Analysis of Three (k, n) Secret Sharing Methods and Development of a (4, n) Method with Valid Participant Authentication, Error Detection, and 100% Repairing of Multiple Damages

Amir Narimani

Submitted to the
Institute of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
August 2015
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Serhan Çiftçioğlu
Acting Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Alexander Chefranov
Supervisor

Examining Committee

1. Prof. Dr. Marifi Güler

2. Assoc. Prof. Dr. Alexander Chefranov

3. Asst. Prof. Dr. Adnan Acan

ABSTRACT

The aim of this thesis is the analysis of three secret sharing methods and development of a new method having better features. We work on Yuan's and Chang-Chen-Wang's methods; the latter one is enhanced.

Yuan proposed two methods which use least significant bits of each pixel that is easiest way to hide a secret black-white image into multiple grayscale cover images by ± 1 operation that is difficult to detect. They are (n, n) as allowing to restore the secret from n covers out of n covers; their (2, 3) only modification is also proposed by Yuan. Chang-Chen-Wang Secret grayscale image Sharing between several grayscale cover images with Authentication and Remedy method (SSAR) has participant authentication and damaged pixels repairing properties while Yuan's methods have not these features. We implemented the algorithms and conducted experiments on them getting Peak Signal to Noise Ratio (PSNR) and Structural Similarity values similar to those obtained in the papers of Yuan and Chang-Chen-Wang. We show that SSAR may fail under made assumption of uniqueness of the covers' identifiers, is not able fake participant recognizing, and has limited by five bits out of eight (62.5%) repairing ability of one corrupted pixel. Error and fake participant detection ability is supported by 4-bit hash value. The SSAR method is (3, n) as allowing to restore a secret from any three of *n* cover images. We correct assumptions on the uniqueness of the identifiers so that SSAR works now correctly and propose (4, n) SSAR enhancement, SSAR-E, allowing 100% exact restoration of a corrupted pixel by the use of any four out of *n* covers, and recognizing a fake participant with the help of cryptographic hash functions, which have 5-bit values that allows better error detection. Also by the use of special permutation having only one loop including all the secret image pixels, SSAR-E is able restoring all the secret image damaged pixels having just one correct pixel left. The performance and size of cover images for SSAR-E are the same as for SSAR.

Keywords: Secret sharing, grayscale images, steganography, authentication, repairing.

ÖZ

Bu tezin amacı üç gizli paylaşım yöntemini ve gelişmekte olan daha iyi özelliklere sahip yeni bir metodu analiz etmektir. Biz Yuan ve Chang-Chen-Wang'ın yöntemleri üzerinde çalışmaktayız.

Yuan, farkına varılması güç olan ± 1 işlemle gizli siyah-beyaz bir görüntüyü birçok gri tonlu görüntüye en kolay şekilde, her bir pikselin en az önemli parçalarını kullanan iki farklı yöntem sunar. Bu yöntemler (n, n), sırrın n kapaklarından n kapaklarının geri kazandırılmasına izin verir, onların (2, 3) tek değişikliği de Yuan tarafından sunulmuştur. Chang-Chen-Wang gizli gri tonlu görüntü paylaşımında, bir takım gri ton kapak görüntüsü ile Belgeleme ve Çözüm yöntemi (SSAR) katılımcı belgelemesi ve zarar görmüş piksel onarım özelliği bulunurken, Yuan'ın yönteminde bu özellikler bulunmamaktadır. Biz Yuan ve Chang-Chen-Wang'ın belgelerinden elde edildiği değerlere benzeyen, algoritma ve yapılan deneylerini Zirve Sinyalinden Ses Oranına (PSNR) ve Yapısal Benzerlik'lerini baz alarak değiştirdik. SSAR'ın kapak tanımlayıcı benzersizlik varsayımına göre başarısızlığa uğrayabileceğini gösterip sahte katılımcı tanımlayamıp, bir bozulmuş pikselin onarma yeteneği bunu sekizde beş parçacık (% 62.5) oranında kısıtladı. Hata ve sahte bir katılımcı tespit etme yeteneği 4-bit karma değeri ile desteklenir. SSAR metodu (3, n) herhangi üç n kapak görüntüsünden sırrı eski haline getirmeye izin verir. SSAR'ın düzgün bir şekilde çalışması için tanımlayıcıların benzersizliği üzerindeki varsayımları düzeltir, SSAR artışını teklif (4, n) ederiz, SSAR-E n kapaklarından herhangi dördünün kullanımı ile bozuk pikselin %100 kesin yenilenmesine izin verir. ve 5-bit değere sahip kriptografik karma fonksiyonları yardımı ile sahte bir katılımcıyı tanır ve daha iyi hata bulunmasına izin verir. Ayrıca özel değişim kullanımın bütün gizli görüntü piksellerinin bulunduğu tek bir döngü ile SSAR-E zarar görmüş gizli görüntü piksellerini geride sadece doğru bir piksel bırakarak geri döndürebilir. SSAR-E'nin performansı ve kapak görüntülerinin boyutu SSAR'ınkı ile aynidir.

Anahtar Kelimeler: Gizli paylaşım, gri tonlamalı görüntü, steganografi, kimlik doğrulama, onarım.

DEDICATION

I dedicate my dissertation work to my family. A special feeling of gratitude to my loving parents for their love, support and encouragement. I also dedicate this work and give special thanks to my best friend and my wife for her contribution to the success of my life.

ACKNOWLEDGMENT

Foremost, I have to thank my family for their love and support throughout my life. A special thanks to Prof. Dr. Işık Aybay, my department chairman to provide facilities for students. I would like to sincerely thank my supervisor Assoc. Prof. Dr. Alexander Chefranov, for his guidance and support throughout this study, his comments were very beneficial and he helped me in all the time of my study and writing of this manuscript. I learned from his insight a lot. I am also grateful to express gratitude to all of the department faculty members for their help and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Secret sharing is one of the management approaches or key creation was invented by Shamir [1] and Blakly [2] as a solution to protect the secrets. Secret sharing is a technique used to hide a piece of information called “secret” and divided into several sections called “share”. In the process of secret transmission, a specific subset of the contributions shares received and the secret rehabilitated. The section which produces shares and distribute them in special way between cover messages called “Dealer”. The secret can be restored only when the shared parts are combined to each other. In other words, individual sharing will not be used alone.

Yuan’s secret sharing methods [3] work on least significant bits of each pixel of secret image which is the most popular method that used in steganography. The secret image is embedded into textured regions of each chosen cover that selected by calculating the maximum of gradient magnitude value of each cover’s pixel. To produce this amount it used Sobel operator [4]. First of all, in the beginning, least significant bits of each pixel of all covers XOR are all together. In embedding phase, checked this value with secret value, if they’re not equal the value of selected cover image increased by one or decreased. Finally, with XOR LSBs of shares we can recovered the secret image.

Chang-Chen-Wang's Secret Sharing with Authentication and Repairing (SSAR) method [5] is intended for sharing a secret grayscale image SI by $n \geq 3$ cover grayscale images {C1,..N} so that exact reconstruction of the secret is possible having any three of the cover images. It is considered as one of the best secret sharing algorithms (see [6]; Table 4, p. 186, in [7]; Tables 1, 2, pp. 1076, 1077 in [8]; p. 198 in [3]). The secret image is embedded into each *k*-th cover image in the form of a base 5 number N_{ik} obtained for *i*-th 8-bit pixel of a secret image SI (assuming linear numbering of the image pixels, SI_i, *i=1,..,S*) using a steganographic Least Significant Base 5 Digit (LSB5D) method that is similar to the Least Significant Bit (LSB) method [3] but works with base 5 digits (with possible values 0..4) contrary to LSB working with binary numbers (with possible values 0..1). So, embedding of the number N_{ik} constructed for one pixel of the secret image needs in the cover a block of pixels of size *m* which is equal to the number of digits in N_{ik} (*m=4* in SSAR). SSAR provides authentication and error detection by the use of a 4-bit hash function of the obtained for *i*-th secret image pixel authenticator au_{ik}, *i=1,..,S, k=1,..,N*. The authenticator au_{ik} (calculated for each cover image *Ck* separately with unique identifier id_k) is concatenated with its hash function value *h(i*|| au_{ik}) and used as a part of the number N_{ik} embedded in the cover, *k=1,..,N.* After extraction, the authenticity (error checking) of the extracted authenticator au'_{ik}' is checked by comparing calculated hash function value *h(i*|| au'_{ik}) versus the extracted *h(i*|| au_{ik}). Repairing ability in the SSAR method is based on the use of a complementary dual chain somehow resembling reparation facilities of DNA [9] where if a gene on one spiral is damaged, it can be restored from the dual spiral, of course, if respective gene of the dual spiral is not damaged.

The original chain is represented by a linear sequence *LS* of pixels of the secret image *SI, LS=(LSi, i=1,..,S)*, and the dual chain is represented by a sequence *LS'=(LS'i, i=1,..,S)* obtained as a permutation of *LS, LS'=perm(LS)* meaning that *LS'* consists of the same elements as *LS* does, but the order of the elements in *LS'* differs from that in *LS,* i.e. for each *LSi* their exists only one *LS'k* such that *LSi=LS'k, I,k=1,..,S*. A partner of each pixel *LSi* of the original sequence is the pixel *LS'i* from the dual chain having the same sequence number. When creating an authenticator and the number N_{ik}, five bits of the partner pixel are used. When extracting N_{ik} from the cover image, and then restoring a secret image pixel, the five bits of the partner image are restored also. So, if a secret image pixel is damaged, its five bits can be restored from its partner if the latter is not damaged. We show that SSAR algorithm has such problems as not ability to counter fake participant attack, limited opportunity for error detection based on four-bit only hash function value, only five out of eight bits recovery in the case of damaging, and is not correct under made assumption of mutually exclusive cover identifiers. We propose an enhancement of SSAR, SSAR-E, solving all the problems mentioned above and allowing also exact repairing of up to *S-1* damaged secret image pixels out of *S*. The rest of the thesis is organized as follows. Chapter 2 introduces Yuan's and Chang-Chen-Wang's methods details and discusses problem definition. In chapter 3 we will show the process of development of these methods. Analysis of the methods is conducted in chapter 4. Chapter 5 describes the proposed enhancement, SSAR-E, and proves its features. Chapter 6 concludes the thesis.

Chapter 2

SURVEY OF YUAN'S AND CHANG-CHEN-WANG'S SSAR METHODS

There are many methods to embed data inside an image. The most popular of them is the LSB method which puts information in least significant bits of the image colors. In this chapter evaluate information embedded based on two LSB methods that in this thesis, the algorithm known as the Yuan's secret sharing methods [3] and I describe another method that embed data within meaningful content of a media, the method known as the Chang-Chen-Wang's method [4].

When a file is created, some numbers of its bits are usually not usable or less important. These bytes can be changed without any significant damage imported to file. This feature helps to put information within these bytes without any person understood.

The easiest way to implement steganography is using least significant bits of each pixel which is called "LSB" method. For this purpose at first the data should be convert into binary format then embed into least significant bits of an image pixels. Of course we want desired image that does not change much.

2.1 The (n, n) Yuan's Secret Sharing Methods

Proposed LSB (n, n) Secret sharing algorithm by Yuan [3] is a sharing of a binary Secret image of $\boldsymbol{A} \in \boldsymbol{F}^{h\times w\times 1}$ in LSB Plane, n gray distinct color image of $C^0, C^1, \dots, C^{n-1} \in \boldsymbol{F}^{h\times w\times 8}$.

Where $\boldsymbol{F}^{h\times w\times m} = \left\{\boldsymbol{A}|\boldsymbol{A} = \left[a_{i,j,l}\right]_{h\times w\times m}\right\}$, h = number of row of an image, w = number of column of an image, m = number of bit of an image. We have m = 1 for binary image, m = 8 for grayscale image and m = 24 for RGB image. Secret can be obtained precisely by XOR of all LSB pages containing images with $S^0, S^1, \dots, S^{n-1}$ sharing. Let's LSB as a cover image $\left[c_{i,j,0}^{t}\right]$ that $0 \le t \le n-1$. Let binary image $\boldsymbol{B} \in \boldsymbol{F}^{h\times w\times 1}$ is calculated as follows:

$$\boldsymbol{B} = \left[c_{i,j,0}^{0}\right] \oplus \left[c_{i,j,0}^{1}\right] \oplus \cdots \oplus \left[c_{i,j,0}^{n-1}\right] = \oplus_{t=0}^{n-1}\left[c_{i,j,0}^{t}\right] \tag{2.1}$$

T selection can be made based on the measurement of certain embedded distortion. Here, the amount of gradient referred as a gain embedded distortion criteria. Gradient magnitude matrix is calculated by the Sobel Operator [4]. So we calculate T selection in (i,j) coordinates as follows:

$$\boldsymbol{T} = arg \max_{t} g_{i,j}^{t} \tag{2.2}$$

2.1.1 The (n, n) 1LSB Secret Sharing Method

The primary method of multi-cover embedded is described below.

```
Input: A binary secret image $\boldsymbol{A} \in \boldsymbol{F}^{h\times w\times 1}$ and n distinct grayscale cover images
$C^0, C^1, \dots, C^{n-1} \in \boldsymbol{F}^{h\times w\times 8}$ that $n \ge 2$.
Output: n shares $S^0, S^1, \dots, S^{n-1} \in \boldsymbol{F}^{h\times w\times 8}$
Construction: Calculate B from n covers according to (2.1)
$S^0 = C^0, S^1 = C^1, \dots, S^{n-1} = C^{n-1}$
Calculate $g^0, g^1, \dots, g^{n-1}$ from n covers, respectively
While secret bits left to embed do
        get next secret pixel $A_{i,j}$
        If $A_{i,j} \ne B_{i,j}$ then
              Calculate T at coordinates (i,j), according to (2.2)
              $S_{i,j}^{T} = \begin{cases} C_{i,j}^{T} - 1, if\ C_{i,j}^{T} = 255 \\ C_{i,j}^{T} + 1, if\ C_{i,j}^{T} = 0 \\ C_{i,j}^{T} \pm 1, othewise \end{cases}$
        end if
end while
Reveal: $\boldsymbol{A}' = \left[s_{i,j,0}^{0}\right] \oplus \left[s_{i,j,0}^{1}\right] \oplus \cdots \oplus \left[s_{i,j,0}^{n-1}\right] = (S^0\&1) \oplus \cdots \oplus (S^{n-1}\&1) = \oplus_{t=0}^{n-1} S^t\&1$
```

Figure 2.1. Pseudo code of 1LSB Secret Sharing Method

2.1.2 The (n, n) 2LSB Secret Sharing Method

More bits in gray images can be reversed by using ± 1 operator, for example $(10000000)_2 - 1 = (01111111)_2$. Therefore, the code bits can be shared on each desired bit plane of cover image. Here, another method is presented which embed code bits in cover images 2LSB (for example, First and Second LSB).

2LSB code (n, n) sharing method used for sharing 2-bit code images (four tones) $\boldsymbol{A} \in \boldsymbol{F}^{h \times w \times 2}$ which secret embedded in cover gray image 2LSB $C^0, C^1, \dots, C^{n-1} \in \boldsymbol{F}^{h \times w \times 8}$. Therefore, this secret will be reconstructed by XOR all values of 2LSBs n output shared $S^0, S^1, \dots, S^{n-1}$.

Let define 2-bit image $\boldsymbol{B} \in \boldsymbol{F}^{h \times w \times 2}$ as follows:

$$[b_{i,j,0}] = \oplus_{t=0}^{n-1}[c_{i,j,0}^t], [b_{i,j,1}] = \oplus_{t=0}^{n-1}[c_{i,j,1}^t] \tag{2.3}$$

(2.3) by applying +1 (even $c_{i,j}^t$s) or -1 (odd $c_{i,j}^t$ s) on $c_{i,j}^t$ can be reversed and $b_{i,j,1}$ can be reversed by applying +1 (odd $c_{i,j}^t$ s) or -1 (even $c_{i,j}^t$s) on $c_{i,j}^t$. So to equalize $B_{i,j}$ with $A_{i,j}$, one or two pixels of cover images should be changed in (i,j) coordinates.

