

# STEGANALYSIS OF LSB INSERTION METHOD IN UNCOMPRESSED IMAGES USING MATLAB

**Mohit Kr. Srivastava**  
Asst. Prof.  
Ph: 9935428632  
mohit1003@yahoo.co.in

**Sharad Kr. Gupta**  
Asst. Prof.  
Ph: 9839164608  
sharad\_mpec@rediffmail.com

**Sushil Kushwaha**  
Asst. Prof.  
Ph: 9335891362  
kushwahasushil@rediffmail.com

**Brishket S. Tripathi**  
Lecturer  
Ph: 9236119790  
brishket@gmail.com

Department of Electronics Engineering , MPEC, Kothi Mandhana, Kanpur

## ABSTRACT

In this paper we have explained a steganalytic tool to detect the presence of hidden message in LSB steganography. It is process of hiding secret information in a cover image. Here we have used LSB insertion method using discrete algorithms. Steganographic programs use the Least Significant Bit (LSB) embedding as the method of choice for message hiding 24-bit, 8-bit color images and grayscale images. It is commonly understood that changes to the LSB of color cannot be detected due to noise that is always presents in digital images. In this paper, we describe a new very accurate and reliable method that can detect LSB. Embedding in randomly scattered pixels in both 24-bit color images and 8-bit grayscale or color images.

## Keywords

Steganography, Stegoimage, LSB insertion, Computer and Network Security

## 1. INTRODUCTION

Steganography brings science to the art of hiding information. The purpose of steganography is to convey a message inside of a conduit of misinterpretation such that the existence of the message is both hidden and difficult to recover when discovered. Basically the information hiding process in a Steganographic system starts by identifying a cover medium's redundant bits. The embedding process creates a stego medium by replacing these redundant bits with data from the hidden message. The basic purpose to make communication unintelligible to those who do not possess the right keys.

The first step is steganography is that to embed and hiding information is to pass both the secret message and the cover message in to the encoder, inside the encoder, one or several protocols will be implemented to embed the secret information into the cover message.

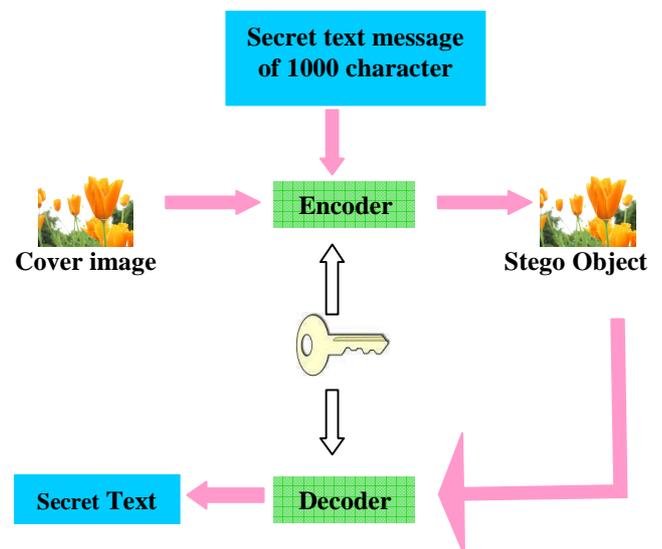


Fig.1: Steganography Process

A key is needed in the embedding process. By using the key we can reduce the chance of third party attackers getting hold of the stego object and decoding it to find out the secret information. In general the embedding process inserts a mark X, in an object Y, a key K, usually produced by a random number generator is used in the embedding process and the resulting marked object Y is generated by mapping

$$X \times Y \times K \longrightarrow Y$$

Having passed through the encoder a stego object will be produced. A stego object is the original

cover object with the secret information embedded inside. This object should look almost identical to the cover object as otherwise a third party attacker can see embedded information. Having produced the stego object, It will be sent off via some communication channel. At the receiving end the stego object is fed into the system the public or private key that can decode the original key that is used inside the encoding process is also needed to detect the secret information.

## 2. LSB INSERTION TECHNIQUES

In LSB insertion method, a random number generator is used to randomly distribute and hide the bits of a secret message into the least significant bit of the pixels within a cover image, a common approach to achieve this is the random interval method. The transmitting and receiving end share the stego key, the output is a random sequence  $K_1, \dots, K_n$  where  $n$  is the length of message bits.

