学号 1206010331

年级 2012



**毕 业 设 计 (论 文)**

**基于排序变换和混沌Logistic映射的图像置乱软件实现**

|  |
| --- |
| **专业年级： 12级计算机科学与技术** |
| **学 号： 1206010331** |
| **姓 名: 王 金 塑** |
| **指导教师： 平 萍** |
| **评 阅 人：** |

**2016 年 6 月**

**中国**

**BACHELOR'S DEGREE THESIS**

**OF HOHAI UNIVERSITY**

**Picture scrambling software development based on sorting transformation and chaotic Logistic mapping**

College ：Hohai University

Subject ：Computer Science and Technology

Name ：Wang Jinsu

Directed by ：PingPing Professor

NANJING CHINA

**郑 重 声 明**

本人呈交的毕业论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计（论文）的研究成果不包含他人享有著作权的内容。对本设计（论文）所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计（论文）的知识产权归属于培养单位。

本人签名： 日期：

摘 要

众所周知，图像是由像素组成的。有序的像素组成了人们所看到的各种各样的图片。图像置乱算法的核心就是将原图像的像素搅乱，使得图像表达的意义失效，从而达到保护图像信息安全的目的。

“基于排序变换和混沌Logistic映射的图像置乱算法”是众多图像置乱算法中的一种。该算法首先利用混沌Logistic映射产生混沌序列，然后通过混沌序列的排序变换来得到图像置换的地址码，最后根据地址码完成图像置乱。

本文主要介绍通过对“基于排序变换和混沌Logistic映射的图像置乱算法”的分析与研究，将原算法中混沌序列排序、生成地址映射数组两个串行执行的算法步骤，改进为混沌序列排序过程中同步生成地址映射数组，使这两个步骤并行执行，从而提高算法效率。并且使用Java语言实现优化后的算法，结合目前市面上较为流行的Android系统，开发出一个图像置乱的安卓软件。

关键词：图像置乱；混沌序列；排序变换；安卓

ABSTRACT

As we know, image is made up of pixels. The ordered pixels form all kinds of pictures we see. The core of image scrambling algorithm is disrupting pixels’ sequence, so that the image can’t express the right meaning to achieve the goal of protecting the image information security.

“Picture Scrambling Algorithm Based on Logistic Mapping and Sort Transformation” is one of many image scrambling algorithms. Firstly this algorithm uses chaos Logistic mapping to produce chaotic sequence, and then get the address codes by sorting the chaotic sequences. Finally, the image scrambling is completed according to the address code.

This paper focuses on the analysis and research of “Picture Scrambling Algorithm Based on Logistic Mapping and Sort Transformation”. And mainly introduces how to put the step of “Sorting the chaotic sequences” and the step of “Getting the address codes” together so that we can make this algorithm more efficient. Also introduces how to use Java to develop the algorithm, and according to this algorithm to develop a mobile phone Android application about picture scrambling.

**Key words:** Picture Scrambling; Chaotic Sequence; Sort transformation; Android

**目 录**

[摘 要 I](#_Toc451710195)

[ABSTRACT II](#_Toc451710196)

[1 绪论 1](#_Toc451710197)

[1.1 研究背景及意义 1](#_Toc451710198)

[1.2 国内外研究现状 1](#_Toc451710199)

[1.3 选题依据及研究内容 2](#_Toc451710200)

[2 图像置乱 4](#_Toc451710201)

[2.1图像置乱算法的由来 4](#_Toc451710202)

[2.2基于像素位置变换的图像置乱算法 5](#_Toc451710203)

[2.3常见的基于像素位置变换的图像置乱算法 5](#_Toc451710204)

[2.3.1基于Arnold变换的图像置乱 5](#_Toc451710205)

[2.3.2基于幻方变换的图像置乱 6](#_Toc451710206)

[2.3.3基于Hilbert变换的图像置乱 8](#_Toc451710207)

[2.3.4基于排序变换和混沌Logistic映射的图像置乱 9](#_Toc451710208)

[3 基于排序变换和混沌Logistic映射的图像置乱算法的分析与优化 10](#_Toc451710209)

[3.1算法思路 10](#_Toc451710210)

[3.2算法复杂度分析 10](#_Toc451710211)

[3.3算法优化 12](#_Toc451710212)

[4 Java实现优化后的算法 15](#_Toc451710213)

[4.1算法辅助类实现 15](#_Toc451710214)

[4.2算法类实现 15](#_Toc451710215)

[4.2.1生成混沌实值序列 16](#_Toc451710216)

[4.2.2生成混沌序列的值与下标反向映射 17](#_Toc451710217)

[4.2.3混沌序列排序同时生成置乱地址集合 17](#_Toc451710218)

[4.2.4单行（列）置乱（解密） 18](#_Toc451710219)

[4.2.5多行（列）置乱（解密） 20](#_Toc451710220)

[4.2.6图像置乱（解密） 20](#_Toc451710221)

[5 Android实现图像置乱软件 21](#_Toc451710222)

[5.1 Android体系结构 21](#_Toc451710223)

[5.2软件设计 22](#_Toc451710224)

[5.2.1系统结构设计 23](#_Toc451710225)

[5.2.2功能设计 23](#_Toc451710226)

[5.3软件实现 24](#_Toc451710227)

[5.3.1开发环境 24](#_Toc451710228)

[5.3.2程序结构 24](#_Toc451710229)

[5.3.3主要代码 25](#_Toc451710230)

[5.4软件演示 28](#_Toc451710231)

[6 工作总结与展望 32](#_Toc451710232)

[6.1工作总结 32](#_Toc451710233)

[6.2展望 33](#_Toc451710234)

[参考文献 34](#_Toc451710235)

[致 谢 35](#_Toc451710236)

1 绪论

1.1 研究背景及意义

在每一天的每一分钟里，数据都在大量的被创造出来，这是一个大数据时代。而图像在这个数据爆发的时代中扮演着不可或缺的角色，据统计，在一分钟之内，FACEBOOK会产生2,460,000条用户分享的内容，其中90%以上的内容包含至少一张图片，WHATSAPP将有347,222张图片被用户发表，INSTAGRAM用户将发送216,000张新的图片[1]。图像由于其优越的信息表达能力，成为了人们相互交流信息的主要方式，随着互联网的蓬勃发展，越来越多的图像经过网络进行传播，这其中也许就涵盖了许多具有机密性的图像，例如个人隐私，商业机密以及军事机密等等。由于互联网是开放的，所以图像在传播过程中很有可能被怀有恶意的攻击者截取，导致机密泄露。因此图像加密技术应运而生。

数字图像在计算机中的存储形式是像素矩阵模式，一张图像中具有的像素数量可以很轻易的达到千万像素级别，因此二维的特性加上大数据量，使得数字图像的加密不能直接采用一些已有的经典算法，例如DES、3DES、AES和RSA[2-4]等，这些算法提供了良好的一维数据流加解密思想，而且也得到了广泛的应用，但由于数字图像的特殊性质，使得这些经典算法并不适合图像的加解密。因此，需要研究适用于数字图像加密的技术。

1.2 国内外研究现状

目前的数字图像加密技术已经较为成熟，国内外已有了很多突破性的进展，当今数字图像的加密技术可以归纳为以下的几个方面：

1、数字图像置乱：图像置乱技术[5]利用了数字图像是由象素矩阵组成的特点，通过将数字图像的象素矩阵置乱，使得图像无法表达正确的信息，并在需要解密时，通过相应的逆运算将被置乱的象素矩阵复原。一般来说，数字图像置乱算法可细分为以下三种类别：在频域上的图像置乱、在位置空间上的图像置乱、在色彩空间上的图像置乱。

2、数字图像分存：将数字图像的信息存储从物理上隔离，即将信息分散保存在各个地方，或者由不同的人进行保存， 这样使得想要窃取该数字图像的攻击者难以获取到完整的图片信息，同时也使拥有保密信息的人之间可以互相制约，从而提高了数字图像的信息保密性。

