國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

博士論文

影像之視覺式分享，權重式分享，修復與高品質資訊隱藏

Visual Cryptography, Weighted Sharing, Damage Repairing, and High Quality Hiding of Images

研 究 生：林憲正

指導教授：林志青 博士

中華民國九十九年十一月

影像之視覺式分享，權重式分享，修復與高品質資訊隱藏

Visual Cryptography, Weighted Sharing, Damage Repairing, and High Quality Hiding of Images

研 究 生：林憲正　　Student：Sian-Jheng Lin

指導教授：林志青 博士　　Advisor：Dr. Ja-Chen Lin

國立交通大學

資訊科學與工程研究所

博士論文

A Dissertation

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

November 2010

Hsinchu, Taiwan, Republic if China

中華民國九十九年十一月

影像之視覺式分享，權重式分享，修復與高品質資訊隱藏

研究生：林憲正　　指導教授：林志青 博士

國立交通大學　　資訊科學與工程研究所

摘要

本論文提出數種保護數位影像的安全性與正確性的方法。為了分散影像的機密，視覺密碼學與影像分享是本論文首先討論的兩大主題。首先，我們提出可翻轉的視覺式分享法。兩張機密影像會被編碼為兩張投影片。疊合兩張投影片可以顯示出第一張機密影像，而翻轉第一張投影片並與第二張投影片疊合，則可顯示第二張機密影像。除此之外，我們亦證明本方法在安全條件下能產生最佳對比值。

在第二主題，我們提出一種權重式機密影像分享法。影像的每個分存會被賦予一個權重值，此數值反映該分存的重要性。收到數份分存後，只要分存權重總和不小於設定的門檻值，即可解碼回機密影像。此權重式分享方法是基於多項式除法而得，且傳統的多項式機密影像分享法只是本方法的特例。另外，藉由觀察 GF(2^k)的特性，我們提出本方法的一套快速編碼演算法。

在第三主題，為了藉由影像來傳輸隱藏的機密，我們提出一種基於權重加總的資訊隱藏方法。比起其他已被提出的方法，本方法的優點為：i) 可調整的隱藏率，且範圍極廣(例如從 0.5 到 4.0 位元/像素)。ii) 在各種不同的隱藏濾下，其結果像影像的 PSNR 值有相當的競爭性。iii) 只要給予隱藏率，其結果影像的 PSNR 值可不須經由真正執行隱藏過程來獲得，而是光藉由查表即可得預測值。

在第四主題，為了保護影像的正確性，我們提出一種具有回復能力的半脆弱型浮水印。其回復資料會編碼成許多分存，並各自隱藏至影像區塊的 DCT 域中。當浮水印影像的某些部份被破壞，這些分存會從影像的未破壞區塊中抽出，然後回復資料會從這些抽出的分存中解碼，並用於修復被破壞的區域。實驗顯示在合理程度下，本浮水印法可以抵抗一些能保存內容的影像處理，像是 JPEG 壓縮，高斯模糊，與亮度調整。

Visual Cryptography, Weighted Sharing, Damage Repairing, and High Quality Hiding of Images

Student : Sian-Jheng Lin Advisor : Dr. Ja-Chen Lin

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

In the dissertation, several technologies to protect digital images are proposed. To diffuse the content of a secret image, Visual Cryptography and Sharing are both studied in the dissertation. First, a Flip Visual Cryptography system is proposed. Two secret images are encoded as two transparencies. Stacking the two transparencies can reveal one secret, and the second secret is revealed by stacking the second transparency with the flipped version of the first transparency. The proposed scheme is proved to have perfect security and conditionally-optimal contrast.

In the second topic of the dissertation, a weighted secret image sharing method is proposed. Each shadow has a weight which indicates the shadow's relative importance. The secret image can be decoded as long as the total sum of received weights reaches a specified threshold. The weighted sharing method is based on polynomial division, and the traditional polynomial-style secret image sharing is a sub-case of ours. Moreover, by observing the characteristics of GF(2^k), a fast sharing algorithm is also proposed.

In the third topic of the dissertation, in order to have confidential transmission, an image-hiding method based on weighted-sum function is proposed. Compared with previous works, the proposed method has following advantages: i) wide range of embedding rate (such as 0.5 to 4.0 bits per pixel); ii) competitive PSNR values over the whole wide range; iii) once an embedding rate is given, our look-up table can predict the PSNR value, even before the actual embedding.

In the fourth topic of the dissertation, in order to protect the accuracy of an image, a semi-fragile watermarking method with recovery ability is proposed. The recovery data are shared among many shadows; then each shadow is embedded in the DCT domain of an image block. When the watermarked image is tampered with in some area, the non-damaged

shadows are extracted from image blocks; then the recovery data are decoded from those non-damaged shadows. The recovery data are then used to repair the damaged area. Experiments show that the proposed method can resist some content-preserving operations within certain degrees, such as JPEG compression, Gaussian noise, and brightness adjustment.

謝誌

這本論文能順利付梓，首先我必須感謝我的家人。您們是我在攻讀博士學位期間最大的心靈支柱。您們是我在寫論文時最常出現在腦海的影像，我對於唸書期間鮮少回家感到抱歉。接下來我要感謝我的指導教授—林志青博士。林教授除了給予我研究上的意見外，亦付出極大的心力在指導我論文的撰寫。我亦要感謝實驗室的學長們：陳尚寬博士，方文聘博士，張御傑博士，趙崑源博士，陳李書滕博士，你們給予我不少論文上的寶貴意見與幫助。當然也不能忘記還有學弟妹們的幫忙：李相賢，葉姿敏，李豐政。最後我要感謝我的幾位好朋友，你們幫我消磨了不少念博班的無聊時光。

Table of Contents

List of Tables


E S A
1896

List of Figures

Chapter 1
Introduction

In this chapter, the motivation of the dissertation and background knowledge are presented in Sec. 1.1 and 1.2, respectively. The overview of the dissertation is given in Section 1.3. Finally, the organization of the remaining chapters is described in Section 1.4.

1.1 Motivation

With the explosive growth of internet services and improvements in hardware in recent years, network sharing of digital multimedia, such as images, audio, and video, has become extremely easy and increasingly commonplace. However, this expansion raises important issues on how to protect the accuracy of those digital media on a network. To discuss this problem, some of the chief aspects of digital image protection are considered in this dissertation.

