南开大学

Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

信息隐藏技术实验四

媒体文件格式剖析

穆禹宸　2012026

年级：2020 级

专业：信息安全、法学双学位班

指导教师：李朝晖

2023 年 4 月 9 日

目录

一、 BMP 文件格式探究

本次实验，我探究的是 BMP 文件结构。

(一) BMP 文件简介

BMP 文件是一种位图文件格式，全称为 Bitmap，它是 Windows 操作系统中的标准图像文件格式之一。BMP 文件可以被多种 Windows 应用程序所支持，因此在 Windows 系统中使用非常广泛。BMP 文件的特点是包含的图像信息较丰富，几乎不进行压缩，因此图像质量非常高，可以保留图像的细节和色彩。但是，由于不进行压缩，BMP 文件的文件体积较大，占用磁盘空间过大，不适合在网络上传输。此外，BMP 文件也不支持透明度等高级特性，因此在某些情况下可能不太适合使用。总的来说，BMP 文件是一种非常实用的图像文件格式，可以保证图像质量，但需要注意文件体积的问题。

(二) BMP 文件结构

首先整体介绍一下 BMP 文件的大致结构。

BMP 文件结构

BMP 文件总体上由 4 部分组成，分别是位图文件头、位图信息头、调色板和图像数据。

1. 位图文件头（bitmap-file header）：包含 BMP 图像文件的类型、显示内容等信息
2. 位图信息头（bitmap-information header）：包含有 BMP 图像的宽、高、压缩方法，以及定义颜色等信息；
3. 彩色表/调色板（color table）：这个部分是可选的，有些位图需要调色板，有些位图，比如真彩色图（24 位的 BMP）就不需要调色板；
4. 位图数据（bitmap-data）：这部分的内容根据 BMP 位图使用的位数不同而不同，在 24 位图中直接使用 RGB，而其他的小于 24 位的使用调色板中颜色索引值。

下面我们来具体介绍一下每个结构的细节。

1. 文件头信息块

文件信息头的大小为 14 字节，存储文件类型，文件大小等信息.

```
struct tagBmpFileHeader //文件头
{
^^Iunsigned short bfType; //标识该文件为 bmp 文件，判断文件是否为 bmp 文件，即用该值与"0x4d42"比较是否相等即可，0x4d42 = 19778
^^Iunsigned long  bfSize;//位图文件大小，包括这 14 个字节。
^^Iunsigned short bfReserved1;//预保留位，暂不用。
^^Iunsigned short bfReserved2;//预保留位，暂不用。
^^Iunsigned long  bfOffBits;//图像数据区的起始位置
}BmpFileHeader;//14 字节:short2 个,long4 个
```

位图文件头（bitmap-file header）包含了图像类型、图像大小、图像数据存放地址和两个保留未使用的字段。

打开 WINGDI.h 文件，搜索”BITMAPFILEHEADER” 就可以定位到 BMP 文件的位图文件头的数据结构定义。

2. 图像描述信息块

图片信息头大小为 40 字节，存储着图像的尺寸，颜色索引，位平面数等信息

```
struct tagBmpInfoHeader //信息头
{
    unsigned long  biSize;//本结构的长度，为 40 个字节。
    long      biWidth;//宽度
    long      biHeight;//高度
    unsigned short biPlanes;//目标设备的级别，必须是 1。
    unsigned short biBitCount; //每个像素所占的位数（bit），其值必须为 1（黑白图像）、
    ↪ 4（16 色图）、8（256 色）、24（真彩色图），新的 BMP 格式支持 32 位色。
    unsigned longbiCompression;//压缩方式，有效的值为 BI_RGB（未经压缩）、BI_RLE8、
    ↪ BI_RLE4、BI_BITFILEDS（均为 Windows 定义常量）。
    unsigned longbiSizeImage;  //图像区数据大小，即实际的位图数据占用的字节数
    long      biXPelsPerMeter;  //水平分辨率，像素每米
    long      biYPelsPerMeter;  //垂直分辨率，单位是像素/米
    unsigned long biClrUsed;   //位图实际用到的颜色数，如果该值为零，则用到的颜色数
    ↪ 为 2 的 biBitCount 次幂。
    unsigned short biClrImportant;//位图显示过程，重要的颜色数；0--所有都重要
}BmpInfoHeader;//40 字节
```

位图信息头(bitmap-information header)包含了位图信息头的大小、图像的宽高、图像的色深、压缩说明图像数据的大小和其他一些参数。打开 WINGDI.h 文件,搜索”tagBITMAPINFOHEADER” 就可以定位到 BMP 文件的位图信息头的数据结构定义。

3. 颜色表

存储调色板的相关信息（由颜色索引数决定）(可以没有此信息，下面的例子就因为采用了 24 位真彩色保存所以没有这部分信息）

```
struct tagRGBPallete //调色板，实际上是一个 RGBPallete 结构的数组，数组的长度由
↪ biClrUsed 指定
{
^^Iunsigned char b;
^^Iunsigned char g;
^^Iunsigned char r;
^^Iunsigned char alpha; //预保留位，暂不用。
}RGBPallete;
```

彩色表/调色板（color table）是单色、16 色和 256 色图像文件所特有的，相对应的调色板大小是 2、16 和 256，调色板以 4 字节为单位，每 4 个字节存放一个颜色值，图像的数据是指

