CHAMELEON

IMAGE | STEGANOGRAPHY

Technical Paper

by Mark David Gan

Chameleon *is an image steganography software developed by Mark David Gan for his thesis at STI College Bacoor, a computer college of the STI Network in the Philippines.*

Chameleon features a novel ***adaptive encoding*** *algorithm for 24-bit true-color images based on the steganographic model conceptualized by Yeuan-Kwen Lee and Ling-Hwei Chen*[1] *for grayscale images.*

STEGANOGRAPHY is the art and science of hiding the existence of information. The term originated from Greek roots that literally mean "covered writing".[2] Unlike cryptography, which simply conceals the content or meaning of a message, steganography conceals the very existence of a message.[3]

In computer-based steganography, several forms of digital media may be used as "cover" for hidden information. Photos, documents, web pages, and even MP3 music files may all serve as innocuous-looking hosts for secret messages.

In covert communications through the Internet, digital images are possibly the most practical type of steganographic medium primarily due to their sheer abundance in the Web. However, one common problem with using digital images is the use of insufficient hiding capacities. Most steganography software hide information by replacing only the *least-significant bits* (LSB) of an image with bits from the file that is to be hidden. This technique is generally called *LSB encoding*.

Figure 1: Parrots.

LSB Encoding

To illustrate LSB encoding, consider the picture on Figure 1. Having a *true-color* palette, this image is composed of red, green, and blue color channels (shown in Figure 2). The pixel at the top-left corner of the picture has the values 122, 119, and 92 for its red, green, and blue color components respectively. In binary, these values may be written as:

```
01111010 01110111 01011100
```

To hide the character "a" in the image, the LSB (the rightmost bit) of each of the three 8-bit color values

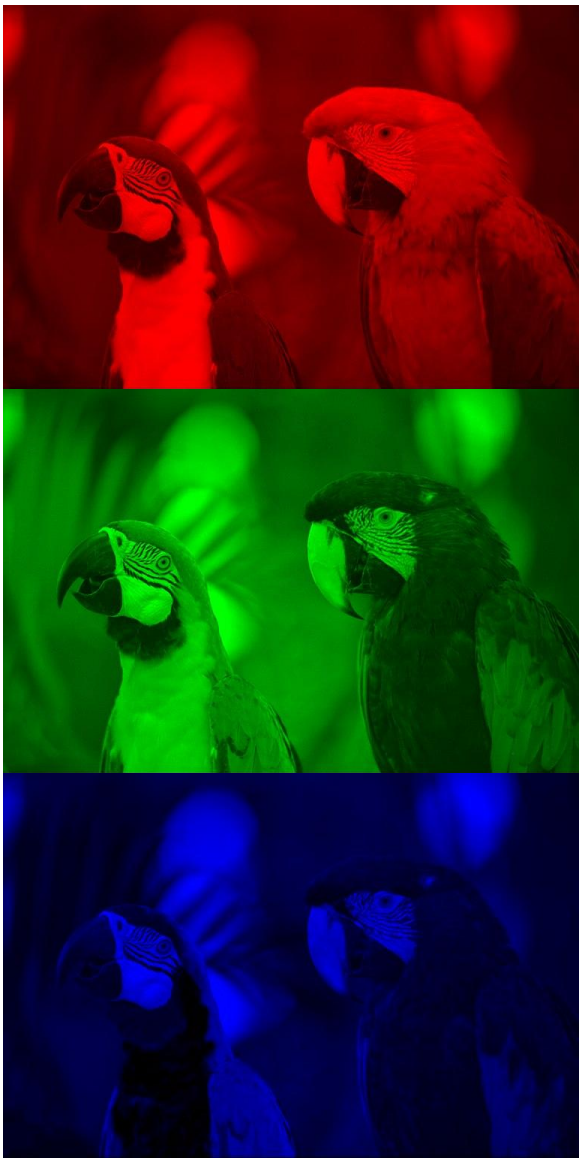

Figure 2: The Red, Green, and Blue Color Channels of the Parrots Picture.

above will be replaced with the bits that form the binary equivalent of the character "a" (i.e., 01100001). This replacement operation is generally called *embedding*. After embedding, the color values would now change to:

```
01111010 01110111 01011101
```

Since there are only three values, only three of the eight bits of the character "a" can fit on this pixel. Therefore, the succeeding pixels of this image will also be used.

