

# 信息隐藏技术第一次大作业实验报告

2113662      张丛

## 实验内容

### 二值图像的可视密钥分享方案

原理

实现

实验结果

### 灰度图像的可视密钥分享方案

原理

实现

实验结果

### 彩色图像的可视密钥分享方案

原理

实现

实验结果

### 叠像术

原理

实现

实验结果

灰度和彩色图像的叠像术

### 小结

## 实验内容



















- 1、实现二值图像的 (2,2) 的可视密钥分享方案。
- 2、有能力同学可以做灰度图像的、彩色图像的。
- 3、还可以做 (t, n) 方案。
- 4、叠像术。

(其中, 1必做, 其它都是选做的。)

## 二值图像的可视密钥分享方案

### 原理

对于二值图像的 (2,2) 可视密钥分享方案, 原始的二值图像被分割成4个子图像。

秘密图像的像素	黑			白		
伪装图像1的像素（部分）						
伪装图像2的像素（部分）						
叠加后的像素						

这些子图像中的每一个都被设计成只有一部分信息，因此单独的子图像并不能泄露原始图像的内容。

两个子图像合并时，叠加后的像素为完全黑色则视为原图的黑像素，叠加后的像素为灰度是则视为原图的白色像素，这样可以得到原始图像的信息。

## 实现

代码如下：

```
function Binary_images()
path = input("请输入要加密的图片：", 's');
origin = imread(path);

% Divide the original image into two encrypted images
[image1, image2] = divideImage(origin);

% Display the original image, image1, and image2
figure;
imshow(origin);
figure;
imshow(image1);
figure;
imshow(image2);

% Merge image1 and image2 to recover the original image
recoveredImage = mergeImages(image1, image2);

% Display the recovered image
figure;
imshow(recoveredImage);

end

function [image1, image2] = divideImage(image)
% Initialize
Size = size(image);
x = Size(1);
y = Size(2);
image1 = zeros(2*x, 2*y);
image1(:, :) = 255;
image2 = zeros(2*x, 2*y);
image2(:, :) = 255;

% Take image1 as the first
for i = 1:x
```

```

for j = 1:y
    key = randi(3);
    son_x = 1 + 2 * (i - 1);
    son_y = 1 + 2 * (j - 1);
    switch key
        case 1
            image1(son_x, son_y) = 0;
            image1(son_x, son_y + 1) = 0;
            if image(i, j) == 0
                % Original is black
                image2(son_x+1, son_y) = 0;
                image2(son_x+1, son_y+1) = 0;
            else
                % Original is white
                image2(son_x, son_y+1) = 0;
                image2(son_x+1, son_y+1) = 0;
            end
        case 2
            image1(son_x, son_y) = 0;
            image1(son_x+1, son_y+1) = 0;
            if image(i, j) == 0
                % Original is black
                image2(son_x, son_y+1) = 0;
                image2(son_x+1, son_y) = 0;
            else
                % Original is white
                image2(son_x, son_y) = 0;
                image2(son_x+1, son_y) = 0;
            end
        case 3
            image1(son_x, son_y) = 0;
            image1(son_x+1, son_y) = 0;
            if image(i, j) == 0
                % Original is black
                image2(son_x, son_y+1) = 0;
                image2(son_x+1, son_y+1) = 0;
            else
                % Original is white
                image2(son_x, son_y) = 0;
                image2(son_x, son_y+1) = 0;
            end
        end
    end
end
end

function mergedImage = mergeImages(image1, image2)
Size = size(image1);
x = Size(1);
y = Size(2);
mergedImage = zeros(x, y);
mergedImage(:, :) = 255;

for i = 1:x
    for j = 1:y
        mergedImage(i, j) = image1(i, j) & image2(i, j);
    end
end

```

end

end

具体来说：

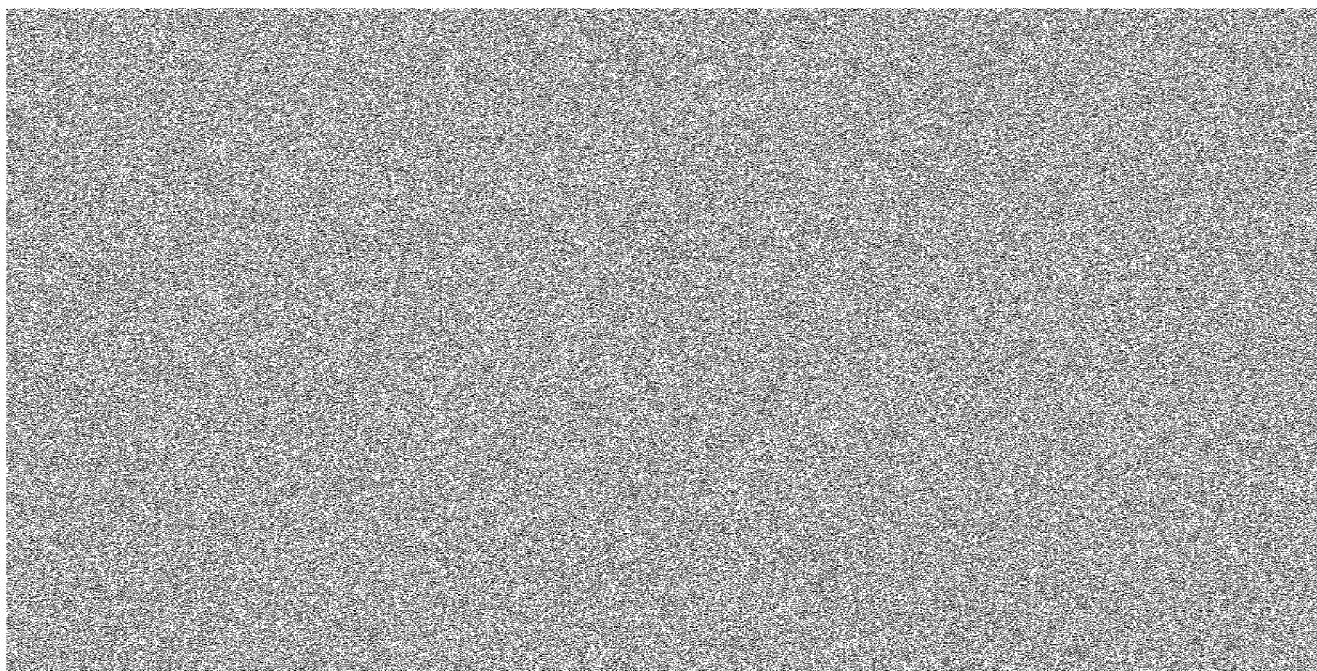
1. `visualCrypModified` 函数是主函数，它负责整个加密和解密过程的控制。首先，输入要加密的图片路径，然后读取该图片。接着，它调用 `divideImage` 函数将原始图片分割成两个加密图像 `image1` 和 `image2`。接着，它显示原始图片、`image1` 和 `image2`。然后，它调用 `mergeImages` 函数将 `image1` 和 `image2` 合并以还原原始图片并显示。
2. `divideImage` 函数接收原始图片作为输入，然后将其分割成两个加密图像。它首先初始化两个空白的图像 `image1` 和 `image2`，然后遍历原始图片的每个像素。对于每个像素，它生成一个随机数 `key`，然后根据 `key` 的值来决定如何将像素分配到 `image1` 或 `image2` 中。最后返回分割后的两个图像。
3. `mergeImages` 函数遍历 `image1` 和 `image2` 的每个像素，然后通过逻辑与运算将对应位置的像素合并为一个新的像素，并将结果存储在 `mergedImage` 中。最后返回合并后的图像。

