BUUCTF 每日打卡 2021-8-1

原创

Σ2333! 于 2021-08-01 21:16:57 发布 179 收藏 1

分类专栏： crypto 文章标签： 加密解密

本文链接：https://blog.csdn.net/weixin_52446095/article/details/119303337

引言

无

[2021i春秋云上巅峰赛]MedicIlmage

题面：

加密代码如下：

```
from PIL import Image
from decimal import *
import numpy as np
import random
getcontext().prec = 20

def f1(x):
    # It is based on logistic map in chaotic systems
    # The parameter r takes the largest legal value
    assert(x>=0)
    assert(x<=1)
```

```
    ...

def f2(x):
    # same as f1
    ...
def f3(x):
    # same as f1
    ...

def encryptImage(path):
    im = Image.open(path)
    size = im.size
    pic  = np.array(im)
    im.close()
    r1 = Decimal('0.478706063089473894123')
    r2 = Decimal('0.613494245341234672318')
    r3 = Decimal('0.946365754637812381837')
    w,h = size
    for i in range(200):
        r1 = f1(r1)
        r2 = f2(r2)
        r3 = f3(r3)
    const = 10**14
    for x in range(w):
        for y in range(h):
            x1 = int(round(const*r1))%w
            y1 = int(round(const*r2))%h
            r1 = f1(r1)
            r2 = f2(r2)
            tmp = pic[y,x]
            pic[y,x] = pic[y1,x1]
            pic[y1,x1] = tmp
    p0 = random.randint(100,104)
    c0 = random.randint(200,204)
    config = (p0,c0)
    for x in range(w):
        for y in range(h):
            k = int(round(const*r3))%256
            k = bin(k)[2:].ljust(8,'0')
            k = int(k[p0%8:]+k[:p0%8],2)
            r3 = f3(r3)

            p0 = pic[y,x]
            c0 = k^((k+p0)%256)^c0
            pic[y,x] = c0

    return pic,size,config
def outputImage(path,pic,size):
    im = Image.new('P', size,'white')
    pixels = im.load()
    for i in range(im.size[0]):
        for j in range(im.size[1]):
            pixels[i,j] = (int(pic[j][i]))

    im.save(path)


def decryptImage(pic,size,config):
    .....
```

```
enc_img = 'flag.bmp'
out_im = 'flag_enc.bmp'

pic,size,_ = encryptImage(enc_img)
outputImage(out_im,pic,size)
```

看起来很复杂，而且函数`f1(x)`，`f2(x)`和`f3(x)`是未知的（虽然三个函数是一样的，不妨都记为f(x)）
根据注释内容搜了一下Logistic map（单峰映象）：

单峰映射（**logistic map**）是种二次的多项式映射（递推关系式），是一个由简单非线性方程产生混沌现象的经典范例。这种映射因生物学家Robert May在1976年发表的一篇论文而著名。单峰映射原本是Pierre François Verhulst用在人口学模型上，后来应用在物种受到限制因素之下的数目[1]。数学上可写成

$$x_{n+1} = rx_n(1 - x_n)$$

其中

- x_n是介于0和1之间的数，表示在第n年的物种数目。
- r是正整数，是根据繁殖和饿死率而得出的数。

又搜了一下针对医学影像的加密系统，里面也提到了一个Logistic映射
基本上可以确定f(x)=rx(1-x)，关键是确定这个r
wiki里面也提到了r不同情况下x的变化趋势：

