Handbook of Tulipan

Secure Steganography Scheme

April 5, 2012

Contents

Chapter 1

Introduction

1.1 Preamble

This guide is published as supplementary part to the first release of TULIPAN Software, a Secure Steganography Scheme.

In this Handbook we assume that the programmer uses Gnu Compiler Collection under a distribution of GNU/Linux Operating System.

1.2 What does Steganography signify?

All histories about the construction and the decay of the cultures and civilizations, have a central axis in common: the secrets. The necessity to hide information caused the great heroes and villains of history to undertake a legendary fight, between devising invulnerable forms to hide the secrets in best way possible and the continuous search to obtain these secrets.

The creative mind of the human kind has found two answers to this need. *Cryptography* or the art of encrypting message, which was employed by Caesar as an example, and the *Stegangraphy* as the art of hiding message. The word steganography is derived from the Greek words "steganos" and "graphein" meaning "covered" and "to write" respectively; in other words, "covered writing". Hiding a message can be done in numerous ways. In Ancient time, a method was to tattoo a person's bald head with a message and after a while hairs would grow concealing the message. Once at the destination the head could again be shaved and the message would be revealed. Before invention of computers, a cover unimportant text could be used to contain the embedded message within it. For instance, every 12th letter of the cover message could be used as the

hidden message. Other methods consisted of letter heights, using various chemicals for invisible ink and even microdot pricks, which became popular during World War II.

Nowadays, The conventional way of securing data transactions is through the use of standard encryption techniques such as RSA, 3DES, and AES. In the few past years, however, a new branch of data security techniques known as digital steganography has evolved and is continuing to receive a great deal of attention from both Academic and Industrial communities.

The question is why steganography attracts so much interest comparing to other forms of secret communication, while in current situation, cryptography is considered as the strongest way of secure communication? One of the drawbacks of cryptography is the mere fact of knowing that communication is going on secretly. On the other hand, with steganography that fact could itself be hidden.

Currently, the most sophisticated forms of steganography are embedding messages within digital pictures such as a Graphics Interchange Format (GIF) or JPEG files, sound files and even video files. The trick is to alter the file's insignificant information (for containing the message) such that the file itself does not show any visual or audio sign of alteration. The sender and recipient just need to agree, perhaps by having the embedding and extracting private algorithms to know how to send and receive a secret message. As code analyzers uncover disguise after disguise, it becomes an ongoing race to find unique ways to conceal messages. Combined with encryption, however, this form of secret communication would then be very strong because once the message is located, it then turns to a case of trying to decrypt it, which would be the case in normal cryptanalysis.

Steganography is important for practical reasons as well, for instance in copyright and watermarking. If you have developed a digital picture, for example, and do not want anyone else to use it unless they pay you the royalties, then you can embed a particular set of bits into the picture (enough bits, not to change it visibly of course) such that you could track and identify your digital property anywhere it is being used. Watermarks can be set up in a fashion that does not allow effective alterations to the media. Also, whenever someone wants to alter your picture it causes an undesirable effect, perhaps by placing a big off-color X on top of the picture.

In the very same way that the cryptanalysis has been born, steganography has an inseparable sister called steganalysis which designs both passive attacks to detect the mere existence of the hidden message in cover media or even tries to decode it and active attacks in purpose of destroying the (probable) hidden message.

During past few years, steganography has grown both in academic and industrial fields. Many algorithms have been developed to implement more secure ways of hiding data and in parallel many statistical attacks to detect the employement of these methods. On the other hand, many commercial products have been developed and published to answer the need for communication market on Internet and other kinds of media.

TULIPAN is a software designed in order to satisfy the goal of a secure steganogrphy scheme. TULIPAN hides the information in ordinary JPEG images (the most popular format used on the web) using one of the most recent and most secure [2] algorithms, called Perturbed Quantization [1].

1.3 Overview of the Handbook

This handbook consists of three independent parts and each part is written for an independent purpose.

Chapter 2, is a detailed explanation of the steganographic algorithm implemented in TULIPAN. The algorithm is chosen for its outstanding result in steganalysis tests but still it is subjected to improvement, in future versions.

Chapter 3, is the user manual and the usage description of TULIPAN. Each entry to the command line of TULIPAN is discussed separately.

