International Journal of Technical Research and Applications e-ISSN: 2320-8163,
www.ijtra.com Volume 3, Issue 2 (Mar-Apr 2015), PP. 28-32

EFFICIENT DATA HIDING SYSTEM USING LZW CRYPTOGRAPHY AND GIF IMAGE STEGANOGRAPHY

Intisar Majeed Saleh, Hanaa Hameed Merzah
AL. Rafidain University College
Baghdad, Iraq
Intisar_alyssiri@yahoo.com, Hanamerza2007@gmail.com

***Abstract*— The combination of steganography and cryptography is considered as one of the best security methods used for message protection, due to this reason, in this paper, a data hiding system that is based on image steganography and cryptography is proposed to secure data transfer between the source and destination. Animated GIF image is chosen as a carrier file format for the steganography due to a wide use in web pages and a LSB (Least Significant Bits) algorithm is employed to hide the message inside the colors of the pixels of an animated GIF image frames. To increase the security of hiding, each frame of GIF image is converted to 256 color BMP image and the palette of them is sorted and reassign each pixels to its new index, furthermore, the message is encrypted by LZW (Lempel _ Ziv_Welch) compression algorithm before being hidden in the image frames. The proposed system was evaluated for effectiveness and the result shows that, the encryption and decryption methods used for developing the system make the security of the proposed system more efficient in securing data from unauthorized users. The system is therefore, recommended to be used by the Internet users for establishing a more secure communication.**

***Keywords*:- Steganography, LSB, Animated GIF, LZW.**

I. INTRODUCTION

The cryptography and steganography techniques occupies a distinctive place in the area of security as they help to solve the problem of transmission of confidential information over the network in the form of a more secure against violations of unauthorized persons . Cryptography is the art and science of creating nonreadable data or cipher so that only intended person is only able to read the data[1], Unfortunately it is sometimes not enough to keep the contents of a message secret, it may also be necessary to keep the existence of the message secret. The technique used to implement this, is called steganography.Steganography comes from the Greek words Steganós (Covered) and Graptos (Writing), it usually refers to information or a file that has been concealed inside a digital Picture, Video, Text or Audio file[2,3]. For this reason, in this paper, we focus on the combination of steganography and cryptography to protect the private information within the digital image ,and more specifically on the GIF image which being used widely on the Internet. When information is hidden inside a carrier file, the data is encrypted with LZW compression method.

The general scheme for embedding data is depicted in Figure 1. A message is embedded in a file by the stegosystem encoder, which has as inputs the original cover, the secret message and a key. The resulting stego object is then transmitted over a communication channel to the recipient where the stegosystem decoder using the same key processes it and the message can be read.

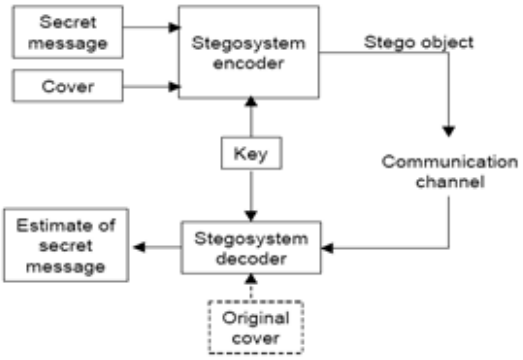


Fig. 1. **A General Steganographic Model**

The steganographic process can be represented using formulas. The stego object is given by:

$$I' = f(I,m,k)$$

where: I' is the stego object, I is the original object, m is the message and k is the key that the two parties share. The stego object may be subject to many distortions, which can be represented as a noise process n:

$$I'' = I'+n(I')$$

At the decoder we wish to extract the signal m, so we can consider the unwanted signal to be I. The embedded signal should resist common signal distortions as those depicted in Figure 2. Two kinds of compression exist: *lossy* and *lossless*. Both methods save storage space but have different results. Lossless compression permits exact reconstruction of the original message; therefore it is preferred when the original information must remain intact. Such compression schemes are the images saved as GIF (Graphic Interchange Format). Lossy compression, on the other hand, does not maintain the original's integrity. Such compression scheme is an image saved as JPEG (Joint Photographic Experts Group). The JPEG formats provide close approximations to high-quality digital photos but not an exact duplicate.

