

武汉大学国家网络安全学院

信息隐藏实验报告

学 号 2021302141097

姓 名 李 宇

实验名称 安全隐写对抗技术

指导教师 王丽娜，任延珍

一、实验名称: 安全隐写对抗技术

二、实验目的:

- 1) 掌握基本的隐写分析技术, 如卡方分析、RS 分析等
- 2) 掌握 LSB 隐写的改进版—LSBM 的原理与实现
- 3) 掌握常用的安全隐写算法, 如 Jsteg、F3、F4、F5 隐写等
- 4) 熟悉 Matlab 编程过程, 能够代码实现安全隐写算法
- 5) 对各类所实现的隐写算法进行安全性分析, 观察其抗检测能力的提升情况
- 6) 比较不同隐写算法隐蔽性之间的差异

三、实验原理:

本次实验共分为 4 个小实验, 分别对应的是 1.1, 1.2, 2.1, 2.2 (接下来统称它们为实验一、二……)。由于不同实验所涉及的原理不同, 接下来对各实验原理进行分析。

(一) 实验一: **LSBM** 隐写算法的消息嵌入和提取

LSBM 主要是利用 LSB(最不重要位) 的思想, 对读取的图片进行选取图像载体像素, 修改其色素中的最低位数值, 来实现信息隐藏。

而 LSB 算法存在较强的值对效应: 偶数 +1 \rightarrow 奇数, 奇数 -1 \rightarrow 偶数。导致灰度值在 $2i$ 和 $2i+1$ 的像素趋于一致数量, 这使得 LSB 隐写很容易被图像灰度直方图检测出来, 即传统的 LSB 抗隐写分析能力较弱。

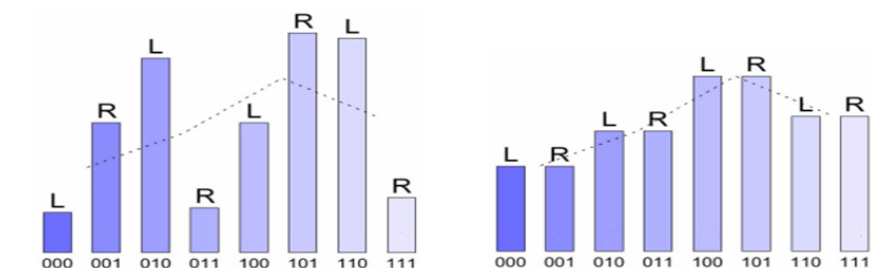


图 1: LSB 值对效应说明

为了克服值对效应, 当载体图像 LSB 和消息比特相不同时, 将像素值随机加

减 1。同时由于灰度图像的像素值在 0-255 之间，为了避免溢出，当载体像素值为 0 时，则加 1；像素值为 255 时，则减 1；其他需要加减 1 的则随机进行。这样就避免了偶数只能加 1 变成奇数和奇数只能减 1 变成偶数的值对效应。

(二) 实验二：卡方分析和 RS 分析

卡方分析是一种用于隐写分析的技术，其原理基于比较载体图像和隐写图像的灰度直方图。其核心思想如下：

1. 设图像中灰度值为 j 的像素数为 h_j ，其中 $0 \leq j \leq 255$ 。在原始图像中，相邻灰度值的像素块数目一般差别很大

2. 在隐写信息隐藏中，秘密信息在嵌入之前往往经过加密，可以看作是 0、1 随机分布的比特流，而且值为 0 与 1 的可能性都是 1/2。如果秘密信息完全替代载体图像的最低位，那么伪装对象相邻灰度值的像素块数目将会比较接近

3. 定量分析载体图像最低位完全嵌入秘密信息的情况。嵌入信息会改变直方图的分布，由差别很大变得近似相等，但是却不会改变 $h_{2i} + h_{2i+1}$ 的值，因为样值要么不改变，要么就在 h_{2i} 和 h_{2i+1} 之间改变

统计样本的实际观测值与理论推断值之间的偏离程度，决定卡方值的大小。如果卡方值越大，二者偏差程度越大；反之，二者偏差越小；若两个值完全相等时，卡方值就为 0，表明理论值完全符合。

同时卡方分析源自于概率论中的内容，其本身也有一套严密的数学推导过程；其数学推导在隐写分析的具体表现如下：

设图像中灰度值为 j 的像素数为 h_j ，在嵌入过程中只存在 $h_{2i} \leftrightarrow h_{2i+1}$ 而不存在 $h_{2i} \leftrightarrow h_{2i-1}$ 的变换，且不会改变 $h_{2i} + h_{2i+1}$ 的值，嵌入信息中 0、1 的分布较为均匀，所以会导致 h_{2i} 和 h_{2i+1} 的值趋于一致。利用这个性质，我们可以分析图像是否被隐写。

定义 $h_{2i}^* = \frac{h_{2i} + h_{2i+1}}{2}$ ，当 h_{2i}^* 很大的时候，根据中心极限定理：

$$\frac{h_{2i} - h_{2i}^*}{\sqrt{h_{2i}^*}} \rightarrow N(0, 1) \quad (1)$$

所以, χ^2 服从卡方分布

$$\chi^2 = \sum_{i=1}^k \frac{(h_{2i} - h_{2i}^*)^2}{h_{2i}^*} \quad (2)$$

结合卡方分布的密度计算函数计算载体被隐写的可能性为:

$$p = 1 - \frac{1}{2^{\frac{k-1}{2}} \Gamma(\frac{k-1}{2})} \int_0^{\chi^2} e^{-\frac{t}{2}} t^{\frac{k-1}{2}-1} dt \quad (3)$$

RS 分析是另一种针对采用伪随机 LSB 嵌入算法的检测方法。RS 分析不但能检测出图像是否隐藏信息, 而且还能比拟准确地估算出隐藏的信息长度。

RS 隐写分析算法考虑图像各个位平面之间具有一定的非线性相关性, 当利用 LSB 隐写算法隐藏秘密信息后, 这种相关性就会破坏。只要能找出衡量这一相关性的方法, 并对隐藏秘密信息前后的情况加以比照, 就有可能设计出隐写分析方法。

RS 隐写分析方法的理论核心是: 任何经过 LSB 隐写的图像, 其最低比特位分布满足随机性, 即 0、1 的取值概率均为 1/2, 而未经过隐写的图像不存在此特性。

RS 的具体实现流程如下所示:

我们用如下的公式来衡量一个图像的平滑度, 平滑度函数体现了一个图像相邻像素的平滑程度, LSB 嵌入会给图像增加噪声, 一般会导致 f 的增大。

$$f(x_1, x_2, \dots, x_n) = \sum_{i=1}^{n-1} |x_{i+1} - x_i| \quad (4)$$

我们还定义三个 LSB 嵌入操作函数 $F(x)$, 分别是:

- 交换函数 F_1 : $2n \leftrightarrow 2n + 1$
- 偏移函数 F_{-1} : $2n \leftrightarrow 2n - 1$
- 恒等函数 F_0 : $F_0(x) = x$

在 LSB 中我们只用到了 F_1 函数, 我们进一步看这个函数实际上是定义了二进制末位 0 到 1 或者 1 到 0 的交换。而 F_{-1} 函数在相邻数对变换的过程中已经涉及到了二进制数字的倒数第二位。我们把图像分为若干个像素组 $G(g_1, g_2, \dots, g_n)$, 设掩码算子 $M(m_1, m_2, \dots, m_n)$, 其中 m_i 取 1、-1 或 0, 且 $-M(-m_1, -m_2, \dots, -m_n)$ 。定义操作如下

$$F_M(G) = (F_{m_1}(g_1), F_{m_2}(g_2), \dots, F_{m_n}(g_n)) \quad (5)$$

把图像像素组 G 进行 F 操作，定义如下三个组：

- **Regular:** $f(F(G)) > F(G)$ ，也就是说对 G 中的元素进行变换之后增大了元素之间的混乱程度， R 表示混乱度增加的块的比例。

- **Singular:** $f(F(G)) < F(G)$ ，也就是说对 G 正的元素进行变换之后减小了元素之间的混乱程度， S 表示混乱度减少的块的比例。

- **Unusable:** $f(F(G)) = F(G)$ ，也就是对 G 中的元素进行变换之后元素之间的差别程度几乎不变， G 表示混乱度基本不变的块的比例。

再定义 R_M 、 R_{-M} 、 S_M 和 S_{-M} ，其中 R_M 表示在 F_M 作用下 **Regular** 占有所有像素组的比例，其他的定义类似。对于一个原始不包含隐藏信息的载体来说具有下面的规律：

$$R_M \approx R_{-M} > S_M \approx S_{-M} \quad (6)$$

但是当在载体中嵌入了秘密信息的话，就会有下面的式子成立：

$$R_{-M} - S_{-M} > R_M - S_M \quad (7)$$

RS 分析方法的核心是由于 **LSB** 隐写仅用到了 F_1 变换，而没有用到 F_{-1} 变换，所以当用 F_1 变换或 F_{-1} 变换去处理隐写图像时，参数呈现了不对称性，用这种不对称性可以进一步估计出隐写率。