In the three color values shown above, only the last value actually changed as a result of LSB encoding, which means almost nothing has changed in the appearance of the image. Nevertheless, even in cases wherein all LSB's are changed, most images would still retain their original appearance because of the fact that the LSB's represent a very minute portion (roughly $^{1}/_{255}$ or 0.39%) of the whole image. The resulting difference between the new from the original color value is called the *embedding error*.

Since there are only three LSB's for each pixel, the total number of bits that can be hidden is only three times the total number of pixels. Having the dimensions 768x512, the image in Figure 1 has a total hiding capacity of 147,465 bytes.

Adaptive Encoding

To maximize hiding capacity, more than just the LSB's of an image must be used. However, simply using more *bit-planes*[*] would not solve the problem since this could cause visible marks or distortions to appear on the processed image.

With adaptive encoding, each pixel uses different hiding capacities as dictated by the pixel's tolerance to modifications. A pixel is said to be more tolerant if higher degrees of changes to its value are possible without changing the general appearance of the image. In essence, smooth and solid areas of an

[*] Lee and Chen provides an excellent discussion about bit-planes in their paper "An Adaptive Image Steganographic Model Based on Minimum-Error LSB Replacement."[4]

image are less tolerant compared to areas with complex textures.

Chameleon implements adaptive encoding based on the steganographic model presented by Lee and Chen in their paper "High Capacity Image Steganographic Model."[1] This model is basically composed of three steps: *capacity evaluation*, *minimum-error replacement*, and *error diffusion*.

Capacity Evaluation

To illustrate capacity evaluation, consider the set of grayscale pixels in Figure 3, wherein pixel *P* represents the pixel being analyzed.

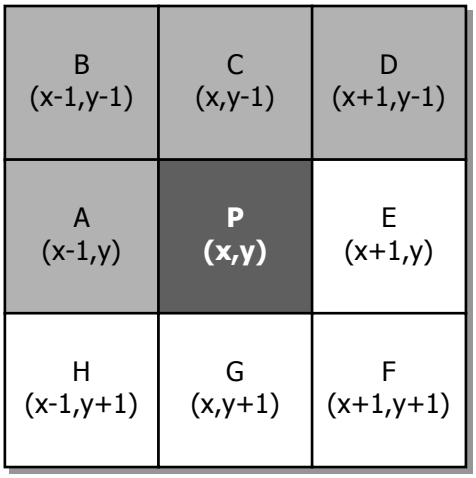

Figure 3: Set of Grayscale Pixels.

In Lee and Chen's model,[1] the first step is to get the *gray value variation* among the pixels on the top and middle-left of the pixel being analyzed. In Figure 3, these are pixels *A*, *B*, *C*, and *D*. The gray value variation is defined as the difference between the maximum and the minimum gray values among the four pixels. The formula for the gray level variation *V* can be written as:

$$V = \max\{A, B, C, D\} - \min\{A, B, C, D\}$$

According to Lee and Chen,[1] the number of bits that can be modified in pixel *P* is the minimum number of bits needed to store the binary value of *V* minus 1. This can be expressed mathematically as:

$$K = \lfloor \log_2 V \rfloor$$

It is important to note that with this technique, embedding can only be performed in a top-to-bottom left-to-right orientation. Only the top and middle-left adjacent pixels are considered in the evaluation because the embedding function has already passed through these pixels. The other four pixels on the bottom and middle-right of pixel *P* are still to undergo the embedding process, which means their values may still change.

