第四章 图像信息伪装技术

随着信息媒体数字化和计算机网络通信技术的发展,信息的快速准确传递成为可能,弹指之间,大洋此岸的消息就可以到达大洋彼岸。可是,在消息“跨越千山万水”的瞬间,还存在着严重的安全隐患,一些别有用心的人或团体可能在消息的传输过程中采取窃取、修改或破坏等手段攻击秘密信息,给消息的收发双方造成严重的危害。

为了解决这个问题,人们试图先用密钥加密信息成密文,然后将密文在网络上传输,接收方收到后再使用密钥将密文解密成原消息,这样就避免了别有用心者窃取、修改秘密信息。这种方法的缺陷是:加密后的密文在网络上传输似乎更能引起人们的兴趣,使他们更热衷于对密文进行解密和攻击。而且,这样一来,别有用心者还可以破坏秘密信息,使接收方无法获得准确完整的信息。

于是,人们又想到了另一种技术——伪装技术,即将秘密信息隐藏在不易被人怀疑的普通文件(即载体文件,这里主要用图像作为载体文件)中,使秘密信息不易被别有用心者发现,当然他们就不易对消息进行窃取、修改和破坏,从而保证了消息在网络上传输的安全性。为了增加安全性,人们通常将加密和伪装这两种技术结合起来使用。

从本章开始,我们将具体接触到许多数字信息的隐写术。为了与后面数字水印技术相区别,我们也将隐写术称为信息隐秘技术。本章主要阐述图像信息作为秘密信息的隐藏,即图像降级隐写。事实上,将一幅图像隐藏到另一幅图像中,本身也就可以看做是一个数字水印系统。之所以将其单独抽出并且放在信息隐秘技术中,仅仅是为了与后面以高斯白噪声为模板的数字水印系统略加区分。而且我们认为,将一幅图像藏于另一幅图像中,秘密图像应该是更为关注的,这较符合信息隐秘的思维特点。当然,如果要将载体作为通信的重点,那就是数字水印所关注的了。在整个介绍信息隐秘技术部分,为了统一起见,我们约定以下名词:称需要隐秘的信息为秘密信息(secret),秘密信息隐藏的媒介叫做载体(cover),隐藏后的结果叫做隐蔽载体(stego-cover),如图 4.1 所示。

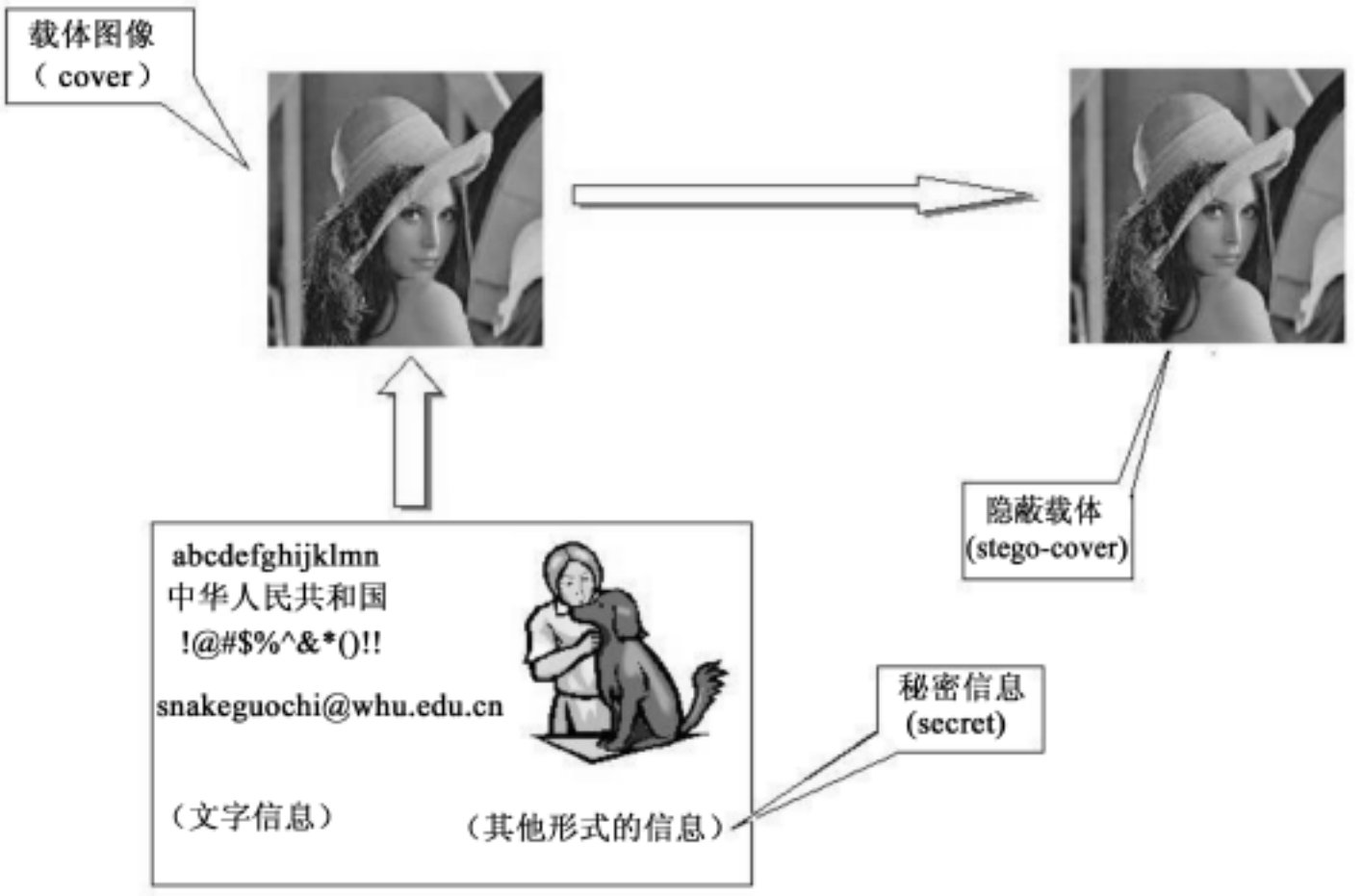

图 4.1　信息隐秘技术的名词约定

4.1　图像降级

在多级安全操作系统中，主体(进程、用户) 和客体(文件、数据库等) 都被分配了一个特定的安全级别，主体对客体的读写一般应满足以下两个规则:

规则 1: 主体只能向下读，不能向上读。

规则 2: 主体只能向上写，不能向下写。

对于第一个规则，我们很容易理解，即主体通常仅允许读取比其安全级别低的客体，而对于安全级别比其高的客体则无法读取。对于第二个规则，我们可以这样理解，如果主体向比其安全级别低的客体写信息，则信息由主体的安全级别变成了客体的安全级别，那么较低安全级别的主体也就可以访问此信息了。因此，主体只能向比其安全级别高的客体写信息，才能保证秘密信息的安全。

我们通常所说的信息降级，就是通过将秘密信息嵌入较低安全级别的客体中，破坏了第二个规则，从而使机密的信息看上去不再机密。在伪装技术中我们经常要这样做，以使秘密信息被伪装成为低级别的信息，不易被发现。图像降级就是伪装技术的一个应用。

图像降级用于秘密的交换图像。下面举例来说明什么是图像降级。

A 方需要将一张重要的图像 m 通过网络传递给 B，但是有一个第三者 C 准备破

坏这次传输，对 A 方来说，如何使 m 安全传输给 B 是一个令人头痛的问题，最终 A 方选择了这样一个方法，如图 4.2 所示。

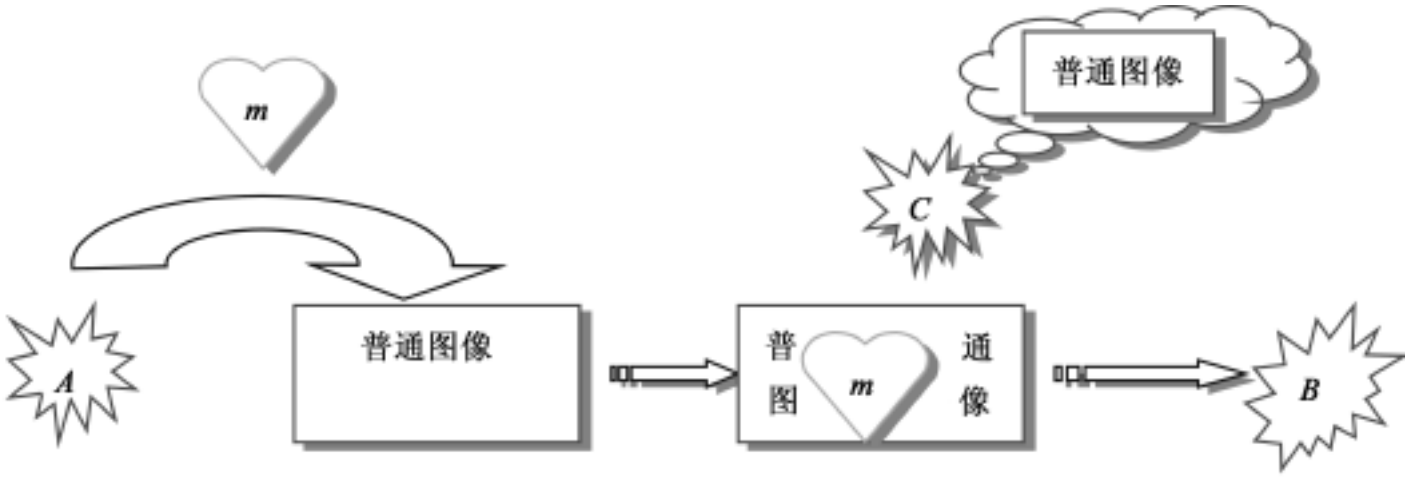

图 4.2　图像降级的直观解释

A 方所用的就是图像降级，普通图像（客体）的安全级别低，而 A 方（主体）的安全级别高，A 方将其秘密信息（m 图）写入普通图像中，破坏了第二个规则，从而降低了 m 图的安全级别，使 C 方和其他人都可以看到，然而在 C 方和其他不知情的人看来，传输的只是一幅普通图像，他们不会注意这张普通图像，因此也就不会看到 m 图，这样，A 方就成功地将秘密信息 m 图传输给了 B 方。