Chapter 4 discusses the points that we have found useful and significant for prospective improvement. This chapter consists of several points to improve TULIPAN to result in a faster and more secure steganography.

As TULIPAN is not a final version program, the Programmer's Guide is not provided. After improving the program structure and the release of final version, the next version of Handbook will contain the Programmer's Guide as well.

Chapter 2

The TULIPAN Steganogrpahy Algorithm

2.1 Introduction

In this project we are trying to use a new method for applying "adaptive stetganography" without bearing its side effects like being "weakly dependant on the key". Using TULIPAN, the sender can hide information in an arbitrary JPEG image using "adaptive steganography" with no need to send extra information to the recipient about the special manner of distributing secret bits.

In practice, a steganographic scheme is considered secure fno existing attack can be modified to build a detector that would be able to distinguish between cover and stego images with a success better than random guessing.

In other adaptive schemes, the adaptively selected components are normally noisy areas or segments with a complex texture. But if the information about the selection is "public" or only weakly dependent on a key, the attacker can apply the same rule and start building an attack. So, the adaptive selection does not necessarily improve steganographic security!

This problem with adaptive steganography could be solved if the selection rule is available only to the sender as side information but in principle unavailable to the recipient (and thus any attacker). For example raw, uncompressed image can be used by sender to embed data in JPEG compressed form of it. This idea is generalizied and applied in "*Perturbed Quantization (PQ)*".

The selection rule is derived from a coding theory problem of transmiting data through a channel which contains a memory with (a large number of) defective cells. To understand better, the

idea of selection rule in PQ can be similar to writing on a partly wet paper such that we can write only on the dry parts. While being sent, the wet part dries out on the way and so nobody (neither the recipient nor the attacker) can distinguish between original wet and dry patterns.

The benefit of the defective memory code (the wet paper code) is that not only does it give the sender control over the embedding modifications but also enables the sender to communicate to the recipient on average the same number of bits as if the recipient knew the set of dry pixels.

2.2 Perturbed Quantization

Take a raw image X (grayscale as an example). During JPEG compression for the first time, performing the Discrete Cosine Transform (DCT), a quantization table is used to divide the DCT coefficients by quantization steps; then according to JPEG standards the compression rounds them to integers, finally being encoded to a JPEG file G. Assuming d_i and D_i as the DCT coefficients before and after rounding, respectively, we define "*changeable coefficients*" (dry coefficients) as those coefficients d_i whose fractional part is in a narrow interval around 0.5. That is $d_i - \lfloor d_i \rfloor \in [0.5 - \epsilon, 0.5 + \epsilon]$.

During compression, we will round changeable coefficients d_j up or down at our will and thus encode up to k bits (k= total number of d_js, 'changeable coefficients') obtaining a compressed and embedded image G.

However, we cannot simply code the message bits as LSBs (Least Significant Bit) of the rounded DCT coefficients D_j because the recipient would not know which coefficients carry message bits. We will show later how the sender can communicate on average k bits to the recipient, who has no information about the set of changeable coefficients.

This method, called Perturbed Quantization (PQ), is a method in which during compression we slightly perturb the quantizer (the process of rounding to integers) for a certain subset of changeable coefficients in order to embed message bits.

Here some reasons are given which certify the formidablity of distinguishing incorrectly quantized values D_i by an attacker trying to find statistical evidence for it:

1. The sender is using side information that is practically removed during quantization and is unavailable to the attacker. It is in general impossible for the attacker to reverse JPEG compression and obtain the uncompressed image.

2. Additional selection rule(s) can be established by the sender to decrease the probability of introducing detectable artifacts more and improve the security. For example, to avoid changing coefficients in the areas of the cover image with the attacker's higher prediction probability.

3. The rounding process $d_j \to D_j$ (quantizing of changable coefficients) has a large stochastic component as a result of the image noise influence on the rounding of changeable coefficients. The authors of [1] are currently working on a better justification of this heuristic statement in a more exact manner using image models.

2.2.1 Information-reducing processes

Using the idea mentioned above, the sender can slightly modify the numerical values (such as pixel/coefficient/color/index values depending on the format of image X) when he is downgrading the digital image (such as using lossy compression, downsizing, quantization, format conversion, recompression, etc.) as he will have access to the values before quantization/rounding is done. So he can repeat the modification whenever he performs any kind of information-reducing process which includes applying quantizers.