## 实验结果

原图：



子图一：



子图二:



两子图叠加复原的结果:





## 灰度图像的可视密钥分享方案

### 原理

处理灰度图像，首先需要将连续调图像变为半色调图像。

在课堂上，我们学习了误差扩散法。

```
for m=1:x
    for n=1:y
        if gray(m,n)>127
            out=255;
        else
            out=0;
        end
        error=gray(m,n)-out;
        if n>1 && m<255 && n<255
            gray(m,n+1)=gray(m,n+1)+error*7/16.0; %右方
            gray(m+1,n)=gray(m+1,n)+error*5/16.0; %下方
            gray(m+1,n-1)=gray(m+1,n-1)+error*3/16.0; %左下方
            gray(m+1,n+1)=gray(m+1,n+1)+error*1/16.0; %右下方
            gray(m,n)=out;
        else
            gray(m,n)=out;
        end
    end
end
```

其基本原理是在像素级别上，将每个像素的灰度值与其最接近的二值化阈值进行比较，然后将差值（误差）传播到周围的像素中，以使得整体的误差被均匀分布。

# 实现

代码如下：

```
function Gray_images_22()
path = input("请输入要加密的图片：", 's');
origin = imread(path);

figure(1);
imshow(origin);
title('原图');

gray=rgb2gray(origin);
figure(2);
imshow(gray);
title('灰度图像');

origin_bw = error_diffusion_dithering(gray);
figure(3);
imshow(origin_bw);
title('半色调图像');

[image1,image2] = divideImage(origin_bw);
figure(4);
imshow(image1);
title('子图1');

figure(5);
imshow(image2);
title('(子图二)');

result = mergeImages(image1,image2);
figure(6);
imshow(result);
title('复原图像');

end

function out_image = error_diffusion_dithering(gray_image)
[X, Y] = size(gray_image);
out_image = zeros(X, Y); % 初始化输出图像

for m = 1:X
    for n = 1:Y
        if gray_image(m, n) > 127
            out = 255;
        else
            out = 0;
        end
        error = gray_image(m, n) - out;

        out_image(m, n) = out; % 将当前像素设置为输出值

        % 误差传播
        if n < Y % 右方
            gray_image(m, n+1) = gray_image(m, n+1) + error * 7 / 16.0;
        end
    end
end
```

```

        if m < X && n > 1 % 左下方
            gray_image(m+1, n-1) = gray_image(m+1, n-1) + error * 3 / 16.0;
        end
        if m < X % 下方
            gray_image(m+1, n) = gray_image(m+1, n) + error * 5 / 16.0;
        end
        if m < X && n < Y % 右下方
            gray_image(m+1, n+1) = gray_image(m+1, n+1) + error * 1 / 16.0;
        end
    end
end
end
end

```

```

function [image1, image2] = divideImage(image)
% Initialize
Size = size(image);
x = Size(1);
y = Size(2);
image1 = zeros(2*x, 2*y);
image1(:, :) = 255;
image2 = zeros(2*x, 2*y);
image2(:, :) = 255;

% Take image1 as the first
for i = 1:x
    for j = 1:y
        key = randi(3);
        son_x = 1 + 2 * (i - 1);
        son_y = 1 + 2 * (j - 1);
        switch key
            case 1
                image1(son_x, son_y) = 0;
                image1(son_x, son_y + 1) = 0;
                if image(i, j) == 0
                    % Original is black
                    image2(son_x+1, son_y) = 0;
                    image2(son_x+1, son_y+1) = 0;
                else
                    % Original is white
                    image2(son_x, son_y+1) = 0;
                    image2(son_x+1, son_y+1) = 0;
                end
            case 2
                image1(son_x, son_y) = 0;
                image1(son_x+1, son_y+1) = 0;
                if image(i, j) == 0
                    % Original is black
                    image2(son_x, son_y+1) = 0;
                    image2(son_x+1, son_y) = 0;
                else
                    % Original is white
                    image2(son_x, son_y) = 0;
                    image2(son_x+1, son_y) = 0;
                end
            case 3
                image1(son_x, son_y) = 0;
                image1(son_x+1, son_y) = 0;
                if image(i, j) == 0

```



```

        % Original is black
        image2(son_x, son_y+1) = 0;
        image2(son_x+1, son_y+1) = 0;
    else
        % Original is white
        image2(son_x, son_y) = 0;
        image2(son_x, son_y+1) = 0;
    end
end
end
end

function mergedImage = mergeImages(image1, image2)
Size = size(image1);
x = Size(1);
y = Size(2);
mergedImage = zeros(x, y);
mergedImage(:, :) = 255;

for i = 1:x
    for j = 1:y
        mergedImage(i, j) = image1(i, j) & image2(i, j);
    end
end
end
end

```

流程如下：

1. 读取原始图像。
2. 将原始图像转换为灰度图像。
3. 使用误差扩散法对灰度图像进行半色调化处理。
4. 将半色调图像分割为两个子图像。
5. 使用合并函数将两个子图像合并为原始图像。

具体函数的作用如下：

- error\_diffusion\_dithering 函数实现了误差扩散法的半色调化处理，将输入的灰度图像转换为半色调图像。
- divideImage 函数将半色调图像分割为两个子图像，用于后续的信息隐藏。
- mergeImages 函数将两个子图像合并为原始图像，用于恢复原始图像。