向调色板的索引。

4. 图像数据区

存储位图数据（由图像尺寸决定），每一个像素的信息在这里存储。在此部分记录着每点像素对应的颜色号，其记录方式也随颜色模式而定，即 2 色图像每像素占 1 位（8 位为 1 字节）；16 色图像每像素占 4 位（半字节）；256 色图像每像素占 8 位（1 字节）；真彩色图像每像素占 24 位（3 字节），所以整个数据区的大小也会随之变化，可得出如下计算公式：图像数据信息大小 =（图像宽度 * 图像高度 * 记录每个像素需要的位数）/8。

如果图像是单色、16 色和 256 色，则紧跟着调色板的是位图数据，位图数据是指向调色板的索引序号。

如果位图是 16 位、24 位和 32 位色，则图像文件中不保留调色板，即不存在调色板，图像的颜色直接在位图数据中给出。

16 位图像使用 2 字节保存颜色值，常见有两种格式：5 位红 5 位绿 5 位蓝和 5 位红 6 位绿 5 位蓝，即 555 格式和 565 格式。555 格式只使用了 15 位，最后一位保留，设为 0。

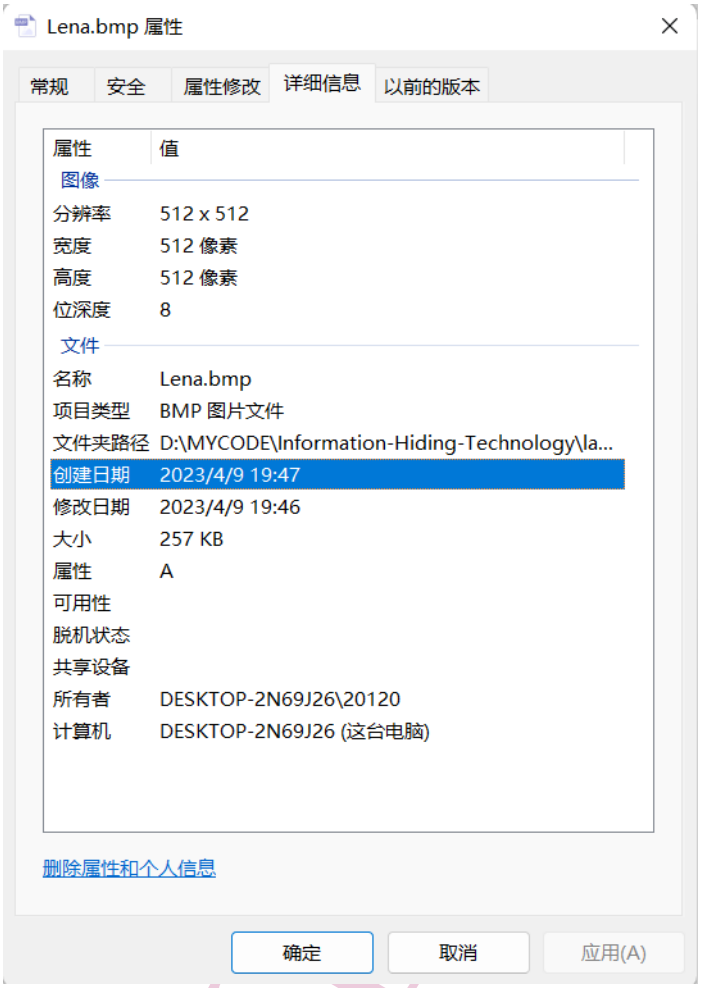
24 位图像使用 3 字节保存颜色值，每一个字节代表一种颜色，按红、绿、蓝排列。

32 位图像使用 4 字节保存颜色值，每一个字节代表一种颜色，除了原来的红、绿、蓝，还有 Alpha 通道，即透明色。

如果图像带有调色板，则位图数据可以根据需要选择压缩与不压缩，如果选择压缩，则根据 BMP 图像是 16 色或 256 色，采用 RLE4 或 RLE8 压缩算法压缩。

(三) 利用 UltraEdit 分析 bmp 文件

右键单击一张 bmp 格式的图片，选择“属性-> 详细信息”



可以看到,这张图片的位深度是 8,也就是说,它是一张位单色的色位图。它的尺寸为 512*512,由（512*512*8）/（8*1024）=257KB，与详细信息中标注的大小相同。接下来用 UltraEdit 打开这张 BMP 图像，显示的是十六进制的代码