由上面的例子，我们知道了什么是图像降级，图像降级是替换系统中的特殊情况，其中，秘密信息和载体都是图像。在下一节中，我们将讲述将秘密图像信息写入载体图像的几种方法。

4.2　简单的图像信息伪装技术

本节将讲述秘密图像信息嵌入载体图像的一种最简单的方法——直接 4bit 替换法以及其改进思路，增加嵌入的容量和提高图像对恶意攻击的鲁棒性。

4.2.1　直接 4bit 替换法

所谓直接 4bit 替换法，就是直接用秘密图像像素值的高 4bit 去替换载体图像像素值的低 4bit，下面来做这个实验。

在实验中我们采用图像处理中常用的 lenna 图像做为载体图像（如图 4.3 所示），图像的大小为 256× 256，选取 woman 图像做为秘密图像（如图 4.4 所示），图像大小也为 256× 256。

我们将 lenna 作为载体，woman 作为秘密信息进行信息隐秘，首要的问题是找到隐秘的空间，也就是载体的冗余空间。图 4.6 是将 lenna 各像素低 4bit 清 0 后的图像。不加严格的分析，我们可以认为图 4.3 与图 4.6 无感观上的差异（事实上有些时

图 4.3　lenna 原图

图 4.4　woman 原图

候是不能这样认为的,后文有分析)。这样一来,图像像素的低 4bit 就构成了冗余空间。对于载体如此,作为秘密信息的图像也应该如此。所以,我们当然也就不必要将一个秘密图像的像素(8bit)隐藏在两个载体像素的低 4bit 上,而只将其像素高 4bit 作为秘密信息嵌入载体,这样可以扩大隐藏容量。除非秘密图像的细节信号非常重要才做完整的隐藏,但那就需要载体的容量是秘密图像的 1 倍才能完成。

去掉低 4bit 的 woman 图如图 4.5 所示,去掉低 4bit 的 lenna 图如图 4.6 所示。

图 4.5　去掉低 4bit 的 woman 图

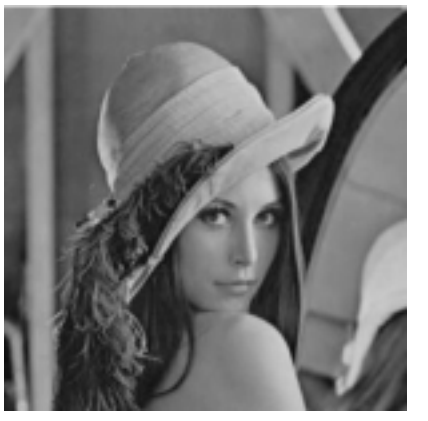

图 4.6　去掉低 4bit 的 lenna 图

编写函数 imagehide.m 完成图像隐藏的实验,函数代码如下:

```
% 文件名: imagehide.m
% 程序员: 王霞仙
% 编写时间: 2004.2.5
% 函数功能: 直接将秘密图像的高 4bit 隐藏在 RGB 载体图像的 R, G, B 层中所选的那一层的低 4bit, 并将秘密图像提取出来, 最后显示。要求载体图像的大小大于等于秘密图像的大小, 且秘密图像是二值或灰度图像
% 输入格式:
% data = imagehide( 'c: \lenna.bmp', 'c: \woman.bmp', 'c: \mix.bmp', 'bmp',3)
% 参数说明:
```

```
%  cover 是载体图像的地址
%  message 是秘密图像的地址
%  goleimage 是隐藏后图像的地址
%  permission 是图像的类型
% level 是作为载体的具体层, R 为 1, G 为 2, B 为 3
% data 是隐藏后图像的矩阵
function  data = imagehide( cover, message, goleimage, permission, level)
% 提取图像信息并分层
cover  =  imread( cover, permission) ;
data = cover;
msg = imread( message, permission) ;
[ row, col]  = size( cover) ;
cover1 = cover(：,：, level) ;
% 置载体图像 R 层的低 4bit 为 0
for i = 1：row
        for j = 1：col /3
              cover1( i, j)  = bitand( cover1( i, j) , 240) ;
        end
end
% 置秘密图像的低 4bit 为 0
takemsg4 = bitand( msg, 240) ;
% 将秘密图像的高 4bit 右移 4 位
shiftmsg4 = bitshift( takemsg4,  - 4) ;
% 图像隐藏
for i = 1：row
        for j = 1：col/3
              cover1( i, j)  = bitor( cover1( i, j) , shiftmsg4( i, j) ) ;
        end
end
% 写回并保存
data(：,：, level)  = cover1;
imwrite( data, goleimage, permission) ;
% 提取秘密图像信息, 检测隐藏效果
data = imread( goleimage, permission) ;
[ row, col]  = size( data) ;
A = data(：,：, level) ;
```

```
for i = 1: row
    for j = 1: col/3
        A( i, j)  = bitand( A( i, j) , 15) ;
    end
end
A = bitshift( A, 4) ;
% 显示结果
subplot( 221) , imshow( cover) ; title( '载体图像') ;
subplot( 222) , imshow( message) ; title( '秘密图像') ;
subplot( 223) , imshow( data) ; title( '隐藏后的图像') ;
subplot( 224) , imshow( A) ; title( '提取的秘密图像') ;
```

由于我们选用的载体是一个 RGB 图像(256×256×3),秘密图像仅仅是一幅灰度图像(256×256),所以将秘密图像隐藏到载体的哪一层是我们关心的问题。图 4.7 是将秘密图像分别藏于 RGB 载体图像的 R, G, B 层所显示的结果。

图 4.7　信息分别隐藏于 R, G, B 层的比较图

按照前面我们的假设推定，图 4.7 中隐藏后的图像应该和原载体图像的差别不明显，但通过观察我们发现，图 4.7 中的隐藏后的图像和原载体图像还是有一些差别的，例如帽子上部的一些地方（圈内部分）。这是由于嵌入的信息容量大造成的。并且在图 4.7 中我们看到，秘密图像隐藏于不同的层后与原载体图像的差别大小是不同的。对于隐藏在 R，G，B 各层的隐蔽载体，图中圈内的部分分别产生泛红、泛绿和泛蓝的现象；秘密图像隐藏在 R 层和 G 层后与原载体图像的差别比隐藏在 B 层时的差别要大。三者都在不同程度上对原始图像造成了一定的破坏。

根据 RGB 图像像素的原理，一个像素的颜色由 R，G，B 三个分量值确定。考虑到第一章中我们阐述的 RGB 颜色模型，将秘密图像隐藏在一层中，容易导致该点的色彩向相应的坐标上发生绝对偏移，从而使得该像素点的色彩的相应分量突出。所以，我们不能笼统地认为图像隐藏在某层比较好而隐藏在某层不好，这是因为对于具体的某个像素点其哪个颜色分量突出是不确定的。但是，我们可以通过改进算法来限制这种颜色沿相应坐标的绝对偏移。

例如，可将秘密图像像素值的高 4bit 分别藏于载体图像 R，G，B 层像素值的最低位或次低位，即可将秘密图像的高 2bit 藏于 R 层，另外 2bit 分别藏于 G 层和 B 层，此时像素色彩的改变就不是沿一个坐标方向而改变，而是在整个 RGB 空间中发生偏移，改变后的颜色坐标点与改变前的颜色坐标点的距离（数学上的范数）比单纯在一个分量上改变的两点距离要小，这样对载体图像的影响会更小。在实际应用中，还应该考虑隐藏的鲁棒性等问题。RGB 像素的等价修改如图 4.8 所示。

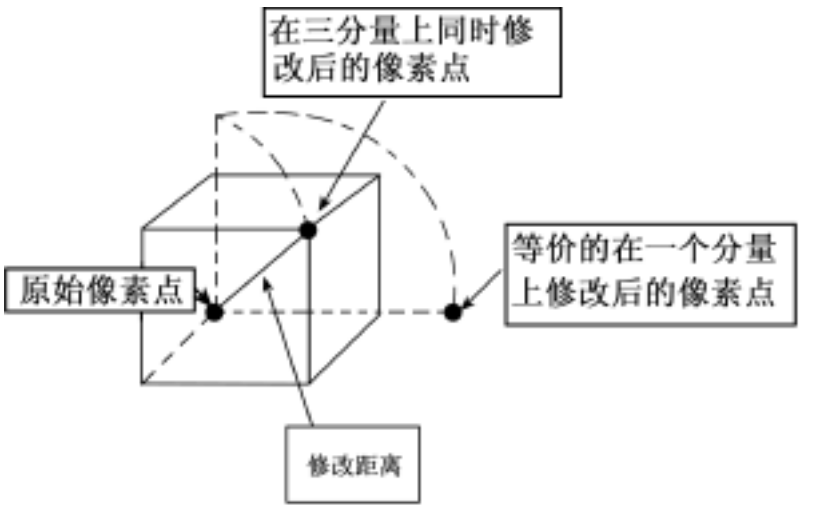

图 4.8　RGB 像素的等价修改

4.2.2　对第 4bit 的考察

在 4.2.1 节中我们看到，直接替换 4bit 后，图像还是有一些变化的，也就是说，替换容量大使图像的保真度降低（可通过实验看到替换 3bit 的效果比替换 4bit 的效果要好）。为了解决这个问题，我们需要采取一些其他技术。