The sequence is then used by the sender to generate the sequence of pixel indices  $y_i$  where,

$$y_1 = K_1$$

$$y_i = y_{i-1} + K_i, \quad i \geq 2.$$

Message bit,  $i$  would then be embedded into the LSB of the pixel,  $y_i$  thus, the order in which the secret message bits are embedded would be determined pseudo randomly. Since the receiver knows the seed  $k$ , he can reconstruct  $k_i$  and therefore the entire sequence of pixel indices  $y_i$ .

In the random insertion method the random location of the pixels depends on a stego key, whose size  $k$  should be in the range  $n < k < I$ . Where  $n$  is the size of message and  $I$  is the size of cover image.

The method commences by searching for the first prime number  $p$  that exceeds the key  $k$ , A primitive root  $a$ , is then obtained, which is a number whose powers generate all the distinct integers from 1 to  $(p-1)$  in some permuted order. Each power of this primitive root to generate these integers is called the discrete algorithm. This primitive root  $a$ , is then used to generate a set of random and distinct numbers,  $y_i = a^i \text{ mod } p$ , where  $i$  is the bit index of the secret message. Bit  $i$  of the secret message then goes into LSB of pixel

$y_i$ . In this way it is ensured that the bits of the secret message are inserted into distinct LSBs.

## 3. ANALAYSIS TO DETECT LSB INSERTION

In a natural uncompressed image 24-bit BMP each pixel is represented by three colors (Red, Green & Blue), each of the color is 8-bit wide. The LSB of any color pixel of the typical scanned real image taken by any source contains least information about the image and is most random in nature. That is why the most appropriate and acceptable technique for hiding information in an uncompressed natural image is based on the replacing the LSB of the color pixels by message bit. So that on an average only half of the LSBs are changed and it is assumed that, embedding message in this way is not going to destroy the statistics of the original image. This assumption is true if and only if the number unique colors in the cover image is comparable to the total number of pixels in that image.

In an uncompressed image the ratio of the number of unique colors to the total number of pixels is approximately 1:6 so that after LSB embedding which is equivalent of introducing noise, the randomness of the LSB pattern will increase. This increase in randomness is reflected in increased in number of closed color pairs which is utilized as the detection tool. The close color pair (X) and unique color (Y) is defined as follows.

Two colors having individual components Red, Green & Blue are closed if the correlation factor of all individual components of both colors are 1 and both colors are unique if any one of the individual components of both colors are 1.

For any uncompressed real image the ratio  $r$  gives us an idea about the relative number of close color pair with that of unique colors where,

$$r = X/Y$$

It has been seen for an original image which does not have any encoded message the value of  $r$  is greater incomparision with an image which has a message already encoded in it. This happens has embedded message behaves as a random noise which increases the number of unique colors  $Y$  abruptly.

#### 4. ALGORITHM FOR SOFTWARE DEVELOPMENT

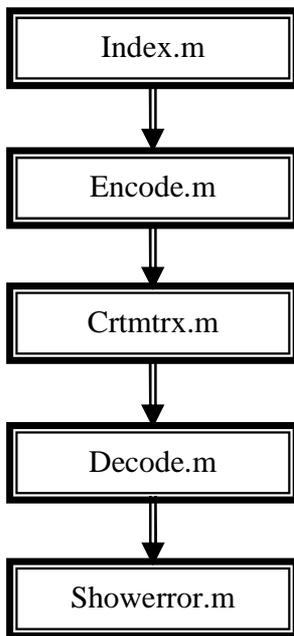


Fig.2 Flow chart for Steganalysis of LSB insertion method

##### 4.1 INDEX.M

It maintains the graphical user interface of program for making user friendly software index.m is main file, which cause various other module it is actually the simplest part you just have to select any of GUI component such as push button, slider or any else and then double click on it a inspector dialog box will open in which there will be various option such as create function, edit function and various other option click on create function block guide will automatically create a function whenever you will press that button you will jump to its callback. Now job of programmer is to edit this function according to its use.

The handles structure contains the handles of all GUI components. GUI automatically passes the handles structure to every callback as an input argument.

