浙江大学

课程名称：	多媒体安全
姓　　名：	庞懿非
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
学　　号：	3190104534
指导教师：	黄劲
完成时间：	2022 年 6 月 18 日

实验四：F3/F4 隐写术分析

一、实验目的

1. 了解数字图像隐写术中的典型算法：F3，F4 的基本原理。

2. 了解图像 DCT 系数的计算。

3. 了解数字水印与隐写术的异同。

二、实验内容与要求

1. 实现 F3、F4 的隐写系统，包括信息嵌入与信息检测。

2. 设计一份 3KB 左右的文本信息数据，使用上述两种隐写方法进行信息的嵌入。绘制原图与两种隐写方法嵌入得到的结果图的 DCT 系数直方图，并分析两种方法对原图 DCT 系数的影响。

3. 阐述隐写系统与数字水印系统的异同。

三、实验环境

1.**IDE**：MATLAB, Eclipse（Java）

2.**OS**：Windows 10

四、实验过程

0.背景与思路介绍

实验四与之前的三个实验有着巨大的差异，主要体现在数据的获取上以及隐写与水印系统本身的差异。在数据的获取上，从 JPEG 流行的时代到如今十几二十年的时间，都没有一个完整的库去直接解析 JPEG 文件的二进制流从而直接获得其半解压缩的 DCT 系数值（现在似乎有基于 C 的 jpeglib 库，但没有仔细研究）。所以，要想获得 DCT 系数，就必须要从像素值数据，和 JPEG 图片生成一样得到最终的 DCT 系数，这一点还是相当复杂的，也遇到了一些问题，在后续会展开描述。另一方面，隐写系统不再需要水印，但是由于系统的限

制相对较少，隐写传输的信息量又可以非常大，在此实验中我实现了 F3，F4 算法，并且对比了 DCT 系数的变化。

1.DCT 系数的获取

DCT 系数(指的是 JPEG 二进制码流中存储的 DCT 系数),需要从像素值经过分块(block), DCT 变换,量化,zigzag 形成以块为单位的系数矩阵,之后要恢复成图片,也要经过反 zigzag, 反量化，逆 DCT 变化生成原图。

下面我将按照操作顺序叙述操作与反操作：

① DCT 变换

$$F(u,v) = c(u)c(v)\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f(i,j)\cos\left[\frac{(i+0.5)\pi}{N}u\right]\cos\left[\frac{(j+0.5)\pi}{N}v\right]$$

$$\mathrm{c}(u) = \begin{cases}\sqrt{\frac{1}{N}}, & u=0\\ \sqrt{\frac{2}{N}}, & u\neq 0\end{cases}$$

DCT 变化作为矩阵的频率变换工具在图像处理中屡见不鲜，在将像素矩阵分块后，对每一个块都进行如上图所示的操作，这时候生成的数据为 8*8 的浮点数矩阵（每块）。当然，查询相关资料后，我们发现可以简化为如下方式实现：

$$F = AfA^T$$

$$A(i,j) = c(i)\cos\left[\frac{(j+0.5)\pi}{N}i\right]$$

同理，DCT 反变换的公式如下所示：

$$f(i,j) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} c(u)c(v)F(u,v)\cos\left[\frac{(i+0.5)\pi}{N}u\right]\cos\left[\frac{(j+0.5)\pi}{N}v\right]$$

$$\mathrm{c}(u) = \begin{cases}\sqrt{\frac{1}{N}}, & u=0\\ \sqrt{\frac{2}{N}}, & u\neq 0\end{cases}$$

也可以使用之前的替换公式，两种操作没有本质的区别，在处理性能上略有差异，本实验没有细致研究：

$$F = AfA^T$$
$$\because\ A^{-1} = A^T$$
$$\therefore\ f = A^{-1}F(A^T)^{-1} = A^TFA$$

DCT 反变换的结果应该是像素值，所以我对反变换的结果进行了四舍五入的存储。

两个方法都在 *JPEG_DCT_Zigzag* 类中，基本结构如下：

```
public static void getDCT(int blocksize,int block[][],double [][]dct_ori) {

    //construct A
    double[][] A1 = new double[blocksize][blocksize];//A1代表变换矩阵
    //construct AT
    double[][] A2 = new double[blocksize][blocksize];
    for (int i=0;i<blocksize;i++)
    {
        for (int j=0;j<blocksize;j++)
            A2[j][i]=A1[i][j];
    }

    // multiply AX
    double[][] AX = new double[blocksize][blocksize];
    for (int i = 0; i < blocksize; i++) {
        for (int j = 0; j < blocksize; j++) {
            for (int k = 0; k < blocksize; k++) {
                AX[i][j] += A1[i][k] * block[k][j];
            }
        }
    }

    // multiply AX AT
    for (int i = 0; i < blocksize; i++) {
        for (int j = 0; j < blocksize; j++) {
            for (int k = 0; k < blocksize; k++) {
                dct_ori[i][j] += AX[i][k] * A2[k][j];
            }
        }
    }
}
```

```
public static void apllyIDCT(int blocksize,  double block[][],int [][]pixels) {

    //the same to above

    for (int i = 0; i < blocksize; i++) {
        for (int j = 0; j < blocksize; j++) {
         AXA[i][j]=Math.round(AXA[i][j]);
         pixels[i][j]=(int)Math.round((AXA[i][j]));
        }
    }

}
```

