

## **Steganography (Encoding text and Hiding it within image)**

*Asim M. Murshid  
College of Science - University of Kirkuk*

### **Abstract**

The simple abstract we can talk about in this project is about the idea of information security has become very important thing in our life, so we can not depend only on data or information encryption but hiding it too which is called (Steganography). So it is a method in which we hide an information inside an other so the real information or data can not be seen. The information we want to hide in this project is a text message to be hidden in a picture which is a BMP (Bit Map) format which is simple. We will use the programming language (Visual Basic) to do this job, Hiding a text message in a picture needs on the other hand getting the image back from the picture so the program will contain this operation too. The data of the BMP picture will be replaced by the hidden text message so it may cause distortion in that place therefore the message must be distributed all over the picture so the distortion would be reduced. In the future we can hide another type of data inside another type of carrier (out world data) such as hiding text inside a voice file.

### **Introduction**

Steganography conceals the fact that a message is being sent. It is a method akin to covert channels, spread spectrum communication and invisible inks which adds another step in security. A message in ciphertext may arouse suspicion while an invisible message will not. This paper introduces steganography by explaining what it is, providing a brief history with illustrations of some methods for implementing steganography, and comparing available software providing steganographic services. Though the forms are many, the focus of the software evaluation in this paper is on the use of images in steganography.

### **Structure of Paper:**

I will define in section 2 the steganography, provide a brief history, and explain various methods of steganography. Then I will review in section 2 a several software applications that provide steganographic services and mention the approaches taken. And in Section 4 I will conclude with a brief discussion of the implications of steganographic technology. Section 5 will list the resources used in researching this topic and additional readings for those interested in more in-depth understanding of steganography.

## **Steganography**

### **Definition**

The word steganography literally means covered writing as derived from Greek. It includes a vast array of methods of secret communications that conceal the very existence of the message. Among these methods are invisible inks, microdots, character arrangement (other than the cryptographic methods of permutation and substitution), digital signatures, covert channels and spread-spectrum communications. Steganography is the art of concealing the existence of information within seemingly innocuous carriers. Steganography can be viewed as akin to cryptography. Both have been used throughout recorded history as means to protect information. At times these two technologies seem to converge while the objectives of the two differ. Cryptographic techniques (scramble) messages so if intercepted, the messages cannot be understood. Steganography, in an essence, (camouflages) a message to hide its existence and make it seem (invisible) thus concealing the fact that a message is being sent altogether. An encrypted message may draw suspicion while an invisible message will not (JDJ01). David Kahn places steganography and cryptography in a table to differentiate against the types and counter methods used. Here security is defined as methods of (protecting) information where intelligence is defined as methods of (retrieving) information.

Signal Security	Signal Intelligence
Communication Security	Communication Intelligence
Steganography (invisible inks, open codes, messages in hollow heels) and Transmission Security (spurt radio and spread spectrum systems)	Interception and direction-finding
Cryptography (codes and ciphers)	Cryptanalysis
Traffic security (call-sign changes, dummy messages, radio silence)	Traffic analysis (direction-finding, message -flow studies, radio finger printing)
Electronic Security	Electronic Intelligence
Emission Security (shifting of radar frequencies, spread spectrum)	Electronic Reconnaissance (eaves-dropping on radar emissions)
Counter-Countermeasures "looking through" (jammed radar)	Countermeasures (jamming radar and false radar echoes)

Table 1: Kahn's Security Table

Steganography has its place in security. It is not intended to replace cryptography but supplement it. Hiding a message with steganography methods reduces the chance of a message being detected. However, if that message is also encrypted, if discovered, it must also be cracked (yet another layer of protection).

## **PC Software that Provide Steganographic Services**

### **Background:**

Steganographic software is new and very effective. Such software enables information to be hidden in graphic, sound and apparently "blank" media. Charles Kurak and John McHugh discuss the implications of downgrading an image (security downgrading) when it may contain some other information. Though not explicitly stated the author(s) of StegoDos mention embedding viruses in images. In the computer, an image is an array of numbers that represent light intensities at various points (pixels) in the image. A common image size is 640 by 480 and 256 colors (or 8 bits per pixel). Such an image could contain about 300 kilobits of data. There are usually two types of files used when embedding data into an image. The innocent looking image which will hold the hidden information is a (container). A (message) is the information to be hidden. A message may be plain-text, ciphertext, other images or any thing that can be embedded in the least significant bits (LSB) of an image. For example: Suppose we have a 24-bit image 1024 x 768 (this is a common resolution for satellite images, electronic astral photographs and other high resolution graphics). This may produce a file over 2 megabytes in size ( $1024 \times 768 \times 24 / 8 = 2,359,296$  bytes). All color variations are derived from three primary colors, Red, Green and Blue. Each primary color is represented by 1 byte (8 bits). 24-bit images use 3 bytes per pixel. If information is stored in the least significant bit (LSB) of each byte, 3 bits can be stored in each pixel. The (container) image will look identical to the human eye, even if viewing the picture side by side with the original. Unfortunately, 24-bit images are uncommon (with exception of the formats mentioned earlier) and quite large. They would draw attention to themselves when being transmitted across a network. Compression would be beneficial if not necessary to transmit such a file. But file compression may interfere with the storage of information. Kurak and McHugh identify two kinds of compression, lossless and lossy. Both methods save storage space but may present different results when the information is uncompressed.