## 实验结果

原图：



灰度图：

灰度图像



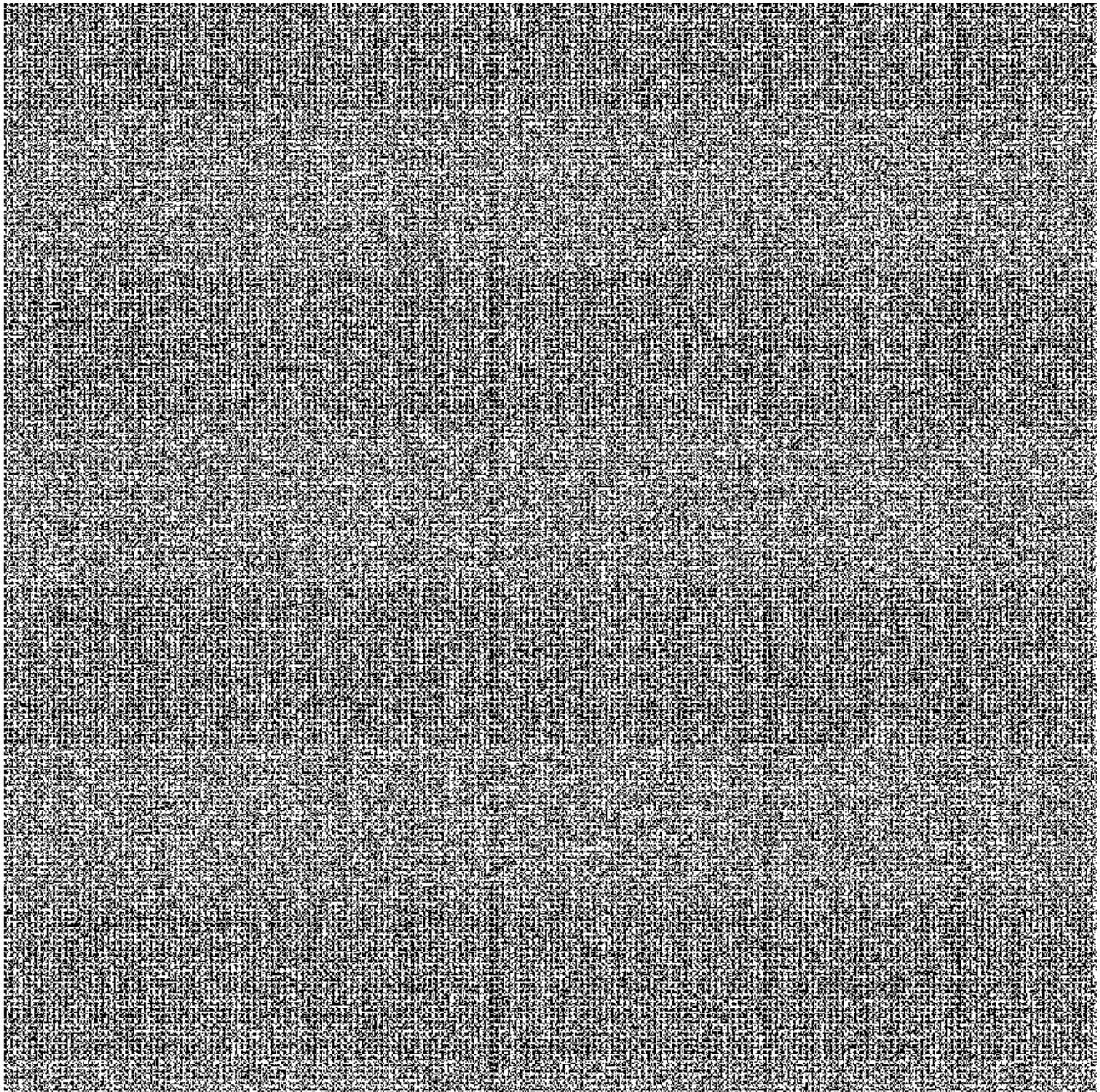
半色调化图：

半色调图像



子图一：

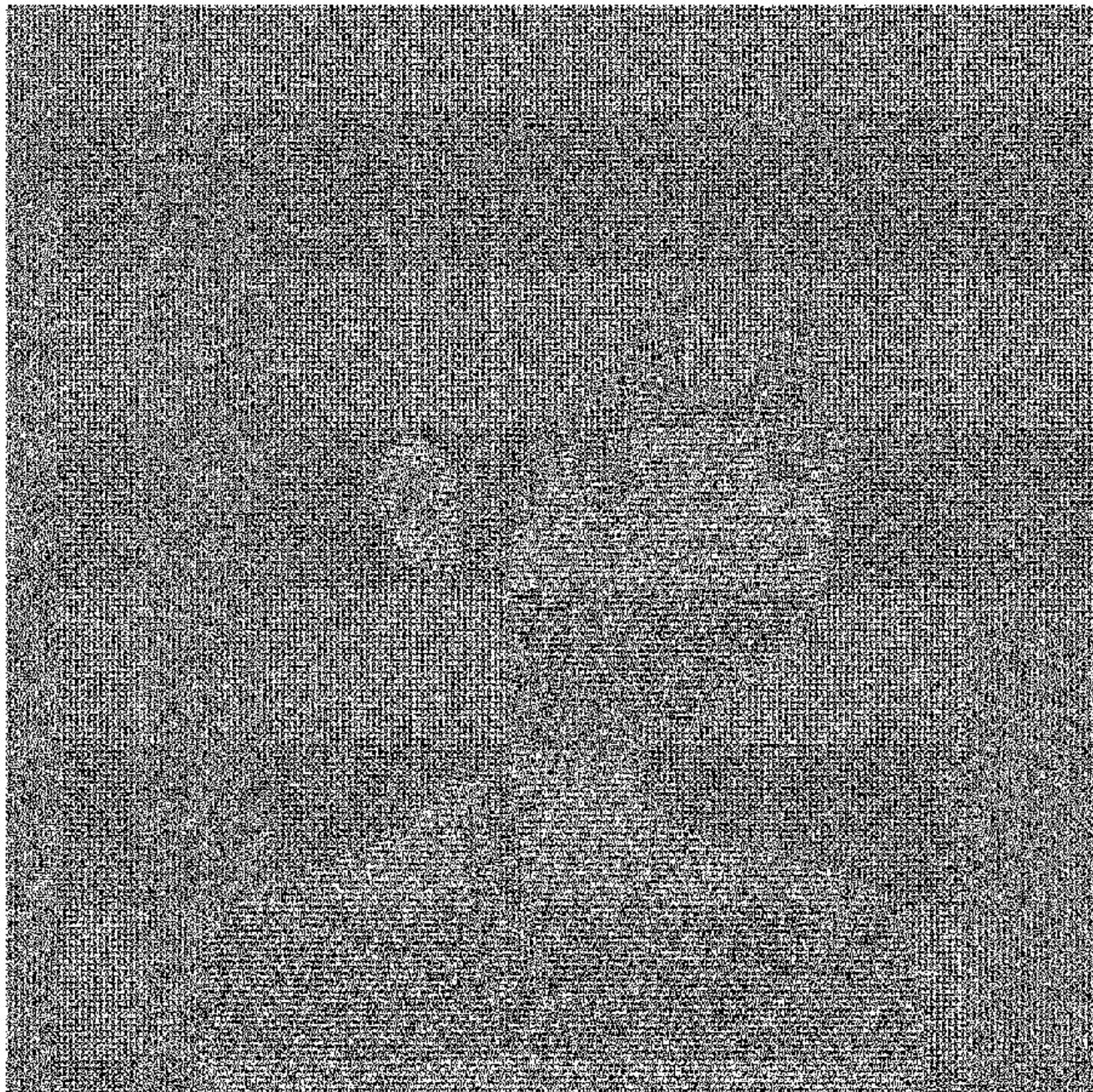
子图1



子图二:



(子图二



合并复原图：



## 彩色图像的可视密钥分享方案

### 原理

将彩色图像的每个分量当做一张图片来处理，即把一张彩色图像看做红、绿、蓝三个分量上的三张图片。

对每一张图片按照 灰度图像 进行半色调处理，然后对每一张图片进行信息分存。R、G、B分量分别分存到两张子图中，最后将得到的子图进行合并可以得到两张彩色子图。

# 实现

代码如下：

```
function RGB_images()
path = input("请输入要加密的图片：", 's');
origin = imread(path);

% 显示原始图像
figure(1);
imshow(origin);
title('原图');

% 分别处理彩色通道
[R, G, B] = splitRGB(origin);

% 对每个通道进行半色调处理并进行信息分存
[image1_R, image2_R] = divideImage(error_diffusion_dithering(R));
[image1_G, image2_G] = divideImage(error_diffusion_dithering(G));
[image1_B, image2_B] = divideImage(error_diffusion_dithering(B));

% 得到两张彩色子图像
image1_rgb = mergeRGB(image1_R, image1_G, image1_B);
figure;
imshow(image1_rgb);
title('子图1');

image2_rgb = mergeRGB(image2_R, image2_G, image2_B);
figure;
imshow(image2_rgb);
title('子图2');

result = mergeImages(image1_rgb, image2_rgb);
figure(6);
imshow(result);
title('复原图像');

end

function [R, G, B] = splitRGB(rgbImage)
% 分离 R、G、B 通道
R = rgbImage(:, :, 1);
G = rgbImage(:, :, 2);
B = rgbImage(:, :, 3);
end

function rgbImage = mergeRGB(R, G, B)
% 获取通道图像的大小
[rows, cols] = size(R);

% 创建一个空白的 RGB 图像
rgbImage = zeros(rows, cols, 3);

% 将分量放置在相应的通道位置上
rgbImage(:, :, 1) = R; % 红色通道
rgbImage(:, :, 2) = G; % 绿色通道
rgbImage(:, :, 3) = B; % 蓝色通道
```

end

```
function image = error_diffusion_dithering(gray)
```

```
Size=size(gray);
```

```
x=Size(1);
```

```
y=Size(2);
```

```
for m=1:x
```

```
    for n=1:y
```

```
        if gray(m,n)>127
```

```
            out=255;
```

```
        else
```

```
            out=0;
```

```
        end
```

```
        error=gray(m,n)-out;
```

```
        if n>1 && n<255 && m<255
```

```
            gray(m,n+1)=gray(m,n+1)+error*7/16.0; %右方
```

```
            gray(m+1,n)=gray(m+1,n)+error*5/16.0; %下方
```

```
            gray(m+1,n-1)=gray(m+1,n-1)+error*3/16.0; %左下方
```

```
            gray(m+1,n+1)=gray(m+1,n+1)+error*1/16.0; %右下方
```

```
            gray(m,n)=out;
```

```
        else
```

```
            gray(m,n)=out;
```

```
        end
```

```
    end
```

```
end
```

```
image=gray;
```

end

```
function [image1, image2] = divideImage(image)
```

```
% Initialize
```

```
Size = size(image);
```

```
x = Size(1);
```

```
y = Size(2);
```

```
image1 = zeros(2*x, 2*y);
```

```
image1(:, :) = 255;
```

```
image2 = zeros(2*x, 2*y);
```

```
image2(:, :) = 255;
```

```
% Take image1 as the first
```

```
for i = 1:x
```

```
    for j = 1:y
```

```
        key = randi(3);
```

```
        son_x = 1 + 2 * (i - 1);
```

```
        son_y = 1 + 2 * (j - 1);
```

```
        switch key
```

```
            case 1
```

```
                image1(son_x, son_y) = 0;
```

```
                image1(son_x, son_y + 1) = 0;
```

```
                if image(i, j) == 0
```

```
                    % Original is black
```

```
                    image2(son_x+1, son_y) = 0;
```

```
                    image2(son_x+1, son_y+1) = 0;
```

```
                else
```

```
                    % Original is white
```

```
                    image2(son_x, son_y+1) = 0;
```

```
                    image2(son_x+1, son_y+1) = 0;
```

```

        end
    case 2
        image1(son_x, son_y) = 0;
        image1(son_x+1, son_y+1) = 0;
        if image(i, j) == 0
            % Original is black
            image2(son_x, son_y+1) = 0;
            image2(son_x+1, son_y) = 0;
        else
            % Original is white
            image2(son_x, son_y) = 0;
            image2(son_x+1, son_y) = 0;
        end
    case 3
        image1(son_x, son_y) = 0;
        image1(son_x+1, son_y) = 0;
        if image(i, j) == 0
            % Original is black
            image2(son_x, son_y+1) = 0;
            image2(son_x+1, son_y+1) = 0;
        else
            % Original is white
            image2(son_x, son_y) = 0;
            image2(son_x, son_y+1) = 0;
        end
    end
end
end
end

function mergedImage = mergeImages(image1, image2)
    % 获取图像尺寸
    Size = size(image1);
    x = Size(1);
    y = Size(2);

    % 初始化合并后的图像
    mergedImage = zeros(x, y, 3);

    % 合并 R 通道
    mergedImage(:, :, 1) = image1(:, :, 1) & image2(:, :, 1);
    % 合并 G 通道
    mergedImage(:, :, 2) = image1(:, :, 2) & image2(:, :, 2);
    % 合并 B 通道
    mergedImage(:, :, 3) = image1(:, :, 3) & image2(:, :, 3);
end

```

流程如下：

1. 输入要加密的彩色图像的路径。
2. 使用 imread 函数读取原始图像。
3. 使用 splitRGB 函数将原始图像分离成红色通道（R）、绿色通道（G）和蓝色通道（B）的三个单独的图像。
4. 使用 error\_diffusion\_dithering 函数将每个通道的图像转换为半色调图像。
5. 使用 divideImage 函数将每个通道的半色调图像分成两个子图像。