我们仍然用 lenna 图像做为载体图像、woman 图像做为秘密信息图像。将这两个图分别分割为 8× 8 的小块,我们对每一个 8× 8 的小块内的像素采用相同的替换策略,而对块与块则不尽相同,要具体情况具体分析。

其假设前提是:如果只对图像进行 3bit 替换,是不会对图像的视觉效果造成影响的。事实上,这种假设是可以成立的,如图 4.9 所示。

图 4.9　将信息隐藏于低 3bit 后的图像

因此,对每一图像块,我们先用 woman 图像像素值的高 3bit 去替换 lenna 图像像素值的低 3bit 是没有问题的,至于第 4bit 则要具体分析。首先,我们引入一个相似度的概念,所谓相似度,是指两图像块中同一坐标下的像素中第 4bit 相同的像素数量占一块图像全部像素的比例,表示为:

$$\mu = \frac{s}{64}$$

其中,s 为第 4bit 相同的像素数量,64 为 8× 8 块中的总像素数量。

根据 μ的取值我们来确定该块各像素第 4bit 的隐藏策略。

我们先计算相应块的载体图像与秘密图像在第 4bit 的相似度 μ如果 μ大于某一阈值 T,则可直接用秘密图像的第 4bit 替换载体图像的第 4bit,如果 μ小于阈值 1 - T,则先将秘密图像的第 4bit 取反后再替换,若 μ介于 1 - T 和 T 之间,则不进行替换。当然,要用一个替换表对第 4bit 进行替换或取反替换了的块进行记录,并且将此表也嵌入到载体图像中。编写函数 fourthbitcmp. m 完成记录替换表的实验,函数代码如下:

```
% 文件名: fourthbitcmp. m
% 程序员: 王霞仙
% 编写时间: 2004. 2. 5
% 函数功能: 计算秘密图像和选择的载体图像层,对于第 4bit 的每一个 8× 8 块,哪些可以用秘密图像去替换载体图像,并返回一个替换表 count,要求两个图像都可以整数 8× 8 分块
% 输入格式: count = fourthbitcmp( 'c: \lenna. bmp', 'c: \woman. bmp', 'bmp',3,0. 7)
```

```
% 参数说明:
% cover 是载体图像的地址
% message 是秘密图像的地址
% permission 是图像的类型
% level 是作为载体的具体层。R 为 1, G 为 2, B 为 3
% count 是替换表
% threshold 是阈值
function count = fourthbitcmp( cover, message, permission, level, threshold)
% 提取图像信息并分层
cover = imread( cover, permission) ;
data = cover;
msg = imread( message, permission) ;
cover1 = cover(:,:, level) ;
% 对 cover 和 msg 的第 4bit 进行处理
tempc = cover1;
tempm = msg;
tempc = bitand( tempc, 8) ;
tempm = bitand( tempm, 8) ;
temp = bitxor( tempm, tempc) ;
[ row, col] = size( temp) ;
% 记录图像每个分块的 n 值
k1 = 0;
k2 = 0;
a = row* col/64;
count = zeros( [ 1 a] ) ;
for i = 1: a
    for m = 1: 8
        for n = 1: 8
            if temp( 8* k1 + m, 8* k2 + n) == 0
                count( 1, i) = count( 1, i) + 1;
                end
            end
    end
    k2 = k2 + 1;
    if k2* 8 == col
        k2 = 0;
```

```
            k1 = k1 + 1;
        end
    end
    % 计算每块的 μ值并与阈值进行比较
    count = count /64;
    for i = 1: a
        if count( i)  >= threshold
            count( i)  =1; % 可以替换
        elseif count( i)  <1 - threshold
            count( i)  = - 1; % 取反
        else
            count( i)  =0; % 不能处理
        end
    end
```

表 4.1 为不同阈值下得到的 4bit 的替换表。显然, lenna 和 woman 两幅图像的第 4bit 相关度较低, 当阈值稍大一点则大多数图像块不能对第 4bit 进行处理了, 所以只在仅替换 3bit 的情况下载体破坏不大(如图 4.9 所示) , 而不加考虑的统一替换第 4bit 就容易出现图 4.7 的情况。

表 4.1　不同阈值下的前 10 块的替换情况: 1 为可替换, - 1 为取反, 0 为不处理

T = 0.7	0	1	0	0	0	0	0	0	1	0
T = 0.6	1	1	0	0	0	- 1	0	- 1	1	1
T = 0.5	1	1	1	1	- 1	- 1	- 1	- 1	1	1

在 4.2.1 节中我们讨论了秘密信息应该隐藏于 RGB 图像的哪一层, 这里再补充说明一下, 依据本算法, 在同一阈值下经不同层计算出的替换表中 0 的个数, 个数越少的层越适宜当做载体。当然, 为了简单起见, 也可以不加分块直接计算秘密图像与载体图像 R、G、B 层中哪一层的相似度高, 就选择哪一层为载体。表 4.2 是将 woman 与 lenna 各层进行分块第 4bit 比较后的统计结果, 可以看到 lenna 的各层对于 woman 都不占优势。

表 4.2　同一阈值下各层替换表中 0 的个数

	R	G	B
T = 0.6	774	767	778
T = 0.7	964	964	964

相对于使用 3bit 替换,用这个方法显然可以提高嵌入信息的容量。相对于 4bit 替换,则不会使隐藏的不可见性降低。

4.3 图像置乱

所谓“置乱”,就是将图像的信息次序打乱,将 a 像素移动到 b 像素的位置上,b 像素移动到 c 像素的位置上……使其变换成杂乱无章难以辨认的图像。置乱实际上就是图像的加密,与加密保证安全性不同的是,将置乱的图像作为秘密信息再进行隐藏,可以很大限度地提高隐蔽载体的鲁棒性,所以图像置乱是信息隐藏中常用的一项技术。下面将介绍三种置乱方法。

4.3.1 变化模板形状的图像置乱算法

变化模板形状的图像置乱算法的思想如下:

① 对原图像取一个固定模板,模板中像素位置排列如图 4.10 所示。

② 做一个与原图像模板不同的置乱模板,如图 4.11 所示,在置乱模板中把图像模板中的像素位置按一定次序填入(在图 4.11 的模板中按从上到下、从左到右的次序依次填入)。

③ 将置乱模板中的像素位置再按一定的次序填回到原图像模板中就得到了置乱后的图像模板(图 4.12 的模板是按从左到右、从上到下的次序依次读取置乱模板中像素位置)。

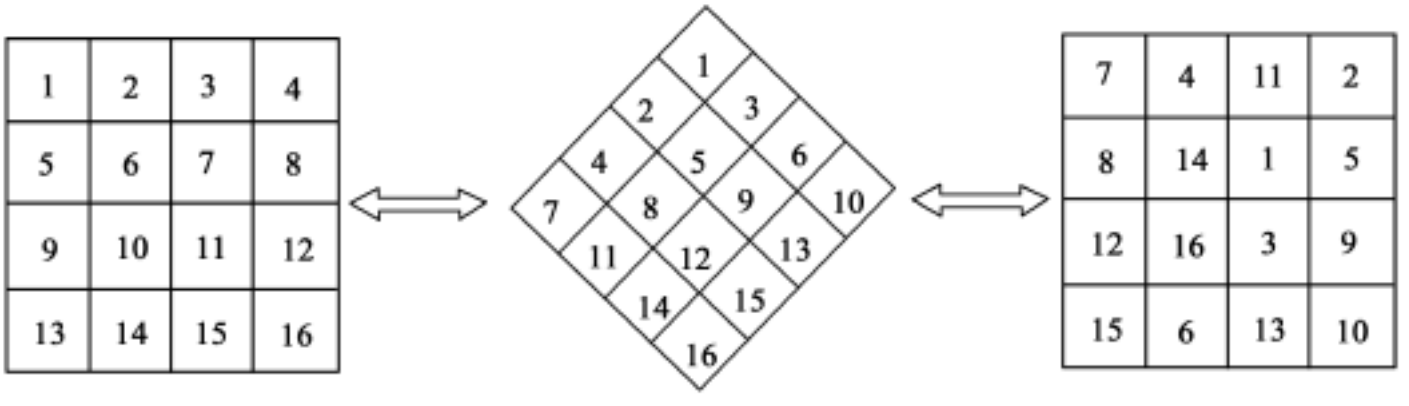

图 4.10 原图像模板　　图 4.11 置乱模板　　图 4.12 置乱后模板

可以发现,这种置乱算法是对合的,即将图 4.12 进行同样的一次变换后就可以得到图 4.10。这给我们编写程序带来了方便。与前面 Zigzag 变换一样,我们也采取查表的方法编写程序。由于我们固定了置乱模板的大小,所以在对图像置乱前我们要对其进行边界修补。如取置乱模板为 32× 32,则要求秘密图像的尺寸为 32× 32,64× 64,128× 128,…。假设一幅图像的尺寸为 32× 31 ,则应该对其增加 1 列数据。32× 32 的置乱变换表如表 4.3 和表 4.4 所示。变换表分为行表和列表,同一坐标下的行列表中的数据结合起来所指示的像素将被置乱到这一坐标下。如置乱行列表

