实验一　基于矩阵编码的隐写工具使用

姓名：王运韬　　**学号**：201628018627123

一、实验目的：

理解矩阵编码类隐写算法 $F5$ 和 MME 的原理，学会使用 $F5$ 和 MME 等隐写工具进行秘密信息的嵌入和提取。

二、实验原理：

矩阵编码隐写的基本原理，对每个原文块，用 k 个嵌入消息比特向量和载体 2^k-1 个 LSB 比特向量共同决定一个嵌入位置，修改该嵌入位置的 LSB，可以是的载体 LSB 比特向量能解码得到嵌入比特向量，基本过程如下：

（1）将载体分块并计算器矩阵编码值。根据给定的 k 得到原文分块 $C=(c_1,c_2,...,c_{2^k-1})$，定义其 LSB 向量为 $CB=(cb_1,cb_2,...,cb_{2^k-1})$，其中 $cb_i=LSB(c_i)=c_i \bmod 2$。用下列公式计算其载体矩阵编码值：

$$mat(CB)=\sum_{i=1}^{2^k-1} cb_i \cdot i$$

其中，i 为 k 比特二进制数，加法为二进制域按抑或操作。

（2）将隐蔽消息分块并计算其数值。嵌入算法将隐蔽消息分为 k 比特的二进制消息快序列，逐块嵌入到每个含有 2^k-1 个样点的原文分块中。以下假设嵌入一个 k 比特的二进制消息块 MB。

（3）计算需要 LSB 取反的位置并做修改。需要修改的系数位置为 $p=mat(CB)\oplus MB$，当 $p>0$ 时才需要修改，嵌入后的分块系数 LSB 应为：

$$SB=(sb_1,sb_2,...,sb_{2^k-1})=\begin{cases}(sb_1,sb_2,...,\neg cb_p,...,sb_{2^k-1}) & p>0\\(sb_1,sb_2,...,sb_{2^k-1}) & p=0\end{cases}$$

其中，$\neg$ 表示取反操作。按照上述修改规则，将得到的 SB 序列替换分块系数的原 LSB。

提取时，计算当前分块信息 $MB=mat(SB)=\sum_{i=1}^{2^k-1} sb_i \cdot i$，再根据每次提取的

MB 组成提取信息序列。

MME 矩阵编码方法与以上方法基本相同，但在修改未知的选取上有所不同。

三、实验步骤：

1.制备秘密信息文件：第一行为自身姓名和学号，第二行开始为任意字符，最终制备的秘密信息文件，其大小控制在500*B* 左右；

2.制备 *F*5 隐写样本：以 *lopez.bmp* 为载体，使用 *f*5_*embed*.*jar* 嵌入制备的秘密信息文件，生成质量因子为90的 *F*5 隐写样本；

3.提取 *F*5 隐写样本中的秘密信息：对于制备的 *F*5 隐写样本，使用 *f*5_*extract*.*jar* 提取其中的秘密信息，并和原始秘密信息文件进行对比；

4.制备 *F*5 载体样本：以 *lopez.bmp* 为载体，使用 *f*5_*embed*.*jar*，但不嵌入任何秘密信息文件，生成质量因子为90的 *F*5 载体样本；

5.计算 *F*5 算法的嵌入效率：对比制备的 *F*5 隐写和载体样本，计算嵌入效率；

6.制备 *MME* 隐写样本：以 *lopez.bmp* 为载体，使用 *mme.jar* 嵌入制备的秘密信息文件，生成质量因子为90的 *MME* 隐写样本；

7.提取 *MME* 隐写样本中的秘密信息：对于制备的 *MME* 隐写样本，使用 *mme.jar* 提取其中的秘密信息，并和原始秘密信息文件进行对比。

8.制备 *MME* 载体样本：以 *lopez.bmp* 为载体，使用 *mme.jar*，但不嵌入任何秘密信息文件，生成质量因子为90的 *MME* 载体样本；