6. 使用 mergeRGB 函数将红、绿、蓝通道的两个子图像合并为彩色子图像。
7. 使用 mergeImages 函数将两个彩色子图像合并为复原图像。

## 实验结果

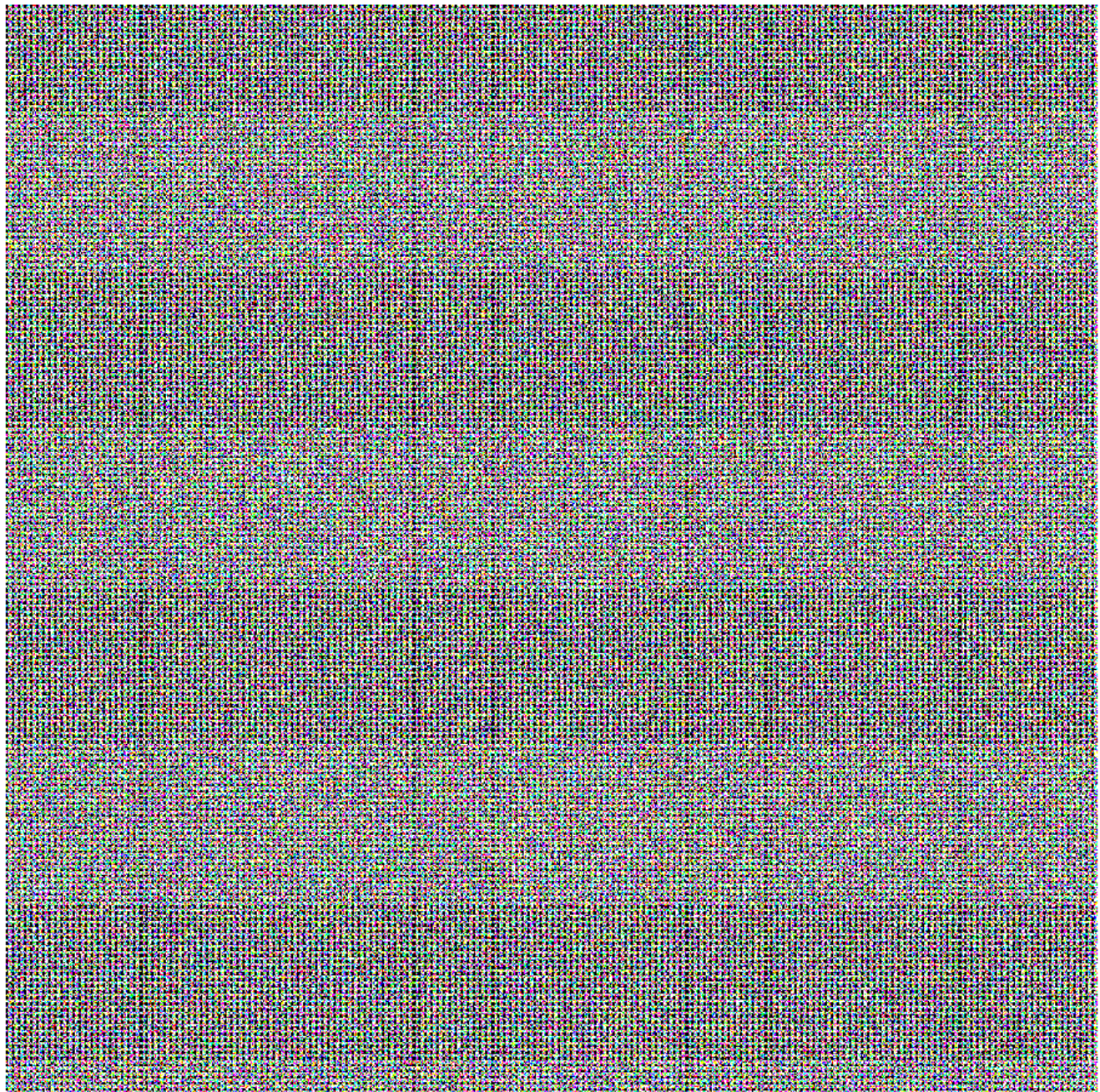
原图：



子图一：



子图1



子图二:

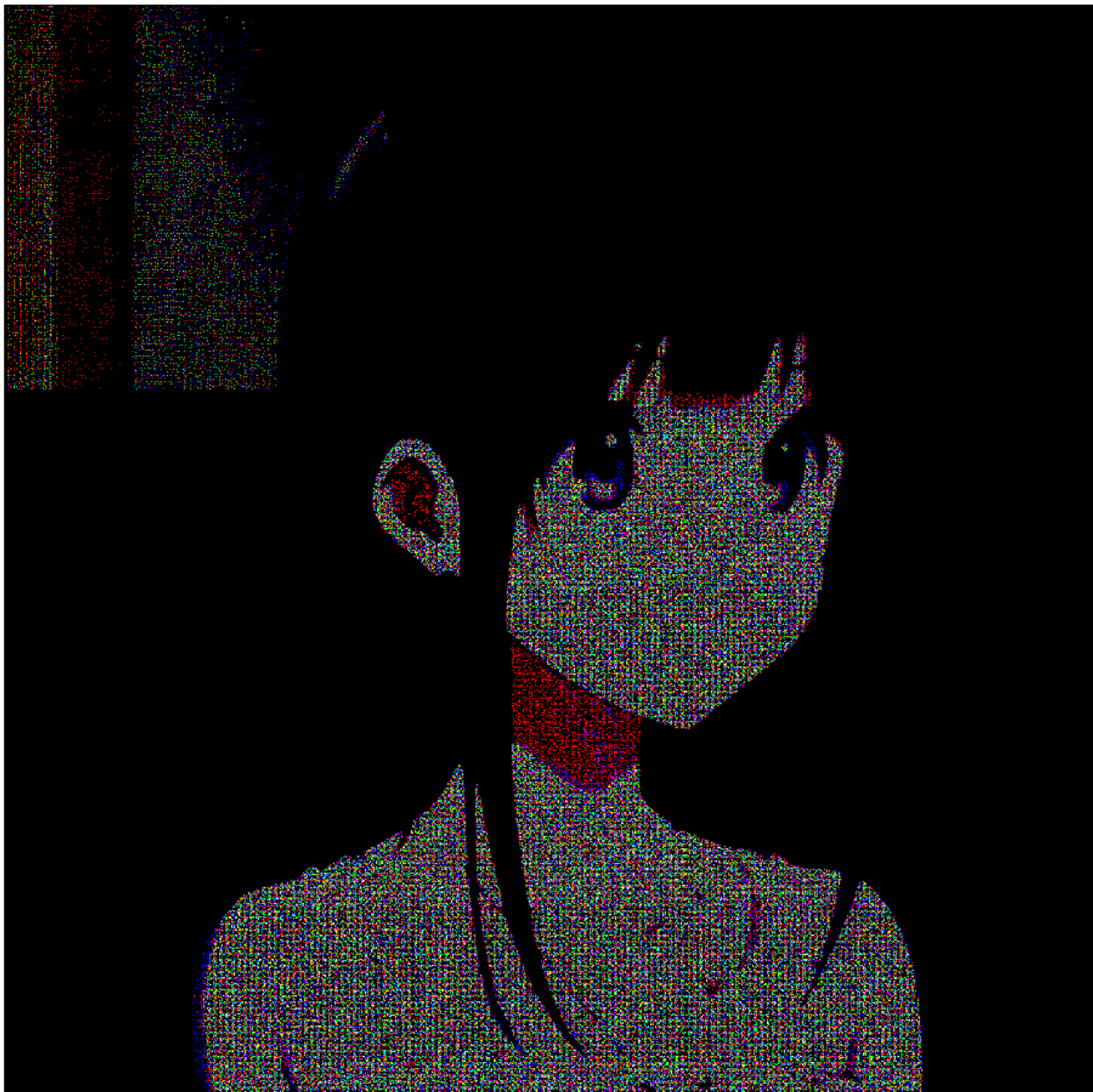


子图2



复原图：





## 叠像术

### 原理

原始图像像素点可能为黑或白，分存原始子图1像素点可能为黑或白，分存原始子图2像素点也可能为黑或白。

因此有下面八种情况：

原始图像	分存子图1	分存子图2
黑	黑	黑
黑	黑	白
黑	白	黑
黑	白	白
白	黑	黑
白	黑	白
白	白	黑
白	白	白

分存方案：

原图	分存于图1	分存于图2	于图1	于图2	合并
黑	黑	黑			
黑	黑	白			
黑	白	黑			
黑	白	白			

# 实现

代码如下：

```
function Image_covering_bw()

path = input("请输入要加密的图片: ','s');
origin = imread(path);

path = input("请输入分存图1: ','s');
origin1 = imread(path);

path = input("请输入分存图2: ','s');
origin2 = imread(path);

% 定义目标大小
target_size = [256, 256];
% 裁剪和缩放图像以适应目标大小
```



```

origin = imresize(origin, target_size);
figure(1);
imshow(origin);
title('原');

origin1 = imresize(origin1, target_size);
figure(2);
imshow(origin1);
title('一');

origin2 = imresize(origin2, target_size);
figure(3);
imshow(origin2);
title('二');

[image1,image2] = divide(origin,origin1,origin2);
figure(4);
imshow(image1);
title('11');

figure(5);
imshow(image2);
title('22');

result = merge(image1,image2);
figure(6);
imshow(result);
title('33');
end

function [image1,image2] = divide(origin,origin1,origin2)
%init
Size=size(origin);
x=Size(1);
y=Size(2);
image1=zeros(2*x,2*y);
image1(:,:)=255;
image2=zeros(2*x,2*y);
image2(:,:)=255;

%take image1 as first
for i = 1:x
    for j = 1:y
        key = randi(4);
        son_x=1+2*(i-1);
        son_y=1+2*(j-1);
        switch key
            case 1
                if origin(i,j)==0 && origin1(i,j)==0 && origin2(i,j)==0 %黑 黑 黑
                    image1(son_x,son_y)=0;
                    image1(son_x+1,son_y)=0;
                    image1(son_x+1,son_y+1)=0;

                    image2(son_x,son_y+1)=0;
                    image2(son_x+1,son_y)=0;
                    image2(son_x+1,son_y+1)=0;

```

白

```
elseif origin(i,j)==0 && origin1(i,j)==0 && origin2(i,j)~=0 %黑 黑

    image1(son_x,son_y+1)=0;
    image1(son_x+1,son_y)=0;
    image1(son_x+1,son_y+1)=0;

    image2(son_x,son_y)=0;
    image2(son_x,son_y+1)=0;
```

黑

```
elseif origin(i,j)==0 && origin1(i,j)~=0 && origin2(i,j)==0 %黑 白

    image1(son_x+1,son_y)=0;
    image1(son_x+1,son_y+1)=0;

    image2(son_x,son_y)=0;
    image2(son_x,son_y+1)=0;
    image2(son_x+1,son_y)=0;
```

白

```
elseif origin(i,j)==0 && origin1(i,j)~=0 && origin2(i,j)~=0 %黑 白

    image1(son_x+1,son_y)=0;
    image1(son_x+1,son_y+1)=0;

    image2(son_x,son_y)=0;
    image2(son_x,son_y+1)=0;
```

黑

```
elseif origin(i,j)~=0 && origin1(i,j)==0 && origin2(i,j)==0 %白 黑

    image1(son_x,son_y)=0;
    image1(son_x,son_y+1)=0;
    image1(son_x+1,son_y)=0;

    image2(son_x,son_y)=0;
    image2(son_x,son_y+1)=0;
    image2(son_x+1,son_y)=0;
```

白

```
elseif origin(i,j)~=0 && origin1(i,j)==0 && origin2(i,j)~=0 %白 黑

    image1(son_x,son_y)=0;
    image1(son_x,son_y+1)=0;
    image1(son_x+1,son_y)=0;

    image2(son_x,son_y)=0;
    image2(son_x,son_y+1)=0;
```

白

```
elseif origin(i,j)~=0 && origin1(i,j)~=0 && origin2(i,j)~=0 %白 白

    image1(son_x,son_y+1)=0;
    image1(son_x+1,son_y+1)=0;

    image2(son_x,son_y)=0;
    image2(son_x,son_y+1)=0;
```

黑

```
elseif origin(i,j)~=0 && origin1(i,j)~=0 && origin2(i,j)==0 %白 白

    image1(son_x+1,son_y)=0;
    image1(son_x+1,son_y+1)=0;
```

```
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

```
end
```

```
case 2
```

```
if origin(i,j)==0 && origin1(i,j)==0 && origin2(i,j)==0 %黑 黑 黑
```

```
image1(son_x,son_y)=0;  
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y)=0;
```

```
image2(son_x,son_y)=0;  
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y+1)=0;
```

```
elseif origin(i,j)==0 && origin1(i,j)==0 && origin2(i,j)~=0 %黑 黑
```

```
image1(son_x,son_y)=0;  
image1(son_x+1,son_y)=0;  
image1(son_x+1,son_y+1)=0;
```

```
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y+1)=0;
```

```
elseif origin(i,j)==0 && origin1(i,j)~=0 && origin2(i,j)==0 %黑 白
```

```
image1(son_x,son_y)=0;  
image1(son_x+1,son_y)=0;
```

```
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

```
elseif origin(i,j)==0 && origin1(i,j)~=0 && origin2(i,j)~=0 %黑 白
```

```
image1(son_x,son_y)=0;  
image1(son_x+1,son_y)=0;
```

```
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y+1)=0;
```

```
elseif origin(i,j)~=0 && origin1(i,j)==0 && origin2(i,j)==0 %白 黑
```

```
image1(son_x,son_y)=0;  
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y+1)=0;
```

```
image2(son_x,son_y)=0;  
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y+1)=0;
```

```
elseif origin(i,j)~=0 && origin1(i,j)==0 && origin2(i,j)~=0 %白 黑
```

```
image1(son_x,son_y)=0;  
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y+1)=0;
```

```
image2(son_x,son_y)=0;
```

```
image2(son_x+1,son_y+1)=0;
```

```
elseif origin(i,j)~=0 && origin1(i,j)~=0 && origin2(i,j)~=0 %白 白
```

```
image1(son_x,son_y)=0;  
image1(son_x,son_y+1)=0;
```

```
image2(son_x,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

```
elseif origin(i,j)~=0 && origin1(i,j)~=0 && origin2(i,j)==0 %白 白
```

```
image1(son_x,son_y)=0;  
image1(son_x,son_y+1)=0;
```

```
image2(son_x,son_y)=0;  
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y+1)=0;
```

```
end
```

```
case 3
```

```
if origin(i,j)==0 && origin1(i,j)==0 && origin2(i,j)==0 %黑 黑 黑
```

```
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y)=0;  
image1(son_x+1,son_y+1)=0;
```