There are 3 famous examples of image downgrading operations F that could be used for steganography based on PQ: Resizing, Decreasing the color depth by d bits, and JPEG compression. Here, in TULIPAN we only take the advantage of JPEG compression:

JPEG compression: For grayscale images, the transformation T maps a square $m_1 \times m_2$ matrix of integers x_{ij}, into a $m_1 \times m_2$ matrix of real numbers u_{ij} in a block-by-block manner. In each 8×8 pixel block B^x, the corresponding block B^u in u_{ij} is $DCT(B^x)./q$, where DCT is the 2D DCT transform, q is the quantization matrix, and the operation ./ is an element-wise division. Then the quantizer Q is applied to vector u by coordinates $Q(u_i) = round(u_i)$.

2.2.2 Memory with defective cells

During this stage, the sender, following some Selection Rule (SR), identifies the set of indices $C \subseteq \{1, ..., n\}$ of coefficeints (or, in general, cover object elements) whose values $u_j, j \in C$, may be perturbed during quantization. As already mentioned above, the sender can, for example, select u_i whose values are close to the middle of the quantization intervals of Q:

$$C = \{i | i \in \{1, ..., n\}, u_i \in [L + 0.5 - \epsilon, L + 0.5 + \epsilon] \text{ for some integer} L\}.$$

The tolerance ϵ, should be a small number for example 0.1. According the selection rule, sender can just modify u_is whose indices are members of C to embed the message inside the image. Now for embedding the message, sender should use the LSB bits of coefficient of $Y = F(X)$ processed image, however they can use the k changeable coefficient but cannot change the $n - k$ wet coefficients. As the recipient does not have any information about the set C, this problem is similar to writing in memory with k nondefective and $n - k$ defective cells. As it will be discussed the sender can store on average k bits of the message in such memory.

In our steganogrphic situation, however, the number of defictive cells is so large as in a typical JPEG image, $n \sim 10^6$ and $k \sim 10^4$. The wet paper code is designed to handle such complexity.

2.3 Wet Paper Code

2.3.1 Encoder and Decoder

In the wet paper encoder, generalizing the selection channel idea mentioned before, one message bit is embedded as the LSB of a group of elements. Assuming q changeable elements in the group, we can attempt to embed q message bits by forming q linearly independent linear combinations of element LSBs instead of just one sum modulo 2 of the individual element LSBs to calculate group

LSB. Following this, the sender and recipient agree on a secret stego key that is used to generate a pseudo-random binary matrix D of dimension $q \times n$. Rounding $u_j, j \in C$, the sender obtains the column vector y', so that the modified binary column vector $b', b'_i = LSB(y'_i), i = 1, \ldots, n$, satisfies

$$Db' = m \tag{2.1}$$

at cost of solving a system of linear equations in GF(2) which is the biggest computational load on the sender.

Using the variable $v = b' - b$ we can rewrite (2.1) to

$$Dv = m - Db \tag{2.2}$$

in which there are k unknowns $v_j, j \in C$, while the remaining $n - k$ values $v_i, i \notin C$, are zeros. Thus, on the left hand side, we can remove from D all $n - k$ columns $i : i \notin C$, and also remove from v all $n - k$ elements v_i with $i \notin C$.The system (2.2) now becomes

$$Hv = m - Db \tag{2.3}$$

where H is a binary $q \times k$ submatrix of D and v is an unknown $k \times 1$ binary vector. The solvablity of this system of linear equations will be discussed later.

To decode, having received the modified stego object $Y' = \{y'_i\}$, the recipient should only form $b'_i = LSB(y'_i)$ and using shared matrix D, gets Db' which is the extracted message $m = Db'$.

2.3.2 Practical encoder/decoder implementation

The main complexity of this communication setup is on the side of the sender to solve q linear equations for k unknowns in GF(2). Assuming that the maximal length message is sent, the complexity of Gaussian elimination for (2.3) is $O(k^3)$. For a medium size image with $n = 10^6$ pixels and with $\epsilon = 0.1$, we have $k \sim 10^4$ for a typical 80% quality JPEG image. While solving a linear system with 10^4 unknowns using Gaussian elimination is doable on a PC, it may require several minutes of calculations, which is impractical for the user.