② 量化

DCT 系数的量化主要是为了降低高频信号，其实通过调整量化表就可以实现相应的效果，在实验中我使用最基础的量化表，如下所示：

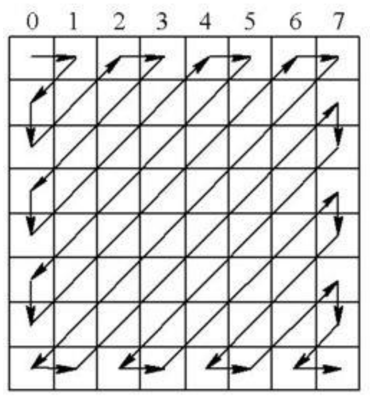
```
static int []quant_table = {
    16,  11,  10,  16,  24,  40,  51,  61,
    12,  12,  14,  19,  26,  58,  60,  55,
    14,  13,  16,  24,  40,  57,  69,  56,
    14,  17,  22,  29,  51,  87,  80,  62,
    18,  22,  37,  56,  68, 109, 103,  77,
    24,  35,  55,  64,  81, 104, 113,  92,
    49,  64,  78,  87, 103, 121, 120, 101,
    72,  92,  95,  98, 112, 100, 103,  99
};
```

量化和反量化其实本质上就是乘除法，我没有定义函数完成这一过程，而是写循环来完成，如下是量化过程：

```
//Quantify
for(int block_num = 0; block_num < blocknumbers; block_num++) {
    for(int i = 0; i < 8; i++) {
        for(int j = 0; j < 8; j++) {
            blocks_dct[block_num][i][j] /= quant_table[i*8+j];
        }
    }
}
```

③ Zigzag

Zigzag 看起来有些像在画一笔画，如下图：



实现 zigzag 的算法并不难，并且对于固定的 8*8 块来说没有必要，我们可以直接使用 zigzag 表来实现这一步骤。

```
static int []zigzag_table = {
    0,    1,    5,    6,   14,   15,   27,   28,
    2,    4,    7,   13,   16,   26,   29,   42,
    3,    8,   12,   17,   25,   30,   41,   43,
    9,   11,   18,   24,   31,   40,   44,   53,
   10,   19,   23,   32,   39,   45,   52,   54,
   20,   22,   33,   38,   46,   51,   55,   60,
   21,   34,   37,   47,   50,   56,   59,   61,
   35,   36,   48,   49,   57,   58,   62,   63
};
```

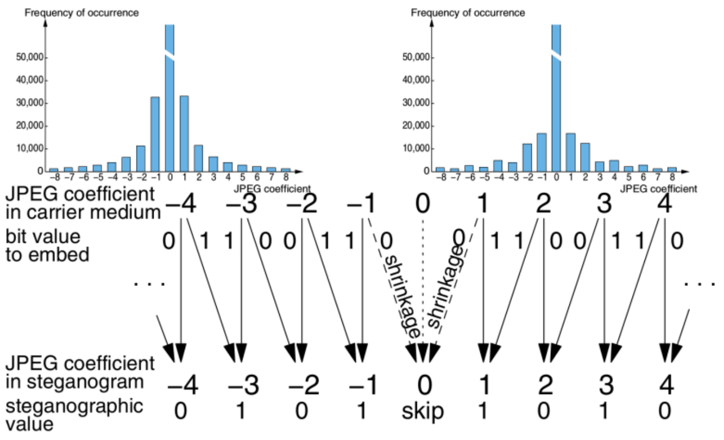
量化，反量化都可以通过这个数组实现，如下所示（图中的函数第一个参数为块的 8*8DCT 系数，第二个参数为 zigzag 后的一维数组序列）：

```
public static void zigzag(double [][]blocks_dct,double []sequence) {
    for(int i = 0; i < 64; i++) {
        int number = zigzag_table[i];
        sequence[number] = blocks_dct[i/8][i%8];
    }
}

public static void unzigzag(double [][]blocks_dct,double []sequence) {
    for(int i = 0; i < 64; i++) {
        int number = zigzag_table[i];
        blocks_dct[i/8][i%8] = sequence[number];
    }
}
```

最后，我们只需要将浮点数转为整数，就得到了真正存在 JPEG 图片中的 DCT 系数。

2. F3 系统实现

F3 隐写系统针对 Jsteg（类似 LSB 算法）系统进行了修改，有效避免了一对只有最低有效位不同的系数在隐写后接近的现象，但是同时也引入了其他问题。其算法示意图如下所示（很直观所以不作过多解释）：



在实现的过程中，我们要对 DCT 系数不断遍历，在能加入信息时写入信息，并且保证 shrinkage 的正常进行。

```
public static boolean writebit(int num[],boolean bit,int index){
    //write bit if possible
    //return true if write success
    //return false if not write or shrink
}
```

编码端的思路如上所示，在解码端的则更加简单。

```
public static String readbit(int num[],int index) {
    int number = num[index];
    if(number == 0) {
        return null;
    }
    else if(Math.abs(number) % 2 == 1) {
        return "1";
    }else if(Math.abs(number) % 2 == 0) {
        return "0";
    }
    return null;
}
```

在 F3 调用端，我的操作方法如下：

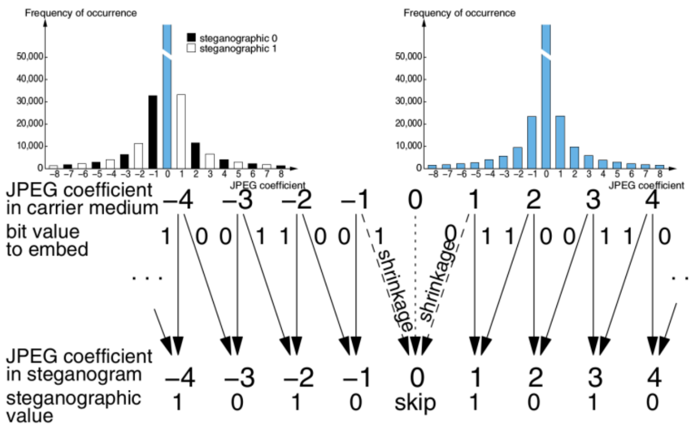
```
for(int i = 0; i < bits_length; ) {
        boolean bit = bits_stream.charAt(i) == '1';
        boolean ret = writebit(int_zg_dct[block_id],bit,index);

        if(ret) {
            i++;
        }
        else {
            //do nothing
        }
        index++;
        if(index == 64) {
            block_id++;
            print(int_zg_dct[block_id]);
            index = 0;
            System.out.println("block " + block_id + " is used!");
        }
        if(block_id >= blocknums) {
            System.out.println("capacity not enough!");
        }
    }
```

bits_stream 为输入文本的二进制码流，获取该字符对当前 block（*block_id*）中的 ***index*** 位置进行判断，如果写入成功就读取下一个字符。每个块的 64 个字符读取结束之后就进入下一个块，按照顺序方向进行（从左到右，从上到下）。解码端也是同理。