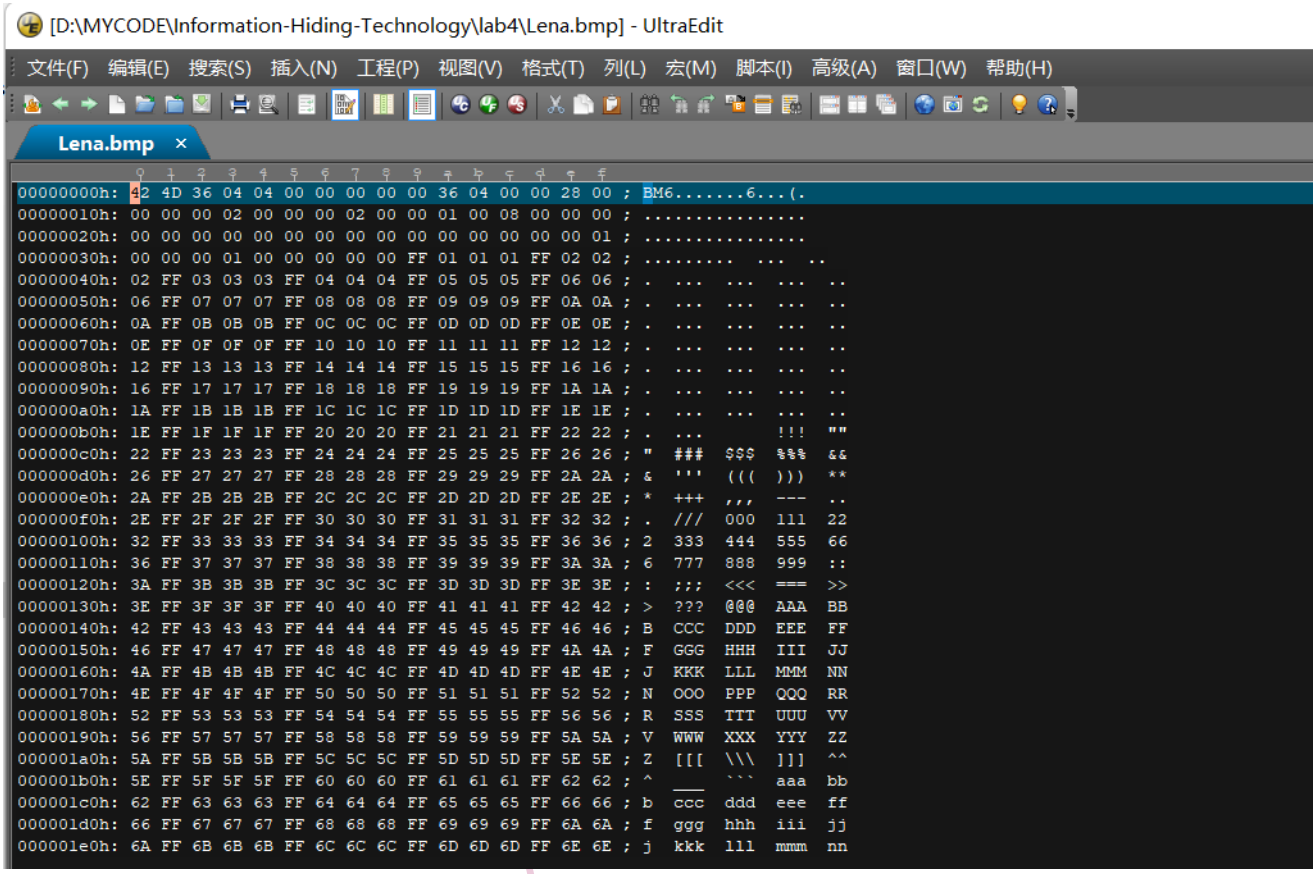


图 1: 文件大小

Windows 系统中的数据采用小端序进行存储，阅读时需要注意顺序。

参照上面的文件信息头结构体的内容对这幅位图的内容进行分析。文件信息头结构体第一个数据是 unsigned short（16 位）类型的 bfType 变量。观察十六进制代码结果可以看到第一行开头的 42 4D 倒着念就是 4D 42(刚好 16 位对应 unsigned short 类型)，即 bftype=0x4D42（转换为十进制为 19778，实际上所有 BMP 图像的 bfType 对应属性都是这个值)。按照这个方法可得出第二个数据 bfSize 类型为 unsigned int（32 位），图中对应的十六进制代码为 00040436 （转换为十进制为 263222)，这代表文件大小为 263222 字节 =257KB，和在属性栏里的文件大小完全相等。接下来利用类似的方法可以从十六进制代码中得到这张位图的文件头信息块和图像描述信息块所存储的信息.

```
unsigned short bfType = 0x4D42 = 19778
unsigned int bfSize = 0x00222C36 = 2239542 字 节 =269986/(1024*1024)
=2.13MB
unsigned short bfReserved1 = 00 00
unsigned short bfReserved2 = 00 00
unsigned int bfOffBits = 0X00000036 = 0x36 = 54字 节

unsigned int biSize = 0x00000028 = 0x28 = 40字 节 ( 图 像 信 息 头 结 构 体
大 小 就 是40字 节 )
long biWidth = 0x00000100 = 0x100 = 256像 素 ;
long biHeight = 0x00000100 = 0x100 = 256像 素 ;
unsigned short biPlanes = 0x0001 = 0x1 = 1;
```

```
unsigned short biBitCount = 0x0010 = 0x10 = 8位 ;
unsigned int biCompression = 0x00000000 = 0;
unsigned int biSizeImage = 0x00069C00 = 433152;( 等 于 bfSize-bfOffBits )
long biXPelsPerMeter = 0x00000000 = 0;
long biYPelsPerMeter = 0x00000000 = 0;
unsigned int biClrUsed = 0x00000000 = 0;
unsigned int biClrImportant = 0x00000000 = 0;
```