3、数字图像信息隐藏技术：信息隐藏技术将信息嵌入到数字媒体中以达到版权保护、识别、认证等目的。这一类技术通常在对通信安全性要求较高时，需要为数字添加水印时，或者对仿制品进行检测时被应用于其中。该技术主要思想是将涉密的数字图像嵌入另一张数字图像内，使其无法直接被人用肉眼看出，利用了公开图像的迷惑性，转移了他人对图像表达信息的注意力，从而将图像信息隐蔽。

4、数子图像水印技术：数字图像水印技术通过将数字水印植入到相关的数字图像中，使得原图看上去虽然没有改变，但当该图引发了版权纠纷后，可以通过相应的逆运算将数字水印提取出来，这样一来就可以知道图像的归属，为解决版权纠纷提供了一个技术上可行的方法。

其中数字图像置乱技术近年来常常被应用于图像加密中。对于一副数字图像，不仅可以在其空间域上进行置乱，还可以在其频率上进行置乱。国内一些研究图像置乱技术的学者把图像置乱技术分为三个类别：

（1）用分形图形学中的方法对空间曲线进行填充，这类方法对图像的像素点进行不重复，不遗漏的遍历，从而达到能够重新排序图像像素的目的，通过改变这些排序后的像素来对图像进行置乱。

（2）基于Arnold变换的一系列图像置乱方法，这些方法采用了古典恺撒密码方法把数字图像从颜色空间和相位空间两方面进行置乱，古典恺撒密码方法是密码学中的一种典型的密码方法。由于Arnold变换具有其周期性，所以Arnold系列的置乱方法正好可以利用这一性质进行置乱图像的还原。

（3）利用其他数学知识和奇特的现象进行图像置乱，例如按位运算，按三角函数变换，或者按异或运算的方法进行图像的置乱，这些方法的安全与否，是和算法本身以及算法的密钥直接相关的，这一类的方法也有其局限性，因为针对数字图像的置乱和解密一般都是采用同一算法进行。

1.3 选题依据及研究内容

对于一幅图像，可以选择从空间域上进行置乱或者从频率上置乱，本文研究的算法属于前者。图像具有像素，而像素具有坐标，这些像素的坐标组成了图像的空间域信息，通过搅乱图像的像素点坐标，即可搅乱图像的空间域，使得图像失真，是阅读者无法理解原图所要表达的信息，从而达到图像加密的目的。之后再通过相反的步骤，将被移动过的像素点恢复原位即可实现图像的还原。

基于排序变换和混沌Logistic映射的图像置乱方法，首先利用混沌Logistic映射产生混沌序列，然后通过混沌序列的排序变换来得到图像置换的地址码，最后根据地址码完成图像置乱。

本文在理解了基于排序变换和混沌Logistic映射的图像置乱方法的相关知识后，通过算法复杂度分析，发现该算法的效率不尽如人意，图像的像素量较大时，该算法在时间复杂度上的表现是不理想的。本文在该算法的基础上，进行了算法步骤上的优化，使得算法在时间复杂度上的表现较原算法更加高效，并根据优化后的算法开发出一个图像置乱的安卓应用。

2 图像置乱

2.1图像置乱算法的由来

随着图像在人们日常生活中的扮演的信息传递的角色分量越来越重，图像信息的安全性日渐被人们所关注，图像加密技术因此蓬勃发展。

说到加密，首先不得不谈起传统的加密技术。传统加密技术采用密钥加密的方式，并且可以根据密钥的公开与否分为公钥加密或者私钥加密。公钥加密也被称为非对称加密，这是因为在公钥加密中，加密和解密的密钥不同，加密密钥会被公开，而解密密钥只有解密者拥有。私钥加密也被称为对称加密，是指在加密和解密的过程中使用的是相同的密钥，并且密钥只有通信的双方知道。这样一来，通过对密钥的保护，避免了信息在传播过程中被人截取后获取其中内容。

信息在互联网中传播的方式是通过计算机网络协议将需要传输的数据信息封装成数据帧后，通过传输层和网络层的TCP/IP协议进行逻辑互联，通过物理层进行物理互联，并将二进制数据转化为规定的正负电压值的电信号在信道中互相传导，从而达到数据信息交互的目的。信息加密技术就是通过对二进制数据进行加密处理来保护数据安全。

从在计算机系统中的表示来看，图像也是二进制的数字，既然是二进制数字，那就意味着可以使用传统加密技术对其进行加密。不过，图像还具有自己的特性，比如图像数据的信息量比一般信息的数据量大，并且图像数据的冗余度也很高，而且图像一般都是以二维或三维的数据格式表示的，要加密还需要将二维或三维的图像数据转换成一维，加密后再转换回二维或三维。这导致使用传统的加密技术对图像进行加密难免造成效率低下，并且安全性也不高。

因此，国内外学者为图像设计了更加高效、安全的图像加密技术。

图像加密技术中，有一个很重要的分支就是本文研究的图像置乱算法。上文有说到传统的加密技术是对数据在计算机底层的表现——二进制上面做加密，但由于图像的特性，使得传统的加密对于图像来说并不适用。图像置乱就是针对图像的特性设计的一种加密技术。

图像置乱不再像传统加密技术一样着眼于数据的底层二进制的表示。而是根据图像自身所具备的特性，即根据像素矩阵与像素灰度值或像素颜色值来进行加密处理。图像置乱的主要目的是将图像本身进行搅乱，使得原图变成一张毫无意义的“乱码图”，无论是人还是计算机都无法从中获取图像所表达的有效信息，从而达到加密的目的。所以可以看出，传统加密技术的加密过程是着眼于在数据的底层表现，而图像置乱技术的加密过程是针对图像数据的顶层表现，这是两者之间最大的区别。

2.2基于像素位置变换的图像置乱算法

由于本文主要研究的算法属于基于像素位置变换的图像置乱算法，因此下文将对该类置乱算法进行详细介绍。

基于像素位置变换的图像置乱的定义[6]如下：

给出一幅大小的图像，假定变换是到自身的一一映射，如下式(2-1)所示



将图像中位置处的元素变换到位置处，得到图像，则称变换是图像的置乱变换。记为。

2.3常见的基于像素位置变换的图像置乱算法

到目前为止，基于图像位置空间的置乱算法有许多成熟的算法已经被应用到了图像置乱中，比如基于Arnold变换、幻方变换、Hilbert曲线、队列变换方法以及本文所研究的基于排序变换和混沌Logistic映射的置乱等等。

2.3.1基于Arnold变换的图像置乱

Arnold变换[7]是Arnold V J在遍历理论研究中提出的一种变换，因常常采用一张猫脸演示而得名。Arnold的变换过程用式(2-2)表示：



其中表示只取小数部分，即*.* 因此，的相空间被限制在单位正方形内，将其采用矩阵形式可描述为式(2-3)



其中， 是变换矩阵，其行列式为. 为图像的长和宽，是模运算，目的是确保置乱后的图像信息的位置范围不变，仍然落到图像区域内部。是位置的像素经过加密得到的新坐标。经过一次加密后图像还留有许多原始信息，即并没有完全达到彻底置乱的作用。在实际应用中，Arnold变换是指经过多次迭代的结果。运用式(2-3)经过反复变换，得到最终的置乱图像。

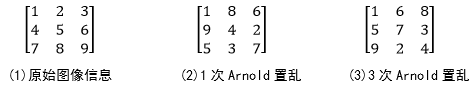


图2-1 Arnold变换示例图

如图2-1所示，(1)图是图像的原始信息，经过一次Arnold变换得到(2)图结果，同样经过三次Arnold变换得到最终的(3)图变换结果。能清楚看到，Arnold变换改变了原始信息存储位置，而信息内容没有发生改变。