The 2LSB multi-cover embeds method described below.

```
**Input:** A 2-bit secret image $\boldsymbol{A} \in \boldsymbol{F}^{h \times w \times 2}$ and n distinct grayscale cover images $C^0, C^1, \dots, C^{n-1} \in \boldsymbol{F}^{h \times w \times 8}$ that $n \geq 2$.
**Output:** n shares $S^0, S^1, \dots, S^{n-1} \in \boldsymbol{F}^{h \times w \times 8}$
**Construction:** Calculate B from n covers according to (2.1)
$S^0 = C^0, S^1 = C^1, \dots, S^{n-1} = C^{n-1}$
Calculate $g^0, g^1, \dots, g^{n-1}$ from n covers, respectively
**While** secret bits left to embed **do**
        get next secret pixel $A_{i,j}$
                $\chi = \{t | C_{i,j}^t \neq 0, C_{i,j}^t \neq 255, 0 \leq t \leq n-1\}$
        **If** $A_{i,j} \neq B_{i,j}$ **then**
            **begin case**
                **Case 1:** $a_{i,j,0} \neq b_{i,j,0}$ and $a_{i,j,1} = b_{i,j,1}$
                    Calculate T at coordinates (i,j), according to (2.2)
                $S_{i,j}^{T} = \begin{cases} C_{i,j}^{T} - 1, if\ C_{i,j}^{T}\ is\ odd \\ C_{i,j}^{T} + 1, if\ C_{i,j}^{T} is\ even \end{cases}$
            **Case 2:** $a_{i,j,0} = b_{i,j,0}$ and $a_{i,j,1} \neq b_{i,j,1}$
                **If** $\chi \neq \emptyset$ **then**
                    $\mathrm{T}_1 = arg \max_t g_{i,j}^{t}\ \ ,\ t \in \chi$
                    $S_{i,j}^{T_1} = \begin{cases} C_{i,j}^{T_1} - 1, if\ C_{i,j}^{T_1}\ is\ even \\ C_{i,j}^{T_1} + 1, if\ C_{i,j}^{T_1} is\ odd \end{cases}$
                    $\mathrm{T}_2 = arg \max_t g_{i,j}^{t}\ \ ,\ t \in \{0,1,\dots,n-1\} - \{\mathrm{T}_1\}$
                    $S_{i,j}^{T_2} = \begin{cases} C_{i,j}^{T_2} - 1, if\ C_{i,j}^{T_2}\ is\ odd \\ C_{i,j}^{T_2} + 1, if\ C_{i,j}^{T_2} is\ even \end{cases}$
                **else**
                    $S_{i,j}^{T} = \begin{cases} C_{i,j}^{T} - 2, if\ C_{i,j}^{T} = 255 \\ C_{i,j}^{T} + 2, if\ C_{i,j}^{T} = 0 \end{cases}$
                **end if**
            **Case 3:** $a_{i,j,0} \neq b_{i,j,0}$ and $a_{i,j,1} \neq b_{i,j,1}$
                **If** $\chi \neq \emptyset$ **then**
                    $\mathrm{T} = arg \max_t g_{i,j}^{t}\ \ ,\ t \in \chi$
                    $S_{i,j}^{T} = \begin{cases} C_{i,j}^{T} - 1, if\ C_{i,j}^{T}\ is\ even \\ C_{i,j}^{T} + 1, if\ C_{i,j}^{T} is\ odd \end{cases}$
                **else**
                    $\boldsymbol{T} = arg \max_t g_{i,j}^{t}$
                    $S_{i,j}^{T_1} = \begin{cases} C_{i,j}^{T_1} - 2, if\ C_{i,j}^{T_1} = 255 \\ C_{i,j}^{T_1} + 2, if\ C_{i,j}^{T_1} = 0 \end{cases}$
                    $\mathrm{T}_2 = arg \max_t g_{i,j}^{t}\ \ ,\ t \in \{0,1,\dots,n-1\} - \{\mathrm{T}_1\}$
                    $S_{i,j}^{T_2} = \begin{cases} C_{i,j}^{T_2} - 1, if\ C_{i,j}^{T_2} = 255 \\ C_{i,j}^{T_2} + 1, if\ C_{i,j}^{T_2} = 0 \end{cases}$
                **end if**
            **end case**
        **end if**
**end while**
Reveal: $\boldsymbol{A}' = (S^0\&3) \oplus \cdots \oplus (S^{n-1}\&3) = \oplus_{t=0}^{n-1} S^t\&3$
```
Figure 2.2. Pseudo code of 2LSB Secret Sharing Method

Note that for n cover images where $\chi \neq \emptyset$ is very rare. For each randomly selected color image, the probability of $C_{i,j}^{t}$ to be equal to 0 or 255 is 2^{-7}. As a result, the probability of $\chi \neq \emptyset$ will be 2^{-7n}.

In addition $P\{a_{i,j,0} = b_{i,j,0}, a_{i,j,1} \neq b_{i,j,1}\} = P\{a_{i,j,0} \neq b_{i,j,0}, a_{i,j,1} \neq b_{i,j,1}\} = 2^{-2}$

Therefore, the probability of ± 2 operator usage will be $2^{-(7n+1)}$, for example in the case of n = 4, this value is less than1.9×10^{-9}.

2.1.3 The (2, 3) Yuan's Method

Yuan proposed a (k, n) Secret Sharing scheme through the proposed (n, n) Secret Sharing methods. His method only works on special conditions that is (2, 3). For example we have three covers {C0, C1, C2} with size 512 x 512 and we want embed the secret image into these covers. In this scheme secret image should be resized to 256 x 256. To construct a (2, 3) Secret Sharing scheme we divide covers into 4 same regions. In first step we share the secret between the upper-left quadrant of C0 and C1 by the (2, 2) LSB Secret Sharing scheme, in second step we share the secret between the upper-right quadrant of C0 and C2 by the (2, 2) LSB Secret Sharing scheme, in third step share the secret between the lower-left quadrant of C1 and C2 by the (2, 2) LSB Secret Sharing scheme and finally Share the secret between the lower-right quadrant of C0 and C1 and C2 by the (3, 3) LSB SS scheme. After embedding we will have three shares {S0, S1, S2} and we can reconstruct a secret from these shares.

2.2 The Chang-Chen-Wang's SSAR Method

The SSAR method takes as input the secret image, *SI*, which may be represented as a sequence of byte-size pixels, *LSi, i=1,..,S*, and a set of cover images, *Ci*, *i=1,..,n >=3*. Each cover image, *Ci*, has a unique identifier, id_i. A key *K*, defines a permutation of *LS*. SSAR has embedding, extraction, and repairing parts. The repairing part is used when errors are detected in the extraction part (replaces a damaged bit by five its most significant bits obtained when restoring its co-partner pixel). This part according to SSAR shall use the secret key *K* but if a valid user repairs the image then the use of the key is meaningless, but if it is made by an illegal user, it is also meaningless because the image is already disclosed. That's why the secret key in the permutation is redundant.

In the following, we give details of the embedding, recovering, and repairing parts, and provide a numerical example of SSAR usage.

SSAR Embedding Part (Steps 1-7):

Step 1. Producing a dual sequence *LS' = permutation (K, LS)*. Denote bits of *LSi* as $a_{1i},..,a_{8i}$, and of its partner LS'_i *as* $b_{1i},..,b_{8i}$, $i=1,..,S$.

Step 2 for each *i*, define three quantities,

$$\alpha_i = \sum_{k=1}^{3} a_{ki} 2^{3-k} \in \{0,..,7\}, \beta_i = \sum_{k=1}^{3} a_{k+3,i} 2^{3-k} \in \{0,..,7\}, \gamma_i = \sum_{k=1}^{2} a_{k+6,i} 2^{2-k} \in \{0,..,3\}$$

$$,i=1,..,S, \tag{2.4}$$

using the secret image chain, *LS*.

Step 3. Concatenate the bits *b1i,..,b4i*, of the partner pixel, *LS'i* with the three quantities (2.4), getting

$$\alpha_i^{'} = \alpha_i + 8b_{1i} \in \{0,..,15\}, \beta_i^{'} = \beta_i + 8b_{2i} \in \{0,..,15\}, \gamma_i^{'} = \gamma_i + 8b_{3i} + 4b_{4i} \in \{0,..,15\}$$

$$,i=1,..,S. \tag{2.5}$$

Step 4. Using (2.5), calculate an authenticator,

$$au_{ik} = (\alpha_i^{'} + \beta_i^{'} \cdot id_i + \gamma_i^{'} \cdot id_i^2) \bmod 17 \in \{0,..,16\} \tag{2.6}$$

For each pixel and each cover, *i=1,..,S, k=1,..,n*. Thus, for each pixel, we get a system of *n>= 3* equations that allows solving them with respect to $\alpha_i^{'}, \beta_i^{'}, \gamma_i^{'}$ after embedding of the authenticators into the cover images by a sender, and the next extraction of the authenticators from the cover images by a receiver.

Step 5. For each authenticator (2.6), au_{ik}, calculate its 4-bit hash function value,

$$hv_{ik} = hash(i \,\|\, au_{ik}) \in \{0,..,15\}, \text{ i=1,..,S, k=1,..,n.} \tag{2.7}$$

Step 6. Using (2.6), (2.7), generate a number

$$N_{ik} = au_{ik} \cdot 34 + hv_{ik} \cdot 2 + b_{5i} \in \{0,..,575\} \qquad (2.8)$$

By mixing an authenticator, its hash, and 5-th bit of *LS'i* in a way allowing getting back its constituent parts. The number can be represented as a four-digit base-5 number.

Step 7. Each cover image C_k is represented as a sequence of *S* 4-pixel blocks so that N_{ik} in the form of four base 5 digit number, $N_{ik}[1..4]$, is embedded into four consecutive pixels *Ck[4(i-1)+1,..,4i]*:

$C_k[4(i-1)+j]' = C_k[4(i-1)+j] - C_k[4(i-1)+j] mod 5 + N_{ik}[j]$, (2.9)

j=1,..,4, i=1,..,S, k=1,..,n.

Embedding in (C.-C. Chang, 2011), see Fig. 4 therein, is done in slightly more complicated way than (2.9) but mainly it complies with (2.9).

Consider now SSAR extraction part that takes *n* covers with embedded by (2.9) numbers N_{ik} representing parts of the secret image *SI* and restores them and their ingredients; restored values have *R* in their names.

SSAR Extraction Part (Steps 1-6):

Step 1. Restore digits of the embedded number using (2.9)

$$NRik[j] = Ck'[4(i-1)+j] \text{ mod } 5, j=1,..,4, i=1,..,S, k=1,..,n. \qquad (2.10)$$

Step 2. Restore constituent parts of the numbers (2.10) using (2.8)

$$auR_{ik} = NR_{ik} div 34; hvR_{ik} = (NR_{ik} \bmod 34) div 2; bR_{5i} = NR_{ik} \bmod 2. \qquad (2.11)$$

Step 3. Check errors (authenticity) of the restored by (2.11) authenticator by recalculating of the hash of the restored authenticator, auR_{ik} and comparing it versus its restored hash function value:

$$hash(i, auR_{ik}) == hvR_{ik} \qquad (2.12)$$

If (2.12) is true then the authenticator is considered valid, otherwise an error is detected.

Step 4. If checking in (2.12) of Step 3 is true (no error) for any three covers, then three entities, $\alpha R_i^{'}, \beta R_i^{'}, \gamma R_i^{'}$, used in calculation of authenticators (2.6) are restored from (2.6) by solving a system of at least three linear modulo 17 algebraic equations with respect to three unknowns.

Step 5. Having restored $\alpha R_i^{'}, \beta R_i^{'}, \gamma R_i^{'}$, values αR_i , βR_i , γR_i together with $bR_{1i},..,bR_{4i}$ from (2.5) can be restored as follows

$$\begin{aligned}&\alpha R_i = \alpha R_i^{'} \bmod 8, \beta R_i = \beta R_i^{'} \bmod 8, \gamma R_i = \gamma R_i^{'} \bmod 4, bR_{1i} = \alpha R_i^{'} div 8, bR_{2i} = \beta R_2^{'} div 8,\\ &bR_{3i} = \gamma R_i^{'} div 8, bR_{4i} = (\gamma R_i^{'} div 4) \bmod 2\end{aligned} \qquad (2.13)$$

Step 6. From (2.13), the original pixel is restored using (2.4)

$$LS_i = 32 \cdot \alpha R_i + 4 \cdot \beta R_i + \gamma R_i . \qquad (2.14)$$

Obtained in (2.11) and (2.13) five bits $bR_{1i},..,bR_{5i}$ of the partner pixel *LS'i* can be used for its restoration in the case if it is damaged. Let's illustrate SSAR by following Example 1.

Example 1. Let the secret image size *S=6*, secret image sequence of pixels is *LS=(15,34,49,105,217,28)*, permutation is *(4,2,1,3,6,5), LS'=(105,34,15,105,28,217)*. Thus, the partner for LS_1 is LS_4, for LS_2 is LS_2, for LS_3 and LS_1, for LS_4 is LS_3, for LS_5 is LS_6, and for LS_6 is LS_5. Thus, if LS_4 is corrupted, its 5 bits may be obtained from the information obtained when restoring LS_1 whose partner is LS_4. Let the number of cover images *N=5*, each of them having *4S=24* pixels. Consider embedding of the pixel *LS1=15* into the 5 cover images. SSAR Embedding Part Step 1 is already applied. According to Step 2 equations (2.4), $\alpha_1 = 0, \beta_1 = 3, \gamma_1 = 3$. Using (2.14), we see that actually $LS_1=32*0+4*3+3=15$. According to Step 3 equations (2.5), and taking into account that *LS₄=105=(1100 1001),* we get $\alpha_1^{'} = 8, \beta_1^{'} = 11, \gamma_1^{'} = 3$. From the last values, according to (2.13), we can get back correct values $\alpha_1 = 0, b_{11} = 1, \beta_1 = 3, b_{21} = 1, \gamma_1^{'} = 3, b_{31} = 0, b_{41} = 0$. Calculate now authenticators of LS_1 pixel according to (2.6) using covers' identifiers as $id_1=1$, $id_2=2$, $id_3=3$, $id_4=4$, $id_5=5$:

$$\begin{aligned} &au_{11} = (8 + 11 \cdot 1 + 3 \cdot 1) \bmod 17 = 5; au_{12} = (8 + 11 \cdot 2 + 3 \cdot 4) \bmod 17 = 8; \\ &au_{13} = (8 + 11 \cdot 3 + 3 \cdot 9) \bmod 17 = 0; au_{14} = (8 + 11 \cdot 4 + 3 \cdot 16) \bmod 17 = 15; \\ &au_{15} = (8 + 11 \cdot 5 + 3 \cdot 25) \bmod 17 = 2; \end{aligned} \tag{2.15}$$

If any three of the authenticators (2.15) are available, we can get $\alpha_1^{'}, \beta_1^{'}, \gamma_1^{'}$ by solving a system of equations (2.6). Say, if the first three authenticators are restored then we have the following system:

$$\begin{aligned} au_{11} &= 5 = (\alpha_1^{'} + \beta_1^{'''} \cdot 1 + \gamma_1^{'} \cdot 1) \bmod 17 \\ au_{12} &= 8 = (\alpha_1^{'} + \beta_1^{'''} \cdot 2 + \gamma_1^{'} \cdot 4) \bmod 17 \\ au_{13} &= 0 = (\alpha_1^{'} + \beta_1^{'''} \cdot 3 + \gamma_1^{'} \cdot 9) \bmod 17 \end{aligned} \tag{2.16}$$

The systems (2.16) has Vandermonde matrix of the coefficients and may be solved, e.g., using Cramer's rule [10] (dividing the determinant of the matrix of the system coefficients with substituted column of the coefficients with the column of the known values by the determinant of the coefficients matrix):

$$\begin{aligned} \alpha_1^{'} &= \det\begin{vmatrix} 5 & 1 & 1 \\ 8 & 2 & 4 \\ 0 & 3 & 9 \end{vmatrix} / \det\begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{vmatrix} = \frac{-1}{2} = \frac{16}{2} \bmod 17 = 8 \\ \beta_1^{'} &= \det\begin{vmatrix} 1 & 5 & 1 \\ 1 & 8 & 4 \\ 1 & 0 & 9 \end{vmatrix} / \det\begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{vmatrix} = \frac{39}{2} = \frac{5}{2} \bmod 17 = 5 \cdot 2^{-1} \bmod 17 = 5 \cdot 9 \bmod 17 = 11 \\ \gamma_1^{'} &= \det\begin{vmatrix} 1 & 1 & 5 \\ 1 & 2 & 8 \\ 1 & 3 & 0 \end{vmatrix} / \det\begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{vmatrix} = \frac{-11}{2} = \frac{6}{2} \bmod 17 = 3 \end{aligned} \tag{2.17}$$

