信息隐藏实验一 LSB隐写和RS分析实现

原创

red1y 于 2022-03-28 20:29:57 发布 164 收藏 1

分类专栏： 网安实验 文章标签： c语言 服务器 安全 算法

一、LSB隐写

1. 简介

- 隐写对象：*bmp* 图像，图像内容为原始像素值
- 隐写方式：将秘密信息转换为二进制比特流，嵌入像素值的最低位中(对图像的影响最小)
- 隐写规则：嵌入0，将对应像素值最低位置为0；嵌入1则置为1
- 提取方式：提取像素值最低位，重新转换为对应数据编码

2. 实现

读取原始图像：图像头+图像内容，我们可以进行隐写的部分为内容部分

```
void readHeader(FILE *image, UCHAR header[]) {
    fread(header, 1, 1078, image); // bmp文件头长度为1078(带调色板)
}

void readImage(FILE *image, UCHAR raw[DEGREE][DEGREE]) {
    int i; // 读取像素值，512*512
    for(i = 0; i < DEGREE; i++) fread(raw + i, sizeof(UCHAR), DEGREE, image);
}
```

嵌入秘密信息：方便起见，秘密信息为随机比特流；此处加上了嵌入率的设定

```
// 嵌入率为 1/rate
void randomCipherWrite(UCHAR raw[DEGREE][DEGREE], int rate) {
    int i, j;
    for(i = 0; i < DEGREE; i++) {
        for(j = 0; j < DEGREE; j++) {
            if(!(getCipher(1) % rate)) { // 生成随机数, 若模rate为0则进行一次嵌入
                if(!getCipher(0)) { // 嵌入0: 2i+1 -> 2i
                    if(raw[i][j] % 2) raw[i][j]--;
                } else { // 嵌入1: 2i -> 2i+1
                    if(!(raw[i][j] % 2)) raw[i][j]++;
                }
            }
        }
    }
}
```

生成嵌入秘密信息的图像，将原始 *bmp* 头和修改后的像素值写入文件，略

二、RS分析

1. 简介

分析对象：可能进行了 **LSB** 隐写的 *bmp* 图像

概念说明：

- 正翻转：$2i \leftrightarrow 2i+1$
- 负翻转：$2i \leftrightarrow 2i-1$
- 零翻转：像素值不变
- 非负翻转：随机进行正翻转或零翻转
  - 对图像进行 **LSB** 隐写相当于进行了一次非负翻转
- 非正翻转：随机进行负翻转或零翻转
- 相关性(混乱度)：将像素值矩阵进行 **Z** 形排序，得到像素序列，通过以下方式计算相关性

$$r=$$

分析原理：

对于原始图像，单独进行一次 *非负翻转* 或一次 *非正翻转*，对 r 的影响是相近的

对于进行了一次 *非负翻转* 的图像，若

再进行一次 *非负翻转*，对 r 的影响很小；就是来回了一遍

再进行一次 *非正翻转*，会显著的影响 r；考虑下例

$$2i \rightarrow 2i+1=2(i+1)$$

图像分块：

- 为了更好的衡量进行翻转后的图像的相关形变化，将原始图像分为多个 8×8 矩阵
- 计算每个矩阵的相关性，可以得到一个相关性序列
- 通过统计翻转前后相关性增加/减少的矩阵块数(比例)，量化翻转带来的影响，记
  - 相关性增加的比例，记为R ；
  - 相关性减少的比例，记为S ；

分析步骤：

- 计算得到待分析图像的相关性序列
- 对待分析图像分别进行非负、非正翻转，分别得到其对应的相关性序列
- 统计
  - 非负翻转后，相关性增加的比例 R
  - 非正反转后，相关性增加的比例 R
- 根据上面的原理分析，可知
  - 若待分析图像为原始图像，则 $R \approx R$
  - 若待分析图像进行了 **LSB** 隐写(即进行了一次非负翻转)，则

    $< R$

2. 实现

读取待分析图像，同上

对待分析图像进行非负翻转(随机正翻转或零翻转)

```
void pos_reverse(UCHAR pos_raw[DEGREE][DEGREE]) { // 非负翻转
    int i, j;
    for(i = 0; i < DEGREE; i++) {
        for(j = 0; j < DEGREE; j++) {
            if(!getCipher(0)) { // 进行正翻转
                if(!getCipher(0)) { // 嵌入0: 2i+1 -> 2i
                    if(pos_raw[i][j] % 2) pos_raw[i][j]--;
                } else { // 嵌入1: 2i -> 2i + 1
                    if(!(pos_raw[i][j] % 2)) pos_raw[i][j]++;
                }
            } else { // 进行0翻转
                /* empty */
            }
        }
    }
}
```

对待分析图像进行非正翻转(随机负翻转或零反转)

```
void neg_reverse(UCHAR neg_raw[DEGREE][DEGREE]) { // 非正翻转
    int i, j;
    for(i = 0; i < DEGREE; i++) {
        for(j = 0; j < DEGREE; j++) {
            if(!getCipher(0)) { // 进行负翻转
                if(!getCipher(0)) { // 嵌入0: 2i-1 -> 2i
                    if(neg_raw[i][j] % 2) neg_raw[i][j]++;
                } else { // 嵌入1: 2i -> 2i-1
                    if(!(neg_raw[i][j] % 2)) neg_raw[i][j]--;
                }
            } else { // 进行0翻转
                /* empty */
            }
        }
    }
}
```

