第5章 基于信息隐藏的多格式文件加密算法应用

CHAPTER 5

5.1 应用背景

信息隐藏一般是指将秘密信息隐藏在其他载体文件中，例如常见的将信息嵌入到图像/音频/视频等多媒体文件，然后通过载体文件的传输实现秘密信息的传递。不同于信息加密带来的视觉差异，信息隐藏的特点是可以将秘密信息隐藏到外观正常的文件中且不发生明显的变化，一般不会引起人们的注意，在不知觉间完成了秘密信息的隐藏和传递，增强了通信的隐蔽性。图像是互联网中常用的多媒体文件，具有明显的可视化效果，且一般存在较多的视觉冗余，适合作为信息隐藏的载体进行嵌入和传输。

人类的视觉系统具有一定的视觉不敏感性，特别是对于图像颜色细微变化一般难以察觉，而这也是选择将图像作为载体进行信息隐藏的基本依据之一。图像最低有效位(Least Significant Bit，LSB)隐藏是一种简单高效的信息隐藏方法，将对载体图像做较小的改变即可嵌入大量的秘密信息，且在视觉上不会对载体图像带来明显的视觉变化。在文件编码方面，Base64是当前网络通信传输中最常用的编码方式之一，使用64个字符“A-Z、a-z、0-9、+、/”的编码方式来表示二进制数据，具有跨平台、格式统一以及易于传输的优点。

传统的LSB信息隐藏往往是将图像或文本隐藏到载体图像，具有一定的局限性。为了能够将多格式文件隐藏到载体图像，本案例采用Base64编码的思路将多格式的文件进行Base64编码，增加字符错位加密后再以字符串的方式嵌入载体图像，进而可完成统一的嵌入和提取操作，形成面向多格式文件的信息隐藏及加密应用。

5.2 信息隐藏加密

5.2.1 LSB信息隐藏

LSB信息隐藏的核心思想是对载体图像的像素最低有效位进行嵌入，只改变最低位或其他低位的位元值，将秘密信息在指定低位进行叠加，避免影响像素高位的位元值，对图像

整体而言属于细微的变化，人眼一般难以直接分辨经 LSB 信息隐藏后的图像。对于待嵌入的秘密信息，可通过二进制流文件读取的方式将其转换为 0、1 序列，进而可以方便地将其叠加到像素的低位，实现对载体图像的 LSB 信息隐藏。综上所述，LSB 信息隐藏的流程如图 5-1 所示。LSB 信息隐藏需要判断可嵌入位置数和待嵌入信息长度的大小，只有可嵌入位置数超出待嵌入信息长度时才能执行低位嵌入。数字图像具有较多的信息容量，且存在一定的视觉冗余，一般能满足秘密信息的嵌入要求。本案例选择最简单的文本文件作为待嵌入信息源，将其嵌入到载体图像最低位，关键步骤及代码叙述如下。

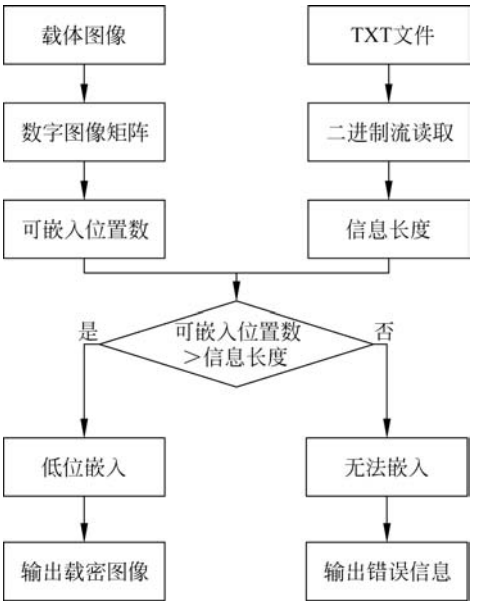

图 5-1　LSB 信息隐藏流程图

1. 文件读取

读取载体图像和待嵌入的秘密文件，注意将秘密文件以二进制流的方式进行读取。

```
% 设置待加密的 txt 文件
txt_file = fullfile(pwd, 'demo.txt');
% 读取 txt 文件
fid = fopen(txt_file,'r');
[msg, msg_num] = fread(fid,'ubit1');
fclose(fid);
% 设置载体图像
img = imread(fullfile(pwd, 'images', 'lena.jpg'));
imo = img;
rgb_flag = false;
if ndims(img) == 3
    % 如果是 RGB,将第三维度水平叠加到第二维度
```

```
    rgb_flag = true;
    img = [img(:,:,1) img(:,:,2) img(:,:,3)];
end
img = double(img);
```

为了演示基本的实验流程，选择彩色的 lena 图像作为载体，并通过第三维度水平叠加到第二维度的方式将其转换为二维矩阵，使用简单的 txt 文件作为秘密文件，具体如图 5-2 和图 5-3 所示。

