# Ciphertext CTF 2020

## Cryptography & Steganography

### Keyless_cipher

**Description:**

we used this custom cipher to encrypt important data... when we deleted the actual data, we realized that we don't have decryption function... can you find a way to decrypt our data?

**Files:**

| | | |
|---|---|---|
| CT_part1 | Size: 23.76 KB | MD5: f980aefb18665c8d6a86683d01930a1b |
| CT_part2 | Size: 24.05 KB | MD5: 6fe1e1800da9cc338e123347a28bc25e |
| main | Size: 2.22 KB | MD5: ce55337829cf5043a9dc13027c3fb2c5 |

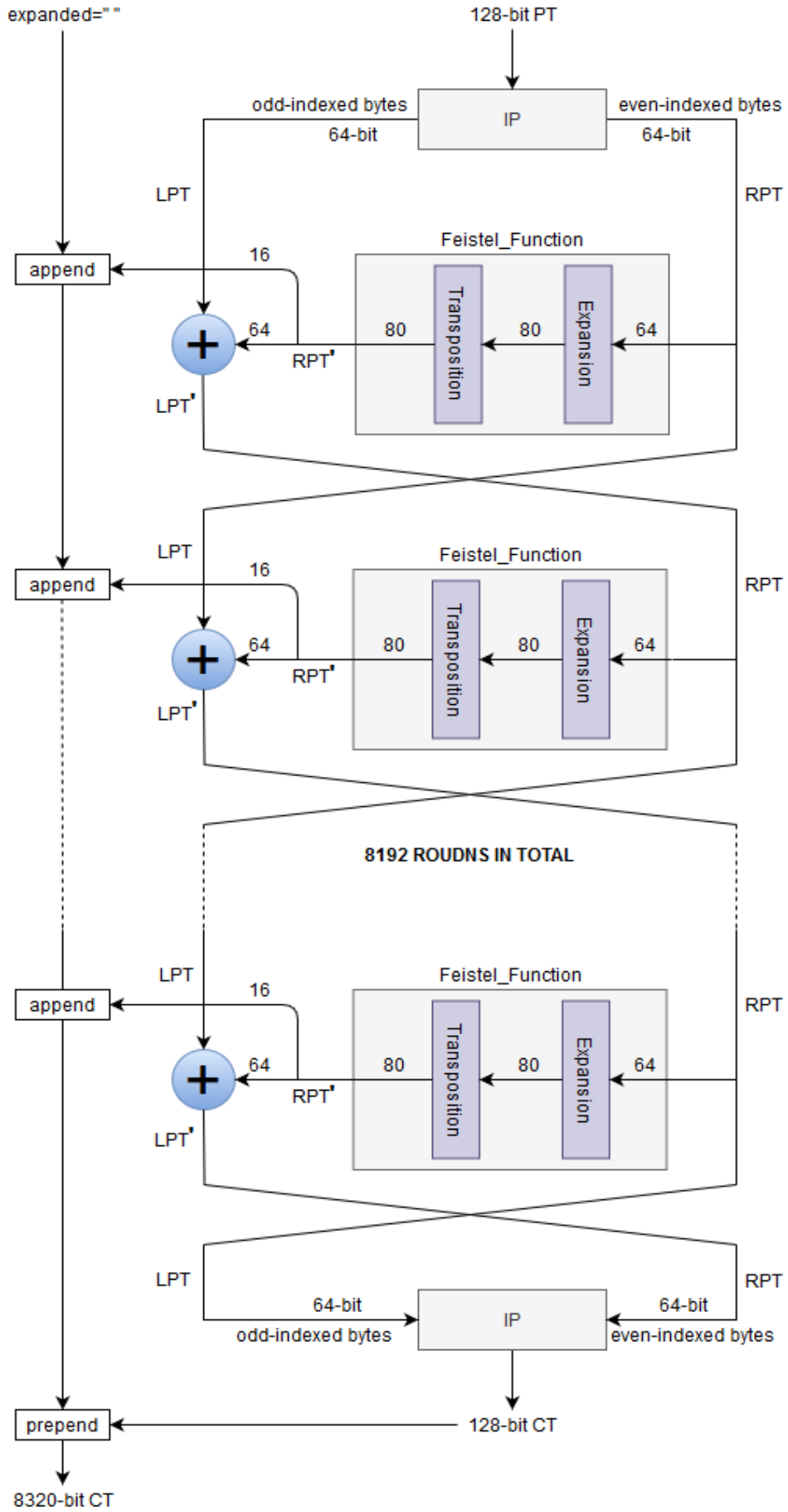
**Solution:**

As always, first what we do is to understand the encryption process, lets take a look at the code:

```
import sys
def progressbar(it, prefix="", size=60, file=sys.stdout):
    count = len(it)
    def show(j):
        x = int(size*j/count)
        file.write("%s[%s%s] %i/%i\r" % (prefix, "#"*x, "."*(size-x), j, count))
        file.flush()
    show(0)
    for i, item in enumerate(it):
        yield item
        show(i+1)
    file.write("\n")
    file.flush()

def IP(BYTES):
    LEFT, RIGHT = '', ''
    for i, j in enumerate(BYTES):
        if i % 2:
            LEFT += j
        else:
            RIGHT += j
    return LEFT, RIGHT

def Feistel_Function(_64BIT):
    def expansion(X):
        X = ''.join(f'{ord(x):08b}' for x in X)
        EX = ''
        X = [X[0+i:8+i] for i in range(0, len(X), 8)]
        i = 0
        while i < len(X):
            EX += (X[i-1][-1]+X[i]+X[i][0])
            i += 1
        return EX
```

```
    def transposition(EX):
        TEX = ''
        for i in [37, 27, 58, 14, 31, 39, 65, 36, 48, 22, 28, 18, 53, 21, 8, 77, 17, 20, 9, 7
, 74, 40, 51, 60, 52, 2, 69, 11, 59, 41, 26, 70, 79, 50, 33, 46, 38, 34, 13, 15, 43, 44, 30,
63, 24, 66, 75, 35, 54, 47, 16, 61, 32, 25, 56, 68, 6, 71, 12, 64, 72, 62, 45, 78, 5, 23, 1,
42, 10, 55, 0, 29, 73, 3, 4, 67, 49, 57, 76, 19]:
            TEX += EX[i]
        TEX = ''.join([chr(int(i, 2)) for i in [TEX[0+i:8+i]
                        for i in range(0, len(TEX), 8)]])
        return TEX

    _80BIT = transposition(expansion(_64BIT))
    return _80BIT

def xor(x, y):
    i = 0
    z = ''
    while i < len(x) and i < len(y):
        z += chr(ord(x[i]) ^ ord(y[i]))
        i += 1
    return z

def FP(LEFT, RIGHT):
    RESULT = ''
    i = 0
    while i < len(LEFT):
        RESULT += RIGHT[i]+LEFT[i]
        i += 1
    return RESULT

def encryption(PT):