By far the best performance and most flexible method was obtained using structured Gaussian elimination by dividing the image into β pseudo-random disjoint subsets B_i and using the Gaussian elimination on each subset separately. This can bring down the computational requirements substantially because the complexity of Gaussian elimination will decrease by the factor of β^3 while the number of solving increases β-times.

Keeping the above result in mind, we'll describe the encoder and decoder algorithms. r_2 is the estimated range of the ratio of the dry to all coefficient and h is the header size of each matrix. k_i is the number of dry coefficients which are consumed by the i-th matrix:

- **E0.** Using a PRNG, generate a random binary matrix D with n/β columns and sufficiently many rows.
- **E1.** Determine the header size $h = log_2(r_2 n/\beta) + 1, q = |m| + \beta h$

- **E2.** $b' \leftarrow b, i \leftarrow 1$.
- **E3.** $q_i = (k_i(q+10)/k), q_i = min\{q_i, 2^h - 1, |m|\}, m^{(i)} \leftarrow$ the next q_i bits in m
- **E4.** Select the first n_i columns and q_i rows from D and denote this submatrix $D^{(i)}$. Solve q_i equations $H^{(i)}v = m(i) - D^{(i)}b^{(i)}$ for k_i unknowns v, where $H(i)$ is a $q_i \times k_i$ submatrix of $D^{(i)}$ consisting of those columns of $D^{(i)}$ that correspond to changeable bits in B_i. If this system does not have a solution, the encoder decreases q_i till a solution is found.
- **E5.** According to the solution vector v, obtain the i-th segment $b'^{(i)}$ of the vector b' by modifying or leaving $b^{(i)}$ unchanged.
- **E6.** Binary encode q_i using h bits and append them to m.
- **E7.** Remove the first q_i bits from m.
- **E8.** $q \leftarrow q - q_i, k \leftarrow k - k_i, i \leftarrow i + 1$.
- **E9.** If $i < \beta$ GOTO E3.
- **E10.** If $i = \beta, q_\beta \leftarrow q$.
- **E11.** Binary encode q_β using h bits and prepend to m, $m^{(\beta)} \leftarrow m$.
- **E12.** Select the first n_β columns and q_β rows from D and denote this submatrix $D^{(\beta)}$. Solve q_β equations $H^{(\beta)}v = m^{(\beta)} - D^{(\beta)}b^{(\beta)}$ for k_β unknown vs. If this system does not have a solution, exit and report failure to embed the message.
- **E13.** According to the solution v, obtain the β-th segment $b'^{(\beta)}$ of the vector b' by modifying or leaving $b^{(\beta)}$ unchanged.

The Decoder Algorithm is as follows:

- **D0.** Using a PRNG, generate a random binary matrix D with n/β columns and sufficiently many rows.
- **D1.** Determine the header length $h = log_2(r_2 n/\beta) + 1$.
- **D2.** $i \leftarrow \beta$.
- **D3.** Select the first n_β columns and h rows from D and denote this submatrix D_h. Obtain h bits as $D_h b'^{(\beta)}$ and decode as q_β
- **D4.** Select the first n_β columns and next $q_\beta - h$ rows from D and denote this submatrix $D^{(\beta)}$. Obtain message bits $m = D^{(\beta)}b'^{(\beta)}$.
- **D5.** $i \leftarrow i - 1$
- **D6.** Decode q_i from the last h bits of m and remove the last h bits from m.
- **D7.** Select the first n_i columns and q_i rows from D and denote this submatrix $D^{(i)}$. Prepend $D^{(i)}b'^{(i)}$ to m, $m \leftarrow D^{(i)}b'^{(i)}\&m$.
- **D8.** If $i > 1$ GOTO 5
- **D9.** ELSE m is the extracted message.

