
重庆大学课程报告



题 目 隐藏式数字水印实现

学 院 计算机学院

专业班级 信息安全 02 班、信息安全 01 班

年 级 2018 级

姓 名 王浩、李涵威

学 号 20184347、20181656

完成时间 2021 年 4 月 9 日

成 绩

指导教师 向涛

重庆大学教务处制

目录

1	数字水印简介.....	4
2	数字水印算法.....	4
2.1	LSB	4
2.1.1	文本描述.....	4
2.1.2	示例展示.....	4
2.1.3	核心代码.....	5
2.2	DCT.....	5
2.2.1	文本描述.....	5
2.2.2	示例展示.....	5
2.2.3	核心代码.....	6
2.3	DWT.....	7
2.3.1	文本描述.....	7
2.3.2	示例展示.....	7
2.3.3	核心代码.....	7
2.4	LSB+Arnold.....	8
2.4.1	文本描述.....	8
2.4.2	示例展示.....	9
2.4.3	对 LSB 算法的增强	9
2.4.3	核心代码.....	10
3	API 实现	10
3.1	技术架构.....	10
3.2	API 文档	11
3.2.1	水印图获取.....	11
3.2.2	嵌入水印	11
3.2.2	提取水印	12
4	Web 应用实现和项目部署.....	12
5	总结.....	13

隐藏式数字水印实现

摘要：数字水印是指将特定的信息嵌入数字信号中，数字信号可能是音频、图片或是视频等，在著作权保护、信息影藏等方面有着重要作用。现有的数字水印技术可以分为浮现式和隐藏式两大类，本项目主要针对后者。在本项目中作者实现了 LSB、DCT、DWT 三种隐藏式图片水印算法，并在传统 LSB 算法的基础上使用 Arnold 图片置乱技术进行改进以弥补 LSB 不能抵御裁剪和遮挡攻击的不足。最终作者将这些算法编写为 REST API，在此基础上实现了一个嵌入和提取数字水印的 web 应用程序。

关键字：数字水印；LSB；DCT；DWT

1 数字水印简介

数字水印（Digital Watermarking）技术是将一些标识信息(即数字水印)直接嵌入数字载体当中(包括图片、文档等)或是间接表示(修改特定区域的结构)，且不影响原载体的使用价值，也不容易被探知和再次修改。但可以被生产方识别和辨认。通过这些隐藏在载体中的信息，可以达到确认内容创建者、购买者、传送隐秘信息或者判断载体是否被篡改等目的。数字水印是信息隐藏技术的一个重要研究方向。 数字水印是实现版权保护的有效办法，是信息隐藏技术研究领域的重要分支。

2 数字水印算法

2.1 LSB

2.1.1 文本描述

一种空域上的数字水印算法。具体的做法为将黑白水印信息存储在图片 RGB 通道的 B 通道的最后一位。用 0 和 1 分别表示黑白存放，因改变的位数处于最后一位，这种微小的像素变化在正常情况下人眼是无法察觉的。例如 RGB (255,255,255)，如果在最后 B 通道的最后一位存放比特 0，那么该像素变为 RGB(255,255,254)，如下

R	G	B
11111111	11111111	11111110

最终水印信息的提取也只需要将 RGB 图片 B 通道的最低有效位全部提取出来作为一个二值图像即可。

2.1.2 示例展示

原图	嵌入水印后图	提取的水印
		

2.1.3 核心代码

水印嵌入：

```
for i in range(mark.shape[0]):
    for j in range(mark.shape[1]):
        blue = pic[i, j, 2]
        # 嵌入二值水印信息
        pic[i, j, 2] = blue - blue % 2 + mark[i][j]
```

水印提取：

```
for i in range(pic.shape[0]):
    for j in range(pic.shape[1]):
        # 提取二值水印信息
        mark[i, j] = pic[i, j, 2] % 2
```

2.2 DCT

2.2.1 文本描述

离散余弦变换（DCT for Discrete Cosine Transform）是与傅里叶变换相关的一种变换，它与离散傅里叶变换类似，但是只使用实数。本项目采用 DCT 变换对图片进行处理，在这个过程中，每次变换图片里不重叠的 8*8 矩阵块，这个矩阵块包含的信息为水印的一个像素，因此根据这个像素的黑白将对这个矩阵块所有的元素进行一个微小的变化，在提取水印的时候就是根据这个细微的变化进行提取。