图 5-2　载体图像

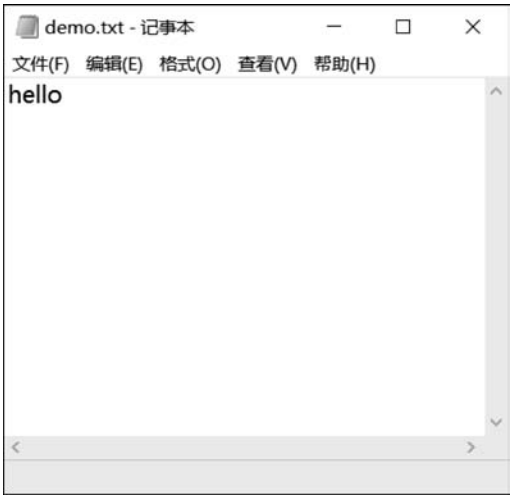

图 5-3　待隐藏文件

2. 嵌入条件判断

通过二进制流的方式读取秘密文件能得到 0、1 序列，为了保证序列能有效地嵌入载体图像的最低位，需要判断图像像素个数与序列长度的关系，如果图像像素个数小于序列长度，则给出报错信息，关键代码如下。

```
% 判断是否符合嵌入条件
[M, N] = size(img);
if msg_num > M * N
    error('载体图像维度无法满足隐藏信息长度要求，请核查!');
end
```

3. 最低位嵌入

遍历秘密信息的二进制序列，将其嵌入到图像像素的最低位。选择最简单的经典 LSB 嵌入方法，先将图像像素最低位设置为 0，然后将秘密信息叠加到最低位，这样即可保持最低位嵌入的有效性，关键代码如下。

```
% 遍历扫描嵌入
```

```
k = 1;
for c = 1:N
    for r = 1:M
        if k > msg_num
            % 如果嵌入完毕
            break;
        end
        % 最低位嵌入
        img(r,c) = img(r,c) - mod(img(r,c),2) + msg(k,1);
        k = k + 1;
    end
end
```

4. 含密图像保存

秘密信息嵌入后，可根据其 RGB 标记进行矩阵重构，并保存到文件，关键代码如下所示。

```
% 嵌入隐藏信息后的图像
if rgb_flag == true
    % 还原 RGB 结构
    sz = size(img);
    img = cat(3, img(:,1:sz(2)/3), img(:,1 + sz(2)/3:2 * sz(2)/3), img(:,1 + 2 * sz(2)/3:end));
end
% 写出文件
imwrite(uint8(img), 'zt_m.png')
% 计算 psnr
peaksnr = psnr(uint8(img),imo)
```

经过如上步骤运行后，将得到嵌入秘密信息的图像文件，如图 5-4 所示。嵌入秘密信息后的图像在视觉上无法分辨与原图(图 5-2)的差别，对应的 PSNR 值为 57.7，由此可见采用经典的 LSB 信息隐藏具有计算简单、失真度低的特点。

图 5-4　含密图像

5.2.2　Base64 编解码

Base64 是一种基于 64 个可打印字符来表示二进制数据的表示方法，字符内容包括“A-Z、a-z、0-9、＋、/”，是一种可逆的编码方式。通过二进制文件流的方式读取文件后，可得到二进制序列，采用 Base64 编解码即可完成文件流与字符串的相互转换，便于进行统一的字符串读写，避免文件形式的隔离性。例如，将图像编码为 Base64 字符串，存储到数据库，待使用时直接读取数据库即可得到编码字符串并解码为可显示的图像形式，这样即可减少图像文件存储维护的成本，提高系统的数据完整性。

Base64 是经典的编解码方法，可直接调用对应的函数接口来实现调用，例如在 Java 环境下使用 Base64. encodeBase64、Base64. decodeBase64 完成编解码，在 Python 环境下使用 base64. b64encode、base64. b64decode 完成编解码，在 MATLAB 环境下使用 matlab. net. base64encode、matlab. net. base64decode 完成编解码。为了保持程序的一致性，对上面提到的 txt 文件在 MATLAB 环境下通过文件流形式读取并进行 Base64 编解码实验，关键代码如下。

```
% 设置待加密的 txt 文件
txt_file = fullfile(pwd, 'demo.txt');
% 读取 txt 文件
fid = fopen(txt_file,'r');
[msg, msg_num] = fread(fid);
fclose(fid);
% base64 编码
bs_code = matlab.net.base64encode(msg)
% base64 解码
bs_decode = matlab.net.base64decode(bs_code);
fid2 = fopen('demo2.txt', 'w');
fwrite(fid2, bs_decode(:));
fclose(fid2);
```

运行此段程序，可对示例 txt 文件进行文件流读取并编码为 Base64 字符串，然后对此字符串进行 Base64 解码，保存为另外一个 txt 文件，运行效果如图 5-5 所示。按照 txt 格式写出解码文件后，可以发现内容与原文件完全相同，这也说明了对文件进行 Base64 编解码的有效性。

