信息隐藏实验报告

实验(三)：STDM 水印嵌入和解码

姓　　名：_____

学　　号：_____

同组成员：_____

专　　业：_____

二〇二二年十二月

第一部分　实验分工与文件结构

1. 实验分工与完成情况

实验分工如下表所示。

（表 1 实验分工与完成情况）

任务点	内容	完成情况	主要贡献者	贡献率
1	在 Lena 图中嵌入一个 64×64(共 4096 位）的 Logo，需要使用 STDM 方法嵌入在 DCT 系数中	√		50%
2	性能比较（不同量化步长）：绘制不同量化步长下，相同噪声攻击强度的性能变化曲线	√		
3	性能比较（不同噪声强度）：绘制在相同的量化步长下，不同强度的噪声攻击下系统性能的变化的曲线	√		50%
4	性能比较：绘制在相同的量化步长下，不同强度的 JPEG 压缩下系统性能的变化的曲线	√		

2. 作业提交的文件结构

```
---2053182-王润霖-实验(三)报告.pdf
---源代码
    ---bitmap_image.cpp
    ---bitmap_image.h
    ---DCT.cpp
    ---DCT.h
    ---LENA.BMP
    ---lena_embed.bmp
    ---LOGO.bmp
    ---logo_decode.bmp
    ---main_test.cpp
    ---STDM_watermark.cpp
    ---STDM_watermark.h
---实验数据和图片
    ---第 1 问数据样例（不同强度噪声图像）
    ---第 2 问数据样例（不同量化步长图像）
    ---第 3 问数据样例（jpeg 压缩图像）
    ---实验数据图表
```

第二部分　相关内容介绍（背景知识）

2.1 DCT 变换

DCT 变换的全称是离散余弦变换(Discrete Cosine Transform)，主要用于将数据或图像的压缩，能够将空域的信号转换到频域上，具有良好的去相关性的性能。DCT 变换本身是无损的，但是在图像编码等领域给接下来的量化、哈弗曼编码等创造了很好的条件，同时，由于 DCT 变换是对称的，所以，我们可以在量化编码后利用 DCT 反变换，在接收端恢复原始的图像信息。DCT 变换在当前的图像分析已经压缩领域有着极为广大的用途，我们常见的 JPEG 静态图像编码以及 MJPEG、MPEG 动态编码等标准中都使用了 DCT 变换。

2.2 STDM 方法

Chen 注意到扩频和量化技术各有千秋，所以他利用扩展变换(Spread Transform (ST)) 技术设计了一种基于扩展变换的抖动调制 (Spread Transform Dither Modulation (STDM))方案，该方案吸收了扩频和量化技术各自的优点。

2.3 JPEG 压缩原理

JPEG 压缩是基于 YUV 颜色空间进行压缩编码的，首先将 RGB 转化成 YUV，然后将像素值减去 128，转化到范围-128~127。后要进行采样，一般来说有 3 中采样方式：4:4:4，4:2:2 和 4:1:1；4:4:4 即不进行下采样，4:1:1 是指一个 2x2 的单元，采样 4 个 Y，1 个 V 和 1 个 U（具体想了解采样的可以看下其它博客）。后直接进行 8x8 的 DCT 变换，将时域像素值转化到频域。后根据参数 quality（质量因子或压缩因子）进行量化。量化之后按 zigzag 扫描之后进行编码。

JPEG 压缩有两张固定的 8x8 的量化表，压缩时根据指定的质量因子计算其具体的量化系数进行量化，质量因子范围 1~100。当质量因子是 100 时，量化因子都是 1，质量因子是 1 是最大程度的压缩。libjpeg 中默认量化表分别是亮度量化表、色度量化表。

由于 LENA.bmp 是黑白图像，我们只需要使用默认亮度量化表，即可通过对 DCT 矩阵进行相应计算，来模拟 JPEG 压缩攻击过程。

第三部分　实验目的、内容和原理

3.1 实验目的

通过实验，学习并掌握 STDM（基于扩展变换的抖动调制）嵌入技术在图像的信息隐藏中的应用；熟悉线性相关器解码的操作；探究不同强度的噪声攻击，不同量化步长下相同噪声攻击，以及在相同的量化步长下不同强度的 JPEG 压缩的性能变化曲线。