Arnold置乱过程简单易懂，因而在图像信息处理中受到广泛使用。在使用Arnold变换时，随着迭代次数的增加，图像的相关性就越来越小。但是这样的加密方法有个弊端，这会使得图像像素点的位置变化具有庞加莱回复性，在迭代一定次数后，加密图像就变回原图了。因此Arnold变换往往会结合其他的加密算法来实现图像置乱。

2.3.2基于幻方变换的图像置乱

在方阵中，放入自然数，下面矩阵是以个自然数构造得到的阶矩阵(2-4)



其中如果矩阵满足式(2-5)



其中，就是标准自然数的幻方矩阵[8]，或简称为阶幻方，就是幻方和。

基于幻方变换的图像算法思路[9]为：

按变换的图像大小确定幻方的阶数，将图像矩阵和幻方按行列一一对应；把中的每个元素按下式(2-6)交换：



得到的变换矩阵，将中元素的位置相应移动。



经过上述的迭代变换分别可以得到做相应的移动可得到次变换后的图像。

其置乱方法的流程图可见图2-2。

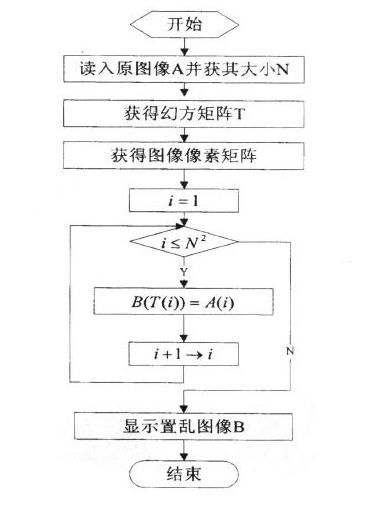


图2-2 幻方置乱流程图

2.3.3基于Hilbert变换的图像置乱

1987年Matlas和Shamir提出了FASS曲线。它是基于空间填充理论的置乱变换为基础的，Giuseppe Peano后来发现了一种平面曲线，能够沿着单位正方形迂回而行，有很好的“空间填充性”。后来，在德国数学家Hilbert的努力研究中，提出了可以填充满单位正方形的新的FASS曲线，后来被命名为Hilbert曲线。根据Hilbert曲线填充遍历路径，能将Hilbert变换路径大体分为9种类型[9]，图2-3是1阶情况下Hilbert曲线具体填充路径。

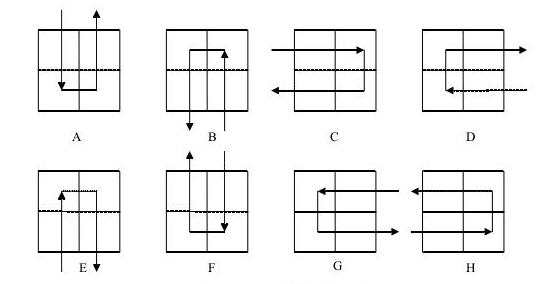


图2-3 Hilbert曲线路径分类

在实际应用中，1阶曲线的路径选取很灵活，置乱方法的选择空间也会相应的得到扩展。对于高阶情况来说，同样有不同的路径走向。图2-4是Hilbert曲线进行置乱处理的示意图。

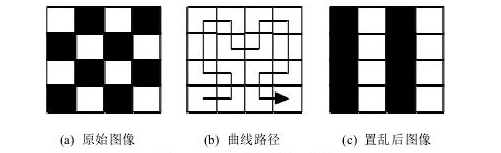


图2-4 Hilbert曲线置乱示意图

从图中可以看出，经过Hilbert曲线置乱变换后的图像，还保持着相邻像素之间的相关性，所以会导致置乱效果得不到提高。为了提高图像的置乱效果，需要进行反复置乱变换，相应的计算量和工作量也就大大增加了。

2.3.4基于排序变换和混沌Logistic映射的图像置乱

该算法即为本文研究的算法，和上述几种算法相同，基于排序变换和混沌Logistic映射的图像置乱算法也是基于图像像素点坐标的空间域置换实现的。目前应用混沌的图像置乱方案一般都是先通过计算混沌模拟序列[10]，再对其进行多值量化来生成置换地址码。由于对混沌模拟序列进行量化要求对混沌轨道的概率分布有相当的了解，而这正是混沌研究的一个仍未解决的难题，并且易受量化精度的影响，因此多值量化所得置换地址码很难完全保持混沌固有的特性。另外，由于置换地址码的特殊要求，要求必须遍历所有置换地址，而为能便利所有的地址，算法不得不经历许多不能生成置换地址码的迭代，这就增加了算法的时间复杂度。而基于排序变换和混沌Logistic映射的图像置乱算法，其置换地址码的产生则不需要对混沌实值序列进行量化，而是通过排序变换直接由混沌模拟序列来产生。该方法不需要对轨道分布具有先验知识，即可以选用任何一个混沌模型，这就极大地扩展了算法的密码空间。该算法不仅大大减少了混沌映射迭代次数，并能很好地利用混沌的特性，同时由于排序变换的强不规则性，也增加了破译置乱图像的难度。

下一章将对该算法进行详细介绍。

3 基于排序变换和混沌Logistic映射的图像置乱算法的分析与优化

3.1算法思路

对于一个数字图像，其大小为，可以利用任一混沌迭代式(3-1)



来产生混沌实值序列，然后通过下面描述的置乱算法即可对图像逐行进行置乱和解密。

置乱算法步骤：

1. 给定迭代初始值（相当于密钥）；
2. 经过次混沌迭代运算得到混沌实值序列；
3. 通过排序变换，将实值序列集合中的个值由小到大排序，形成有序序列；
4. 确定混沌实值序列中每个在有序序列中的位置编号，形成置换地址集合，其中，；
5. 按置换地址集合对图像的第行像素进行置换，同时将其第列像素置换至第列，；
6. 置，对到行，重复步骤（2）到步骤（5）。

解密算法步骤：

对于给定的密钥（迭代初始值），可采用类似置乱的步骤，即只需将步骤（5）改为：按置换地址集合对图像的第行像素进行置换，其第列像素置换至第列，即可实现图像的解密。

3.2算法复杂度分析

由于置乱算法和解密算法的步骤基本相同，下面对于上述的置乱算法的步骤进行空间复杂度和时间复杂度[11]上的分析。

假定图像的大小为，且，对行像素（每行个像素，假定）进行置乱的复杂度分析如下：



1.空间复杂度：

（1）置乱算法通过混沌迭代运算获取一个混沌实值序列，由于该序列后面将用于排序变换和生成置换地址集合，需要一个大小的空间保存该实值序列；

（2）将混沌实值序列按升序排序后，得到一个有序的实值序列，该序列将用于和原混沌实值序列比对后生成置换地址集合，也需要一个的空间保存；

（3）通过比对原混沌实值序列和排序后的有序序列，可以得到一个记录每个的位置编号的集合，即置换地址集合。该集合需要的空间保存；

（4）在更具置换地址集合进行像素置换时，由于置换前后的位置的不同，不能直接在原像素矩阵中进行置换，否则将可能使得被置换的像素被覆盖，导致置换不正确。因此，在置换时需要一个大小为的临时数组存放被置换的像素，之后使用这个数组对原像素矩阵的行或者列进行像素位置置换。

总的来说，置乱和解密算法的空间复杂度，量级都是这一级别的。

2.时间复杂度：

（1）生成混沌实值序列的方法一般都是用推导出，因此获取一个长度为的实值序列，需要迭代次，即时间复杂度为；

（2）对生成的混沌实值序列进行排序，其时间复杂度和选取的排序算法直接相关，比如复杂度为的选择、插入、冒泡等常见的排序算法，或者复杂度为的快排、归并、希尔以及堆排这样的高效算法。这里假定选择了复杂度为的排序算法；