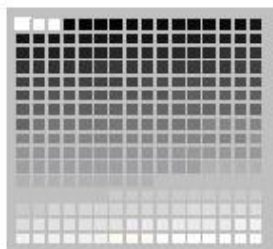
- Lossless compression is preferred when there is a requirement that the original information remain intact (as with steganographic images). The original message can be reconstructed exactly. This type of compression is typical in GIF2 and BMP3 images.

- Lossy compression, while also saving space, may not maintain the integrity of the original image. This method is typical in JPG4 images and yields very good compression.


To illustrate the advantage of lossy compression, Renoir's Le Moulin de la Galette was retrieved as a 175,808 byte JPG image 1073 x 790 pixels with 16 million possible colors. The colors were maintained when converting it to a 24-bit BMP file but the file size became 2,649,019 bytes! Converting again to a GIF file, the colors were reduced to 256 colors(8-bit) and the new file is 775,252 bytes. The 256 color image is a very good approximation of Renoir's painting.

1. Graphic Interchange Format developed by CompuServe to be a device – independent method of storing images.
2. Windows and OS/2 bitmap picture file.
3. Joint Photography experts Group (JPG/JPEG) is a device-independent method for storing images which supports 24-bit images.

Most steganographic software available does not support, nor recommends, using JPG files (an exception is noted later in the paper). The next best alternative to 24-bit images, is to use 256 color (or gray-scale) images. These are the most common images found on the Internet in the form of GIF files. Each pixel is represented as a byte (8-bits). Many authors of the steganography software and articles stress the use of gray-scale images (those with 256 shades of gray or better). The importance is not whether the image is gray-scale or not, the importance is the degree to which the colors change between bit values. Gray-scale images are very good because the shades gradually change from byte to byte. The following is a palette containing 256 shades of gray.



Fig(1): gray scale palettes with 256 scales

A similar image with 16 shades of gray (four-bit color) may look very close to one with 256 shades of gray  but the palette has less variations with which to work. The subtleties permit data to be stored without the human eye catching the changes. Many argue that gray-scale images render the "best" results for steganography. However, using gray-scale or color is not as important as the subtleties in color variation. Consider the following two 256 color palettes.



Fig(2): 4 bits scale palettes with 256 color scales

### **Encoding data and hiding it with LSB method:**

As the defining of the algorithm's name, it deals with the least Value (from all the 8 bits) in the byte. Changing the last bit data, and replacing it with our message data, can be done by this algorithm:

- 1- Looping until our message ends.
- 2- Encoding our secret data, to be encoded secret data (by using one of the encoding methods, using (encoding + hiding) methods, get the most highest protection on my translating information in any translating data media.
- 3- Disassembling the covering data, in singular bytes, and disassembling the encoded secret data in same way.
- 4- Erase (zeroing) the data in the last bit of each cover data's byte.
- 5- Adding the message bit value in it.
- 6- Assembling the message data bytes again, with the encoded secret data.
- 7- Surly, the opposite way, is to extracting (separating) the encoding secret data.
- 8- And apply the opposite encoding method to decode the original (source) data.

Now I'll explain these steps as programming codes in Visual basic language, and here is the code of encoding message: Dim key As Byte  
Private Sub Form\_Load()  
Dim lowupcase as byte

```

key = InputBox(" what is the key ? ", "Key Cipher")
End Sub
Private Sub secret_text_KeyPress(KeyAscii As Integer)
If uppercase (secret_text.text) then lowupcase=64
Else: lowupcase=96
End if
embedded_text = embedded_text & Chr(((key + KeyAscii -
lowupcase) Mod 26) + lowupcase)
End Sub

```

After encoding it, we shall hiding it by using the LSB method, the last significant bit is the last (form the right), see this shape:

2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>
128	64	32	16	8	4	2	1

Most significant bit least significant bit

It was the simplest code of a simplest method of encoding message by using the Cesar addition algorithm.