The first research topic of this dissertation is Visual Cryptography with double secrets. For Visual Cryptography, the most important issue is security. Most single-secret VC schemes (e.g. [1-3]) prevent any single transparency from leaking out any of the pixel values of the enclosed secret image. Restated, most VC schemes satisfy their single-secret security requirements. However, security issues concerning the relation of pixels between multiple input secret images have seldom been discussed. In Chapter 2, a multiple-secrets VC scheme with perfect security is defined as each single transparency leaking out neither pixel value nor the relation of the pixel values between multiple secret images. There are two possible branches in this design: 1) the stacking result representing black pixels in the secret image is restricted to being 100% opaque; 2) the stacking result representing black pixels in the secret image is not restricted to being 100% opaque. In the proposed scheme, the first branch is called opaque-oriented FVC, and the second is called non-opaque-oriented FVC. We will demonstrate that the contrast in our design here is conditionally optimal, no matter whether 1) or 2) is used. Throughout this chapter, the word "*conditionally optimal*" means that the contrast is optimal if the double-secrets non-expanded FVC scheme is required to have perfect security.

The second research topic of this dissertation is (t, n) weighted sharing. Polynomial-based secret sharing [4, 5] is a technique of protecting the secret message (e.g. encryption keys or important files). The secret message is dispersed among n shadows, where the size of each shadow is the same as the secret message. Any t of the n shadows can recover the secret message, but any t–1 or fewer shadows cannot obtain any information pertaining to the secret message. Based on the technology used, the secret message can be preserved in many places in order to disperse risk. On the other hand, if we do not persist on perfect security (i.e. any t–1 or less shadows cannot obtain any information about the secret message), there are other approaches, such as the Information Dispersal Algorithm (IDA)[6] and secret image sharing [7]. The size of each generated shadow is 1/t of the secret message. For practical exercises, we may assign a weight to each shadow, where the weight refers to the "information ratio" of the secret message. If the sum of obtained shadow weights is not less than a specific threshold t, the secret message can be decoded. When it comes to the issue of secret image sharing among weighted participants, this problem can be solved by simply applying Thien and Lin's method [7]. However, to further improve the execution time in the weighted secret image sharing phase, a fast weighted secret image sharing method is proposed in this chapter. In Chapter 3, we propose a weighted scheme for secret image sharing, and propose an encoding algorithm that allows linear running time.

The third research topic of this dissertation is weighted-sum function hiding for gray-scale images. With a data hiding method, a secret message can be embedded in a cover-media to generate a stego-media containing full information for the secret message. This stego-media is very similar to the cover-media, and is very hard for the human eye to distinguish. Using this technology, a secret channel is built privately between a sender and a receiver, and it is very difficult for others to detect any transmission between the two. For gray-scale images, modulus-based data hiding [8] and LSB matching [9, 10] are two good hiding methods which can achieve high PSNR values (a metrical formula of calculating the similarity of two images) under some embedding rates (bpp, bits per pixel). Here we try to generalize both the modulus-based embedding method [8] and the LSB matching method [9]. Our generalization will create two benefits: 1) giving better PSNR than [8], as shown in Tables 4.3 and 4.4 later, and 2) giving a wider range of embedding rates than [9] did, particularly non-integer embedding rates. Therefore, the new product will be an all-in-one method with competitive quality everywhere over a wide range of embedding rates (including, but not limited to, rates which are non-integer or smaller than one). In Chapter 4, we establish a generalized version of

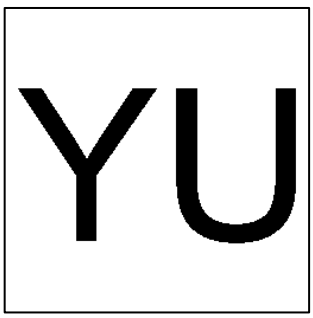
the two hiding methods introduced above for gray-scale images. Our experiments show that our proposed method has a wide embedding rate (0.5-4.0 bpp) and competitive PSNR values.

The final research topic of this dissertation is semi-fragile watermarking with recovery ability for single images. Semi-fragile watermarking is a watermarking approach for image authentication. The authentication data are embedded in the watermarked image by data hiding technology. When we examine the integrity of the image, the authentication data are extracted for checking. Some semi-fragile watermarking methods [11, 12] and recovery ability methods [13-17] have been proposed in recent years. In Chapter 5, we proposed a semi-fragile watermarking with recovery ability. By embedding the recovery data in DCT domain, the method can resist some operations, such as JPEG compression, Gaussian noise, or brightness adjustment, within a pre-defined degree.

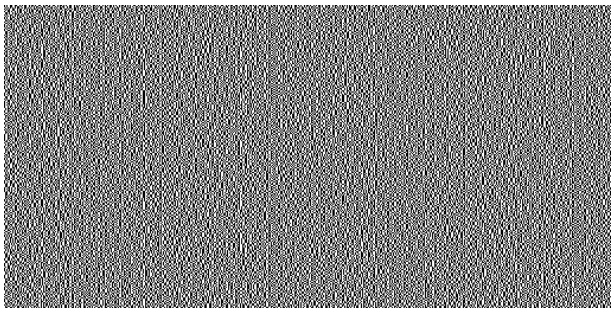
1.2 Related studies

1.2.1 Visual Cryptography

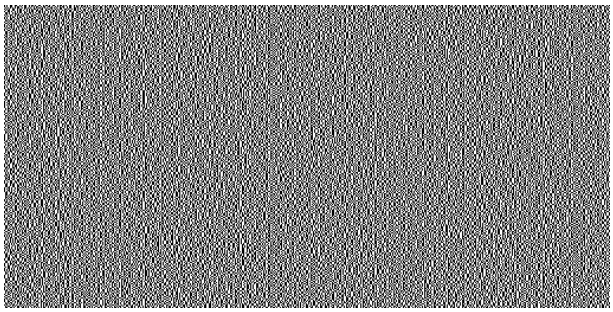
Introduced by Naor and Shamir[1], visual cryptography (VC) is an approach to decrypt secret images using the human visual system. Using VC, a secret image can be revealed by stacking the transparencies generated in the encryption process. Since the decoding process of VC depends on the inspection of stacked images using the naked eye, it has the potential to be utilized in critical environments without computer resources. We may use the simple example in Fig. 1.1 to describe VC. Fig. 1.1(a) shows the binary secret image. After using the encoding process proposed by Naor and Shamir [1], the generated transparencies are extremely noisy, as in (b) and (c). Fig. 1.1(d) shows the result of stacking the two transparencies (b) and (c) together.



