S-Box

This is a cryptography challenge that requires you to write some code to decrypt a cipher. You are given the cipher plus the code and key used to encrypt it.

The challenge:

> *To get this flag you'll have to decrypt the encrypted data below. The script used to encrypt the data is attached. You also have the key that was used. You'll have to write some code to reverse the encryption and recover the plaintext flag. I suppose it could also be done by hand, but I would not recommend it.*
>
> *The encryption uses a substitution cipher - a simplified version of the encryption described in this Computerphile video (with substitution boxes only, no permutation): https://www.youtube.com/watch?v=DLjzI5dX8jc*

This is the provided cipher:

```
01111001101110010111101101110001011000110011000001111011001011010101111111011011100101011111101111011001001001101111110111100001000100111111110110100000110100111001000110010000101000001000000010101000100001101110110011011010111010011001111011001101100000011100011011100101011111
```

This is the provided key:

```
1K39ko7f8Pf3vADHA2qc7aPHHamwKyTt0B6UV
```

This is the provided crypto code:

```
#!/usr/bin/env python3
# very weak encryption script version 1.0
import sys

sbox_values = [11, 6, 8, 15, 9, 13, 14, 0, 7, 3, 12, 4, 10, 2, 1, 5]

def str_to_bin(plaintext=''):
    # converts string to list of ascii binaries
    return [bin(ord(x))[2:].zfill(8) for x in plaintext]

def bin_to_str(bits=None):
    # converts list of ascii binaries to string
    return ''.join([chr(int(x, 2)) for x in bits])

def bin_to_dec(bits):
    # converts 4 bit binary to decimal
    dec = 0
    if bits[0] == '1': dec = 8
    if bits[1] == '1': dec += 4
    if bits[2] == '1': dec += 2
    if bits[3] == '1': dec += 1
    return dec

def dec_to_bin(dec):
    # converts decimal (0-15) to 4 bit binary
    bits = list('0000')
    if dec - 8 >= 0:
        dec -= 8
        bits[0] = '1'
    if dec - 4 >= 0:
        dec -= 4
        bits[1] = '1'
    if dec - 2 >= 0:
        dec -= 2
        bits[2] = '1'
    if dec - 1 >= 0:
        bits[3] = '1'
    return ''.join(bits)

# function can be used for decrpytion
# def sbox_reverse(s):
#     # reverses sbox substituion
#     for i in sbox_values:
#         if s == i:
#             return sbox_values.index(i)

def encrypt(plaintext):
    encrypted_char_list = []  # list to hold encrypted characters
    bin_ascii_chars_list = str_to_bin(plaintext)  # convert plaintext message to
list of binary ascii values

    bin_ascii_key_list = str_to_bin(key)  # convert key to list of binary values
    xored_bin_ascii_list = []  # list to hold xored values
```

```
    for i in range(len(bin_ascii_chars_list)):
        # xor key characters with plaintext characters bit by bit
        xored_char = ''
        for j in range(0,8):  # ascii binary value is bits, go through one by one
            xored_char += (str(int(bin_ascii_chars_list[i][j]) ^
int(bin_ascii_key_list[i][j])))  # xor each bit, change back to string
        xored_bin_ascii_list.append(xored_char)  # make new list of xored binary
char values

    for bin_ascii in xored_bin_ascii_list:  # for each xored binary character value
        bits_list = []  # list to hold 2 4 bit pairs from each binary ascii value -
ex: A = 0100 and 0001
        encrypted_bin_list = []  # list to hold encrypted ascii binary value
        bits_list.append(bin_ascii[:4]), bits_list.append(bin_ascii[4:])  # split 8
bit binary to 2 4 bit binaries
        for bits in bits_list:  # for each 4 bit binary
            dec = bin_to_dec(bits)  # convert binary to decimal
            for _ in range(4):  # substitube decimal value 4 times
                dec = sbox_values[dec]
            encrypted_bin_list.append(dec_to_bin(dec))  # convert substituted
decimal back to 4 bit binary, and combine into list of 8 bit values
        encrypted_char_list.append(''.join(encrypted_bin_list))  # build list of
encrypted ascii binary values
    cipher = ''.join(encrypted_char_list)  # create encrypted string
    return cipher

plaintext = input('enter some plaintext: ')  # get user input - plaintext
key = input('enter a key: ')  # get user input - key
if (len(plaintext) != len(key)):
    # if key and plaintext are not the same length, exit
    print('\nERROR: plaintext and key should be the same length!')
    sys.exit(0)
cipher = encrypt(plaintext)  # run encrypt function
print('cipher:', cipher)  # print cipher results
```

And this is my decryption code:

```
#!/usr/bin/env python3
import binascii

sbox_values = [11, 6, 8, 15, 9, 13, 14, 0, 7, 3, 12, 4, 10, 2, 1, 5]

def str_to_bin(plaintext=''):
    # converts string to list of ascii binaries
    return [bin(ord(x))[2:].zfill(8) for x in plaintext]

def bin_to_str(bits=None):
    # converts list of ascii binaries to string
    return ''.join([chr(int(x, 2)) for x in bits])

def bin_to_dec(bits):
    # converts 4 bit binary to decimal
    dec = 0
    if bits[0] == '1': dec = 8
    if bits[1] == '1': dec += 4
    if bits[2] == '1': dec += 2
    if bits[3] == '1': dec += 1
    return dec

def dec_to_bin(dec):
    # converts decimal (0-15) to 4 bit binary
    bits = list('0000')
    if dec - 8 >= 0:
        dec -= 8
        bits[0] = '1'
    if dec - 4 >= 0:
        dec -= 4
        bits[1] = '1'
    if dec - 2 >= 0:
        dec -= 2
        bits[2] = '1'
    if dec - 1 >= 0:
        bits[3] = '1'
    return ''.join(bits)

def sbox_reverse(s):
    # reverses sbox substituion
    for i in sbox_values:
        if s == i:
            return sbox_values.index(i)

def decrypt(cipher):
    # TODO write decrpytion function
    plaintext = ''
    plaintext_char_list = []
    bin_chars = [cipher[i:i+8] for i in range(0, len(cipher), 8)]
    unsboxed = []
    bin_ascii_key = str_to_bin(key)

    for b in bin_chars:
```

```
        bits = []
        bin_char = []
        bits.append(b[:4]), bits.append(b[4:])
        for bit in bits:
            d = bin_to_dec(bit)
            for _ in range(4):
                d = sbox_reverse(d)
            s = d
            bin_char.append(dec_to_bin(s))
        bin_char = ''.join(bin_char)
        unsboxed.append(bin_char)

    xored_char = ''
    for i in range(len(unsboxed)):
        xored_char = ''
        for j in range(0,8):
            xored_char += (str(int(unsboxed[i][j]) ^ int(bin_ascii_key[i][j])))
        plaintext_char_list.append(xored_char)
    plaintext = bin_to_str(plaintext_char_list)
    return plaintext
cipher = input('enter a cipher:')
key = input('enter a key:')
plaintext = decrypt(cipher)
print('plaintext:', plaintext)
```

Running this code and entering the ciphertext and key will give you the flag:
flag{be_sure_2_drink_your_Ovaltine}