y el archivo **out.txt** que contiene información adicional del proceso.

Descripción del reto:
Analiza la receta del dulce.

candy.py

El archivo **candy.py** implementa varias funciones criptográficas, combinando **generación de claves, cifrado afín y XOR dinámico**.

```python
from random import randint
import sys

def generator(g, x, p):
    return pow(g, x) % p

def encrypt(plaintext, key):
    cipher = []
    for char in plaintext:
        cipher.append(((ord(char) * key * 311)))
    return cipher

def is_prime(p):
    v = 0
    for i in range(2, p + 1):
        if p % i == 0:
            v = v + 1
    if v > 1:
        return False
    else:
        return True

def dynamic_xor_encrypt(plaintext, text_key):
    cipher_text = ""
    key_length = len(text_key)
    for i, char in enumerate(plaintext[::-1]):
        key_char = text_key[i % key_length]
        encrypted_char = chr(ord(char) ^ ord(key_char))
        cipher_text += encrypted_char
    return cipher_text

def test(plain_text, text_key):
    p = 97
    g = 31
    if not is_prime(p) and not is_prime(g):
        print("Enter prime numbers")
        return
    a = randint(p - 10, p)
    b = randint(g - 10, g)
    print(f"a = {a}")
    print(f"b = {b}")
    u = generator(g, a, p)
    v = generator(g, b, p)
    key = generator(v, a, p)
    b_key = generator(u, b, p)
    shared_key = None
    if key == b_key:
        shared_key = key
    else:
        print("Invalid key")
        return
    semi_cipher = dynamic_xor_encrypt(plain_text, text_key)
    cipher = encrypt(semi_cipher, shared_key)
    print(f'cipher is: {cipher}')

if __name__ == "__main__":
    message = sys.argv[1]
    test(message, "trudeau")
```

Out.txt

El archivo **out.txt** nos proporciona algunos valores usados durante la generación de la clave y el cifrado:

El script cifra un mensaje en dos fases:

1. **Fase XOR Dinámico**: El texto se invierte y se cifra utilizando una **operación XOR** basada en la clave "trudeau".
2. **Cifrado Afín**: El resultado del XOR se multiplica por 311 * shared_key para producir los valores del cifrado final.

Para resolver este reto, realicé los siguientes pasos:

1. **Reconstruí la clave compartida** usando los valores de a = 89 y b = 28 con los mismos parámetros p = 97 y g = 31.
2. **Desencripte los valores del cifrado** dividiendo cada número por el valor 311 * shared_key para obtener los caracteres originales.
3. **Aplicación inversa del XOR** para recuperar el texto claro.

```python
def generator(g, x, p):
    return pow(g, x) % p

def decrypt(cipher, shared_key):
    # Dividir cada valor por 311 * shared_key para obtener los valores ASCII originales
    decrypted_chars = [int(c // (shared_key * 311)) for c in cipher]
    return "".join(chr(char) for char in decrypted_chars)

def dynamic_xor_decrypt(cipher_text, text_key):
    decrypted_text = [""] * len(cipher_text)
    key_length = len(text_key)

    # Aplicar XOR dinámico inverso
    for i, char in enumerate(cipher_text):
        key_char = text_key[i % key_length]
        decrypted_text[len(cipher_text) - 1 - i] = chr(ord(char) ^ ord(key_char))

    return "".join(decrypted_text)

# Valores dados del archivo out.txt
a = 89
b = 28
p = 97
g = 31

# Reconstrucción de la clave compartida
u = generator(g, a, p)
v = generator(g, b, p)
shared_key = generator(v, a, p)

# Cifrado proporcionado en out.txt
cipher = [
    16794, 20526, 11196, 9330, 0, 115692, 156744, 0, 123156, 110094,
    42918, 156744, 115692, 48516, 0, 3732, 22392, 41052, 70908, 48516,
    29856, 0, 20526, 102630, 54114, 24258, 93300
]

# Paso 1: Desencriptar usando la clave compartida
semi_decrypted = decrypt(cipher, shared_key)

# Paso 2: Aplicar XOR dinámico inverso para obtener el texto original
flag = dynamic_xor_decrypt(semi_decrypted, "trudeau")

print(f"La bandera es: {flag}")
```

Perfecto!! Ahora tenemos nuestra flag.

```
ShyByte{Crypto_1sN0t!_easy}
```