And this example:

If we have this character (B), we will convert it to its equal ASCII number (66), and add it to our key cipher (let it be 30),

$$66-65=1;$$

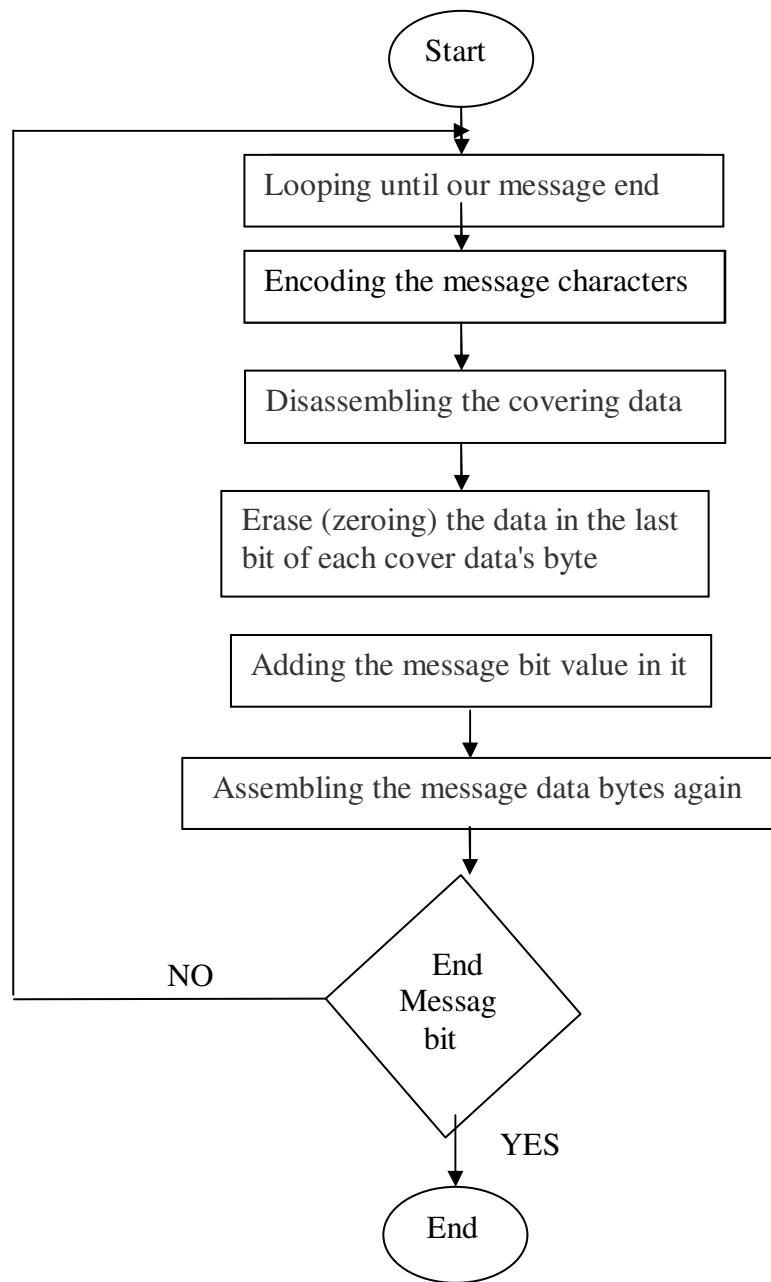
$$1+30=31;$$

$$.31 \text{ mod } 26 =5$$

$$c+65=70;$$

$$\text{char}(70)=\text{F};$$

and then we will apply the LSB on the character F, as a binary implementation as 010000101, and disassembling it to 8 bit, and we hide each bit in one byte of the cover image data. That means we will need a cover data's size at least 8 times more than the secret data. The follow flow chart shows you all the process of encoding and hiding data in another data:



Flow chart

**Evaluation Method:**

Various steganographic software packages were explored. The evaluation process was to determine limitations and flexibility of the software readily available to the public. Message and container files were selected before testing. This proved to be a problem with some packages due to limitations of the software. The images selected had to be altered to fit into the

constraints of the software and other containers were used. In all, a total of 25 files were used as containers (much more than I have room to discuss). The files used for evaluation included two (message) files and two (container) files. The (message) files are those to be hidden in the innocent looking (container) files. Message 1 contains the following plain-text and will be referred to as M1:

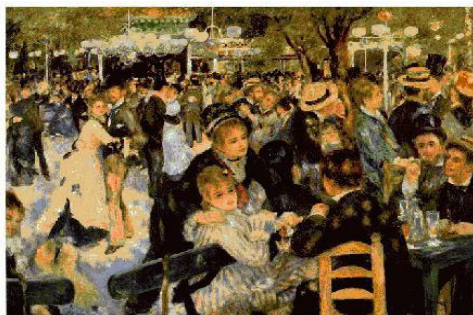
*Steganography is the art and science of communicating in a way which hides the existence of the communication.*

In contrast to cryptography, where the (enemy) is allowed to detect, intercept and modify messages without being able to violate certain security premises guaranteed by a cryptosystem, the goal of steganography is to hide messages inside other (harmless) messages in a way that does not allow any (enemy) to even detect that there is a second secret message present. Message 2 is a satellite image which will be referred to as M2:



Fig (3): 4 bits scale palettes with 256 color scales

The satellite photograph is of a major Soviet strategic bomber base near Dolon, Kazakhstan taken August 20, 1966. An Executive Order, signed by President Clinton on 23 February 1995, has authorized the declassification of satellite photographs collected by the U.S. intelligence community during the 1960's.



Figure(4): Renoir's Le Moulin de la Galette - Container C16



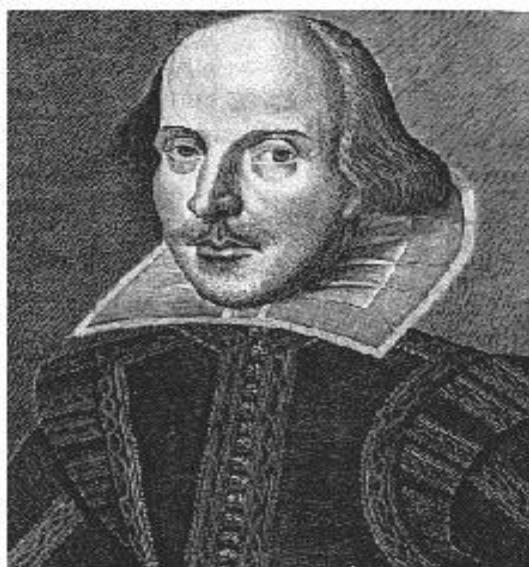


Fig (5): Droeshout engraving of William Shakespeare - Container C27

The image of Shakespeare is too small to contain M2, but M1 could be embedded without any degradation of the image. For the most part, all the software tested could handle the 518 byte plain- text message, however, only two could handle the image labeled M2. Of the two, only one software package could reliably handle 24-bit images and other formats consistently: S-Tools by Andy Brown. Next, an attempt was made to embed messages M1 and M2 using each software package. If the software could not handle processing these containers (C1 and C2), other containers were tried. All the software could embed M1 into some container. These files were reviewed before and after applying steganographic methods.

### **Software Evaluation**

The following software packages were reviewed with respect to steganographic manipulation of images: Hide and Seek v4.1,Stego Dos v0.90a,White Noise Storm, and S-Tools for Windows v3.00. Nearly all the authors encourage encrypting messages before embedding them in images as an added layer of protection and reviewing the images after embedding data.Even with the most reliable software tested, there may be some unexpected results.

#### Hide and Seek v 4.1:

Hide and Seek versions 4.1 and 5.0 by Colin Maroney have similar limitations with minimum image sizes(320 x 480). In version 4.1 if the image is smaller than the minimum, then the stego-image is padded with black space. If the cover image is larger, the stego-image is cropped to fit.

In version 5.0 the same is true with minimum image sizes. If any image exceeds 1024 x 768, an error message is returned. The Hide and Seek 1.0 for Windows 95 version seems to have these issues resolved and is a much improved steganography tool. Version 4.1 is evaluated here to illustrate limitations of some steganography tools. Hide and Seek 4.1 is free software which contains a series of DOS programs that embed data in GIF files and comes with the source code. Hide and Seek uses the Least Significant Bit of each pixel to encode characters, 8 pixels per character and spreads the data throughout the GIF in a somewhat random fashion. The larger the message the more likely the resulting image will be degraded. Since the data is dispersed "randomly" and the message file header is encrypted, there is no telling what is in an embedded file. Unfortunately the hidden file can be no longer than 19,000 bytes because the maximum display used is 320x480 pixels. Each character takes 8 pixels to hide  $((320 \times 480) / 8 = 19200)$ . C2 (Shakespeare) was used to embed M1. The original image of Shakespeare is 222 x 282 pixels and 256 shades of gray. The resulting image was forced to 320 x 480 pixels. Instead of "stretching" the image to fit, large black areas were added to the image making it 320 x 480. The image on the left is the original C2 and the image on the right is embedded with M1.

