International Research Journal of Computer Science (IRJCS)
Issue 5, Volume 2 (May 2015)
ISSN: 2393-9842
www.irjcs.com

Cryptography and Steganography for Securing Confidential Images

Tejasvi Kattimani T R
Dept.Of ISE, GMIT

Veeragangadhara Swamy T M
Dept.Of ISE, GMIT

Amith Shekhar C
Dept.Of ISE, GMIT

Abstract— we have a concept that can provide security to images. Cryptography and Steganography were the technologies used for this purpose from a very long period. We are combining both the technologies to provide a double layered protection to the images. Our concept encrypts and hides the confidential image in another image. Hence securing the confidential image is done successfully.

Keywords— Cryptography, Steganography, Confidential image, Stego image, Cover image.

I. INTRODUCTION

To actively embrace the internet as a distribution medium, the incorporation of digital Steganography and Cryptography as a security feature for our valuable documents has evolved to be an essentiality. Our goal is to design a digital Steganography solution that implants Data from Text file on images thus providing an essential first step in authenticity and privacy protection. Our solution supports Authentication of images for content confidentiality and Privacy protection. Encryption is the solution for hiding data. The encryption techniques applied for data cannot be extended to images because of its limitations. The other technique used for hiding image is steganography. In this technique the image can be hidden in other image. The change in the cover image can be observed in the histogram of the stego image. And also hacker can try out standard methods of steganography

II. SYSTEM DESIGN

The encryption of the image happens by the scrambling of the image in the manner explained below. The entire image is first broken up into cells, and arranged in rows and columns. Below figure arrangement shows a picture dissected into cells. We have shown the numbering in order make the explanation clear.

X	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	11	12	13	14	15	16	17	18	19	20
3	21	22	23	24	25	26	27	28	29	30
4	31	32	33	34	35	36	37	38	39	40
5	41	42	43	44	45	46	47	48	49	50
6	51	52	53	54	55	56	57	58	59	60
7	61	62	63	64	65	66	67	68	69	70
8	71	72	73	74	75	76	77	78	79	80
9	81	82	83	84	85	86	87	88	89	90
10	91	92	93	94	95	96	97	98	99	100

Figure 1: Image decomposed into cells.

After obtaining the decomposed table the program conducts a 2D reverse of the entire matrix. The result of the operation is shown in the figure

X	1	2	3	4	5	6	7	8	9	10
1	10	9	8	7	6	5	4	3	2	1
2	20	19	18	17	16	15	14	13	12	11
3	30	29	28	27	26	25	24	23	22	21
4	40	39	38	37	36	35	34	33	32	31
5	50	49	48	47	46	45	44	43	42	41
6	60	59	58	57	56	55	54	53	52	51
7	70	69	68	67	66	65	64	63	62	61
8	80	79	78	77	76	75	74	73	72	71
9	90	89	88	87	86	85	84	83	82	81
10	100	99	98	97	96	95	94	93	92	91

Figure 2: The result of 2D reverses of the entire matrix of cells.

After the 2D reverse of the matrix the matrix obtained is shown in the figure above. The next step is to reverse the odd rows only. The result of which is shown in the figure below.

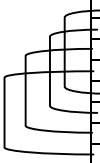
X	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	20	19	18	17	16	15	14	13	12	11
3	21	22	23	24	25	26	27	28	29	30
4	40	39	38	37	36	35	34	33	32	31
5	41	42	43	44	45	46	47	48	49	50
6	60	59	58	57	56	55	54	53	52	51
7	61	62	63	64	65	66	67	68	69	70
8	80	79	78	77	76	75	74	73	72	71
9	81	82	83	84	85	86	87	88	89	90
10	100	99	98	97	96	95	94	93	92	91

Figure 3: Result after the reversal of the odd rows.

In the next step we calculate 10 delimiters. Based on the delimiters we shuffle the rows and columns. The delimiters specify the numbers of cells to be shuffled with same number of adjacent cells. Column wise shuffling of the cells is shown in the figure below.
We will be demonstrating only one delimiter swapping. Assume that the delimiter is 3 then the swapping happens in the process shown below.