若隐写率是 p ，那么前面计算的一组值是在 $\frac{p}{2}$ 的像素经过 F_1 变换情况下得到的。现在将待检测图像所有像素都经过一次 F_1 变换，那么相对于原始图像有 $1 - \frac{p}{2}$ 的像素，计算此时的 R_M 、 R_{-M} 、 S_M 和 S_{-M} 。将这两组四对统计量连线并相交得到隐写率是 p 。

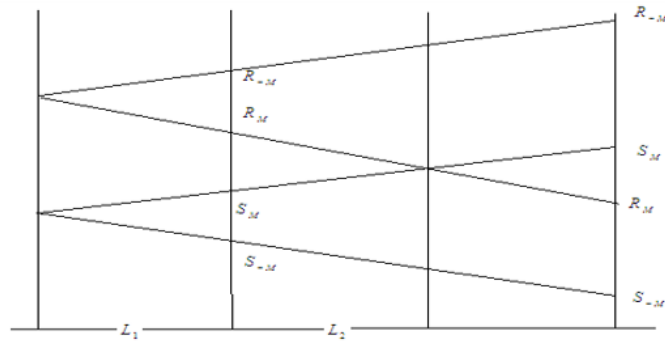


图 2: RS 分析示意图

(三) 实验三：F4 隐写算法的消息嵌入和提取

由于 F4 隐写算法是基于 F3 隐写算法的改进，而 F3 隐写算法又是基于 Jsteg 算法的改进，故在介绍 F4 算法前需要首先清楚 Jsteg 与 F3 隐写算法的原理。

1.Jsteg 隐写算法

Jsteg 是一种 LSB 隐写算法，它可以将秘密信息嵌入到图像的 DCT 系数中。该算法的核心思想是将原始图像的 AC 系数中最低的一个位平面“替换”为要隐藏的秘密信息。具体来说，Jsteg 隐写方法将秘密信息嵌入在量化后的 DCT 系数的 LSB 上，原始值为 -1、0、+1 的 DCT 系数例外。提取时，也只是将含密图像中不等于 -1、0、+1 的量化 DCT 系数的 LSB 取出。

* 不使用 0 的原因：DCT 系数中 0 的比例最大 (一般在 60% 以上)，而压缩编码是利用大量连续的零实现的

* 不使用 1 和 -1 的原因：系数中的 1 或 -1 若修改为 0，接收端就无法区别是未使用的 0 还是嵌入后得到的 0

Jsteg 算法的优点是嵌入容量大、简单易实现。然而，它也存在“值对现象”，容易被卡方分析方法分析出来。

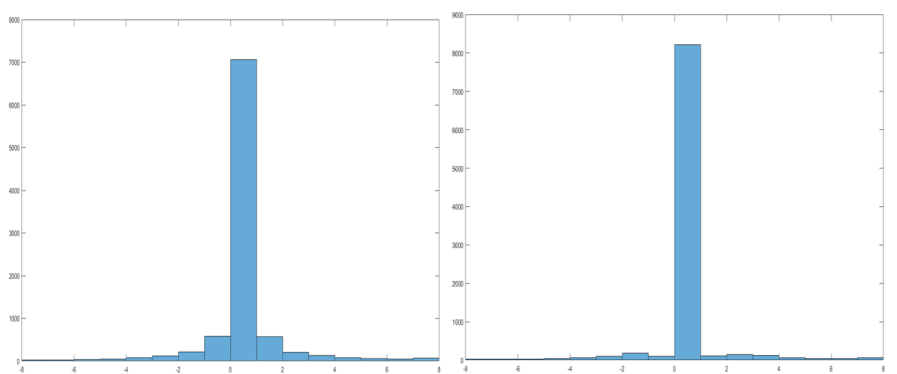


图 3: Jsteg “值对现象”说明图

2.F3 隐写算法

为了解决 Jsteg 存在的“值对现象”，F3 隐写算法改进了 JSteg 中的替换规则，避免了值对效应。同时由于值为 -1，0，+1 的 DCT 系数再 JPEG 图像中占比很大，F3 隐写利用了原始值为 +1 和 -1 的 DCT 系数，提高了嵌入率。

F3 隐写的具体替换规则如下所示：

- * 跳过 0
- * 若 AC 系数为 $2i$ ，秘密比特为 0，该系数不变
- * 若 AC 系数为 $2i$ ，秘密比特为 1，且 $i \geq 0$ ，该系数变为 $2i-1$
- * 若 AC 系数为 $2i$ ，秘密比特为 1，且 $i < 0$ ，该系数变为 $2i+1$
- * 若 AC 系数为 $2i+1$ ，秘密比特为 1，该系数不变
- * 若 AC 系数为 $2i+1$ ，秘密比特为 0，且 $i \geq 0$ ，该系数变为 $2i$
- * 若 AC 系数为 $2i+1$ ，秘密比特为 0，且 $i < 0$ ，该系数变为 $2i+2$
- * 若 AC 系数变为 0，则该秘密比特需要重新隐藏

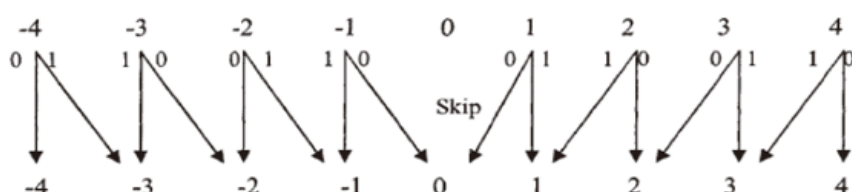


图 4: F3 隐写替换规则说明图

3.F4 隐写算法

在应用 F3 替换算法时，由于载体图像中绝对值为 1 的数值较多，当其被修改为 0 时，嵌入算法会继续嵌入直到找到一个偶数值，或将一个奇数改为偶数。即绝对值为 1 的数允许嵌入 1，但需要使用或制造一个偶数来嵌入 0。因此 F3 隐写的替换规则将会使得载密图像的 DCT 系数中的偶系数增多。F4 信息隐藏算法改进了 F3 信息隐藏算法中用到的替换规则，解决了这个问题。

F4 隐写的具体替换规则如下所示：

F4 替换时不考虑 AC 系数为 0 处的使用，同时，若 AC 系数变为 0，则该秘密比特需要重新隐藏。这样载密图像的 AC 系数中，除 0 外，正奇数和负偶数代表 1，正偶数和负奇数代表 0。

表 1: F4 隐写替换规则

AC 系数	i	待嵌入的比特信息	操作
$2i$	$i \geq 0$	0	系数不变
$2i$	$i \geq 0$	1	系数-1
$2i$	$i < 0$	0	系数 +1
$2i$	$i < 0$	1	系数不变
$2i + 1$	$i \geq 0$	0	系数-1
$2i + 1$	$i \geq 0$	1	系数不变
$2i + 1$	$i < 0$	0	系数不变
$2i + 1$	$i < 0$	1	系数 +1

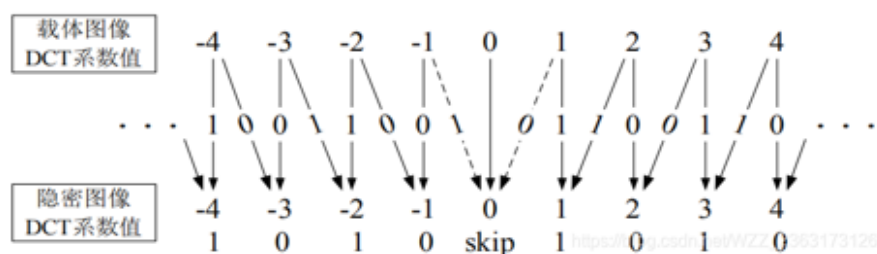


图 5: F4 隐写替换规则说明图

(四) 实验四: F5 隐写算法的消息嵌入和提取

F5 隐写算法是一种高级隐写技术，它通过把数据隐藏在音频、图像等文件中，不改变载体文件的外观和感觉，实现秘密信息的传递。与其他隐写技术不同，F5 隐写可以嵌入更大的数据，并具有较强的抗检测能力。

F5 隐写算法的主要步骤包括：

- 1) 选择载体图像：选择一张 JPEG 格式的图像作为载体图像。
- 2) 分块处理：将载体图像分为若干个 8×8 的非重叠块。
- 3) DCT 变换：对每个图像块进行 DCT 变换，得到 DCT 系数。
- 4) 嵌入数据：实施矩阵编码，修改 DCT 系数

从 F5 的主要步骤可以看到，其前几步与常规的 F3,F4 隐写算法并没有太大区别，主要改变便集中在最后一步：实施矩阵编码。F5 本质上是基于汉明码的矩阵编码隐写，通过借助矩阵编码中纠错编码的相关理论和技术，通过编码方法减少每

个载体分组中所需要的嵌入修改次数，以实现更高的嵌入效率。

汉明码 (Hamming) 是计算机组成原理与操作系统中常见的概念，借助汉明码的纠错特性，便可以实现仅修改一位数据来隐藏多位密文信息。结合汉明码的长度定义可知，若想一次修改隐藏 3 位信息 ($r=3$)，则所需的 AC 系数个数为 $n=2^r-1$ ，即需要 7 位 AC 系数作为载体，将这 7 位 AC 系数与汉明码检验矩阵相乘的结果与隐藏信息进行异或运算，便可知需要修改哪一位 AC 系数，从而达到一次修改隐藏 3 位信息。以上述设置为例，具体的流程如下所示：