3.2 实验内容

在 Lena 图中嵌入一个 64×64(共 4096 位) 的 Logo，需要使用 STDM 方法嵌入在 DCT 系数中。

（1）性能比较（不同量化步长）：绘制不同量化步长下，相同噪声攻击强度的性能变化曲线；

（2）性能比较（不同噪声强度）：绘制在相同的量化步长下，不同强度的噪声攻击下系统性能的变化的曲线；

（3）性能比较：绘制在相同的量化步长下，不同强度的 JPEG 压缩下系统性能的变化的曲线。

其中，使用 matlab 编写程序添加高斯噪声；使用对 bmp 图像的 dct 矩阵按照 jpeg 压缩原理进行计算的结果，模拟 jpeg 压缩攻击。

3.3 实验原理

3.3.1 BMP 图像文件存储格式

3.3.1.1 BMP 格式介绍

BMP 是 Bitmap（位图）的简称，是 Windows 操作系统中的标准图像文件格式。其特点是由于几乎不进行压缩，所以包含的图像信息较丰富，但同时也到之占用的磁盘空间较大。

3.3.1.2 BMP 文件格式

位图文件由 4 个部分组成：

（1）位图头文件（bitmap-file header)

（2）位图信息头（bitmap-information header)

（3）颜色表（color table)：使用索引来表示图像，此时颜色表就是索引与其对应的颜色之间的映射表（即通过索引值，结合颜色表，找到对应的像素信息）

（4）位图数据（data bits)

3.3.1.3 位图头文件与位图信息头

BMP 文件中，数据的存储方式为小端方式（little endian)，即假设一个数据需要多个字节来表示，那么数据的存放字节的顺序为“低地址存放地位数据，高地址存放高位数据”。在十六进制中，一个数字占 4 位，因此每个字节可以存储 2 个十六进制数字。

位图头文件的结构体可以通过以下代码理解：

```
typedef struct tagBITMAPFILEHEADER
{
    UINT16 bfType;    // 19778，必须是 BM 字符串，对应的十六进制为 0x4d42,十进制为 19778，否则不是 bmp 格式文件
```

```
    DWORD bfSize; // 文件大小 以字节为单位(2-5 字节)
    UINT16 bfReserved1; // 保留，必须设置为 0 (6-7 字节)
    UINT16 bfReserved2; // 保留，必须设置为 0 (8-9 字节)
    DWORD bfOffBits;    // 从文件头到像素数据的偏移  (10-13 字节)
} BITMAPFILEHEADER;
```

同样地，可以定义位图信息头的结构体，在后文的代码中体现。

3.3.2 DCT 原理

3.3.2.1 DCT 变换简介

傅里叶变换表明，任何信号都能表示为多个不同振幅和频率的正弦或者余弦信号的叠加。如果采用的是余弦函数，则信号分解过程称为余弦变换；若输入信号是离散的，则称之为离散余弦变换（Discrete Cosine Transform，DCT）。

3.3.2.2 DCT 性质

离散余弦变换具有两点性质，第一个为可分离性，第二个为能量集中性。

DCT 的可分离性就是可以将二维 DCT 拆分为两个方向的一维 DCT。

图像编码中，图像以矩阵形式存储，是一个二维数组，对图像进行二维 DCT，可以将时域图像转为频域能量分布图，实现**能量集中**从而去除空间冗余。

图像中，低频分量可以看作是基本图像，高频分量为边缘轮廓细节信息（也可能是噪声）。绝大多数能量都包含了大量平坦区域，因此大部分能量集中于低频区域（左上角），从而可以达到去除空间冗余的目的。（其中，DCT 变换尺寸越大，DCT 去相关性能越好，但是 DCT 的计算复杂度会随之增大）。