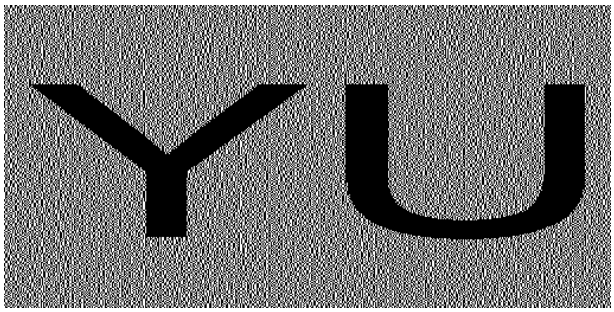
(a)



(b)



(c)



(d)

Fig. 1.1. An example of VC. (a): a secret image; (b-c): the two transparencies generated for (a) using the VC scheme of Naor and Shamir [1]; (d) the result of stacking (b) and (c).

Many VC related studies have been proposed. For example, [18-21] introduced multi-secret VC; [20, 22-25] proposed non-expanded VC so that the created transparency could be compact; and some other VC schemes [26-29] enabled VC to have more applications. The aforementioned Wu and Chang [3] proposed a method to generate two circle transparencies for sharing two secret images. When rotating one transparency by a pre-specified angle and then stacking it with another transparency, the second secret image could be revealed. With their method, the size of each transparency was four-fold larger than that of each secret image. Fang and Lin [18] used two rectangular transparencies to share two secret images. In their method, besides revealing one secret image by stacking the two transparencies, shifting one of the transparencies and then stacking them again could also reveal another secret image. The size of each transparency was also four-fold that of each secret image. Shyu et al. [19] extended the multi-secret VC scheme of Wu and Chang [3] from single rotation to several rotations so that they could encode # images in two transparencies. Nevertheless, the transparencies were still 2# times the size of each secret image.

To optimize usage of the transparencies, reducing the size of the transparencies is also an topic for study. There are several non-expanded VC schemes. For example, Yang [20] introduced a probability-based method and Shyu [22] presented a random-grid-based method. In both methods, the size of each transparency is the same as that of the secret image. Therefore, their methods are particularly suitable for situations with storage restriction. However, in their methods, only one secret image is hidden when several transparencies are created.

1.2.2 Polynomial secret sharing, secret image sharing, information dispersal algorithm, and Reed-Solomon code

In 1979, two independent researchers, Blakley [5] and Shamir [4] proposed a polynomial secret sharing scheme. In their (*t*, *n*) threshold scheme, a dealer distributes a secret number into *n* shadows, with each of *n* participants holding one shadow. The generated shadows have two properties: (1) any information about the secret message (except for the message size) cannot be extracted from any *t*–1 or less shadow. (2) the size of each shadow is the same as the size of the secret message. If a secret sharing scheme has property 1, the sharing scheme is called *perfect security*. Moreover, if the sharing scheme has property 1 and 2, the sharing scheme is called *ideal*. Shamir's polynomial secret sharing[4] is an ideal secret sharing scheme. Later, Shamir [4] introduced the concept of weighted secret sharing in his seminal work. In Shamir's weighted secret sharing with the (*t*, *n*) threshold scheme, each of the *n* participants is assigned with a positive integer weight w_i where *i*=1, 2,..., *n* and $1 \le w_i \le t-1$. Then the dealer would distribute a secret number into $\sum_{i=1}^{n} w_i$ shadows, and the number of shadows that each participant held would be equal to their corresponding weight value. The secret could be reconstructed if the sum of the weights of the received participants is no less than the threshold *t*.

When the secret data is a secret image rather than a secret number, using Blakley's or Shamir's (*t*, *n*) threshold scheme [4, 5] to share the secret image will waste much memory space because the size of the secret image is usually very large. To reduce memory space, Thien and Lin [7] proposed a secret image sharing method derived from Shamir's scheme, and Tso [30] proposed a secret image sharing method based on Blakley's scheme. In both methods, the size of each shadow is smaller than that of the secret image. In addition, based on Thien and Lin's secret image sharing method, the progressive secret image sharing schemes [26, 27, 31] were proposed in succession.

An Information Dispersal Algorithm (IDA) [6] was proposed by Rabin. Under this scheme, a file can be divided into *n* shadows, and any *t* of the *n* shadows can reconstruct the file. IDA does not care about the security of shadows, and the major advantage of IDA is that the size of each shadow is 1/*t*, which is smaller than Shamir's method 1. P. Béguin and A. Cresti [32] prove that the size of each shadow 1/*t* is minimal, if the entropy of the file is maximal. Preparate [33] proposed a fast sharing method based on Fast Fourier Transform (FFT) over an finite field.

Reed-Solomon code (RS code) [34] utilizes error control coding proposed by Reed and Solomon. The t information digits are transformed into n digits to form a code, and if any $\lfloor (n-t)/2 \rfloor$ of the n digits are modified, the t information digits can also be precisely extracted. Preparate [33] later extended this method, proposing the concept of sharing RS code.

1.2.3 Data hiding methods

Data hiding is a technology which can embed data in images. The LSB (Least Significant Bits) substitution method is probably the simplest embedding method. For example, if two secret bits $(11)_2$=3 are to be embedded in an 8-bit pixel value $(10010100)_2$=148, the two least significant bits of 148 are replaced, and the stego-pixel value is $(10010111)_2$=151, which can extract $(11)_2$ easily. To improve the LSB substitution method, Thien and Lin [8] use $(10010011)_2$=147 as the stego-pixel, since 147 is closer to 148 than 151 is, and the last two bits of 147 can still extract $(11)_2$ easily. Other papers have been published based on this or similar observations. For example, Lin et al. [35] introduced an embedding algorithm which extended the modified LSB substitution method by using a distortion tolerance. Yang [36] embedded data based on an inverted pattern approach to improve the stego-image's quality in the LSB method, and Wang [37] used a threshold to decide the modulus base of the embedding function.