9.计算 *MME* 算法的嵌入效率：对比制备的 *MME* 隐写和载体样本，计算嵌入效率；

10.计算并对比 *F*5 和 *MME* 算法引入的隐写扰动：对于每种隐写算法，根据制备的隐写和载体样本，计算像素值均方误差，验证 *MME* 算法引入的隐写扰动较小。

四、实验结果：

1.样本文件说明：

Fig 1. 样本文件-lopez.bmp

属性	值
图像	
分辨率	315 x 199
宽度	315 像素
高度	199 像素
位深度	24
文件	
名称	lopez.bmp
项目类型	BMP 文件
文件夹路径	C:\用户\Charles_CatKing\桌面\School\上课教...
创建日期	2016/10/7 12:21
修改日期	2016/9/26 22:59
大小	184 KB
属性	A

Fig 2. 样本文件信息

2. 隐写及秘密消息提取使用说明：

```
命令提示符

C:\Users\Charles_CatKing>cd C:\Users\Charles_CatKing\Desktop\School\上课教材\信息隐藏_赵险峰\lab\实验一

C:\Users\Charles_CatKing\Desktop\School\上课教材\信息隐藏_赵险峰\lab\实验一>prompt Charles_wyt:

Charles_wyt:java -jar f5_embed.jar
F5/JpegEncoder for Java(tm)

Program usage: java Embed [Options] "InputImage"."ext" ["OutputFile"[.jpg]]

You have the following options:
-e <file to embed>      default: embed nothing
-p <password>           default: "abc123", only used when -e is specified
-q <quality 0 ... 100>  default: 80
-c <comment>            default: "JPEG Encoder Copyright 1998, James R. Weeks and BioElectroMech.  "

"InputImage" is the name of an existing image in the current directory.
  ("InputImage may specify a directory, too.) "ext" must be .tif, .gif,
  or .jpg.
Quality is an integer (0 to 100) that specifies how similar the compressed
  image is to "InputImage."  100 is almost exactly like "InputImage" and 0 is
  most dissimilar.  In most cases, 70 - 80 gives very good results.
"OutputFile" is an optional argument.  If "OutputFile" isn't specified, then
  the input file name is adopted.  This program will NOT write over an existing
  file.  If a directory is specified for the input image, then "OutputFile"
  will be written in that directory.  The extension ".jpg" may automatically be
  added.

Copyright 1998 BioElectroMech and James R. Weeks.  Portions copyright IJG and
  Florian Raemy, LCAV.  See license.txt for details.
Visit BioElectroMech at www.obrador.com.  Email James@obrador.com.
Steganography added by Andreas Westfeld, westfeld@inf.tu-dresden.de
```