(1,1)坐标下的数据分别为16和17,就表示原图像(16,17)坐标下的像素将被置乱到(1,1)坐标下。

表4.3　　32×32菱形置乱表,行表

16	15	17	14	16	18	13	15	17	19	12	14	16	18	20	11	13	15	17	19	21	11	12	14	16	18	20	22	10	12	13	15
17	19	21	22	9	11	12	14	16	18	20	22	23	8	10	12	13	15	17	19	21	22	24	8	9	11	12	14	16	18	20	22
23	25	7	8	10	12	13	15	17	19	21	22	24	25	6	8	9	11	12	14	16	18	20	22	23	25	26	6	7	8	10	12
13	15	17	19	21	22	24	25	27	5	6	8	9	11	12	14	16	18	20	22	23	25	26	27	5	6	7	9	10	12	13	15
17	19	21	22	24	25	27	28	4	5	7	8	9	11	13	14	16	18	20	22	23	25	26	27	28	4	5	6	7	9	10	12
13	15	17	19	21	23	24	25	27	28	29	3	4	5	7	8	9	11	13	14	16	18	20	22	23	25	26	27	28	29	3	4
5	6	7	9	10	12	13	15	17	19	21	23	24	25	27	28	29	30	3	3	4	5	7	8	9	11	13	14	16	18	20	22
23	25	26	27	28	29	30	2	3	4	5	6	7	9	10	12	13	15	17	19	21	23	24	25	27	28	29	30	30	2	3	3
4	5	7	8	9	11	13	14	16	18	20	22	23	25	26	27	28	29	30	31	2	2	3	4	5	6	7	9	10	12	14	15
17	19	21	23	24	25	27	28	29	30	30	31	1	2	3	3	4	5	7	8	9	11	13	14	16	18	20	22	23	25	26	27
28	29	30	31	31	1	2	2	3	4	5	6	7	9	10	12	14	15	17	19	21	23	24	26	27	28	29	30	31	31	32	1
1	2	3	3	4	5	7	8	9	11	13	14	16	18	20	22	23	25	26	27	28	29	30	31	31	32	1	1	2	2	3	4
5	6	7	9	10	12	14	15	17	19	21	23	24	26	27	28	29	30	31	31	32	32	1	1	1	2	3	4	4	6	7	8
9	11	13	15	16	18	20	22	23	25	26	27	28	29	30	31	31	32	32	1	1	1	2	2	3	4	5	6	7	9	10	12
14	15	17	19	21	23	24	26	27	28	29	30	31	31	32	32	32	1	1	1	1	2	3	4	4	6	7	8	9	11	13	15
16	18	20	22	23	25	26	27	28	29	30	31	31	32	32	32	1	1	1	1	2	2	3	4	5	6	7	9	10	12	14	16
17	19	21	23	24	26	27	28	29	30	31	31	32	32	32	32	1	1	1	2	2	3	4	5	6	7	8	10	11	13	15	17
18	20	22	24	25	26	27	29	29	30	31	32	32	32	32	1	1	1	2	2	3	4	5	6	7	9	10	12	14	16	18	19
21	23	24	26	27	28	29	30	31	31	32	32	32	1	1	2	2	3	4	5	6	7	8	10	11	13	15	17	18	20	22	24
25	26	27	29	29	30	31	32	32	32	1	1	2	2	3	4	5	6	7	9	10	12	14	16	18	19	21	23	24	26	27	28
29	30	31	31	32	32	1	2	2	3	4	5	6	7	8	10	11	13	15	17	19	20	22	24	25	26	28	29	30	30	31	32
32	1	2	2	3	4	5	6	7	9	10	12	14	16	18	19	21	23	24	26	27	28	29	30	31	31	32	2	2	3	4	5
6	7	8	10	11	13	15	17	19	20	22	24	25	26	28	29	30	30	31	32	2	3	3	4	5	6	8	9	10	12	14	16
18	19	21	23	24	26	27	28	29	30	31	31	2	3	4	5	6	7	8	10	11	13	15	17	19	20	22	24	25	26	28	29
30	30	31	3	3	4	5	6	8	9	10	12	14	16	18	20	21	23	24	26	27	28	29	30	31	3	4	5	6	7	8	10
11	13	15	17	19	20	22	24	25	26	28	29	30	30	3	4	5	6	8	9	10	12	14	16	18	20	21	23	24	26	27	28
29	30	4	5	6	7	8	10	11	13	15	17	19	20	22	24	25	26	28	29	30	4	5	6	8	9	10	12	14	16	18	20

续表

21	23	24	26	27	28	29	5	6	7	8	10	11	13	15	17	19	20	22	24	25	26	28	29	5	6	8	9	11	12	14	16
18	20	21	23	24	26	27	28	6	7	8	10	11	13	15	17	19	21	22	24	25	27	28	6	8	9	11	12	14	16	18	20
21	23	25	26	27	7	8	10	11	13	15	17	19	21	22	24	25	27	8	9	11	12	14	16	18	20	21	23	25	26	8	10
11	13	15	17	19	21	22	24	25	9	11	12	14	16	18	20	21	23	25	10	11	13	15	17	19	21	22	24	11	12	14	16
18	20	21	23	11	13	15	17	19	21	22	12	14	16	18	20	22	13	15	17	19	21	14	16	18	20	15	17	19	16	18	17

表 4.4　　32× 32 菱形置乱表, 列表

17	18	17	20	18	16	23	19	18	14	27	21	19	17	11	32	24	20	19	15	7	6	28	22	20	18	12	2	13	1	25	21
20	16	8	28	21	7	29	23	21	19	13	3	21	30	14	2	26	22	21	17	9	29	13	8	22	8	30	24	22	20	14	4
22	4	19	31	15	3	27	23	22	18	10	30	14	26	31	9	23	9	31	25	23	21	15	5	23	5	15	12	20	32	16	4
28	24	23	19	11	31	15	27	3	26	32	10	24	10	32	26	24	22	16	6	24	6	16	22	9	13	21	1	17	5	29	25
24	20	12	32	16	28	4	8	25	27	1	11	25	11	1	27	25	23	17	7	25	7	17	23	25	10	10	14	22	2	18	6
30	26	25	21	13	1	17	29	5	9	9	28	26	28	2	12	26	12	2	28	26	24	18	8	26	8	18	24	26	24	15	11
11	15	23	3	19	7	31	27	26	22	14	2	18	30	6	10	10	6	3	29	27	29	3	13	27	13	3	29	27	25	19	9
27	9	19	25	27	25	19	24	16	12	12	16	24	4	20	8	32	28	27	23	15	3	19	31	7	11	11	7	31	14	4	30
28	30	4	14	28	14	4	30	28	26	20	10	28	10	20	26	28	26	20	10	5	25	17	13	13	17	25	5	21	9	1	29
28	24	16	4	20	32	8	12	12	8	32	20	29	15	5	31	29	31	5	15	29	15	5	31	29	27	21	11	29	11	21	27
29	27	21	11	29	22	6	26	18	14	14	18	26	6	22	10	2	30	29	25	17	5	21	1	9	13	13	9	1	21	5	16
30	16	6	32	30	32	6	16	30	16	6	32	30	28	22	12	30	12	22	28	30	28	22	12	30	12	11	23	7	27	19	15
15	19	27	7	23	11	3	31	30	26	18	6	22	2	10	14	14	10	2	22	6	18	7	17	31	17	7	1	31	1	7	17
31	17	7	1	31	29	23	13	31	13	23	29	31	29	23	13	31	13	23	4	12	24	8	28	20	16	16	20	28	8	24	12
4	32	31	27	19	7	23	3	11	15	15	11	3	23	7	19	27	2	8	18	32	18	8	2	32	2	8	18	32	18	8	2
32	30	24	14	32	14	24	30	32	30	24	14	32	14	24	30	1	5	13	25	9	29	21	17	17	21	29	9	25	13	5	1
32	28	20	8	24	4	12	16	16	12	4	24	8	20	28	32	3	9	19	1	19	9	3	1	3	9	19	1	19	9	3	1
31	25	15	1	15	25	31	1	31	25	15	1	15	25	31	6	14	26	10	30	22	18	18	22	30	10	26	14	6	2	1	29
21	9	25	5	13	17	17	13	5	25	9	21	29	10	20	2	20	10	4	2	4	10	20	2	20	10	4	2	32	26	16	2
16	26	32	2	32	26	16	2	16	26	15	27	11	31	23	19	19	23	31	11	27	15	7	3	2	30	22	10	26	6	14	18
18	14	6	26	10	22	21	3	21	11	5	3	5	11	21	3	21	11	5	3	1	27	17	3	17	27	1	3	1	27	17	3
17	28	12	32	24	20	20	24	32	12	28	16	8	4	3	31	23	11	27	7	15	19	19	15	7	27	11	4	22	12	6	4
6	12	22	4	22	12	6	4	2	28	18	4	18	28	2	4	2	28	18	4	13	1	25	21	21	25	1	13	29	17	9	5

续表

4	32	24	12	28	8	16	20	20	16	8	28	23	13	7	5	7	13	23	5	23	13	7	5	3	29	19	5	19	29	3	5
3	29	19	2	26	22	22	26	2	14	30	18	10	6	5	1	25	13	29	9	17	21	21	17	9	14	8	6	8	14	24	6
24	14	8	6	4	30	20	6	20	30	4	6	4	30	27	23	23	27	3	15	31	19	11	7	6	2	26	14	30	10	18	22
22	18	9	7	9	15	25	7	25	15	9	7	5	31	21	7	21	31	5	7	5	24	24	28	4	16	32	20	12	8	7	3
27	15	31	11	19	23	23	8	10	16	26	8	26	16	10	8	6	32	22	8	22	32	6	8	25	29	5	17	1	21	13	9
8	4	28	16	32	12	20	24	11	17	27	9	27	17	11	9	7	1	23	9	23	1	7	30	6	18	2	22	14	10	9	5
29	17	1	13	21	18	28	10	28	18	12	10	8	2	24	10	24	2	7	19	3	23	15	11	10	6	30	18	2	14	29	11
29	19	13	11	9	3	25	11	25	20	4	24	16	12	11	7	31	19	3	12	30	20	14	12	10	4	26	12	5	25	17	13
12	8	32	20	31	21	15	13	11	5	27	26	18	14	13	9	1	22	16	14	12	6	19	15	14	10	17	15	13	16	15	16