X	1	2	3	4	5	6	7	8	9	10
1	4	5	6	1	2	3	7	8	9	10
2	17	16	15	20	19	18	14	13	12	11
3	24	25	26	21	22	23	27	28	29	30
4	37	36	35	40	39	38	34	33	32	31
5	44	45	46	41	42	43	47	48	49	50
6	57	56	55	60	59	58	54	53	52	51
7	64	65	66	61	62	63	67	68	69	70
8	77	76	75	80	79	78	74	73	72	71
9	84	85	86	81	82	83	87	88	89	90
10	97	96	95	100	99	98	94	93	92	91

Figure 4: Result after swapping the first 3 columns with the next 3 columns



X	1	2	3	4	5	6	7	8	9	10
1	44	45	46	41	42	43	47	48	49	50
2	57	56	55	60	59	58	54	53	52	51
3	64	65	66	61	62	63	67	68	69	70
4	77	76	75	80	79	78	74	73	72	71
5	4	5	6	1	2	3	7	8	9	10
6	17	16	15	20	19	18	14	13	12	11
7	24	25	26	21	22	23	27	28	29	30
8	37	36	35	40	39	38	34	33	32	31
9	84	85	86	81	82	83	87	88	89	90
10	97	96	95	100	99	98	94	93	92	91

Figure 5: The result swapping the first 4 rows with the next 4 rows.

The process is continued for 10 rows in the same manner. Different delimiters are calculated for each of the rows. Decryption is the reverse of the process explained above. But the order of their appearance in the execution process changes according to the algorithm. Ten delimiters are calculated separately for rows and columns. The delimiters calculated decide on the size of the blocks to be swapped.

Least significant bit (LSB) insertion is a common, simple approach to embedding information in a cover image. The least significant bit (in other words, the 8th bit) of some or all of the bytes inside an image is changed to a bit of the secret message. When using a 24-bit image, a bit of each of the red, green and blue color components can be used, since they are each represented by a byte. In other words, one can store 3 bits in each pixel. An 800 × 600 pixel image, can thus store a total amount of 1,440,000 bits or 180,000 bytes of embedded data. For example a grid for 3 pixels of a 24-bit image can be as follows:

(00101101 00011100 11011100)
(10100110 11000100 00001100)
(11010010 10101101 01100011)

When the number 200, which binary representation is 11001000, is embedded into the least significant bits of this part of the image, the resulting grid is as follows:

(00101101 00011101 11011100)
(10100110 11000101 00001100)
(11010010 10101100 01100011)

Although the number was embedded into the first 8 bytes of the grid, only the 3 underlined bits needed to be changed according to the embedded message. On average, only half of the bits in an image will need to be modified to hide a secret message using the maximum cover size. Since there are 256 possible intensities of each primary color, changing the LSB of a pixel results in small changes in the intensity of the colors.

These changes cannot be perceived by the human eye - thus the message is successfully hidden. With a well-chosen image, one can even hide the message in the least as well as second to least significant bit and still not see the difference.

In the above example, consecutive bytes of the image data – from the first byte to the end of the message – are used to embed the information. This approach is very easy to detect. A slightly more secure system is for the sender and receiver to share a secret key that specifies only certain pixels to be changed. Should an adversary suspect that LSB steganography has been used, he has no way of knowing which pixels to target without the secret key.

In its simplest form, LSB makes use of BMP images, since they use lossless compression. Unfortunately to be able to hide a secret message inside a BMP file, one would require a very large cover image. Nowadays, BMP images of 800 × 600 pixels are not often used on the Internet and might arouse suspicion. For this reason, LSB steganography has also been developed for use with other image file formats.

III. MAJOR FLOW DIAGRAM

It explains the output of the various modules and the form of the image in various modules. The Embed module does the entire operation of encrypting and hiding the image in other image. The entire module is subdivided into two sub-modules. The two sub-modules are Encryption and the Hide sub-modules.

The first sub-module in the module is Encryption module. This takes up the job of encrypting the image as per the algorithm explained the further sections. The module takes up the input images JPEG form. The confidential image is taken as the input to this module. The output of this module is the encrypted image. The encrypted image is in JPEG format.

The encrypted image is fed as input to the next sub-module which is Hide module. The module takes two images as input. One is the encrypted image which comes as output from Encryption module and the other is the cover image provided by the user. The encrypted image is hidden is the cover image. The steganographic method is explained in the preceding sections.