Fig 3. f5_embed 隐写方法使用说明

Fig 4. f5_extract 秘密消息提取方法使用说明

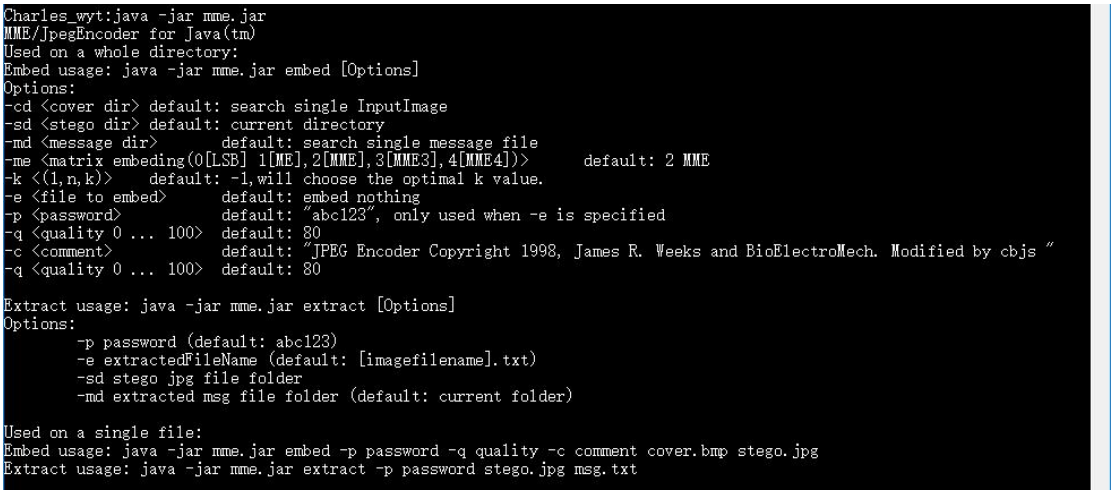

Fig 5. MME 隐写及秘密消息提取方法使用说明

3.制备秘密信息文件：

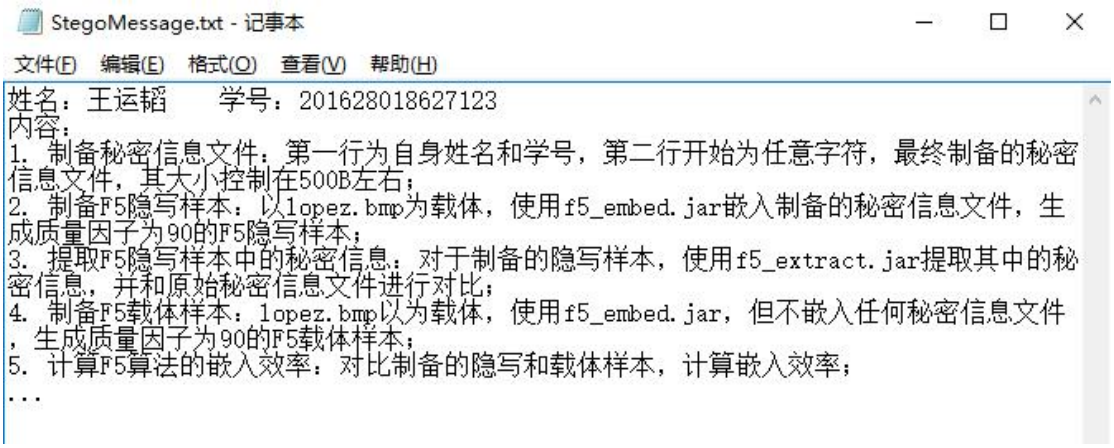

Fig 6. 秘密信息文本内容

Fig 7. 秘密信息文本大小

3.制备 *F5* 隐写样本：

```
Charles_wyt:java -jar f5_embed.jar -e StegoMessage.txt -p CatKing -q 90 lopez.bmp lopez_F5_90_StegoMessage_CatKing.jpg
DCT/quantisation starts
315 x 199
got 99840 DCT AC/DC coefficients
one=9399
large=11363
expected capacity: 15968 bits
expected capacity with
default code: 1996 bytes (efficiency: 1.5 bits per change)
(1, 3, 2) code: 1330 bytes (efficiency: 1.8 bits per change)
(1, 7, 3) code: 854 bytes (efficiency: 2.2 bits per change)
(1, 15, 4) code: 530 bytes (efficiency: 2.7 bits per change)
(1, 31, 5) code: 321 bytes (efficiency: 3.2 bits per change)
(1, 63, 6) code: 189 bytes (efficiency: 3.8 bits per change)
(1, 127, 7) code: 95 bytes (efficiency: 3.8 bits per change)
Permutation starts
Embedding of 4624 bits (574+4 bytes) using (1, 7, 3) code
2067 coefficients changed (efficiency: 2.2 bits per change)
918 coefficients thrown (zeroed)
4624 bits (578 bytes) embedded
Starting Huffman Encoding.
```