（3）生成置换地址集合需要遍历原混沌实值序列和排序后的实值序列，从而得知原序列中的元素在有序序列中的位置，从而得到地址集合。因此这一步骤的算法复杂度是；

（4）根据置换地址集合对一行像素进行位置置换，只需遍历一次该行像素即可。复杂度为；

（5）对每一行都进行上述操作，则行置乱下来，复杂度为



即为量级。

通过分析可知，该算法的空间复杂度较好，为量级。但时间复杂度不太理想，为量级。

为方便计算，这里将排序算法换成级的算法，上述的时间复杂度是总的量级，如果细分一下的话，真正的复杂度公式应该为：

**

这是行置乱，如果再加上列置乱，即。

假设现在有一副图像，大小为，则对该图像进行置乱的话，需要进行次操作，也就意味着用这样的置乱过程对一幅普通大小的图片进行置乱，将可能遭遇到的最坏情况，将进行五亿零五十万次基本操作。

这是比较大的量级，所以本文考虑将原算法的置乱过程进行优化，尽可能地降低算法消耗的时间资源。

3.3算法优化

通过原算法步骤可以看出，该算法的算法复杂度主要集中在步骤（3）和步骤（4）上，步骤（3）是对混沌实值序列进行排序，步骤（4）则是生成置换地址集合。

步骤（3）可以直接通过排序算法来优化，选择效率较高的高级排序算法，使复杂度降为，但步骤（4）需要遍历原序列和排序后的序列，其复杂度固定为，整体算下来，总的时间复杂度变为：



取，



较之前采用复杂度为的排序算法的置乱，效率提高，但这仅仅优化了步骤（3），步骤（4）的复杂度依然无变化。

因此，在经过细致的研究后，本文采用了另一种更高效的优化方式。

原算法的置乱步骤中，步骤（4）的复杂度无法优化，是因为步骤（4）是基于步骤（3）进行的，这导致步骤（4）必须等到排序完成后，遍历两组序列，使得圈复杂度上升到平方级别。

但是经过思考不难发现，步骤（4）的目的是生成置换地址集合，而置换地址集合其实就是表示原序列中的元素在排序后序列中位置的一个数组。如果采取一种排序算法，使得每一趟排序后，当前排序的元素所在的位置，就是整个序列排序后，它所处的位置，那么无须等到序列完全有序后再去经过双重循遍历两个序列来生成地址集合，可以直接在每一趟排序后直接生成当前元素的地址映射，当排序完成后，地址集合也就同时生成了。

这使得原本串行执行的步骤（3）和步骤（4）转变成了并发执行，步骤（3）内包含了步骤（4），这样对算法的效率将会产生较大的提升。

那么，什么样的排序算法可以达到每一趟排序后当前元素位置即为完全有序序列中该元素的位置的要求呢。答案是选择排序算法[12]。

回顾选择排序的思路，每一趟比较中，选取当前下标元素，使与逐一对比，找到中最小的元素和对比，如果比大则，将与的位置置换，反之则保持位置不变，其中。

这样的排序性质，保证了每一趟排序后，当前下标为的元素一定是范围内最小的元素。这意味着当前元素的下标即为完全有序序列中该元素的下标。

由此可见，可以采用选择排序将原来的步骤（3）和步骤（4）变为并发执性，从而进行对原算法的优化。

选择算法的时间复杂度是，现在步骤3、4合一后，混沌序列的排序和置换地址集合的生成的复杂度直接由排序算法决定，因此优化后的算法复杂度分析如下：



取，



较原算法的方案效率提升，比之前采用高级排序算法的方案再度提升了近的效率。考虑到选择排序算法的平均复杂度表现，提升的效率将更大。

因此本文最后选择采用该优化方案实现基于排序变换和混沌Logistic映射的图像置乱算法。

4 Java实现优化后的算法

确定了算法优化的方案之后，本文使用Java语言将该置乱算法实现。实现的算法主要分为两个Java类，算法辅助类ArrayFunctions，与算法类myAlgorithms。

4.1算法辅助类实现

考虑到之后需要将算法移植到Android中实现APP，而Android中的图像像素矩阵的数据结构又是一维整形数组，所以为了方便算法根据图像的二维性质置乱或解密，需要将一维数组转换为二维数组，并且在置乱或解密完成后再将二维数组还原成一维数组，以及为了在列置乱或解密时，需要将图像矩阵行列互换，从而使得行与列可以共用一套置乱或解密的算法，本文首先实现了一个辅助类——ArrayFunctions，该类封装了一系列关于数组操作的方法。

该类结构如下表(4-1):

表4-1 ArrayFunctions类结构表

|  |  |  |
| --- | --- | --- |
|  | ArrayFunctions类 | |
| 方法 | 方法描述 |
| 1 | change(int arr1[],int arr2[][],int M,int N) | 一维数组转二维数组 |
| 2 | recovery(int arr1[][],int arr2[],int M,int N) | 二维数组转一维数组 |
| 3 | arr\_change(int arr[][],int temp[][],int M,int N ) | 二维数组行列互换 |
| 4 | arr\_copy(double arr1[],double arr2[],int N) | 数组拷贝 |

其中change()和recovery()用于一维数组和二维数组互相转换，arr\_change()用于列置乱或解密时的像素矩阵行列互换，arr\_copy()用于拷贝混沌实值序列，便于置乱或解密时的操作。

4.2算法类实现

基于辅助类ArrayFunctions，myAlgorihtms类实现了优化后的基于排序变换和混沌Logistic映射的图像置乱算法。

该类结构如下表(4-2):

表4-2 myAlgorihtms类结构表

|  |  |  |
| --- | --- | --- |
|  | myAlgorithms类 | |
| 方法 | 方法描述 |
| 1 | produce\_logisticArray(double x,double arr[],int N) | 生成混沌实值序列 |
| 2 | produce\_map(HashMap m,double logistic\_array[],int N) | 混沌序列的值-下标反向映射 |
| 3 | SelectSort(double arr[],int length,HashMap m,int address\_arr[]) | 核心算法，选择排序过程中同时生成置换地址映射集合 |
| 4 | rowEncrypt(int pixel[][],double x1,int i,int M,int N) | 单行置乱 |
| 5 | rowEncrypt\_interface(int pixel[][],double x1,int M,int N) | 行置乱接口 |
| 6 | columnEncrypt(int pixel[][],double x1,int i,int M,int N) | 单列置乱 |
| 7 | columnEncrypt\_interface(int pixel[][],double x1,int M,int N) | 列置乱接口 |
| 8 | rowDecrypt(int pixel[][],double x1,int i,int M,int N) | 单行解密 |
| 9 | rowDecrypt\_interface(int pixel[][],double x1,int M,int N) | 行解密接口 |
| 10 | columnDecrypt (int pixel[][],double x1,int i,int M,int N) | 单列解密 |
| 11 | columnDecrypt\_interface(int pixel[][],double x1,int M,int N) | 列解密接口 |
| 12 | encrypt(int pixel[][],double x1,int M,int N) | 图像置乱总接口 |
| 13 | decrypt(int pixel[][],double x1,int M,int N) | 图像解密总接口 |

下面将分别介绍上述方法在置乱算法实现中起到的作用。

4.2.1生成混沌实值序列

通过关于基于排序变换和混沌Logistic映射的图像置乱算法的介绍，可以知道该算法的第一个步骤是要选用一种混沌迭代式生成混沌实值序列，再依据混沌实值序列进行接下来的算法步骤。

本文选用的混沌迭代式如下式(4-1):



其中，为控制参数，。当时，该系统有一个定常解（即初值取时会使生成的序列全部为），而且无论初值取为何值，通过多次迭代，序列会最终收敛于。

当时，定常解为、和，多次迭代后序列会收敛于这两个值中的一个。当时，系统由倍周期通向混沌。特别地，当时，系统进入混沌状态，迭代生成的值处于一种伪随机分布的状态，而且取值越接近，混沌性越强。当时，Logistic映射的Lyapunov指数为。

produce\_logisticArray()方法实现了上述混沌迭代式，代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | void produce\_logisticArray**(**double x**,** double arr**[],** int N**)**  **{**  double u **=** 3.9999999**;**  arr**[**0**]** **=** x**;**  **for** **(**int i **=** 1**;** i **<** N**;** **++**i**)**  **{**  arr**[**i**]** **=** u**\***arr**[**i **-** 1**]** **\*** **(**1 **-** arr**[**i **-** 1**]);**  **}**  **}** |