更进一步对 docx 文件进行此项操作，查看编解码后的效果，具体如图 5-6 所示。按照与编码文件同样的文件格式写出解码文件后，可以发现内容与原文件完全相同，类似地也可以推广到其他格式的文件。

因此，对多格式的文件按文件流方式读取并进行 Base64 编码，可将其统一为字符串形式，但需注意在解码写出文件时保持文件格式的一致性。

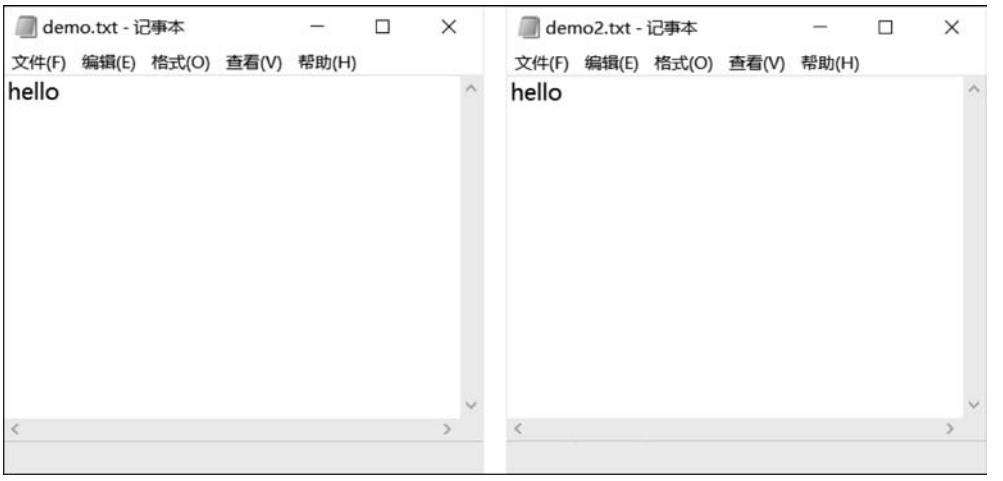

图 5-5 对 txt 文件进行 base64 编解码效果对比

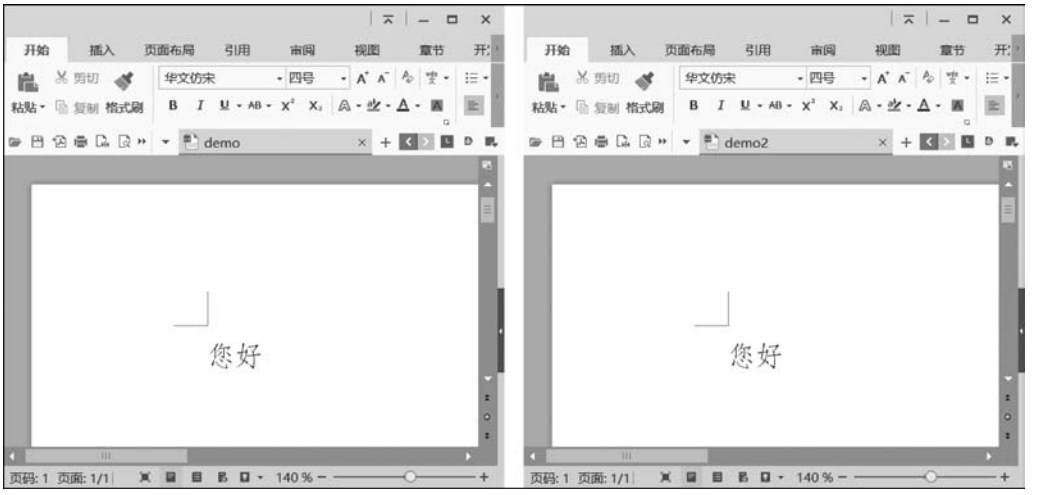

图 5-6 对 docx 文件进行 base64 编解码效果对比

5.2.3 多格式文件隐藏加密

不同格式的文件可通过文件流 Base64 编码的方式统一做成字符串形式，进而可将文件格式信息和编码字符串组合存储为 txt 文件，再对其进行 LSB 信息隐藏即可完成多格式的文件隐藏。在此过程中，参考之前的混沌加密案例，可对 txt 文件内容通过 Logistic 混沌加密的方式进行置乱，进而实现多格式的文件隐藏加密效果。综上所述，多格式文件隐藏加密的流程如图 5-7 所示。

多格式文件隐藏加密的预操作过程是对文件流进行 Base64 编码及 Logistic 混沌加密，进而可加入字符串结束的标志位并将其统一存储为 txt 格式的文件，之后即可参照 LSB 信