值得注意的是由于水印图像的每个像素保存在原图 8*8 区域的 DCT 域中，所以水印图片的大小必须不超过原图最短边的 1/8。

2.2.2 示例展示

原图	嵌入水印后图	提取的水印
		

2.2.3 核心代码

水印嵌入:

```
for i in range(row):
    for j in range(col):

BLOCK=np.float32(im_array[i*block_width:i*block_width+block_width,
j*block_width:j*block_width+block_width])
BLOCK=scipy.fft.dct(BLOCK) # 矩阵 DCT
if mark_array[i][j]==False: # 根据二值矩阵赋值给 a
    a=-1
else:
    a=1
BLOCK[:, :, 2]=BLOCK[:, :, 2]*(1+a*0.03) # 矩阵块的微小变化
BLOCK=scipy.fft.idct(BLOCK).astype(np.uint8) #逆 DCT

im_array[i*block_width:i*block_width+block_width,j*block_width:j*block_width+block_width]=BLOCK
```

水印提取:

```
for i in range(row):
    for j in range(col):

BLOCK_ORIGIN=np.float32(im_array[i*block_width:i*block_width+block
_width,j*block_width:j*block_width+block_width])

BLOCK_MARKED=np.float32(marked_array[i*block_width:i*block_width+b
lock_width,j*block_width:j*block_width+block_width])
BLOCK_ORIGIN=scipy.fft.idct(BLOCK_ORIGIN)
BLOCK_MARKED=scipy.fft.idct(BLOCK_MARKED)

bm=BLOCK_MARKED[1,1,2]
# 获取水印拟 DCT 后指定位置元素, 本质上
# 要是这个矩阵块的元素都是可以采用的
bo=BLOCK_ORIGIN[1,1,2] # 获取嵌入后水印逆 DCT 后元素
a=bm/bo-1 # 进行差距检测
if a<0:
    decode_pic[i,j]=False
else:
    decode_pic[i,j]=True
```

2.3 DWT

2.3.1 文本描述

DWT 即离散小波变换，是与傅里叶变换一样，小波变换的基本思想是将信号展开成簇基函数之加权和，即用一簇函数来表示信号和函数。本项目采用两次 DWT 处理图片的 RGB 通道的 B 通道，取返回的结果的近似矩阵进行处理。

与前面 DCT 中对水印图片大小的限制类似，在本项目的实现中 DWT 嵌入的水印大小不能超过原图最短边的 1/4。

2.3.2 示例展示

原图	嵌入水印后图	提取的水印
		

2.3.3 核心代码

水印嵌入：

```
idx = random.sample(range(ca2_size), mark_copy_size)
ca2_flatten = ca2.flatten()
mark_copy_flatten = mark_copy.flatten()
for i in range(mark_copy_size):
    c = ca2_flatten[idx[i]]
    z = c % mark_w
    # 水印对应二进制为 1
    if mark_copy_flatten[i]:
        # 将 z 替换成 c-k*mark_w，就能发现水印为 1
        # 时 f 模 mark_w 为 1/4mark_w
        if z < mark_w/4:
            f = c-mark_w/4-z
        else:
            f = c+mark_w*3/4-z
    # 水印对应的二进制位为 0
```

```

else:
    # 将 z 替换成 c-k*mark_w, 就能发现水印为 1
    # 时 f 模 mark_w 为 3/4mark_w
    if z < mark_w*3/4:
        f = c+mark_w/4-z
    else:
        f = c+mark_w*5/4-z
    ca2_flatten[idx[i]] = f

```

水印提取:

```

idx = random.sample(range(ca2w_size), mark_copy_size)
res_mark = mark_copy.flatten()
# 就算是从新建立一个 zeros 矩阵初始化效果也一样
ca2w_flatten = ca2w.flatten()
for i in range(mark_copy_size):
    c = ca2w_flatten[idx[i]]
    z = c % mark_w
    if z < mark_w/2:
        res_mark[i] = False
    else:
        res_mark[i] = True