As far as the identifiers appearing in (2.6) are mutually exclusive, Vandermonde determinant (Vandermonde matrix) used in denominators of (2.17) is non-zero, and the solution exists. As far as the calculations are done modulo 17, the Vandermonde determinant also may be equal to zero if any two identifiers are congruent modulo 17 but it is not the case for the current example. We see that (2.17) returns back correct $\alpha_1^{'}, \beta_1^{'}, \gamma_1^{'}$ values. Next, in Step 5, according to (2.7), hash function values are calculated for the authenticators. In [5], equations (2.9)-(2.11), define how 4-bit resulting hash function value is to be calculated using some generic hash function. For simplicity and getting particular results, let hash function be $h(x) = (13x + 11) \bmod 16,.$ Then, according to (2.7) in Step 5,

$$\begin{aligned}
hv_{11} &= h(1 \| au_{11}) = h(1 \| 5) = h(001 \| 00101) = h(00100101) = h(37) = 12, \\
hv_{12} &= h(1 \| au_{12}) = h(1 \| 8) = h(001 \| 01000) = h(00101000) = h(40) = 3, \\
hv_{13} &= h(1 \| au_{13}) = h(1 \| 0) = h(001 \| 00000) = h(00100000) = h(32) = 11, \\
hv_{14} &= h(1 \| au_{14}) = h(1 \| 15) = h(001 \| 01111) = h(00101111) = h(47) = 14, \\
hv_{15} &= h(1 \| au_{15}) = h(1 \| 2) = h(001 \| 00010) = h(00100010) = h(34) = 5.
\end{aligned} \tag{2.18}$$

Now, let us apply Step 6 using (2.8), (2.15), and (2.16) and taking into account that LS_4=($b_{11}..b_{81}$) = (1100 1001), we get the numbers for embedding and their representation as 4-digit base 5 numbers:

$$\begin{aligned}
N_{11} &= au_{11} \cdot 34 + hv_{11} \cdot 2 + b_{51} = 5 \cdot 34 + 12 \cdot 2 + 1 = 195 = 1240 \\
N_{12} &= au_{12} \cdot 34 + hv_{12} \cdot 2 + b_{51} = 8 \cdot 34 + 3 \cdot 2 + 1 = 279 = 2104 \\
N_{13} &= au_{13} \cdot 34 + hv_{13} \cdot 2 + b_{51} = 0 \cdot 34 + 11 \cdot 2 + 1 = 23 = 0043 \\
N_{14} &= au_{14} \cdot 34 + hv_{14} \cdot 2 + b_{51} = 15 \cdot 34 + 14 \cdot 2 + 1 = 539 = 4124 \\
N_{15} &= au_{15} \cdot 34 + hv_{15} \cdot 2 + b_{51} = 2 \cdot 34 + 5 \cdot 2 + 1 = 79 = 0304
\end{aligned} \tag{2.19}$$

Let us assume that the first four bytes of the five cover images where the numbers (2.19) are to be embedded are as follows:

C_1=(12,240,251,128,...), C_2=(137,49,56,122,..), C_3=(17,1,67,78,..),

C_4=(190,217,218,16,..), C_5=(123,213,65,41,..) (2.20)

From (2.19), (2.20), according to (2.9) in Step 7,

C_1=(11,242,254,125,...), C_2=(137,46,55,124,..), C_3=(15,0,69,78,..), C_4=(194,216,217,19,..), C_5=(120,213,65,44,..) (2.21)

For the Step 1 of the SSAR Extraction Part, applying modulo 5 operations to (2.21), we restore the 4-digit base 5 numbers (2.19), 1240=195, 2104=279, 0043=23, 4124=539, 0304=79. From (2.19), applying Step 2 equations (2.21), we restore authenticators (2.15), *5, 8, 0, 15, 2*, their hash function values (2.18), 12,3,11,14,5, and b_{51} =1. Checking of (2.12) in Step 3 returns all true values, hence, any three of the five authenticators obtained may be used for restoring $\alpha_1^{'} = 8, \beta_1^{'} = 11, \gamma_1^{'} = 3$, as it is made in (2.16) for the first three authenticators in accordance with Step 4. And lastly, the original values $\alpha_1 = 0, \beta_1 = 3, \gamma_1 = 3$ are restored according to Step 5 equation (2.13) together with $(b_{11}, b_{21}, b_{31}, b_{41}) = (1100)$.

Thus, we see how the SSAR method works for embedding a secret image pixel and five bits of its partner into four consecutive pixels of the cover images. If in our example the partner pixel after extraction is found out in the checking (2.12) to be incorrect, its five most significant bits can be gained from the results of extraction of LS1 containing in the case under consideration five correct most significant bits of *LS_4*.

2.3 Steganography System Quality Evaluation Metrics

In order to evaluate the image steganography performance, some quality measurements such as SNR, PSNR and MSE are presented.

Mean Square Error (MSE) defined as mean squares differences between the original image and image after embedding. This measure calculated as follows.

$$\mathrm{MSE} = 1/XY \left[\sum_{i=1}^{X}\sum_{j=1}^{Y}(c(i,j) - e(i,j))^2\right] \quad (2.22)$$

Where X and Y are the length and width of the image, respectively. C(i,j) is the original image pixel value and e(j,j) is steganography image pixel value within (i,j).

Signal to noise ratio (SNR) measures image sensitivity. This criteria measures the original strength relative to background noise. SNR value can be obtained by:

$$SNR_{dB} = 10\log_{10}\left(\frac{P_{signal}}{P_{noise}}\right) \quad (2.23)$$

Signal peak to noise ratio (PSNR) determines the amounts of damages imported to image by embedded data and calculated as follows:

$$PSNR = 10\log_{10}(L * L/MSE) \quad (2.24)$$

Where L is the image signal peak which is equal to 255 for an 8-bit image.

2.4 Problem Definition

Yuan’s method use binary image as a secret image and embed it into grayscale covers but Chang-Chen-Wang’s method use grayscale image. Chang-Chen-Wang’s method has an ability that can be repair the images after tampering by hackers and for detecting errors used a hash function but Yuan’s method doesn’t have these features. Chang-Chen-Wang’s method claiming to have participant authentication and damaged pixels repairing properties. Chang-Chen-Wang’s method cannot repair corrupted pixel fully, because is limited by five bits out of eight (62.5%) repairing ability of one corrupted pixel. Error detection ability is supported by 4-bit hash value but it doesn’t use cryptographic hash functions also this method cannot restore secret image when a lot of damaged pixels. In chapter 5 we proposed a method that it has the ability to recover fully by eight bits, and with using cryptographic hash functions, which have 5-bit values that allows better errors detection and by use of special permutation having only one loop including all the secret image pixels, new method is able restoring all the secret image damaged pixels having just one correct pixel left.

Chapter 3

IMPLEMENTATION OF THE YUAN'S AND CHANG-CHEN-WANG'S SSAR METHODS

At first, Yuan's method [3] is investigated in 1LSB and 2LSB normally. Then implement Chang-Chen-Wang's method and shown how it works.

3.1 1LSB (n, n) Secret Sharing Yuan's Method

Yuan's method includes both 1LSB and 2LSB. In 1LSB, Appendix A, least significant bit is used for each pixel. At first cover images and secret image must be converted to a set of bits matrix so that least significant bit in each pixel is used. We created a function that convert each double value into binary value.

According to (2.3), the XOR resulting of all LSBs covers are calculated as B value. In this function each LSB values of covers should be XOR together. For each cover, gradient magnitude is obtained by Sobel operator and for embeds each pixel of secret image. For calculating gradient magnitude we used already function in MATLAB.

A comparison between the secret image and obtained value of B in advance will be done. If these two pixels were not equal, according to embedding algorithm and based on g value, at first phase, cover number and pixel value related to that cover earned, and if its value was equal to 255, a one unit subtracted from it and if it was equal to zero, a one unit will be added, otherwise, one unit added or subtracted randomly and this process continued until all pixels within the secret image are checked. All covers and gradient magnitude obtained used as input parameters of this function.

```
1  for i=1:r
2     for j=1:c
3         if(A(i,j)~=B(i,j))
4             [t,T]= max([g1(i,j),g2(i,j),g3(i,j),g4(i,j)]);
5             if(CT(i,j,T)==255)
6                 ST(i,j) = CT(i,j,T)-1;
7                 Embeding_map(i,j,T) = 0;
8             elseif(CT(i,j,T)==0)
9                 ST(i,j,T) = CT(i,j,T)+1;
10                 Embeding_map(i,j,T) = 0;
11             else
12                 if(rand(1)>0.5)
13                     ST(i,j,T) = CT(i,j,T)-1;
14                     Embeding_map(i,j,T) = 0;
15                 else
16                     ST(i,j,T) = CT(i,j,T)+1;
17                     Embeding_map(i,j,T) = 0;
18                 end
19             end
20         end
21     end
22 end
```

In this function at line 4, at first we found maximum of g value in pixel (i, j) and we got T that is the number of cover we want embed our secret image value into it. Then we checked the cover value, if is equal to 255 then decrease one unit from the value and if is equal to 1, one unit increased, otherwise we use a random function for generating a random value from +1 and -1. Also here for drawing embedding map if any value of covers changed, we put zero into this matrix then draw it as a plot in MATLAB. Finally, to recover secret image, all least significant bit value of embedded images must be XOR to each other. After recovering secret image we calculate PSNR and MSSIM for each embedded images. For MSSIM we used already function in MATLAB but for calculating PSNR we used instructions of calculate PSNR in Chang-Chen-Wang's paper [5].

```
function My_psnr=My_PSNR(I,J)
   X = double(I);
   Y = double(J);

   MSE = sum((X(:)-Y(:)).^2) / prod(size(X));
   My_psnr = 10*log10(255 * 255/MSE);
end
```

Here we explain the special case of (n, n) of 1LSB and 2LSB methods that n = 2 for 1LSB and n = 4 for 2LSB. For this purpose, 512 × 512 sized images are used. Also a two-tone image (binary) and a four-tone 512 × 512 image are used. Examples of these images are shown below.

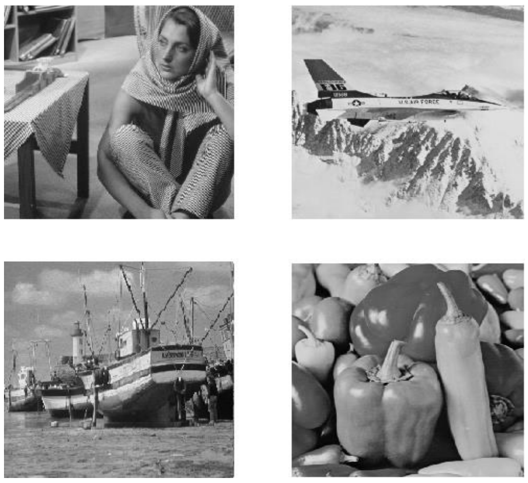


Figure 3.1. Grayscale Cover Images

A sample of images used as cover images in simulations is shown. As shown in the figure 3.1, gray images (8-bit) are used for this purpose. As explained above, with the purpose of separation, two one-bit and two-bit images are used as a secret image that shown in following figure.

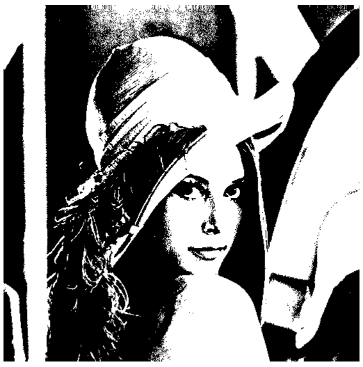


Figure 3.2. Secret Image (Two Tone)

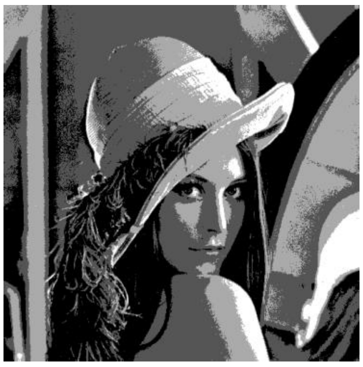


Figure 3.3. Secret Image (Four Tone)

At first 1Bit LSB Secret Sharing method were evaluated. See implementation results of this algorithm for (2, 2) model as shown below.

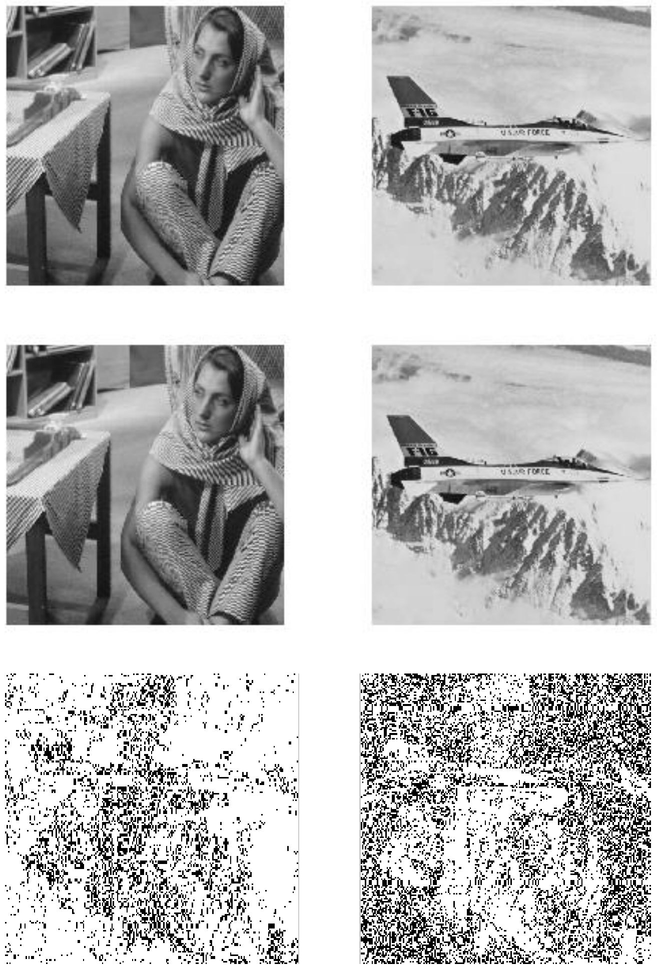


Figure 3.4. Results of 1LSB (2, 2) – first row is cover images – second row is embedded images – third row is embedding map of each images

The amounts reported for PSNR in two output images (55.70 dB and 53.04 dB) indicates that embedded data which using this algorithm could not make drastic changes in output image quality such that there is not any appearance change to be recognized. Also structural similarity mean value (SSIM) of output cover images in comparison to original image are very close to one (0.999 and 0.998) which acknowledge it.

After this stage, secret extracted from the input cover images. Input binary image and extracted binary image are shown.