```
image2(son_x,son_y)=0;  
image2(son_x+1,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

```
elseif origin(i,j)==0 && origin1(i,j)==0 && origin2(i,j)~=0 %黑 黑
```

```
image1(son_x,son_y)=0;  
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y+1)=0;
```

```
image2(son_x+1,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

```
elseif origin(i,j)==0 && origin1(i,j)~=0 && origin2(i,j)==0 %黑 白
```

```
image1(son_x,son_y)=0;  
image1(son_x,son_y+1)=0;
```

```
image2(son_x,son_y)=0;  
image2(son_x+1,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

```
elseif origin(i,j)==0 && origin1(i,j)~=0 && origin2(i,j)~=0 %黑 白
```

```
image1(son_x,son_y)=0;  
image1(son_x,son_y+1)=0;
```

```
image2(son_x+1,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

黑

```
elseif origin(i,j)~=0 && origin1(i,j)==0 && origin2(i,j)==0 %白 黑
```

```
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y)=0;  
image1(son_x+1,son_y+1)=0;
```

```
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

白

```
elseif origin(i,j)~=0 && origin1(i,j)==0 && origin2(i,j)~=0 %白 黑
```

```
image1(son_x,son_y)=0;  
image1(son_x+1,son_y)=0;  
image1(son_x+1,son_y+1)=0;
```

```
image2(son_x+1,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

白

```
elseif origin(i,j)~=0 && origin1(i,j)~=0 && origin2(i,j)~=0 %白 白
```

```
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y)=0;
```

```
image2(son_x+1,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

黑

```
elseif origin(i,j)~=0 && origin1(i,j)~=0 && origin2(i,j)==0 %白 白
```

```
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y)=0;
```

```
image2(son_x,son_y)=0;  
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y)=0;
```

```
end
```

```
case 4
```

```
if origin(i,j)==0 && origin1(i,j)==0 && origin2(i,j)==0 %黑 黑 黑
```

```
image1(son_x,son_y)=0;  
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y+1)=0;
```

```
image2(son_x,son_y)=0;  
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y)=0;
```

白

```
elseif origin(i,j)==0 && origin1(i,j)==0 && origin2(i,j)~=0 %黑 黑
```

```
image1(son_x,son_y)=0;  
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y)=0;
```

```
image2(son_x,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```



黑

```
elseif origin(i,j)==0 && origin1(i,j)~=0 && origin2(i,j)==0 %黑 白
```

```
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y)=0;  
  
image2(son_x,son_y)=0;  
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y+1)=0;
```

白

```
elseif origin(i,j)==0 && origin1(i,j)~=0 && origin2(i,j)~=0 %黑 白
```

```
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y)=0;  
  
image2(son_x,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

黑

```
elseif origin(i,j)~=0 && origin1(i,j)==0 && origin2(i,j)==0 %白 黑
```

```
image1(son_x,son_y)=0;  
image1(son_x+1,son_y)=0;  
image1(son_x+1,son_y+1)=0;  
  
image2(son_x,son_y)=0;  
image2(son_x+1,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

白

```
elseif origin(i,j)~=0 && origin1(i,j)==0 && origin2(i,j)~=0 %白 黑
```

```
image1(son_x,son_y+1)=0;  
image1(son_x+1,son_y)=0;  
image1(son_x+1,son_y+1)=0;  
  
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y+1)=0;
```

白

```
elseif origin(i,j)~=0 && origin1(i,j)~=0 && origin2(i,j)~=0 %白 白
```

```
image1(son_x+1,son_y)=0;  
image1(son_x+1,son_y+1)=0;  
  
image2(son_x,son_y+1)=0;  
image2(son_x+1,son_y+1)=0;
```

黑

```
elseif origin(i,j)~=0 && origin1(i,j)~=0 && origin2(i,j)==0 %白 白
```

```
image1(son_x+1,son_y)=0;  
image1(son_x+1,son_y+1)=0;  
  
image2(son_x,son_y)=0;  
image2(son_x+1,son_y)=0;  
image2(son_x+1,son_y+1)=0;
```

```
end
```

```
end
```

```
end
```

```
end
```

```
end
```

```

function image = merge(image1,image2)
Size=size(image1);
x=Size(1);
y=Size(2);
image=zeros(x,y);
image(:,:)=255;