Lee and Chen also noted a distinct characteristic of the *human visual system* (HVS) that is crucial to capacity evaluation. They stated that for the human eye, "the greater the gray-scale is, the more change of the gray-scale could be tolerated."[1] Simply put, this means that the closer a pixel is to the color white, the more tolerant it is to modifications. Considering this, an *upper boundary* for the capacity of a pixel can be set based on its intensity. Lee and Chen provided the following condition[1]:

$$\text{if } P > 191 \text{ then } U = 5, \text{ else } U = 4$$

A threshold of 5 is set for U because the more significant bits of the pixels must not be allowed to change so that the upper boundary may still be calculated accurately in the decoding process. The succeeding section will explain why the constants 191, 5, and 4 were chosen for the condition for *U*.

Although not explicitly stated in their paper, it is understandable that Lee and Chen's capacity evaluation function calculates a pixel's hiding capacity based on the *range* of gray values of surrounding pixels. Such calculation is sufficient for grayscale images.

However, since Chameleon is designed for true-color images, Lee and Chen's model would have to be implemented differently. First of all, capacity evaluation will have to be performed on each of the three color channels independently. As a result, one color component of a pixel may increase while another decreases.

In such cases, the increase in total *embedding error* may result to visible distortions. To compensate for this, the original formula for *gray value variation* is replaced with a formula for *color intensity variation* based on the "texture" formed by pairs of adjacent pixels on the top and middle-left sides of the pixel being evaluated. This new formula may be expressed in the form:

$$V = \text{round}\,[(|C-A| + |A-B| + |B-C| + |C-D|) / 4]$$

This modified capacity evaluation function provides a more sensitive and more accurate estimate of a pixel's tolerance to modifications.

Minimum-Error Replacement

In order to minimize the changes made to a pixel as a result of embedding, *minimum-error replacement* (MER) is also incorporated in Chameleon. The idea behind MER is to adjust the bit that is immediately succeeding the modified LSB's of a particular color value in such a way that the change caused by the embedding operation is minimal.

For example, if a binary number `1000` (decimal number 8) is changed to `1111` (decimal number 15) because its three LSB's were replaced with embedded data, the difference from the original number is 7. This difference in the original value of a color component is called the *embedding error*. By adjusting the fourth bit from a value of `1` to a value of `0`, the binary number now becomes `0111` (decimal number 7) and the embedding error is reduced to 1 while at the same time preserving the value of the three embedded bits.

For every K number of LSB's used for embedding in a particular color value, the maximum embedding error for that color value is 2^K-1, or the maximum value for a set of K bits.[4] Since MER adjusts the bit next to the modified LSB's, the embedding error is restricted to a maximum value of $2^{(K-1)}$.

Going back to the calculation of the upper boundary in capacity evaluation, the constants 191, 5, and 4 were chosen with consideration to how MER works. A threshold of 5 was selected since embedding five bits into an 8-bit value (a byte) using MER would mean that the sixth LSB might also change. Since the only remaining bits that are protected from modification are the two most significant bits (MSB), the value of a byte when only these two bits are set to `1` is 192 (i.e., $2^7 + 2^6$). Whatever the value of the six LSBs, the value of the byte will always be greater than 191 as long as both of the two MSB's are set to `1`.

Error Diffusion

Since the capacity-evaluation technique presented by Lee and Chen analyzes only the top and middle-left adjacent pixels, distortions in the contour of certain areas in an image might still be made. For an image to adapt to the changes in color caused by embedding, an *error diffusion* technique must also be employed.

Lee and Chen presented an error diffusion technique called *Improved Gray-Scale Compensation* (IGSC), which makes up for the embedding error in the grayscale value of a pixel by spreading that error evenly across the adjacent pixels. This is done by adding ¼ of the embedding error to the intensity or grayscale value of the bottom and middle-right adjacent pixels. These pixels are depicted in Figure 3 as pixels *E*, *F*, *G*, and *H*.

In simpler terms, when a pixel's intensity increases, the intensities of the four adjacent pixels on its bottom-right sides are decreased. Performing such correcting operations allow the image to preserve the average color intensity of the image as well as possible correlations[*] in the image's bit-planes.

Security Measures

In the case wherein an unauthorized user learns that information is hidden within an image, the threat of unauthorized access to the secret information arises. This is particularly an issue with Lee and Chen's model since it requires embedding to be performed in an essentially linear pattern, which makes it easy for an attacker to successfully extract the hidden data.