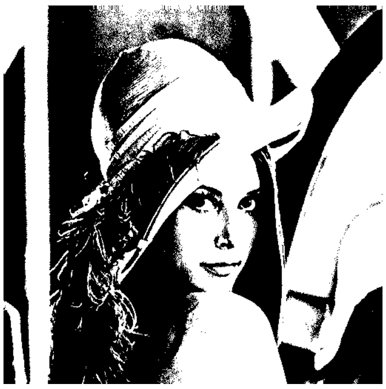
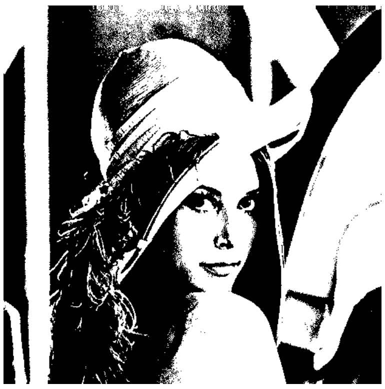



Figure 3.5. Recovered 2-tone image

3.2 2LSB (n, n) Secret Sharing Yuan's Method

In 2LSB, Appendix B, all of functions are similar to 1LSB just one function is different. In embedding phase ate first value X should be calculated then if two pixels were not equal, we have three cases. In case 1 if LSB_1 of two matrices are not equal and LSB_2 of these matrices are equal, according to embedding algorithm and based on g value, we use some instructions for embedding this value into cover selected and another cases we have another instructions. This process continued until all pixels within the secret image are checked. Finally, to recover secret image, all least significant bit value of embedded images must be XOR to each other. After recovering secret image we calculate PSNR and MSSIM for each embedded images.

```
for i=1:r
   for j=1:c
       X = find_X(CT(i,j,:));

%******************************************************************
       %case1:

%******************************************************************
       if(a(i,j,1)~=B(i,j,1) && a(i,j,2)==B(i,j,2) )
              [ST,Embeding_map] =
              mbed_case1(ST,gT,CT,i,j,Embeding_map);

%*****************************************************************
       %case2:

%*****************************************************************
        elseif(a(i,j,1)==B(i,j,1) && a(i,j,2)~=B(i,j,2) )
            [ST,Embeding_map]=
            embed_case2(ST,gT,CT,X,i,j,Embeding_map);

%*****************************************************************
        %case3:

%*****************************************************************
        elseif(a(i,j,1)~=B(i,j,1) && a(i,j,2)~=B(i,j,2) )
            [ST,Embeding_map] =
             embed_case3(ST,gT,CT,X,i,j,Embeding_map);
        end
    end
end
```

For embedding the secret image value in this algorithm, we have three cases. Like previous algorithm we have some conditions that the value of secret image and B that calculate by equation (2.3) are not equal. We should find all of covers number that values of these covers are not equal to 255 or 0, then we will find g from these numbers just in case 2 and 3.

```
1  function [ST,Embeding_map] =
2  embed_case1(ST,gT,CT,i,j,Embeding_map)
3  [t, T ]= max([gT(i,j,1),gT(i,j,2),gT(i,j,3),gT(i,j,4)]);
4  if(mod(CT(i,j,T),2)==0)
5     ST(i,j,T) = CT(i,j,T)+1;
6     Embeding_map(i,j,T) = 0;
7  else
8     ST(i,j,T) = CT(i,j,T)-1;
9     Embeding_map(i,j,T) = 0;
10 end
```

In case 1 and line 3 we found maximum of gradient magnitude value and put it into T value. Then we checked is odd or even, if it was even we added 1 to this value otherwise subtracted by 1.

```
function [ST, Embeding_map] =
embed_case2(ST,gT,CT,X,i,j,Embeding_map)
if ~(isempty(X))
   [aa, bb ]= sort( gT( i,j, X(1:length(X)) ) ,'descend' );
   T1= X(bb(1));

   if(mod(CT(i,j,T1),2)==0)
       ST(i,j,T1) = CT(i,j,T1)-1;
       Embeding_map(i,j,T1) = 0;
    else
        ST(i,j,T1) = CT(i,j,T1)+1;
        Embeding_map(i,j,T1) = 0;
    end
    [aa, bb ]= sort( gT( i,j, : ) ,'descend' );
    T2 =bb(1);
    if(T2==T1)
        T2 = bb(2);
    end
    if(mod(CT(i,j,T2),2)==0)
        ST(i,j,T2) = CT(i,j,T2)+1;
        Embeding_map(i,j,T2) = 0;
    else
        ST(i,j,T2) = CT(i,j,T2)-1;
        Embeding_map(i,j,T2) = 0;
    end
else
    [t, T ]= max([gT(i,j,1),gT(i,j,2),gT(i,j,3),gT(i,j,4)]);
    if( CT(i,j,T)==0 )
        ST(i,j,T) = CT(i,j,T)+2;
        Embeding_map(i,j,T) = 0;
    elseif( CT(i,j,T)==255 )
        ST(i,j,T) = CT(i,j,T)-2;
        Embeding_map(i,j,T) = 0;
    end
end
```

For case 2 and 3 we act according to Fig 2.2 in chapter 2. At first we need find X, then obtained g meets X, then we checked this value is odd or even and act according to the algorithm.

After 1 Bit LSB SS algorithm analyzed, 2Bit LSB algorithm performance will be evaluated. Implemented result of this algorithm for (4, 4) model shown.

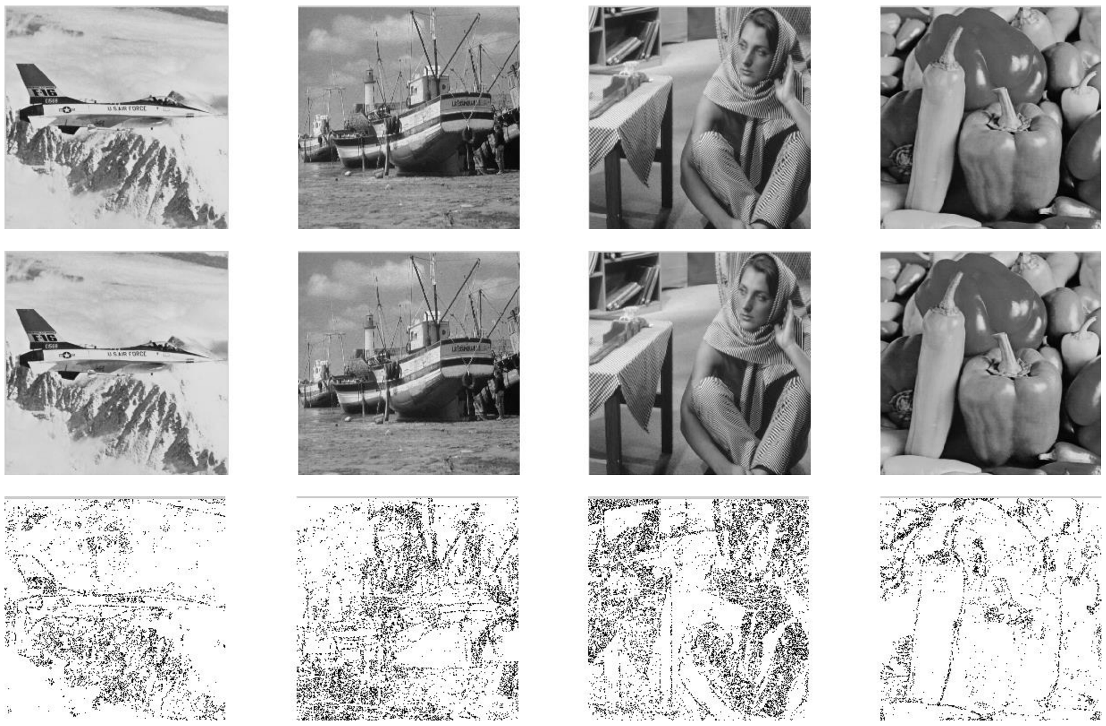


Figure 3.6. Results of 2LSB (4, 4) – first row is Cover images – second row is Embedded images – thirs row is Embedding map of each images

The above amounts reported for PSNR in four output images (56.46 dB, 52.63 dB, 55.59 dB and 53.1 dB) indicates that embedded data which using this algorithm could not make drastic changes in quality of output image such that there is not any appearance change to be recognized. Also the structural similarity mean value (MSSIM) of output cover images in comparison to original image are very close to one (0.999) which acknowledge it.

After this stage, code images in cover image are extracted and input binary image with extracted output binary image are shown here.

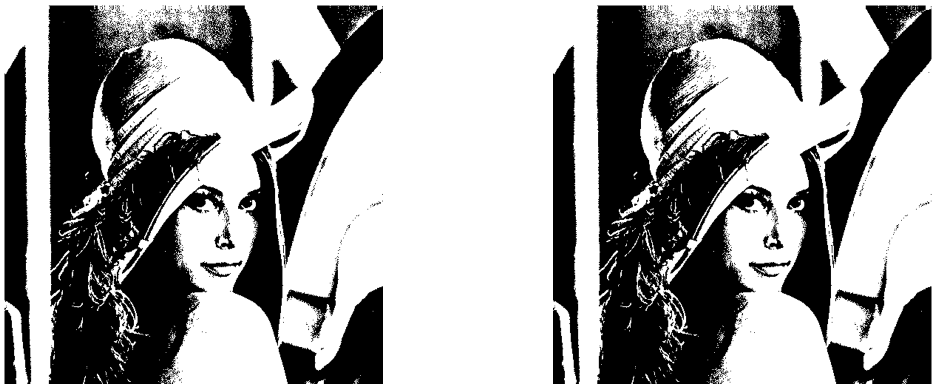


Figure 3.7. Recovered Four tone Images

3.3 Chang-Chen-Wang's SSAR Method

In this method, Appendix C, at first we need shuffled pixel values and we used Henon_Map function for this that we can see the instruction following. We used a key for shuffling that we define at the first of codes.

```
function shuffled_index = Henon_map(count, key)
   x = zeros(1, count+1);
   x(1) = 0.8912;
   y = zeros(1, count+1);
   y(1) = 1;
   a = key;
   b = 0.1;
   for i = 2:count+1
      x(i) = 1 - a * x(i-1)^2 + y(i-1);
        y(i) = b * x(i-1);
    end

    [tmp, shuffled_index] = sort(x(2:count+1));
end
```

Then we should obtain the authentication code that in chapter 2 we describe this that how we can calculate it. This function shown the instructions.

```
[hd, wd] = size(SI);
S = hd*wd;
[X, Y] = Permutation_phase(hd*wd, K);
[LS] = SI(X);
[LS_prim] = SI(Y);
% -------------- Three Quantities --------------
alpha = bitget(LS, 8) * 4 + bitget(LS, 7) * 2 + bitget(LS, 6);
beta = bitget(LS, 5) * 4 + bitget(LS, 4) * 2 + bitget(LS, 3);
```

```
9  gama = bitget(LS, 2) * 2 + bitget(LS, 1);
10
11 alpha_prim = alpha + 8 * bitget(LS_prim, 8);
12 beta_prim = beta + 8 * bitget(LS_prim, 7);
13 gama_prim = gama + 8 * bitget(LS_prim, 6) + 4 *
14 bitget(LS_prim,5);
15
16 b5y = bitget(LS_prim, 4);
17 % ------------- Authenticator, Hash Function and Hidden Data ----
18 for i=1:S
19    for k=1:n
20        au(:,i,k) = mod(int16(alpha_prim(i)) + int16(beta_prim(i))
21        * id(k) + int16(gama_prim(i)) * (id(k) ^ 2), 17);
22
23        str = strcat(dec2bin(i,3),dec2bin(au(:,i,k),5));
24        temp = string2hash(str);
25        hv(:,i,k) = bitget(temp, 5) * 8 + bitget(temp, 6) * 4 + …
26        bitget(temp, 7) * 2 + bitget(temp, 8);
27
28        b5y(:,i,k) = b5y(i);
29    end
30 end
31 %--------------- Embedding Phase ---------------------
32 N = au * 34 + int16(hv) * 2 + int16(b5y);
33
34 for i=1:n
35    N_five_base = dec2base(N(:,:,i), 5, 4);
36    Covers_prim_tmp = embedding_algorithm(N_five_base,
37    Covers(:,:,i));
38    Covers_prim(:,:,i) = reshape(Covers_prim_tmp,(hd*2),(wd*2));
39 end
```

In line 7-9 we can calculate three numbers α, β, γ and in line 11-14 obtain α', β', γ' then in line 20 we calculate au for any covers, finally we can get N value. Before embedding we should convert N value into 4-digit of base 5 of N, then embed each digit into covers that we see the corresponding code below.

```
1  Nx1 = N_five_base(:, 1) - '0';
2  Nx2 = N_five_base(:, 2) - '0';
3  Nx3 = N_five_base(:, 3) - '0';
4  Nx4 = N_five_base(:, 4) - '0';
5
6  [ht wd] = size(Cover);
7  S = (ht*wd)/4;
8
9  for i=1:S
10    for j=1:4
11          switch j
12                case 1
13                      Cover_prim(4*(i-1)+j) = Cover(4*(i-1)+j) -
14                      mod(Cover(4*(i-1)+j),5) + Nx1(i);
15                case 2
```

```
16                      Cover_prim(4*(i-1)+j) = Cover(4*(i-1)+j) -
17                      mod(Cover(4*(i-1)+j),5) + Nx2(i);
18                case 3
19                      Cover_prim(4*(i-1)+j) = Cover(4*(i-1)+j) -
20                      mod(Cover(4*(i-1)+j),5) + Nx3(i);
21                case 4
22                      Cover_prim(4*(i-1)+j) = Cover(4*(i-1)+j) -
23                mod(Cover(4*(i-1)+j),5) + Nx4(i);
24          end
25    end
26 end
```

In line 12-23 we embedded each digit into covers, we used two for loops that one loop start from 1 to S that is number of secret pixels and another one start from 1 to 4 that is number of digits of each N value. At the result of this procedure we get an image with some blocks with size 4. In reconstruction phase, before participants join the truth of their own shares can be verified and in this phase we can check whether the share is authentic.

```
for k=1:n
   C = Covers_prim(:,:,k);
   for i=1:S
         str ='';
         for j=1:4
              NR_tmp(j) = mod(C(4*(i-1)+j),5);
              str = strcat(str,num2str(NR_tmp(j)));
         end

         NR(:,i,k) = base2dec(str, 5);
         auR(:,i,k) = floor(NR(:,i,k)/(34));
         hvR(:,i,k) = floor(mod(NR(:,i,k), 34)/2);
         b5yR(:,i,k) = floor(mod(NR(:,i,k),2));
   end
end
% ---------------------- Check errors (authenticity) -----------
for k=1:n
    hv(:,:,k) = generate_ax(auR(:,:,k), 1:size(auR,2));

    if (hv(:,:,k) == hvR(:,:,k))
       authenticated(k) = k;
    end
    ER(:,:,k) = reshape(255*(hv(:,:,k) == hvR(:,:,k)), ht/2,
    wd/2);
end
```

Here, authentication code re-calculated and checked with first authenticator. If they are equal, this pixel is authentic pixel otherwise is inauthentic. Finally, in reconstruction phase, at least we need three legal participants to join together that can be reconstruct the secret image and we used Cramer's rule for solving this system. Corresponding code:

```
A(:,1) = [1 1 1];
A(:,2) = delta;
A(:,3) = delta .^ 2;

for i=1:S
   b(:,1) = au(1,i,delta);

   A1 = A;A1(:,1)=b;
   A2 = A;A2(:,2)=b;
   A3 = A;A3(:,3)=b;
   D = int16(det(A));
   D1 = int16(det(A1));
   D2 = int16(det(A2));
   D3 = int16(det(A3));

   alpha_prim(i) = mod(int16(mod(D1 * mulinv(D,17),17)),17);
   beta_prim(i) = mod(int16(mod(D2 * mulinv(D,17),17)),17);
   gama_prim(i) = mod(int16(mod(D3 * mulinv(D,17),17)),17);
 end
```

For Chang-Chen-Wang's method four 512 × 512 cover images are used. Also a 256 × 256 image for secret image is used. Examples of these images are shown below.

Figure 3.8. Secret Image

The amounts reported for PSNR in four output images are (42.04 dB, 42.01 dB, 42.06 and 42.04) indicates that embedded data which using this algorithm not good same as

Yuan's method [3] and output image quality lower than Yuan's method [3]. Also structural similarity mean value (SSIM) of output cover images in comparison to original image are (0.797, 0.798, 0.789 and 0.788) which acknowledge it.

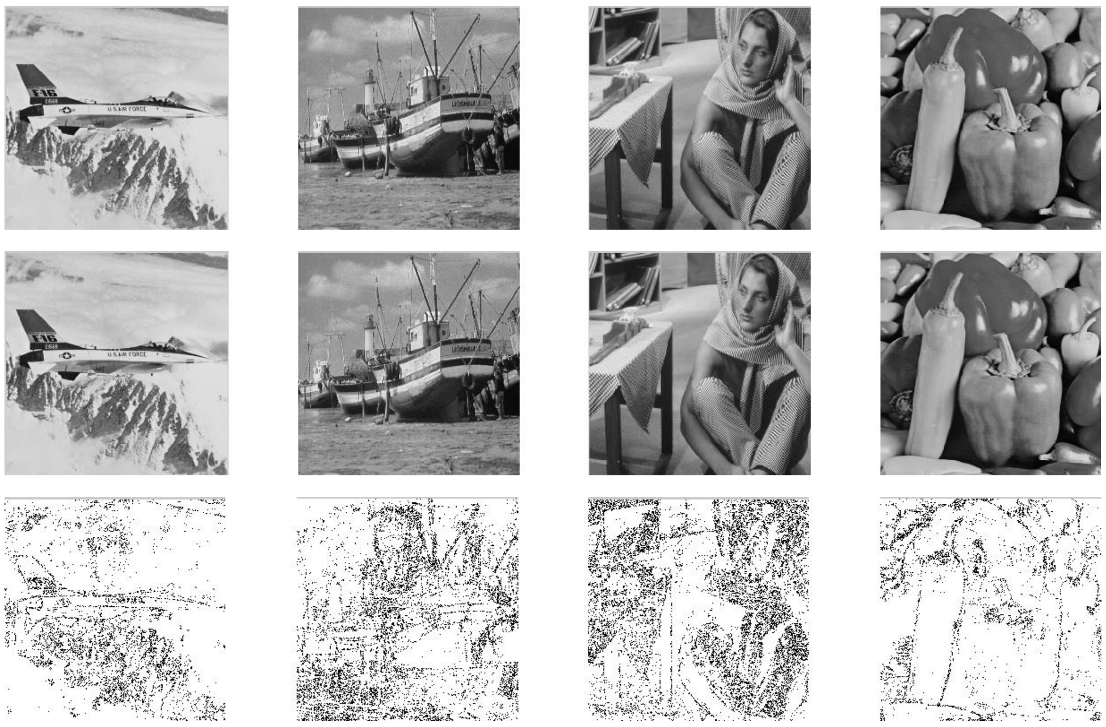


Figure 3.9. Results of Chang-Chen-Wang's Method

The following table shows the values of all three methods. According to values obtained, the 1LSB method with least destruction on an image, has high quality after embedding information inside of covers.