3.3.3 STDM 水印嵌入与解码

3.3.3.1 STDM 水印嵌入原理

我们定义 x 在 w 上的投影如式(1)所示：

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i w_i}{N} \tag{1}$$

我们同样可以将 $\bar{X}、\bar{s}、\bar{S}、\bar{y}、\bar{Y}、\bar{v}、\bar{V}$ 分别定义为 X、s、S、y、Y、v 和 V 在 w 上的投影。

由此，得到 STDM 的嵌入规则，如式(2)所示：

$$s_i = x_i + \left[q_b(\bar{x}) - \bar{x}\right] \cdot w_i, (i = 1,2,\ldots,N) \tag{2}$$

3.3.3.2 STDM 水印解码原理

假设 V 代表攻击，为了方便起见，我们同时假设 V 独立于 X，那么受到攻击后的作品如式(3)所示：

$$Y_i = S_i + V_i = X_i + \left[q_b\left(\overline{X}\right) - \overline{X}\right] \cdot w_i + V_i \tag{3}$$

由于 STDM 本质上是一个量化水印技术，所以我们可以使用量化水印的解码方法。不同的是，由于信息是嵌入在 $\overline{x}$ 中的，因此我们需要使用 y 在 w 上的投影来解码。即：

$$If\left|q_0\left(\overline{Y}\right) - \overline{Y}\right| < \left|q_1\left(\overline{Y}\right) - \overline{Y}\right| \Rightarrow b' = 0; otherwise\left(b' = 1\right) \tag{4}$$

当然如果 $\left|d[0] - d[1]\right| = \Delta/2$，那么也可以采用如式(5)所示的解码法则：

$$If\left|q_0\left(\overline{Y}\right) - \overline{Y}\right| < \Delta/4 \Rightarrow b' = 0; otherwise\left(b' = 1\right) \tag{5}$$

第四部分　实验过程与参数说明

4.1 参数说明

在头文件中，已经详细介绍了每一个参数的含义和各个函数的功能，并在注释中书写了主要思想。

```
class STDM_watermark {
protected:
    double delta;                       //量化步长
    string name_image;                  //嵌入图像文件名
    string name_mark;                   //嵌入水印文件名
    bitmap_image image;                 //嵌入图像文件
    bitmap_image mark;                  //嵌入水印文件
    bitmap_image de_image;              //需要解码图像文件
    int image_width;                    //图像数据宽度
    int image_height;                   //图像数据高度
    int mark_width;                     //水印数据宽度
    int mark_height;                    //水印数据高度
    vector<double>image_data;           //原始图像数据
    vector<double>image_embed;          //嵌入扩频水印后图像数据
    vector<double>decode_image;         //需要解码图像信息
    vector<int>mark_image;              //原始嵌入水印数据信息
    vector<int>mark_decode;             //对嵌入图像解码后获得水印数据信息
    void get_dct_data(double in_data[8][8], double data[34]);                    //
获取 DCT 中频系数—需要嵌入水印的数据
    void preserver_dct_data(double in_data[34], double data[8][8]);              //
保存嵌入水印后数据到图像位图数据中
    void embed_8x8_image(double in_data[8][8], double out_data[8][8], int b);  //
对 8x8 图像块进行水印嵌入
    int decode_8x8_image(double data[8][8]);                                     //
对 8x8 图像块进行水印解码
    double count_error();
```

```
//统计水印解码错误率
    unsigned char* transform_bit_to_byte();
//将水印存储数据转为 bmp 存储格式
public:
    STDM_watermark(string n_image, string n_mark);
    void embed_watermasking();                    //水印嵌入
    void decode_watermasking();                   //水印解码
    void watermask_embed_and_decode();            //水印嵌入及解码并统计解码错误
率
};
```

```
class DCT {
protected:
    int modification(double a); //修正函数，在 DCT 反变换过程中使用,调整图像像素
值在指定范围内，求其最近整数
public:
    void dc_transform(double(*input)[NUM], double(*output)[NUM]);//DCT 正变换
    void Inverse_dc_transform(double(*input)[NUM], double(*output)[NUM]);//DCT
反变换
};
```

```
class bitmap_image
{
protected:
    string bitmap_image_name;                  //图像名字
    BITMAPINFOHEADER bitmap_image_message;     //图像数据信息
    int bitmap_image_width;                    //图像宽度
    int bitmap_image_height;                   //图像高度
    int bitmap_image_len_width;                //位图数据每行字节数
    short bitmap_image_type;                   //图像类型
    int bitmap_image_color_num;                //颜色位数
    RGBQUAD* pcolortable;                      //颜色表
    unsigned char* bitmap_image_data;          //位图数据
    void initial_data();                        //初始化图像参数信息
public:
    int width() const;              //返回图片的宽度
    int height() const;             //返回图片的高度,位图信息列数
    int len_width() const;          //返回位图信息列数
    unsigned char* image_data();   //返回位图数据
    ~bitmap_image();
    bool read_image(string image_name);          //读取图像数据信息
    bool preserve_image(string image_name, unsigned char* image_data); //存储图
像信息
};
```