计算待分析图像、非负翻转图像、非正翻转图像的相关性序列

```
// 计算相关性序列
void getRelevance(UCHAR raw[DEGREE][DEGREE], int relv[]) {

    UCHAR pixel[BLOCKD * BLOCKD] = {0}; // Z形排序后的像素值

    int i, j, bcnt = 0;
    for(i = 0; i < DEGREE; i += 8) { // 分块
        for(j = 0; j < DEGREE; j += 8) { // 分块
            getBlockZIndex(raw, i, j, pixel); // 得到分块矩阵的Z形排序
            relv[bcnt++] = getBlockRelevance(pixel); // 计算改矩阵的相关性
        }
    }
}
// 计算矩阵Z形排序
void getBlockZIndex(UCHAR raw[DEGREE][DEGREE], int sx, int sy, UCHAR pixel[]) {
    int index = 0, x = 0, y = 0, i;
```

```
    int index = 0, x = 0, y = 0, i;
    while(index < (BLOCKD * BLOCKD + BLOCKD) / 2) {
        for(;y >= 0; x++, y--) pixel[index++] = raw[sx + x][sy + y]; y++; // 向坐下走
        for(;x >= 0; x--, y++) pixel[index++] = raw[sx + x][sy + y]; x++; // 向右上走
    }
    x++; y--;
    while(index < BLOCKD * BLOCKD) {
        for(;x < BLOCKD; x++, y--) pixel[index++] = raw[sx + x][sy + y]; x--, y+=2; // 向坐下走
        for(;y < BLOCKD; x--, y++) pixel[index++] = raw[sx + x][sy + y]; y--; x+=2; // 向右上走
    }
}
// 计算像素序列相关性
int getBlockRelevance(UCHAR pixel[]) {
    int relevance = 0, i;
    for(i = 1; i < BLOCKD * BLOCKD; i++) {
        relevance += abs(pixel[i] - pixel[i - 1]);
    }
    return relevance;
}
```

比较相关性变化，计算得到 R 或 S

```
/*
 * @param raw_relv 待分析图像的原始相关性序列
 * @param mod_relv 翻转图像的相关性序列
 **/
double getRm(int raw_relv[], int mod_relv[]) {
    int i, insb = 0;
    for(i = 0; i < BLOCK_CNT; i++) {
        if(raw_relv[i] < mod_relv[i]) insb++;
    }
    return insb * 1.0 / BLOCK_CNT;
}
```

输出比较 R ， R

3. 运行验证

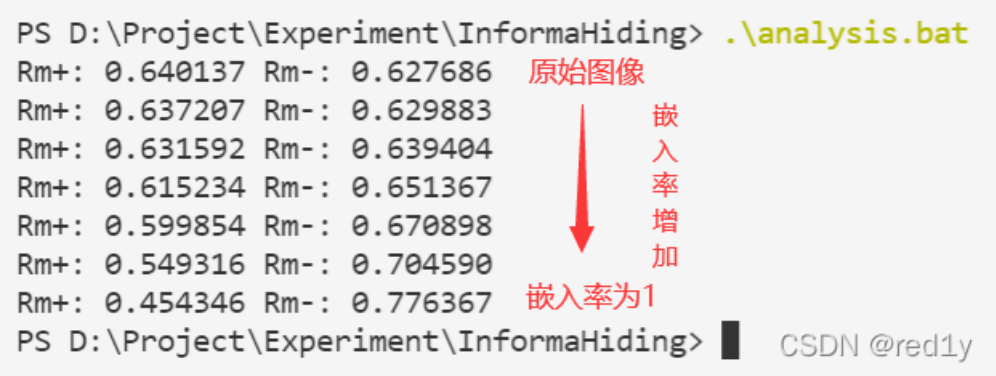
对原始图像分别进行嵌入率为 1，1/2，1/4，1/8，1/16，1/32的 **LSB** 隐写

```
@echo off
.\write .\image\lena_gray.bmp .\image\1.bmp 1
.\write .\image\lena_gray.bmp .\image\2.bmp 2
.\write .\image\lena_gray.bmp .\image\4.bmp 4
.\write .\image\lena_gray.bmp .\image\8.bmp 8
.\write .\image\lena_gray.bmp .\image\16.bmp 16
.\write .\image\lena_gray.bmp .\image\32.bmp 32
```

对原始图像及以上图像进行 RS 分析

```
@echo off
.\analysis .\image\lena_gray.bmp
.\analysis .\image\32.bmp
.\analysis .\image\16.bmp
.\analysis .\image\8.bmp
.\analysis .\image\4.bmp
.\analysis .\image\2.bmp
.\analysis .\image\1.bmp
```

查看运行结果，可以看出，随着嵌入率的增加，$R_{}$ 和 $R_{}$ 的差值越来越大



参考源码