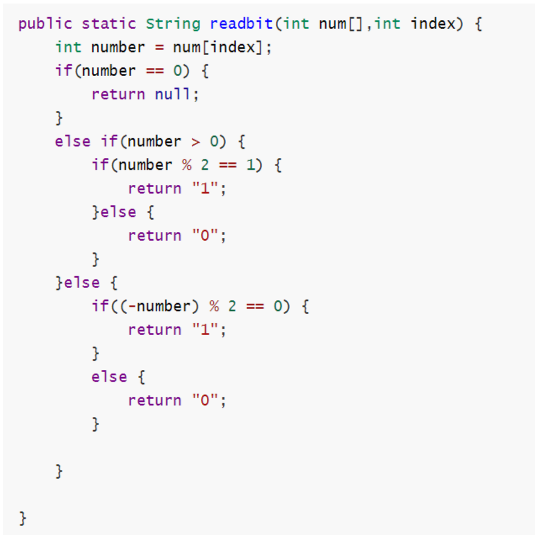
3. F4 系统实现

F4 是在 F3 的基础上改进的，F3 系统由于对 0 的 shrink 导致嵌入 0 的概率大大增加，很多时候造成 P(2i)>P(2i-1)，F4 系统在正负数上采用了非对称的嵌入方法，使得 0 和 1 因为 shrink 增加的概率几乎相同，从而避免了上述问题。但同样，F4 系统自身仍旧存在很多其他的问题。

F4 算法的原理图如下所示：



在编解码上，无论是 F3 还是 F4，都是按照算法示意图进行信息嵌入，并无太多差别，我仍然以较为简单的解码端代码进行展示。



```
public static String readbit(int num[],int index) {
    int number = num[index];
    if(number == 0) {
        return null;
    }
    else if(number > 0) {
        if(number % 2 == 1) {
            return "1";
        }else {
            return "0";
        }
    }else {
        if((-number) % 2 == 0) {
            return "1";
        }
        else {
            return "0";
        }

    }

}
```

F4 系统与 F3 系统除了编解码端的微小差异之外，几乎可以完全重用。

4. 输入文本处理

因为 F3 和 F4 系统都是基于二进制的，所以需要提前将输入文本进行处理。

为了实现的简便，我统一规定使用全英文本来避免不同字符编码集之间可能的问题。所有相关处理都在 ***bitsstream*** 类中完成。

将文本中的字符串读取到 String 对象中，调用 `*getBytes*`方法生成 byte 数组，之后调用如下方法对每个 byte 处理生成二进制流。

```java
public static String bytesToString(byte[] bytes) {
    StringBuilder result_string = new StringBuilder();
    for(int i = 0; i < bytes.length; i++) {
        String tmp = getBinaryStrFromByte(bytes[i]);
        result_string.append(tmp);
    }
    result_string.append("11111111");
    return new String(result_string);
}
```

在最后加入 8 个 1 作为标识符，即 EOF。

```java
public static String getBinaryStrFromByte(byte b){
    String result ="";
    byte a = b; ;
    for (int i = 0; i < 8; i++){
        byte c=a;
        a=(byte)(a>>1);
        a=(byte)(a<<1);
        if(a==c){
            result="0"+result;
        }else{
            result="1"+result;
        }
        a=(byte)(a>>1);
    }
    return result;
}
```

5.性能分析和系统调整（数据分析）

在使用了一些 jpg 图像进行隐写之后，我发现很多块的 DCT 系数高频区域几乎都为 0，很多块甚至只有直流系数，这其实对隐写术来说并不是有利的条件，一方面的 capcity 的损失，另一方面则是变化的高频噪声很容易被发现。

除此以外，一些在块的交流系数在变成像素的过程中，在最后的量化过程（比如 double 转为 int 像素，像素值超出范围后 snapping）会出现一些变化，这导致对于一些块，在经过这些量化之后无法恢复成之间的 DCT 系数，在实验中我曾多次遇到这个问题，有些时候这些变化会导致 EOF 提前出现，有些时候则会出现些许乱码，在下面的实验分析中我会进行样例分析和系统调整说明。

此外，我仍然测试了不同长度文本数据对 DCT 系数的影响程度，因为 3KB 的数据量比起整个图片的块的数量还过小。一张 2448*3264 的 jpg 图像有将近 800 万个块，传输 2.4 万个 bit 的数据还是相对过小了，尽管很多块只有 DC 系数可以传输信息。