此外，在图像置乱机制中引入一个简单的密钥控制。将由密钥生成的第一个[128, 255]的随机整数与置乱的结果进行模2加。编写程序 diamondreplace. m **完成置乱实验。其中需要调用查表程序** replace32 fun. m, **函数代码如下**:

(1) **主函数**: diamondreplace. m

```
% 文件名: diamondreplace. m
% 程序员: 王霞仙
% 编写时间: 2004. 3. 1
% 函数功能: 本函数将完成对输入的图像信号按菱形置换策略进行置乱
% 输入格式举例: result = diamondreplace( secretimage, 1983)
% 参数说明:
% matrix 为输入图像矩阵
% key 为控制密钥
% result 为置乱后的结果
function result = diamondreplace( matrix, key)
% 分析原图像尺寸并补遗
[ m, n]  = size( matrix) ;
rowadd = 32-mod( m, 32) ;
coladd = 32-mod( n, 32) ;
if rowadd== 32
        rowadd = 0;
end
if coladd== 32
        coladd = 0;
```

```
end
input = uint8( zeros( [ m + rowadd  n + coladd] ) ) ;
input( 1: m, 1: n)  = matrix;
% 密钥生成随机数
rand(  'seed ', key) ;
control = randint( 1, 1, [ 128  255] ) ;
% 查表置乱
fun = @replace32fun; % 调用子函数
result = blkproc( input, [ 32  32] , fun) ;
result = bitxor( result, control( 1 , 1) ) ;
```

(2) **查表函数**: replace32fun. m

```
function  result = replace32fun( matrix)
% 行转换表
row = [ 16    15    17    14    16    18    …] % 此处略去, 具体内容请见表 4. 3
col = [ 17    18    17    20    18    16    …]  % 此处略去, 具体内容请见表 4. 4
for i = 1: 32
        for j = 1: 32
            result( i, j)  = matrix( row( i, j) , col( i, j) ) ;
        end
end
```

图 4. 13 是对 woman 进行置乱的效果。

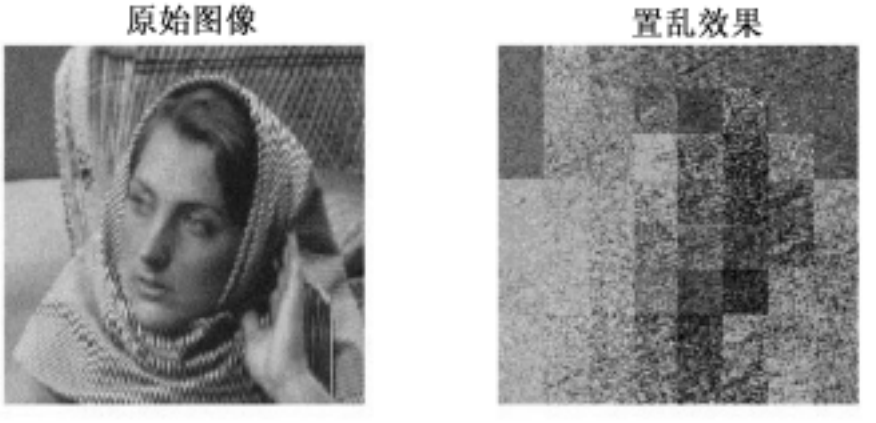

图 4. 13　woman 菱形置乱后的效果

上面置乱模板的宽度是固定的。大家也可以做成是随原图像模板中像素的多少而变化的随机置乱模板。当然, 也可以选取不同形状的置乱模板, 如与原图像模板宽度相同或不同的矩形、星形等。总之, 将图像的原始信息破坏得越大越好, 不过, 这种破坏一定是可以复原的。

4.3.2　图像的幻方变换

相传在大禹治水的时候，黄河支流洛水中突然出现了一只大乌龟，龟甲背有 9 种花点的图案。人们将图案中的花点数了一下，竟惊奇地发现 9 种花点数正巧是 1 到 9 这 9 个数，而且 9 个数所排成的方阵具有绝妙的性质，横的 3 行、纵的 3 列以及两对角线上各自的数字之和都为 15。后来人们就称这个图案为“ 河图洛书 ”。洛书因其性质之独特而使人们大感兴趣，对其进行了多方面的研究。我国汉朝的一本叫《数术记遗》的书，把幻方问题叫“ 九宫算 ”，幻方叫九宫图。宋朝数学家杨辉把类似于“ 九宫图 ”的图形叫“ 纵横图 ”。

我国的纵横图传到欧洲，它的多彩的变幻特征吸引了西方的数学家们，他们对此也很感兴趣，并称纵横图为“ 幻方 ”，意为“ 魔幻的方阵 ”。著名的数学家欧拉和汉弥尔顿，大发明家富兰克林都对幻方有过深入的探讨。1759 年，数学家欧拉发表了独具一格的“ 马步幻方 ”。1901 年法国数学家里利发明了“ 平方幻方 ”。1958 年美国数学家霍纳造出了“ 双重幻方 ”等。西方研究幻方的热潮，一浪高过一浪。现在幻方这个起源于我国的神秘的数学问题，已经在世界上形成了一个丰富的体系，成为数学研究的重要课题。

下面，我们就从数学的角度定义一下平面的二维幻方。

定义 4.1　(幻方)　n 阶方阵 $M=\{m_{ij}\}$, $i,j=1,2,3,\cdots,n$ 为二维幻方，当且仅当

① " $K\in\{1,2,\cdots,n\}$，有 $\sum_{i=1}^{n} m_{iK}=\sum_{j=1}^{n} m_{Kj}=c$　(c 为仅与 n 有关的常数)

② $\sum_{i=j} m_{ij}=\sum_{i+j-1=n} m_{ij}=c$

特别地，当 $m_{ij}\in\{1,2,3,\cdots,n^2\}$ 且两两不同时，称 M 为 n 阶标准幻方。此时 $c=\frac{n(n^2+1)}{2}$。

直观地解释上面的定义就是，将 n^2 个不同的数填入 n 阶方阵的 n^2 个方格中，若 n 行 n 列及两条主对角线上诸数之和都等于一常数 c，则称此 n 阶幻方，c 称为该 n 阶幻方的幻和。求解一个平面二维幻方的游戏可以利用一定的规则去得出。如图 4.14 所示。用以下公式可以求取一定阶数的标准幻方，对于 n 阶标准幻方 $M^{(n)}=\{m_{ij}\}$, $i,j=1,2,3,\cdots,n$，将其写成如下形式：

$$k_1\equiv ai+bj+p(\mathrm{mod}n)$$

$$k_2\equiv ci+dj+q(\mathrm{mod}n)$$

$$m_{ij}=nk_1+k_2+1 \qquad (4.1)$$

其中，n 不是 2，3 的倍数，a，b，c，d，p，q 都是整数，则可以得到相应的必要条件：

① a，b，c，d 与 n 互质。

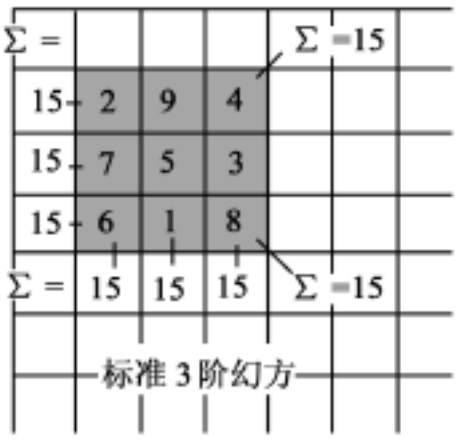

图 4.14　3 阶标准幻方

② (ad - bc) 、(a ±b) 、(c ±d) 都不等于 0 并与 n 互质。

③ p, q 是修正参数。

利用式(4.1) 列出的必要条件, 我们可以编程“ 凑 ”出一个 n 阶标准幻方的解。当然, 当 n 是 2, 3 的倍数时, a, b, c, d 需详细讨论且对 k_1k_2 要作修正。表 4.5 列出了几组常用的标准幻方的求解参数。

表 4.5　　常用的标准幻方的求解参数

n	a	b	c	d	p	q
5	1	2	1	3	0	0
7	1	2	1	3	0	0
11	2	1	1	3	0	0
13	5	2	1	7	1	0
一些与 6 互质的 n	1	2	1	3	0	0

图 4.15 是取阶为 5, 7, 11 时的标准幻方示意图。当然, 这只是一种解的情况, 解是不惟一的。

最后, 对于奇数阶的二维平面幻方, 我们在这里不加证明地给出一个计算机求解的算法, 便于大家编程。

对于 n 阶(n 为奇数) 标准幻方, 其中 1 ～n^2 个数可用如下规律存放:

① 将 1 放在第 1 行的中间一列。

② 从 2 到 n^2 依次按以下规则存放: 每一个数存放在前一数的上一行、后一列的位置上。但有以下四种情况为特殊:

- 如果前一个数存放在第 1 行, 最后一个数存放在第 n 行。
- 如果前一个数存放在第 n 行, 最后一个数存放在第 1 行。
- 如果前一个数存放在第 1 行第 n 列的位置上, 则后一个数存放在前一个数的