for i=1:x
    for j=1:y
        image(i,j)=image1(i,j)&image2(i,j);
    end
end
end

```

流程如下：

#### 1. 主函数 Image\_covering\_bw:

- 获取要加密的原始图像路径、分存图1的路径和分存图2的路径。
- 读取这三张图像。
- 定义了目标大小为 256\*256 像素，并裁剪并缩放到目标大小。
- 调用 divide 函数对裁剪并缩放后的原始图像、分存图1和分存图2进行处理，得到两张加密后的图像。
- 调用 merge 函数对这两张加密后的图像进行合并，得到最终的加密结果。

#### 2. divide 函数:

- 接受裁剪并缩放后的原始图像 origin、分存图1 origin1 和分存图2 origin2 作为输入。
- 首先初始化两张空白图像 image1 和 image2，大小是原图的两倍，并且像素值全部初始化为白色（255）。
- 对原图中的每个像素进行遍历，随机选择一个分存图，并根据原图、分存图1 和分存图2 中每个像素的像素值，决定将黑色（0）像素放在哪张分存图中。
- 返回处理后的两张图像 image1 和 image2。

#### 3. merge 函数:

- 接受两张分存图像 image1 和 image2 作为输入。
- 遍历这两张图像的每个像素，并将它们进行逻辑与（AND）操作，得到最终的加密结果。
- 返回合并后的加密结果图像。

## 实验结果

加密图：



分存图一：



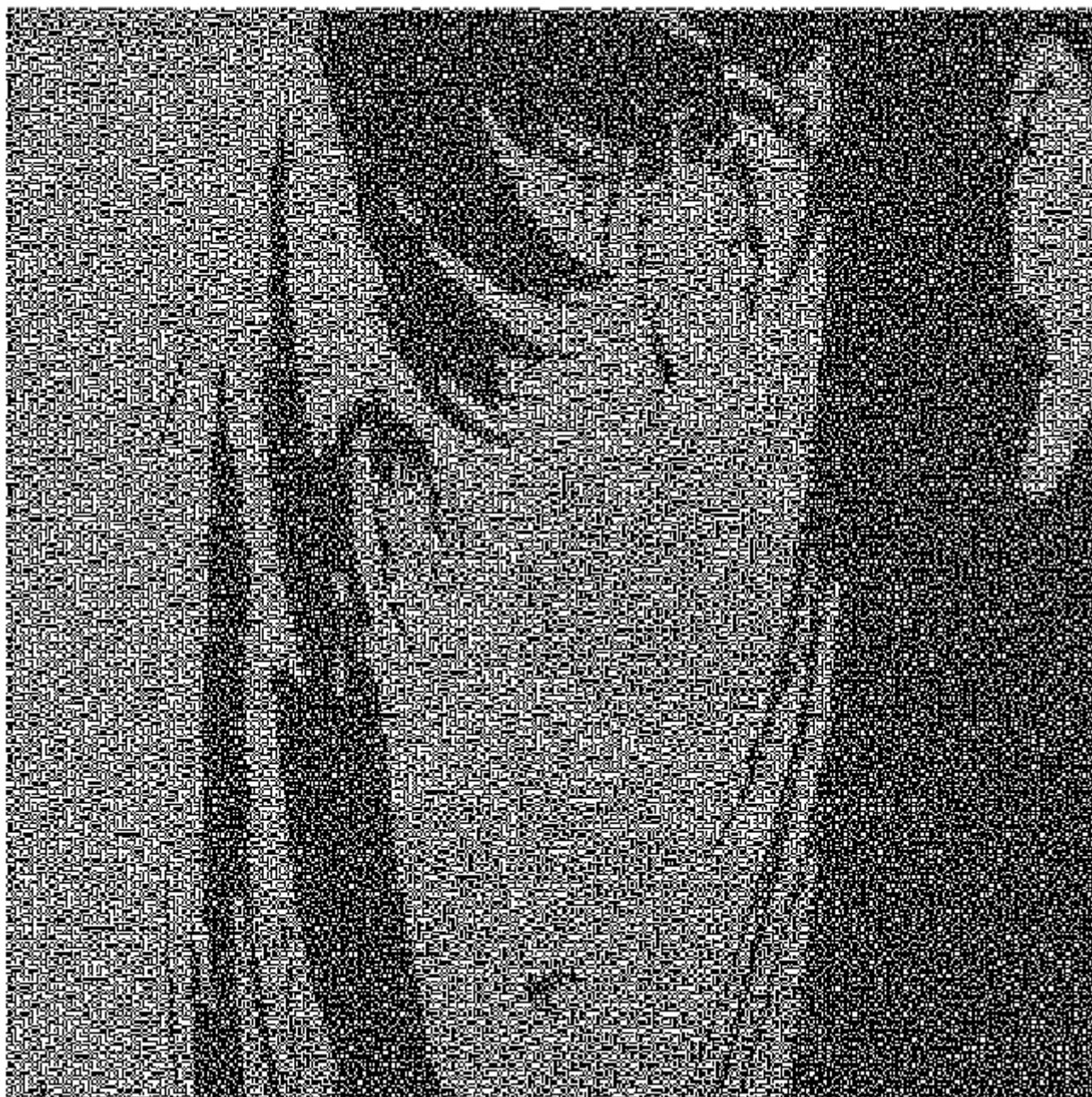
分存图二：



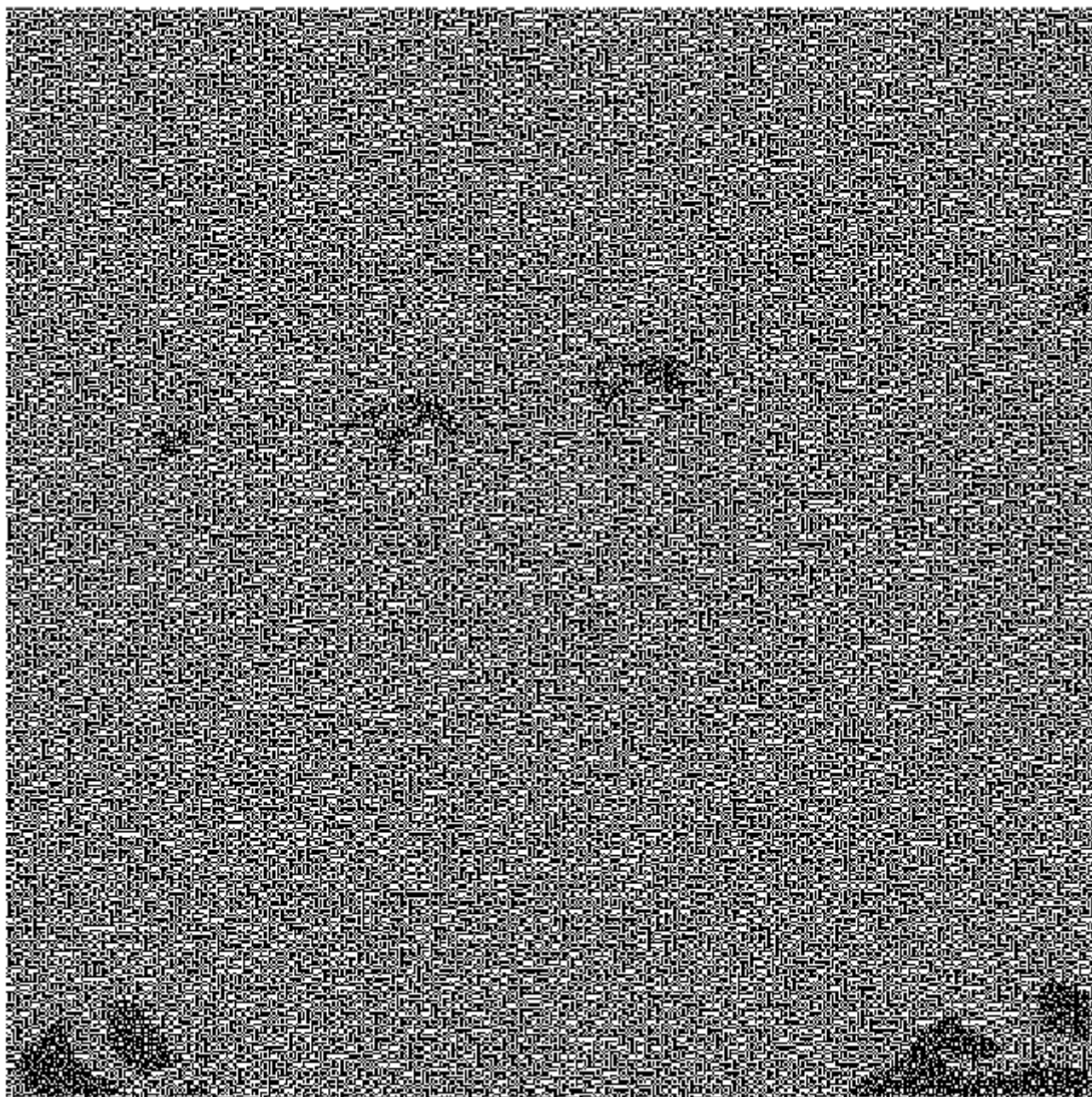
## 二值化图像



伪装图一：



伪装图二：



复原图：



## 灰度和彩色图像的叠像术

与可视密钥类似的，灰度图像需要先进行半色调化处理，对彩色图像则需要把每个分量当做一张图片来处理。

## 小结

在课堂上学习了可视密钥分享方案和叠像术，在实际实验的时候其实能够发觉自己学得还不够扎实，在课堂上学的理论其实只学了个大似。

但亲自完成实验后，课上所学的知识也确实巩固提高了，多少有了一些理解，而且亲自做下来还能发现这几个实现还挺有趣的。

同时，也是第一次在 `matlab` 上做实验，自学了一些`matlab`的操作等等。