五、实验分析与结论

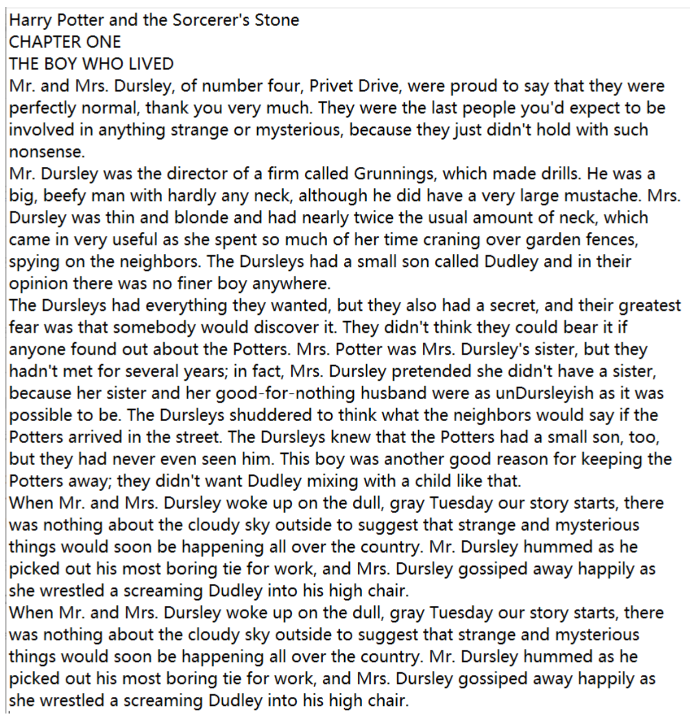
由于 DCT 变换的特性，一些小的取整变化以及限定阈值可能对高频信号带来相对较大的改变，而任意一位的改变都可能导致整个码流的分析错误，在实验中我本发现了一个具体的例子，但是后续验证发现这个块的系数仍然保持稳定，但是仍然有一些例外的情况会导致错误。

错误的发现是通过对编码后的 DCT 系数存储进行解密以及使用编码后的 DCT 系数重新逆变换回图片像素再读取像素变换至 DCT 系数进行解密。其实可以将二者的 DCT 系数读出来进行对比，必然有一些地方出现了细小的差异。在数字水印系统中，我们称为嵌入失效；不过在隐写术中，我们可以更换符合我们要求的图片。

```
add_F3("test.txt","more_data_48.jpg");
decode_F3("test1.jpg");
decode_F3_FromFile("code.txt");
```

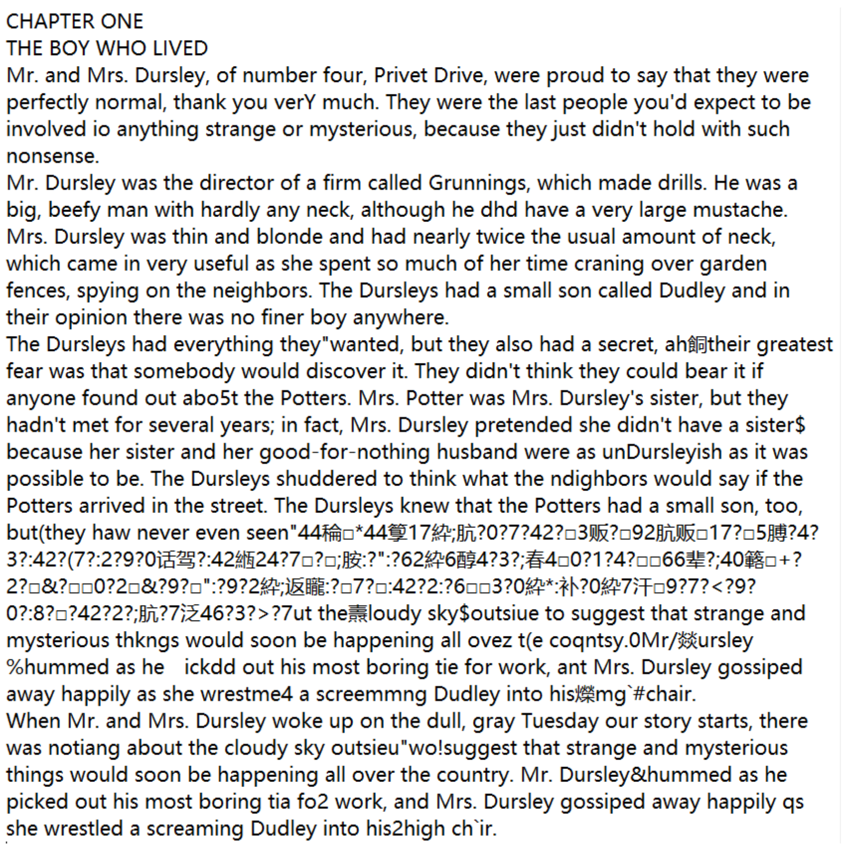
调用第一个函数进行 F3 隐写，并且生成了图片，第二个函数是通过从生成的图片中截取信息，第三个函数是在 F3 隐写生成的 DCT 系数上（存储到了文件 code.txt 中）进行解密分析。