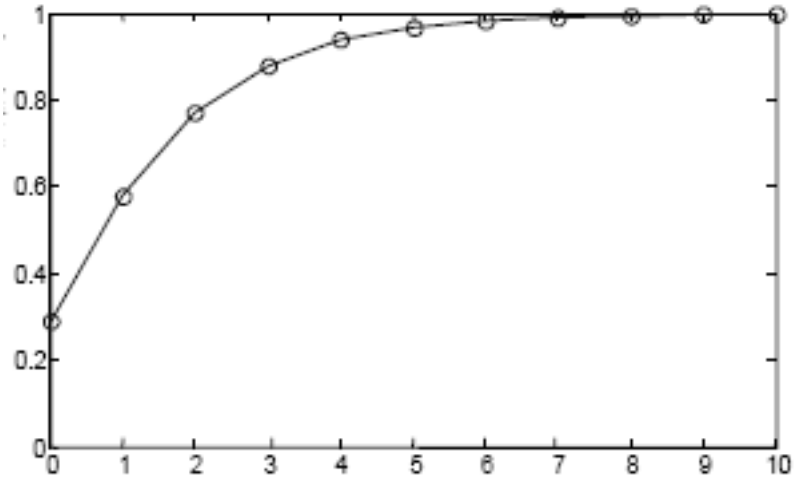

Figure 2.1: Probability that a random $q \times k$ binary matrix has rank k (for large k). The x axis indicates $q - k$.

2.4 Average Capacity

As it was discussed in previous section, the ability of the sender to encode a q bit message, depends on solvablity of $q \times k$ matrix H. That system has a solution for an arbitrary message m as long as $rank(H) \geq q$. The elegant result on which wet paper code is based, is that for large k the probability that the rank of a random $q \times k$ binary matrix becomes maximum (equal to k), quickly approaches 1 as q increases slightly more than k. This result is shown in Figure 2.1.

This means that on average, the sender will be able to communicate k bits to the recipient using the wet paper code.

2.5 Double Compression

In TULIPAN we use the PQ embedding method based on double compression. The method takes a (single compressed) JPEG file as the cover image and produces a double compressed JPEG file as the stego image. The sender and recipient can use the LSB of DCT to embed the message. The sender chooses the second quality factor $Q2 < Q1$ (to make the recompression information-reducing) to derive some dry coefficients in the downgraded image. During the quantization process of the lower quality image, the sender enforces that the quantized value (D_j) of the d_j changeable DCT coefficient in the stego file to be either l or $l + 1$ according to the encoded message bit and stores it as the value of the quantized D_j DCT coefficient in the stego image.

The numerical experiments on downgrading the JPEG images qualities indicates that if the sender sets the second quality according to

$$Q2 = 2(Q1 - 50) \tag{2.4}$$

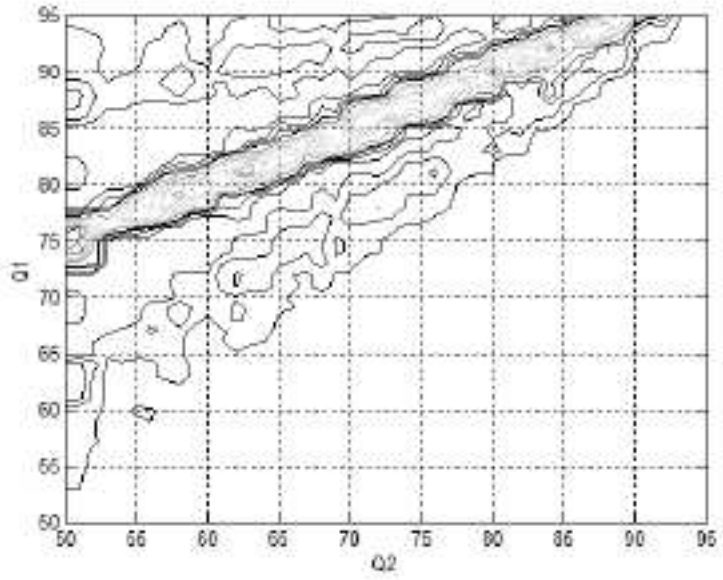

Figure 2.2: Embedding capacity expressed in bits per non-zero DCT coefficient (of the double-compressed image) averaged over 20 test images. Note the prominent ridge for quality factors satisfies $Q2 = 2(Q1 - 50)$.

The number of changeable coefficient increase to about the maximum. This result is depicted in Figure 2.2.

2.6 Steganalysis

An steganalysis on a bulk of gray scale images has been done using a blind steganlayzer. Random messages of various length (0.4, 0.2, 0.1, and 0.05 bits per non-zero DCT coefficient of the stego image) were embedded into a subset of these images using 6 modern steganographic schemes, F5, F5 without matrix embedding, OutGuess, Model based Steganography without and with deblocking and PQ. Then the detection reliability factor of the blind steganalyser is calculated for detecting the existence of message inside the image. The detection reliablity factor of "0" means complete random detection and the complete defeat of the steganalyzer. On the other hand, the detection reliablity factor of "1" means that the steganalyzer can distinguish the stego images from the ordinary images perfectly. The result of this experiment is shown in Table 2.1.