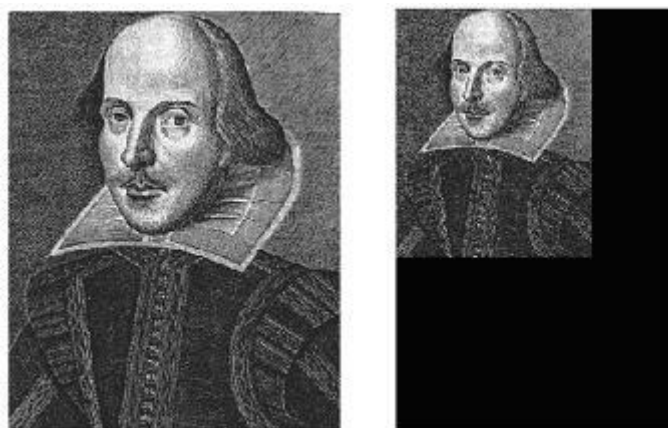


Fig. (6) : Result of using Hide and seek for embedding M1 into C2

### Stego Dos

Stego Dos is also known as Black Wolf's Picture Encoder version 0.90a. This is Public Domain software written by Black Wolf (anonymous). This is a series of DOS programs that require far too much effort for the results. It will only work with 320x200 images with 256 colors. To encode a message, one must:

- Run gesscr. This starts a TSR which will perform a screen capture when printscreen is pressed.
- View the image with a third-party image viewing software (not included with StegoDos) and press printscreen to save the image in message.scr.
- Save your message to be embedded in the image as message.dat.
- Run encode. This will merge message. dat with message.scr.
- Use a third party screen capturing program (not included with StegoDos) to capture the new image from the screen.
- Run putscr and capture the image displayed on the screen.

Decoding the message is not as involved but still requires a third party program to view the image. To decode a message, one must:

- Run getscr. This starts a TSR which will perform a screen capture when printscreen is pressed.
- View the image containing a message with a third-party image viewing software (not included with StegoDos) and press printscreen to save the image in message.scr.
- Run decode. This will extract the stored message from message.scr.

Due to the size restrictions, M2 and C1 could not be used. C2 (Shakespeare) and a number of other containers were tested (both color and gray-scale) with M1. Every one of them were obviously distorted. There was little distortion within the C2 image, but it was cropped and fitted into a 320 x 200 pixel image. The image on the left is the original C2 file. The image on the right contains the M1 message:

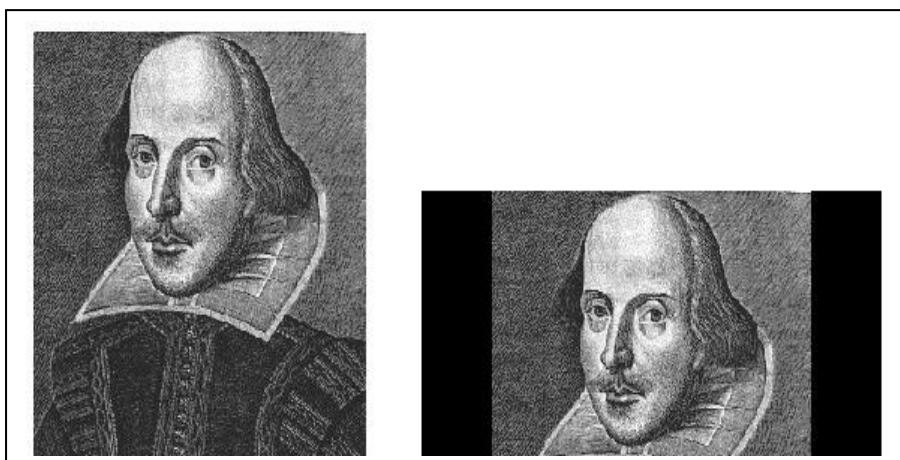


Fig.(7): Result of embedding M1 in C2 with StegoDos

This application uses the Least Significant Bit method with less success than the others. It also appends an EOF (end of file) character to the end of the message. Even with the EOF character, the message retrieved from the altered image most likely contained garbage at the end. The following is the original message (M1) and a portion of the message extracted from the image created with StegoDos: Steganography is the art and science of communicating in a way which hides the existence of the communication. In contrast to cryptography where the (enemy) is allowed to detect, intercept and modify messages without being able to violate certain security premises guaranteed by a cryptosystem, the goal of steganography is to hide messages inside other (harmless) messages in a way that does not allow any "enemy" to even detect that there is a second secret message present. The original file is 518 bytes. The extracted file is around 8 kilobytes: Steganography is the art and science of communicating in a way which hides the existence of the communication. In contrast to cryptography, where the (enemy) is allowed to detect, intercept and modify messages without being able to violate certain security premises guaranteed by a cryptosystem, the goal of steganography is to hide messages inside other (harmless) messages in a way that does not allow any (enemy) to even detect that there is a second secret message present.