##### 4.2 ENCODE.M

This module is used for encoding purpose. It inputs a BMP image and message to be encoded and the key for protecting the code is provided by calling of program. This code is prepared for

encoding any message upto 1000 character. Hence first job is to convert any message to message of 1000 character and then converting it into ASCII code. Hence finally we have a matrix of 7x1000. Now our job is to form a zero matrix of same dimension as image and then provide that matrix to crtmtx module. This will return a matrix of same dimension whose 7000 element would be 1. We will find index of these position and choose as reference and modulated that position element of picture matrix.

```

Function success= Encode(msg,key,F,F1)
% ENCODE(msg,key) Encodes a text message
num2add = 1000-length(msg);
% Number of spaces to add to end of MSG.
if num2add < 0, error('This message is too long to
encode. '), end
newmsg = [msg, repmat(' ',1,num2add)];
% 1000 chars always encoded.
msgmat = dec2bin(newmsg)-48;
% Each row is a bin. rep. of an ascii char.
pic1 = imread(fullfile(F,F1));
B = pic1(:,:,1); [piclngth pichght] = size(B);
% Choose the first page.
dim1 = piclngth-2; dim2 = pichght-3;
keyb = key(end:-1:1);
rows = cumsum(double(key));
columns = cumsum(double(keyb));
% Coord pairs for KEY (rows, columns)
A = zeros(dim1,dim2);
% This matrix will house the hiding points.
A = crtmtx(A,rows,columns,dim1,dim2,key);
idx = find(A==1);
% This same index will be used for pic matrix.
for vv = 1:1000
% This is the encoder.
for uu = 1:7
if msgmat(vv, uu)==1;
if rem(B(idx(uu+7*(vv-1))),2)==0
B(idx(uu+7*(vv-1))) = B(idx(uu+7*(vv-1)))+1;
end
elseif rem(B(idx(uu+7*(vv-1))),2)==1
B(idx(uu+7*(vv-1))) = B(idx(uu+7*(vv-1)))-1;
end
end
end
newpic = pic1; newpic(:,:,1) = B;
imwrite(newpic,sprintf('%sENC%s',F,F1))
success = 1;
  
```

### 4.3 CRTMTRX.M

This module is not called by main program (index.m) but by encoding and decoding module. It uses a key for security purpose. A zero matrix provided by encoding module and finally form a matrix which modulated form of zero matrix whose 7000 element are 1. These are those element we store the information in picture message. Finally this module is also called by decoding module so that we can obtain those position where information is hidden and extract that information easily.

```
function [varargout] = judp(actionStr,varargin)
% Usage :
% mssg = judp('receive',21566);
% judp('send',21566,'192.167.1.2','test.bmp')
SEND = 1;
RECEIVE = 2;
DEFAULT_LENGTH = 100000;
DEFAULT_TIMEOUT = 10000;
if strcmpi(actionStr,'send')
action = SEND;
if nargin ~= 4
error(['mfilename 'm--SEND mode requires 4 input
arguments.']);
end
port = varargin{1};
host = varargin{2};
filename = varargin{3};
elseif strcmpi(actionStr,'receive')
action = RECEIVE;
if nargin ~= 2
error(['mfilename 'm--RECEIVE mode requires 2
input arguments.']);
end
port = varargin{1};
packetLength = DEFAULT_LENGTH;
timeout = DEFAULT_TIMEOUT;
else
error(['mfilename 'm--Unrecognised actionStr "'
actionStr '".']);
end
if ~isnumeric(port) || rem(port,1)~=0 || port < 1025 ||
port > 65535
error(['mfilename 'm--Port number must be an integer
between 1025 and 65535.']);
end
if action == SEND
if ~ischar(host)
error(['mfilename 'm--Host name/IP must be a string
(e.g., "www.example.com" or "208.77.188.166").']);
end
```

```
end
import java.io.*
import java.net.DatagramSocket
import java.net.DatagramPacket
import java.net.InetAddress
if action == SEND
try
addr = InetAddress.getByName(host);
fp = fopen(filename,'r');
data = fread(fp,'int8');
fclose(fp);
packet = DatagramPacket(data, length(data), addr,
port);
socket = DatagramSocket;
socket.send(packet);
socket.close;
catch lasterr
errorStr = sprintf('%s.m--Failed to send UDP
packet.\nJava error message
follows:\n%s',mfilename,lasterr);
try
socket.close;
catch lasterr
errorStr = sprintf('%s.m--Failed to send UDP
packet.\nJava error message
follows:\n%s',mfilename,lasterr);
end
error(errorStr);
end
else
try
socket = DatagramSocket(port);
socket.setSoTimeout(timeout);
packet =
DatagramPacket(zeros(1,packetLength,'int8'),packetLe
ngth);
socket.receive(packet);
socket.close;
mssg = packet.getData;
mssg = mssg(1:packet.getLength);
inetAddress = packet.getAddress;
sourceHost = char(inetAddress.getHostAddress);
fp = fopen('C:\received.wav','w');
fwrite(fp,mssg);
fclose(fp);
sprintf('%s', 'File Transfer Successful');
if nargin > 1
varargout{2} = sourceHost;
end
catch receiveError
% Determine whether error occurred because of a
timeout.if
~isempty(strfind(receiveError.message,'java.net.Socke
tTimeoutException'))
```

```

errorStr = sprintf('%s.m--Failed to receive UDP
packet; connection timed out.\n',mfilename);
else
errorStr = sprintf('%s.m--Failed to receive UDP
packet.\nJava error message
follows:\n%s',mfilename,receiveError.message);
end
try
socket.close;
catch lasterr
sprintf('There is some error : %s',lasterr);
end
error(errorStr);
end
end