4.2 实验过程

4.2.1 BMP 文件读取与写回

扩频水印的嵌入与解码主要涉及：

（1）对 BMP 存储的位图数据进行操作；

（2）保存水印嵌入成功图像；

（3）保存水印解码后水印图像。

因此，程序需要在文件信息头读取图像宽度、高度等信息，以及颜色表和位图数据信息。读取内容的存储，主要利用动态空间的申请实现。

由于函数较长，这里仅附上函数名及其功能的说明。

```
//读取图像数据信息
bool bitmap_image::read_image(string image_name)
```

```
//存储图像信息
bool bitmap_image::preserve_image(string image_name, unsigned char* image_data)
```

```
//返回图片的宽度
int bitmap_image::width() const //返回图片的宽度
//返回图片的高度，位图信息列数
int bitmap_image::height() const //返回图片的高度
//返回位图信息列数
int bitmap_image::len_width() const
//返回位图数据
unsigned char* bitmap_image::image_data()
```

4.2.2 DCT 正反变换

根据 DCT 正反变换的原理公式，对输入 N×N 的矩阵数据进行计算。

由于本次实验是对图像数据进行 DCT 正反变换，而 DCT 变换后数据类型为 double。故需对 DCT 反变换后结果进行修正，使其代表像素值介于 0-255 之间的整数，以便减少误差。