Fig 8. F5 隐写样本制备（Cover: lopez.bmp, Quality:90, 嵌入秘密消息）

Fig 9. 载体文件

Fig 10. 隐写文件

4.提取 *F5* 隐写样本中的秘密信息：

```
Charles_wyt:java -jar f5_extract.jar -p CatKing -e StegoMessage_F5_Extract.txt lopez_F5_90_StegoMessage_CatKing.jpg
Huffman decoding starts
Permutation starts
99840 indices shuffled
Extraction starts
Length of embedded file: 574 bytes
(1, 7, 3) code used
```

Fig 11. 提取 F5 隐写样本中的秘密消息

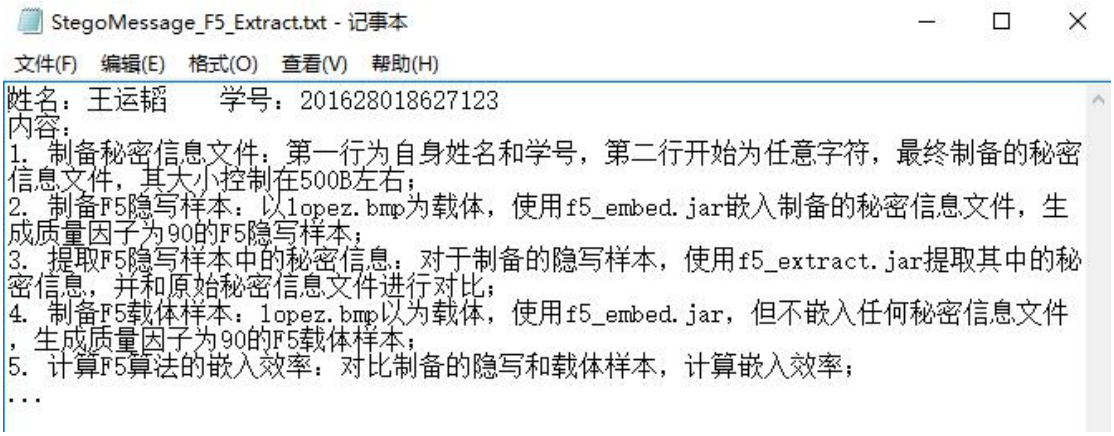

Fig 12. 提取的秘密消息

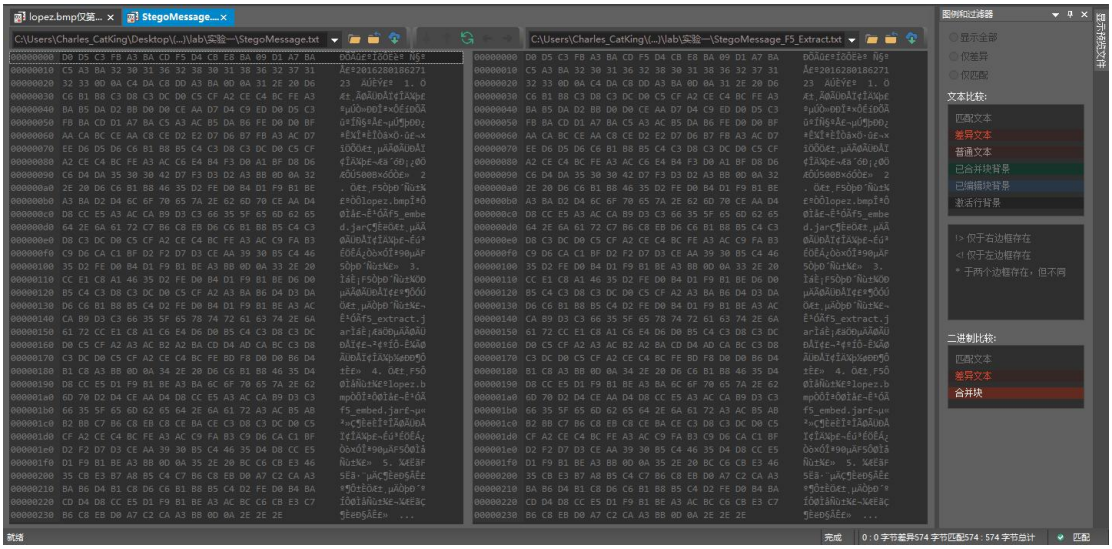

Fig 13. 嵌入秘密消息与提取的秘密消息码流对比

5.制备 *F5* 载体样本（不嵌入秘密消息）：

```
Charles_wyt:java -jar f5_embed.jar -p CatKing -q 90 lopez.bmp lopez_F5_90_NoMessage_CatKing.jpg
DCT/quantisation starts
315 x 199
got 99840 DCT AC/DC coefficients
one=9399
large=11363
expected capacity: 15968 bits
expected capacity with
default code: 1996 bytes (efficiency: 1.5 bits per change)
(1, 3, 2) code: 1330 bytes (efficiency: 1.8 bits per change)
(1, 7, 3) code: 854 bytes (efficiency: 2.2 bits per change)
(1, 15, 4) code: 530 bytes (efficiency: 2.7 bits per change)
(1, 31, 5) code: 321 bytes (efficiency: 3.2 bits per change)
(1, 63, 6) code: 189 bytes (efficiency: 3.8 bits per change)
(1, 127, 7) code: 95 bytes (efficiency: 3.8 bits per change)
Starting Huffman Encoding.
```