#### White Noise Storm

White Noise Storm by Ray (Arsen) Arachelian is a very versatile steganography application for DOS. Embedding M1 in the containers C1 and C2 was rather trivial and no degradation could be detected. White Noise Storm was the first software tested that could embed M2 into C1- notice the (noise) interfering with the image integrity. The image on the left is the original C2. The image on the right contains message M1:

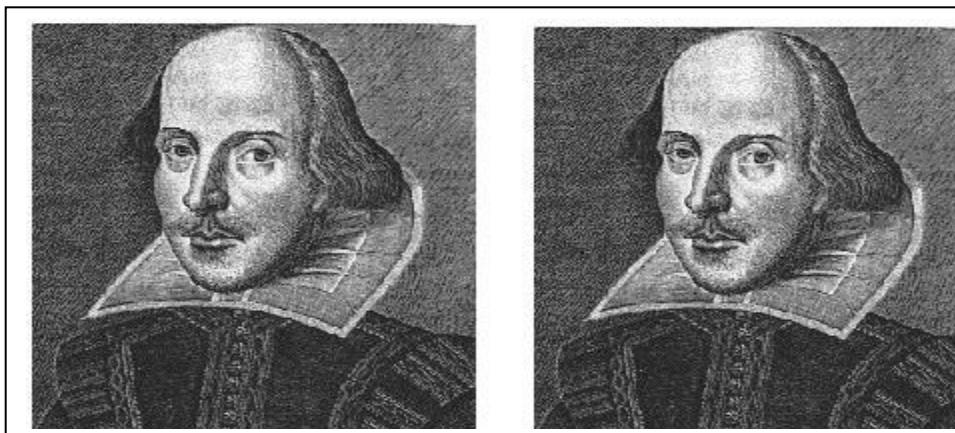


Fig. (8) : Result of embedding M1 in C2 using white Noise storm..

Arachelian encourages encrypting the message before embedding it into an image. White Noise Storm (WNS) also includes an encryption routine to (randomize) the bits within an image. His use of encryption with steganography is well integrated, but is beyond the scope of this paper. WNS was designed based on the idea of spread spectrum technology and frequency hopping. Instead of having X channels of communication which are changed with a fixed formula and passkey. Eight channels are spread within a number of 8-bits\*W byte channels. W represents a random sized window of W bytes. Each of these eight channels represents one single bit, so each window holds one byte of information and a lot of unused bits. These channels rotate among themselves, for instance bit 1 might be swapped with bit 7, or all the bits may rotate positions at once. These bits change location within the window on the byte level. The rules for this swapping are dictated not only by the passphrase but also by the previous window's random data (similar to DES block encryption). WNS also used the Least Significant Bit (LSB) application of steganography and applies this method to PCX8 files. The software extracts the LSBs from the container image and stores them in a file. The message is encrypted and applied to these bits to create a (new) set of LSBs. These are then (injected) into the container image to create a new image. The documentation that accompanies White Noise Storm is well organized and explains some of the theory behind the implementation of encryption and steganography. The main disadvantage of applying the WNS encryption method to steganography is the loss of many bits that can be used to hold information. Relatively large files must be used to hold the same amount of information other methods provide.

### S-Tools

Steganography Tools (S-Tools) for Windows 3.00 by Andy Brown is the most versatile steganography tool of any applications tested. It includes several programs that process GIF and BMP images (ST-BMP.EXE), audio WAV files (ST-WAV.EXE) and will even hide information in the (unused) areas on floppy diskettes (ST-FDD.EXE). In addition to supporting 24-bit images, S-Tools also includes a barrage of encryption routines (Idea, MPJ2, DES, 3DES and NSEA) with many options. S-Tools applies the LSB methods discussed before to both images and audio files. Due to the lack of resources, only images were tested. Brown developed a very nice interface with prompts and well developed on-line documentation.

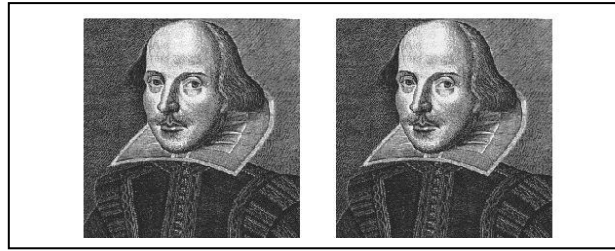


Fig.(9):Left is the original C2.Right is C2 with M1 embedded with S-Tools.