As can be seen from Table 2.1, the PQ algorithm significantly outperforms existing steganographic algorithms for JPEG images.

bpc	F5	F5_111	OG	MB1	MB2	PQ
0.05	0.241	0.645	0.879	0.220	0.163	~ 0
0.1	0.539	0.922	0.993	0.415	0.310	0.048
0.2	0.956	0.996	0.991	0.704	0.570	0.098
0.4	1.000	1.000	U	0.938	0.824	0.174
0.6	1.000	1.000	U	0.983	U	U
0.8	1.000	1.000	U	0.992	U	U

Table 2.1: Detection reliability for F5, F5 without matrix embedding (1,1,1), OutGuess 0.2 (OG), Model based Steganography without and with deblocking (MB1 and MB2, respectively), and the proposed Perturbed Quantization during double compression for different embedding rates (U = unachievable rate). All but the PQ algorithm, were tested with Q = 80. The PQ algorithm was tested with Q1 = 85 and Q2 = 70.

2.7 Discussion and Conclusions

For implementing TULIPAN we used PQ algorithm as it showed the best result from the aspect of security. This result was predictable prior to implementation due to the brand new idea of adaptive stegonography with side information totally unknown by the recipient and the attacker.

However, as the idea is new and independent of the idea of other steganographic schemes, The designers of TULIPAN hope that they can employ other steganographic methods combined by PQ to increase its security.

Chapter 3

TULIPAN User's Guide

3.1 Preliminaries

The TULIPAN is an image steganography tool. It means that the intention of its use is to hide a secret message inside an image an send it such that even the intention of sending such message becomes confidential.

For handling this communication, the sender and the recipient should use TULIPAN in two phases. First the sender should hide the message in an image. This step is called *Embedding Process* in this manual. The details of embedding process is explained in Section 3.3. The sender transmits the image on any insecure channel to the recipient. The recipient should use TULIPAN as well to get the secret message out of the image. This step is called *Retrieval Process* in this guide. The details of retrieval process is explained in 3.4.

This version of TULIPAN is compiled to be run under GNU/Linux Operating Systems, however it can be recompiled for any other environment. As TULIPAN is a terminal based application, for running it:

1. If X-Windows system is running, open a terminal window.
2. Change the directory to where "tulipan" file exists.
3. Run Tulipn using following command:

   `./tulipan <SPECIFICATIONS>`

   For the detail of <SPECIFICATIONS> , refer to Sections 3.2, 3.3 and 3.3.

3.2 General Usage

For embedding process, user should use following command:

```
./tulipan -encode <Message filename> <Cover image filename> <Output image
filename> <Key> <Humidity threshold> <Recompression quality> <No of matrices>
```

The details of the options can be found in Section 3.3.

For the retrieval process, user should use following command:

```
./tulipan -decode <Stego image filename> <Output message filename> <Key>
<Humidity threshold> <No of matrices>
```

The details of the options can be found in Section 3.4.

3.3 Embedding Process

3.3.1 Input/Output Files

For enabling TULIPAN to embed a secret message, two input files should be prepared and specified in TULIPAN command lines; these files are mentioned in Section 3.2 as *Message filename* and *Cover image filename* respectively.

- **Message filename**: should be the complete path of the file that contains the secret message. The Message file can contain anything including text or binary data, image, voice etc.
- **Cover image filename**: should specify complete path of a JPEG image. Although any JPEG image can be used as Cover Image, user should choose the size and quality of image regarding to the size of Message file. If a small low quality image is chosen, TULIPAN fails to fit the whole message inside the image and prompts an error message.
- **Output image filename**: should specify complete path of a new JPEG image. The result of embedding the message file into the Cover image will be a new JPEG image. The tulipan stores this image in the file specified by Output image filename. After running TULIPAN, the sender can transmit this file to the recipient to let them read the Message File.

3.3.2 Secret Key

In TULIPAN, the secret key has a complicated structure to offer maximum security to the user. To reach a secure steganography, the user should set each part of the secret key appropriately.

- **Key** is the main part of Secret Key and is an ordinary string with ordinary size. Anybody who intends to read the Message in Output image needs to know the key string.