首先确定 3×7 的汉明阵 H ：

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

若载体 $a = (1001011)^T$ ，秘密数据 $s = (100)^T$ ，计算得到 $c = H \cdot a = (100)^T$ ， $d = c + s = (000)^T$ ，则 $r=0$ ，因此无需对 a 进行修改， $a' = a$ ；提取时 $s = H \cdot a' = (100)^T$ 。

若载体 $a = (1001011)^T$ ，秘密数据 $s = (011)^T$ ，计算得到 $c = H \cdot a = (100)^T$ ， $d = c + s = (111)^T$ ，则 $r=7$ ，因此修改 a 中的第 7 位， $a' = (1001010)^T$ ；提取时 $s = H \cdot a' = (011)^T$ 。

因此，F5 隐写算法的实现步骤如下：

- 1) 载体 a 的获得：需要进行一次 AC 系数到 a 的映射；取出 n 个非 0 的 DCT 系数，用正奇数和负偶数代表 1、负奇数和正偶数代表 0，从而得到载体 a
- 2) 如果需要修改 (r 不为 0)，将要改动的 DCT 系数绝对值减 1，符号不变；需要检验修改后的 DCT 系数为 0
- 3) 如 DCT 系数不为 0，则返回第一步，继续下一组嵌入
- 4) 如为 DCT 系数 0，则以上操作无效，要重新取出 n 个非 0 的 DCT 系数（原来的 n 个 DCT 系数中的一个变为 0，所以新取出的 DCT 系数包含原来的 $n-1$ 个系数和一个真正新的 DCT 系数），再次嵌入这 k 比特秘密数据

四、实验内容:

四个实验所用的示例图像和隐藏文件如下所示:

所用图像: penguin.bmp

隐藏文件: information.txt、information2.txt

其中 information.txt 中待隐写的信息较多, 其目的使为了在后续能够看到明显隐写分析效果。

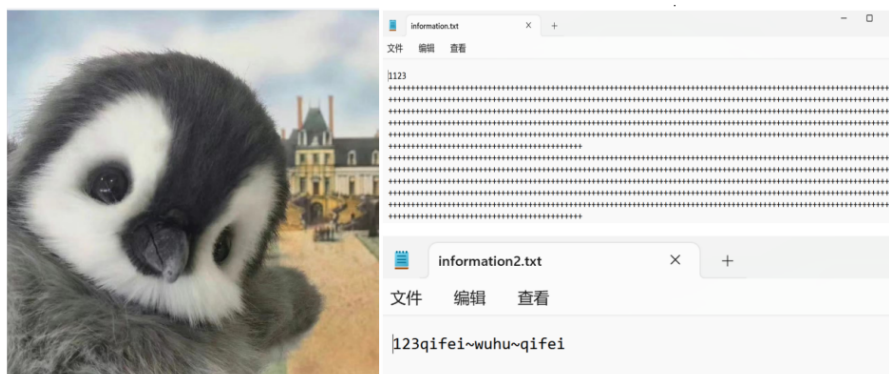


图 6: 实验所用示例图像与文本

(一) 实验一:

实现 LSBM 隐写算法的消息嵌入和提取, 并进行安全性分析。

(二) 实验二:

卡方和 RS 分析任选一个实现, 并且利用实现的分析方法检测不同嵌入率下 LSBR 和 LSBM 的情况, 进行对比分析。

(三) 实验三:

实现 F4 隐写算法的消息嵌入和提取, 并进行安全性分析。

(四) 实验四:

实现 F5 隐写算法的消息嵌入和提取, 并进行安全性分析。

同时, 对于上述四个实验, 要求所有隐写算法的嵌入参数统计分布 (如: 像素

值，JPEG 系数）与 cover 分布进行对比分析，观察安全隐写算法在统计分布保持方面的性能提升。并用卡方，或者 RS 方法对改进后的安全隐写算法进行隐写分析，观察其抗检测能力的提升情况。

五、实验环境:

- 操作系统: Windows11 家庭版
- 基本硬件: CPU: AMD Ryzen 5 5600H, 16G 内存
- 所用软件: Matlab 2023a

六、实验步骤:

(一) 实验一: LSBM 隐写

由于上一次实验中已经完成了 LSB 隐写的代码实现，因此只需在上次代码的基础上修改信息嵌入规则即可。将原本的最低位改为待隐藏的 bit 信息规则改为，当选中像素点的最低位与待嵌入 bit 信息不同时，随机的进行加 1 或减 1，这样当想要嵌入 0 时，若选中像素点的最低位为 0 时便无需更改；若最低位为 1 时，随机的加 1 或减 1，这也使得隐藏信息位仍然和待隐藏信息相同。

由 LSBM 的原理可知，此处只需修改嵌入规则，而随机取点算法与提取算法可完全与 LSB 相同。具体的嵌入代码如下所示：

```
1 % B为待嵌入bit信息长度，A数组中存放了待嵌入数据
2 [x,y]=randinterval(data,B,key);
3 for i=1:B
4     data_last=mod(data(x(i),y(i)),2);
5     if data_last==A(p)
6         p=p+1;
7         continue;
8     end
9     if rand(1)>0.5
10         data(x(i),y(i))=data(x(i),y(i))+1;
11         if data(x(i),y(i))>255
12             data(x(i),y(i))=255;
13         end
14     else
15         data(x(i),y(i))=data(x(i),y(i))-1;
```

```
16         if data(x(i),y(i))<0
17             data(x(i),y(i))=0;
18         end
19     end
20     p=p+1;
21 end
```

可以看到，在具体的代码实现中，首先通过调用第一次实验中的 `randinterval` 函数来随机选点，之后对于选择的像素点进行信息嵌入。当待嵌入信息与选择像素点最低位信息相同时，直接执行 `continue`；若不相同，则通过调用随机数产生函数 `rand`，随机生成 $[0,1]$ 之间的数，以同等的概率随机进行加 1 或减 1。

（二）实验二：卡方分析和 RS 分析

6.2.1 卡方分析

卡方分析有两种，一种是针对整个图像进行卡方分析，整个图像得出一个隐写概率 P ；另一种是对图像分块进行卡方分析，可分为每 10%，20% 等，或者对图像进行 $8*8$ 进行分块，对每个分块得出一个隐写概率并绘制直方图。在此实验中，我实现了对整个图像进行卡方分析，已经将图像每 10% 像素点进行分块，从而进行分块分析。卡方分析的流程图如下所示：

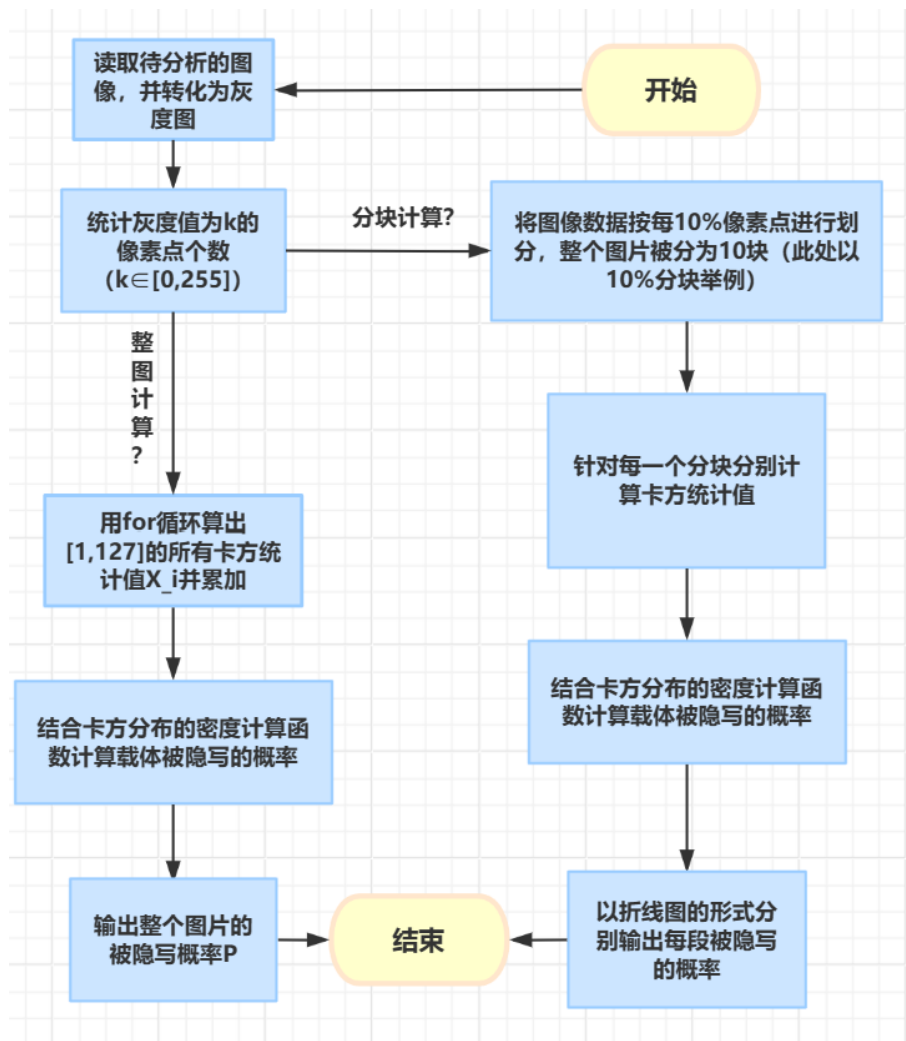


图 7: 实验所用示例图像与文本

以下从 3 个方面对卡方分析过程进行记录。

1) 统计不同像素值的个数卡方分析需要获得不同像素值（0-255）每个像素值的在图片中出现的次数，为了后续方便进行统计量的构造，首先使用一个简单的二层循环获得该值，需要注意的是 matlab 中矩阵下标从 1 开始，而像素值从 0 开始，存在错位，在处理矩阵时需要格外小心。具体代码如下所示：

```

1  function [count]=get_count(data)
2  [m,n]=size(data);
3  count=zeros(1,256);
4  for i=1:m
5      for j=1:n
6          pixel=data(i,j);
7          count(pixel+1)= count(pixel+1)+1;  %1-256对应着0-255像素值
  