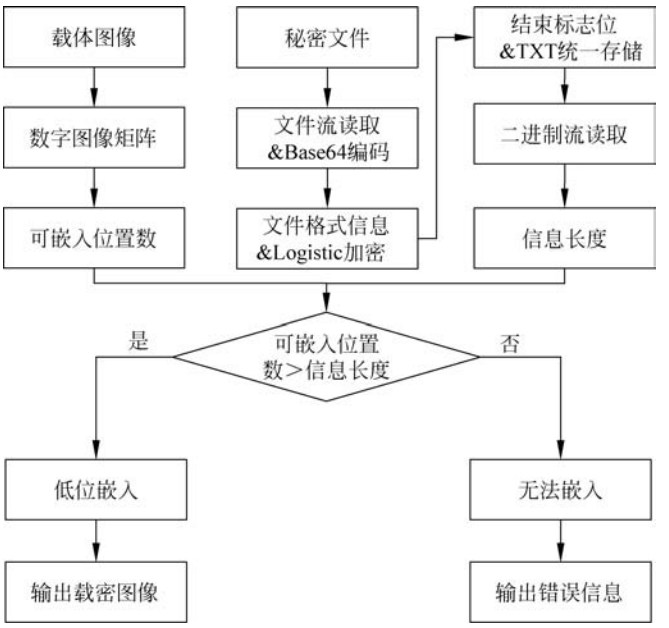

图 5-7　多格式文件隐藏加密流程图

息隐藏流程(见图 5-1)进行信息隐藏。下面以某 docx 文件为例,根据多格式文件隐藏加密流程进行实验,主要步骤如下。

1. 秘密文件 Base64 编码

以文件流的方式对秘密文件进行读取,并对其进行 Base64 编码,关键代码如下所示。

```
% 设置待加密的文件
in_file = fullfile(pwd, 'demoz.docx');
% 读取文件
fid = fopen(in_file,'r');
[msg, msg_num] = fread(fid);
fclose(fid);
% base64 编码
bs_code = matlab.net.base64encode(msg)
```

运行此段程序,可对待加密的文件读取并编码为 Base64 字符串,其中待加密文件如图 5-8 所示。

经过 Base64 编码后,得到的字符串长度为 21608,选中载体图为彩色的 512×512 大小的 lena.jpg,像素个数为 512×512×3=786432,所以可满足此编码结果的隐藏要求。

2. 文件格式信息及 Logistic 混沌加密

前面讨论过,Base64 解码后需要确定文件的格式才能将文件流正确写出,所以这里将文件格式信息与 Base64 编码字符串进行融合,关键代码如下所示。

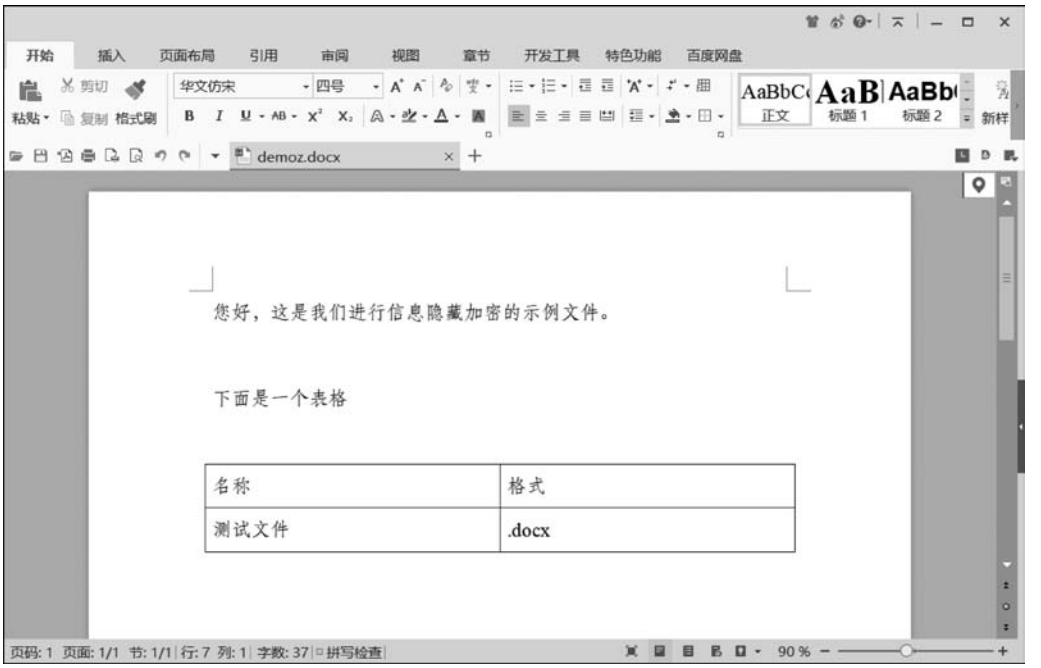

图 5-8　待加密文件

```
% 文件格式
[~, ~, ext] = fileparts(in_file);
% 字符串融合
bs_code = [ext ';' bs_code];
```