二、 BMP 文件信息隐藏

(一) LSB 方法

LSB（Least Significant Bit）方法是一种常用的信息隐藏技术，它的基本思想是将要隐藏的信息嵌入到载体图像的最低有效位中，这样可以在不影响载体图像质量的前提下，实现信息的隐蔽传输。

以在 Lena.bmp 中隐藏 lion.bmp 为例，介绍这种信息隐藏方式。

实验表明，像素的最低有效位对图片的视觉效果影响很小，因此可以在此处隐藏信息。

代码实现如下:

```
function LSB_ImageHiding()
    x=imread("Lena.bmp"); %载体图像
    m=imread("lion.bmp"); %水印图像
    imshow(x,[]);
    imshow(m,[]);
    WaterMarked=Hide(x,m);
    watermark=Extract(WaterMarked);
end

function WaterMarked = Hide(origin,watermark)
    [Mc,Nc]=size(origin);
    WaterMarked=uint8(zeros(size(origin)));

    for i=1:Mc
        for j=1:Nc
            WaterMarked(i,j)=bitset(origin(i,j),1,watermark(i,j));
        end
    end

    imwrite(WaterMarked,'lsb_watermarked.bmp','bmp');
    figure;
    imshow(WaterMarked,[]);
    title("WaterMarked Image");
end

function WaterMark=Extract(WaterMarked)
    [Mw,Nw]=size(WaterMarked);
    WaterMark=uint8(zeros(size(WaterMarked)));
```

```
29
30      for i=1:Mw
31          for j=1:Nw
32              WaterMark(i,j)=bitget(WaterMarked(i,j),1);
33          end
34      end
35
36      figure;
37      imshow(WaterMark,[]);
38      title('WaterMark');
39  end
```

对整个代码，我们由于已在前一个实验之中做过，因此在此进行简单解释：

LSB 方法

整体流程如下：

1. 首先，使用 imread 函数读取了两张图像，一张是载体图像"Lena.bmp"，另一张是要隐藏的水印图像"lion.bmp"。
2. 然后，使用 imshow 函数分别显示了这两张图像，以便于观察和调试。
3. 接着，调用 Hide 函数将水印图像嵌入到载体图像中，生成一个新的图像 WaterMarked。Hide 函数的具体实现是将水印图像的二进制数据嵌入到载体图像的最低有效位中。
4. 再调用 Extract 函数从 WaterMarked 图像中提取出隐藏的水印信息。Extract 函数的具体实现是从 WaterMarked 图像的最低有效位中提取出二进制数据，并将其转换为水印图像。
5. 最后，将提取出的水印图像赋值给变量 watermark，以便于后续的处理和分析。