Rather than using a pixel as the embedding unit, the LSB-matching method [9, 10] considers a block of several pixels simultaneously. Mielikainen [10] proposed an embedding method which embeds 2 bits in a block of 2 pixels. Li et al. [9] defined a generalized LSB matching (G-LSB-M) scheme to further reduce distortion. Zhang and Wang [38] embedded a digit which has $(2z+1)$ possible values in each z-pixel block. When the secret data contains images, another type of research focuses on [39-41] the redundancy of the secret images to improve stego-image quality. For various media formats, Tseng et al. [42] embedded data in binary images, and Wu et al. [43] in palette-based images, whereas Liu and Liao [44] used JPEG images. Lee et al.'s method [45] was for binary images, and the embedding was based on Hamming codes to reduce the frequency of flipping pixels. Wang and Lu [46] used a Vector Quanization (VQ) index file as the host media. Tseng and Hsieh [47] even proposed a reversible method, so that the host images could be recovered from the stego-images without loss, but the price of being reversible was a smaller embedding rate (for example, it embedded only 0.22 bits per pixel to get a Lena stego-image of 47.31 dB PSNR). Some other kinds of

data embedding consider the content of the host image, i.e. embedding more bits in the coarse area of the host image. For example, Wu and Tsai [48] introduced a method based on pixel value differencing, whereby each block of the host image embeds a dynamic number of bits by altering the pixel value difference. Likewise, Wang et al. [49] used a pixel value differencing and modulus function to effectively reduce distortion. Zhang and Wang [50] dynamically changed the base of secret data to control the embedding rate of the stego pixels. Yang et al. [51] also proposed adaptive data embedding in edge areas, and then the use of a modified LSB substitution method to reduce distortion. Yang et al. [52] estimated the amount of the embedded data by exploiting the brightness, edges, and texture of the host image.

1.2.4 Fragile watermarking and Semi-fragile watermarking

An image authentication method generates some data which will be used to check the accuracy of the digital media in the future; the authentication data can be stored in another file (this is the so-called digital signature approach [53, 54]), or embedded in the digital media itself (i.e. watermarking approach [55-66]). In recent years, some watermarking studies have focused on image tampered-region detection and recovery [58-66].

Lin et al. [59] proposed a watermarking technique for tamper detection and recovery, based on a three-level hierarchical structure and block-mapping sequence. In the three-level hierarchical structure, a block is judged as “applicable” if the block passes three inspections. If a block is judged as “non-applicable”, then the recovery data is embedded in LSBs of another block whose address is determined by a block-mapping sequence. Lee and Lin [63] proposed a watermarking technique which embeds dual watermarks in an image. The detection algorithm is similar to Lin et al.’s method [59], but the block size is 2×2, rather than the 4×4 used in Lin’s method [59]. If a block is judged as “non-applicable”, then the copies of recovery data are embedded in LSBs of another two blocks, which are addressed by a block-mapping sequence. The two copies of recovery data (dual watermarks) are used to increase the chances for block recovery. In Wang and Tsai’s method [64], the image is divided into two regions; for the Region-of-Interest (ROI), the recovery data are encoded by fractal encoding, and embedded in other blocks which are selected by permutation. For remaining regions, no recovery data are embedded. If a damaged block is located in an ROI, then the fractal code is extracted for recovery; otherwise, the block is recovered by an image-inpainting technique. Chan and Chang [66] proposed an image authentication method based on Hamming code consisting of three components; the Hamming code, Torus

automorphism and bit rotation. The parity check bits for each pixel were generated by the Hamming code. The embedding locations for the parity check bits were decided by Torus automorphism, and the bit rotation was used to improve security. Zhang and Wang [67] proposed an elegant watermarking method, which can restore a tampered region without error. This method is based on reversible data hiding, which can extract the whole host image from the stego-image without error.

In general, if a watermarked image is processed by some content-preserving operations (i.e., JPEG compression), verification ability should still exist within a certain level of the operation. This type of watermarking method is said to be *semi-fragile*, and in the semi-fragile watermarking method proposed by Ho and Li [11], users choose the lowest JPEG quality factor they can tolerate, and the verification data is generated and embedded in the quantized DCT domain. Their experiments demonstrated that their method could resist JPEG compression (up to a level of Quality factor *QF*). In the Lin et al. [12] method, users also choose the lowest JPEG quality factor they can tolerate, and the verification data is generated from the low/middle frequency of the DCT domain, followed by embedding in the high frequency domain. Their experiments showed that their method can also resist JPEG compression (up to a *QF* level). However, these two methods [11, 12] only embed verification data, and there is no recovery data.

There are some semi-fragile watermarking techniques which also embed recovery data in the watermarked image, and the image itself can recover any tampered regions. Lin and Chang [13] proposed two approaches to semi-fragile watermarking, one of which has verification ability only, while the other has both verification and recovery ability. The verification data is generated from the DCT coefficients, and the recovery data is generated from a quarter-size shrunken sub-image of the host image (if recovery ability is required). Then all of the generated data is embedded in the DCT domain. Their method can resist both JPEG compression and brightness adjustment within a reasonable range. Hsieh et al. [14] also proposed a watermarking scheme with damage-recovery ability. The recovery data is calculated from the host image, and then three copies of the recovery data are embedded in the DCT domain of the host image. Their experiments showed that their method could resist JPEG compression, brightness adjustment and contrast adjustment. Jiang and Liu [15] proposed an authentication-recovery scheme. Their verification data is a random number sequence generated by a key, and their recovery data is generated from DCT coefficients of the host image. The two sets of data are embedded in the two-LSBs (the two least significant

bits) of the image. Their experiments showed that their method could resist JPEG compression and small-area replacement of the watermarked image.

1.3 Overview of this dissertation

In this dissertation, several techniques to protect digital images are proposed for various applications. The proposed methods contain a flip VC method, a fast weighted image sharing method, a data hiding method, and a semi-fragile watermarking method. Fig. 1.2 shows the framework of the dissertation, and the brief overview of each proposed method is given in the subsections below.