- 0和1之间：不论启始值数值为何，x_n会越来越少，最后趋近于0。
- 1和2之间：不论启始值数值为何，x_n会快速的趋近$\frac{r-1}{r}$。
- 2和3之间：经过几次迭代，x_n也会越来越接近$\frac{r-1}{r}$，但一开始会在这个值左右振动，而收敛速率是线性的。
- 3：x_n仍然会越来越接近$\frac{r-1}{r}$，但收敛速率极为缓慢，不是线性的。
- 3和$1+\sqrt{6}$（约3.45）之间：针对几乎所有的初值，x_n最后会在2个值之间持续的震荡，即x_n最后会是a,b,a,b...的变化，其数值和r有关。
- 3.45和大约3.54之间，针对几乎所有的初值，x_n最后会在4个值之间持续的震荡。
- 约大于3.54：x_n最后会在8个、16个、32个值……之间持续的震荡，至于r何时会令x_n的值由n个到2n个，则和费根鲍姆常数$\delta = 4.669...$有关。
- 约为3.5699：这样的振动消失，整个系统开始在混沌的状态之中。针对几乎所有的初值，都不会出现固定周期的震荡，初值再微小的变化，随着时间都会使结果产生明显的差异，这就是典型混沌的特性。
- 大于3.5699：整个系统在混沌的状态之中。不过，当中有些特定的r值还是使系统变成非混沌，有周期性的结果，这些区间称为“稳定岛”。例如当r约大于3.82，会出现3个值的周期，再大一点出现6个值及12个值的周期。
- 当r从大约3.5699到大约3.8284之间，系统混沌性质发展的现象有时会称为Pomeau–Manneville场景，其特征是周期性的震荡和非周期性的行为会穿插出现。此特征可以应用在半导体元件中[2]。也有其他区域会使系统的周期为5个值，不管任意周期都存在某特定的r，使周期为指定值。
- 大于4：针对几乎所有的初值，系统最后都会超过区间[0,1]并且发散。

注释中说是在混沌系统中，可以排除r小于3的情况
比较常见也是最初研究的是r=4的情况，那姑且认为r=4
再看加密函数`encryptImage(path)`，分成两段看
先看第一段：

```
    im = Image.open(path)
    size = im.size
    pic  = np.array(im)
    im.close()
    r1 = Decimal('0.478706063089473894123')
    r2 = Decimal('0.613494245341234672318')
    r3 = Decimal('0.946365754637812381837')
    w,h = size
    for i in range(200):
        r1 = f1(r1)
        r2 = f2(r2)
        r3 = f3(r3)
    const = 10**14
    for x in range(w):
        for y in range(h):
            x1 = int(round(const*r1))%w
            y1 = int(round(const*r2))%h
            r1 = f1(r1)
            r2 = f2(r2)
            tmp = pic[y,x]
            pic[y,x] = pic[y1,x1]
            pic[y1,x1] = tmp
```

先导入图片，这没什么好说的
然后r1，r2，r3经过200次f(x)后和常量const生成新的坐标，对原来的图片像素进行移位
要解密的话，只要存储原来坐标的列表或者矩阵，然后换回来就行
再看第二段：

```
    p0 = random.randint(100,104)
    c0 = random.randint(200,204)
    config = (p0,c0)
    for x in range(w):
        for y in range(h):
            k = int(round(const*r3))%256
            k = bin(k)[2:].ljust(8,'0')
            k = int(k[p0%8:]+k[:p0%8],2)
            r3 = f3(r3)

            p0 = pic[y,x]
            c0 = k^((k+p0)%256)^c0
            pic[y,x] = c0
```

先随机生成了一个`config = (p0,c0)`，不过幸运的是p0和c0的范围都在`random.randint(100,104)`，遍历config的话也只要25种情况
后面就是对每个像素的加密，也可以分成两段
前一段：

```
k = int(round(const*r3))%256
k = bin(k)[2:].ljust(8,'0')
k = int(k[p0%8:]+k[:p0%8],2)
r3 = f3(r3)
p0 = pic[y,x]
```

就是搞了个k，解密的时候也只要把每一次的k存储起来就行
后一段：

```
p0 = pic[y,x]
c0 = k^((k+p0)%256)^c0
pic[y,x] = c0
```

这里把之前生成的p0和c0给换了，也就是`pic[y,x] = c0 = k ^ ((k+p0)%256) ^ c0`，而这里的`p0`是未加密时该点的像素值，这里的`c0`是加密过的前面一点的像素值
解密的时候只要将加密后的像素值与前一点的像素值还有之前储存的k的值作异或就可以得到`(k+p0)%256`，记为`k_p`
又因为k和p0都是小于等于256的，为了得到p0，也就是原来的像素值，只要

```
k_p -= k
if k_p < 0:
 k_p += 256
```