下面(即第 2 行第 n 列)。

● 如果按照上述规则确定的位置上已经有数存放了,则直接将这一个数存放在前一个数的下面(即行数加 1,列数不变)。

用这个算法可以很方便地求得大阶数的幻方。当然,所求出的幻方与前面我们用数论知识推导出的幻方在形态上存在着差异,但这并不影响其幻方本身的性质。

编写函数 magicsquares. m 完成实验,函数代码如下:

```
% 文件名: magicsquares. m
% 程序员: 王霞仙
% 编写时间: 2004. 7. 15
% 函数功能: 本函数将完成 n 阶二维幻方的求取 . 要求 n 为奇数
% 输入格式举例: result = magicsquares( 5)
% 参数说明:
% n 为阶数
% result 为求得的二维幻方
function result = magicsquares( n)
if mod( n, 2)  == 0
    error( '要求为奇数' );
end
result = zeros( n) ;
j = floor( n /2)  + 1; % 中间 1 列
i = n + 1; % 便于以后从第 n 行开始考虑起
result( 1, j)  = 1;
for k = 2: n* n    % 依次考虑后 n^2 - 1 个数
    i = i - 1;
    j = j + 1;    % 行数减 1, 列数加 1
    if i < 1  && j > n        % 特殊情况 4
        i = i + 2;
        j = j - 1;
    else
        if i < 1              % 特殊情况 1
            i = n;
        end
        if j > n              % 特殊情况 2
            j = 1;
        end;
    end;
```

```
        if result( i, j)  == 0
            result( i, j)  = k;
        else                          % 特殊情况 3
            i = i + 2;
            j = j - 1;
            result( i, j)  = k;
        end
    end
```

标准 5 阶幻方

20	3	11	24	7
21	9	17	5	13
2	15	23	6	19
8	16	4	12	25
14	22	10	18	1

Σ = 65

标准 7 阶幻方

26	36	4	21	31	48	9
34	44	12	22	39	7	17
42	3	20	30	47	8	25
43	11	28	38	6	16	33
2	19	29	46	14	24	41
10	27	37	5	15	32	49
18	35	45	13	23	40	1

Σ = 175

标准 11 阶幻方

38	52	66	69	83	97	100	114	7	21	24
61	75	78	92	106	120	2	16	30	44	47
84	98	101	115	8	22	25	39	53	56	70
107	121	3	17	31	34	48	62	76	79	93
9	12	26	40	54	57	71	85	99	102	116
32	35	49	63	77	80	94	108	111	4	18
55	58	72	86	89	103	117	10	13	27	41
67	81	95	109	112	5	19	33	36	50	64
90	104	118	11	14	28	42	45	59	73	87
113	6	20	23	37	51	65	68	82	96	110
15	29	43	46	60	74	88	91	105	119	1

Σ = 671

图 4.15　n 阶标准幻方

下面我们结合标准幻方来看看其在图像置乱上的作用。我们知道，要图像置乱是为了增加隐藏的鲁棒性。一个置乱的图像无论是合法用户还是非法用户都看不懂，要使合法用户能完整地读取秘密信息，就要满足两个条件：

①仅有合法用户知道的密钥参与运算。

②置乱算法本身可以保证图像复原，即算法是可逆的。

前面介绍的菱形置乱算法是一个对合算法，当然满足条件②。那么幻方置乱算法呢？

幻方置乱的思想其实也是查表思想。以图 4.14 所示的 3 阶标准幻方为例，数

据 2 就表示图像块在 1 那个地方的像素变换到的位置，数据 3 就表示图像块在 2 那个地方的像素变换到的位置，依此类推，数据 1 表示图像块在 9 那个地方的像素变换到的位置。当然，在具体编写程序时最好将数值转化为行、列标分别查找，以提高速度。具体算法实现我们在后面阐述。

幻方置乱运算具有准对合性。假设记 n 阶图像块（对应于 n 阶标准幻方）I 的幻方置乱为 I_1 = Magic(I)，则相应的对 I_1 再进行幻方置乱得到 I_2 = Magic(I_1) = $Magic^2$(I)，依此类推，一般地有 I_{n2} = $Magic^{n^2}$(I) = I，图像还原。图 4.16 是 3 阶标准幻方 9 次变换复原流程示意图。

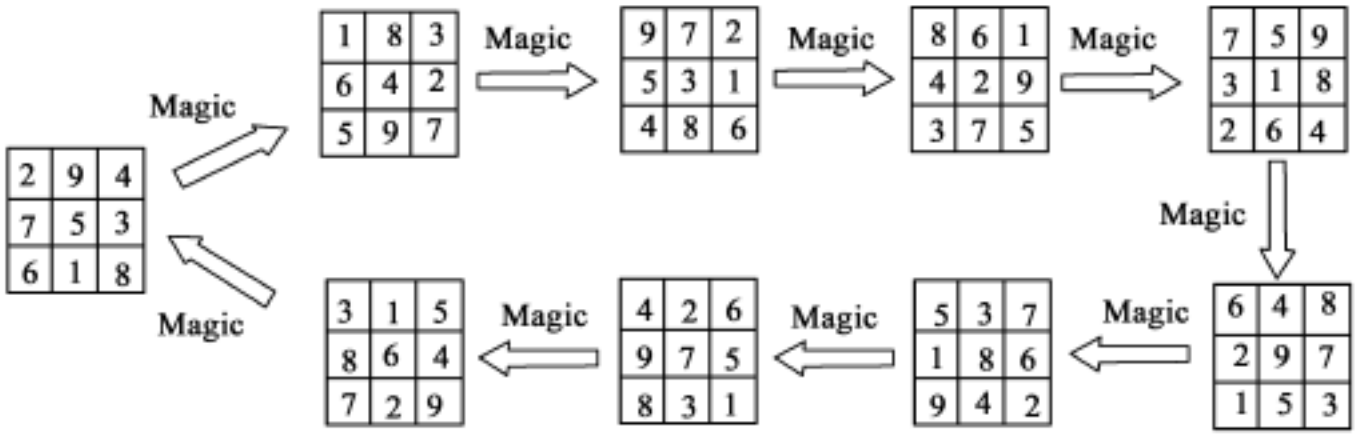

图 4.16　3 阶标准幻方 9 次变换复原示意图

由此，我们也可以得到密钥控制的方法。设算法加密钥为 ekey，则解密钥为 dkey 分别表示输入数据执行幻方置乱的次数，仅要求 ekey + dkey = n^2 即可。

实验采用 11 阶标准幻方作置乱模板，表 4.6 是由 11 阶幻方改成的行列查找表。

表 4.6　　11 阶标准幻方对应的行列变换表

数据	行标	列标	数据	行标	列标	数据	行标	列标	数据	行标	列标
1	11	11	31	4	5	61	2	1	91	11	8
2	2	7	32	6	1	62	4	8	92	2	4
3	4	3	33	8	8	63	6	4	93	4	11
4	6	10	34	4	6	64	8	11	94	6	7
5	8	6	35	6	2	65	10	7	95	8	3
6	10	2	36	8	9	66	1	3	96	10	10
7	1	9	37	10	5	67	8	1	97	1	6
8	3	5	38	1	1	68	10	8	98	3	2

续表

数据	行标	列标	数据	行标	列标	数据	行标	列标	数据	行标	列标
9	5	1	39	3	8	69	1	4	99	5	9
10	7	8	40	5	4	70	3	11	100	1	7
11	9	4	41	7	11	71	5	7	101	3	3
12	5	2	42	9	7	72	7	3	102	5	10
13	7	9	43	11	3	73	9	10	103	7	6
14	9	5	44	2	10	74	11	6	104	9	2
15	11	1	45	9	8	75	2	2	105	11	9
16	2	8	46	11	4	76	4	9	106	2	5
17	4	4	47	2	11	77	6	5	107	4	1
18	6	11	48	4	7	78	2	3	108	6	8
19	8	7	49	6	3	79	4	10	109	8	4
20	10	3	50	8	10	80	6	6	110	10	11
21	1	10	51	10	6	81	8	2	111	6	9
22	3	6	52	1	2	82	10	9	112	8	5
23	10	4	53	3	9	83	1	5	113	10	1
24	1	11	54	5	5	84	3	1	114	1	8
25	3	7	55	7	1	85	5	8	115	3	4
26	5	3	56	3	10	86	7	4	116	5	11
27	7	10	57	5	6	87	9	11	117	7	7
28	9	6	58	7	2	88	11	7	118	9	3
29	11	2	59	9	9	89	7	5	119	11	10
30	2	9	60	11	5	90	9	1	120	3	6
备注: 行、列标均从 1 到 11									121	4	2

查找表的使用方法为: 假设置乱的结果矩阵是 result, 输入矩阵为 matrix, i, j 为矩阵元素的行、列标, 则:

$$result(i, j) = matrix(row(Magic_k(i, j)), col(Magic_k(i, j)))$$

其中, $Magic_k$ 是对 11 阶标准幻方作 k 次 Magic 操作的结果, 如对 11 阶标准幻方作 10 次 Magic 操作得到 $Magic_{10}$, 如表 4.7 所示。$Magic_{10}(1, 1) = 28$。再查表 4.6 可得 $row(28) = 9$, $col(28) = 6$。则置乱结果的第(1, 1)位置上的像素 result(1, 1)就是原始图像的第(9, 6)位置上的像素。

表 4.7　　11 阶标准幻方 10 次 Magic 操作的结果: $Magic_{10}$

28	42	56	59	73	87	90	104	118	11	14
51	65	68	82	96	110	113	6	20	34	37
74	88	91	105	119	12	15	29	43	46	60
97	111	114	7	21	24	38	52	66	69	83
120	2	16	30	44	47	61	75	89	92	106
22	25	39	53	67	70	84	98	101	115	8
45	48	62	76	79	93	107	121	3	17	31
57	71	85	99	102	116	9	23	26	40	54
80	94	108	1	4	18	32	35	49	63	77
103	117	10	13	27	41	55	58	72	86	100
5	19	33	36	50	64	78	81	95	109	112

至于 Magic 操作, 用伪 C 代码描述为:

```
Magic = {11 阶标准幻方};
//密钥控制执行次数
for ( i = 1; i <= ekey or dkey ; i ++ )
    {
  for ( r = 1; r <= 11; r ++ )
      for ( c = 1; c <= 11; <c ++ )
        {
         Magic( r, c) = Magic( r, c) -1;
        if ( Magic( r, c) == 0)
            Magic( r, c) = 11^2;
        }
}
```