Pseudo-random Encoding Pattern

To compensate for the model's limitation, Chameleon generates a *pseudo-random*[†] pattern of embedding based on a user-given password. But instead of randomizing the sequence in which a certain pixel coordinate is processed, just as in other steganography software (which is of course not applicable to Lee and Chen's model), Chameleon randomizes the sequence in which a color channel is processed.

Simply put, for each set of data bits to be hidden, Chameleon pseudo-randomly selects a color channel in which embedding is to be performed. As a result, the hidden information is scattered piece-by-piece throughout the three color channels. This turns out to be a very crucial advantage of having multiple color channels, as compared to grayscale images which only have one color channel.

Password Verification

The seed used by Chameleon in generating the pseudo-random encoding pattern is actually the computed MD5[‡] hash value of the user-given password. Since hash values have a tendency of not being unique (i.e., two or more values can have the

[*] In their paper,[4] Lee and Chen explains how certain correlations in the bit-planes of an image can suggest the presence of steganography and compromise the security of hidden information.

[†] Real random values are not based on any other value and are completely accidental. The term "pseudo-random" indicate that a value, although not pre-determined nor fixed, is essentially based on a certain *seed* or base value. This therefore means that the same value can be generated repeatedly as long as the corresponding seed is used.

[‡] MD5 is the strengthened version of the 128-bit MD4 message digest algorithm.

same corresponding hash value), it is possible (although very unlikely) for an unauthorized user to use an incorrect password that coincidentally generates the same hash value with that of the correct password.

For this reason, Chameleon also stores within the image the SHA-1* hash value of the user-given password. Each time a user enters a password for the extraction process, the SHA-1 hash value of that password is compared with the stored hash value. Therefore, even if an incorrect password coincidentally generates the same MD5 hash value with that of the correct password, access will still be denied since it will not have the same SHA-1 hash value with that of the correct password.

Accordingly, even if an incorrect password coincidentally generates the same SHA-1 hash value with that of the correct password, extraction will still be unsuccessful since it will not have the same MD5 hash value with that of the correct password.

Data Encryption

As a supplement to all other security measures employed by Chameleon, the RC4† data encryption algorithm is also integrated with the encoding process. Before a data file is embedded in an image, the file is first encrypted using RC4.

The password used for encryption is also the user-given password used in the embedding process so that the user is required to remember only one password. Because of this password, even in cases wherein the presence of steganography has been discovered, an attacker would still need the correct password to both extract and decrypt the hidden data file.

File Wiping

Chameleon also features a *file wiping* (a.k.a. "file shredding") option which allows users to permanently delete files from the computer. Typically, when a file is deleted, it is simply unregistered from the file system but its actual contents are left intact in the storage media. This makes it possible for an attacker to recover the contents of the deleted file. To ensure data security, the actual contents of the file must be erased. Such an operation is generally called file wiping.

In wiping a file, the first step that Chameleon performs is to move the file in the system's temporary file folder and rename the file. This step removes any identifying information that a file's name or location may provide. Next, Chameleon overwrites the file with a bit pattern of `01010101`. After that, the file is overwritten again but this time with a bit pattern of `10101010`. Finally, the file is overwritten with `0`'s and then deleted from the storage media. This scheme is a simpler and faster version of the U.S. Department of Defense's DOD 5220.22-M standard for data deletion.

In covert communications, this feature is very useful in erasing an extracted message file, or even the *stego image*‡ carrying it, once the intended recipient has

* SHA-1, which stands for Secure Hash Algorithm, is a 160-bit hash algorithm based on MD4.

† RC4 is a variable key-size *stream cipher* algorithm designed for fast software implementation.

‡ The processed image in which hidden information is embedded is generally called a *stego image*. This is the output of the embedding process.

read its contents. The sender of the stego image may also use this feature in erasing the original copies of the message file and the *cover image*.* Erasing the original copy of cover images provides better security since the existence of two different copies of the same image may suggest the presence of steganography.

The file wiping feature is also automatically used by Chameleon in erasing all the temporary files used in every encoding and decoding task.