The only apparent limitations were the resources available. There were times large 24-bit images would bring the Windows to a halt. A very useful feature is a status line that displays the largest message size that can be store in an open container file. This saved the time of attempting to store a message that is too large for a container. After hiding the message, the (new) image will be displayed and let you toggle between the new and original images. At times the new image looked to be grossly distorted, but after saving the new image looked nearly identical to the original. This may be due to memory limitations. On occasion a saved image was actually corrupted and could not be read. A saved image should always be reviewed before sending it out. S-Tools provided the most impressive results. Unlike the obvious distortions in (A Cautionary Note on Image Downgrading), S-Tools maintained remarkable image integrity. The following figure illustrates the text message M1 embedded in container C2.

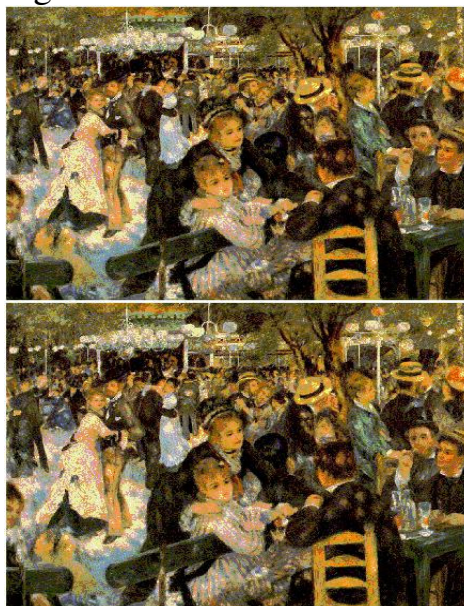


Fig.(10):The following is the original C1(top)and C1 embedded with M2(airfield):

The following is derived from S-Tools BMP-How it is done by Andy Brown: S-Tools works by (spreading) the bit-pattern of the message file to be hidden across the least-significant bits of the color levels in the image. S-Tools tries to reduce the number of image colors in a manner that preserves as much of the image detail as possible. It is difficult to tell the difference between a 256 color image and one reduced to 32. S-Tools adds some extra information on to the front of the message file before hiding. 32 bits of time-dependent random garbage is added first. This step means that two identical hidden files that are encrypted in CBC or PCBC mode will never encipher to the same ciphertext. The 32 bit length of the hidden file is then included. This is required for S-Tools to be able to extract the hidden file. Encryption will conceal this value. To further conceal the presence of a file, S-Tools picks its bits from the image based on the output of a random number generator. This is designed to defeat an attacker who might apply a statistical randomness test to the lower bits of the image to determine whether encrypted data is hidden there (well-encrypted data shows up as pure white noise). The random number generator used by S-Tools is based on the output of the MD5 message digest algorithm, and is not easily (if at all) defeatable.

- Software not tested but worth noting

The following software packages were reviewed but not tested: Jpeg-Jsteg v4 and Stealth v1.1.

#### Jpeg-Jsteg v4

Cryptography and steganography rely on retrieving a message in its original form without losing any information. Such is the idea behind lossless compression. Since JPG images use lossy encoding to compress its data, it is generally thought that steganography would be infeasible with such images. (This version of the Independent JPEG Group's JPEG Software has been modified for 1-bit steganography in JFIF output files) (Independent JPEG Group). The Jpeg-Jsteg software comes with source code and instructions for compiling the code on various platforms. According to the Independent JPEG Group (IJPEG), the JFIF format is composed of lossy and non-lossy stages. Information can be inserted between these stages without corrupting the image. As discussed earlier with Renoir's *Le Moulin de la Galette* compression is a great advantage JPG images have over other formats. JPEG images are becoming more abundant on the Internet because large images with unlimited colors can be stored in relatively small files (a 1073 x 790 pixel image with 16 million

colors can be stored in a 170 Kilobyte file. The same image is over 2 Megabytes if converted to a BMP).

#### Stealth v1.1

Stealth by Henry Hastur in and of itself is not a steganographic program or method. It is usually found with steganographic software on the Internet and is used to complement the steganographic methods. Stealth is a filter that strips off the PGP header that is on a PGP encrypted file. This leaves only the encrypted data. Why is this important? Applying steganography to an encrypted message is more secure than a (plain text) message. However, many encryption applications add header information to the encrypted message. This header information identifies the method used to encrypt the data. For example, if a cracker has identified hidden data in an image and has successfully extracted the encrypted message, a header for the encryption method would point the cracker in the right direction for additional cryptanalysis. But, if the header is removed, the cracker cannot determine the method for encryption. Some steganography software (White Noise Storm and S-Tools) provide this step in security, but others do not.