    LPT, RPT = IP(PT)
    expanded = ''
    for _ in progressbar(range(8192), "Encryption: ", 50):
        RPT_prime = Feistel_Function(RPT)
        expanded += RPT_prime[8:]
        RPT = RPT[:8]
        LPT = xor(LPT, RPT_prime)
        LPT, RPT = RPT, LPT
    CT = FP(LPT, RPT)+expanded
    print(len(CT))
    print('[+] Encryption Completed!')
    return CT

open('CT_part1', 'w').write(encryption(open('flag.txt').read()[:16].strip()))
open('CT_part2', 'w').write(encryption(open('flag.txt').read()[16:].strip()))
```

Looks scary huh?... it is not, the only part that is mainly interesting for us is the $encryption()$ function, we can see that this is a kind of Feistel networks, lets draw a diagram to visualize it and get a better understanding:




Now let's take some notes:

1. We can get rid of the expansion

2. The internal implementation of the functions used in the encryption function don't need to be reversed, we will use them in reverse order
3. $LPT' = LPT \oplus RPT'[:8]$ where $RPT' = Feistel\_Function(RPT)$
4. We know the value of $LPT'$ and $RPT$, hence we can calculate $RPT'$ then calculate the value of $LPT$

**Steps to decrypt:**

1. Take the first 128-bit of the cipher text and discard the rest
2. Perform the initial permutation function (IP) to get the latest LCT and RCT
3. Iterate for 8192 times, each time do:
   1. Swap(RCT,LCT) first because it was performed the last in the encryption
   2. LCT = xor(LCT, Feistel_Function(RCT)[:8]) according to our notes 3 and 4.
4. Perform the final permutation function (FP) to reconstruct the PT. that's it.

Just add the decryption function to the given code and call it with the cipher text as argument:

```
def decryption(CT8320):
    CT128 = CT8320[:128]
    LCT, RCT = IP(CT128)
    for _ in progressbar(range(8192), "Decryption: ", 50):
        LCT, RCT = RCT, LCT
        LCT = xor(LCT, Feistel_Function(RCT)[:8])
    PT128 = FP(LCT, RCT)
    print('[+] Decryption Completed!')
    return PT128


print(decryption(open('CT_part1','rb').read().decode().strip()))
print(decryption(open('CT_part2','rb').read().decode().strip()))
```

output:

```
Decryption: [##################################################] 8192/8192

[+] Decryption Completed!

CTCTF{r3ally_cr@

Decryption: [##################################################] 8192/8192

[+] Decryption Completed!

ppy_c1ph3r_1234}
```

The flag is: CTCTF{r3ally_cr@ppy_c1ph3r_1234}.