Other Features

To further increase the hiding capacity of an image, Chameleon also features integrated file compression. For this purpose, it uses the Zlib format of the open-source Zlib compression library.

For every encoding task, each selected data file is first compressed, and then encrypted, before being embedded (in a pseudo-random pattern) into the cover image.

Chameleon also embeds along with the data file a bit stream of *metadata* (or header information) placed in a pseudo-random address within the cover image. Like the encoding pattern for the data file, the metadata address is also based on the user-given password.

The metadata encoded by Chameleon is composed of the following information:

1. 20-byte string for the SHA-1 hash value of the user-given password.
2. 255-byte string for the filename of the hidden data file.
3. 8-byte value for the date and time in which the hidden data file was last modified or created.
4. 4-byte integer for the size of the data file in terms of bytes.
5. 16-byte string for the MD5 *checksum* of the hidden data file.

As discussed earlier, the SHA-1 hash value of the user-given password is included for password verification. On the other hand, the filename of the data file is included to provide descriptive information for the intended recipient of the hidden information. The file date and time is also included for reference. The file size also provides information to the recipient but is mainly used by Chameleon to keep track of how much data to extract from a stego image.

The last piece of metadata, the data file's checksum, is included to ensure that the extracted file is correct and complete. This is useful in cases wherein a stego image has been modified or tampered with, which often leads to the corruption of the hidden data file.

Image File Formats

Specifically, Chameleon is applicable to any 24-bit true-color *bitmap* (a.k.a. "raster images") that does not use *lossy compression*. Lossy compression refers to a compression scheme wherein "some data is deliberately discarded to achieve massive reductions in the size of the compressed file."[5] A good example of which is the popular JPEG compression, which uses the Discrete Cosine Transform (DCT). Since

* The original unprocessed form of the image is called a *cover image*. This is an input of the embedding process.

Chameleon encodes hidden information within the actual color values of an image's pixels, formats such as JPEG are not supported. When a JPEG image is selected to be a cover, the resulting stego image will automatically be converted into *lossless* format.

Chameleon uses the open-source FreeImage graphics library which allows it to save stego images in the BMP (Windows Bitmap), PNG (Portable Network Graphics), TIFF (Tagged Image File Format), TARGA (TrueVision Advanced Raster Graphics Adapter), and PPM (Portable Pixelmap) formats. Nevertheless, stego images created with Chameleon maybe converted into any other 24-bit true-color image format that does not use lossy compression.

Chameleon can also be used to open (as a cover image), but not save (as a stego image), images stored in the JPEG, PCD (Kodak PhotoCD), PCX (PC Paintbrush), and PSD (Adobe Photoshop) formats.

Conclusion

Through adaptive encoding, Chameleon is able to utilize *optimum* hiding capacities in every cover image. This allows users to hide files of larger sizes while at the same time preserve the general appearance of any cover image used.

Furthermore, the implementation of various security measures provides a high level of protection for the hidden data.

Although limited to lossless image formats, which are in any case standard and considerably widespread, Chameleon is still useful in real-world applications especially in cases wherein large volumes of sensitive data need to be transmitted secretly over public communications channels such as the Internet.

References

[1] Lee, Yeuan-Kwen and Ling Hwei Chen. "High Capacity Image Steganographic Model." *Vision, Image and Signal Processing*. IEEE Proceedings 147.3 (2000): 288-294.

[2] Khan, David. "The History of Steganography." *Information Hiding: First International Workshop*. Lecture Notes in Computer Science 1174 (1996): 1-5.

[3] Anderson, Ross J. and Fabien A. P. Petitcolas. "On The Limits of Steganography." *Special Issue on Copyright & Privacy Protection*. IEEE Journal of Selected Areas in Communications 16.4 (1998): 474-481.

[4] Lee, Yeuan-Kwen and Ling Hwei Chen. "An Adaptive Image Steganographic Model Based on Minimum-Error LSB Replacement." *Ninth National Conference on Information Security*. (1999): 8-15.

[5] Pfaffenberger, Bryan and David Wall. Que's Computer and Internet Dictionary. 6th ed. Indianapolis: Que, 1995.