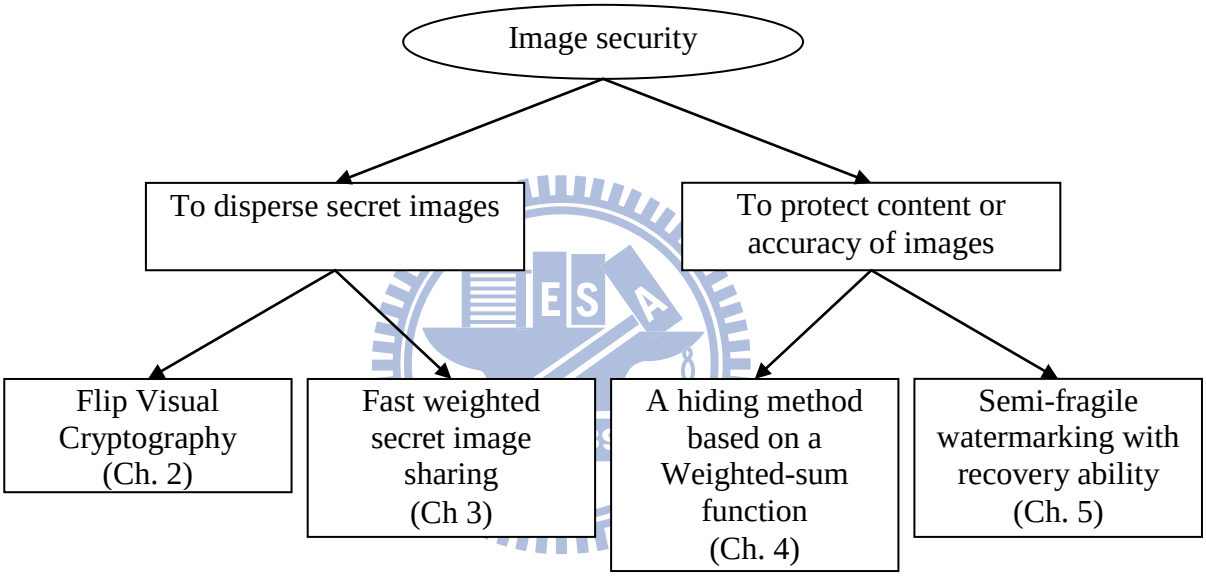


Fig. 1.2. The framework of this dissertation.

1.3.1 Flip Visual Cryptography (FVC) with perfect security, conditionally optimal contrast, and no expansion

In Chapter 2, a flip visual cryptography (FVC) scheme is proposed. The proposed FVC scheme encodes two secret images into two dual-purpose transparencies. Sixteen basis matrices are designed to encode a pair of pixels of the two secret images, respectively. If the stacking result representing black pixels in the secret image is restricted (or not restricted, respectively) to be 100% opaque, we have two designs called opaque-oriented FVC and non-opaque-oriented FVC in the proposed scheme. We also prove that the contrast in our design here is conditionally optimal, no matter whether opaque-oriented FVC or non-opaque-oriented FVC is used.

1.3.2 Fast weighted secret image sharing

Chapter 3 contains two topics. First, we bring up a weighted secret image sharing method. The method is based on polynomial division over a finite field. The size of each shadow depends on the weight choosen by the user. When an image has sufficient shadows where the sum of weights is larger than a pre-defined threshold, the secret image can be decoded by utilizing the extended Lagrange polynomial. When all weights are defined as 1, the proposed method is the same with Thien and Lin's method [7]. Then, by observing characteristics of GF(2^k), a fast encoding algorithm under GF(2^k) is proposed. The encoding algorithm is a recursive function, and the running time depends only on the size of the secret image.

1.3.3 Weighted-sum function (WSF) – a gray-scale image hiding method with competitive PSNR over a wide range of embedding rates

In Chapter 4, a hiding method is proposed based on a weighted-sum function. With this method, *m* secret bits are embedded in z pixels, and the secret bits can be extracted by executing a weighted-sum function. To minimize distortion, two optimization patterns are proposed. First, to reduce the running time of obtaining the best values of stego-pixels, a table T is dynamically generated and the stego-pixels are calculated by looking up table *T*; second, to decide the weight values in weighted-sum functions with various embedding rates, some suggested weights based on exhaustive research are given in Table 3.2. The advantages of the proposed method include: (1) A wide range of embedding rates (such as 0.5 to 4 bits per pixel), (2) Competitive image quality over the whole wide range, (3) Once the embedding rate is given, our look-up table can predict the PSNR value, even before the actual embedding.

1.3.4 Authentication and recovery of an Image by using sharing and lattice-embedding

In Chapter 5, we propose a semi-fragile watermarking method based on secret sharing and lattice-embedding. Using this method, a host image is transformed into an 8×8 DCT domain, and the coefficients in each DCT block are shared among many shadows by two-layer sharing[68]. Each shadow is then embedded in a DCT block by lattice-embedding. Because the shadow is embedded in the DCT domain, shadow data that pass certain degree of JPEG compression remain intact. However, the repairing area is smaller than the fragile version, due to the smaller embedding capacity. As shown in experiments, the watermarked

image can resist some content-preserving operations such as JPEG compression, Gaussian noise, or brightness adjustment.

1.4 Organization

The organization of the rest chapters of the dissertation is listed below. Flip Visual Cryptography is addressed in Chapter 2. Fast weighted secret image sharing is addressed in Chapter 3. A hiding method based on a weighted-sum function is addressed in Chapter 4. An image authentication method using semi-fragile watermarking with recovery ability is addressed in Chapter 5. Finally, the conclusion and the future works of this dissertation are given in Chapter 6.

Chapter 2

Flip Visual Cryptography (FVC) with perfect security, conditionally optimal contrast, and no expansion

This chapter proposes a flip visual cryptography (FVC) scheme with perfect security, conditionally optimal contrast, and no expansion of size. The proposed FVC scheme encodes two secret images into two dual-purpose transparencies. Stacking the two transparencies can reveal one secret image. Flipping one of the two transparencies and then stacking with the other transparency can reveal the second secret image. The proposed scheme is proved to have conditionally optimal contrast: its contrast is optimal if the double-secrets non-expanded FVC scheme is required to have perfect security. The perfect security is also proved.