我们以如下输入文本为例（存储在 test1.txt 中）：

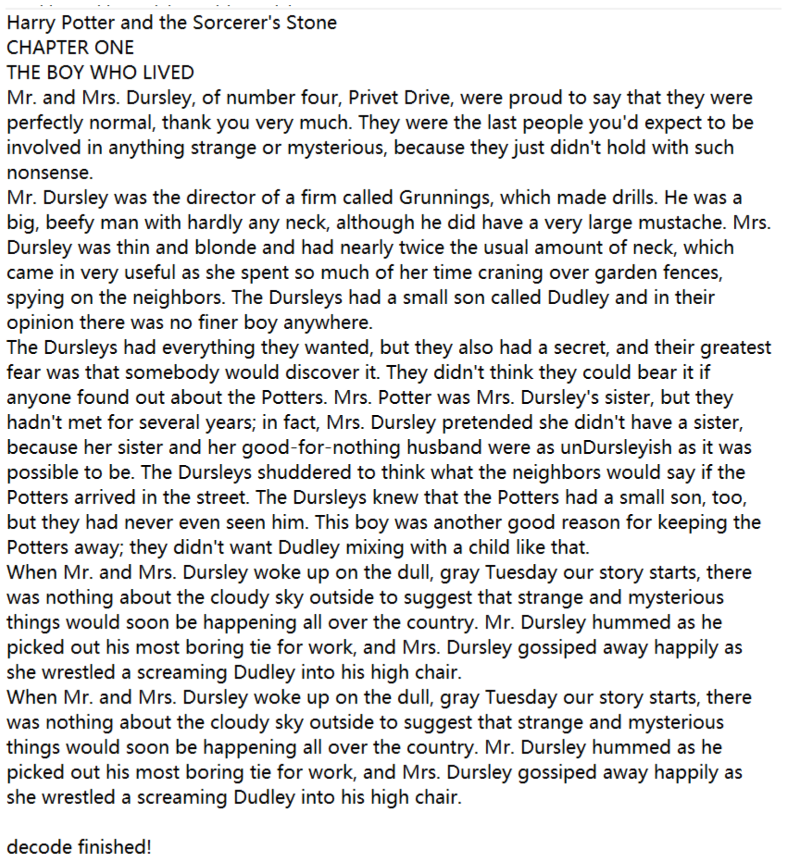


Harry Potter and the Sorcerer's Stone
CHAPTER ONE
THE BOY WHO LIVED
Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense.
Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large mustache. Mrs. Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences, spying on the neighbors. The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere.
The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They didn't think they could bear it if anyone found out about the Potters. Mrs. Potter was Mrs. Dursley's sister, but they hadn't met for several years; in fact, Mrs. Dursley pretended she didn't have a sister, because her sister and her good-for-nothing husband were as unDursleyish as it was possible to be. The Dursleys shuddered to think what the neighbors would say if the Potters arrived in the street. The Dursleys knew that the Potters had a small son, too, but they had never even seen him. This boy was another good reason for keeping the Potters away; they didn't want Dudley mixing with a child like that.
When Mr. and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there was nothing about the cloudy sky outside to suggest that strange and mysterious things would soon be happening all over the country. Mr. Dursley hummed as he picked out his most boring tie for work, and Mrs. Dursley gossiped away happily as she wrestled a screaming Dudley into his high chair.
When Mr. and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there was nothing about the cloudy sky outside to suggest that strange and mysterious things would soon be happening all over the country. Mr. Dursley hummed as he picked out his most boring tie for work, and Mrs. Dursley gossiped away happily as she wrestled a screaming Dudley into his high chair.

该文本大小为 2299B，使用上述两种解码方式解码的结果分别展示如下：



```
CHAPTER ONE
THE BOY WHO LIVED
Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were
perfectly normal, thank you verY much. They were the last people you'd expect to be
involved io anything strange or mysterious, because they just didn't hold with such
nonsense.
Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a
big, beefy man with hardly any neck, although he dhd have a very large mustache.
Mrs. Dursley was thin and blonde and had nearly twice the usual amount of neck,
which came in very useful as she spent so much of her time craning over garden
fences, spying on the neighbors. The Dursleys had a small son called Dudley and in
their opinion there was no finer boy anywhere.
The Dursleys had everything they"wanted, but they also had a secret, ah銅their greatest
fear was that somebody would discover it. They didn't think they could bear it if
anyone found out abo5t the Potters. Mrs. Potter was Mrs. Dursley's sister, but they
hadn't met for several years; in fact, Mrs. Dursley pretended she didn't have a sister$
because her sister and her good-for-nothing husband were as unDursleyish as it was
possible to be. The Dursleys shuddered to think what the ndighbors would say if the
Potters arrived in the street. The Dursleys knew that the Potters had a small son, too,
but(they haw never even seen"44稐□*44蕶17綷;肮?0?7?42?□3贩?□92肮贩□17?□5膊?4?
3?:42?(7?:2?9?0话驾?:42緬24?7□?□;胺:?":?62綷6醇4?3?;春4□0?1?4?□□66辈?;40籗□+?
2?□&?□□0?2□&?9?□":?9?2綷;返朧:?□7?□:42?2:?6□□3?0綷*:补?0綷7汗□9?7?<?9?
0?:8?□?42?2?;肮?7泛46?3?>?7ut the藁loudy sky$outsiue to suggest that strange and
mysterious thkngs would soon be happening all ovez t(e coqntsy.0Mr/燚ursley
%hummed as he   ickdd out his most boring tie for work, ant Mrs. Dursley gossiped
away happily as she wrestme4 a screemmng Dudley into his爣mg`#chair.
When Mr. and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there
was notiang about the cloudy sky outsieu"wo!suggest that strange and mysterious
things would soon be happening all over the country. Mr. Dursley&hummed as he
picked out his most boring tia fo2 work, and Mrs. Dursley gossiped away happily qs
she wrestled a screaming Dudley into his2high ch`ir.
```