4.2.3 嵌入水印

由于有了上一次实验的基础，只需要将算法原理改为 STDM 原理即可。

（1）首先将整个图像分成 8×8 的小块，假设这样共得到 M 小块。例如：512×512 的 Lena 图，则 M=4096。

（2）对每小块进行 DCT 变换，使用 STDM 算法原理。

（3）根据水印嵌入公式进行水印嵌入，后对数据进行 DCT 反变换，利用得到数据替换原始对应位置数据信息。

（4）将 si 放回原来的位置。

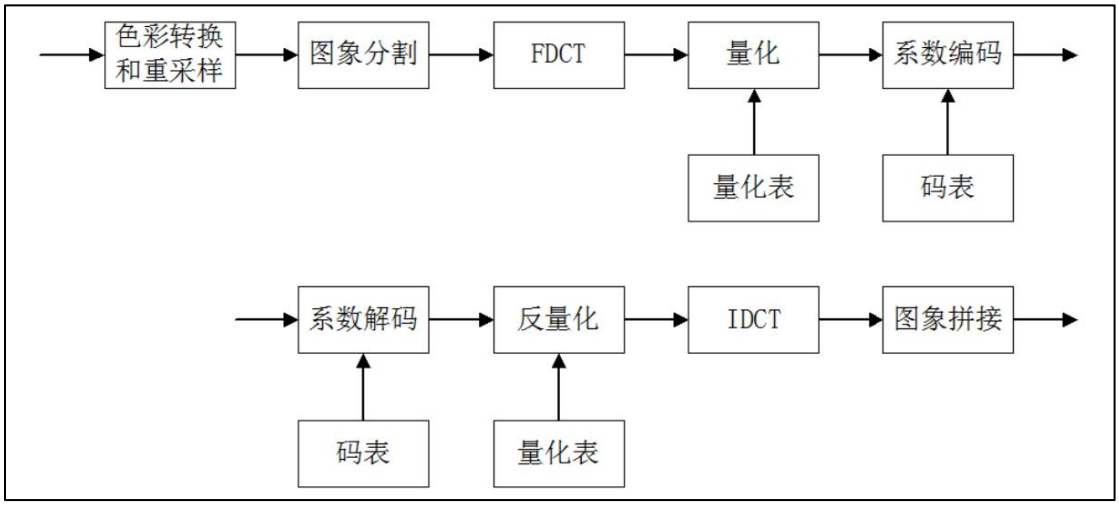
4.2.4 水印解码

读取输入的已嵌入水印图像信息，将其划分为 8×8 的小块，进行 DCT 正变换，利用线性相关器进行水印解码，并保存解码所得水印数据信息，同时，计算解码错误率。

4.2.5 模拟攻击

使用 matlab 程序对图像加入高斯噪声，模拟噪声攻击，分别计算不同量化步长下，相同噪声攻击强度的性能变化曲线，以及相同的量化步长下，不同强度的噪声攻击下系统性能的变化的曲线。