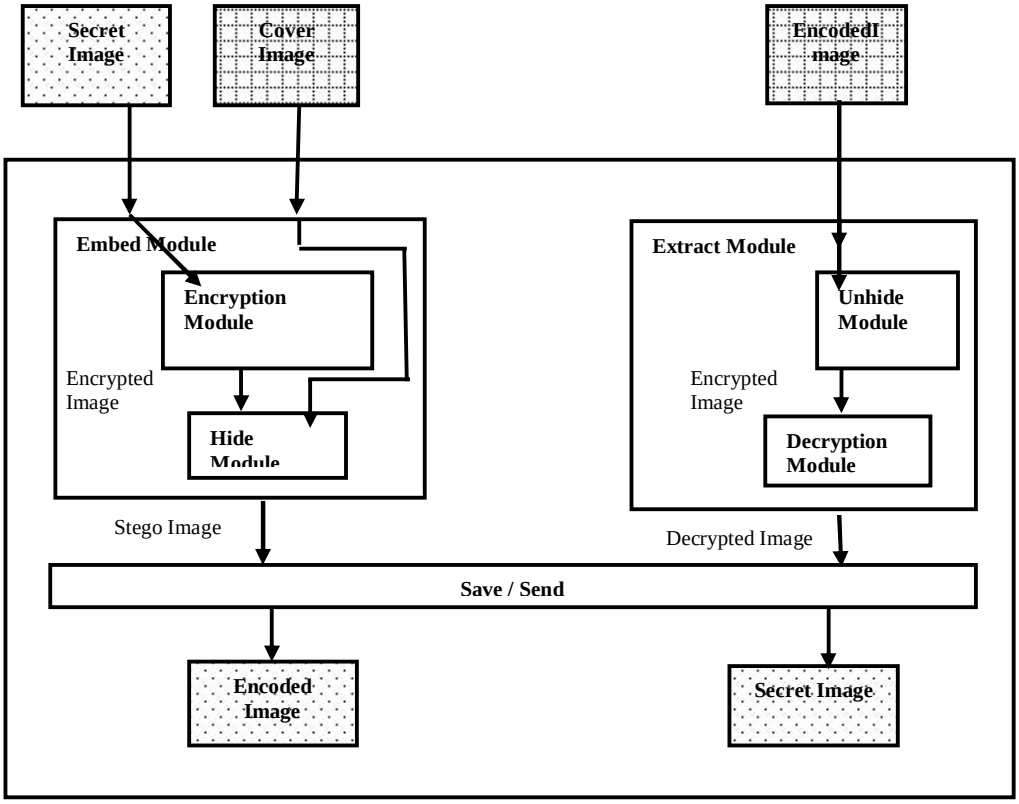


Figure 6: Architecture

The next module is the Extract module. This module is to extract the secret image from the encoded image. The encoded image is taken as input from the user and is fed into the extract module. There are two sub-modules in the extract module. One is the Unhide sub-module and the other is decrypt sub-module

IV. IMPLEMENTATION DETAILS

We will discuss about the working of the concept along with the algorithms and the pseudo code of the certain major parts are given and explained.

Algorithm: Encryption

Input: An image in JPEG format.
Step 1: Take the input image and convert it to a matrix of pixels.
Step 2: Reverse the entire matrix of pixels. The reversal is done column wise.
Step 3: Reverse only the odd rows of the matrix.
Step 4: Calculate the 10 “delimiter” values from the array of 10 keys for columns.
Step 5: Using the delimiter as size of a block of pixels, swap the column of pixels with the next block of columns of same size.
Step 6: Repeat the step 5 for every 10 delimiter calculated.

Step 7: Calculate the 10 “delimiter” values from the array of 10 keys for rows.
Step 8: Using the delimiter as size of a block of pixels, swap the rows of pixels with the next block of rows of same size.
Step 9: Repeat the step 8 for every 10 delimiter calculated.
Step 10: Reconstruct the image from the buffered image.