根据 Base64 编码的字符集内容，可通过将文件格式加入“;”分隔符的方式将二者进行融合。然后，采用 Logistic 混沌加密的方式进行循环置乱加密，关键代码如下。

```
% Logistic 混沌加密
num = length(bs_code);
% 分支参数
u = 3.9999;
% 初值设置
x0 = 1/sqrt(2);
% 保留 4 位小数
x0 = str2num(sprintf('%.4f', x0));
xn(1) = x0;
for i = 1 : num + 1e3
    % 迭代计算
    xn(i + 1) = u * xn(i) * (1 - xn(i));
end
% 取指定长度的序列
xn = xn(length(xn) - num + 1:end);
% 对应到位置序列
[~, ind] = sort(xn,'descend');
```

```
% 循环置乱加密
loop = 1e3;
coef_vec1 = bs_code;
for i = 1:loop
    for j = 1: num
        % 置乱加密
        coef_vec2(j) = coef_vec1(ind(j));
    end
    coef_vec1 = coef_vec2;
end
% 加密结果
coef_vec = coef_vec1;
```

这里设置 Logistic 的分支系数为 3.9999，初值为 0.7071，循环次数为 1000，这也是 Logistic 混沌加密的关键参数，待解密时也需要同样的设置。之后，可将此加密后的字符串进行 txt 存储，作为待隐藏的文件，关键代码如下。

```
% 写出 txt 文件
fid = fopen('demoz.txt', 'wt');
fprintf(fid,'%s#', coef_vec);
fclose(fid);
```

运行后将得到 txt 文件，存储了加密后的字符串内容且以 # 作为字符串结束的标志，如图 5-9 所示。

图 5-9　待隐藏文件

3. 信息隐藏

待隐藏文件统一为 txt 格式，可直接应用 LSB 信息隐藏的步骤进行相关操作，执行后得

到的含密图像(即包含隐藏信息的图像)如图 5-10 所示。嵌入秘密信息后的图像在视觉上无法分辨与原图(图 5-2)的差别,对应的 PSNR 值为 57.7,由此可见采用 Base64 编码及 Logistic 混沌加密后进行 LSB 信息隐藏也能得到正确的加密结果。

图 5-10　含密图像

5.3　信息提取解密

5.3.1　LSB 信息提取

LSB 信息提取是 LSB 信息隐藏的逆过程,针对前面对测试文件 demo.txt 进行的信息隐藏,读取含有隐藏信息的图像,然后遍历像素解析 0、1 序列,并将其通过文件流的方式写入 txt 文件。综上所述,LSB 信息隐藏的流程如图 5-11 所示。

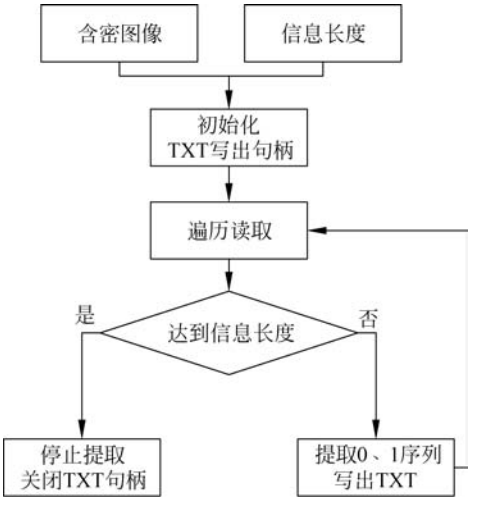

图 5-11　LSB 信息提取流程图

如图 5-11 所示，LSB 信息提取的关键在于按照指定的信息长度对图像遍历，提取 0、1 序列写入文件流，关键代码如下。

```
img = imread('zt_m.png');
if ndims(img) == 3
    % 如果是 RGB,将第三维度水平叠加到第二维度
    img = [img(:,:,1) img(:,:,2) img(:,:,3)];
end
img = double(img);
[M,N] = size(img);
% 写出提取出的 txt 文件
fpath = './demo_tiqu.txt';
% 设置空 txt 文件
fid = fopen(fpath,'wt');
fclose(fid);
fid = fopen(fpath,'w + ');
% 信息的长度
msg_num = 40;
p = 0;
h = waitbar(0, '', 'Name', '正在处理...');
for c = 1:N
    waitbar(c/N, h, sprintf('Process: %d%%', round(c/N * 100)));
    for r = 1:M
        p = p + 1;
        % 判断是否停止
        if p > msg_num
            break;
        end
        % 按文件流的方式写入
        if bitand(img(r,c),1) == 1
            fwrite(fid,1,'ubit1');
        else
            fwrite(fid,0,'ubit1');
        end
    end
end
waitbar(0.99, h, sprintf('Process: %d%%', round(0.99 * 100)));
close(h);
% 关闭文件流
fclose(fid);
```

运行此段程序，可对前面隐藏信息后的图像进行读取和解析，提取隐藏的信息并写文件，具体效果如图 5-12 和图 5-13 所示。

按照 txt 格式将文件提取后，可以发现内容与原文件完全相同，这也说明了对文件进行 LSB 信息隐藏的有效性。同时，需要注意的是这里设置了隐藏序列的长度，用于判断是否停止提取，这作为一个数值信息在传递过程中容易丢失或混淆，为此可考虑加入特殊字符标记的方法来判断停止提取，5.3.2 节多格式文件提取解密过程中将进行此项处理。