Fig 14. F5 隐写样本制备（Cover: lopez.bmp, Quality:90，不嵌入秘密消息）

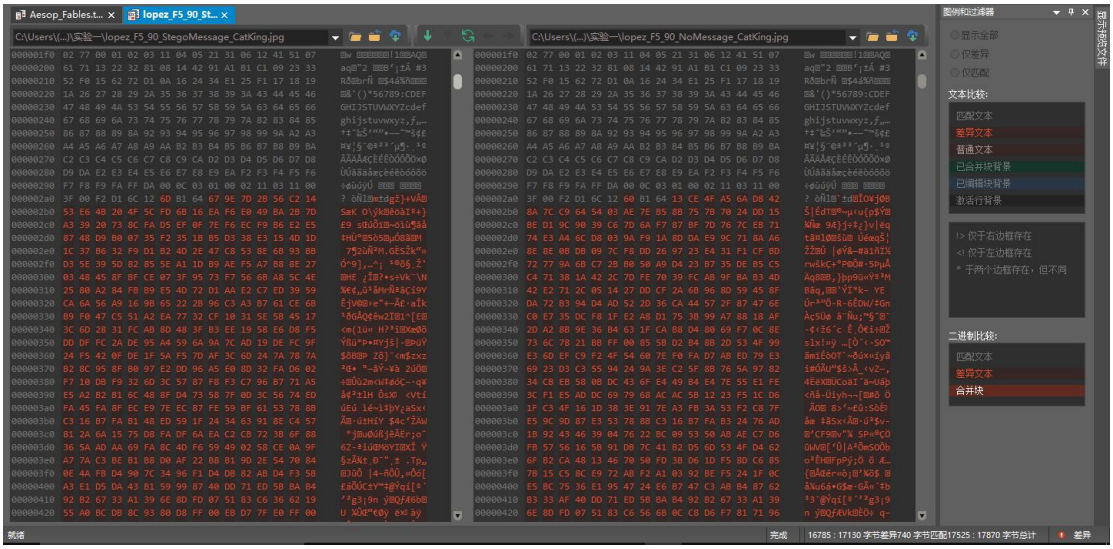

Fig 15. 嵌入秘密消息和不嵌入秘密消息的文件码流对比

6.计算 *F5* 算法的嵌入效率：

```
expected capacity: 15968 bits
expected capacity with
default code: 1996 bytes (efficiency: 1.5 bits per change)
(1, 3, 2) code: 1330 bytes (efficiency: 1.8 bits per change)
(1, 7, 3) code: 854 bytes (efficiency: 2.2 bits per change)
(1, 15, 4) code: 530 bytes (efficiency: 2.7 bits per change)
(1, 31, 5) code: 321 bytes (efficiency: 3.2 bits per change)
(1, 63, 6) code: 189 bytes (efficiency: 3.8 bits per change)
(1, 127, 7) code: 95 bytes (efficiency: 3.8 bits per change)
Permutation starts
Embedding of 4624 bits (574+4 bytes) using (1, 7, 3) code
2067 coefficients changed (efficiency: 2.2 bits per change)
918 coefficients thrown (zeroed)
4624 bits (578 bytes) embedded
Starting Huffman Encoding.
```

Fig 16. F5 隐写算法嵌入效率说明

嵌入效率：平均每修改一个位置所能传输的隐蔽消息信息量，若信息量用比特表示，则计算公式为：

$$e = \frac{\text{平均每载体样点承载的信息比特}}{\text{平均每载体样点被修改量}} bits / \text{次修改}$$

嵌入的信息比特：4624 *bits*，2067 个系数被修改，则嵌入效率为 2.24 *bits* / 次修改，约为 2.2 *bits* / 次修改，如上图所示。

7.制备 *MME* 隐写样本：

```
Charles_wyt:java -jar mme.jar embed -p CatKing -e StegoMessage.txt -q 90 lopez.bmp lopez_MME_90_StegoMessage_CatKing.jpg

Quality:90      K:-1    ME[MME] used

Cover:lopez.bmp Stego:lopez_MME_90_StegoMessage_CatKing.jpg     password:CatKing
Embedding of 4624 bits (574+4 bytes) using (1, 15, 4) code
1637 coefficients changed (efficiency: 2.8 bits per change)
4625 bits (578 bytes) embedded
```

Fig 17. MME 隐写样本制备（Cover: lopez.bmp, Quality:90, 嵌入秘密消息）

Fig 18. 载体文件

Fig 19. 隐写文件

8.提取 *MME* 隐写样本中的秘密信息：

```
Charles_wyt:java -jar mme.jar extract -p CatKing -e Message_MME_Extract.txt lopez_MME_90_StegoMessage_CatKing.jpg
StegoFile:lopez_MME_90_StegoMessage_CatKing.jpg MsgFile:Message_MME_Extract.txt password:CatKing
Extraction starts
(1, 15, 4) code used
Complete: 574 bytes extracted
```

