Initial observations: a padding is used, with value 0xf3. The challenge was released in a contest named TFCCTF. The encrypted flag has 48 bytes, so we have 3 blocks of 16 chars each. The final block contains an unknown number of padding bytes. The key is unknown and randomized. The key is used cyclically.

Solving this challenge requires 2 key assumptions:

1) If the i-th char of the 3rd block is a padding bytes, then the i-th chars of the 1st and 2nd block are printable. If this proves true, then we found 2 characters of the flag. If this proves false, we have uncovered all of the padding.

2) Given this was a chall in TFCCTF, it's safe to assume the 1st 6 chars of the flag are TFCCTF. As a result, we know 18 characters of the flag. We could assume TFCCTF{, but we find out "{" from the 1st assumption.

To solve, we make a program that solves the 1st and 2nd assumptions, to print as much of the flag as we can.

```
import string

def recover_flag_with_assumption(ciphertext, block_size=16, pad_guess=0x3f):
    printable_chars = set(string.printable)
    flag = ['*'] * len(ciphertext)
    recovered_key = [None] * block_size
    for i in range(block_size - 1, -1, -1):
        current_index = 2 * block_size + i
        recovered_key[i] = ciphertext[current_index] ^ pad_guess
        valid = True
        for block_idx in range(2):
            prev_index = block_idx * block_size + i
            prev_char = ciphertext[prev_index] ^ recovered_key[i]
            if chr(prev_char) not in printable_chars:
                valid = False
                break
        if valid:
            flag[current_index] = '?'
            for block_idx in range(2):
                prev_index = block_idx * block_size + i
                prev_char = ciphertext[prev_index] ^ recovered_key[i]
                if chr(prev_char) in printable_chars:
                    flag[prev_index] = chr(prev_char)
        else:
            break
    return ''.join(flag)

def complete_flag_with_known_prefix(ciphertext, block_size=16,
known_prefix="TFCCTF"):
    flag = ['*'] * len(ciphertext)
    for i, char in enumerate(known_prefix):
        flag[i] = char
    for i in range(6):
        index_2 = block_size + i
        flag[index_2] = chr(ord(flag[i]) ^ ciphertext[index_2] ^ ciphertext[i])
        index_3 = 2 * block_size + i
        flag[index_3] = chr(ord(flag[i]) ^ ciphertext[index_3] ^ ciphertext[i])
    return ''.join(flag)

with open('output.txt', 'r') as f:
    ciphertext = bytes.fromhex(f.read().strip())

p1_flag = recover_flag_with_assumption(ciphertext)
p2_flag = complete_flag_with_known_prefix(ciphertext)

print(f"Part 1 of flag: {p1_flag}")
print(f"Part 2 of flag: {p2_flag}")
final_flag = ''.join([c1 if c1 != '*' else c2 for c1, c2 in zip(p1_flag,
p2_flag)]).strip("?")
print(f"Final flag: {final_flag}")
```
This program turns out to print the whole flag. Lucky us!

Made with love by: AndreiCat