图 5-12 待隐藏文件

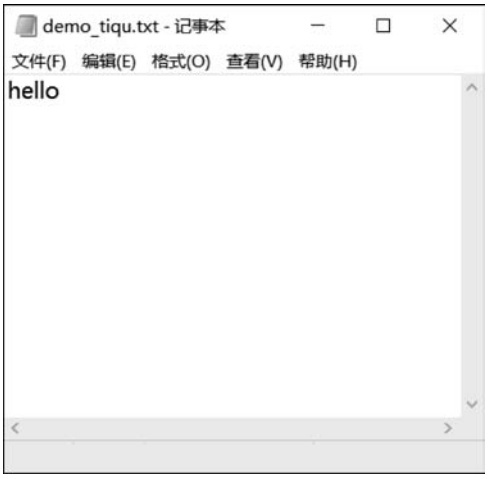

图 5-13 提取的文件

5.3.2 多格式文件提取解密

根据 LSB 信息隐藏及提取的过程，可将 txt 以二进制文件流的形式隐藏和提取，而 txt 文件可存放任意的字符串信息。因此，可对多格式文件进行 Base64 编码和混沌加密得到字符串信息，将其写入 txt 文件，然后可按照二进制文件流的形式进行信息隐藏和提取，最后经混沌解密和 Base64 解码即可还原文件。为了减少文件格式、编码长度的信息存储和传递，将格式信息写入待加密文件的格式信息也写入到 Base64 编码后的字符串，并对混沌加密后的字符串添加特殊字符标记“#”，用于表示序列结束。综上所述，多格式文件提取解密的流程如图 5-14 所示。

如图 5-14 所示，多格式文件提取解密的关键在于未出现停止符标记的情况下对图像遍历，提取 0、1 序列写入文件流，通过停止符标记拆分字符串进行混沌解密，获取图像格式并进行 Base64 解码，写出指定格式的文件。下面我们以前面进行的某 docx 文件隐藏加密为例，根据多格式文件隐藏解密流程进行实验，主要步骤如下。

1. 图像读取

与前面的图像读取方式相同，读取含密图像，如果是 RGB 格式则将其第三维度水平叠加到第二维度，并获取其行列数[M,N]。

2. 信息提取

与前面的 LSB 信息提取过程类似，遍历提取像素低位，写出二进制的 txt 文件流，只是这里我们加入了停止符标志位“#”用于判断是否停止提取，并对字符串进行截取，只保留有效的字符串内容，关键代码如下。

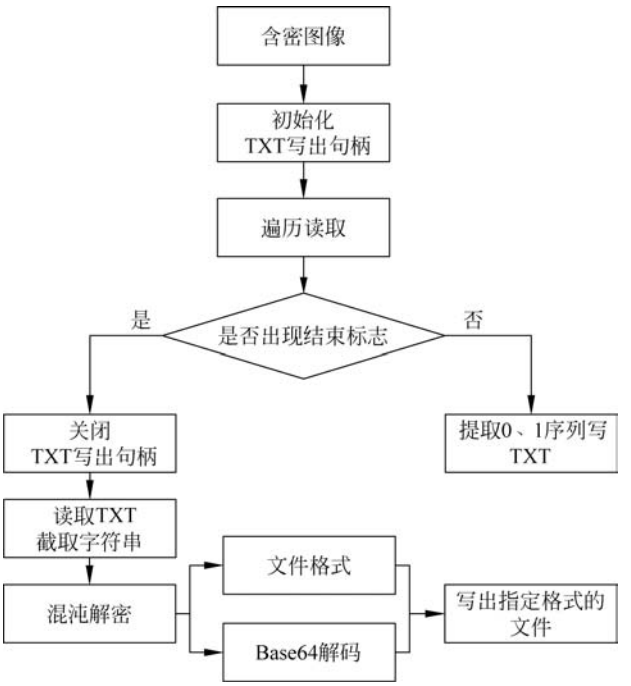

图 5-14　多格式文件提取解密流程图

```
% 临时写出的 txt 文件
fpath = './tmp.txt';
fid = fopen(fpath,'wt');
fclose(fid);
fid = fopen(fpath,'w + ');
h = waitbar(0, '', 'Name', '正在处理...');
for c = 1:N
    waitbar(c/N, h, sprintf('Process: %d %% ', round(c/N * 100)));
    for r = 1:M
        % 按文件流的方式写入
        if bitand(img(r,c),1) == 1
            fwrite(fid,1,'ubit1');
        else
            fwrite(fid,0,'ubit1');
        end
    end
    % 判断是否停止
    fid2 = fopen(fpath, 'r');
    now_msg = fgetl(fid2);
    fclose(fid2);
    % 检查停止符标记
    ind = strfind(now_msg, '#');
```

```
    if ~isempty(ind)
        % 出现停止符,停止提取,截取有效内容
        fclose(fid);
        fid = fopen(fpath,'wt');
        % 保留停止符之前的内容
        fprintf(fid, '%s', now_msg(1:ind(1) - 1));
        fclose(fid);
        fid = 0;
        break;
    end
end
waitbar(0.99, h, sprintf('Process: %d%%', round(0.99 * 100)));
close(h);
if fid~ = 0
    fclose(fid);
end
```