The remainder of this chapter is organized as follows: The proposed opaque-oriented FVC scheme and non-opaque-oriented FVC scheme are stated in Sec. 2.1, respectively. Experimental results are shown in Sec. 2.2. Some discussions are shown in Sec. 2.3, and the conclusions are in Sec. 2.3. In Sec. 2.5, we prove that the contrast 1/6 (and 1/4, respectively) is conditionally optimal among the opaque-oriented FVC schemes (and non-opaque-oriented FVC schemes, respectively) that use basis-matrices design with perfect security.

Notations in this chapter:

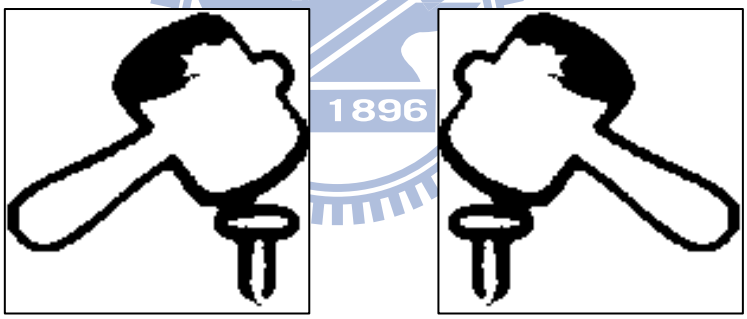
ht	The height of the secret image.
wh	The width of the secret image.
S_1, S_2	Two binary secret images in which the size is $ht \times wh$.
B, W	Black and white.
T_1, T_2	Two generated transparencies.
S_1', S_2'	Two stacking results which are similar to S_1 and S_2, respectively.
r	The width of basis matrices.
b	the minimal *luminance transmission* to represent B in stacking results.
w	the minimal *luminance transmission* to represent W in stacking results.
α	The contrast which is $w - b$.
$\otimes$	Stacking operation.

2.1 Opaque-oriented FVC and Non-Opaque-oriented FVC

In this section, we design two FVC methods which are opaque-oriented FVC and non-opaque-orient FVC. This section includes three subsections: (1). Definition of the problem; (2). The 16 basis matrices of opaque-oriented FVC; (3). The 16 basis matrices of non-opaque-oriented FVC.

2.1.1 Problem definition

Two $ht \times wh$ binary secret images, denoted by S_1 and S_2, are encoded to get two $ht \times wh$ transparencies T_1 and T_2, respectively. Without the loss of generality, the goal of the proposed FVC scheme is that the secret image S_1 can be decoded by stacking T_1 and T_2 together; whereas the secret image S_2 can be decoded by flipping T_1 over and then stacking with T_2. Fig. 2.1 illustrates the operation to flip a transparency over. Notably, the transparency in Fig. 2.1(a) is not a transparency created by our method, because our transparency is completely noise-like. Fig. 2.1 is just to explain the flip-over operation; and the explanation would have been impossible to understand if Fig. 2.1(a), and hence Fig. 2.1(b), had been completely noise-like.



(a) (b)

Fig. 2.1. (a): A transparency; (b): The transparency after flipping.

Let $S_1 = \{s_1(i, j) \mid i \in Z_{ht}, j \in Z_{wh}\}$ and $S_2 = \{s_2(i, j) \mid i \in Z_{ht}, j \in Z_{wh}\}$ be the two given black-and-white secret images. Each pixel $s_1(i, j)$ and each pixel $s_2(i, j)$ are binary in value W (white) pixel or B (black) pixel. Let $T_1 = \{t_1(i, j) \mid i \in Z_{ht}, j \in Z_{wh}\}$ and $T_2 = \{t_2(i, j) \mid i \in Z_{ht}, j \in Z_{wh}\}$ be the two transparencies to be generated. In the design of transparencies T_1 and T_2, represent every "*opaque*" pixel of a transparency by 1, and represent every "*transparent*" pixel of a transparency by 0. (To distinguish between secret image and transparency image, the words "*opaque* and *transparent*", rather than "Black and White", are used when the image being talked about is a transparency, rather than an input secret image.) In Definition 2.1, the

stacking operation is symbolized by the symbol "⊗" which is in fact the OR operator. This coincides with the real world experience: in real world, if we stack two transparencies, the places where we can see through are the places where both transparencies are transparent (both are 0s).

Definition 2.1 (Stacking operation ⊗)

The stacking operation for transparencies is symbolized by "⊗", where 0⊗0=0, 0⊗1=1, 1⊗0=1, and 1⊗1=1.

■

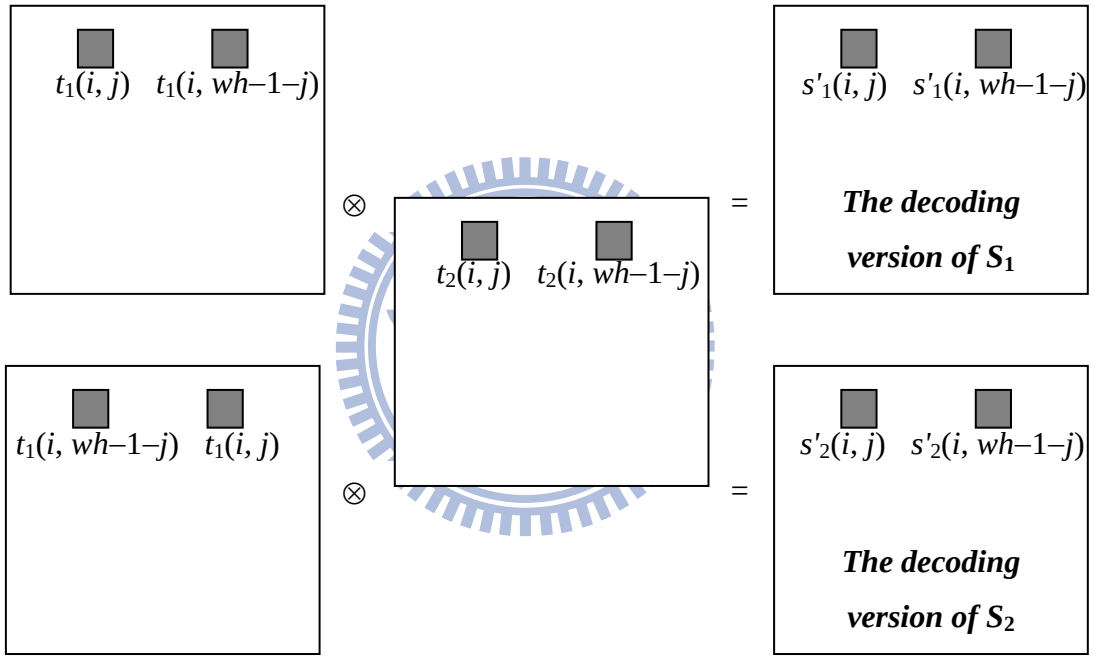


Fig. 2.2. Stacking transparencies T_1 and T_2 to decode secrets S_1 and S_2 of size $ht \times wh$ each. (Stacking T_1 and T_2 to decode secret S_1; Flipping T_1 over and then stacking with T_2 to decode secret S_2)