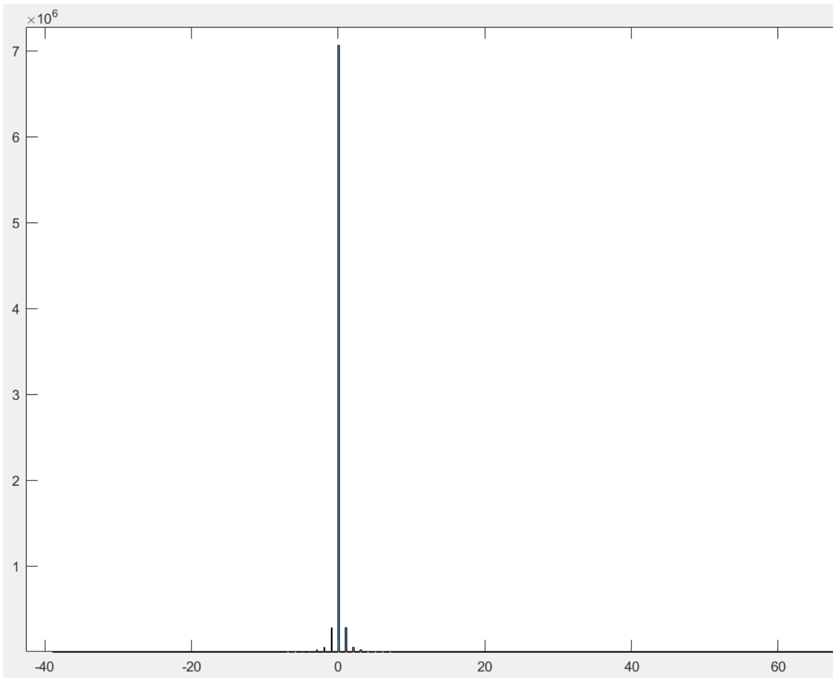


```
Harry Potter and the Sorcerer's Stone
CHAPTER ONE
THE BOY WHO LIVED
Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were
perfectly normal, thank you very much. They were the last people you'd expect to be
involved in anything strange or mysterious, because they just didn't hold with such
nonsense.
Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a
big, beefy man with hardly any neck, although he did have a very large mustache. Mrs.
Dursley was thin and blonde and had nearly twice the usual amount of neck, which
came in very useful as she spent so much of her time craning over garden fences,
spying on the neighbors. The Dursleys had a small son called Dudley and in their
opinion there was no finer boy anywhere.
The Dursleys had everything they wanted, but they also had a secret, and their greatest
fear was that somebody would discover it. They didn't think they could bear it if
anyone found out about the Potters. Mrs. Potter was Mrs. Dursley's sister, but they
hadn't met for several years; in fact, Mrs. Dursley pretended she didn't have a sister,
because her sister and her good-for-nothing husband were as unDursleyish as it was
possible to be. The Dursleys shuddered to think what the neighbors would say if the
Potters arrived in the street. The Dursleys knew that the Potters had a small son, too,
but they had never even seen him. This boy was another good reason for keeping the
Potters away; they didn't want Dudley mixing with a child like that.
When Mr. and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there
was nothing about the cloudy sky outside to suggest that strange and mysterious
things would soon be happening all over the country. Mr. Dursley hummed as he
picked out his most boring tie for work, and Mrs. Dursley gossiped away happily as
she wrestled a screaming Dudley into his high chair.
When Mr. and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there
was nothing about the cloudy sky outside to suggest that strange and mysterious
things would soon be happening all over the country. Mr. Dursley hummed as he
picked out his most boring tie for work, and Mrs. Dursley gossiped away happily as
she wrestled a screaming Dudley into his high chair.

decode finished!
```

在文本量较少的时候，使用两种方法完全没有区别，但是当文本数量增大，一些小的错误会导致码流串位从而无法得到精确的数据，这些错误很可能是由于数值在放入图像的时候

带来了巨大的抖动而引起的，一个位的变化或者丢失对于整个信息的破译来说都是致命性的。所以在后续我主要以从文本中直接读取 DCT 系数为例进行演示，尤其是在后续使用大量文本的过程中，很难找到一个不出错的图片，使用文本的方式进行形象的演示是可行的。**当然，在真正隐写时，我们要找到可以完整传输信息而不出错的图片才算隐写成功**。

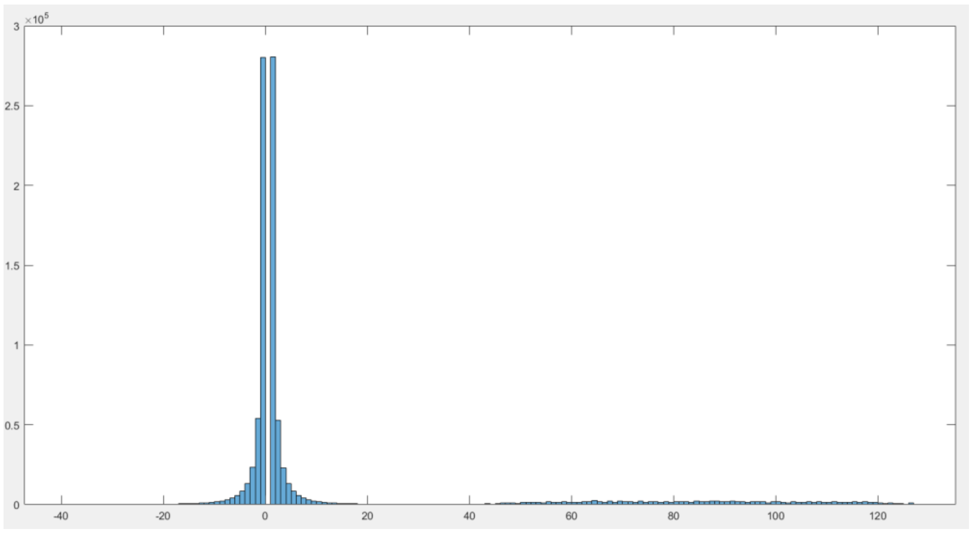
如上两张图，左为原图，右为隐写图，二者从视觉上无差异，那么在统计性质（直方图）上效果将在下面直方图展现。（直方图系数直接保存在文件 ori.txt 和 new.txt 中）



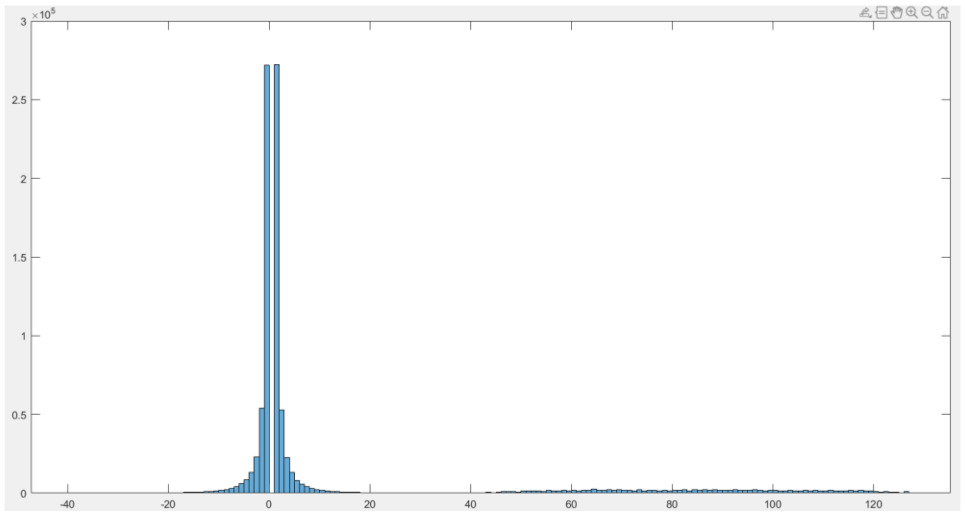
如上是原图（测试全部使用 more_data_48.jpg）由于 DCT 系数中 0 的数量过多，我们直接将 0 去掉查看结果（代码如下）。

```
data=load("ori.txt");
data(data==0) = [];
data=int8(data);
histogram(data)
```