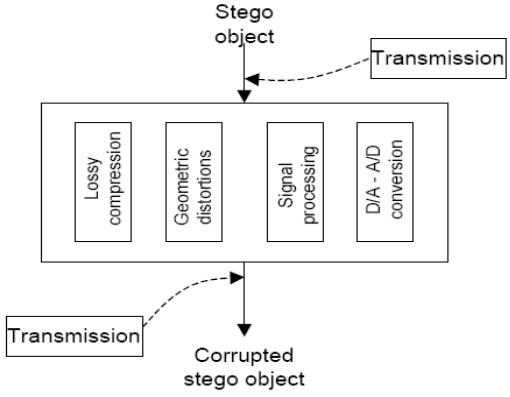


Fig. 2: common signal distortions over the transmission channel

Geometric distortions are specific to images and video and include operations such as: rotation, translation, scaling. Cropping is another type of geometric distortion that leads to irretrievable loss of data. The signal can be processed in many ways, including resampling, analog-to-digital and digital-to-analog conversions, requantization, recompression, printing, rescanning, copying, filtering and so on. All those processes introduce additional degradation into the object that a steganographic scheme must withstand.[4]

II. LEST SIGNIFICANT BIT (LSB)

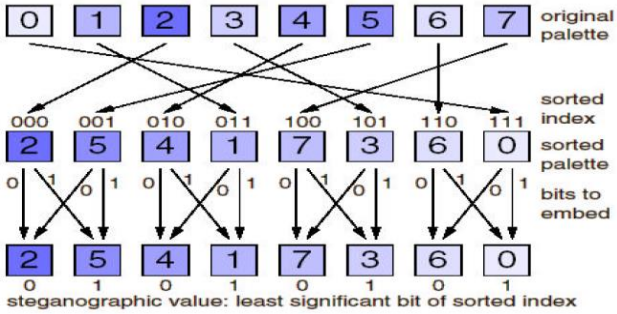
Least significant bit (LSB) insertion is a common, simple approach to embedding information in a cover image [5]. In this method, we can take the binary representation of the hidden_data and overwrite the LSB of each byte within the cover_image. If we are using 24-bit color, the amount of change will be minimal and indiscernible to the human eye. As an example, suppose that we have three adjacent pixels (nine bytes) with the following RGB encoding:

10010101 00001101 11001001
10010110 00001111 11001010
10011111 00010000 11001011

Now suppose we want to "hide" the following 9 bits of data (the hidden data is usually compressed prior to being hidden): 101101101. If we overlay thes 9 bits over the LSB of the 9 bytes above, we get the following (where bits in **bold** have been changed):

10010101 00001100 11001001
10010111 00001110 11001011
10011111 00010000 11001011

Note that we have successfully hidden 9 bits but at a cost of only changing 4, or roughly 50%, of the LSBs. Similar methods can be applied to 8-bit palette based images (like GIF images) but the changes, as the reader might imagine, are more dramatic[6]. This is alleviated in this paper by sorting the palette and reassigns each pixel to the index of its color in the new palette before the embedding process.



III. LEMPEL-ZIV-WELCH (LZW)

LZW is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978. The algorithm is designed to be fast to implement but is not usually optimal because it performs only limited analysis of the data.

The compressor algorithm builds a string translation table from the text being compressed. The string translation table maps fixed-length codes (usually 12-bit) to strings. The string table is initialized with all single-character strings (256 entries in the case of 8-bit characters). As the compressor character-serially examines the text, it stores every unique two-character string into the table as a code/character concatenation, with the code mapping to the corresponding first character. As each two-character string is stored, the first character is sent to the output. Whenever a previously-encountered string is read from the input, the longest such previously-encountered string is determined, and then the code for this string concatenated with the extension character (the next character in the input) is stored in the table. The code for this longest previously-encountered string is output and the extension character is used as the beginning of the next word.

The decompressor algorithm only requires the compressed text as an input, since it can build an identical string table from the compressed text as it is recreating the original text. However, an abnormal case shows up whenever the sequence *character*/*string*/*character*/*string*/*character* (with the same character for each *character* and string for each *string*) is

encountered in the input and *character*/*string* is already stored in the string table. When the decompressor reads the code for *character*/*string*/*character* in the input, it cannot resolve it because it has not yet stored this code in its table. This special case can be dealt with because the decompressor knows that the extension character is the previously-encountered *character*.