Some of the important codes are given now. First of all the image is decomposed into a blocks of cells. For this the cell width and height is to be calculated which is calculated as follows.

```
$cell_w = ceil($factor * $w);
$cell_h = ceil($factor * $h);
```

Here the term factor has the value 0.10. The “ceil” function returns the top limit value of the fraction or decimal value. For example it returns 6 if the value is between 5 and 6.

For reversing the entire matrix of pixels we have a built-in function called “array_reverse”. The use of the function in the project is given below.

```
for ($i = 0; $i < count($cell_img_array); $i++) {
 $cell_img_array[$i] = array_reverse($cell_img_array[$i]);
}
```

In the same manner we will be reverse only the odd rows of the matrix of pixels.

Further the calculation of the delimiter for rows and columns is given below. The delimiter identifies the end of a block of rows of columns. The block of columns is swapped with the block coming after the delimiter value.

```
$delim1 = round($keys[0] * count($cell_img_array));
$delim2 = round($keys[1] * count($cell_img_array));
.
.
.
$delim10 = round($keys[9] * count($cell_img_array));
```

The round function returns the rounded value of the decimal value, for example it returns a value 6 if the value is between 5.5 and 6.0 and value 5 if the decimal is between 5.0 and 5.5. The count function gives the count of the number of cells or pixels in the entire matrix.After calculating the delimiters the next job is to swap the columns and rows accordingly. The delimiter identifies the end of a block of rows of columns. The block of columns is swapped with the block coming after the delimiter value.

The code below is the code for swapping the columns .

```
for ($j = 0; $j < count($cell_img_array[0])-$delim9; $j++)
 {
          if ($j % 2 != 0)
            {
                for ($i = 0; $i < count($cell_img_array); $i++)
                  {
                        $tmp = $cell_img_array[$i][$j];
                        $cell_img_array[$i][$j]=$cell_img_array[$i][$j+$delim9];
                        $cell_img_array[$i][$j + $delim9] = $tmp;
                  }
            }
}
```

The code below is the code for swapping the columns.

```
for ($i = 0; $i < count($cell_img_array) - $delim1; $i++) {
  if ($i % 2 != 0) {
          $tmp = $cell_img_array[$i];
          $cell_img_array[$i] = $cell_img_array[$i + $delim1];
          $cell_img_array[$i + $delim1] = $tmp;
   }
}
```

After completing the procedure explained above we have to store the results in an image using the function image reconstruct as shown below.

```
$img = reconstruct($cell_img_array);
```

Before explaining the code we will give brief information of the algorithm used for encryption and then explain the code.

Algorithm: Decryption

Input: An image in JPEG format.
Step 1: Take the input image and convert it to a matrix of pixels.
Step 2: Calculate the 10 "delimiter" values from the array of 10 keys for rows.
Step 3: Using the delimiter as size of a block of pixels, swap the rows of pixels with the next block of rows of same size.
Step 4: Repeat the step 3 for every 10 delimiter calculated.
Step 5: Calculate the 10 "delimiter" values from the array of 10 keys for columns.
Step 6: Using the delimiter as size of a block of pixels, swap the column of pixels with the next block of columns of same size.
Step 7: Repeat the step 6 for every 10 delimiter calculated.
Step 8: Reverse only the odd rows of the matrix.
Step 9: Reverse the entire matrix of pixels. The reversal is done column wise.
Step 10: Reconstruct the image from the buffered image.

Decryption is the reverse process of encryption. The blocks of codes explained in the encryption part are used in the decryption. But the order of their appearance in the execution process changes according to the algorithm. Ten delimiters are calculated separately for rows and columns. The delimiters calculated decide on the size of the blocks to be swapped.

Algorithm: Hide

Input: Two images in JPEG format.
Step 1: Take the encrypted image and convert it to an array of bits.
Step 2: Obtain a boundary of 3 characters
Step 3: Check whether the secret image along with the boundary can fit in cover image.
Step 4: Convert the data into an array of true/false.
Step 5: Read the RGB values of the cover image.
Step 6: If the bit in data is 0
        set the cover image pixel to 0
    else
        set the cover image pixel to 1
Step 7: Repeat the steps 5 and 6 till the end of data bits.
Step 8: Reconstruct the image from the buffered image.