原图示意图效果如下：



隐写之后效果如下：

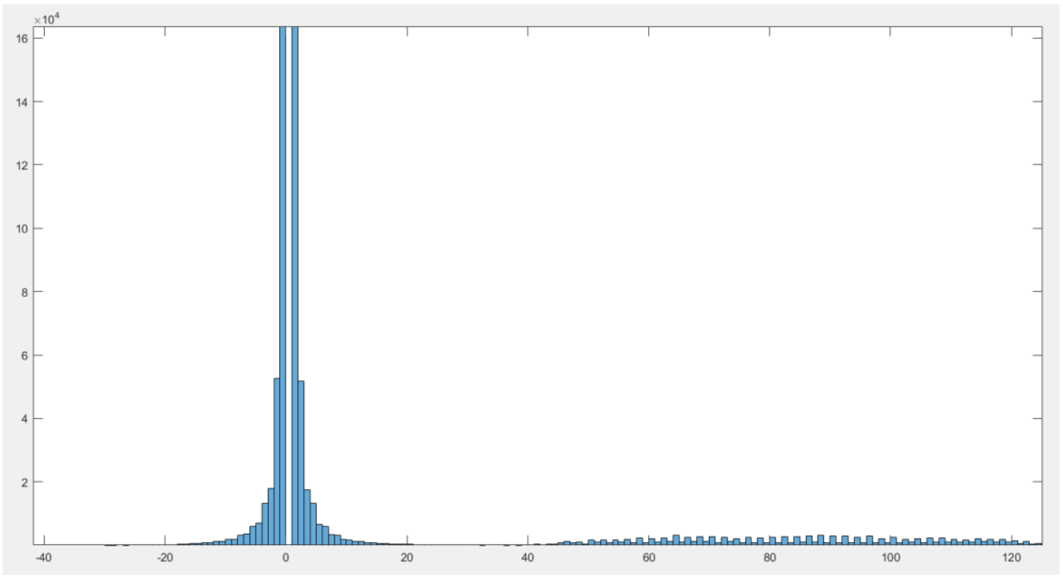


直观上并没有太大变化，所以我将每个值的统计数绘制表格如下展示具体差异（以正数系数为例，负数同理）：

	1	2	3	4	5	6
原图中个数	280633	52986	22916	13052	8257	5726
隐写后个数	272316	52916	22544	13049	8157	5728

可以看到，奇数如 3,5 明显比相邻偶数如 4,6 减少了很多，当隐写的字符增多之后，这个性质将更加明显。

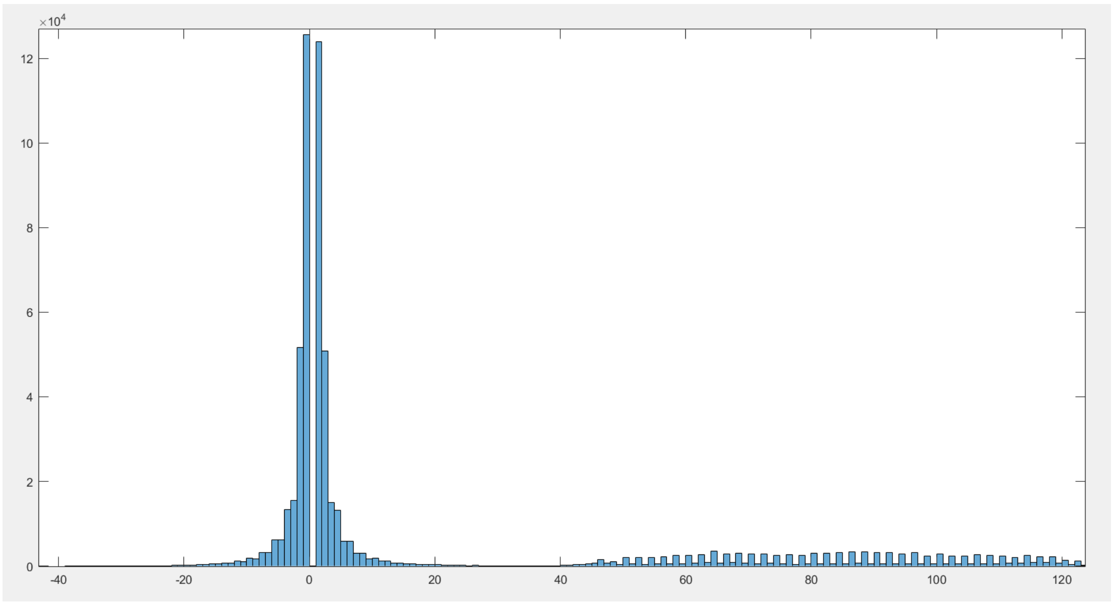
之后，我们使用安全的版本隐写一个 33KB 的英文文本（test2.txt），解码端可以得到正确解码的文件。（文本均选自哈利波特）



在这个图中，F3 系统的缺陷更加明显了一些，我们统计其数据如下：

	1	2	3	4	5	6
原图中个数	280633	52986	22916	13052	8257	5726
隐写后个数	169410	51800	17412	13183	6539	5879