编写函数 magicreplace. m 完成置乱实验, 其中需要调用查表函数 replacemagicfun. m。前主函数存放 11 阶标准幻方, 子函数存放表 4.6, 函数代码如下:

(1) 幻方置乱主函数: magicreplace. m

```
% 文件名: magicreplace. m
% 程序员: 王霞仙
% 编写时间: 2004. 3. 1
% 函数功能: 本函数将完成对输入的图像信号按幻方置换策略进行置乱
% 输入格式举例: result = magicreplace( secretimage, 1, 1983)
% 参数说明:
% matrix 为输入图像矩阵
% key 为控制密钥
```

```
% eord 为 1 表示置乱变换, 为 0 表示复原变换
% result 为置乱后的结果
function result = magicreplace( matrix, eord, key)
% 分析原图像尺寸并补遗
[ m, n]  = size( matrix) ;
rowadd = 11-mod( m, 11) ;
coladd = 11-mod( n, 11) ;
if rowadd == 11
        rowadd = 0;
end
if coladd == 11
        coladd = 0;
end
input = uint8( zeros( [ m + rowadd  n + coladd] ) ) ;
input( 1: m, 1: n)  = matrix;
% 密钥生成随机数
rand( 'seed', key) ;
control = randint( 1, 1, [ 1  121] ) ;
%11 阶标准幻方
magic =
[ 38     52     66     69     83     97    100    114      7     21     24
  61     75     78     92    106    120      2     16     30     44     47
  84     98    101    115      8     22     25     39     53     56     70
 107    121      3     17     31     34     48     62     76     79     93
   9     12     26     40     54     57     71     85     99    102    116
  32     35     49     63     77     80     94    108    111      4     18
  55     58     72     86     89    103    117     10     13     27     41
  67     81     95    109    112      5     19     33     36     50     64
  90    104    118     11     14     28     42     45     59     73     87
 113      6     20     23     37     51     65     68     82     96    110
  15     29     43     46     60     74     88     91    105    119      1  ];
if eord  == 0
        control = 121-control;
elseif eord  == 1
        control = control;
else
```

```
        error( 输入参数错误 ) ;
end
% 幻方变换主过程
for define = 1: key% control
        for r = 1: 11
                for c = 1: 11
                        magic( r, c)  = magic( r, c) -1;
                        if magic( r, c)  == 0
                                magic( r, c)  = 121;
                        end
                end
        end
end
% 查表置乱
fun = @ replacemagicfun; % 调用子函数
result = blkproc( input, [ 11 11] , fun, magic) ;
```

(2) **行列转换表子函数**: replacemagicfun. m

```
        %11 阶幻方的行列查找表程序
        function result = replacemagicfun( matrix, P1)
        % 初始化 11 阶幻方的行列查找表
        row = [ 11, 2, 4, 6, 8, 10, 1, 3, 5, 7, 9, 5, 7, 9, 11, 2, 4, 6, 8, 10, 1, 3, 10, 1, 3, 5, 7, 9,
11, 2, 4, 6, 8, 4, 6, 8, 10, 1, 3, 5, 7, 9, 11, 2, 9, 11, 2, 4, 6, 8, 10, 1, 3, 5, 7, 3, 5, 7, 9, 11, 2,
4, 6, 8, 10, 1, 8, 10, 1, 3, 5, 7, 9, 11, 2, 4, 6, 2, 4, 6, 8, 10, 1, 3, 5, 7, 9, 11, 7, 9, 11, 2, 4, 6,
8, 10, 1, 3, 5, 1, 3, 5, 7, 9, 11, 2, 4, 6, 8, 10, 6, 8, 10, 1, 3, 5, 7, 9, 11, 3, 4] ;
        col = [ 11, 7, 3, 10, 6, 2, 9, 5, 1, 8, 4, 2, 9, 5, 1, 8, 4, 11, 7, 3, 10, 6, 4, 11, 7, 3, 10, 6,
2, 9, 5, 1, 8, 6, 2, 9, 5, 1, 8, 4, 11, 7, 3, 10, 8, 4, 11, 7, 3, 10, 6, 2, 9, 5, 1, 10, 6, 2, 9, 5, 1,
8, 4, 11, 7, 3, 1, 8, 4, 11, 7, 3, 10, 6, 2, 9, 5, 3, 10, 6, 2, 9, 5, 1, 8, 4, 11, 7, 5, 1, 8, 4, 11, 7,
3, 10, 6, 2, 9, 7, 3, 10, 6, 2, 9, 5, 1, 8, 4, 11, 9, 5, 1, 8, 4, 11, 7, 3, 10, 6, 2] ;
        for i = 1: 11
                for j = 1: 11
                        result( i, j)  = matrix( row( P1( i, j) ) , col( P1( i, j) ) ) ;
                end
        end
```

图 4.17 是对 woman 做不同次数幻方置乱的效果图。很遗憾的是，由于我们没有采用阶数更高的幻方作置乱模板，置乱出来的结果仍可以看见原始图像的轮廓。但这种方法对于文本图像的置乱效果还是比较好的。所谓“文本图像”是指以图像

的形式保存的文字内容。图 4.18 是汪国真的一首诗被置乱的结果，在置乱的过程中文字内容是完全不可读的。当然，大家有兴趣可以按前述的方法自己编程推算出高阶标准幻方，相信效果会更好。另外，将低阶幻方作置乱模板置乱的结果再以图像块为单位进一步置乱，也能取得良好的置乱效果。