通过默认亮度量化表模拟 JPEG 压缩攻击过程，JPEG 压缩的过程如图 1 所示。



（图 1 JPEG 压缩过程）

我们通过如下的亮度量化表对图片的 dct8*8 矩阵进行量化和反量化，同时乘以压缩系数，得到不同压缩强度攻击下的 JPEG 图像结果。

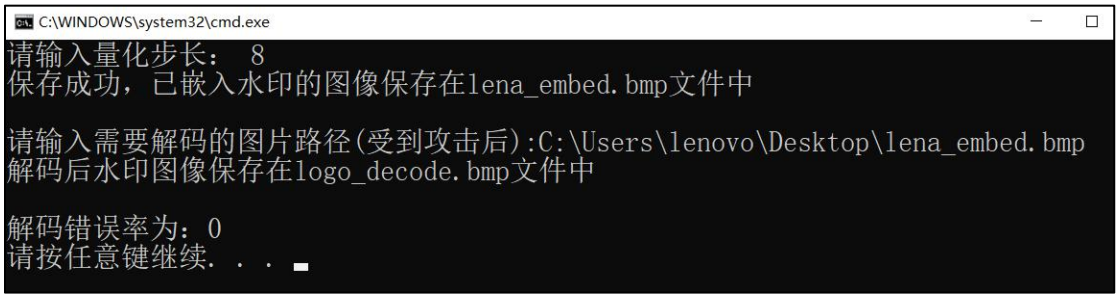
第五部分　实验结果与分析讨论

5.1 代码正确性验证

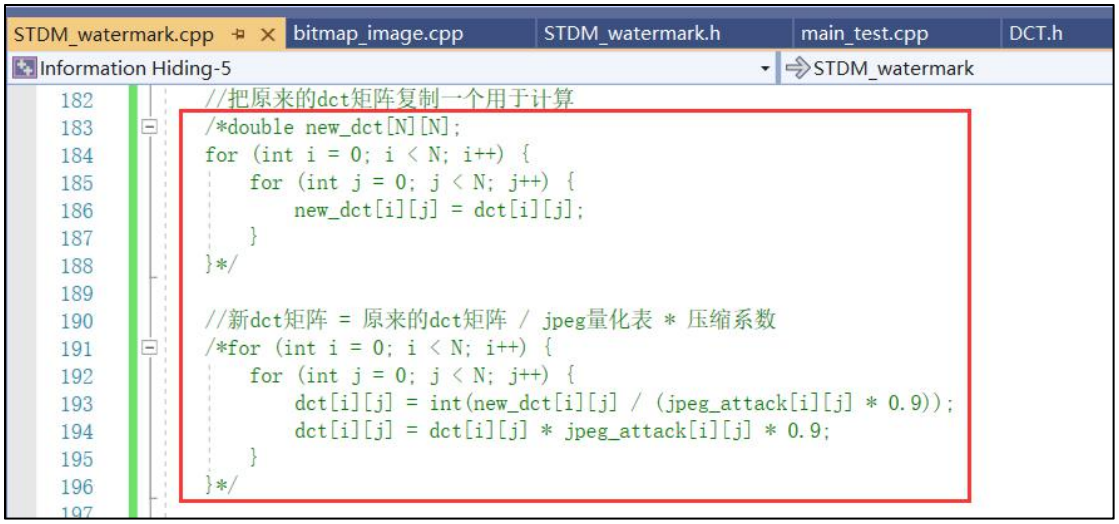
STDM 算法能达到零错误率，错误主要是反 DCT 变换时类型转化引起的误差。

经验证，在 N=8，步长为 8 时，能达到 0 错误率；如图 2 所示。

注意，由于 JPEG 压缩攻击写在代码中，在不进行 JPEG 压缩攻击时，均需要去掉这部分代码，如图 3 所示。



（图 2，在 N=8，步长为 8 时，算法能达到 0 解码错误率）



（图 3，STDM_watermark.cpp 中用于模拟 JPEG 压缩攻击的代码，在不攻击时要注释掉）

5.2 性能比较（不同噪声强度）

绘制在相同的量化步长下，不同强度的噪声攻击下系统性能的变化的曲线。

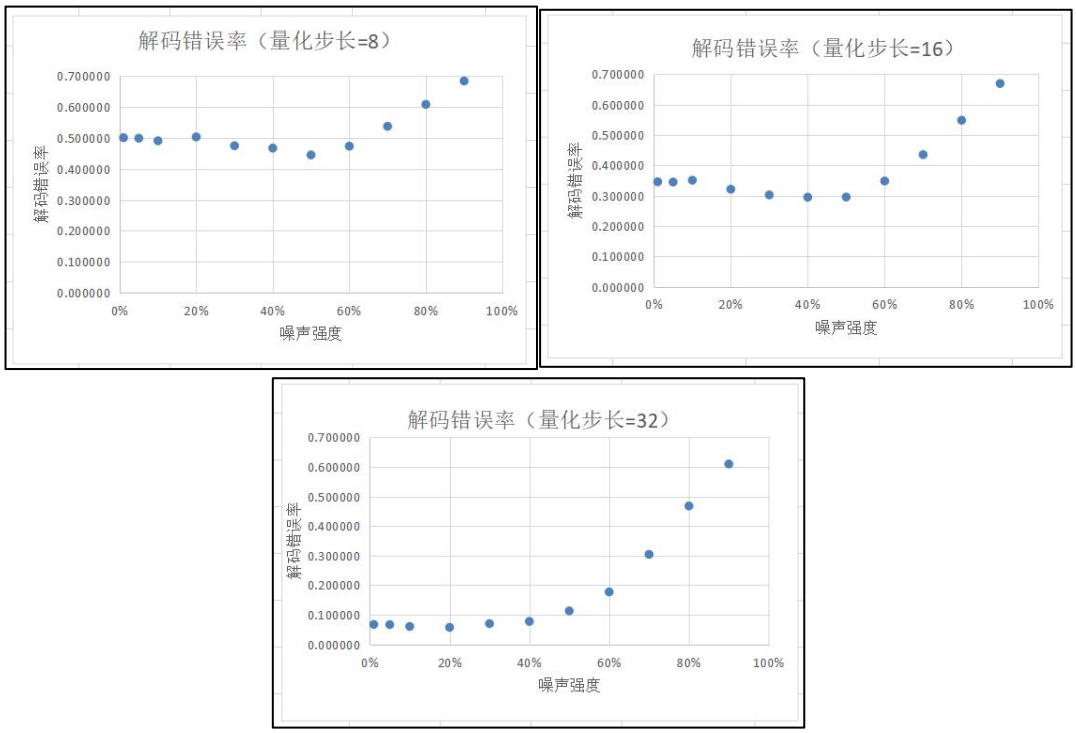
实验数据如表 2 所示：

（表 2，在相同的量化步长下，不同强度的噪声攻击下系统性能；实验采用 8、16、32 三组量化步长，噪声强度从 1%至 90%不等）

量化步长：8		量化步长：16		量化步长：32	
噪声	解码错误率	噪声	解码错误率	噪声	解码错误率
1%	0.500732	1%	0.345703	1%	0.067627
5%	0.499023	5%	0.344971	5%	0.066895