Table 3.1. PSNR and SSIM of three methods

	Airplane	**Ship**	**Girl**	**Pepper**
1LSB PSNR (dB)	58.59	56.36	55.46	59.43
2LSB PSNR (dB)	56.46	52.63	55.59	53.1
SSAR PSNR (dB)	42.04	42.01	42.06	42.04
1LSB SSIM	0.999	0.999	0.999	0.999
2LSB SSIM	0.996	0.996	0.996	0.996
SSAR SSIM	0.797	0.798	0.789	0.788

Also table 3.2 shows differences of our obtained value and paper's value in 1LSB method [3].

Table 3.2. PSNR and SSIM of 1LSB methods

	Airplane	**Lena**
Our PSNR (dB)	54.74	53.63
Yuan's PSNR	54.48	53.83
Our SSIM	0.998	0.998
Yuan's SSIM	0.998	0.998

Table 3.3 shows differences of our obtained value and paper's value in 2LSB method [3].

Table 3.3. PSNR and SSIM of 2LSB methods

	Airplane	**Lena**
Our PSNR (dB)	51.18	51.04
Yuan's PSNR	51.29	50.98
Our SSIM	0.996	0.996
Yuan's SSIM	0.996	0.996

And table 3.4 shows differences of our obtained value and paper's value in SSAR method [5].

Table 3.4. PSNR and SSIM of SSAR methods

	Baboon	**Airplane**	**Sailboat**	**Lena**	**Pepper**
Our PSNR (dB)	42.02	42.05	42.04	42.21	42.04
Chang's PSNR	45.10	45.11	45.10	45.12	45.12
Our SSIM	0.990	0.9659	0.9783	0.9733	0.9638

Chapter 4

ANALYSIS OF THE YUAN'S AND CHANG-CHEN-WANG'S SSAR METHODS

As mentioned earlier, our purpose is to analyze Yuan's method [3] and Chang-Chen-Wang's method [5] then optimize the Chang-Chen-Wang's method.

4.1 Yuan's versus Chang-Chen-Wang's SSAR Methods

Unlike Yuan's methods [3], in Chang-Chen-Wang's method [5], input secret image is type of grayscale and input image to cover image ratio is one-to-two while the Yuan's methods [3] are one-to-one. Chang-Chen-Wang's method [5] has an ability that can be repair an image when this image tampered by a hacker. Chang-Chen-Wang's method [5] use a hash function for detecting error and has an extra procedure that each cover to restore the image must be authenticated but Yuan's methods [3] have not these features. Yuan's methods [3] use a simple algorithm to retrieve a secret image and after embedding, the photos have little damage. As a result, the final image will be displayed with high quality and therefore it will be very hard to detect. But because of the complexity of the Chang-Chen-Wang's method [5], the image quality is lower than Yuan's methods [3] but with advantages such as authentication and the ability to recover an image we can ignore this objection.

4.2 Chang-Chen-Wang's SSAR Method Analysis

As it was stated in the Chapter 1, SSAR under its assumption may be working incorrectly. This we prove by the following counterexample Example 2. In the Example 1, we have seen that denominators used in (2.17) are non-zero because identifiers appearing in (2.16) are all distinct, and hence, Vandermonde determinant is not zero. But as far as (2.17) uses arithmetic modulo 17, it shall not be also equal to zero modulo 17. It might happen that the determinant is non-zero but is zero modulo 17; in such a case, the solution does not exist as it is shown in the Example 2.

Example 2. Let's consider the same conditions as in the Example 1, but the identifiers of the cover images are (1,2,18,4,5), instead of the used (1,2,3,4,5). Then in 2.15, 2.16

$$au_{13} = (8 + 11 \cdot 18 + 3 \cdot 324) \bmod 17 = 5 \tag{4.1}$$

And the determinant used in the denominators of (2.17) will be as follows

$$\det \begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 18 & 324 \end{vmatrix} \bmod 17 = (628 + 18 + 4 - 2 - 324 - 72) \bmod 17 = \\ (324 - 52) \bmod 17 = 272 \bmod 17 = 0 \tag{4.2}$$

As far as the denominator is equal to zero in (2.23), the solution of (2.16) does not exist if $id_3 = 18$ instead of $id_3 = 3$ used in the Example 1. The reason of the problem that in spite of the identifiers are mutually exclusive, they are not mutually exclusive modulo 17. The system of equations (2.16) with the last equation replaced according to (2.22) by

$$au_{13} = 5 = (\alpha_1' + \beta_1''' \cdot 18 + \gamma_1' \cdot 324) \bmod 17$$

thus, has no solution.

It is claimed in [5] that SSAR can recognize malicious users with the help of hash function but it is known and can be used by fake users to get a valid hash for the artificially created authenticators. Also, the hash function used is only four-bit, and needs extension for greater reliability. In the next chapter 5, we propose an enhancement of SSAR, SSAR-E method that always works correctly, counters fake participants, has greater bit-size hash function, and allows exact repairing of up to n-1 damaged pixels out of n given that at least one pixel is not corrupted. In SSAR, only 5 bits out of 8 are repaired, but in the SSAR-E, all 8 bits are repaired exactly.

Chapter 5

DEVELOPMENT OF SSAR-E METHOD

Our proposed enhancement basically is the same as SSAR but has the following modifications to

- Have it working correctly for assumptions used;
- Repair all eight bits of the partner pixel;
- Withstand fake participant attack and extend hash function value bit size from 4 to 5;
- Restore up to S-1 corrupted secret image pixels out of S;
- Allow the secret image disclosing to the authorized parties only.

The modifications are presented below.

Modification 1 (to have the SSAR correctly working). Covers' identifiers are not just mutually exclusive as required in [5], p. 3075, Section 3.1, but mutually exclusive modulo 17, i.e. $(\forall i, j)(i \neq j \rightarrow id_i - id_j \neq 0 \bmod 17$. In that case, actually, systems like (2.16) are always solvable because the determinant of the coefficient matrix is not only are not zero but also not zero modulo 17 as it is required by the algorithm.

Modification 2 (to have an opportunity of repairing all eight bits of the partner pixel, not just five bits). In SSAR, from each pixel, three entities are constructed, $\alpha_i, \beta_i, \gamma_i$, which are 3, 3, and 2 bit entities, extended to 4-bit entities $\alpha_i^{'}, \beta_i^{'}, \gamma_i^{'}$ by the use of 1, 1, and 2 bits of the partner pixel. Thus, four bits of the partner pixel are mixed with the three entities, and the 5-th bit of the partner pixel is embedded in the numbers obtained in Step 6, equation (2.8). In SSAR-E, we work with four two-bit entities, and SSAR Embedding Part, Steps 2-4 are rewritten as follows.

Step 2. For each i, define four quantities,

$$\alpha_i = \sum_{k=1}^{2} a_{ki} 2^{2-k} \in \{0,..,3\}, \beta_i = \sum_{k=1}^{2} a_{k+2,i} 2^{2-k} \in \{0,..,3\}, \gamma_i = \sum_{k=1}^{2} a_{k+4,i} 2^{2-k} \in \{0,..,3\},$$
$$\delta_i = \sum_{k=1}^{2} a_{k+6,i} 2^{2-k} \in \{0,..,3\} \quad ,$$

$i=1,..,S$ (5.1)

Step 3. Concatenate the bits $b_{1i},..,b_{8i}$, of the partner pixel, *LS'i* with the four quantities (5.1), getting

$$\begin{aligned} \alpha_i^{'} &= \alpha_i + 8b_{1i} + 4b_{2i} \in \{0,..,15\}, \beta_i^{'} = \beta_i + 8b_{3i} + 4b_{4i} \in \{0,..,15\}, \\ \gamma_i^{'} &= \gamma_i + 8b_{5i} + 4b_{6i} \in \{0,..,15\}, \delta_i^{'} = \delta_i + 8b_{7i} + 4b_{8i} \in \{0,..,15\} \end{aligned} , \ i=1,..,S. \quad (5.2)$$

Step 4. Using (2.5), calculate an authenticator,

$$au_{ik} = (\alpha_i^{'} + \beta_i^{'} \cdot id_i + \gamma_i^{'} \cdot id_i^2 + \delta_i^{'} \cdot id_i^3) \bmod 17 \in \{0,..,16\} \quad (5.3)$$

For each pixel and each cover, *i=1,..,S, k=1,..,n*. Thus, for each pixel, we get a system of *n >= 4* equations that allows solving them with respect to $\alpha_i^{'}, \beta_i^{'}, \gamma_i^{'}, \delta_i^{'}$ after embedding of the authenticators into the cover images by a sender, and the next extraction of the authenticators from the cover images by a receiver.

Also Steps 4-6 of the SSAR Extraction Part are to be rewritten as follows.

Step 4. If checking in (2.12) of Step 3 is true (no error) for any four covers, then four entities, $\alpha R_i^{'}, \beta R_i^{'}, \gamma R_i^{'}, \delta R_i^{'}$, used in calculation of authenticators (5.3) are restored from (5.3) by solving a system of at least four linear modulo 17 algebraic equations with respect to four unknowns. That system is always solvable because of the condition introduced in Modification 1.

Step 5. Having restored $\alpha R_i^{'}, \beta R_i^{'}, \gamma R_i^{'}, \delta R_i^{'}$, values αR_i , βR_i , γR_i , δR_i together with $bR_{1i},..,bR_{8i}$ from (5.2) can be restored as follows

$$\begin{aligned}&\alpha R_i = \alpha R_i^{'} \bmod 4, \beta R_i = \beta R_i^{'} \bmod 4, \gamma R_i = \gamma R_i^{'} \bmod 4, \delta R_i = \delta R_i^{'} \bmod 4,\\ &bR_{1i} = \alpha R_i^{'} div 8, bR_{2i} = (\alpha R_2^{'} div 4) \bmod 2, bR_{3i} = \beta R_i^{'} div 8, bR_{4i} = (\beta R_i^{'} div 4) \bmod 2,\\ &bR_{1i} = \gamma R_i^{'} div 8, bR_{2i} = (\gamma R_2^{'} div 4) \bmod 2, bR_{3i} = \delta R_i^{'} div 8, bR_{4i} = (\delta R_i^{'} div 4) \bmod 2\end{aligned} \qquad (5.4)$$

Step 6. From (5.4), the original pixel is restored using (5.1)

$$LS_i = 64 \cdot \alpha R_i + 16 \cdot \beta R_i + 4 \gamma R_i + \delta R_i \qquad (5.5)$$

Obtained in (5.4) eight bits $bR_{1i},..,bR_8$ of the partner pixel *LS'i* can be used for its restoration in the case if it is damaged.

Modification 3 (to have SSAR resistant to fake participant attack and hash function value bit number extension from 4 to 5). SSAR uses any hash function that does not allow countering fake participant attack. To resist the attack, the hash function shall be cryptographic, i.e. having a secret key parameter shared by the valid communicating parties. Also, as far as all the participant pixel bits are embedded into four entities (see Modification 2), one bit used in the number (2.8) for keeping 5-th bit of the partner pixel, now can be used for the hash function value keeping. Thus, Steps 5-6 of the SSAR Embedding Part are rewritten as follows.

Step 5. For each authenticator (2.6), au_{ik}, calculate its 4-bit hash function value,

$$hv_{ik} = hash_{SK}(i \,||\, au_{ik}) \in \{0,..,31\}, \; i=1,..,S, \; k=1,..,N, \qquad (5.6)$$

Where *SK* is a secret key shared by the valid communicating parties.

Step 6. Using (5.3), (5.6), generate a number

$$N_{ik} = au_{ik} \cdot 34 + hv_{ik} \in \{0,..,575\} \qquad (5.7)$$

By mixing an authenticator and its hash in a way allowing getting back its constituent parts. The number can be represented as a four-digit base-5 number.

SSAR Extraction Part Steps 2-3 are also to be rewritten accordingly:

Step 2. Restore constituent parts of the numbers (2.10) using (5.7)

$$auR_{ik} = NR_{ik} div 34; hvR_{ik} = NR_{ik} \bmod 34 \tag{5.8}$$

Step 3. Check errors (authenticity) of the restored by (5.8) authenticator by recalculating of the hash of the restored authenticator, auR_{ik}, and comparing it versus its restored hash function value (see 5.8):

$$hash_{SK}(i \| auR_{ik}) == hvR_{ik}. \tag{5.9}$$

If (5.9) is true then the authenticator is considered valid, otherwise an error is detected.

Modification 4 (to have an opportunity of repairing *S-1* damaged pixels out of *S*). SSAR Embedding Part uses (see Step 1) some permutation to produce a dual sequence *LS'*. In the Example 1, the permutation *(4,2,1,3,6,5)* was used so that the partner for LS_1 is LS_4 (LS_1=>LS_4), for LS_2 is LS_2 (LS_2=>LS_2), for LS_3 and LS_1 (LS_3=>LS_1), for LS_4 is LS_3 (LS_4=>LS_3), for LS_5 is LS_6 (LS_5=>LS_6), and for LS_6 is LS_5 (LS_6=>LS_5), where *A=>B* denotes that *B* is a partner of *A*. We see that the permutation defines three loops of partnership: LS_1=>LS_4=>LS_3=>LS_1, LS_2=>LS_2, LS_5=>LS_6=>LS_5. For example, if LS_1 is damaged, it may be restored from LS_3, that in turn ca be damaged and restored from LS_4 if LS_4 is correct. Thus, we see that if we have a loop of partnership of length three, then if its elements are damaged, they can be restored from only one correct remained pixel. But if a loop is of length one, then there is no repairing opportunity for that pixel. It is desirable having the loops of partnership of the maximal possible length to have maximal repairing facility. Maximal possible loop involves all *S* elements of the secret image pixels Hence, the permutations to be used in the SSAR-

E are to be such that they have only one loop involving all pixels of the image, e.g., it may be a permutation which is circular right shift, *(6,1,2,3,4,5)* for *S=6*, where $LS_1 \Rightarrow LS_6 \Rightarrow LS_5 \Rightarrow LS_4 \Rightarrow LS_3 \Rightarrow LS_2 \Rightarrow LS_1$. In that case, if LS_1 is damaged it may be restored from LS_2, which may be restored from LS_3, which may be restored from LS_4, which may be restored from LS_5, which may be restored from LS_6 If it is correct. Thus, five pixels, may be restored from just one remained correct pixel LS_6.Thus, to have an opportunity of repairing of *S-1* pixels out of *S* just from one remaining correct pixel, we need in Step 1 of SSAR Embedding Part using of not any permutation but a special permutation having only one loop involving all image's pixels. Thus, we are to rewrite Step 1 as follows. Note also that secret key K usage in the permutation is not necessary as far as it is not used further in SSAR.

Step 1. Producing a dual sequence LS'=permutation(LS), where permutation() involves all *S* elements in a single loop. Denote bits of LS_i as $a_{1i},..,a_{8i}$, and of its partner LS'_i as $b_{1i},..,b_{8i}$, $i=1,..,S$.