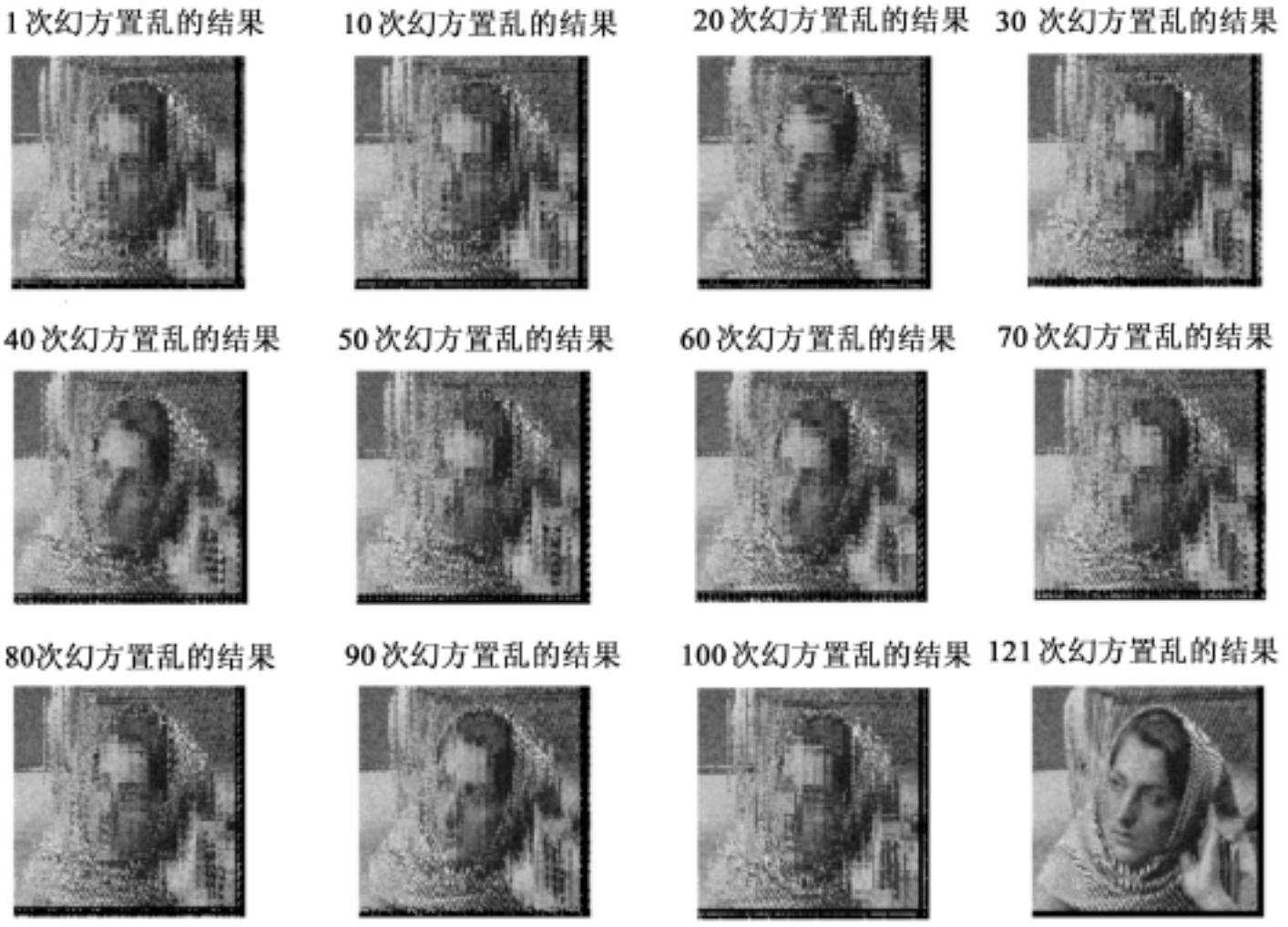

图 4.17　woman 幻方置乱后的效果

图 4.18　文本图像置乱后的效果

4.3.3　图像的 Hash 置乱

前面的两种置乱都是对图像分块进行的，而且其共同的问题是密钥控制并不得力。下面介绍的一种图像置乱方法实际上就是我们在 2.6.3 节中介绍的 Hash 置换的特例——对于 m× n 个像素点，我们要求随机置换 m× n 个，就完成了图像的 Hash 置乱。鉴于该算法具有无冲突(collision) 和强密钥控制的特点，显然是一个很好的图像置乱算法。需要说明的是，这种算法不是对合的，所以在实现上较前两种复杂一些。另外，其算法执行起来也比较费时间。编写函数 hashdisturb. m 完成实验，具体 hashreplacement. m 的部分请参见 2.6.3 节。

```
% 文件名: hashdisturb. m
% 程序员: 王霞仙
% 编写时间: 2004. 3. 2
% 函数功能: 本函数将完成对输入的图像信号按 Hash 置换策略进行置乱
% 输入格式举例: result = hashdisturb( secretimage, 1, 1983, 421, 1121)
% 参数说明:
% matrix 为输入图像矩阵
% key1 -key3 为控制密钥
% eord 为 1 表示置乱变换, 为 0 表示复原变换
% result 为置乱后的结果
function result = hashdisturb( matrix, eord, key1, key2, key3)
% 分析原图像尺寸并补遗
[ m, n]  = size( matrix) ;
% 调用随机置换函数
[ row, col]  = hashreplacement( matrix, m*  n, key1, key2, key3) ;
% 置乱函数
count = 1;
if eord == 1
    for i = 1∶ m
        for j = 1∶ n
            result( i, j)  = matrix( row( count) , col( count) ) ;
            count = count + 1;
        end
    end
end
% 复原函数
if eord == 0
```

```
        for i = 1: m
            for j = 1: n
                result( row( count) , col( count) )  = matrix( i, j) ;
                count = count + 1;
            end
        end
    end
```

图 4.19 是取密钥 1983, 421, 1121 下的置乱和复原的效果图。显然, 由于是全幅度的置乱, 原始图像信息荡然无存了。

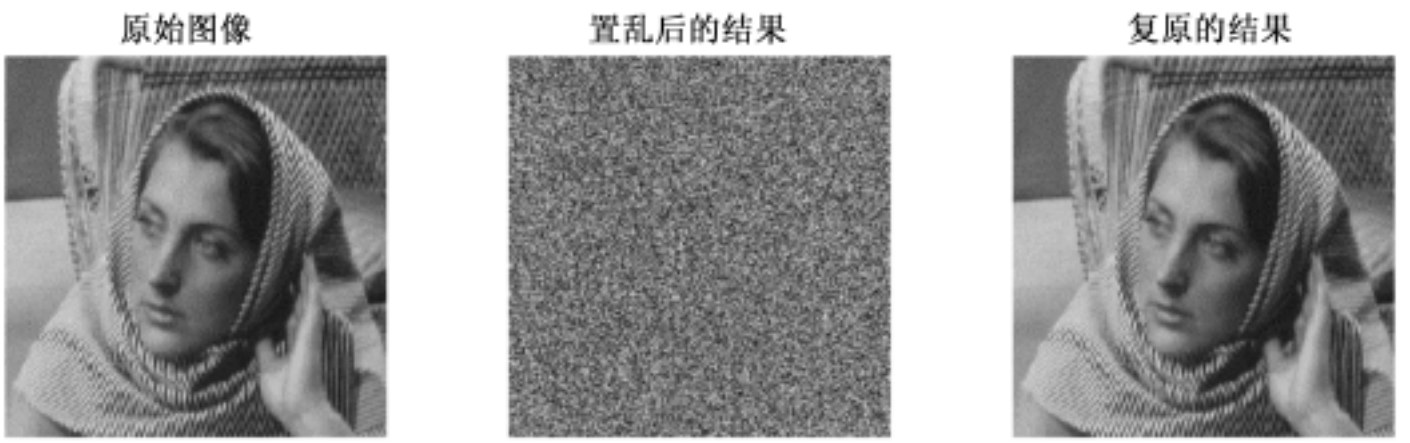

图 4.19　woman 的 Hash 置乱后的效果

4.3.4　隐藏置乱图像的优点

经过多次置乱后, 图像就会彻底地改变, 从置乱后的图像上根本看不到原图像的任何特征。使用置乱方法为什么可以增加图像伪装的鲁棒性呢? 下面我们就来讨论这个问题。

置乱图像隐藏的抗恶意攻击性能如图 4.20 所示。

首先, 将图像置乱后, 将得到一幅杂乱无章的图像, 这个图像无色彩、无纹理、无形状, 从中无法读取任何信息, 那么, 将这样一幅图嵌入到另一幅普通图像时就不易引起那幅图色彩、纹理、形状的太大改变, 甚至不会发生改变, 这样人眼就不易识别, 从而逃出了第三方的视线。

其次, 由于秘密图像是置乱后的图像, 根据上述图像的“三无”特征, 第三方根本不可能对其进行色彩、纹理、形状等的统计分析, 即便他们截取到了秘密图像, 也是无能为力的。如果第三者企图对秘密图像进行反置乱, 这也是不可能的, 由于图像置乱有很多种方法, 每种方法又可以使用不同的置乱模板算法, 设置不同的参数, 使用者有很大的自由度, 他可以根据自己的想法得到不同的结果, 相反, 这给企图截获秘密信息的第三方带来了很大的困难, 使他们需要耗费巨大的计算量来穷举测试各种可

能性。

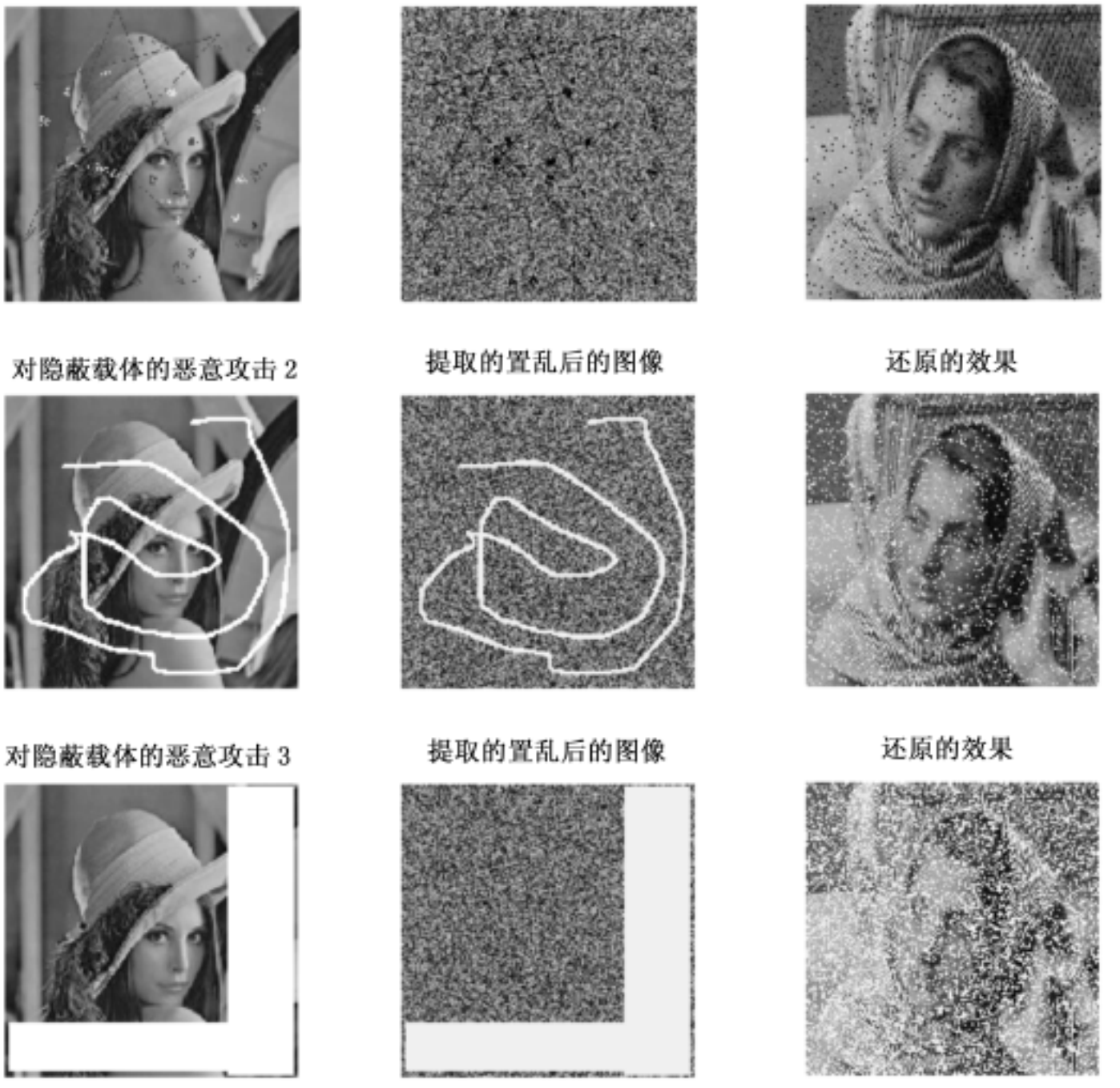

图 4.20　置乱图像隐藏的抗恶意攻击性能

最后，我们再设想一下，如果第三方反置乱不成，在隐蔽载体上恶意修改怎么办？通过实验我们知道，用置乱的方法是可以抵抗这些攻击的，因为对秘密图像进行反置换的过程，就使得第三方在图像上所涂、画的信息分散到画面的各个地方，形成了点状的随机噪声，对视觉影响的程度不大。图 4.20 是我们随意对隐蔽载体进行 3 种恶意攻击后提取的秘密图像内容。可以看到，即使是在攻击 3 下，秘密图像的轮廓依然可见，这是在未置乱图像的隐藏下不可想像的。当然，为了使提取的信息更为清晰，最好对破坏严重的图像进行边界保持的中值滤波等方面的处理，以去除随机噪声。