Fig. 2.2 illustrates the effect of stacking two transparencies T_1 and T_2 and describes what will happen when people flip T_1 over and then stack it with T_2. The two pixel values $[s_1(i, j), s_1(i, wh-1-j)]$ are called a symmetric pair, and so are $[s_2(i, j), s_2(i, wh-1-j)]$. To design a flip visual cryptography (FVC) scheme, possible values of the quadruple $[s_1(i, j), s_1(i, width-1-j), s_2(i, j), s_2(i, wh-1-j)]$ should be considered simultaneously. For each quadruple $[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]$ of secret pixels, the quadruple $[t_1(i, j), t_1(i, wh-1-j), t_2(i, j), t_2(i,$

$wh-1-j$)] of transparency pixels must meet the following four requirements simultaneously:

1) $s_1(i, j)$ is decoded by stacking $t_1(i, j)$ and $t_2(i, j)$;

2) $s_1(i, wh-1-j)$ is decoded by stacking $t_1(i, wh-1-j)$ and $t_2(i, wh-1-j)$;

3) $s_2(i, j)$ is decoded by stacking $t_1(i, wh-1-j)$ and $t_2(i, j)$;

4) $s_2(i, wh-1-j)$ is decoded by stacking $t_1(i, j)$ and $t_2(i, wh-1-j)$;

with the use of the symbol $\otimes$, the four requirements read:

$$s'_1(i, j) = t_1(i, j) \otimes t_2(i, j);$$
$$s'_1(i, wh-1-j) = t_1(i, wh-1-j) \otimes t_2(i, wh-1-j);$$
$$s'_2(i, j) = t_1(i, wh-1-j) \otimes t_2(i, j);$$
$$s'_2(i, wh-1-j) = t_1(i, j) \otimes t_2(i, wh-1-j). \quad (2.1)$$

Here, $[s'_1(i, j), s'_1(i, wh-1-j), s'_2(i, j), s'_2(i, wh-1-j)]$ are the *stacking results* to show the quadruple $[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]$. Since we are dealing with visual decoding, the stacking results $[s'_1(i, j), s'_1(i, wh-1-j), s'_2(i, j), s'_2(i, wh-1-j)]$ do not need to be completely identical to the original secret values $[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]$. Therefore, a prime symbol has been added to s to denote the stacking result.

Definition 2.2. The 16 basis matrices of a Flip VC (FVC) system are defined according to Fig. 2.2. In detail, each FVC system is defined according to its $2^4 = 16$ basis matrices $\{C_{WWWW}, C_{WWWB}, C_{WWBW}, \ldots, C_{BBBW}, C_{BBBB}\}$ of 4-by-r each, and r is a constant. All 4-by-r elements of each basis matrix $C_{[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]} \in \{C_{WWWW}, C_{WWWB}, C_{WWBW}, \ldots, C_{BBBW}, C_{BBBB}\}$ are 1-bit in value. Notably, $s_1(i, j) \in \{W, B\}$, and so are the values of $s_1(i, wh-1-j)$, $s_2(i, j)$, and $s_2(i, wh-1-j)$. Hence, there are $2^4=16$ basis matrices to cover the 16 possible readings $\{WWWW, WWWB, \ldots, BBBB\}$ of the 4-dimensional input vector $[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]$.

■

In the definition above, we stated that each FVC system is defined according to its $2^4=16$ basis matrices. This is because people can use the 16 basis matrices to encode any two secret images S_1 and S_2 to get two transparencies. In general, to encode four secret pixels $[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]$ grabbed from secret images S_1 and S_2, just choose randomly a column from the corresponding basis matrix $C_{[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]}$, then copy the

four elements of the chosen column to the four transparency pixels $t_1(i, j)$, $t_1(i, wh-1-j)$, $t_2(i, j)$, $t_2(i, wh-1-j)$ of T_1 and T_2, respectively.

To make sure the generated transparencies are secure and useful in unveiling the input secret images, the 16 basis matrices must satisfy the following Security and Contrast constraints. If these two constraints are satisfied, then the FVC defined by these sixteen basis matrices is called a *valid* FVC.

I. (Security constraint). In each 4-by-r basis matrix, the first and the second rows together consist of $a_0^{\mathrm{U}} \times r$ columns of $[0\ 0]^{\mathrm{T}}$, $a_1^{\mathrm{U}} \times r$ columns of $[0\ 1]^{\mathrm{T}}$, $a_2^{\mathrm{U}} \times r$ columns of $[1\ 0]^{\mathrm{T}}$, $a_3^{\mathrm{U}} \times r$ columns of $[1\ 1]^{\mathrm{T}}$, where

$$a_0^{\mathrm{U}} + a_1^{\mathrm{U}} + a_2^{\mathrm{U}} + a_3^{\mathrm{U}} = 1. \tag{2.2}$$

The value of a_i^{U} used by any two basis matrices must be identical. Likewise, the third and the fourth rows together consist of $a_0^{\mathrm{L}} \times r$ columns of $[0\ 0]^{\mathrm{T}}$, $a_1^{\mathrm{L}} \times r$ columns of $[0\ 1]^{\mathrm{T}}$, $a_2^{\mathrm{L}} \times r$ columns of $[1\ 0]^{\mathrm{T}}$, and $a_3^{\mathrm{L}} \times r$ columns of $[1\ 1]^{\mathrm{T}}$, where

$$a_0^{\mathrm{L}} + a_1^{\mathrm{L}} + a_2^{\mathrm{L}} + a_3^{\mathrm{L}} = 1. \tag{2.3}$$

The value of a_i^{L} used by any two basis matrices must be identical.