Fig 20. 提取 MME 隐写样本中的秘密消息

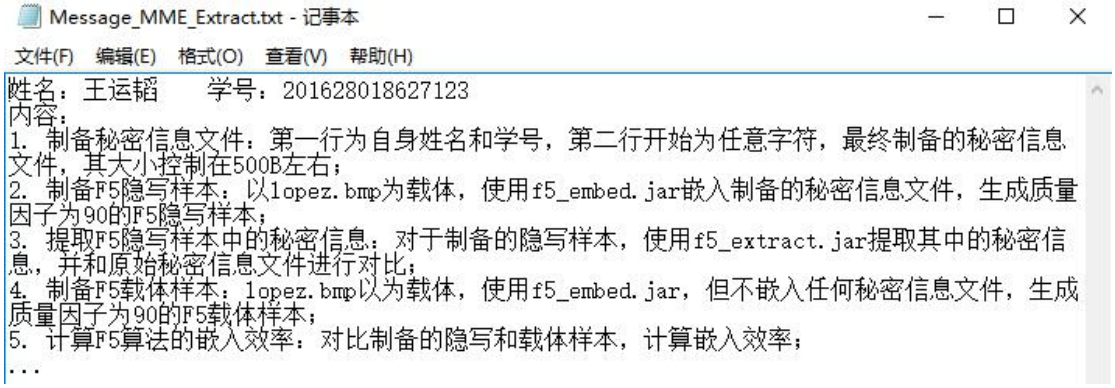

Fig 21. 提取的秘密消息

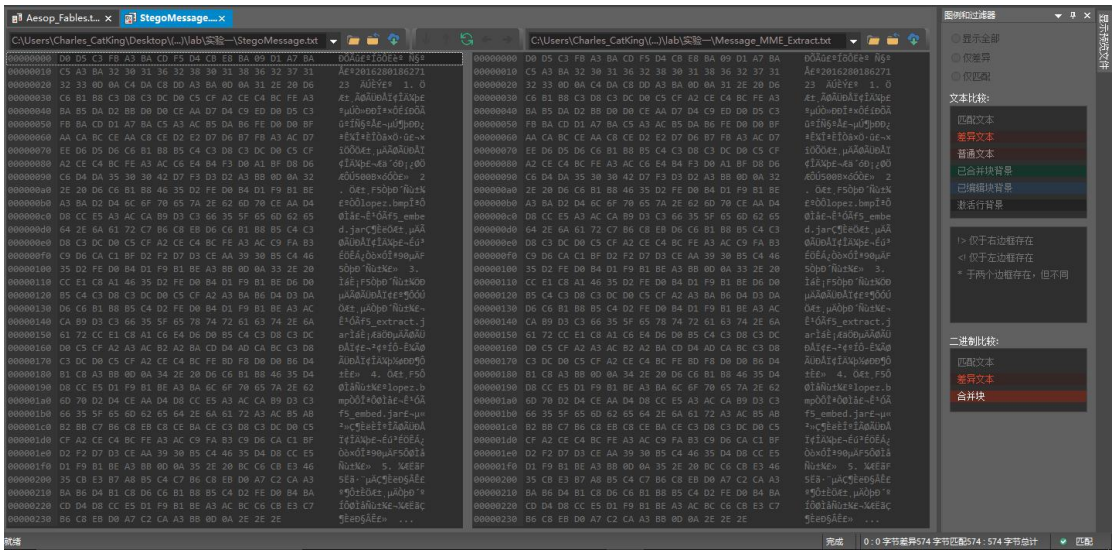

Fig 22. 嵌入秘密消息和不嵌入秘密消息的文件码流对比

10.制备 *MME* 载体样本（不嵌入秘密消息）：

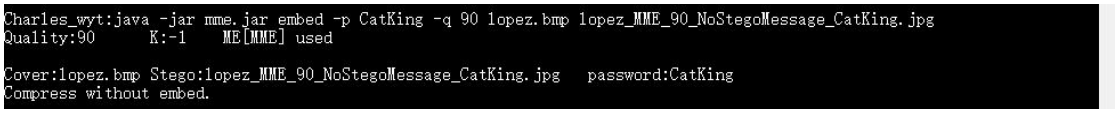

Fig 23. MME 隐写样本制备（Cover: lopez.bmp, Quality:90, 不嵌入秘密消息）

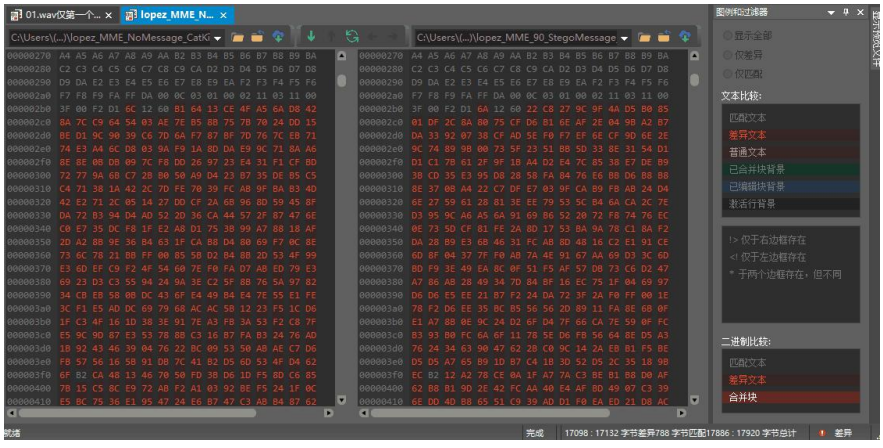

Fig 24. 嵌入秘密消息和不嵌入秘密消息的文件码流对比

12.计算 *MME* 算法的嵌入效率：

```
Cover:lopez.bmp Stego:lopez_MME_90_StegoMessage_CatKing.jpg     password:CatKing
Embedding of 4624 bits (574+4 bytes) using (1, 15, 4) code
1637 coefficients changed (efficiency: 2.8 bits per change)
4625 bits (578 bytes) embedded
```

Fig 25. MME 算法的嵌入效率说明

嵌入效率：平均每修改一个位置所能传输的隐蔽消息信息量，若信息量用比特表示，则计算公式为：

$$e=\frac{\text{平均每载体样点承载的信息比特}}{\text{平均每载体样点被修改量}}bits/\text{次修改}$$

嵌入的信息比特：4624 *bits*，1637 个系数被修改，则嵌入效率为 $2.82\,bits/$次修改，约为 $2.8\,bits/$次修改，如上图所示。

11.计算并对比 $F5$ 和 *MME* 算法引入的隐写扰动：分别对 $F5$ 算法和 *MME* 算法制备的隐写样本和载体样本，计算其像素值均方误差，将其作为扰动大小的评估。

均方根计算公式 $MSE=\sqrt{\dfrac{\sum_{i=1}^{n}x_i^2}{n}}$

$F5$ 算法：

Fig 26. F5 隐写算法　　Fig 27. MME 隐写算法

MSE：$MSE\_F5=5.3131$　$MSE\_F5=5.0287$

$MSE\_F5>MSE\_MME$，由此说明 *MME* 引入的隐写扰动更小。

五、实验结论

$F5$ 实现了基于海明码的矩阵编码隐写，在一个分组上最多修改 $R=1$ 次，采用的基本嵌入方法是基于 $F4$。