该方法为了保证序列的强混沌性，选取控制参数的值为，然后接收一个初始密钥，使用混沌迭代式生成长度为的混沌实值序列。

4.2.2生成混沌序列的值与下标反向映射

为了实现对混沌序列进行排序的同时生成置换地址集合，需要在每一趟排序的过程中，获取当前下标元素在原混沌序列中的下标，然后通过当前下标和原序列的下标生成地址集合。

考虑到数组只能单向地通过下标访问元素值的特性，要想通过元素值找到其在数组中的下标需要的时间，算上选择排序的复杂度，整体将上升到级。所以本文采用了以空间换时间的方式来降低时间复杂度，即建立一个混沌序列的值与下标的反向映射Map。

produce\_map()方法实现了值与下标的反向映射的建立，代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | void produce\_map**(**HashMap m**,** double logistic\_array**[],** int N**)**  **{**  **for** **(**int i **=** 0**;** i **<** N**;** **++**i**)**  **{**  m**.**put**(**logistic\_array**[**i**],** i**);**  **}**  **}** |

4.2.3混沌序列排序同时生成置乱地址集合

由于选择排序法的每一趟排序后，当前位置的元素即为整个序列排序后该元素的位置，所以选择排序法能够将原算法的混沌序列排序与置乱地址集合生成两个步骤合二为一。

SelectSort()方法实现了排序过程中的置乱地址集同步生成，代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | void SelectSort  **(**double arr**[],** int length**,** HashMap m**,** int address\_arr**[])**  **{**  **if** **(**arr **==** **null** **||** length **<=** 0**)return;**  int index **=** 0**;**  **for** **(**int i **=** 0**;** i **<** length**;** **++**i**)**  **{**  index **=** i**;**  **for** **(**int j **=** i**;** j **<** length**;** **++**j**)**  **{**  **if** **(**arr**[**j**]** **<** arr**[**index**])**index **=** j**;**  **}**  **if** **(**index **!=** i**)**  **{**  double temp**;**  temp**=**arr**[**i**];**  arr**[**i**]=**arr**[**index**];**  arr**[**index**]=**temp**;**  **}**  int address**=**Integer**.**parseInt**(**String**.**valueOf**(**m**.**get**(**arr**[**i**])));**  address\_arr**[**address**]** **=** i**;**  **}**  **}** |

SelectSort()的参数arr为待排序的混沌序列，m为混度序列值与下标的反向映射Map，address\_arr为待生成的置乱地址集。

代码第20、21行意为在选择排序的每一趟排序过后，通过m.get(arr[i])获取当前下标i的元素值在原混沌序列中的下标，保存在变量address中，由于置乱地址集数组的下标表示原混沌序列元素的位置，置乱地址集数组的值表示经过排序后元素所在的位置。所以通过address\_arr[address]=i（i为排序后元素的位置，address为元素在原混沌序列的位置）即可生成当前元素的置乱地址信息。

4.2.4单行（列）置乱（解密）

通过置乱地址集合即可对像素矩阵进行单行的置乱，只需将每一个像素的位置移动到对应的映射地址即可。

rowEncrypt()方法实现了单行置乱的功能，代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | double rowEncrypt**(**int pixel**[][],** double x1**,** int i**,**int M**,** int N**)**  **{**  ArrayFunctions af**=new** ArrayFunctions**();**  //生成混沌序列  double logistic\_array**[]** **=** **new** double**[**N**]** **;**  produce\_logisticArray**(**x1**,** logistic\_array**,** N**);**  //建立值与下标映射的Map  HashMap m**=new** HashMap**<**String **,** Double**>();**  produce\_map**(**m**,** logistic\_array**,** N**);**  //拷贝混沌序列  double temp\_logArr**[]=new** double**[**N**];**  af**.**arr\_copy**(**logistic\_array**,** temp\_logArr**,** N**);**  int address\_array**[]=new** int**[**N**];**  //对混沌序列进行排序，采用选择，并且同时生产地址映射表  SelectSort**(**temp\_logArr**,** N**,** m**,** address\_array**);**  //用一个暂存数组保存被置乱后的象素数组  int temp**[]=new** int**[**N**];**  **for** **(**int j **=** 0**;** j **<** N**;** **++**j**)**  **{**  temp**[**address\_array**[**j**]]** **=** pixel**[**i**][**j**];**  **}**  //正式置乱原图  **for** **(**int j **=** 0**;** j **<** N**;** **++**j**)**  **{**  pixel**[**i**][**j**]** **=** temp**[**j**];**  **}**  **return** logistic\_array**[**N **-** 1**];**  **}** |

该方法首先调用produce\_logisticArray()方法生成混沌序列，再调用produce\_map()方法生成值-下标反向映射，接着调用SelectSort()方法生成置乱地址集合，最后根据置乱地址集合对像素矩阵的一行进行像素位置置乱。其中用到了一个临时数组temp，作用是暂存置乱后的行像素，避免直接在像素矩阵的行上进行操作，导致像素信息被覆盖而无法正确完成像素置乱。

由于单列的置乱和单行的置乱功能相同，columnEncrypt()实现了列加密功能，在此不再赘述。

单行和单列的解密仅仅在像素位置置乱这一步骤不同，即进行与置乱相反的操作即可解密，只需将代码第20行的temp[address\_arr[j]]=pixel[i][j]改为temp[j]=pixel[i][address\_array[j]]即可实现解密功能。

rowDecrypt ()方法和columnDecrypt()方法分别实现了行与列的解密功能，在此也不再赘述。

4.2.5多行（列）置乱（解密）

对图像像素矩阵的每一行执行rowEncrypt()方法即可实现图像的行置乱，同理也可以实现列置乱，行置乱加上列置乱可以使得图像的置乱效果更好，但在解密时需要按照置乱时相反的顺序来解密，比如先行置乱后列置乱，则解密时要先列解密再行解密。

rowEncrypt\_interface()方法实现了多行置乱，代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | void rowEncrypt\_interface**(**int pixel**[][],** double x1**,** int M**,** int N**)**  **{**  double x **=** x1**;**  //对每一行都进行置乱  **for** **(**int i **=** 0**;** i **<** M**;** **++**i**)**  **{**  x **=** rowEncrypt**(**pixel**,** x**,** i**,** M**,**N**);**  **}**  **}** |

类似与多行置乱的rowEncrypt\_interface()，rowDecrypt\_interface()，columnEncrypt\_interface(),columnDecrypt\_interface()分别实现了多列置乱、多行解密与多列解密。

4.2.6图像置乱（解密）

基于上述一系列方法，本文实现的算法最终通过使用encrypt()方法先后调用rowEncrypt\_interface()与columnEncrypt\_interface()实现图像的行列置乱，与置乱相似，用decrypt()方法先后调用columnDecrypt\_interface()与rowDecrypt\_interface()实现图像的行列解密。

5 Android实现图像置乱软件

根据选题要求，本文使用Android平台实现图像置乱软件。

5.1 Android体系结构

Android是Google开发的操作系统，也是近年来市面上非常流行的移动开发平台。Android最初是由私人进行开发的操作系统，当时Android主要开发用于支持手机。Google于2005年收购了Android,并且在2007年联合众多硬件厂商，以及软件开发商和电信运营商着手建设开放手机联盟，并且共同研发Android系统。之后，在2008年Google以开源的方式发布了Android源码，并且在同年发布了第一部Android智能手机。如今搭载了Android系统的手机的全球市场份额已经超过了80%，使其成为了全球用户量最大的移动设备操作系统。