```

2.4 LSB+Arnold

2.4.1 文本描述


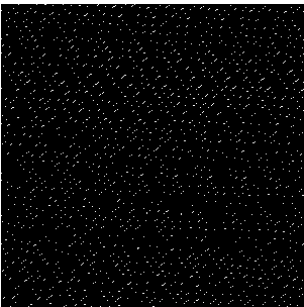
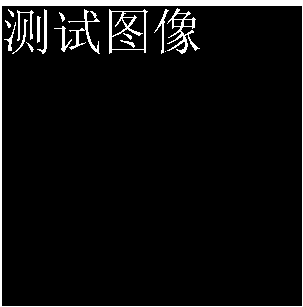
普通的 LSB 算法只是简单的水印图像的每个像素保存在原图的最低有效位, 这种水印既无法做到保密, 也无法抵抗裁剪和部分遮挡攻击。一种简单的改进就是将水印图片置乱之后再嵌入原图, 这样部分遮挡后提取出来的水印仍然能和嵌入的水印大体相似。常见的图片置乱算法骑士巡游、Arnold 和仿射变换等, 本项目主要实现了 Arnold 置乱算法。

LSB 已经介绍过了, 这里介绍 Arnold 算法, 它就是像素的位置转化, 具体公式如下:

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} 1 & b \\ a & 1+ab \end{bmatrix} \begin{bmatrix} x_{ori} \\ y_{ori} \end{bmatrix} \bmod(N)$$

值得注意的是, 该算法只能作用在正方形图片中, 公式中的 N 即表示图片的大小。其中 a 和 b 是两个控制参数, 可以看作加密算法中的密钥, 在不同的 a 和 b 控制下置乱效果是不同的。并且为了达到完全混乱的效果, 上述置乱一般会进行多轮。在本项目实现中 a 和 b 固定取 1234, 置乱轮次为 5 次。

2.4.2 示例展示

置乱前	置乱后	恢复后
		

可以看到置乱后基本上是达到了完全混乱的效果，并且恢复出来的图片和置乱前的图片完全一致。

2.4.3 对 LSB 算法的增强

没有使用置乱算法的 LSB 水印在部分遮挡时表现如下：

原图	嵌入水印后进行遮盖的图	提取的水印
		

使用 Arnold 置乱后的 LSB 水印在部分遮挡后的表现如下：

原图	嵌入水印后进行遮盖的图	提取的水印
		

2.4.3 核心代码

置乱:

```
for _ in range(shuffle_times):
    for ori_x in range(h):
        for ori_y in range(w):
            new_x = (1*ori_x + b*ori_y)% N
            new_y = (a*ori_x + (a*b+1)*ori_y) % N
            if mode == '1':
                arnold_image[new_x, new_y] = image[ori_x, ori_y]
            else:
                arnold_image[new_x, new_y, :] = image[ori_x,
ori_y, :]
```

恢复:

```
for _ in range(shuffle_times):
    for ori_x in range(h):
        for ori_y in range(w):
            new_x = ((a*b+1)*ori_x + (-b)* ori_y)% N
            new_y = ((-a)*ori_x + ori_y) % N
            if mode == '1':
                decode_image[new_x, new_y] = image[ori_x, ori_y]
            else:
                decode_image[new_x, new_y, :] = image[ori_x,
ori_y, :]
```

3 API 实现

3.1 技术架构

由于数字水印算法采用 Python 进行实现，考虑到本项目所需要的 API 并不复杂，所以选择了轻量的 Flask 框架进行实现。结合前端需求，主要实现了三个 API:

- 1) 水印图获取
- 2) 嵌入水印
- 3) 提取水印

3.2 API 文档

3.2.1 水印图获取

URL: /api/getmark

Type: GET

Path Parameters:

名称	类型	描述
fontname	string	文字字体（SimSun、SimHei、SimKai、Microsoft YaHei）
fontsize	number	文字字体大小
text	string	水印图片上的文字
size	number	水印图片大小（像素）

Response:

```
{
  'status': 'ok',
  'image': b64_img
}
```

3.2.2 嵌入水印

URL: /api/embed

Type: POST

Form Data:

名称	类型	描述
image	file	需要嵌入水印的原图
algorithm	string	水印算法（LSB、DCT、DWT）
fontname	string	文字字体（SimSun、SimHei、SimKai、Microsoft YaHei）
fontsize	number	文字字体大小
text	string	水印图片上的文字
size	number	水印图片大小（像素）

Response:

```
{
  'status': 'ok',
}
```

```
{
  'mark': mark_url,
  'marked_pic': marked_pic_url
}
```

3.2.2 提取水印

URL: /api/extract

Type: POST

Form Data:

名称	类型	描述
algorithm	string	水印算法（LSB、DCT、DWT）
marked_pic	file	含水印的图片
pic	file	（可选）不含水印的原图
mark	file	（可选）水印图片

Response:

```
{
  'status': 'ok',
  'mark': b64_img
}
```

4 Web 应用实现和项目部署

Web 应用主要采用了 UIKit 和 jQuery 框架，最终效果如下图所示。

印

Digital watermarking

[嵌入水印](#)[提取水印](#)[样图下载](#)[算法简介](#)[关于](#)[仓库](#)

嵌入水印



选择图片

↓ LSB

↓ 微软雅黑

↓ 20px

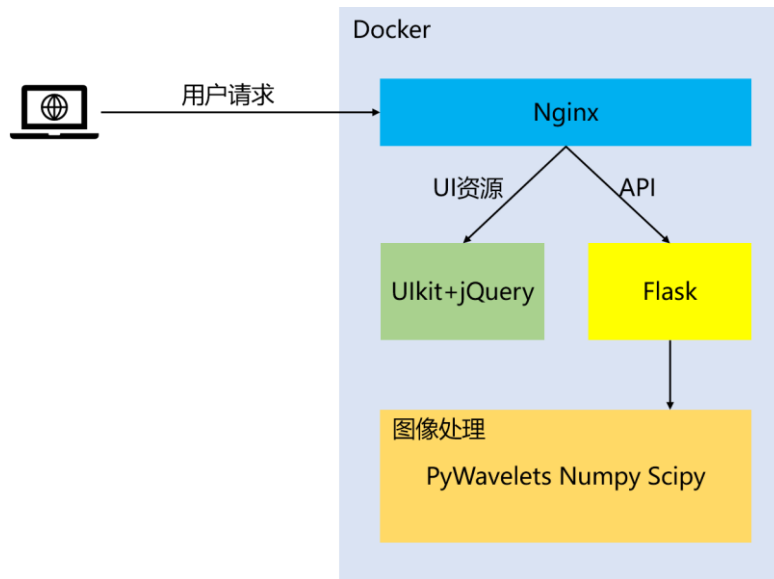
水印大小 (像素)

水印文本

水印预览

嵌入水印

在前后端都实现后本项目采用了 Docker 进行部署，前端部分采用 Nginx 部署为静态网站，后端 API 采用 Nginx 反向代理解决跨域问题。最终部署架构如下图所示。



5 总结

本项目实现了 LSB、DCT、DWT 数字水印算法，并将其封装为 REST API，在此基础上实现了一个在线添加和提取数字水印的 Web 应用程序。该程序具有友好的前端界面和操作逻辑，可以用于展示和体验数字水印技术。

但是由于时间原因和参与人员能力有限，项目中不免有一些疏漏和不足。欢迎提出建议和意见。

项目地址: <https://wm.iamwh.cn/>

项目仓库: <https://github.com/iamwhcn/digital-watermark>

本项目的人员分工如下:

王浩: 置乱算法实现、后端接口开发、前端界面开发、文档撰写、项目部署

李涵威: 水印算法开发、前端界面开发、查找文献、前后端接口调试、文档撰写

参考文献

- [1] 尹浩, 林闯, 邱锋, 等. 数字水印技术综述[J]. 计算机研究与发展, 2005, 42(7): 1093.
- [2] Jiansheng M, Sukang L, Xiaomei T. A digital watermarking algorithm based on DCT and DWT[C]//Proceedings. The 2009 International Symposium on Web Information Systems and Applications (WISA 2009). Academy publisher, 2009: 104.