Modification 5 (to allow the secret image disclosing to the authorized parties only). The SSAR method allows disclosing the secret image to any person having any three out of n covers using equations (2.6). We can prevent it by the use of cryptographic hiding the sequence of authenticators (producing again sequence of numbers modulo 17) before the numbers (2.8), in SSAR, and (2.30), in SSAR-E, embedding in the cover images. Cryptographic hiding of the sequence of authenticators may be made, e.g., by the use Counter mode of any block cipher (see [11], p.207 therein, modified so that the block cipher output is taken modulo 17 and added/subtracted with/from plaintext/ciphertext when encrypting/decrypting. Initial counter value and the cipher key shall be shared by the authorized sender and receiver. Thus, we introduce in

SSAR-E Embedding Part the Step 4’ for hiding the identifiers, and Step 3’ in the SSAR-E Extraction Part after Step 4 and Step 3, respectively.

SSAR-E Embedding Part Step 4’. Encrypt the authenticator, au_{ik} , using an encryption algorithm, , with secret key, SK1, and current Counter value as follows:

$$au_{ik} = au_{ik} + E_{SK1}(Counter) \bmod 17; \qquad Counter = Counter + 1$$

SSAR-E Extraction Part Step 3’. Decrypt the authenticator, au_{ik} , using a decryption algorithm, $D_K(E_K(x)) = x$, such that for any valid secret key K and plaintext x, with secret key, SK1, and current Counter value as follows:

$$au_{ik} = au_{ik} - D_{SK1}(Counter) \bmod 17; \qquad Counter = Counter + 1$$

Thus, we introduced four modifications for the SSAR method to get its enhancement, SSAR-E, having such properties as

- Correct working under made assumptions;
- An opportunity of repairing all eight bits of the partner pixel instead of just five bits;
- Resistance to the fake participant attack that is based on the use of cryptographic hash functions;
- Greater probability of error detection because of the use of 5-bit hash function contrary to four bits used in SSAR;
- An opportunity of repairing up to S-1 damaged pixels out of S due to the use of the dual sequence obtained by the special permutation having all S pixels involved in one partnership loop contrary to an arbitrary permutation used in SSAR;
- Allowing the secret image disclosing to the authorized people only using cryptographic hiding of the authenticators.

Thus, the SSAR-E method having Embedding, Extraction, and repairing parts is as follows. Assumption of SSAR-E is defined in Modification 1.

SSAR-E Embedding Part (Steps 1-4, 4', 5-7).

Step 1. Produce a dual sequence LS'=permutation(LS), where permutation() involves all S elements in a single loop. Denote bits of LSi as $a_{1i},..,a_{8i}$, and of its partner LS'i as $b_{1i},..,b_{8i}$, $i=1,..,S$.

Step 2. For each i, define four quantities by (2.24)

Step 3. Concatenate the bits $b_{1i},..,b_{8i}$, of the partner pixel, LS'i, with the four quantities (2.24), getting (2.25).

Step 4. Using (2), calculate an authenticator (23) for each pixel and each cover, i=1,..,S, k=1,..,n>=4. Thus, for each pixel, we get a system of n>= 4 equations that allows solving them with respect to after embedding of the authenticators into the cover images by a sender, and the next extraction of the authenticators from the cover images by a receiver.

Step 4'. Encrypt the authenticator, au_{ik}, using an encryption algorithm, , with secret key, SK1, and current Counter value as follows:

$$au_{ik} = au_{ik} + E_{SK1}(Counter) \bmod 17; \qquad Counter = Counter + 1$$

Step 5. For each authenticator (2.6), au_{ik}, calculate its 4-bit hash function value (2.29).

Step 6. Using (2.26), (2.29), generate the number (2.30) by mixing an authenticator and its hash in a way allowing getting back its constituent parts. The number can be represented as a four-digit base-5 number.

Step 7. Each cover image Ck is represented as a sequence of S 4-pixel blocks so that in the form of four base 5 digit number, $N_{ik}[1..4]$, is embedded into four consecutive pixels Ck[4(i-1)+1,..,4i] by (2.9).

SSAR-E Extraction Part (Steps 1-3, 3', 4-6)

Step 1. Restore digits of the embedded number using (2.9) get (2.10).

Step 2. Restore constituent parts of the numbers (2.10) using (2.30) get (2.31).

Step 3. Check errors (authenticity) of the restored by (2.31) authenticator by recalculating of the hash of the restored authenticator, auR_{ik} , and comparing it versus its restored hash function value (see (2.31)) using the condition (2.32). If (2.32) is true then the authenticator is considered valid, otherwise an error is detected.

Step 3'. Decrypt the authenticator, au_{ik} , using a decryption algorithm, D , such that $D_K(E_K(x)) = x$ for any valid secret key K and plaintext x, with secret key, SK1, and current Counter value as follows:

$$au_{ik} = au_{ik} - D_{SK1}(Counter) \bmod 17; \qquad Counter = Counter + 1$$

Step 4. If checking in (2.12) of Step 3 is true (no error) for any four covers, then four entities, $\alpha R_i^{'}, \beta R_i^{'}, \gamma R_i^{'}, \delta R_i^{'}$, used in calculation of authenticators (2.26) are restored from (2.26) by solving a system of at least four linear modulo 17 algebraic equations with respect to the four unknown entities. That system is always solvable because of the condition introduced in Modification 1 the Vandermonde determinant of the system is non-zero.

Step 5. Having restored $\alpha R_i^{'}, \beta R_i^{'}, \gamma R_i^{'}, \delta R_i^{'}$, values αR_i , βR_i , γR_i , δR_i together with $bR_{1i},..,bR_{8i}$ from (2.25) can be restored by (2.27).

Step 6. From (2.27), the original pixel is restored using (2.24) by (2.28).

Chapter 6

CONCLUSION AND FUTURE WORK

This thesis was analysis of two secret sharing methods that their shares by multi cover adaptive steganography. Both of methods are share-constructing method. In the construction step we used grayscale images. For revealing the secret image we used simple Boolean operation on Yuan’s method but used three linear algebraic equations modulo 17 for Chang-Chen-Wang’s method.

Yuan’s method share the secret bits of an image among the edges and texture regions of particular covers by using the gradient-based measure but Chang-Chen-Wang’s method share these bits within meaningful contents of covers Images also fake cover images and errors in the covers are recognized and may be repaired. In Chang-Chen-Wang’s method, the SSAR algorithm is proposed claiming to solve the problem. It uses three entity representation of every secret image pixel and mixes them with four bits of the partner pixel from the dual pixel sequence obtained by shuffling of the secret image pixel sequence. For each such three entities and each cover having a unique identifier, an authenticator is calculated as a second order polynomial of the identifier and the three entities used as its coefficients. If number of covers is greater or equal to three, any three of such authenticators can be used for unique restoration of the three entities from which the secret image pixel may be restored. The authenticators are embedded into the cover images together with their hash function values (for their next checking after un-hiding) and the 5-th bit of the partner pixel. SSAR method may be classified as *(3,n)* secret sharing method meaning that for restoration of the secret method, it is sufficient having any three out of *n* cover images used for embedding of the secret image. Analysis of SSAR shows that this ability of restoration of the secret image from any three covers is based on the solvability of a system of three linear algebraic modulo 17 equations that is expected to be provided assuming that the covers' identifiers are mutually exclusive. We show by the counterexample that this condition is not sufficient and prove that it shall be extended to mutually exclusive modulo 17 identifiers which is used in our proposed enhanced method, SSAR-E. Hiding of the authenticators is done by the Least Significant Base 5 Digit method with the authenticator represented as a 4-digit base 5 number, needing four cover image's pixels per one authenticator. In the course of the authenticator un-hiding, the secret image pixel is restored together with five bits of the partner pixel if the authenticator is classified as a correct one. In the proposed SSAR-E, four two-bit entities are produced from a secret image pixel that allows embedding in the authenticator of all eight bits of the partner pixel thus improving remedy facilities of SSAR method but it is now *(4, n)* secret sharing method, i.e. any four cover images out of *n* are necessary for the secret image un-hiding. Correctness of an authenticator is checked by the use of a 4-bit hash function value in SSAR; we extend the function to a 5-bit one that allows more reliable checking of the authenticators. The SSAR method using usual hash functions is able detecting errors but is not able countering fake participant attack when some cover gets embedded fake authenticators together with their correctly calculated known hash function. In the proposed SSAR-E method, a cryptographic hash function having a secret key value shared by the valid communicating parties is used to counter fake participant attack. If it is found out that most pixels are corrupted, this may serve as an indicator of the fake participant attack. The SSAR method uses an arbitrary permutation for shuffling the pixels of the secret image to get the dual sequence having partner pixels of the secret image. If the pixel is corrupted, in SSAR-E, it can be fully 100% restored from the partner pixel contrary to the SSAR where only five out of 8 bits (62.5%) could be restored. If the partner pixel in SSAR method is also corrupted then repairing is not possible generally (because its partner may be itself). In SSAR-E, the permutation for the dual sequence generating is selected so that it involves all the secret image pixels in a single partnership loop meaning that if a pixel and its partner are corrupted but the partner of the partner is not corrupted they may be restored from the latter, and so up to *S-1* corrupted pixels in the partner loop can be restored from the last in the partnership loop correct pixel that is a very important feature of the secret sharing method not supported by the SSAR method.

However, both of methods have same goal but one of them has easy instructions for embedding and recovery, another one vice versa and one of them can reconstruct the secret after tampering by hacker but another one cannot.

I intend to study this topic as of my future work, moreover, improved performance of the (k, n) model of presented in the Hai- Dong Yuan's paper and Chang-Chen-Wang's paper by using n specific covers.

REFERENCES

[1] Shamir, A. (1979). How to share a secret, *Communications of the ACM.* , vol. 22, 612–613.

[2] Blakley, G. R. (1979). Safeguarding cryptographic keys, in *AFIPS Conf. Proc.*, vol. 48, 313–317.

[3] Yuan, H.D. (2014). Secret Sharing with multi-cover adaptive steganography, *Information Sciences.*, vol. 254, 197-212.

[4] Gonzalez, R.C., & Woods R.E. (2002), *Digital Image Processing, Prentice Hall*, NJ, Vol. 2/E.

[5] Chang, C.C. & Chen, Y.H. & Wang H.C. (2011). Meaningful secret sharing technique with authentication and remedy abilities, *Information Sciences.*, vol. 181, 3073-3084.

[6] Lin, P.Y. & Chen, Y.H. & Hsu, M.C. & Juang, F.M. (2013). Secret Sharing Mechanism with Cheater Detection, *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)* in Kaohsiung, 1-4.

[7] Salehi, S. & Balafar, M.A. (2015). An Investigation of Image Secret Sharing, *International Journal of Security and its Applications*, vol. 9, 163-190.

[8] Wu, X. & Sun, W. (2013). Secret image sharing scheme with authentication and remedy abilities based on cellular automata and discrete wavelet transform, *Journal of Systems and Software,* vol. 86, 1068-1088.

[9] DNA repair (last access date 20.07.2015), https://en.wikipedia.org/wiki/DNA_repair,.

[10] Cramer's rule (last access date 20.07.2015), https://en.wikipedia.org/wiki/Cramer%27s_rule,.

[11] Stalling, W. (2009). *Cryptography and Network Security,* vol. 5/E.

APPENDICES

Appendix A: 1LSB (n, n) SS Yuan Algorithm

Appendix A-1: Code for Importing Images

```
clc;
clear all;
addpath('pics1/')
pic_m = 8;
%==================================================================
n = 2;
Cover_size = 512;
Covers = uint8( zeros(Cover_size,Cover_size,n) );
img_name = 'C';
for i=1:n
    img = imread([img_name, '',num2str(i) '.jpg']);
    C = size(img);
    if(length(C)> 2)
        img = rgb2gray(img);
    end
    img = imresize(img, 'OutputSize',[Cover_size Cover_size]);
    Covers(:,:,i) = img;
end

A = imread('Secret.bmp');
Secret_size = 512;
A = imresize(A, 'OutputSize',[Secret_size Secret_size]);
%==================================================================
for i=1:n
    c = image_to_3Dmat(Covers(:,:,i),pic_m);
    Covers_bit(:,:,:,i) = c;
end
%==================================================================
```

Appendix A-2: Code for Calculating (2.2)

```
function B = binaryimageB(Covers)
[r,c,d,t]= size(Covers);

B = xor(Covers(:,:,1,1),Covers(:,:,1,2));
for i=3:t
    B = xor(B, Covers(:,:,1,i));
end

end
%==================================================================
```

Appendix A-3: Code for Calculating Gradient Magnitude

```
for i=1:n
    g = cacl_g(double(Covers(:,:,i)));
    gT(:,:,i) = g;
en
%==================================================================
function g = cacl_g(image)
    [g,Gdir] = imgradient(image);
end
```

Appendix A-4: Code for Embedding Phase

```
[S,Embeding_map] = embed_loop(A,B,Covers,gT);
%==================================================================
function [ST,Embeding_map] = embed_loop(A,B,Covers,gT)

[r,c]=size(A);

ST = Covers;
Embeding_map = ones(size(Covers));
for i=1:r
    for j=1:c
        if(A(i,j)~=B(i,j))
            [t,T]= max(gT(i,j,:));
            if(Covers(i,j,T)==255)
                ST(i,j,T) = Covers(i,j,T)-1;
                Embeding_map(i,j,T) = 0;
            elseif(Covers(i,j,T)==0)
                ST(i,j,T) = Covers(i,j,T)+1;
                Embeding_map(i,j,T) = 0;
            else
                if(rand(1)>0.5)
                    ST(i,j,T) = Covers(i,j,T)-1;
                    Embeding_map(i,j,T) = 0;
                else
                    ST(i,j,T) = Covers(i,j,T)+1;
                    Embeding_map(i,j,T) = 0;
                end
            end
        end
    end
end
%==================================================================
```

Appendix A-5: Code for Converting Double Value of Image into Binary

```
for i=1:n
    s = image_to_3Dmat(S(:,:,i),pic_m);
    s_bits(:,:,:,i) = s;
end
%==================================================================
function outmatrix = image_to_3Dmat(image,pic_m)
[r,c]=size(image);
outmatrix= zeros(r,c,pic_m);
h=waitbar(0,'please wait...');

for i=1:r
    for j=1:c
        x1 = dec2bin(image(i,j));
        for l=1:length(x1)
            if( x1(l)=='1')
                outmatrix(i,j,length(x1)-l+1) = 1;
            end
        end
    end
    waitbar(i/r)
end
close(h);

%==================================================================
```

Appendix A-6: Code for Calculating PSNR and SSIM and Show Results

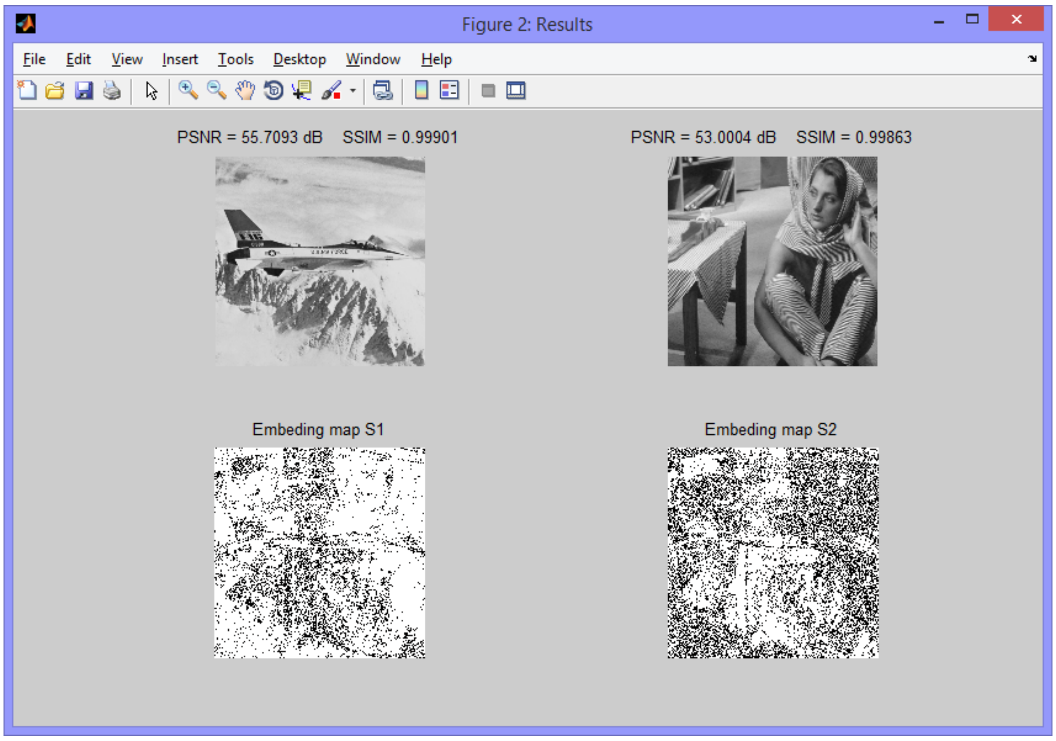
```
figure('units','normalized','outerposition',[0 0 1 1]);
set(gcf, 'name', 'Results');
for i=1:n
    subplot(3,n,i); imshow(Covers(:,:,i)); title(['Cover
',num2str(i)]);
    subplot(3,n,i+n); imshow(uint8(S(:,:,i)));
    PSNR = My_PSNR(Covers(:,:,i),S(:,:,i));
    [ssimval, ssimmap] = ssim(uint8(S(:,:,i)),Covers(:,:,i));
    title(['PSNR = ',num2str(PSNR) ,' dB    SSIM =
',num2str(ssimval)]);
    subplot(3,n,i+2*n); imshow(Embeding_map(:,:,i));
end
%=====================================================================
tic
    Ap = binaryimageB(s_bits);
toc;
time = toc;
%=====================================================================
figure;
set(gcf, 'name', 'Secret Image and Image Recovery');
subplot(1,2,1);
imshow(A);
title('input image (two tone)');
subplot(1,2,2);
imshow(Ap);
title('extraced image (two tone)');
%=====================================================================
function My_psnr=My_PSNR(I,J)

    X = double(I);
    Y = double(J);

    MSE = sum((X(:)-Y(:)).^2) / prod(size(X));
    My_psnr = 10*log10(255 * 255/MSE);

end
%=====================================================================
```