Android的体系结构层次主要分为四层，由下自上分别是操作系统层，库和运行环境层，应用程序框架层以及应用层（图5-1）。

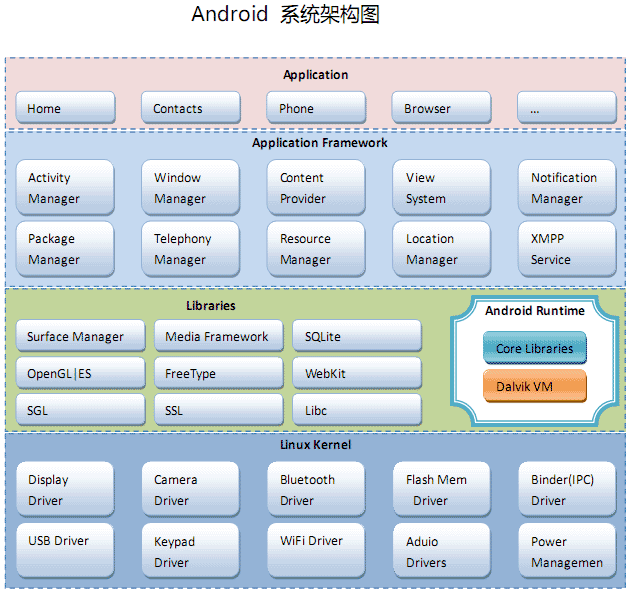


图5-1 Android系统架构图

由于Android源于Linux系统，所以Android操作系统层使用的是Linux系统内核。其中主要使用了列如Wifi驱动、键盘驱动、显示驱动、内存驱动、声卡驱动、蓝牙驱动、能源管理驱动等一些与移动设备息息相关的驱动程序。

Android的中间层由各种库和运行环境组成。用到的库有SGL、SSL、OpenGL ES 1.0、C语言标准库、SQLite、WebKit,FreeType等等与移动设备平台应用密切相关的中间件。Android的运行环境是指虚拟机技术，Dalvik。Dalvik虚拟机和一般Java虚拟机（Java VM）不同，它执行的不是Java标准的字节码（Bytecode），而是Dalvik可执行格式（.dex）中的执行文件。在执行的过程中，每一个应用程序即一个进程（Linux的一个Process）。二者最大的区别在于，Java VM是基于栈的虚拟机（Stack-based），而Dalvik是基于寄存器的虚拟机（Register-based）。显然，后者最大的好处在于可以根据硬件实现更大的优化，这更适合移动设备的特点。

Android应用程序框架层是为开发Andorid应用的开发者服务的。该层提供了各种应用开发所需的API，包括各种UI控件，以及相应的事件响应机制，消息传递机制等。为了实现一个Android应用，可以通过使用Android应用程序框架层的五个部分，Activity（活动）、Broadcast Intent Receiver（广播意图接收者）、Service（服务）、Content Provider（内容提供者）和Intent and Intent Filter（意图和意图过滤器）。

最上层应用层主要是面对用户的，包括Android自带的一些核心应用如主屏幕、联系人、电话、浏览器等，或者是由开发者基于应用程序框架层提供的API而开发的应用程序。

5.2软件设计

本文所实现的图像置乱软件使用的技术属于上述Android体系结构层次中的第三层，即应用程序框架层，实现过程中主要用到了框架中的Activity以及Intent and Intent Filter组件来实现软件主体功能。其中Activity利用了基类View来实现用户界面的渲染，以及用户事件监听，并且加入Java实现的置乱算法来实现业务逻辑。Intent负责在不同的组件之间传递消息，将一个组件的请求意图传达给另一个组件。

5.2.1系统结构设计

本文采用前后端分离的分层系统架构模式，分为前端用户界面渲染层以及后端业务处理逻辑层。由于该软件直接对图像进行处理，所以无需用到服务层。

图像置乱软件的系统架构层次如下图(5-2)



图5-2 置乱软件系统架构层次图

由架构层次图可知，图像置乱软件需要三个UI界面，分别是开始界面，运行界面，直方图界面。其中开始界面为用户打开硬软件时的开机界面，之后由该界面跳转至运行界面，运行界面主要负责图像置乱与加密的业务逻辑，在该界面点击直方图可跳转至直方图界面查看直方图信息。

5.2.2功能设计

主要为下述五个主体功能：

**图片获取**：获取用户手机存储中的图片，并展示在运行界面；

**图片置乱**：通过用户输入的密钥，对用户所选择的图片进行置乱并展示在运行界面；

**图片解密**：通过用户输入的密钥，对置乱后的图片进行解密并展示在运行界面；

**添加噪声**：为图片添加噪声，用于测试置乱算法对噪声的抗性；

**图片直方图**：用图表的形式展示当前运行界面图片的直方图信息。

5.3软件实现

5.3.1开发环境

软件开发环境如下表(5-1)

表5-1 软件开发环境

|  |  |
| --- | --- |
| 开 发 环 境 | |
| IDE： | eclipse 4.4.2 |
| OS： | Windows 8.1 |
| SDK： | Android SDK 4.4.2 |
| LANGUAGE： | Java & Android |

5.3.2程序结构

程序共有五个Java类，分别是StartActivity，RunTimeActivity，BarchartActivity，ArrayFunctions，myAlgotithms。UML类结构图如下图(5-3)

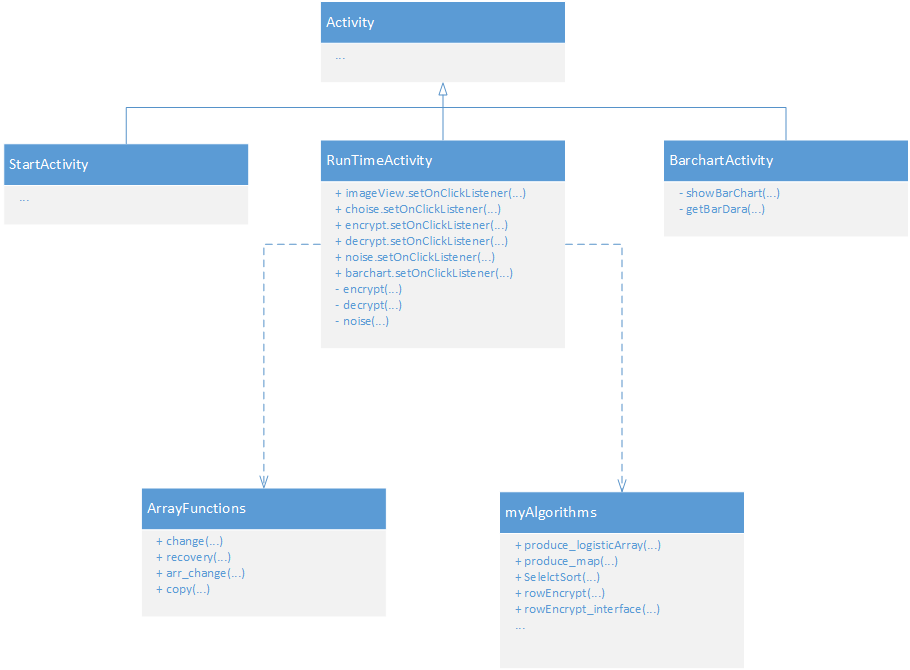


图5-3 UML类结构图

StartActivity，RunTimeActivity，BarchartActivity三个类继承了Android应用程序框架的组件Activity,其中StartActivity用于展示程序开始界面，BarchartActivity用于展示图片的直方图界面，而RunTimeActivity主要负责程序主要的功能逻辑和程序的主界面，RunTimeActivity通过使用ArrayFunction类与myAlgorithms类，实现图片的加密解密过程。