The input to the module is encrypted image and the cover image. First the encrypted image is converted to an array of data bits. A boundary is obtained using the code shown below. A boundary is an array of three characters such that it is not in the cover image.

```
do
 {
    $boundary = chr(rand(0,255)).chr(rand(0,255)).chr(rand(0,255)); } while(strpos($data,$boundary)!=false &&
strpos($hidefile['name'],$boundary)!==false);
```

The data bits are then appended with the boundary in order to identify the bits during extraction.

```
$data = $boundary.$hidefile['name'].$boundary.$data.$boundary;
```

After appending the boundary with the data bits we have to check whether the data bits fit into the cover image. For this the following code checks whether the size of data bits is less than the cover image size.

```
if(strlen($data)*8 > ($attributes[0]*$attributes[1])*3)
    {
        // remove images
        ImageDestroy($outpic);
        ImageDestroy($pic);
```

```
return "Cannot fit ".$hidefile['name']." in ".$maskfile['name'].".<br />".$hidefile['name']." requires mask to contain at least ".(intval((strlen($data)*8)/3)+1)." pixels.<br />Maximum filesize that ".$maskfile['name']." can hide is ".intval((($attributes[0]*$attributes[1])*3)/8)." bytes";
}
```

After checking for compatibility the next job is to fit the image in the cover image for which the below code is used.

```
for($j=0 ; $j<sizeof($cols) ; $j++)
        {
                if($make_odd[$i+$j]===true && is_even($cols[$j]))
                {
                        // is even, should be odd
                        $cols[$j]++;
                } else if($make_odd[$i+$j]===false && !is_even($cols[$j])){
                        // is odd, should be even
                        $cols[$j]--;
                } // else colour is fine as is
        }
```

We will explain the process of retrieving the secret image from the cover image.

Algorithm: Unhide

Input: Stego image
Step 1: Take the input image and convert it to an array of bits.
Step 2: Obtain the boundary of 3 characters from the first 8 pixels.
Step 3: Convert RGB of each pixel into binary
Step 4: Extract the last bit from the byte
Step 5: Repeat the steps 3 and 4 until you get the boundary.
Step 6: Reconstruct the image from the buffered image

In order to get the boundary from the image the below code is used. First it is set to null and then the 3 characters are retrieved from it.

```
$bin_boundary = "";
        for($x=0 ; $x<8 ; $x++) {
                $bin_boundary .= rgb2bin(ImageColorAt($pic, $x,0));
        }
```

V. CONCLUSION

To conclude we want to say that the image can be encrypted and hidden successfully in a cover image. The encoded image can be successfully saved in the same machine or sent to a machine in the same LAN. The encoded image can also be sent to a person's mail inbox successfully. The encoded image is decoded successfully

ACKNOWLEDGEMENT

Our most sincere thanks go to our advisor Dr.Yuvaraj B N, Professor, Dept. Of CSE, NIE, Mysore. We thank him for providing us an opportunity to work in the area of Cryptography and Steganography for securing confidential images. We thank him for consistent guidance, encouragement and support during initial development of this paper. He has been helping us to improve our English communication and writing skills.

REFERENCES

[1] Open LAMP 3rd Edition By James Lee.
[2] A Detailed Look at Steganographic Techniques and their Use By Bret Dunbar, Sans InfoSec Reading Room, Jan 2002
[3] Security Analysis of A Chaos-based Image Encryption Algorithm By Shiguo Lian, JinSheng Sun, Zhiquan Wang
[4] N. F. Johnson and S. Katzenbeisser, ``A survey of steganographic techniques", in S. Katzenbeisser and F. Peticolas (Eds.): Information Hiding, pp.43-78. Artech House, Norwood, MA, 2000.
[5] J. D. Gibbons and S. Chakraborti, Nonparametric statistical inference, Marcel Dekker, New York, 1992.
[6] L. Breiman, Probability, SIAM, Philadelphia, 1992.
[7] O. Dabeer, K. Sullivan, U. Madhow, S. Chandrasekharan, and B. S. Manjunath, ``Detection of hiding in the least significant bit", Proc. CISS, The Johns Hopkins University, Mar. 2003.
[8] http://www.gdlibrary.com
[9] http://www.jjtc.com/Security/stegtools.htm
[10] http://www.stego.com/howto.html