A. Compressor Algorithm

Build a table and store all possible strings in it
STRING = get input character
WHILE there are still input characters DO
  CHARACTER = get input character
  IF STRING+CHARACTER is in the string table then
    STRING = STRING+character
  ELSE
    output the code for STRING
    add STRING+CHARACTER to the string table
    STRING = CHARACTER
  END of IF
END of WHILE
output the code for STRING

B. Decompressor Algorithm

```
Build a table and store all possible strings in it
Read OLD_CODE
OLD_CODE = get translation of OLD_CODE
output OLD_CODE
CHARACTER = OLD_CODE
WHILE there are still input characters DO
    Read NEW_CODE
    IF NEW_CODE is not in the string table THEN
       STRING = OLD_CODE
      STRING = STRING+CHARACTER
    ELSE
       STRING = get translation of NEW_CODE
    END of IF
    output STRING
    CHARACTER = first character in STRING
    add OLD_CODE + CHARACTER to the string table
  OLD_CODE = get translation of NEW_CODE
END of WHILE
```

IV. Graphic Interchange Format (GIF)

The Graphics Interchange Format (GIF) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability. The format supports up to 8 bits per pixel, allowing a single image to reference a palette of up to 256 distinct colors chosen from the 24-bit RGB color space. It also supports animations and allows a separate palette of 256 colors for each frame. GIF images are compressed using the Lempel-Ziv-Welch (LZW) lossless data compression technique to reduce the file size without degrading the visual quality.

A. General File Format

- The **GIF Signature** identifies the data following as a valid GIF image stream. It consists of the following six characters. The first three characters are G I F and the last three characters are a version number for this particular GIF definition.
- The **Screen Descriptor** describes the overall parameters for all GIF images following. This information is stored in a series of 8-bit bytes as described below.

```
      bits
  7 6 5 4 3 2 1 0   Byte #
 +-----------------+
 |                 | 1
 +-Screen Width -+     Raster width in pixels (LSB first)
 |               | 2
 +---------------+
 |               | 3
 +-Screen Height-+     Raster height in pixels (LSB first)
 |               | 4
 +-+-----+-+-----+     M = 1, Global color map follows Descriptor
 |M| cr  |0|pixe l | 5   cr+1 = # bits of color resolution
 +-+-----+-+-----+     pixel+1 = # bits/pixel in image
 |  background     | 6   background=Color index of screen background
 +-----------------+     (color is defined from the Global color
 |0 0 0 0 0 0 0 0 | 7    map or default map if none specified)
 +-----------------+
```

- The **Global Color Map** is optional but recommended for images where accurate color rendition is desired. The existence of this color map is indicated in the 'M' field of byte 5 of the Screen Descriptor. The number of color map entries following a Screen Descriptor is equal to 2**(# bits per pixel), where each entry consists of three byte values representing the relative intensities of red, green and blue respectively. The structure of the Color Map block is:

```
    bits
 7 6 5 4 3 2 1 0  Byte #
+---------------+
| red intensity |  1    Red value for color index 0
+---------------+
|green intensity|  2    Green value for color index 0
+---------------+
| blue intensity|  3    Blue value for color index 0
+---------------+
| red intensity |  4    Red value for color index 1
+---------------+
|green intensity|  5    Green value for color index 1
+---------------+
| blue intensity|  6    Blue value for color index 1
+---------------+
:               :      (Continues for remaining colors)
```

- The **Image Descriptor** defines the actual placement and extents of the following image within the space defined in the Screen Descriptor. Also defined are flags to indicate the presence of a local color lookup map, and to define the pixel display sequence. The information stored in image descriptor is:

```
    bits
 7 6 5 4 3 2 1 0  Byte #
+---------------+
|0 0 1 0 1 1 0 0|  1    ',' - Image separator character
+---------------+
|               |  2    Start of image in pixels from the
+-  Image Left -+       left side of the screen (LSB first)
|               |  3
+---------------+
|               |  4
+-  Image Top  -+       Start of image in pixels from the
|               |  5    top of the screen (LSB first)
+---------------+
|               |  6
+- Image Width -+       Width of the image in pixels (LSB first)
|               |  7
+---------------+
|               |  8
+- Image Height-+       Height of the image in pixels (LSB first)
|               |  9
+-+-+-+-+-+-+-----+     M=0 - Use global color map, ignore 'pixel'
|M|I|0|0|0|pixel| 10    M=1 - Local color map follows, use 'pixel'
+-+-+-+-+-+-+-----+     I=0 - Image formatted in Sequential order
                        I=1 - Image formatted in Interlaced order
                        pixel+1 - # bits per pixel for this image
```

follows the Image Descriptor that applies only to the following image. At the end of the image, the color map will revert to that defined after the Screen Descriptor. Note that the 'pixel' field of byte 10 of the Image Descriptor is used only if a Local Color Map is indicated. This defines the parameters not only for the image pixel size, but determines the number of color map entries that follow. The bits per pixel value will also revert to the value specified in the Screen Descriptor when processing of the image is complete.