10%	0.490723	10%	0.351074	10%	0.060791
20%	0.503174	20%	0.321289	20%	0.057617
30%	0.474854	30%	0.302490	30%	0.070313
40%	0.467041	40%	0.295166	40%	0.077881
50%	0.445068	50%	0.295654	50%	0.113281
60%	0.473389	60%	0.348389	60%	0.177490
70%	0.537354	70%	0.435303	70%	0.304443
80%	0.608154	80%	0.548584	80%	0.468108

90%　　0.683594　　90%　　0.669678　　90%　　0.610107

实验曲线如图 4 所示：



（图 4，在相同的量化步长下，不同强度的噪声攻击下系统性能曲线；实验采用 8、16、32 三组量化步长，噪声强度从 1%至 90%不等）

5.3 性能比较（不同量化步长）

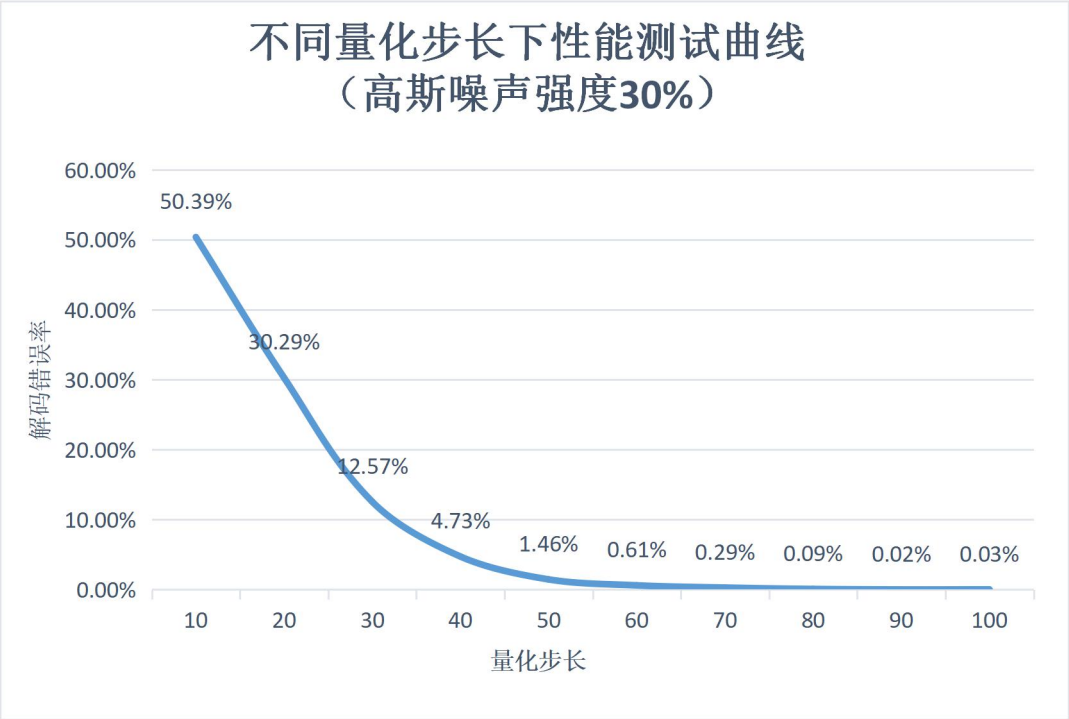
绘制在不同的量化步长下，相同噪声攻击强度的性能变化曲线。

实验数据如表 3 所示：

（表 3，在不同的量化步长下，相同噪声攻击强度的性能；实验采用 10-100 量化步长，噪声强度 30%）

量化步长	10	20	30	40	50
错误率	0.503906	0.302979	0.125732	0.0473633	0.014648
量化步长	60	70	80	90	100
错误率	0.006104	0.002929	0.000976	0.002685	0.003906