Appendix A-7: Screenshot of PSNR and SSIM 1LSB Method



PSNR = 55.7093 dB SSIM = 0.99901

PSNR = 53.0004 dB SSIM = 0.99863

Embeding map S1

Embeding map S2

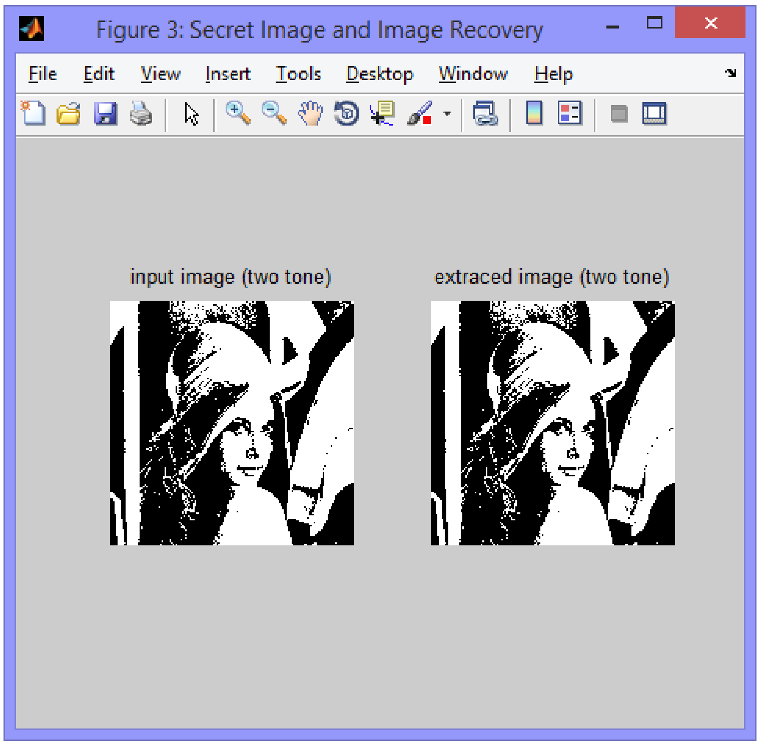


Figure 3: Secret Image and Image Recovery
File Edit View Insert Tools Desktop Window Help

input image (two tone)

extraced image (two tone)

Appendix B: 2LSB (n, n) SS Yuan Algorithm

Appendix B-1: Code for Importing Images

```
clc;
clear all;
addpath('pics/')
pic_m = 8;
%====================================================================
n = 4;
Cover_size = 256;
Covers = uint8( zeros(Cover_size,Cover_size,n) );
img_name = 'C';
for i=1:n
    img = imread([img_name, '',num2str(i) '.jpg']);
    C = size(img);
    if(length(C)> 2)
        img = rgb2gray(img);
    end
    img = imresize(img, 'OutputSize',[Cover_size Cover_size]);
    Covers(:,:,i) = img;
end
%====================================================================
A = imread('Secret.gif');
Secret_size = 256;
A = imresize(A, 'OutputSize',[Secret_size Secret_size]);
a = image2bit_to_3Dmat(A);
%====================================================================
for i=1:n
    c = image_to_3Dmat(Covers(:,:,i),pic_m);
    Covers_bit(:,:,:,i) = c;
end
%====================================================================
```

Appendix B-2: Code for Calculating (2.2)

```
B(:,:,1) = binaryimageB(Covers_bit(:,:,1,:));
B(:,:,2) = binaryimageB(Covers_bit(:,:,2,:));
%====================================================================
function B = binaryimageB(Covers)
[r,c,d,t]= size(Covers);

B = xor(Covers(:,:,1,1),Covers(:,:,1,2));
for i=3:t
    B = xor(B, Covers(:,:,1,i));
end

end
%====================================================================
```

Appendix B-3: Code for Calculating Gradient Magnitude

```
for i=1:n
    g = cacl_g(double(Covers(:,:,i)));
    gT(:,:,i) = g;
end
%====================================================================
function g = cacl_g(image)
```

```
    [g,Gdir] = imgradient(image);
end
```

Appendix B-4: Code for Embedding Phase

```
[S] = embed_loop(a,B,Covers,gT);
%====================================================================
function [ST] = embed_loop(a,B,CT,gT)
[r,c,m]=size(a);
ST = CT;
for i=1:r
    for j=1:c
        X = find_X(CT(i,j,:));

%*******************************************************************
        %case1:

%*******************************************************************
        if(a(i,j,1)~=B(i,j,1)  && a(i,j,2)==B(i,j,2) )
            ST =embed_case1(ST,gT,CT,i,j);

%*******************************************************************
            %case2:

%*******************************************************************
        elseif(a(i,j,1)==B(i,j,1)  && a(i,j,2)~=B(i,j,2) )
            ST= embed_case2(ST,gT,CT,X,i,j);

%*******************************************************************
        %case3:

%*******************************************************************
        elseif(a(i,j,1)~=B(i,j,1)  && a(i,j,2)~=B(i,j,2) )
            ST = embed_case3(ST,gT,CT,X,i,j);
        end
    end
end
%====================================================================
function ST = embed_case1(ST,gT,CT,i,j)
[t,T]= max(gT(i,j,:));
if(mod(CT(i,j,T),2)==0)
    ST(i,j,T)  = CT(i,j,T)+1;
else
    ST(i,j,T)  = CT(i,j,T)-1;
end
%====================================================================
function [ST, Embeding_map] =
embed_case2(ST,gT,CT,X,i,j,Embeding_map)
if ~(isempty(X))
    [aa, bb ]= sort( gT( i,j, X(1:length(X)) ) ,'descend' );
    T1= X(bb(1));

    if(mod(CT(i,j,T1),2)==0)
        ST(i,j,T1)  = CT(i,j,T1)-1;
        Embeding_map(i,j,T1)  = 0;
    else
        ST(i,j,T1)  = CT(i,j,T1)+1;
        Embeding_map(i,j,T1)  = 0;
    end
    [aa, bb ]= sort( gT( i,j,  : ) ,'descend' );
    T2 =bb(1);
    if(T2==T1)
        T2 = bb(2);
    end
    if(mod(CT(i,j,T2),2)==0)
        ST(i,j,T2) = CT(i,j,T2)+1;
        Embeding_map(i,j,T2) = 0;
    else
        ST(i,j,T2) = CT(i,j,T2)-1;
        Embeding_map(i,j,T2) = 0;
    end
else
    [t, T ]= max([gT(i,j,1),gT(i,j,2),gT(i,j,3),gT(i,j,4)]);
    if( CT(i,j,T)==0 )
        ST(i,j,T) = CT(i,j,T)+2;
        Embeding_map(i,j,T) = 0;
    elseif( CT(i,j,T)==255 )
        ST(i,j,T) = CT(i,j,T)-2;
        Embeding_map(i,j,T) = 0;
    end
end
%=====================================================================
function [ST,Embeding_map] =
embed_case3(ST,gT,CT,X,i,j,Embeding_map)
if ~(isempty(X))
    [t, bb ]= max( gT( i,j, X( 1:length(X) ) ) );
    T=X(bb);
    if(mod(CT(i,j,T),2)==0)
        ST(i,j,T) = CT(i,j,T)-1;
        Embeding_map(i,j,T) = 0;
    else
        ST(i,j,T) = CT(i,j,T)+1;
        Embeding_map(i,j,T) = 0;
    end

else
    [aa, bb ]= sort( gT( i , j , : ) ,'descend' );
    T1= bb(1);
    if( CT(i,j,T1)==0 )
        ST(i,j,T1) = CT(i,j,T1)+2;
        Embeding_map(i,j,T1) = 0;
    elseif( CT(i,j,T1)==255 )
        ST(i,j,T1) = CT(i,j,T1)-2;
        Embeding_map(i,j,T1) = 0;
    end
    T2 =bb(2);
    if( CT(i,j,T2)==0 )
        ST(i,j,T2) = CT(i,j,T2)+1;
        Embeding_map(i,j,T2) = 0;
    elseif( CT(i,j,T2)==255 )
        ST(i,j,T2) = CT(i,j,T2)-1;
        Embeding_map(i,j,T2) = 0;
    end
end
%=====================================================================
```

Appendix B-5: Code for Converting Double Value of Image into Binary

```
for i=1:n
    s = image_to_3Dmat(S(:,:,i),pic_m);
    s_bits(:,:,:,i) = s;
end
```

```
%======================================================================
function outmatrix = image_to_3Dmat(image,pic_m)
[r,c]=size(image);
outmatrix= zeros(r,c,pic_m);
h=waitbar(0,'please wait...');

for i=1:r
    for j=1:c
        x1 = dec2bin(image(i,j));
        for l=1:length(x1)
            if( x1(l)=='1')
                outmatrix(i,j,length(x1)-l+1) = 1;
            end
        end
    end
    waitbar(i/r)
end
close(h);

%======================================================================
```

Appendix B-6: Code for Calculating PSNR and SSIM and Show Results

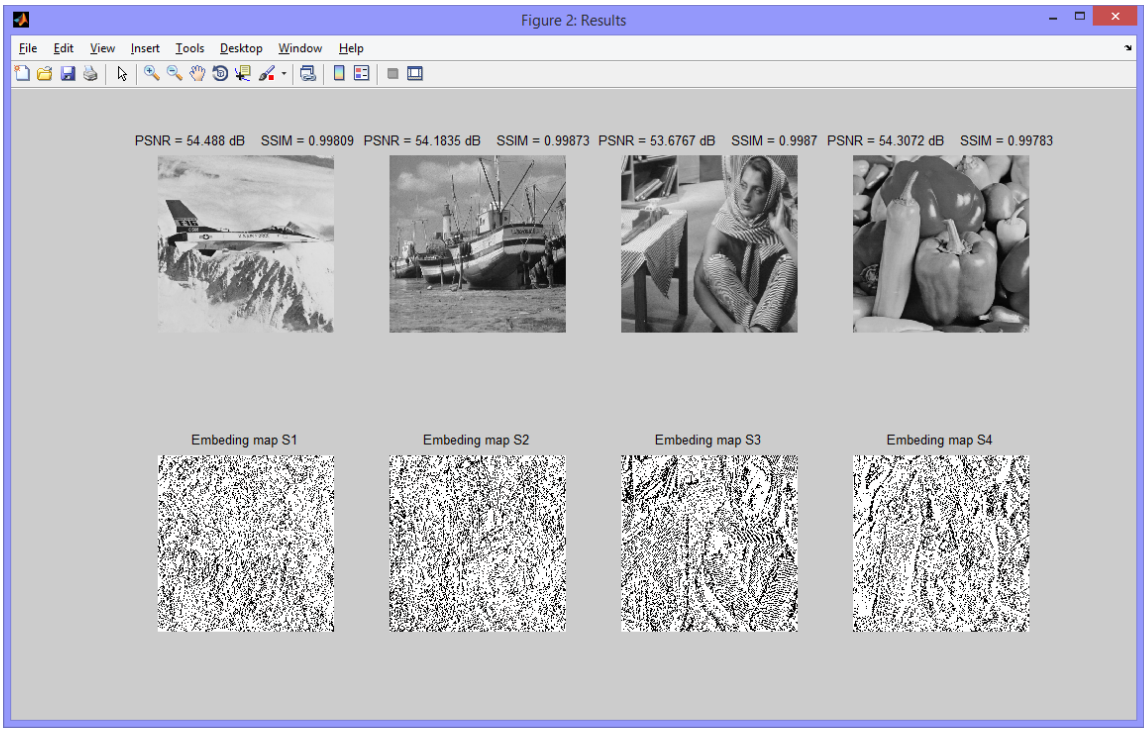
```
figure('units','normalized','outerposition',[0 0 1 1]);
set(gcf, 'name', 'Results');
for i=1:n
    subplot(3,n,i); imshow(Covers(:,:,i)); title(['Cover
',num2str(i)]);
    subplot(3,n,i+n); imshow(uint8(S(:,:,i)));
    PSNR = My_PSNR(Covers(:,:,i),S(:,:,i));
    [ssimval, ssimmap] = ssim(uint8(S(:,:,i)),Covers(:,:,i));
    title(['PSNR = ',num2str(PSNR) ,' dB    SSIM =
',num2str(ssimval)]);
    subplot(3,n,i+2*n); imshow(Embeding_map(:,:,i));
end
%======================================================================
tic
    Ap1 = binaryimageB(s_bits(:,:,1,:));
    Ap2 = binaryimageB(s_bits(:,:,2,:));
    [r,c]=size(Ap1);
    for i=1:r
        for j=1:c
            Ap(i,j,1) = Ap1(i,j);
            Ap(i,j,2) = Ap2(i,j);
        end
    end
%     Ap = convert_to_image(Ap1,Ap2,r,c);
toc;
time = toc;
%======================================================================
figure;
set(gcf, 'name', 'Secret Image and Image Recovery');
subplot(1,2,1);
imshow(A);
title('input image (Four tone)');
subplot(1,2,2);
imshow(Ap);
title('extraced image (Four tone)');
%======================================================================
function My_psnr=My_PSNR(I,J)
```

```
     X = double(I);
    Y = double(J);

    MSE = sum((X(:)-Y(:)).^2) / prod(size(X));
    My_psnr = 10*log10(255 * 255/MSE);

end
%====================================================================
```

Appendix B-7: Screenshot of PSNR and SSIM Result in 2LSB Method



PSNR = 54.488 dB SSIM = 0.99809 PSNR = 54.1835 dB SSIM = 0.99873 PSNR = 53.6767 dB SSIM = 0.9987 PSNR = 54.3072 dB SSIM = 0.99783

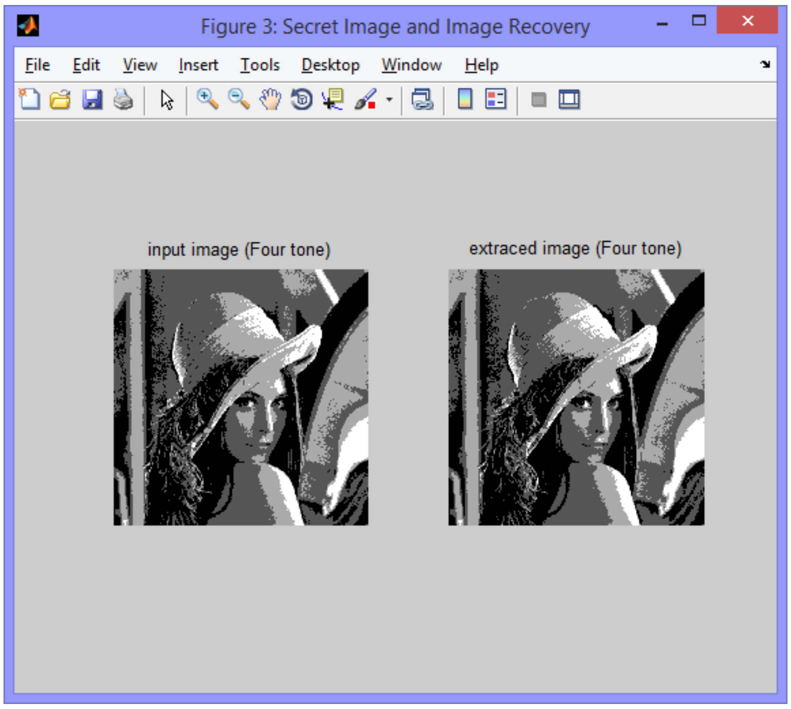


Figure 3: Secret Image and Image Recovery
File Edit View Insert Tools Desktop Window Help
input image (Four tone)
extraced image (Four tone)