运行此段程序,可在不设置信息长度的前提下对隐藏信息后的图像进行读取和解析,判断是否达到停止标志,提取隐藏的信息并写出到 txt 文件。但此时提取出的内容依然是经混沌加密后的字符串,下面根据设置的 Logistic 分支参数、初值和循环次数进行混沌解密。

3. 混沌解密

参照混沌加密过程,此处设置 Logistic 分支参数为 3.9999,初值为 1/sqrt(2),循环次数为 1000 进行的混沌解密,关键代码如下。

```
% 提取待解密的内容
fid2 = fopen(fpath, 'r');
bs_code = fgetl(fid2);
fclose(fid2);
% Logistic 混沌解密
num = length(bs_code);
% 分支参数
u = 3.9999;
% 初值设置
x0 = 1/sqrt(2);
% 保留 4 位小数
x0 = str2num(sprintf('%.4f', x0));
xn(1) = x0;
for i = 1 : num + 1e3
    % 迭代计算
    xn(i + 1) = u * xn(i) * (1 - xn(i));
end
% 取指定长度的序列
xn = xn(length(xn) - num + 1:end);
% 对应到位置序列
[~, ind] = sort(xn,'descend');
% 循环置乱还原
loop = 1e3;
```

```
coef_vec1 = bs_code;
coef_vec = coef_vec1;
% 循环解密
for i = 1:loop
    coef_vec2 = coef_vec;
    for j = 1:num
        % 置乱还原
        coef_vec(ind(j)) = coef_vec2(j);
    end
end
```

运行此段程序，可对前面提取出的字符串进行混沌解密，注意这里如果参数设置错误将无法得到正确的字符串结果。通过混沌解密后，可得到包含文件格式及 Base64 编码的字符串，最后对该字符串进行指定格式的 Base64 解码即可还原秘密文件。

4. 秘密文件 Base64 解码

通过对设置的字符串间隔符进行解析，提取出文件格式和 Base64 字符串，并将其解码写出，关键代码如下。

```
% 解析字符串
ind = strfind(coef_vec, ';');
ext = coef_vec(1:ind(1) - 1);
bs_code = coef_vec(ind(1):end);
% Base64 解码
bs_decode = matlab.net.base64decode(bs_code);
fid2 = fopen(['demo_tiqu'ext], 'w');
fwrite(fid2, bs_decode(:));
fclose(fid2);
```

运行此段程序，可得到文件格式和 Base64 字符串，将其按指定格式保存，运行效果如图 5-15 所示。按照解析出的文件格式及 Base64 编码内容进行解码并写出文件，可以发现内容与原文件完全相同，能够产生 docx 文件隐藏加密的效果。更进一步，考虑到对文件流进行 Baes64 编解码的通用性，类似的过程也可以推广到其他格式的文件。读者可以尝试设置不同的混沌加密参数，形成个性化的多格式文件隐藏加密工具。

图 5-15　多格式文件提取解密的效果对比

5.4 集成应用开发

为了更好地集成对比不同步骤的处理效果，贯通整体的处理流程，本案例开发了一个GUI界面，集成加密隐藏、提取解密等关键步骤，并显示处理过程中产生的中间结果图像。其中，集成应用的界面设计如图5-16所示。