为了出现课堂出出现的 P(2i)>P(2i-1)现象，我们使用更大的 48KB 文本（存储在 test3.txt）进行隐写，得到如下结果：

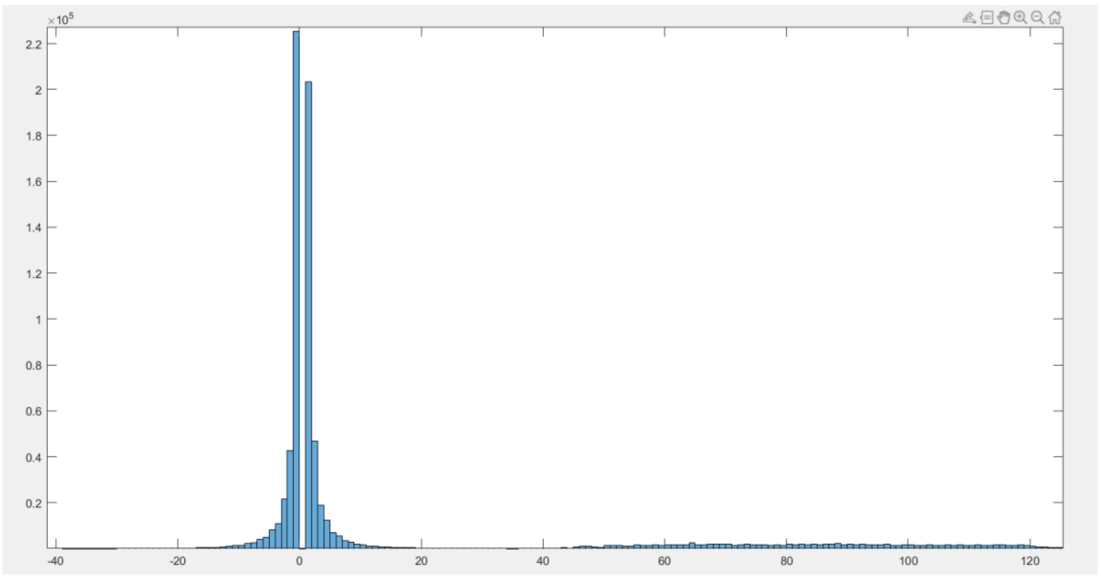


可以看到，右边一些点已经出现了 P(2i)>P(2i-1)的现象，统计数据如下：

	1	2	3	4	5	6
原图中个数	280633	52986	22916	13052	8257	5726
隐写后个数	123873	50849	15022	13205	5886	5898

在这里，这张图片也几乎接近了隐写容量的上限了，我之后也没有继续测试更大的文本数据。从这些数据可以很清晰的看到 F3 系统的缺点，不过在一般情况下传输少量的文字是没有任何问题的。如上所有的测试也都可以复现，但是请不要试图从生成的 jpg 图片去解密，如果想直接从图片解密信息，可以尝试多张图片和少量文本进行不断尝试和适配。

同样，对于 F4 系统而言，与 F3 系统的差异主要来源于不具有 P(2i)>P(2i-1)的现象，于是我们直接使用 48KB 的文本进行压力测试区观察图片的 DCT 系数变化。



	1	2	3	4	5	6
原图中个数	280633	52986	22916	13052	8257	5726
F3 隐写后	123873	50849	15022	13205	5886	5898
F4 隐写后	203270	46858	18859	12233	6982	5488

尽管 F4 算法有效避免了 F3 算出可能出现的相邻奇数偶数大小关系颠倒的情况，但是还是会导致整个非 0 系数的下降，由于 DCT 系数中的 AC 分量都很小，很多值都只是+-1，这导致大量的值在隐写中被 shrink，整个分布曲线向中心收缩，其实对于检测端来说还是很容易观察的，尤其是按照图片块顺序逐次检测，会发现更加明显的规律。即使从肉眼上两张图片几乎无差异（左原图，右 F4 隐写）。

无论是F3算法还是F4算法，只需要使用简单的DCT系数直方图检测就可以破解，也可以使用与GCD模型的差异或者比较相似的块做差来获得差异。

思考题

阐述隐写系统与数字水印系统的异同。

相同点：

都可以隐藏信息在图片中，可以进行通信等用途。

二者都会关注鲁棒性，关注传输的抗干扰性。

二者都会保证信息的唯一性和正确性。

在隐藏存在性的目的下，二者都不会轻易让人产生察觉。

二者都可以嵌入在视频、音频、图像等媒体中。

不同点：

关注点不全相同，水印系统会对effectiveness、fidelity等进行关注，隐写系统则更加关注统计不可检测性，且一般的隐写系统会有机器来检测。

隐写是不可知道存在性的，而水印的一类——公开水印可以被知道水印存在。

隐写系统和水印系统的主要功能不同；水印系统主要用于认证、防伪、防复刻、设备标记等，而隐写系统多用于间谍活动或者危害性活动。

水印系统的载体co通常不可更换，但是隐写系统基于的载体信息可以改变。

在图像隐写系统中，通常为了不引起怀疑只能使用jpg文件，但是水印系统不受此限制。

六、实验感想

这个实验在获取 DCT 系数这里困惑了我一段时间，因为我一直觉得可以直接从 jpeg 文件中读取 DCT 系数，就如同从图像中读取像素一样简单，但事实并非如此；同时还伴随图像处理的同时带来的微小误差导致一直无法正确解码，或者说解码总出现各种小错误。我将原图、隐写后的 DCT 以及新图的 DCT 系数全部拿出来比对，发现了一些系数的微小变化带来的不良后果，并且即使使用文本的方式重新恢复错误。最终得到了一些直方图的展现，总体来说领悟到了 F3，F4 系统的一些特性。

当然隐写系统实现需要我们选取一张好的图片，能够胜任隐写的各个环节，尤其是在经过变换取整 snap 等操作之后逆变换仍然不出现错误。这也需要大量的筛选，在这一步上，我确实需要改进，因为在实验中找到了替代方案，所以就使用从文本中直接读取隐写后的 DCT 值进行了解密，虽然流程上不太符合，但是原理上仍然是正确的。

更多的信息可以参考报告内容。