#### Conclusion and Comments

Steganography has its place in security. It is not intended to replace cryptography but supplement it. Hiding a message with steganography methods reduces the chance of a message being detected. However, if that message is also encrypted, if discovered, it must also be cracked (yet another layer of protection). There are an infinite number of steganography applications. This paper explores a tiny fraction of the art of steganography. It goes well beyond simply embedding text in an image. Steganography does not only pertain to digital images but also to other media (files such as voice, other text and binaries; other media such as communication channels, the list can go on and on). Consider the following example:

A person has a cassette tape of Pink Floyd's (The Wall.) The plans of a Top Secret project (e.g., device, aircraft, covert operation) are embedded, using some steganographic method, on that tape. Since the alterations of the (expected contents) cannot be detected, (especially by human ears and probably not easily so by digital means) these plans can cross borders and trade hands undetected. How do you detect which recording has the message? This is a trivial (and incomplete) example, but it goes far beyond simple image encoding in an image with homogeneous regions. Part of secrecy is selecting the proper mechanisms. Consider encoding using an



Mandelbrot image. In and of itself, steganography is not a good solution to secrecy, but neither is simple substitution and short block permutation for encryption. But if these methods are combined, you have much stronger encryption routines (methods). For example (again over simplified): If a message is encrypted using substitution (substituting one alphabet with another), permute the message (shuffle the text) and apply a substitution again, then the encrypted ciphertext is more secure than using only substitution or only permutation. NOW, if the ciphertext is embedded in an (image, video, voice, etc.) it is even more secure. If an encrypted message is intercepted, the interceptor knows the text is an encrypted message. With steganography, the interceptor may not know the object contains a message.

## **References**

- Andrew D.Mcdonald and Markus G.Kuhn,.(2000):StegFs,A steganographic file system for linux.
- Bret Dunbar,.(2002): A detailed look at steganographic techniques and their use.
- Drew Mclellan,.(2002): Flash satay,Embedding flash while supporting standards
- Gray C.Kessler,.(2001):steganography,Hiding Data within Data.
- James C.Judge,.(2001): Steganography,Past, Present, Future.
- Jessica Fridrich,.(2006): Wet paper codes with improved embedding efficiency.
- Kawaguchi, E,Eason Ro,.(2001):principle and applications of B PCS steganography .
- Mark Owens,.(2002): A discussion of covert channels and steganography.
- Neil F. Jahnson,.(1995):steganography technical Report.
- R.Chandramouli,.(2004): Web search steganalysis,some challenges and approaches.
- Tom Olzak,.(2007): protect your organization from steganographic data theft.
- W. Bender et al,.(1996): Techniquis for data hiding.

## فن الاختزال (تجفير وإخفاء نص داخل صورة)

عاصم مجيد مرشد

جامعة كركوك – كلية العلوم

### الخلاصة

ان الخلاصة الموجزة التي يمكن ان نتكلمها عن هذا المشروع هي ان فكرة أمنية البيانات أصبحت شيئاً مهماً جداً في حياتنا بحيث لم نكتفي فقط بتشفير الرسائل المرسله بل باخفائها أيضاً، حيث ان فن الاختزال هو فن اخفاء البيانات والمعلومات ضمن معلومات اخرى ظاهرة للعيان لكن في الحقيقة ليست هي المعلومة الأصلية التي نبحث عنها، فهي مخبأة ضمن المعلومة الظاهرة، والمعلومة التي نريد اخفائها هنا في هذا البحث هو نص ويتم اخفاؤه ضمن صورة، والصورة التي سنستخدم عليها هي من نوع تنسيقات BMP حيث أن تنسيقها معروف وبسيط، وسنستخدم على لغة البرمجة فيجوال بيسك كأداة لاخفاء النص ضمن الصورة. ان اخفاء نص معين ضمن صورة يتطلب امكانية استخراج النص من الصورة مرة اخرى لذلك سيتم تكوين البرنامج المعاكس لاخفاء النص وهو استخراج النص. ان النص سيحل محل بيانات الصورة وبالتالي ممكن ان يشوه الصورة فقد يعرف من التشويه بأن الصورة حاوية على معلومات مخبأة لذلك فان النص سيوزع على كل اجزاء الصورة كي لايرى التشويه وكي لا يتم ملاحظته، يمكن في المستقبل اخفاء بيانات من نوع آخر في (حاملات البيانات) من نوع آخر كاخفاء نص داخل ملف صوتي أو غير ذلك.