5.3.3主要代码

RunTimeActivity负责图像置乱软件的5个主要功能，实现代码如下：

**图片获取功能：**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | choise**.**setOnClickListener**(new** View**.**OnClickListener**()** **{**  public void onClick**(**View v**){**  Intent intent**=new** Intent**(**Intent**.**ACTION\_GET\_CONTENT**);**  intent**.**setType**(**"image/\*"**);**  startActivityForResult**(**intent**,**GALLERY\_REQUEST\_CODE**);**  **}**  **});** |

上述代码为选择图片按钮（choise）添加点击事件监听方法，按钮的作用为访问用户手机上的相册，并将选择的图片显示在运行界面的图片框（ImageView）内。

**图片置乱功能：**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | encrypt**.**setOnClickListener**(new** View**.**OnClickListener**()** **{**  public void onClick**(**View v**){**  key1 **=** **(**EditText**)**findViewById**(**R**.**id**.**key1**);**  imageView**=(**ImageView**)**findViewById**(**R**.**id**.**imageView**);**  String key1Str**=**key1**.**getText**().**toString**().**trim**();**  **if(**imageView**.**getDrawable**()==null)**  **{**  Toast**.**makeText**(**RunTimeActivity**.this,** "请选择一副图片"**,** Toast**.**LENGTH\_LONG**).**show**();**  **return;**  **}**  **else**  **{**  **if(**key1Str**.**length**()==**0**)**  **{**  Toast**.**makeText**(**RunTimeActivity**.this,** "请输入密钥"**,** Toast**.**LENGTH\_LONG**).**show**();**  **return;**  **}**  **else**  **{**  //获取输入的密钥  EditText text**=(**EditText**)**findViewById**(**R**.**id**.**key1**);**  String str **=** text**.**getText**().**toString**();**  double x**=**Double**.**valueOf**(**str**);**  **if(**x**>**0**&&**x**<**1**)**  **{**  encrypt**(**x**);**  key1**.**setText**(**""**);**  **}**  **else**  **{**  Toast**.**makeText**(**RunTimeActivity**.this,** "密钥应为0~1之间的任意小数(不包括0与1)"**,** Toast**.**LENGTH\_LONG**).**show**();**  **return;**  **}**  **}**  **}**  **}**  **});** |

上述代码为置乱按钮（encrypt）添加点击事件监听方法，按钮的作用为通过用户输入的密钥，对ImageView中的图片进行置乱。

通过**(**EditText**)**findViewById**(**R**.**id**.**key1**)**获取用户输入的密钥key，通过(ImageView)findViewById(R.id.imageView)获取当前图片，将密钥转换为double类型的变量保存入x中，然后通过encrypt(x)对图片进行加密。

encrypt(double x)的代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | void encrypt**(**double x**)**  **{**  //获取算法对象  myAlgorithms ma**=new** myAlgorithms**();**  ArrayFunctions af**=new** ArrayFunctions**();**  //获取图像像素矩阵的行数与列数  Bitmap bmp**=** **((**BitmapDrawable**)**imageView**.**getDrawable**()).**getBitmap**();**  int M**=**bmp**.**getHeight**(),**N**=**bmp**.**getWidth**();**  //获取图像像素矩阵  int**[]** pixel**=new** int**[**M**\***N**];**  bmp**.**getPixels**(**pixel**,** 0**,** N**,** 0**,** 0**,** N**,** M**);**  //像素矩阵转二维  int **[][]**pixels **=** **new** int**[**M**][**N**];**  af**.**change**(**pixel**,** pixels**,** M**,** N**);**  //进行加密  ma**.**encrypt**(**pixels**,** x**,** M**,** N**);**  //加密后矩阵降一维  af**.**recovery**(**pixels**,** pixel**,** M**,** N**);**  //生成加密后的图像  Bitmap bitmap **=** Bitmap**.**createBitmap**(**pixel**,** 0**,** N**,** N**,** M**,** Bitmap**.**Config**.**ARGB\_8888**);**  imageView**.**setImageBitmap**(**bitmap**);**  **}** |

ArrayFunctions和myAlgorithms是上文介绍过的算法类，encrypt(double x)方法首先通过((BitmapDrawable)imageView.getDrawable()).getBitmap()将图片转换为Bitmap类型，该类型内封装了一系列关于图片的方法。

其中getHeight()、getWidth()、getPixels()分别是Bitmap类型中用于获取图片高度，宽度，以及像素矩阵的方法。通过这三个方法获取到图片的高，宽以及像素矩阵后，调用myAlgorithms类中的encrypt()方法即可对图片的像素矩阵进行置乱。

**图片解密功能：**

图像解密功能与图像加密功能类似，只需调用myAlgorithms类中的decrypt()方法即可。

**噪声功能：**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | void noise**(){**  int i**;**  //获取图像像素矩阵的行数与列数  Bitmap bmp**=** **((**BitmapDrawable**)**imageView**.**getDrawable**()).**getBitmap**();**  int M**=**bmp**.**getHeight**(),**N**=**bmp**.**getWidth**();**  //获取图像像素矩阵  int**[]** pixel**=new** int**[**M**\***N**];**  bmp**.**getPixels**(**pixel**,** 0**,** N**,** 0**,** 0**,** N**,** M**);**  **for(**i**=**0**;**i**<(**M**\***N**);**i**+=**50**){**  pixel**[**i**]** **=** 255**;**  **}**  Bitmap bitmap **=** Bitmap**.**createBitmap**(**pixel**,** 0**,** N**,** N**,** M**,** Bitmap**.**Config**.**ARGB\_8888**);**  imageView**.**setImageBitmap**(**bitmap**);**  **}** |

噪声功能主要通过遍历像素矩阵，将矩阵中的部分像素的值置为255，用于测试置乱和解密的抗噪声能力。

**直方图功能：**

直方图功能使用了github的一个开源图表组件：BarChart，该组件内有很多关于图表的接口，使用时只需传给接口符合要求的参数即可实现图表的渲染，由于该组件是开源的，所以在此不再赘述，有兴趣者可以访问github查看相关源码以及API。

5.4软件演示

以下演示使用的是搭载5.1版本Android系统，型号为Huawei TAG-AL00的移动智能手机。

（1）图5-4为APP图标，5-5为运行界面

|  |  |
| --- | --- |
| QQ图片20160516212234 | QQ图片20160516212253 |
| 图 5-4 APP名称与图标 | 图5-5 运行界面 |

APP名称为PicEncrypt意为图像加密，图标选用了一张包含众多金属工具的工具袋，意味着加密和解密的过程。进入APP后，跳转到运行界面，界面由5个按钮分别是图片选择、置乱、解密、噪声以及直方图，还有一个图片展示框和一个密钥输入框组成。APP默认展示了一张图片于图片展示框中，用户也可以选择自己手机上的图片进行操作。

点击图片选择将访问手机的相册选择图片后返回运行界面，并将选择的图片显示在图片展示框中。访问相册和选择图片见图5-6和图5-7

|  |  |
| --- | --- |
| QQ图片20160516212303 | QQ图片20160516212242 |
| 图5-6 访问用户相册 | 图5-7 展示用户选择图片 |

选择了图片后，在密钥输入框中输入密钥即可进行图像置乱操作，密钥输入框限制只能输入数字，并且密钥的大小为0~1之间的浮点数，当输入的密钥超出范围，将提示用户输入范围内的密钥。密钥输入和提示见图5-8与图5-9。

|  |  |
| --- | --- |
| QQ图片20160516220201 | QQ图片20160516220205 |
| 图5-8 密钥输入 | 图5-9 密钥范围提示 |

输入密钥后，点击置乱按钮可对图片进行置乱，置乱后再输入正确的密钥点击解密可将图片还原，若输入的密钥与原密钥不相等，则无法还原图片。

图片置乱见图5-10与图5-11，图片解密见图5-12与5-13。

|  |  |
| --- | --- |
| QQ图片20160516212246 | D:\学习\毕设\图\QQ图片20160516212250.jpg |
| 图5-10 输入密钥点击置乱 | 图5-11 置乱后的图片 |
| QQ图片20160516212230 | QQ图片20160516212307 |
| 图5-12 输入密钥点击解密 | 图5-13 解密后的图片 |