- **Humidity Threshold** For the security reason, TULIPAN does not use all part of JPEG image to embed the message. As it is defined in Section 2.2.2, only the dry elements of the JPEG image are allowed to be used in embedding process. The Humidity Threshold is a measure for TULIPAN to decide if an element is dry or wet. Humidity Threshold should be a number between 1 and 100. If Humidity Threshold sets to 100, all elements are considered as wet. The less the Humidity Threshold is, the more secure steganography is gained. The more the Humidity Threshold is, the greater Message can be embedded in the image.

- **Recompression Quality** is a number between 0 to 100. For embedding the message, TULIPAN recompresses the original image, to provide some dry elements. These dry elements will be used in embedding process. The important factor is that the recompression quality should be less than the original image quality. For an 80% quality JPEG image, the 60% recompression quality is recommended.

- **No. of Matrices** Many random matrices are used to code the message data. The user should specify the number of matrices in the command. No. of Matrices greatly influences the performance of TULIPAN and is needed for the retrieval process as well. Using too few Matrices, results in Big Matrices which slowly can be processed. On the other hand, using too many matrices wastes the embedding space for storing the headers and the embedding process may fail due to lack of embedding space. For a 600×800 image, 500 matrices is a typical choice.

3.4 Retrieval Process

3.4.1 Input/Output Files

For enabling TULIPAN to retrieve a secret message, the outputted image of TULIPAN should be prepared and specified in TULIPAN command lines. Also, a new message file is needed, so that TULIPAN stores the result of retrieval process inside it. These files are mentioned in Section 3.2 as *Output Message filename* and *Stego image filename* respectively.

- **Stego image filename**: should specify complete path of the JPEG image which is outputted by previous call to TULIPAN program with "-encode" option. This image is typically what sender transmits to recipient through an insecure channel.

- **Output message filename**: This should be the complete path of the new file. TULIPAN will use this file to store the retrieved information from Stego image.

3.4.2 Secret Key

The secret key of the recipient is the same as the secret key of the sender. With the exception that the recipient does not need to know the recompression quality to retrieve the message.

Chapter 4

Prospective Improvements to Tulipan Software

As the deadline was a major concern in developing the first version of TULIPAN, it is programmed to reach an executable working release as soon as possible by ignoring some important points about software engineering and security. For above reason, although current release of TULIPAN is fully practical, its usage is not recommended for sensitive application.

The developers of TULIPAN had above point in mind during the development of TULIPAN. In each step that they ignored some programming or security point to accelerate the implementation, they have submitted that item in a database. This chapter contains the entry of that database.

- **Improving the steganogrphy algorithm**

  Although the algorithm used in TULIPAN is one of the most secure steganography algorithms recently designed, there is still a long way for improvement. When there are more dry coefficients than is needed many policies can be employed to get a more secure steganographic scheme. This is a place where we can simultaneously use two or more steganography algorithms. Our intention is to use our second choice, Model-Based Steganography in embedding process.

- **Adding Cryptographic Algorithm** It is much more convenient that the data be encrypted before it is embedded in the cover image. Encryption offers two different aspects to the security:

  1. If the scheme is broken the data remain secure.

2. The encryption decreases the redundancy of the data and results in more secure steganography. The statistical and blind steganalysis become much harder on the encrypted data.

Although it is not necessary but it is of convenience to include a cryptographic algorithm in TULIPAN.

- **Using random generator to randomly order the coefficients**

  The order of DCT Coefficients should be specified by a random generator to defuse the impact of steganography widely in the whole image. However, in this implementation the coefficients are used in the same order as they have been saved in the images. This approach was chosen for the ease of debugging and should be corrected in the improved version.

- **Usage of Multilevel Humidity** In current version, the humidity of each coefficient is computed and will be sent to the embedder function as a boolean variable of Humid/Dry. For having a more secure scheme, the embedder should have graded humidity to choose the drier coefficient among others when there are extra choices.

- **Allowing Free Size Requests** In current version all requests for encoding length and the matrices have size which is integer multiple of BINARY_STREAM_BLOCK_SIZE. However, for the sake of efficient usage of the embedding space, other sizes should be permitted.

  This, for example, puts limitation on header size. The header size is supposed to be a integer multiple of byte, but it leads to such a waste of capacity and is not acceptable. The inner loop of InjectMessage Function should be changed significantly.