```

```

8         end
9     end
10 end

```

2) 图像进行分块处理

若此时进行的是整个图片的卡方分析，则可直接忽略此步骤。当图像分块进行处理时，需要将图片按照像素点的个数进行分块提取，在这里使用的是每 10% 像素点划分为 1 块，具体来讲便是 500*500 的图像中，每块的大小为 500*50，实现代码如下所示：

```

1 function [splite_data]=divide(data,num)
2     [m,n]=size(data);
3     tn=floor(n/num);
4     splite_data=zeros(num,m,tn);
5     sy=1;
6     ey=tn;
7     for i=1:num
8         splite_data(i,:,:)=data(:,sy:ey);
9         sy=sy+tn;
10        ey=ey+tn;
11    end
12 end

```

3) 进行卡方分析

准备工作结束后，便可以构建卡方分析所需的统计量，在实验原理处已经很清楚的给出了所需构造的统计量，最后直接调用 matlab 内置函数 `chi2cdf` 获得卡方分布的对应概率，便可获得图像的隐写概率，但需要注意的是此处卡方分布的自由度并不是固定值，而是由图像自身决定的，代码实现如下所示：

```

1 function [p]=get_lsb_P(count)
2     X=0;
3     k=0;
4     for i=0:127
5         a=2*i;
6         b=2*i+1;
7         %1-256 对应着 0-255 像素值
8         h_2i=count(a+1);
9         h_2i1=count(b+1);
10        h_2i_star=(h_2i+h_2i1)/2;
11        if h_2i_star~=0

```

```

12         X=X+(h_2i-h_2i_star)^2/h_2i_star;
13         k=k+1;
14     end
15 end
16 p=1-chi2cdf(X,k);
17 end

```

6.2.2 RS 分析

RS 分析相比卡方分析要稍微麻烦一些，需要首先对图像进行分块处理，并定义一系列函数，分别按照 M 与 $-M$ 求出相应的变量值，最终通过解出一个特定方程得出图像的隐写概率。RS 分析的流程如下所示：

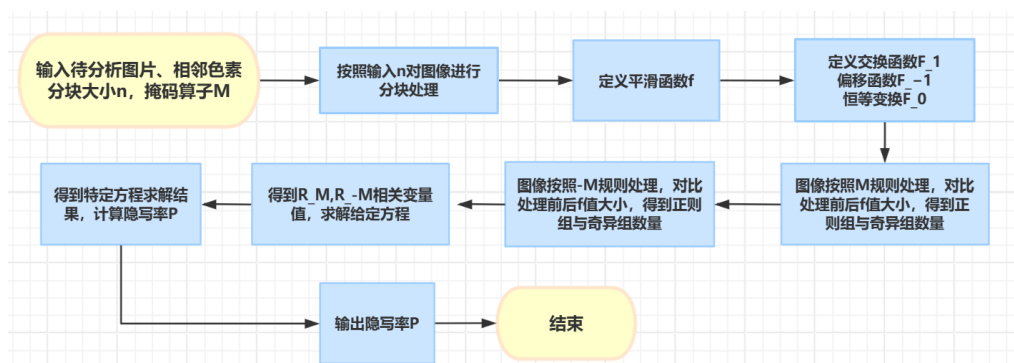


图 8: 实验所用示例图像与文本

RS 实现过程可主要分为以下几步：

1) 将图像进行分块处理

由于 RS 分析需要将图像处理为与 M 相同长度的小块，由于这里 M 为 1×4 的向量，故将图像整体分为多个 2×2 的图像块，这样当 M 与图像块作用时可以做到一一对应。分块过程也很简单，将图像 `data` 输入，最终输出一个 `num*4` 的矩阵，其中 `num` 为分块个数，4 代表每个 2×2 小块，在此处采用 Z 字形存储，具体的代码如下所示：

```

1 function [data_M,num]=split_data(data)
2     [m,n]=size(data);
3     t1=floor(n/2);
4     t2=floor(m/2);
5     num=t1*t2;

```

```

6     data_M=zeros(num,4);
7     k=0;
8
9     for i=1:2:m-2
10         for j=1:2:n-2
11             k=k+1;
12             data_M(k,1)=data(i,j);
13             data_M(k,2)=data(i,j+1);
14             data_M(k,3)=data(i+1,j);
15             data_M(k,4)=data(i+1,j+1);
16         end
17     end
18 end

```

2) 定义平滑函数 f 与处理函数

这里的平滑函数 f 即计算一个分组中的像素的空间相关性/混乱程度，处理函数是指将图像按照掩码算子 M 的值进行处理 (在代码实现中定义为 FF，需要输入图像分块和 M，返回值为处理后的图像像素值)，实现代码如下所示：

```

1 function [r]=f(temp)
2     r=abs(temp(1)-temp(2))+abs(temp(2)-temp(3))+abs(temp(3)-temp(4));
3 end
4
5 function [v]=FF(temp,M)
6     for i=1:4
7         if M(i)==1
8             t=mod(temp(i),2);
9             if t==0
10                 temp(i)=temp(i)+1;
11             else
12                 temp(i)=temp(i)-1;
13             end
14         elseif M(i)==-1
15             t=mod(temp(i),2);
16             if t==0
17                 temp(i)=max(temp(i)-1,0);
18             else
19                 temp(i)=min(temp(i)+1,255);
20             end
21         end
22     end

```



```
23     v=temp;
24 end
```

3) 将掩码算子 M 应用于图像

每个图像块首先按照掩码算子 M 进行处理，对比处理前后像素的空间相关性的变化，来更新正则组和奇异组的个数；按照掩码算子 $-M$ 进行处理。之后再将每个分块中像素点的最低位进行取反处理，再分别进行掩码算子 M 与 $-M$ 处理。循环处理完所有的图像块，得到所有所需的变量。在代码实现中正则组和奇异组的个数 (按照 M 与 $-M$, 最低位是否取反, 可得到 8 组变量) 存储再矩阵 R 之中，具体的实现代码如下所示：

```
1  for i=1:num
2      temp=data_M(i,:);
3      value=f(temp);
4      value1=f(FF(temp,M));
5      if value1>value
6          R(1,1)=R(1,1)+1;
7      elseif value1<value
8          R(1,2)=R(1,2)+1;
9      end
10     value1=f(FF(temp,-M));
11     if value1>value
12         R(1,3)=R(1,3)+1;
13     elseif value1<value
14         R(1,4)=R(1,4)+1;
15     end
16 end
17 for i=1:num
18     temp=data_M(i,:);
19     temp=bitxor(temp,1);
20     value=f(temp);
21     value1=f(FF(temp,M));
22     if value1>value
23         R(2,1)=R(2,1)+1;
24     elseif value1<value
25         R(2,2)=R(2,2)+1;
26     end
27     value1=f(FF(temp,-M));
28     if value1>value
29         R(2,3)=R(2,3)+1;
```

```

30     elseif value1<value
31         R(2,4)=R(2,4)+1;
32     end
33 end

```

4) 求解特定方程，得出隐写率

当所需的变量都准备好后，还要求解一个特定的方程，方程如下所示：

$$2(d_1+d_0)x^2+(d_{-0}-d_{-1}-d_1-3d_0)x+d_0-d_{-0}=0$$

图 9: 方程说明

其中方程参数的定义如下所示：

$$d_0=R_M(p/2)-S_M(p/2), d_1=R_M(1-p/2)-S_M(1-p/2)$$

$$d_{-0}=R_{-M}(p/2)-S_{-M}(p/2), d_{-1}=R_{-M}(1-p/2)-S_{-M}(1-p/2)$$

图 10: 参数说明

可以借助 `matlab` 中内置的 `solve` 函数求解该方程，取解中绝对值较大，记为 px ，图片的隐写概率 $P = px/(px - 0.5)$ ，此部分的实现代码如下：

```

1 d_0=R(1,1)-R(1,2);
2 d_1=R(2,1)-R(2,2);
3 d_00=R(1,3)-R(1,4);
4 d_11=R(2,3)-R(2,4);
5
6 syms x
7 equation=2*(d_1+d_0)*x^2+(d_00-d_11-d_1-3*d_0)*x+d_0-d_00==0;
8 solution=solve(equation,x);
9 x1=double(solution(1));
10 x2=double(solution(2));
11 if abs(x1)>abs(x2)
12     px=x2;
13 else
14     px=x1;
15 end
16 p=px/(px-0.5)