1. 实验结果

实验结果如下所示：

图 2: 载体图像

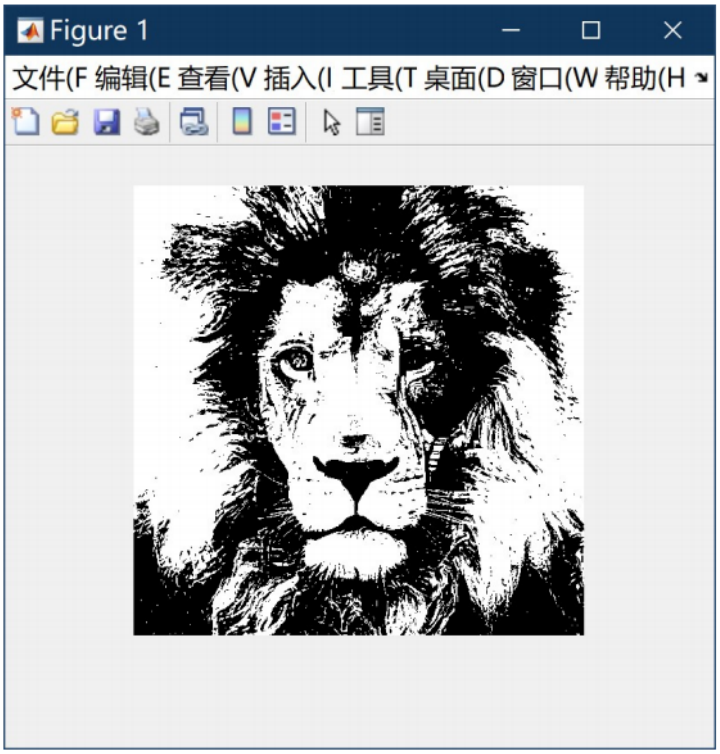


图 3: 隐藏图像

图 4: 隐藏后图像

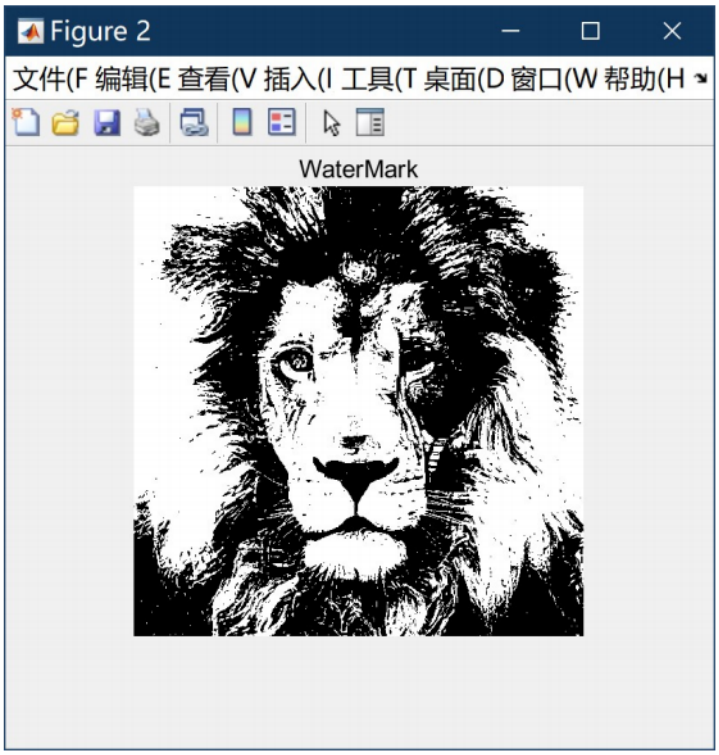


图 5: 解密后图像

可以看到，成功隐藏并提取了水印图像。

(二) 其他方法

1. 将秘密信息隐藏至文件尾部

隐藏函数中，依次读取载体图像、水印文件，并将二者依次写入伪装文件中。水印提取函数中，首先读取伪装图像头部信息中的文件长度（即载体图像文件长度）字段，以确定水印文件在整个文件中的偏移，之后，读取水印图像的长度字段，根据该长度，将水印图像读取到新的文件中。

代码如下所示:

```
function Method2()
    x=input('请输入载体图像: ','s'); %载体图像
    m=input('请输入要隐藏的秘密信息图像: ','s'); %水印图像
    WaterMarked=Hide(x,m);
    watermark=Extract(WaterMarked);
end

function WaterMarked = Hide(origin,watermark)
    fidx=fopen(origin,'r');
    [x,xlength]=fread(fidx,inf,'uint8');

    fidm=fopen(watermark,'r');
    [m,mlength]=fread(fidm,inf,'uint8');

    %写入所有数据
    fid=fopen('Method2_watermarked.bmp','w');
    fwrite(fid,x);
    fwrite(fid,m);

    fclose(fid);
    fclose(fidx);
    fclose(fidm);

    WaterMarked=imread('Method2_watermarked.bmp');

    figure;
    imshow(WaterMarked,[]);
    title("WaterMarked Image");
    WaterMarked="Method2_watermarked.bmp";
end

function WaterMark=Extract(WaterMarked)
    fid=fopen(WaterMarked,'r');
    fseek(fid,2,"bof");

    moffset=fread(fid,1,'uint32');
    fseek(fid,moffset+2,"bof");
    mlen=fread(fid,1,"uint32");
    fseek(fid,moffset,"bof");
```

```
    fWaterMark=fread(fid,mlen,'uint8');
    fclose(fid);
    mfid=fopen("method2_watermark.bmp",'w');
    fwrite(mfid,fWaterMark,'uint8');
    fclose(mfid);


    WaterMark=imread('method2_watermark.bmp');
    figure;
    imshow(WaterMark,[]);
    title("WaterMark");
end
```

重点解释关于隐藏和提取的函数:

将秘密信息隐藏至文件尾部

针对function WaterMarked

1. 首先，使用 fopen 函数打开载体图像文件和水印图像文件，分别读取它们的二进制数据。这里使用了’r’参数表示以只读方式打开文件。
2. 然后，将载体图像和水印图像的二进制数据写入到一个新的文件"Method2_watermarked.bmp" 中。这里使用了’w’参数表示以写入方式打开文件。
3. 最后，使用 imread 函数读取新生成的文件"Method2_watermarked.bmp"，并将其赋值给变量 WaterMarked。同时，使用 imshow 函数显示生成的水印图像，并将其命名为"WaterMarked Image"。
4. 最后，将文件名"Method2_watermarked.bmp" 赋值给变量 WaterMarked，以便于后续的处理和分析。

针对function WaterMarked

1. 首先，使用 fopen 函数打开包含水印信息的图像文件，并使用 fseek 函数定位到文件的第二个字节处。这里使用了’bof’参数表示从文件开头开始计算偏移量。
2. 然后，从文件中读取水印信息的偏移量 moffset 和长度 mlen。这里使用了’uint32’参数表示读取 32 位的无符号整数。
3. 接着，使用 fseek 函数定位到水印信息的起始位置，并从文件中读取水印信息的二进制数据 fWaterMark。
4. 然后，将读取到的二进制数据写入到一个新的文件"method2_watermark.bmp" 中。这里使用了’w’参数表示以写入方式打开文件。
5. 最后，使用 imread 函数读取新生成的文件"method2_watermark.bmp"，并将其赋值给变量 WaterMark。同时，使用 imshow 函数显示提取出的水印图像，并将其命名为"WaterMark"。

结果如下:

图 6: 隐藏后图像

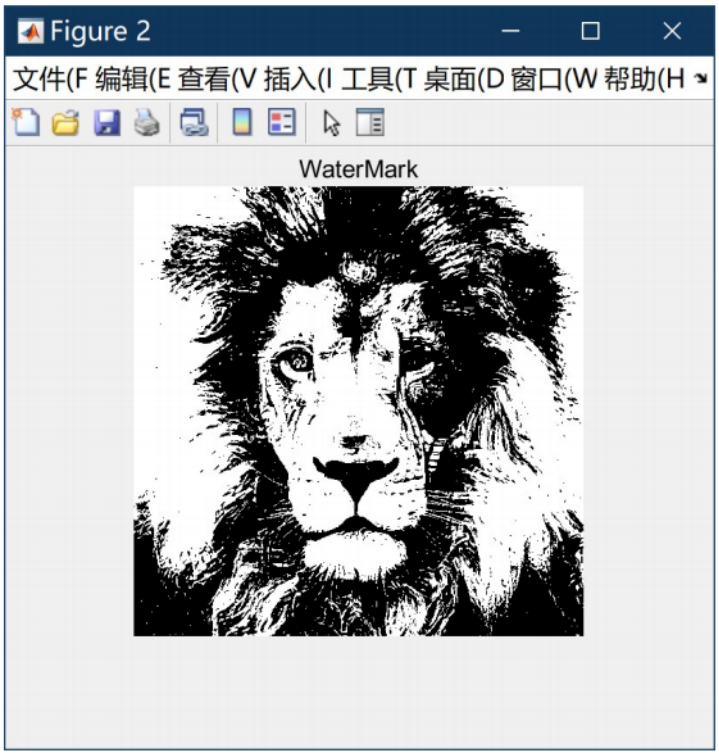


图 7: 解密后图像

然后，再观察一下，文件大小，发现隐藏后的载体文件大小正好等于两个文件大小相加之和：

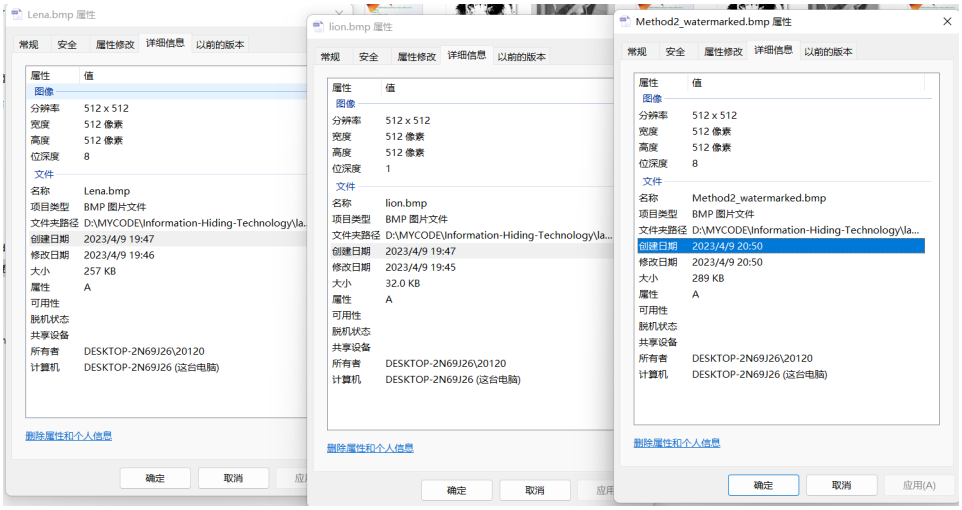


图 8: 大小对比

2. 将秘密信息隐藏至文件头与图像数据之间

代码和上一个方法差不多，仅仅是改变了偏移位置，因此不做太多赘述。

```
function Method3()
    x=input('请输入载体图像：','s'); %载体图像
    m=input('请输入要隐藏的秘密信息图像：','s'); %水印图像
    WaterMarked=Hide(x,m);
    watermark=Extract(WaterMarked);
end

function WaterMarked = Hide(origin,watermark)
    fidx=fopen(origin,'r');

    %获取图像数据的偏移
    fseek(fidx,10,"bof");
    xoffset=fread(fidx,1,"uint32");

    %读取出文件头
    fseek(fidx,0,"bof");
    [xhead,xhlength]=fread(fidx,xoffset,'uint8');
    %读取出图像数据
    [xtail,xtlength]=fread(fidx,inf,"uint8");

    fidm=fopen(watermark,'r');
    [m,mlength]=fread(fidm,inf,'uint8');

    %写入所有数据，将水印信息放到文件头和图像数据之间的位置
    fid=fopen('Method3_watermarked.bmp','w');
    fwrite(fid,xhead);
    fwrite(fid,m);
```

```
28      fwrite(fid,xtail);
29
30      %还要更改图像数据偏移的字段，加上水印信息的文件长度
31      fseek(fid,10,"bof");
32      fwrite(fid,xoffset+mlength,"uint32");
33
34      fclose(fid);
35      fclose(fidx);
36      fclose(fidm);
37
38      WaterMarked=imread('Method3_watermarked.bmp');
39
40      figure;
41      imshow(WaterMarked,[]);
42      title("WaterMarked Image");
43      WaterMarked="Method3_watermarked.bmp";
44  end
45
46  function WaterMark=Extract(WaterMarked)
47      fid=fopen(WaterMarked,'r');
48
49      %获取载体图像的实际大小
50      fseek(fid,2,"bof");
51      xlen=fread(fid,1,'uint32');
52
53      %读取整个文件，得到整个文件的大小
54      fseek(fid,0,"bof");
55      [A,truelen]=fread(fid,inf,'uint8');
56
57      %相减，即可得到秘密信息的大小
58      mlen=truelen-xlen;
59
60      %获取载体图像数据的偏移，减去秘密信息的大小，就得到了秘密信息的偏移
61      fseek(fid,10,"bof");
62      xoffset=fread(fid,1,"uint32");
63      moffset=xoffset-mlen;
64
65      %由此，即可读出秘密信息
66      fseek(fid,moffset,"bof");
67      m=fread(fid,mlen,"uint8");
68
69      fidm=fopen("method3_watermark.bmp","w");
70      fwrite(fidm,m,"uint8");
71      fclose(fidm);
72
73      fclose(fid);
74
75      WaterMark=imread('method3_watermark.bmp');
```

```
76      figure;
77      imshow(WaterMark,[]);
78      title("WaterMark");
79  end
```