从等价消息提取的角度看，矩阵编码提供了很多嵌入方法，如果用修改次数衡量隐蔽性，那可以用嵌入效率作为指标。但是如果从信号扰动的角度看，多次嵌入的扰动不一定比一次扰动程度低。

MME 基于这一原因，在等价嵌入方式中进行优化选择的矩阵编码，在要求修改的情况下，选扰动最小的嵌入方法（包括不分解、仅仅修改1次的情况）进行分组嵌入，引入的隐写扰动更小，体现了一定的自适应思想。

参考文献

[1].A. Westfeld. F5 — a steganographic algorithm. In Proc. IH'01, LNCS 2137:289 - 302, Springer-Verlag, 2001.

[2].J. Fridrich and D. Soukal. Matrix embedding for large payloads, IEEE Trans.Information Forensics and Security, 1(3): 390-395, 2006.

[3].Y. Kim, Z. Duric, and D. Richards. Modified matrix encoding technique for minimal distortion steganography. In Proc. IH'06, LNCS 4437: 314 – 327,Springer-Verlag, 2007.

附录：

MSE 计算代码：

Code:

```
clc;
clear all;
close all;

h = figure;
image_cover = imread('lopez.bmp');
image_stego = imread('lopez_F5_90_StegoMessage_CatKing.jpg');
subplot(121);imshow(image_cover);title('Cover');
subplot(122);imshow(image_stego);title('Stego');
D = image_cover - image_stego;
MSE = sum(D(:).*D(:)) / numel(image_cover);
annotation(h,'textbox',[0.4 0.8 0.2 0.05],'String',['MSE=' num2str(MSE)]);
% 注：只需更改文件名即可
```

注：文件命名方式：

"cover_file_name" + "stego_algorithm" + "quality" + "stego_message_file_name" + "password".jpg