```

(三) 实验三：F4 隐写

由于 F4 隐写是在 F3 隐写的基础上修改了隐写规则，而 F3 隐写又是基于 Jsteg 进行的，因此我在编写代码的过程中也是先实现了 Jsteg，修改隐写规则得到 F3，又进一步修改隐写规则得到 F4。整体实现流程如下图所示：

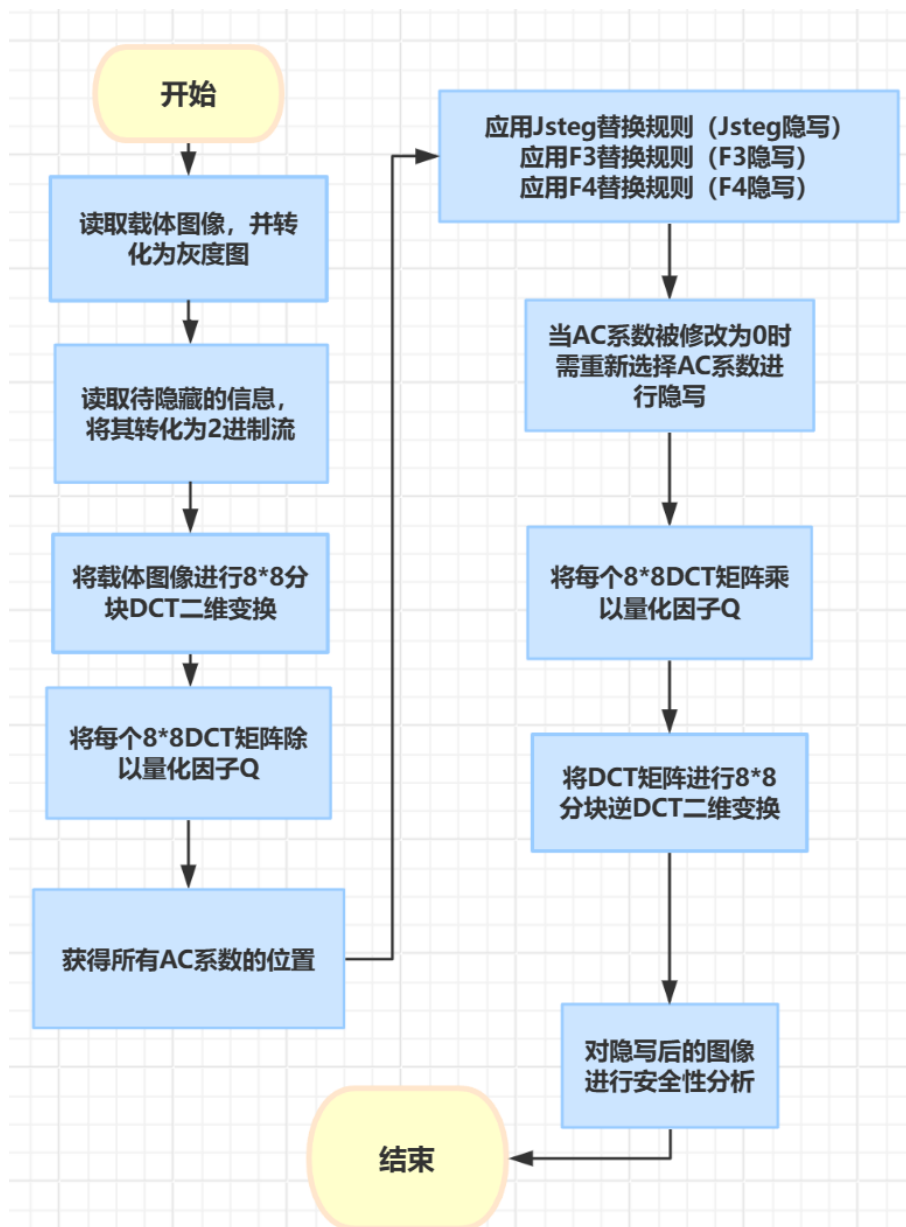


图 11: 参数说明

1) 对图像进行 DCT 变换并量化

无论上面提到的哪一种隐写都是发生在频域上的 AC 系数，即 DCT 矩阵中除了左上角的元素 (DC 系数) 外都可以进行修改，这一步也是后续隐写进行的基础，

类似于第一次实验的 DCT 隐写，首先对图像进行 8*8 分块 DCT 变换，再除以量化因子 Q。代码如下：

```
1 data=rgb2gray(imread(imgpath));
2 origin_data=data;
3 Q=[16 11 10 16 24 40 51 61
4     12 12 14 19 26 58 60 55
5     14 13 16 24 40 57 69 56
6     14 17 22 29 51 87 80 62
7     18 22 37 56 68 109 103 77
8     24 35 55 64 81 104 113 92
9     49 64 78 87 103 121 120 101
10    72 92 95 98 112 100 103 99];
11 [h,w]=size(data);
12 D=zeros(h,w); %零时存储矩阵
13 for i=1:h/8
14     for j=1:w/8
15         D(8*(i-1)+1:8*i,8*(j-1)+1:8*j)=dct2(data(8*(i-1)+1:8*i,8*(j-1)+1:8*j));
16         D(8*(i-1)+1:8*i,8*(j-1)+1:8*j)=round(D(8*(i-1)+1:8*i,8*(j-1)+1:8*j)./Q)
17     end
18 end
```

2) 在频域上进行隐写

Jsteg 隐写时首先需要挑出 AC 系数不为 0, 1, -1 的位置，然后将 bit 信息逐位进行嵌入，嵌入规则也十分简单。嵌入信息为 0 时，若 AC 系数为偶数则无需修改，若为奇数，则将 AC 系数的绝对值进行减 1 处理；嵌入信息为 1 时，若 AC 系数为奇数则无需修改，若为偶数，则将 AC 系数的绝对值进行加 1 处理。在提取时只需判断 AC 系数的奇偶便可得知原本嵌入信息的值。由于代码较为简单，此处就不进行展示，Jsteg 隐写的完整代码在提交附带的压缩包中。

F3 隐写相比 Jsteg 只是修改了隐写规律。在 AC 系数的选择上只排除了为 0 的位置，将 bit 信息逐位进行嵌入，F3 隐写利用了原始值为 +1 和 -1 的 DCT 系数，提高了嵌入率。关于 F3 的嵌入规则也已经在实验原理处进行了详细的说明，在此也就不再赘述。但由于 AC 系数中绝对值为 1 的数值较多，当其被修改为 0 时，嵌入算法会继续嵌入直到找到一个偶数值，或将一个奇数改为偶数。为了解决这个问题，便又引入了 F4 隐写 (F3 隐写的代码也在压缩包中)。

F4 隐写的核心原则便是 AC 系数中，除 0 外，正奇数和负偶数代表 1，正偶数

和负奇数代表 0。在进行嵌入时便是根据当前 AC 系数的值与待嵌入信息，按照上面的原则对 AC 系数进行修改。实现代码如下：

```
1 function [r,flag]=insert_F4(v,b)
2     r=v;
3     flag=0;
4     if b==1
5         if (~mod(r,2)&& r<0) || (mod(r,2)&& r>=0)
6             flag=1;
7             return;
8         elseif ~mod(r,2)&& r>=0
9             r=r-1;
10            if r~=0
11                flag=1;
12            end
13            return
14        elseif mod(r,2)&& r<0
15            r=r+1;
16            if r~=0
17                flag=1;
18            end
19            return
20        end
21    else
22        if (~mod(r,2)&& r>=0) || (mod(r,2)&& r<0)
23            flag=1;
24            return;
25        elseif ~mod(r,2)&& r<0
26            r=r+1;
27            if r~=0
28                flag=1;
29            end
30            return
31        elseif mod(r,2)&& r>=0
32            r=r-1;
33            if r~=0
34                flag=1;
35            end
36            return
37        end
38    end
39 end
```

在完成嵌入后，再将 DCT 矩阵乘以量化因子 Q，并进行逆 DCT 变换，便可得到时域上的载体图像。

3) F4 信息提取

信息提取过程十分简单，只需先将图像转换到频域上，对于所有的 AC 系数按照除 0 外，正奇数和负偶数代表 1，正偶数和负奇数代表 0 的规则进行提取，当已提取的长度与嵌入信息长度相等时便可退出。提取的核心代码如下：

```
1 for i=1:length
2     x=site(1,i);
3     y=site(2,i);
4     if (D(x,y)>0 && mod(D(x,y),2)) || (D(x,y)<0 && ~mod(D(x,y),2))
5         fwrite(fid,1,'bit1 ');
6         r(i)=1;
7     elseif (D(x,y)>0 && ~mod(D(x,y),2)) || (D(x,y)<0 && mod(D(x,y),2))
8         fwrite(fid,0,'bit1 ');
9         r(i)=0;
10    end
11 end
```

(四) 实验四：F5 隐写

F5 隐写相比 F4 进行了较大的改变，它灵活借助了汉明码的思想，实现 1 次修改多位嵌入的效果，这使得在嵌入同等长度信息的比特时，F5 隐写相比于之前的隐写可以做到最少次修改，这进一步提升了隐写的效率与隐蔽性。