- **Raster Data:** The format of the actual image is defined as the series of pixel color index values that

make up the image. The image pixel values are processed as a series of color indices which map into the existing color map. The resulting color value from the map is what is actually displayed. This series of pixel indices, the number of which is equal to image-width*image-height pixels, are passed to the GIF image data stream one value per pixel, compressed and packaged according to a version of the LZW compression algorithm.

V. The Working System Algorithms

a) The Hiding Algorithm

Input: The stego-image represented by animated GIF image.
Output: The text message.
Step1: extract all frames from the stego-image (animated GIF) and convert them to a 256-color BMP images.
Step2: sort the palette of all BMP images and reassign each pixel to its new color index.
Step3: for i=1 to no. of bits in the LZW text code.
Step4: for j=1 to no. of pixels in each frame.
Step5: for k=1 to no. of image frames.
Step6: extract the LZW code bit from the current pixel by using LSB algorithm.
Step7: next k.
Step8: next j.
Step9: next i.
Step10: decompress the LZW text code by using LZW decompressor algorithm.

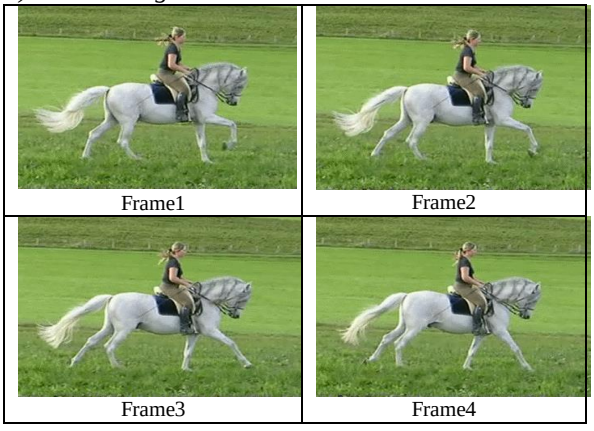
b) The Extracting Algorithm

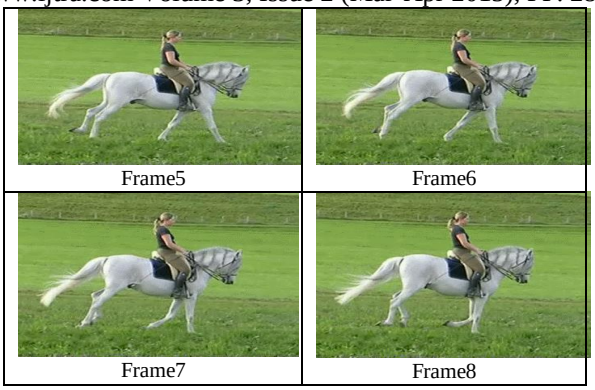
Input: The text message and the cover image represented by animated GIF image.
Output: The stego- image represented by animated GIF image.
Step1: read the text message.
Step2: compress the message by using LZW compressor algorithm.
Step3: extract all frames from the cover image (animated GIF) and convert them to a 256-color BMP images.
Step4: sort the palette of all BMP images and reassign each pixel to its new color index.
Step5: for i=1 to no. of bits in the LZW text code.
Step6: for j=1 to no. of pixels in each frame.
Step7: for k=1 to no. of image frames.
Step8: hide the LZW code bit in the current pixel by using LSB algorithm.
Step9: next k.
Step10: next j.
Step11: next i.
Step12: return the original palette of the frames and reassign each pixel to its new color index.
Step13: covert the BMP images to an animated GIF image (stego-image).

VI. Practical Implementation

We have completed a project using this technique using Visual Basic 6.0. Various animated GIF images are used as source image and the results are noted. Let us see one of the results here:

a) Cover Image Frames





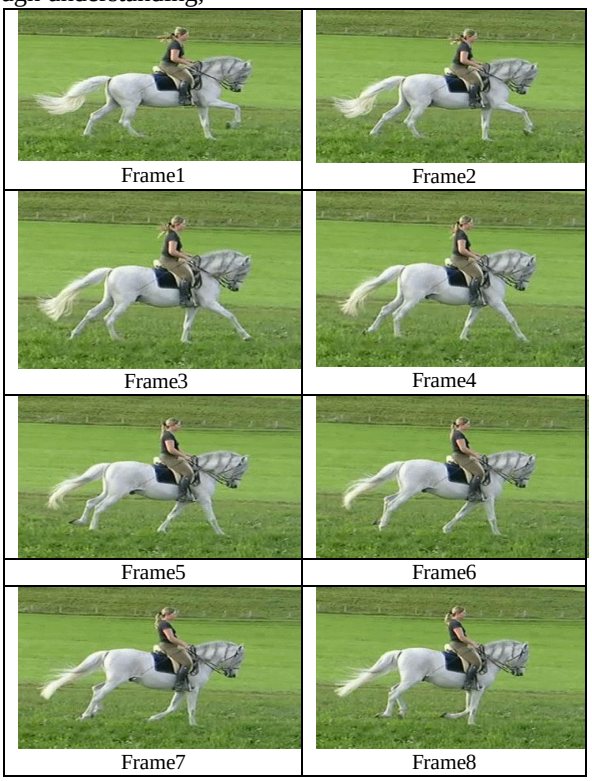
b) The Text Message

You give but little when you give of your possessions. It is when you give of yourself that you truly give. For what are your possessions but things you keep and guard for fear you may need them tomorrow? And tomorrow, what shall tomorrow bring to the over prudent dog burying bones in the trackless sand as he follows the pilgrims to the holy city? And what is fear of need but need itself? Is not dread of thirst when your well is full, the thirst that is unquenchable?

There are those who give little of the much which they have--and they give it for recognition and their hidden desire makes their gifts unwholesome. And there are those who have little and give it all. These are the believers in life and the bounty of life, and their coffer is never empty. There are those who give with joy, and that joy is their reward. And there are those who give with pain, and that pain is their baptism.

And there are those who give and know not pain in giving, nor do they seek joy, nor give with mindfulness of virtue; They give as in yonder valley the myrtle breathes its fragrance into space. Through the hands of such as these God speaks, and from behind their eyes He smiles upon the earth.

It is well to give when asked, but it is better to give unasked, through understanding;

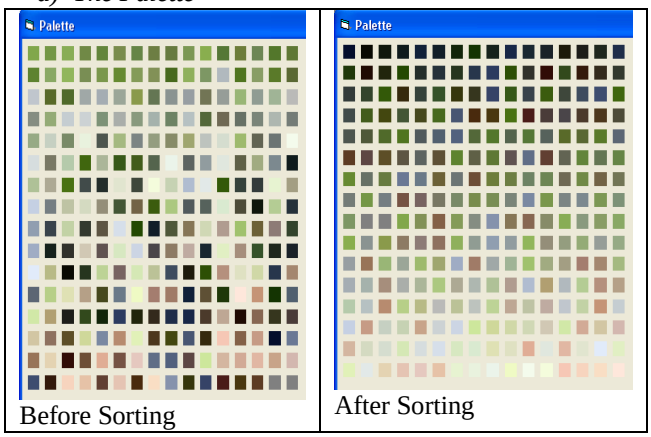


nd to the open-handed the search for one who shall receive is joy greater than giving. And is there aught you

would withhold? All you have shall some day be given; therefore give now, that the season of giving may be yours and not your inheritors'

c) Stego-Image Frames

d) The Palette



Before Sorting | After Sorting

VII. Conclusions

1- The processing of compressed a text message by using LZW compression method considered as an encryption method, therefore the detection of the hidden message become more complex.

2- Hide the text in all frames of the animated image and the animation property of the image make the observation of the hidden text very difficult.

3- The maximum size of the embedded text message could be very huge depend on the LZW code of the text and number of frames in the image.

4- The lossless compression method used in the image and in the embedded text result to extract the text without any changes in the message.

References

[1] Atul Kahate (2009), Cryptography and Network Security, 2nd edition, McGraw-Hill.

[2] Mohamed Elsading Eltahir, Laiha Mat, B. B Zaidan and A. A. Zaidan," High Rate Video Streaming Steganography" ,International Conference on Information Management and Engineering(ICIME09) Session 10,P.P 550-553,2009.(Conference proceding).

[3] Fazida Othman, Miss Laiha. Maktom, A.Y.TAQA, B. B Zaidan, A. A Zaidan,"An Extensive Empirical Study for the impact of Increasing Data Hidden on the Images Texture",Internatioal Conference on future Computer and Communication(ICFCC 09),Session 7,P.P 477-481,2009(Confernce proceding).

[4] Richard Popa "An Analysis of Steganographic Techniques".

[5] Johnson, N.F. & Jajodia, S., "Exploring Steganography: Seeing the Unseen",Computer Journal,February 1998.

[6] Murshed, Md. M. "Steganography using LSB hiding "Upper lowa University Fayette, lowa, USA.

[7] "Graphics Interchange format, Version 87a". 15 June 1987.Retrieved13 October 2012.