实验曲线如图 5 所示：



（图 5，在不同的量化步长下，相同噪声攻击强度的性能变化曲线；实验采用 10-100 量化步长，噪声强度 30%）

5.4 性能比较（不同强度 JPEG 压缩）

绘制在相同的量化步长下，不同强度的 JPEG 压缩下系统性能的变化的曲线。

实验数据如表 4 所示：

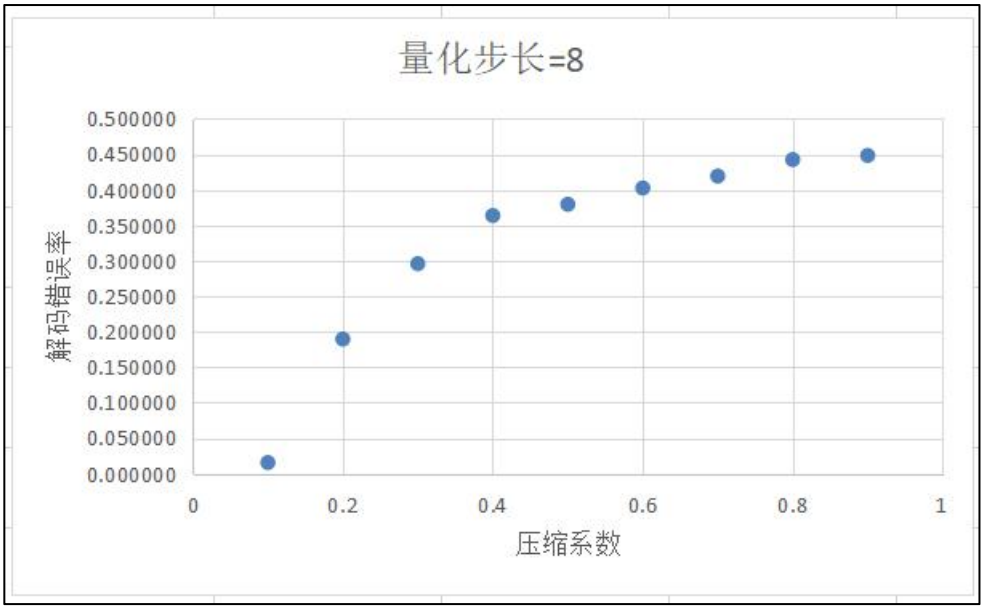
（表 4，在相同的量化步长下，不同强度的 JPEG 压缩下系统性能；实验采用 8 量化步长，压缩系数 0.1-0.9）

量化步长=8

压缩系数	解码错误率
0.1	0.015625
0.2	0.189697
0.3	0.295898
0.4	0.363770
0.5	0.379639
0.6	0.402588

0.7	0.419434
0.8	0.442627
0.9	0.448486

实验曲线如图 6 所示：



（图 6，在相同的量化步长下，不同强度的 JPEG 压缩下系统性能；实验采用 8 量化步长，压缩系数 0.1-0.9）

第六部分　实验结论

（1）通过绘制在相同的量化步长下，不同强度的噪声攻击下系统性能的变化的曲线，观察到：噪声攻击强度与系统解码错误率呈现正相关的趋势，特别是量化步长=32 时，明显观察到：噪声攻击强度越强，系统解码错误率越高。

（2）通过绘制在不同的量化步长下，相同噪声攻击强度的性能变化曲线，观察到：噪声攻击强度相同时，量化步长越大，系统解码错误率越低，呈现负相关。

（3）通过绘制在相同的量化步长下，不同强度的 JPEG 压缩下系统性能的变化的曲线，观察到：在相同的量化步长下，JPEG 压缩强度越大，系统解码错误率越高，呈现正相关的趋势。

第七部分　参考文献

[1] 水印嵌入实验：DCT 部分程序参考于 https://blog.csdn.net/BigDream123/article/details/104395587 离散余弦变换（DCT）的 C++实现，已在理解的基础上进行较大修改。
[2] JPEG 压缩原理：原理参考于：https://blog.csdn.net/shayashi/article/details/104182418。