将图片置乱后，点击噪声可以为图片添加若干噪声点，再点击解密后可以比对加了噪声后对图片解密的影响，从而判断置乱解密算法的抗噪性。

添加噪声见图5-14，添加噪声后的解密见图5-15。

|  |  |
| --- | --- |
| QQ图片20160516212238 | QQ图片20160516212300 |
| 图5-14 点击噪声 | 图5-15 解密后的图片 |

由图5-15可以看出，添加噪声后再解密的图片并没有受到太大影响，图片依然能够表达出正确的信息，因此可以证明基于排序变换和混沌Logistic映射的图像置乱算法的抗噪性是比较强的。

点击直方图按钮可以进入直方图界面，在该界面将显示图片的直方图，将图片的RGB属性分为三个柱形图，分别代表RGB三原色。纵坐标表示像素个数，横坐标表示像素的值。直方图功能见图5-16和图5-17。

|  |  |
| --- | --- |
| QQ图片20160516212257 | QQ图片20160516212212 |
| 图5-16 直方图 | 图5-17 放大后的直方图 |

6 工作总结与展望

本文至此就基本结束了，下面对这次毕业设计做一下总结，并且谈谈自己对图像加密技术未来的展望。

6.1工作总结

由于我最初版本的算法实现是用C++完成的，之后为了能够将算法嵌入Android中，又将C++实现的算法移植为Java版本，所以我将这次毕业设计的工作分为五个阶段，分别是算法分析，C++算法实现，算法优化，代码移植以及Android软件实现。

算法分析阶段主要阅读了导师给的参考论文，了解了图像置乱的概念，并且对基于排序变换和混沌Logistic映射的图像置乱算法进行了细致的分析，包括算法的空间复杂度和时间复杂度，形成了算法实现的大致思路。当时虽然觉得算法的时间复杂度较高，但是还没想到要对算法进行优化。

分析阶段用了大概一周的时间，之后便开始了算法的实现。由于我对C++语言的熟悉程度比Java高，所以考虑先用C++实现算法。分析阶段产生的思路比较清晰，所以算法实现没有耗费太多时间。

然而在用准备的测试用例测试算法程序的功能和性能时，发现当像素矩阵比较庞大的时候，算法的耗时很长，甚至一些用例跑到一半程序就崩溃了，所以我开始考虑算法优化。

重新分析置乱算法的算法步骤后发现这个算法最复杂的部分就是混沌实值序列的排序与置乱地址集合的生成，而且这两个步骤还是针对单行或单列像素的，并不是针对整个像素矩阵的，这使得图像置乱的时间大多消耗在这两个步骤上面。由于当时的我正好在复习各种排序算法，对各个排序算法的特性比较熟悉，于是就想到了选择排序算法每一趟排序都使得当前元素是其之后所有元素中最小的这个特性，正好能够将置乱地址的生成结合在排序的过程中去，这样将两个步骤从串行变成了并发，从而完成对算法的优化。

之后为了实现基于Android平台的图像置乱软件，我又将C++实现的优化后的代码移植到了Java中去，所幸C++和Java的很多语法都相同，移植过程并没有增加多少工作量。

Android软件实现的阶段工作量比较大，由于我对Android开发接触的很少，所以基本都是从零开始。不过在查阅了很多关于Android开发的书籍和资料和写了一些练手的Demo后，我才发现其实要实现图像置乱并不需要用到很多高级的Android开发技术，只需要Android中应用程序框架层中的Activity和Intent组件即可实现主体功能，使用Activity组件可以完成前端展现和按钮事件监听，使用Intent组件可以完成界面跳转和消息传递，本文所开发出的图像置乱软件正是基于这些功能实现的。明白了这些之后，开发图像置乱软件就比较轻松了。

6.2展望

基于排序变换和混沌Logistic映射的图像置乱算法仅仅是众多图像置乱算法中的一种，如同一叶之于茂林，一粟之于沧海。这个算法本身的特性导致本文虽然对其做了相应的优化，仍然不能从根本上提升这个算法的效率。该算法的核心在于通过地址映射记录像素置乱位置，而正好整个算法的复杂度也集中在这一点上。

但我相信在这个注重信息安全的时代，肯定会出现或者已经出现了更优秀的图像置乱算法，这种算法不需要每一行每一列像素进行复杂的加密操作，而只需遍历像素矩阵一次，通过特定的约束对像素矩阵进行处理实现加密，之后再通过同样的约束将像素矩阵还原从而实现解密。这样从复杂度上看这种算法仅仅是O(N)级别的，这对于效率的提示是巨大的，时间复杂度能优化到这个量级已经是最优的了。

然而，对于图像置乱来说，最重要的则是算法的安全性、抗干扰能力等其他能够提升对图像信息保护能力的特性，这才是图像置乱技术的初衷。所以一个更优秀的图像算法不仅仅具有优秀的时间复杂度表现，更需要有优秀的图像信息保护能力，希望图像置乱技术将来能够发展的更远，更高效，更安全。

参考文献

[1] 199IT.一分钟互联网产生的数据[EB/OL].http://www.ithome.com/html/it/82766.htm，2014

[2] 沈世镒编著.近代密码学[M].广西:广西师范大学出版社，1998

[3] 卢开澄编著.计算机密码学——计算机网络中的数据保密与安全(第二版)[M].北京:清华大学出版社，1998

[4] ArtoSalomaa.公钥密码学[M].丁存生，单炜娟译.北京:国防工业出版社，1998

[5] 王育民,何大可编著.保密学——基础与应用.陕西:西安电子科技大学出版社，1990

[6] 谭永杰，张文娟，崔海花著.图像置乱技术综述[J].河南:周口师范学院学报，2010

[7] 刘婷，闵乐泉著.基于Arnold变换的图像置乱密码的安全性分析[J].武汉:武汉大学学报，2011年07期

[8] 陈巧林，廖晓峰，陈勇等.改进的基于混沌序列的幻方变换图像加密[J].计算机工程与应用，2005，(35):267-275+334

[9] 王武军.基于Hilbert变换的数字图像置乱新算法[J].信息网络安全，2012，(03):71-73

[10] 刘向东，焉德军，朱志良.基于排序变换的混沌图像置乱算法[J].中国图像图形学报，2005年5月第10卷第5期

[11] MarkAllenWeiss.数据结构与算法分析——C语言描述[M].冯舜玺译.北京:机械工业出版社，2004，(2):11-25

[12] 张乃孝，陈光等著.算法与数据结构——C语言描述（第三版）[M].北京:高等教育出版社，2013，(8):252-262

[13] 知行.Android系统架构剖析[EB/OL].http://www.cnblogs.com/zhixing/archive/2012/10/31/2748908.html

致 谢

本文能够顺利完成，离不开平萍老师地细心教导，老师对于算法的见解和分析总是非常深刻到位，而且平萍老师对待学生认真负责的态度也让我倍受感动，没有平萍老师在我质疑原算法的效率不高时给我的鼓励，可能我也没有足够的信心能完成对算法的优化，能够选中平萍老师带的毕业设计我感到非常幸运。在此，我要对导师表示诚挚的敬意和衷心的感谢。

四年的大学生活也离不开其他老师的支持与帮助，没有各位老师们的付出，我也无法打下扎实的编程基础。是老师们的教导让我开阔了视野，夯实了基础，能够走向社会，能对社会贡献自己的力量。身边的同学们也给了我一个和谐友爱的学习生活氛围，使我愉快的度过了人生中青春而美好的阶段，给我留下了无法忘怀的深刻记忆。

除此之外，感谢百忙之中抽出时间对我的论文进行评阅的老师们，你们辛苦了！

最后，再次感谢四年来支持我、关心我的老师、同学、亲人和朋友们！