在本实验中采用的是一次隐写 3 位 bit 数据，由汉明码的性质可知每次需要的载体长度为 7 位。实现过程可由下述步骤完成：

1) 图像转化到频域

此步骤与 F4 隐写除了校验矩阵外完全相同，故不再赘述。

2) 信息嵌入过程

在 F5 隐写中，首先根据信息的长度按照每 3 位进行分组，分为多组；之后开始载体位置的选择过程，在图像对应的 DCT 矩阵中，按 Z 字型进行遍历，当 AC 系数不为 0 时将其进行记录，AC 系数为正奇数或负偶数记为 1，负奇数或正偶数记为 0，当载体记录达到 7 位时进行嵌入。

在嵌入时，首先将 7 位载体数据与校验矩阵进行相乘，得到一个 1*3 向量，将该向量与待嵌入的 3 位 bit 信息组成的向量进行异或运算，得到的 3 位结果可视为一个十进制数的二进制表示，将其转化为十进制后，便可得知需要修改载体中的哪一位。在修改后需要检查修改过后的 DCT 值是否为 0，若为 0 则重新选择 1 位数据作为载体信号。

嵌入过程代码如下所示：

```
1 for i=1:h
2     for j=1:w
3         if (D(i,j)>0 && mod(D(i,j),2)==1)|| (D(i,j)<0 && mod(D(i,j),2)==0)
4             %正奇数或负偶数为1
5             a(k)=1;
6             sit(k,1)=i;
7             sit(k,2)=j;
8             k=k+1;
9         elseif (D(i,j)<0 && mod(D(i,j),2)==1)|| (D(i,j)>0 && mod(D(i,j),2)==0)
10            %负奇数或正偶数为0
11            a(k)=0;
12            sit(k,1)=i;
13            sit(k,2)=j;
14            k=k+1;
15        end
16        if(k>7)
17            if B-num>=2
18                data_bit=[A(num),A(num+1),A(num+2)]';
19            elseif B-num==1
20                data_bit=[A(num),A(num+1),0]';
21            elseif B==num
22                data_bit=[A(num),0,0]';
23            end
24
25            temp=M*a;
26            temp=mod(temp,2);
27            n=bitxor(data_bit,temp); %异或
28            n=n(1)*4+n(2)*2+n(3);
29            %% 修改第n位的DCT值
30            if n>0 %需要修改，否则不需要修改 绝对值减一
31                if D(sit(n,1),sit(n,2))<0
32                    stego(sit(n,1),sit(n,2))=D(sit(n,1),sit(n,2))+1;
33                elseif D(sit(n,1),sit(n,2))>0
```

```

32         stego(sit(n,1),sit(n,2))=D(sit(n,1),sit(n,2))-1;
33     end
34     %% 检查修改过后的DCT值是否为0，若为0则重新选择1位数据作为载体信
      号
35     if stego(sit(n,1),sit(n,2))==0
36         k=k-1;
37         sit(n:k-1,:)=sit(n+1:k,:);
38         a(n:k-1)=a(n+1:k);
39         continue;
40     end
41 end
42 num=num+3;
43 k=1;
44 end
45 if(num>B)
46     break;
47 end
48 end
49 if(num>B)
50     break;
51 end
52 end

```

3) F5 信息提取

信息提取过程即为嵌入的逆过程，将图像转化到频域后，依旧是每 7 个不为 0 的 AC 系数为一组，与校验矩阵相乘得到嵌入的 3 位数据，当提取的信息与嵌入信息长度相等时便可退出。核心提取代码如下：

```

1  for i=1:h
2      for j=1:w
3          if (D(i,j)>0 && mod(D(i,j),2)) || (D(i,j)<0 && ~mod(D(i,j),2))
4              a(k)=1;
5              k=k+1;
6          elseif (D(i,j)>0 && ~mod(D(i,j),2)) || (D(i,j)<0 && mod(D(i,j),2))
7              a(k)=0;
8              k=k+1;
9          end
10         if k>7 %表示集满7个数据
11             temp=M*a;
12             temp=mod(temp,2);
13             for c=1:3

```



```

14         num=num+1;
15         r(num)=temp(c);
16         if num==length
17             jumpaway=1;
18             break
19         end
20     end
21     if jumpaway
22         break;
23     end
24     k=1;
25 end
26 end
27 if num>=length
28     break;
29 end
30 end

```

七、实验结果与分析:

(一) 实验一: LSBM

7.1.1 图像隐写对比

通过运行信息嵌入代码,可得嵌入图像的数据,使用子图同时输出原始图像与修改图像如图 12所示:

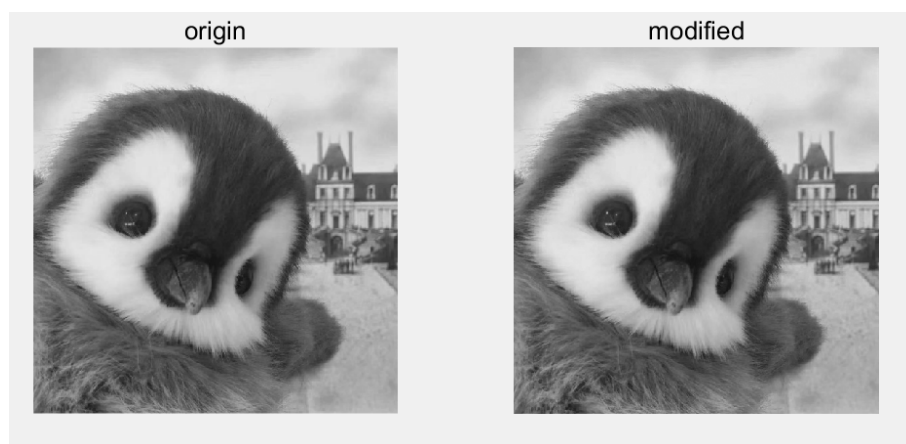


图 12: LSBM 图像对比

7.1.2 信息提取呈现

将上述被修改后的图像送入提取函数中，可提取出隐藏的信息，如图 13 所示。
 r 中存放了隐藏信息的二进制表示， r_{ASCII} 中存放了对应的字符数据。

```
r_ASCII =  
  
'1123+++++  
+++++  
+++++  
+++++  
1123+++++
```

图 13: LSBM 隐写信息提取

7.1.3 隐写直方图对比

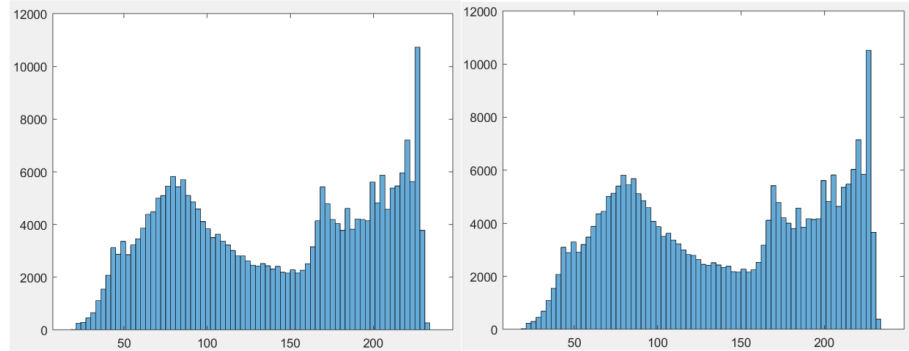


图 14: 隐写直方图

由图 14 可以看到，即使嵌入了大量的信息（此处嵌入了 97728bit 信息）LSBM 的直方图并没有明显变化，这说明 LSBM 能够有效的消除 LSB 中存在的值对效应。

7.1.4 安全性分析

关于 LSBM 的安全性分析放在实验二中进行，在此处就不再重复展示。

(二) 实验二: 卡方与 RS 分析

首先先给出 LSBR 与 LSBM 的嵌入结果图对比图，如图 15 所示，可以看到从视觉上两者并没有明显的区别 (嵌入率为 35%)。

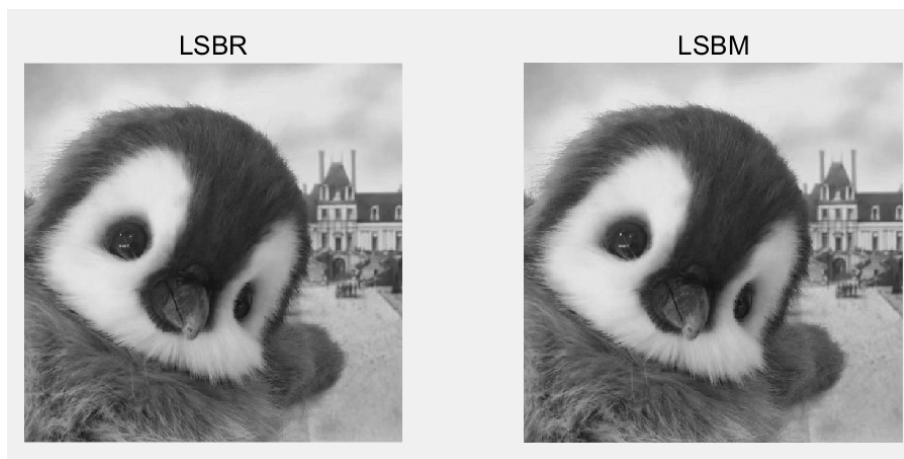


图 15: LSBR 与 LSBM 隐写对比图

7.2.1 LSBR 与 LSBM 的整块卡方分析

整块卡方分析即整个图像为一个单位进行卡方分析，运行代码得到两者的隐写概率分别为：5.839e-07 与 0(35% 嵌入率)，这一结果是在我们的意料之内的，虽然 LSBR 的隐写概率十分接近 0，但仍然大于 LSBM 对应的 0。之所以 LSBR 的隐写概率很小的原因是因为整块分析时会使得卡方分析的自由度较大，这也说明了卡方分析并不适合整图分析。

