J-5000 加密算法

设计思路

2022-12-18

FTY

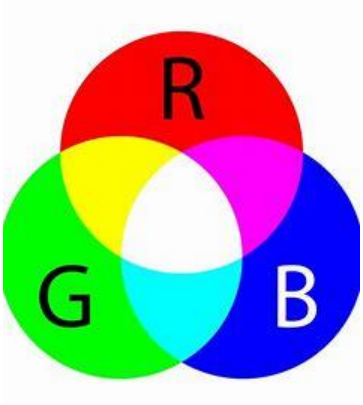
目录

总述

J-5000 算法是 FTY 独立瞎编的轻量级加密算法，以颜色偏差为基础原理打乱原图的颜色，使原图失真，解密时根据密码簿里面的偏差数组依次逐像素逆向偏移，即可得到原图。这种加密方法运用在个人的加密是非常可靠的，在没有密码簿的情况下，暴力破解加密图片一次性成功的概率为 1 / 241^3^5000。这是一个天文数字。所以，在保存好密码簿的情况下，J-5000 算法是日常中加密图片的不错的选择。

算法详解

基础原理



计算机显示颜色大多使用 RGB 形式来储存颜色，一个 RGB 元组的范围是(0-255,0-255,0-255)，也就是说，只要按照一定的差值偏移一个 RGB 元组中的数值，就可以使原图失真。而 J-5000 算法就是通过 5000 个差值数组有规律的逐像素偏移原图的颜色，从而达到加密效果。

细致来讲，一个 J-5000 密码簿中内置了 5000 个差值数组，在加密/解密的时候，不断地循环调用每一个差值数组（第一次用 0 号，第二次用 1 号……第 5001 次用 0 号….）由于数组数量繁多，所以肉眼较难看出图片的原貌。

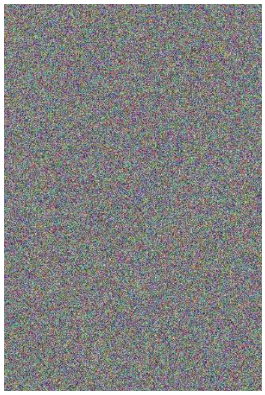
密码簿的生成

J-5000 密码簿的生成非常简单但是却充满随机性，每一个数组均由 python 随机生成三个范围内的整数，循环 5000 次后，将数组存入 npy 文件中，一个 J-

5000 密码簿就制作完成了。

二次加密

二次加密看似很浪费时间，但却是弥补 J-5000 缺陷的重要部分，由于是循环遍历差值数组，所以每隔一些像素，一定会大量出现使用相同差值数组的像素点，若原图中使用相同差值数组的几个像素点颜色相同或差距不大，



那么加密后的图片就依稀能看见物体的外轮廓。（如右图 注：为了使这样的效果更加明显，右图加密时仅采用了 200 组差值数组）但是由于相邻的每一个像素点颜色的差较大，所以只要换一个方向重新加密一次就能彻底打破这个规律。（如左图）。

逆向解密

先定义一个差值数组[-85,52,63]，受到这个数组影响的像素点会完全偏离原本的颜色，这时候将数组中的数字全部乘-1，就可以得到一个新的数组[85,-52,-63]使用这个数组再次影响一次加密后的像素，就可以得到原像素了。解密的程序结构和加密基本相同，只不过在得到差值数组后，将其中的每一个数字都取相反数，得到逆向差值数组，使用这个数组即可解出该像素的原颜色。通过这样的逆向方法遍历完加密图片的所有像素后，即可得到原

图。（如右图，图中左侧为加密后的图片，右侧为解密后的结果）

理论可靠性分析

分析该算法的缺点

我在构思这个算法的时候就已经想到了这个必然存在的问题，由于该算法使用的差值数组只有 5000 个，一个像素点在两次加密中使用的两个差值数组也就有 25 000 000 种可能，倘若有一张像素数量超过 2500 万个（如 5000x5001），那就必然存在有像素点在两次加密中使用的偏差数组是完全相同的。而且只要像素点数量超过了 2500 万这个阈值，那么上述的相同像素点会大量地出现。空口无凭，我进行了一次 5001x5001 尺寸的模拟加密（即不是真正的加密图片，进行这个操作的主要目的是收集加密时在指定像素点运用的偏差数组的数据），结果在意料之内：在 25 010 001 个像素点中，出现了 4 个运用了相同偏差数组的像素点，而通过简单的计算就可以知道，随着图片的尺寸越来越大，这样的像素点会呈指数式增长。也就是说，没有密码簿的情况下，如果破译人员了解图中的大致内容并且精准地找到了这些使用相同数组的像素点，就有概率解出一个或多个密码簿中的偏差数组，从而实现无密码簿的破译。

相同尺寸下，位图中的颜色数量越少，其体积就越小；相反，颜色数量越多，体积就越大。J-5000 算法会使图片的颜色数量骤增，其体积也会随着颜色数量的骤增而骤增。在测试中，一张 10.4M 的图片加密后体积达到了惊人的 46.2M。

由于 Python 程序自身的缺陷，J-5000 算法在进行除了 PNG 以外的其他格式的图片时，会出现不可控的失真，从而导致程序无法正常运行。

所以，J-5000 算法的缺点有三：

1. 在加密像素点数量超过 2500 万的图片时，会出现严重的可靠性问题。
2. 使用 J-5000 加密的图像体积会非常大。
3. J-5000 算法只支持 PNG 格式文件的加密。

总结

这个算法可以说只是我一时的灵光乍现，它可以达到正常的加密效果，却也存在着诸多缺陷。未来我会尝试优化该算法，使其变得更加可靠。