- **Adding error correction coding** Perturbed Quantization does not offer any solution for robustness of the scheme nor does TULIPAN. A little change in a matrix header can damage all remainder data. It is essential to add error correction to the scheme to get a robust steganogrpahy.

- **Including other layers during recompression and embedding process** Current version of TULIPAN recompresses and embeds the message into the luminance layer and stores the chrominance layers in their original quality. More than wasting embedding space, the ignorance of chrominance layers, is a handle to be abused by blind steganlizers to detect the existence of hidden message. For these reasons all layers should be employed both in recompression and embedding.

- **Capsulating all file accesses in one class** Currently in TULIPAN Implementation, each class opens files in the class code, whenever needed. In contrast, in more sophisticated projects like Crypto++, all file access is done in a unique class and other classes use that class for any File I/O Operation.

- **Eliminating Optimized Number of Rows** In current version when a matrix is not full-row-rank, TULIPAN eliminates one row of it and tries again which is not optimized for all matrices. TULIPAN should compute an optimized estimation to eliminate many rows in each step.

- **Taking the benefits of the fact that most coefficients are zero, in constructing parity sequence** As most coefficients are zero, they should not waste time in constructing and waste memory on transmitting parity sequence.

- **Precomputation of Quantization Table Division** During recompression process we change the representation of coefficients from a quantization table into another one. This contains many repetitive computation, which can be computed once and stored in a table for further usages.

- **Imitating the JPEG Rounding Function** In rounding 0.5 the JPEG library uses a method that should be imitated in the embedding process. It is like random walk for up or down Rounding. It is important because we should flee from Blind Steganalyzers, in others word to forge the recompressed image as an original JPEG Image. However, it is not so critical because we use these points for our purpose and we change them randomly to stop the biases.

- **Using other random generators** Although random generator used in TULIPAN, SHA-512, is certified by NIST, it is not necessarily the best and most secure random generator. A research for choosing a more secure random generator and its implementation is needed.

- **Including ratio r in secret scheme** The ratio r as discussed in Section 2.3.2 should be known to both sides of communication and is a part of secret scheme.

- **Considering the inclusion of constant in Gaussian Elimination** It needs a statistical survey, to see if most of the time the code matrix is found to be full-row-rank or not. If it is the case, the constants of equations should be eliminated simultaneously. If it is not, they should be recomputed after the discovery of the right matrix.

- **Putting embedding function in the right class** In current version `MessageInjection` function is a member of DynamicMatrix Class for ease of access to the elements of the matrix but should be transfered to WetEncoder Class.

- **Adding interface functions to all classes** For the sake of speed we give some direct access to protected-in-nature members of classes. Such access should be controlled by employment of interface functions.

- **Using std::string instead of char*** Usage of std::string instead of char* whenever we deal with strings like key, gives more human readable code.

- **SAFETY FIRST Concerns** Includes usage of `const` type for function arguments.

- **Performance Concerns** Includes dropping unnecessary castation in `long-double` operations, usage of register variables, dropping unnecessary allocation/freeing intermediates, Omitting unnecessary BinaryStream-BinarySequence Conversion, using TotalSize for BinaryMatrix Class, Using optimal random size, better Full-Row-Rank detection algorithm, using `int` instead of `long` for logarithmic size variables, etc.

  For example, each time we make a new random matrix, we allocate a new block of memory for it. However, the memory allocation from heap is a slow process and as all of matrices

have almost the same size, they can be stored in a same block of memory. The code should be changed so that the same memory block is used for random generation.

Another critical example is that when the matrix fails the Gaussian elimination test, some initializations need not be done again and we should take them out of the loop.

It is worthy of mention that memory management is not perfect and the program isn't proved against leakage.

- **Readable Code Concerns** This item considers changing order of the functions to an appropriate order and wrapping codes inside functions.

Bibliography

[1] J. Fridrich, M. Goljan, and D. Soukal, *Perturbed quantization steganography with wet paper codes*, ACM Multimedia Workshop, Magdeburg, Germany, September 20-21 , 2004.

[2] M. Kharrazi, H. T. Sencer, N. Memon, *Benchmarking steganographic and steganalysis techniques*, Polytechnique University, Brooklyn, USA, 2004.