7.2.2 LSBR 与 LSBM 的分块卡方分析

按照每 10% 对图像进行分块，对每一块进行卡方分析结果如图 16所示。从图中可以看到，在分块进行卡方分析时，LSBR 的隐写率较高，多个位置处的隐写率超过了 0.7；反观 LSBM，其大部分的位置的隐写率为 0，最高处的隐写率仅为 0.2，这说明了相比于 LSBR，LSBM 具有更强的隐蔽性，在面对卡方分析检测技术时更不容易被发现。

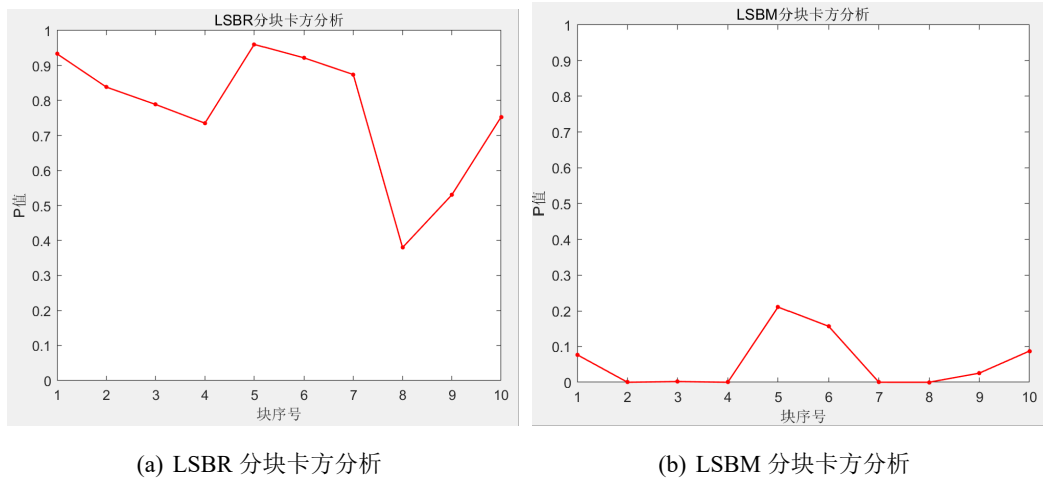


图 16: LSBR 与 LSBM 的卡方分析

另一点值得注意的是，卡方分析也会受载体本身的影响，此处采用的图像为作者经过多次试验选择的图像，本实验所用图像和 Lena 图像直接进行卡方分析时的结果如图 17所示，可看到原始图像的隐写率始终较低；但如果采用常见的 Lena 图像时即在未修改时都具有较高的隐写率，故得出结论，卡方分析与原始载体图像密切相关。

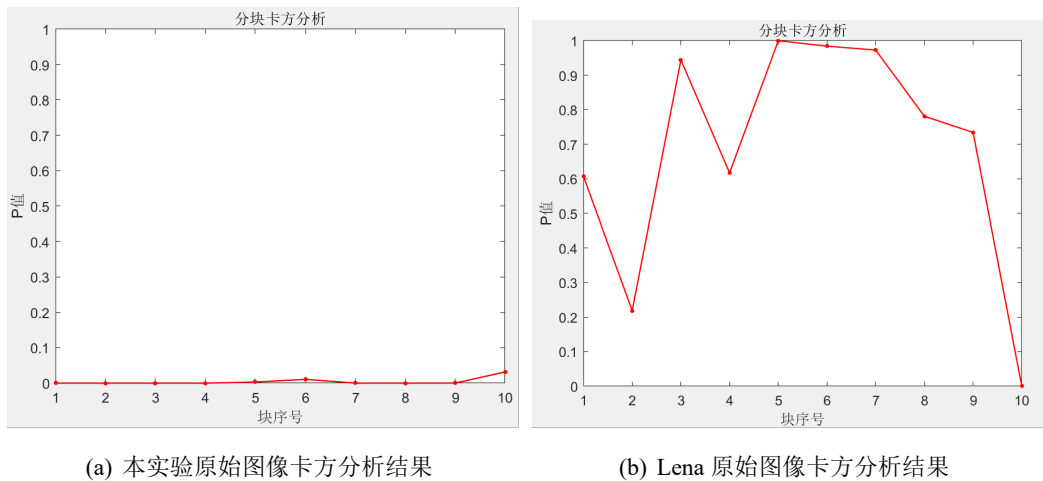


图 17: 不同载体图像的影响

7.2.3 LSBR 与 LSBM 的整块 RS 分析

RS 分析就相对简单一些，没有繁琐的分块过程，直接整个图像得到一个隐写结果。对于本身嵌入率为 35% 的图片，运行代码得到 LSBR 嵌入率为 0.3889，对

于 LSBM 嵌入率为 0。可以看到，即使是 RS 分析，LSBM 仍然具有较好的隐蔽性。

7.2.4 不同嵌入率下 LSBR 与 LSBM 隐蔽性探究

上述的实验结果是在嵌入率约为 35% 下得到的，为了进一步探究不同嵌入率下 LSBR 与 LSBM 的隐蔽性，在这里手动设置不同的嵌入率。但由于随机选点算法的设计，这里不能够做到百分百嵌入，最大嵌入率约为 55%，故探究在嵌入率为 10% 到 50% 之间不同嵌入的影响。

首先是不同嵌入率下的卡方分析，由于卡方分析更适合分块进行处理，因此在这里按每 10% 像素点进行划分，将每个块的隐写率进行平均运算，实验结果如图 18 所示。

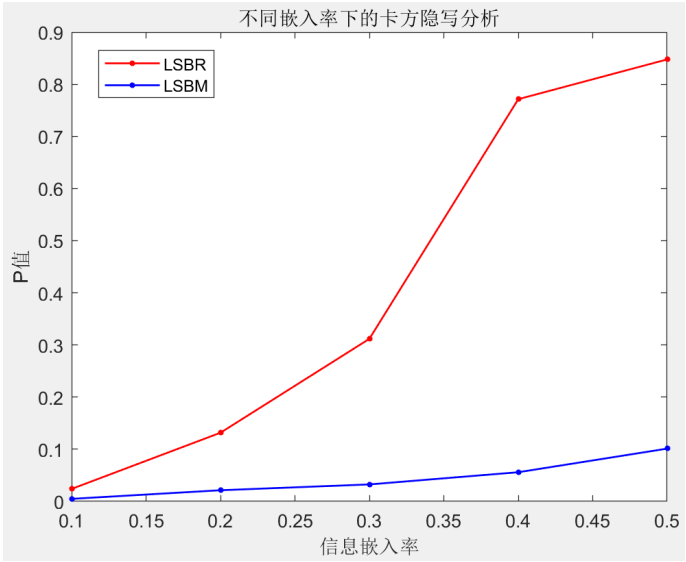


图 18: 不同嵌入率下的卡方隐写分析

其次是在不同嵌入率下的 RS 分析，直接对整图进行分析得到的结果如图 19 所示。但此处的 RS 分析存在一些小问题，即嵌入率为 0.3 之内时，对于 LSBR 嵌入率的预测还是比较准确的；当嵌入率高于 0.3 时，对于 LSBR 嵌入率的预测出现了一定的偏差，预测结果会偏高一些。