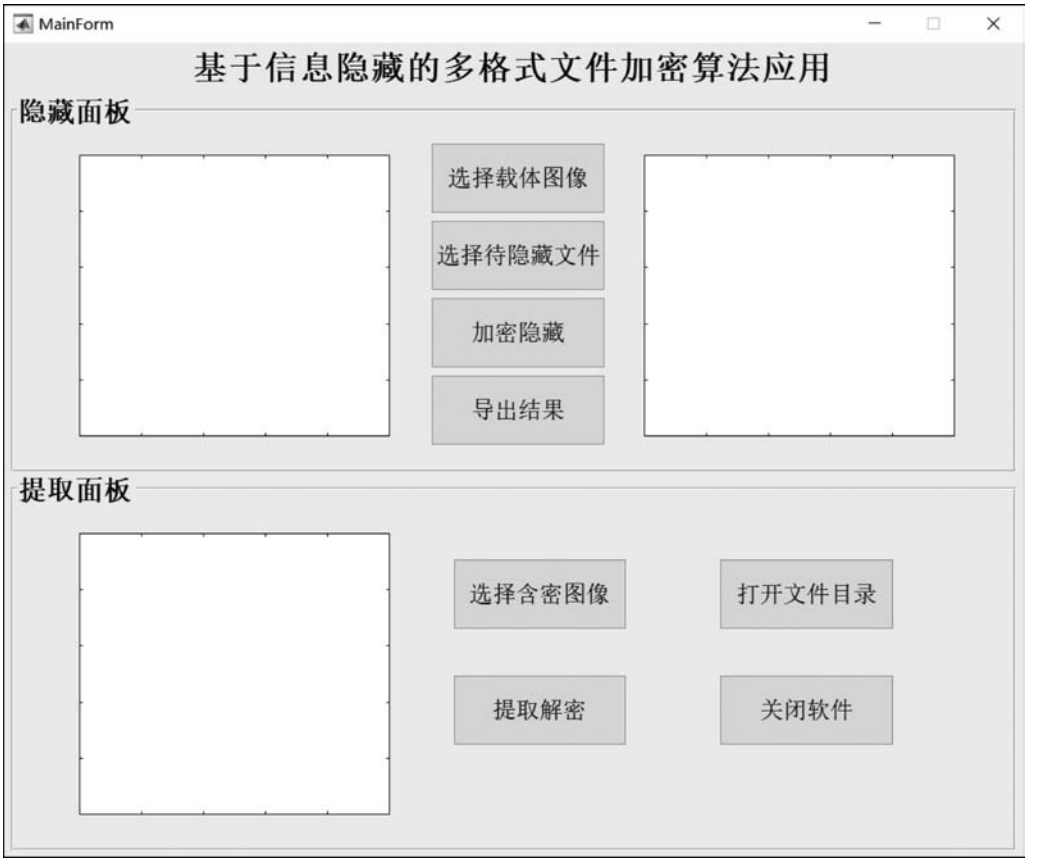

图5-16 界面设计

1. 隐藏模块

单击“选择载体图像”按钮可弹出文件选择对话框，可选择待处理图像并显示到左侧的窗口；单击选“择隐藏文件”按钮可弹出文件选择对话框，可选择待处理文件及存储文件路径；单击“加密隐藏”按钮可读取选择的待隐藏文件，进行Base64编码及Logistic混沌加密，通过LSB信息隐藏写入到载体图像得到含密图像，显示在右侧窗口；单击“导出结果”按钮即可将含密图导出存储。

2. 提取模块

单击“选择含密图像”按钮可弹出文件选择对话框，可选择待处理图像并显示在左侧的窗口；单击“提取解密”按钮可对含密图像进行 LSB 信息提取，并经 Logistic 混沌解密得到文件格式及 Base64 字符串信息，最后进行 Base64 解码写出指定格式的文件；单击“打开文件目录”按钮即可查看导出的文件结果。

为了验证处理流程的有效性，选择另外的图像作为载体图像，选择某 pdf 文件作为待隐藏加密文件，具体分别如图 5-17 和图 5-18 所示。

图 5-17　载体图像

图 5-18　待隐藏加密的文件

此实验对选中的载体图像和 pdf 文件进行处理，单击“加密隐藏”按钮将 pdf 文件进行 Base64 编码、Logistic 混沌加密、LSB 信息隐藏，最终单击“导出图像”按钮将含密图像导出保存，具体效果如图 5-19 所示。经信息隐藏后的含密图与原图相比在视觉上没有可视化的差异，计算二者 PSNR 值为 63.9583，表明此实验能得到良好的信息隐藏效果。

下一步，读取含密图像并进行提取解密，具体效果如图 5-20 所示。在保持 Logistic 混沌加密参数一致的情况下，对含密图像进行信息提取、混沌解密及 Base64 解码写出，单击“打开文件目录”可查看提取解密的结果文件 demo_tiqu.pdf。

图 5-19　信息加密隐藏实验图

原始 pdf 文件与加密效果分别如图 5-21 和图 5-22 所示，按照解析出的文件格式及 Base64 编码内容进行解码并写出文件，可以发现内容与原文件完全相同，能够产生 pdf 文件隐藏加密的效果。注意，这里的程序内部默认设置了统一的 Logistic 混沌加密参数，读者可以尝试修改参数查看解密效果，验证多格式文件加密隐藏的安全性，也可将其应用于其他格式的文件，做进一步的应用延伸。

图 5-20　信息提取解密实验图

图 5-21　待隐藏加密的文件

图 5-22 提取解密的文件

5.5 案例小结

随着信息技术的迅速发展，信息隐藏加密应用也越来越广泛，特别是多格式文件的隐藏加密能够起到文件的匿名存储和传递的效果，具有一定的研究意义和实用价值。本章对基础的 LSB 信息隐藏提取、Base64 编解码、Logistic 混沌加密三个步骤进行融合处理，统一了不同格式文件的入口，并通过加入特殊标记的方式来实现自动 LSB 提取的功能。读者可以使用其他的方法对实验过程进行个性化修改，例如引入其他的信息隐藏方法或对基础的 LSB 算法进行改进，也可引入其他的可逆加密编解码方法等，进一步延伸应用。