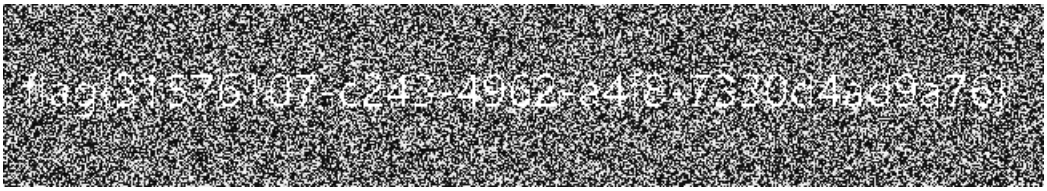
得到的k_p就是原来的像素值
分析完毕，完整的解密代码如下：

```
from PIL import Image
import numpy as np
from decimal import *
getcontext().prec = 20

def f(x):
    # It is based on logistic map in chaotic systems
    # The parameter r takes the largest legal value
    assert (x >= 0)
    assert (x <= 1)
    r = 4
    x = r * x * (1 - x)
    return x

im = Image.open('flag_enc.bmp')
pixels = im.load()
size = im.size
w, h = size
print(size)
# print(np.array(im))
pic = np.zeros((h, w))
const = 10 ** 14
r1 = Decimal('0.478706063089473894123')
r2 = Decimal('0.613494245341234672318')
r3 = Decimal('0.946365754637812381837')
for i in range(im.size[0]):
    for j in range(im.size[1]):
        pic[j][i] = pixels[i, j]
print(pic)

for i in range(200):
    r1 = f(r1)
    r2 = f(r2)
    r3 = f(r3)
p0 = range(100, 105)
c0 = range(200, 205)
config = []
# 列出(p0, c0)的所有情况
for p in p0:
    for c in c0:
        config.append((p, c))

# 记录原先的坐标x1, y1
```

```
list_x1 = np.zeros((h, w))
list_y1 = np.zeros((h, w))
for x in range(w):
    for y in range(h):
        x1 = int(round(const * r1)) % w
        y1 = int(round(const * r2)) % h
        list_x1[y][x] = x1
        list_y1[y][x] = y1
        r1 = f(r1)
        r2 = f(r2)

list_k = []
for conf in config:
    arrk = np.zeros((h, w))
    k = 0
    p0 = conf[0]
    c0 = conf[1]
    for x in range(w):
        for y in range(h):
            k = int(round(const * r3)) % 256
            k = bin(k)[2:].ljust(8, '0')
            k = int(k[p0 % 8:] + k[:p0 % 8], 2)
            r3 = f(r3)
            p0 = int(pic[y, x])
            arrk[y][x] = k
    list_k.append(arrk)
print(list_k)

list_p = []
for conf in config:
    x1 = 0
    y1 = 0
    picture = pic.copy()
    arrk = list_k[config.index(conf)]
    p0 = conf[0]
    c0 = conf[1]
    for x in range(w-1, -1, -1):
        for y in range(h-1, -1, -1):
            k = int(arrk[y][x])
            p0 = pic[y][x]
            if (y-1) >= 0:
                c0 = int(pic[y-1][x])
            elif (y - 1) < 0 and (x-1) < 0:
                pass
            else:
                c0 = int(pic[h-1][x-1])

            k_p = k ^ (int(pic[y][x])) ^ c0
            k_p -= k
            if k_p < 0:
                k_p += 256
            picture[y][x] = k_p

    for x in range(w - 1, -1, -1):
        for y in range(h - 1, -1, -1):
            x1 = int(list_x1[y][x])
            y1 = int(list_y1[y][x])
            tmp = picture[y1][x1]
            picture[y1][x1] = picture[y][x]
```

```
            picture[y][x] = tmp
    list_p.append(picture)
    print(picture)

for k in range(len(list_p)):
    im = Image.new('P', size, 'white')
    pixels = im.load()
    for i in range(im.size[0]):
        for j in range(im.size[1]):
            pixels[i, j] = (int(list_p[k][j][i]))
    im.save('p' + str(k) + '.bmp')
```

最后得到25张图片，除了第一张图片，都能看到flag
挑了一张比较清晰的：



看不清的话可以和别的比对着看

结语

还有一道crtrsa比赛的时候没有做出来，后来在badm0nkey前辈的指导下解出来了，留着明天讲
感谢Phoenix大佬帮我指出错误
希望继续坚持