隐藏函数中，读取载体图像的头部和数据部分、水印文件，按照载体图像头部-水印文件-载体图像数据部分的顺序写入伪装文件中。

水印提取函数中，读取伪装图像头部信息中的文件长度（即载体图像文件长度）字段，并通过读取整个文件来获取整个文件真正的长度，二者相减，即可确定水印文件的大小，再用头部信息中的图像数据偏移字段减去此大小，即可确定水印文件在整个文件中的偏移，根据该长度、偏移，即可将水印图像读取到新的文件中。

结果如下：

图 9: 隐藏后图像

图 10: 解密后图像

然后，再观察一下，文件大小，发现隐藏后的载体文件大小正好等于两个文件大小相加之和：

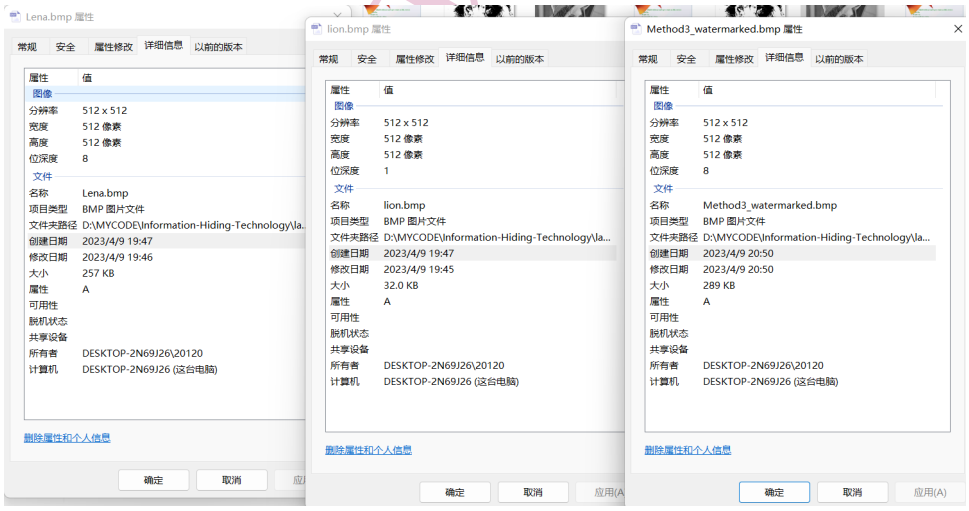


图 11: 大小对比

3. 将秘密信息隐藏至文件头的保留字段中

bmp 文件头部的第 6-9 字节，这四个字节是保留字节，一般设置为 0，这个地方可以用于隐藏秘密信息。代码实现如下：

```
function Method4()
    x=input('请输入载体图像：','s'); %载体图像
    m=input('请输入要隐藏的秘密信息(int型)：'); %秘密信息
```

```
    WaterMarked=Hide(x,m);
    watermark=Extract(WaterMarked);
end

function WaterMarked = Hide(origin,watermark)
    fidx=fopen(origin,'r');

    fseek(fidx,0,"bof");
    [x,xlength]=fread(fidx,inf,'uint8');

    fid=fopen('Method4_watermarked.bmp','w');
    fwrite(fid,x);

    %加入秘密信息
    fseek(fid,6,"bof");
    fwrite(fid,watermark,"uint32");

    fclose(fid);
    fclose(fidx);

    WaterMarked=imread('Method4_watermarked.bmp');

    figure;
    imshow(WaterMarked,[]);
    title("WaterMarked Image");
    WaterMarked="Method4_watermarked.bmp";
end

function WaterMark=Extract(WaterMarked)
    fid=fopen(WaterMarked,'r');

    %获取秘密信息
    fseek(fid,6,"bof");
    m=fread(fid,1,'uint32');

    fclose(fid);

    WaterMark=m;

    WaterMark
end
```

重点解释关于隐藏和提取的函数:

将秘密信息隐藏至文件尾部

针对function WaterMarked

1. 首先，使用 fopen 函数打开载体图像文件，并使用 fseek 函数定位到文件的开头处。

这里使用了’bof’参数表示从文件开头开始计算偏移量。

2. 然后，从文件中读取载体图像的二进制数据 x，并计算其长度 xlength。这里使用了’uint8’参数表示读取 8 位的无符号整数。
3. 接着，使用 fopen 函数创建一个新的文件”Method4_watermarked.bmp”，并将载体图像的二进制数据写入到该文件中。
4. 然后，使用 fseek 函数定位到文件的第 6 个字节处，将要隐藏的水印信息写入到文件中。这里使用了’uint32’参数表示写入 32 位的无符号整数。
5. 最后，关闭文件句柄，使用 imread 函数读取新生成的文件”Method4_watermarked.bmp”，并将其赋值给变量 WaterMarked。同时，使用 imshow 函数显示生成的水印图像，并将其命名为”WaterMarked Image”。
6. 最后，将文件名”Method4_watermarked.bmp” 赋值给变量 WaterMarked，以便于后续的处理和分析。

针对function WaterMarked

1. 首先，使用 fopen 函数打开包含水印信息的图像文件，并使用 fseek 函数定位到文件的第 6 个字节处。这里使用了’bof’参数表示从文件开头开始计算偏移量。
2. 然后，从文件中读取隐藏的水印信息 m。这里使用了’uint32’参数表示读取 32 位的无符号整数。
3. 最后，关闭文件句柄，将读取到的水印信息赋值给变量 WaterMark，并输出到命令行窗口中。

隐藏函数中，读取载体图像，将其写入伪装文件中，之后，将伪装图像的偏移为 6 的四个字节修改为秘密信息。

水印提取函数中，只需读取伪装图像偏移为 6 的四个字节，即可得到秘密信息。运行代码，结果如下：结果如下：

图 12: 隐藏后图像

```
>> Method4
请输入载体图像：Lena.bmp
请输入要隐藏的秘密信息(int型)：2012026

WaterMark =

     2012026
```

图 13: 学号隐藏

然后，再观察一下，文件大小，发现隐藏后的载体文件正好等于隐藏前的文件大小:

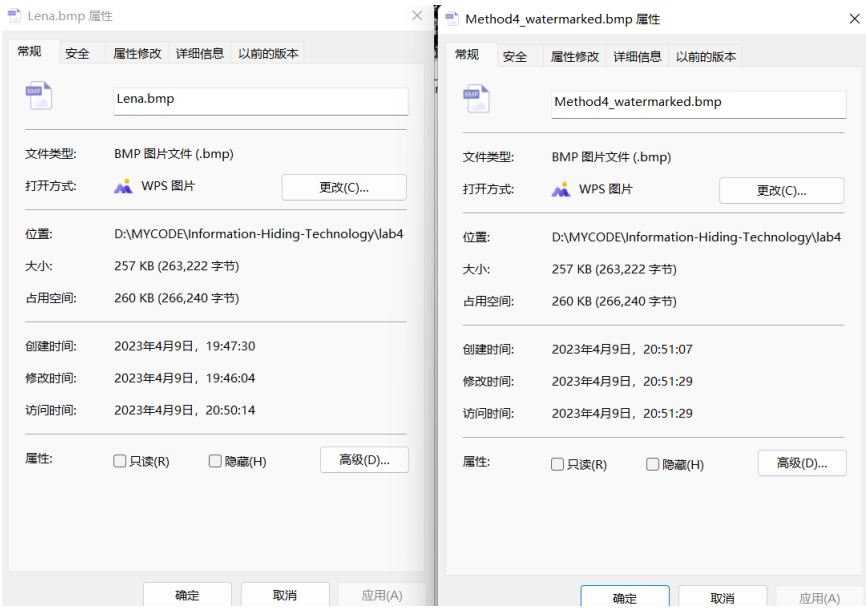


图 14: 大小对比

三、 实验心得

在本次实验中, 我们学习了多种图像信息隐藏和提取的方法, 包括基于 LSB 方法、基于 DCT 方法、基于文件 IO 的方法等。通过实验，我对这些方法有了更深入的了解和体会。

首先，基于 LSB 方法的图像信息隐藏和提取是最简单、最常用的一种方法。它的原理是将要隐藏的信息嵌入到载体图像的最低有效位中，从而实现信息的隐藏。这种方法的优点是实现简单，嵌入的信息容量较大，但是安全性较低，易被攻击者检测到和破解。

其次，基于 DCT 方法的图像信息隐藏和提取是一种比较高级的方法。它的原理是将要隐藏的信息嵌入到载体图像的 DCT 系数中，从而实现信息的隐藏。这种方法的优点是安全性较高，嵌入的信息容量较大，但是实现较为复杂，需要对图像进行 DCT 变换和量化。

最后，基于文件 IO 的图像信息隐藏和提取是一种比较简单的方法。它的原理是将要隐藏的信息直接写入到图像文件中，从而实现信息的隐藏。这种方法的优点是实现简单，但是安全性较低，易被攻击者检测到和破解。

通过本次实验，我深刻认识到了信息隐藏和提取的重要性和难度。在实际应用中，我们需要根据具体的需求和安全要求选择合适的信息隐藏和提取方法，并结合其他的加密技术来提高信息的安全性。同时，我们还需要注意信息隐藏和提取的效率和容量，以便于在实际应用中得到更好的体验和效果。

总之，本次实验让我对图像信息隐藏和提取有了更深入的了解和体会，也让我认识到了信息安全的重要性和难度。我相信这些知识和经验对我的未来学习和工作都会有很大的帮助。