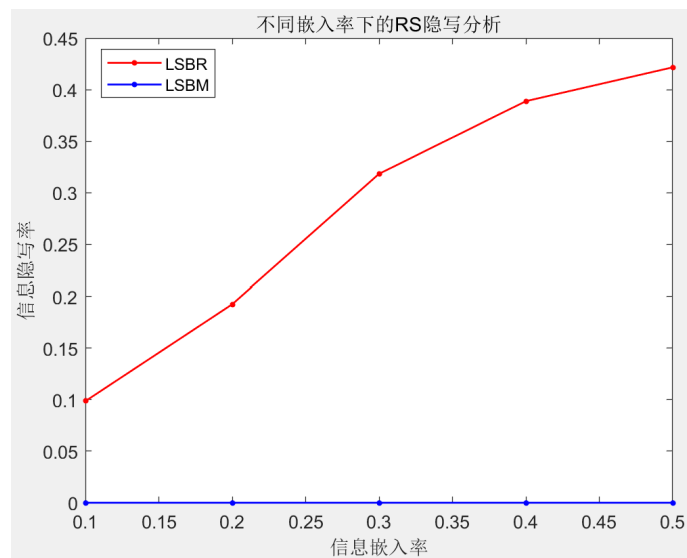


图 19: 不同嵌入率下的 RS 隐写分析

从上可以看到，不同的嵌入率下，LSBM 始终具有更高的隐蔽性，更难以被检测算法发现。

(三) 实验三:F4 隐写算法

7.3.1 图像隐写对比

通过运行信息嵌入代码，可得嵌入图像的数据，使用子图同时输出原始图像与修改后的图像如图 20所示。

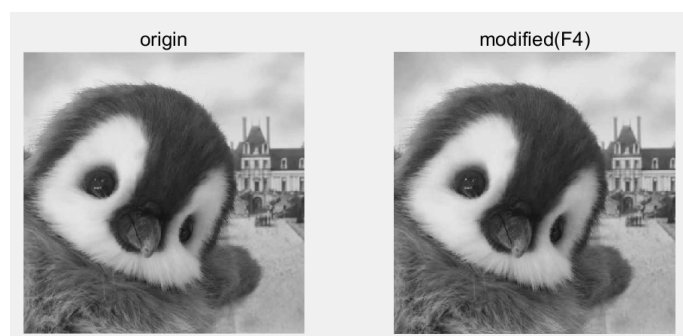
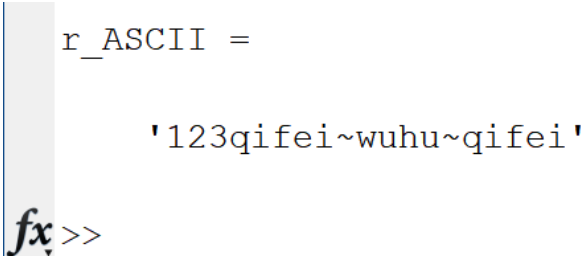


图 20: F4 隐写算法效果图

7.3.2 信息提取呈现

将上述被修改后的图像送入提取函数中，可提取出隐藏的信息，如图 21所示。 r_{ASCII} 中存放了对应的字符数据 (这里已经将提取的二进制数据每 8 位转化为了它们对应的 ASCII 码)。

A screenshot of a MATLAB command window. The prompt 'fx>>' is on the left. The command 'r_ASCII =' is entered, followed by a new line containing the string '123qifei~wuhu~qifei' in single quotes.

```
fx>> r_ASCII =  
      '123qifei~wuhu~qifei'
```

图 21: F4 隐写算法信息提取

7.3.3 安全性分析

在此部分，对 F4 隐写进行安全性分析，考虑 F4 算法在不同的嵌入率下与传统 LSB 隐写进行对比，分别进行卡方分析与 RS 分析。其中卡方分析仍采用实验二中的分块原则，具体的处理方法与之前的实验完全相同。分析结果如图 22所示，可以看到无论是何种嵌入率下、何种隐写分析技术下，F4 隐写均具有极高的隐蔽性。

但至于为什么隐写率始终为 0，自处考虑到可能是由于载体本身具有较好的性质，在无任何嵌入时都具有近似 0 的嵌入率；其次是由于嵌入的信息量较少，这是因为在 F4 隐写算法中能够进行修改的是 AC 系数不为 0 的位置，当修改为 0 时，还需要重新选点，故导致了实际能够嵌入的信息有限。

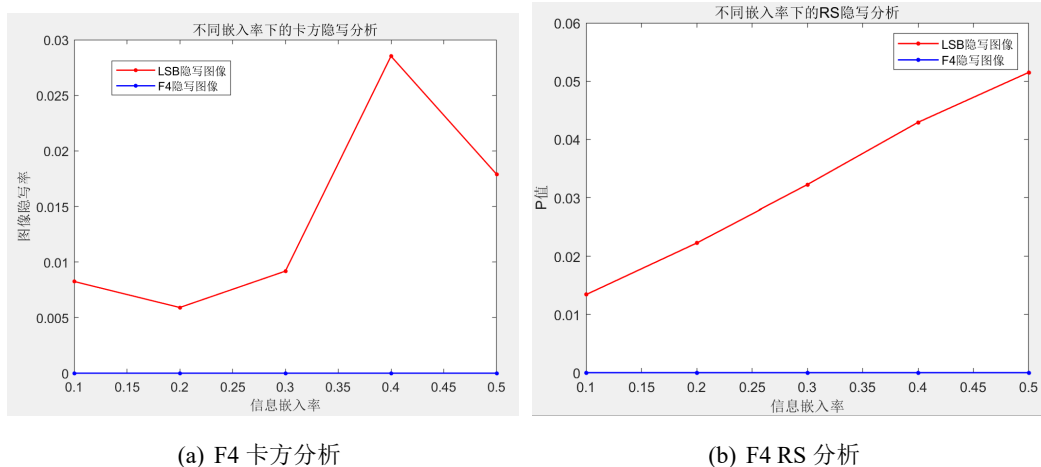


图 22: F4 安全性分析

(四) 实验四:F5 隐写算法

7.4.1 图像隐写对比

通过运行信息嵌入代码,可得嵌入图像的数据,使用子图同时输出原始图像与修改图像如图 23所示。

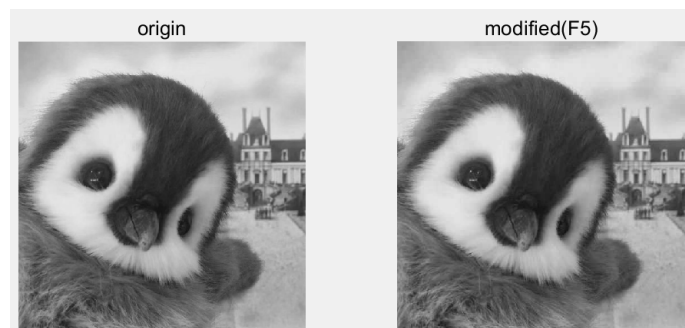


图 23: F5 隐写算法效果图

7.4.2 信息提取呈现

将上述被修改后的图像送入提取函数中,可提取出隐藏的信息,如图 24所示。
 r_{ASCII} 中存放了对应的字符数据 1(这里已经将提取的二进制数据每 8 位转化为了它们对应的 ASCII 码)。


```

r_ASCII =
'123qifei~wuhu~qifei'

fx>>

```

图 24: F5 隐写算法信息提取

7.4.3 安全性分析

在此部分，对 F5 隐写进行安全性分析，考虑 F4 算法在不同的嵌入率下与传统 LSB 隐写进行对比，分别进行卡方分析与 RS 分析。其中卡方分析仍采用实验二中的分块原则，具体的处理方法与之前的实验完全相同。分析结果如图 25 所示，可以看到无论是何种嵌入率下、何种隐写分析技术下，F5 均具有极高的隐蔽性。关于隐写率较低的原因同实验四。

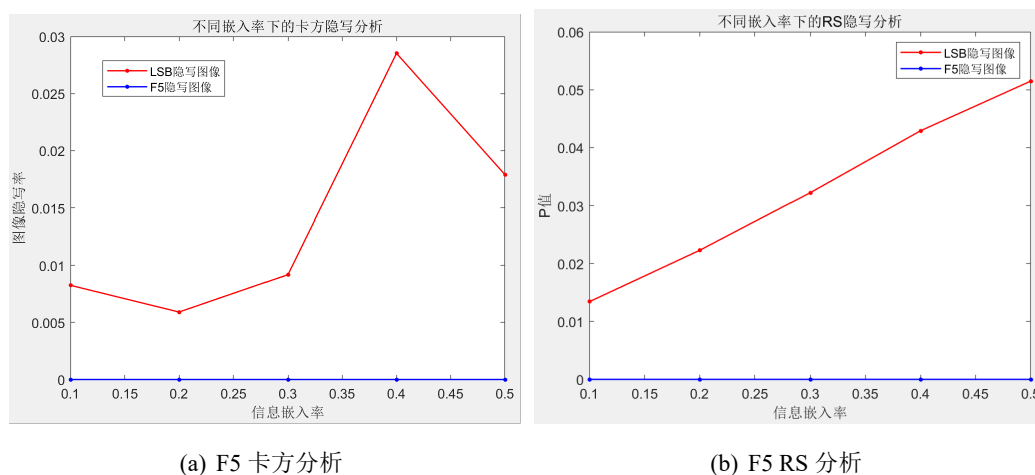


图 25: F5 隐写安全性分析

八、总结及心得体会:

1) 通过此次实验掌握了 LSB 隐写的进阶版 LSBM，虽然改进的思想很简单，但有效的解决了 LSB 中存在的值对效应问题，并且通过安全性分析得出其具有更高的隐蔽性。

2) 此次实验掌握了卡方分析和 RS 分析的原理，能够对图像分块进行安全性分析，并将它们应用到各种隐写算法上，体会了不同隐写算法在面对隐写分析时不

同的隐蔽表现。

3) 掌握了 DCT 变换后的隐写操作，如 Jsteg、F3、F4、F5 隐写等，其中前三个隐写算法是层层递进的，不断完善 AC 系数的修改规则，从中也可体会到隐写效率与隐写隐蔽性的提升。

4) 在学习 F5 隐写时，又着重复习了曾在计算机组成原理中学到的汉明码，体会了利用汉明码的纠错机制而设计的 F5 隐写算法的巧妙之处，通过一位修改，实现隐藏多位信息，体会到了不同领域知识交叉融合的魅力。

5) 通过此次实验进一步提升了代码能力，能够借助 matlab 实现更多更复杂的功能。