```

#### 4.4 DECODE.M

This module is called by main program (index.m). It is provided by and encoded image exciting part of this program is to we store encoded image on same here our original image lies hence we just have to supply the part as input is calls crtmtrx.m which return a matrix that contain those position where encoding is done and then we run same nested loop as we used in encoding loop an finally we get back original matrix.

```

function msg = Decode(key,F,F1)
% DECODE(key) Decodes the message hidden by
encode in pic, using key.
pic2 = imread([F F1]);
B = pic2(:,1); [piclngh pichght] = size(B);
% Choose the top page.
dim1 = piclngh-2; dim2 = pichght-3;
keyb = key(end:-1:1);
rows = cumsum(double(key)); columns =
cumsum(double(keyb));
A = zeros(dim1,dim2);
% This matrix houses the hiding points.
A=crtmtrx(A,rows,columns,dim1,dim2,key);
idx = find(A==1);
msgmat = zeros(1000,7);
for vv = 1:1000
% This is the decoder.
for uu = 1:7
if rem(B(idx(uu+7*(vv-1))),2)==1
msgmat(vv,uu) = 1;
end
end
end
msg = char(bin2dec(num2str(msgmat)));

```

#### 4.5 SHOWERROR.M

This is a short module that is used to show a error. It returns a message box which contain error message an icon representing error.

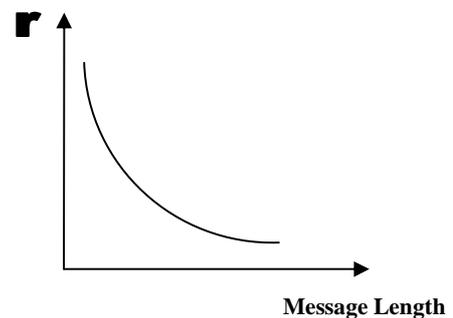
```

function [y] = showError(message,icon)
y = msgbox(message,'Secured Steganography with
File Transfer',icon);

```

#### 5. CONCLUSION

The work in this paper is successfully completed. As we consider many message of different length which shows that the message length is inversely proportional to the ratio ( r ), as shown in following figure.



Here we have a limitation also that this technique does not work properly, when we consider a .jpg format of any cover image because such type of compression technique is lossy technique. LSB insertion technique is working perfectly with Lossless compression techniques.

#### 6. REFERENCES

- [1] J. Fridrich and M. Goljan, "Steganalysis of LSB Embedding in color and Grayscale image", in preparation for the special issue on security in Magazine IEEE Multimedia.
- [2] R. J. Anderson and Petitcolas, F.A.P., "On the limits of steganography", *IEEE Journal of selected Areas in Communications*.
- [3] Curran, K. and Bailey, K. "An evaluation of image based steganography method". International Journal of Digital Evidence, Fall2003.
- [4] Jakson, J.T., Gregg, H., Gunsch, Claypoole, R.L., and Lamont, G.B. "Blind Steganography detection using a computational immune system: A Work in progress". International Journal of DigitalEvidence.
- [5] Digital Image Processing using MATLAB by Gonzale Woods.
- [6] N. Provos and P. Honeyman,"Hide and seek: An introduction to Steganography".