Appendix C: Chang Algorithm

Appendix C-1: Code for Importing Images

```
clc;
clear all;
close all;
addpath('pics/')
%===================================================================
SI = imread('Secret.jpg'); % *************** Secret Image
*****************
Secret_size = 256;
SI = imresize(SI, 'OutputSize',[Secret_size Secret_size]);
n = 5; %************* Number of Covers ***************

%------------- Input Covers ---------------
Cover_size = 512;
Covers = uint8( zeros(Cover_size,Cover_size,n) );
img_name = 'C';
for i=1:n
    img = imread([img_name, '',num2str(i) '.jpg']);
    C = size(img);
    if(length(C)> 2)
        img = rgb2gray(img);
    end
    img = imresize(img, 'OutputSize',[Cover_size Cover_size]);
    Covers(:,:,i) = img;
end

%-------------------------------------------------------------------
```

Appendix C-2: Code for Shuffling Phase

```
id = (1:n); % ---------- Covers Identifier ------------
K = 1.76; % -------- Permutation Key ----------

[hd, wd] = size(SI);
S = hd*wd;
[X, Y] = Permutation_phase(hd*wd, K);
[LS] = SI(X);
[LS_prim] = SI(Y);
%===================================================================
function [LS, LS_Prim] = Permutation_phase(L, K)
    LS = 1:L;
    LS_Prim = Henon_map(L, K);
end
%===================================================================
function shuffled_index = Henon_map(count, key)
    x = zeros(1, count+1);
    x(1) = 0.8912;
    y = zeros(1, count+1);
    y(1) = 1;
    a = key;
    b = 0.1;
    for i = 2:count+1
        x(i) = 1 - a * x(i-1)^2 + y(i-1);
        y(i) = b * x(i-1);
    end
    [tmp, shuffled_index] = sort(x(2:count+1));
end
```

Appendix C-3: Code for Generate Authenticator and Hash Value

```
% -------------- Three Quantities --------------
alpha = bitget(LS, 8) * 4 + bitget(LS, 7) * 2 + bitget(LS, 6);
beta = bitget(LS, 5) * 4 + bitget(LS, 4) * 2 + bitget(LS, 3);
gama = bitget(LS, 2) * 2 + bitget(LS, 1);

alpha_prim = alpha + 8 * bitget(LS_prim, 8);
beta_prim = beta + 8 * bitget(LS_prim, 7);
gama_prim = gama + 8 * bitget(LS_prim, 6) + 4 * bitget(LS_prim, 5);

b5y = bitget(LS_prim, 4);
% ------------- Authenticator, Hash Function and Hidden Data -------
for i=1:S
    for k=1:n
      au(:,i,k) = mod(int16(alpha_prim(i)) + int16(beta_prim(i)) *
      id(k) + int16(gama_prim(i)) * (id(k) ^ 2), 17);

      str = strcat(dec2bin(i,3),dec2bin(au(:,i,k),5));
      temp = string2hash(str);
      hv(:,i,k) = bitget(temp, 5) * 8 + bitget(temp, 6) * 4 + ...
      bitget(temp, 7) * 2 + bitget(temp, 8);

      b5y(:,i,k) = b5y(i);
    end
end
%--------------- Embedding Phase ---------------------
N = au * 34 + int16(hv) * 2 + int16(b5y);
```

Appendix C-4: Code for Ebmedding Phase

```
%-------------------------------------------------------------------
for i=1:n
      N_five_base = dec2base(N(:,:,i), 5, 4);
      Covers_prim_tmp = embedding_algorithm(N_five_base,
      Covers(:,:,i));
      Covers_prim(:,:,i) = reshape(Covers_prim_tmp,(hd*2),(wd*2));
end

function [Cover_prim] = embedding_algorithm(N_five_base,Cover)

Nx1 = N_five_base(:, 1) - '0';
Nx2 = N_five_base(:, 2) - '0';
Nx3 = N_five_base(:, 3) - '0';
Nx4 = N_five_base(:, 4) - '0';

[ht wd] = size(Cover);
S = (ht*wd)/4;

for i=1:S
    for j=1:4
        switch j
            case 1
                  Cover_prim(4*(i-1)+j) = Cover(4*(i-1)+j) -
                  mod(Cover(4*(i-1)+j),5) + Nx1(i);
```

```
                case 2
                      Cover_prim(4*(i-1)+j) = Cover(4*(i-1)+j) -
                      mod(Cover(4*(i-1)+j),5) + Nx2(i);
                case 3
                      Cover_prim(4*(i-1)+j) = Cover(4*(i-1)+j) -
                      mod(Cover(4*(i-1)+j),5) + Nx3(i);
                case 4
                      Cover_prim(4*(i-1)+j) = Cover(4*(i-1)+j) -
                      mod(Cover(4*(i-1)+j),5) + Nx4(i);
             end
         end
end

end
```

Appendix C-5: Code for Extract Phase

```
function [ SI, Time, Covers_prim] = Extraction( Covers_prim, n, K )

Covers = double(Covers_prim(:,:,1));
[ht, wd] = size(Covers);

%------------------------------------------------ Extracting phase
S = (ht*wd)/4;
for k=1:n
   C = Covers_prim(:,:,k);
   for i=1:S
      str ='';
      for j=1:4
          NR_tmp(j) = mod(C(4*(i-1)+j),5);
          str = strcat(str,num2str(NR_tmp(j)));
      end

      NR(:,i,k) = base2dec(str, 5);
      auR(:,i,k) = floor(NR(:,i,k)/(34));
      hvR(:,i,k) = floor(mod(NR(:,i,k), 34)/2);
      b5yR(:,i,k) = floor(mod(NR(:,i,k),2));
    end
end
% ------------------- Check errors (authenticity) ------------------
for k=1:n
      hv(:,:,k) = generate_ax(auR(:,:,k), 1:size(auR,2));

      if (hv(:,:,k) == hvR(:,:,k))
            authenticated(k) = k;
      end
      ER(:,:,k) = reshape(255*(hv(:,:,k) == hvR(:,:,k)), ht/2,
wd/2);
end
% ------------------ Extraction Phase ------------------------------
if (size(authenticated,2) >= 3)

        delta = randperm(3,3);
tic
        [ alpha_prim, beta_prim, gama_prim ] = Extract_Phase( S,
auR, delta );

        [alpha, beta, gama, b1y, b2y, b3y, b4y] = ...
```

```
            reconstructing_algorithm(alpha_prim, beta_prim, 
gama_prim);

        LS = uint8(alpha * 32 + beta * 4 + gama);
        LS_prim = uint8(b1y * 2^7 + b2y * 2^6 + b3y * 2^5 + b4y * 
2^4 + double(b5yR(:,:,1)) * 2^3 + 5);


SI = uint8(reshape(LS, ht/2, wd/2));
else
    disp('Sorry, The number of legal participants fewer than 
three.');
    SI = zeros(ht/2, wd/2);
end
toc;
Time = toc;
end

% ------------------ Extraction ------------------------------

 function [ alpha_prim, beta_prim, gama_prim ] = Extract_Phase( S, 
au, delta )

    A(:,1) = [1 1 1];
    A(:,2) = delta;
    A(:,3) = delta .^ 2;

    for i=1:S
        b(:,1) = au(1,i,delta);

        A1 = A;A1(:,1)=b;
        A2 = A;A2(:,2)=b;
        A3 = A;A3(:,3)=b;
        D = int16(det(A));
        D1 = int16(det(A1));
        D2 = int16(det(A2));
        D3 = int16(det(A3));

        alpha_prim(i) = mod(int16(mod(D1 * mulinv(D,17),17)),17);
        beta_prim(i) = mod(int16(mod(D2 * mulinv(D,17),17)),17);
        gama_prim(i) = mod(int16(mod(D3 * mulinv(D,17),17)),17);

    end

end
```

Appendix C-6: Code for Recover Damaged Image

```
for i=1:n
      I = Covers_prim(:,:,n);
      position = [ht/2 wd/2];
      box_color = {'green'};
      text_str = 'I CAN FLY';
      Covers_prim(:,:,n) = ...

rgb2gray(insertText(I,position,text_str,'FontSize',14,'BoxColor',box
_color,'BoxOpacity',0.4));
```

```
end
for k=1:n
      C = Covers_prim(:,:,k);
      for i=1:S
            str ='';
            for j=1:4
                  NR_tmp(j) = mod(C(4*(i-1)+j),5);
                  str = strcat(str,num2str(NR_tmp(j)));
            end

            NR(:,i,k) = base2dec(str, 5);
            auR(:,i,k) = floor(NR(:,i,k)/(34));
            hvR(:,i,k) = floor(mod(NR(:,i,k), 34)/2);
            b5yR(:,i,k) = floor(mod(NR(:,i,k),2));
        end
end
for k=1:n
      hv(:,:,k) = generate_ax(auR(:,:,k), 1:size(auR,2));

      if (hv(:,:,k) == hvR(:,:,k))
            authenticated(k) = k;
      end

      ER(:,:,k) = reshape(255*(hv(:,:,k) == hvR(:,:,k)), ht/2, wd/2)
end
      perm_index = Henon_map((ht/2)*(wd/2), K)
      [tmp, sorted_index] = sort(perm_index)
      LS_prim_sorted = LS_prim(sorted_index)

      inauthentic_pixels = (reshape(ER(:,:,delta(1)), 1,
      ht*wd/4)==0)|(reshape(ER(:,:,delta(2)), 1,
      ht*wd/4)==0)|(reshape(ER(:,:,delta(3)), 1, ht*wd/4)==0)

      inauthentic_indeces = getIndexOf(inauthentic_pixels, 1)
      SI(inauthentic_indeces) = LS_prim_sorted(inauthentic_indeces)

      authentic_indeces = getIndexOf(inauthentic_pixels, 0)
      SI(authentic_indeces) = LS(authentic_indeces)

      SI = uint8(reshape(SI, ht/2, wd/2))
```

Appendix C-7: Code for Calculating PSNR, SSIM and Show Results

```
figure('units','normalized','outerposition',[0 0 1 1]);
set(gcf, 'name', 'Covers before and after embedding');
for i=1:n
    subplot(2,n,i); imshow(Covers(:,:,i)); title(['Cover
',num2str(i)]);
    subplot(2,n,i+n); imshow(uint8(Covers_prim(:,:,i)));
    PSNR = My_PSNR(Covers(:,:,i),Covers_prim(:,:,i));
    [ssimval, ssimmap] =
ssim(uint8(Covers_prim(:,:,i)),Covers(:,:,i));
    title(['PSNR = ',num2str(PSNR) ,' dB    SSIM =
',num2str(ssimval)]);
end
%====================================================================
figure;
set(gcf, 'name', 'Secret Image and Image Recovery');
subplot(1,2,1);
```

```
imshow(SI);
title('input image (Grayscale Image)');
subplot(1,2,2);
imshow(uint8(SI_R));
title('extraced image (Grayscale Image)');
%==================================================================
diff = isequal(SI,SI_R);
if diff == 1
    disp('Images are same')
else
    disp('Images are not same')
end;
```

Appendix C-8: Screenshot of PSNR and SSIM Result in Chang-Chen-Wang's Method

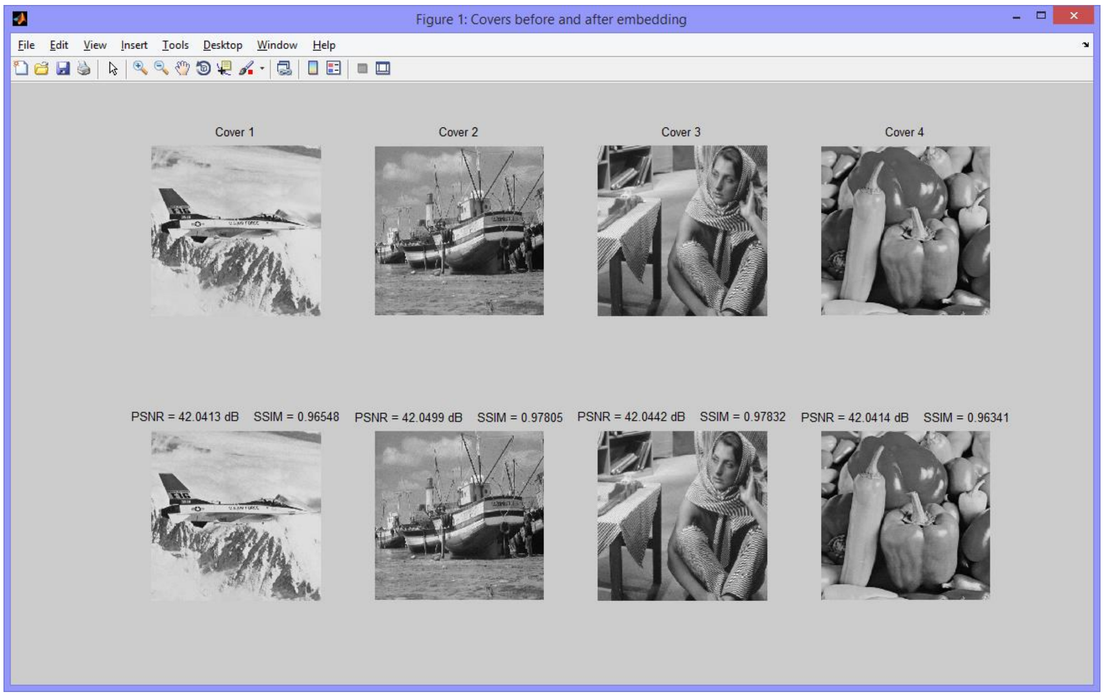


Figure 1: Covers before and after embedding
File Edit View Insert Tools Desktop Window Help
Cover 1
Cover 2
Cover 3
Cover 4

PSNR = 42.0413 dB   SSIM = 0.96548   PSNR = 42.0499 dB   SSIM = 0.97805   PSNR = 42.0442 dB   SSIM = 0.97832   PSNR = 42.0414 dB   SSIM = 0.96341

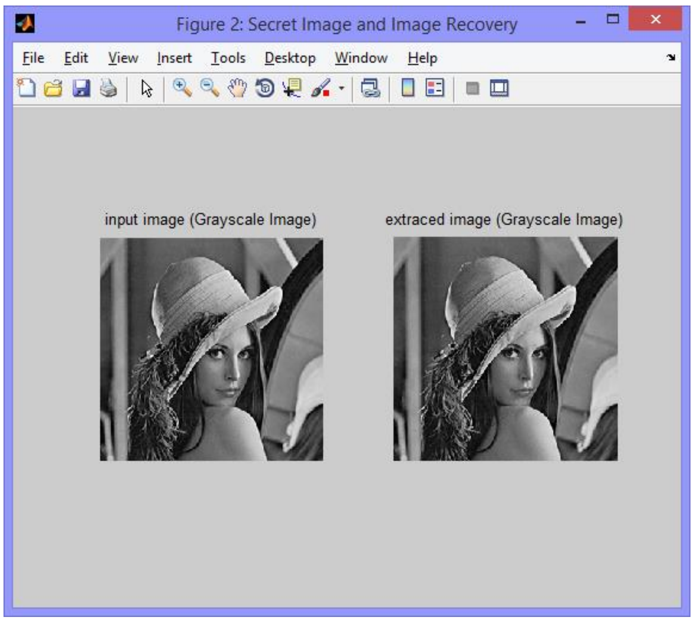


Figure 2: Secret Image and Image Recovery
File Edit View Insert Tools Desktop Window Help
input image (Grayscale Image)
extraced image (Grayscale Image)