II. (Contrast constraint). Get the contrast according to the contrast evaluation process stated below. The contrast constraint requires that the obtained value α must be positive.

Contrast evaluation: The contrast of a Flip VC is evaluated in the following manner. For each basis matrix, items 1-4 are evaluated below:

1. the *luminance transmission* of $s_1(i, j)$, which is the percentage of 0s in the stacking result when the 1st and 3rd rows are stacked;
2. the *luminance transmission* of $s_1(i, wh-1-j)$, which is the percentage of 0s in the stacking result when the 2nd and 4th rows are stacked;
3. the *luminance transmission* of $s_2(i, j)$, which is the percentage of 0s in the stacking result when the 2nd and 3rd rows are stacked;
4. the *luminance transmission* of $s_2(i, wh-1-j)$, which is the percentage of 0s in the stacking result when the 1st and 4th rows are stacked.

Then, since each of the four pixels $s_1(i, j)$, $s_1(i, wh-1-j)$, $s_2(i, j)$, and $s_2(i, wh-1-j)$ only

have two possible values $\{W, B\}$, there are 2^4=16 basis matrices (e.g. Table 2.1). These 16 matrices are distinguished from each other using a quadruple naming system. For example, if $[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]$ is $[B,W,B,B]$, then the corresponding basis matrix is called $C_{BWBB.}$ Now, for each of these 16 matrices, measure its luminance transmission of $s_1(i, j)$. If the first subscript in the matrix name is B, i.e., if $s_1(i, j)=B$, then store its luminance transmission of $s_1(i, j)$ in a pool called Black-pool. Otherwise, store it in a so-called White-pool (Therefore, 16/2=8 of the 16 luminance transmission of $s_1(i, j)$ will be in Black-pool, and the remaining 16–8=8 will be in White-pool.). After that, for each of the 16 basis matrices, measure its luminance transmission of $s_1(i, wh-1-j)$. If the second subscript in the matrix name is B, then store its luminance transmission of $s_1(i, wh-1-j)$ in a pool called Black-pool. Otherwise, store it in a so-called White-pool. Repeat this process analogously for the 16 luminance transmissions of $s_2(i, j)$ according to the third subscript of the matrices' names. Also repeat this process analogously for the 16 luminance transmissions of $s_2(i, wh-1-j)$ according to the fourth subscript of the matrices' names. Together, we have 8+8+8+8=32 numbers in the Black-pool, and 8+8+8+8=32 numbers in the White-pool. The minimum of the 32 numbers in White-pool is called w (the minimal *luminance transmission* to represent W), and the maximum of the 32 numbers in Black-pool is called b (the maximal *luminance transmission* to represent B). Define contrast α as

$$\alpha = w - b > 0. \tag{2.4}$$

Remark. In all VC methods, the stacking result is always with a contrast value smaller than 100% –0%=100%=1, and this makes the stacking result always looks less clear than the input secret image (for example, compare Fig. 1.1(a) and Fig. 1.1(d)). In general, contrast is an important measure specifying the visual quality of the stacking result for a VC method. Roughly speaking, a decoded result with higher contrast is usually clearer.

Theorem 2.1. When a FVC defined by 16 basis matrices satisfy the Security and the Contrast constraint addressed in Definition 2.2, then the generated transparencies are secure and useful in unveiling the input secret images.

Proof: **i)** About the *Security constraint*, its purpose is that: no information about the two secret images can be extracted if someone only gets a transparency. Below we prove the security of the two secret images when someone only obtains transparency T_1. (The proof is likewise if T_1 is replaced by transparency T_2).

The definition of Security constraint reads that "The value of a_i^{U} used by any two basis matrices must be identical (This cross-matrices requirement also holds for a_i^{L}, respectively.)". Hence, if a set of basis matrices do not satisfy the security constraint, then the value of a_i^{U} (or a_i^{L}) used by some basis matrices may be different. For example, if the matrix C_{BBBB} in Table 2.1 is replaced by

$$C'_{BBBB} = \begin{matrix} t_1(i,j) \\ t_1(i,m-1-j) \\ t_2(i,j) \\ t_2(i,m-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix},$$

then the first and the second rows in C'_{BBBB} are with $[0\ 0]^{\mathrm{T}}\times 1$, $[0\ 1]^{\mathrm{T}}\times 1$, $[1\ 0]^{\mathrm{T}}\times 0$, and $[1\ 1]^{\mathrm{T}}\times 4$, while the first and second rows in remaining 16–1=15 matrices of Table 2.1 are with $[0\ 0]^{\mathrm{T}}\times 1$, $[0\ 1]^{\mathrm{T}}\times 1$, $[1\ 0]^{\mathrm{T}}\times 1$, and $[1\ 1]^{\mathrm{T}}\times 3$. Since the first and second rows are used to encode $t_1(i, j)$ and $t_1(i, wh-1-j)$ in the same transparency T_1, so if an intruder finds in T_1 a pair of pixels $[t_1(i, j), t_1(i, wh-1-j)]=[1, 1]$, then his best guess of the four corresponding secret pixels in secret images S_1 and S_2 should be [*BBBB*]. Likewise, if he finds in T_1 a pair of pixels $[t_1(i, j), t_1(i, wh-1-j)]=[1, 0]$, then he knows that the four corresponding secret pixels cannot be [*BBBB*]. In summary, the transparency T_1 is not a secure transparency, because it has secret-leaking problem.

The paragraph above shows the necessity of the security constraint (to ensure that no information about the secret images can be extracted). Below we show the sufficiency of the security constraint. Assume a set of sixteen basis matrices satisfies the security constraint. Therefore, in each 4-by-r basis matrix, the first and the second rows together consist of $a_0^{\mathrm{U}}\times r$ columns of $[0\ 0]^{\mathrm{T}}$, $a_1^{\mathrm{U}}\times r$ columns of $[0\ 1]^{\mathrm{T}}$, $a_2^{\mathrm{U}}\times r$ columns of $[1\ 0]^{\mathrm{T}}$, $a_3^{\mathrm{U}}\times r$ columns of $[1\ 1]^{\mathrm{T}}$, where the value of a_i^{u} used by any two basis matrices must be identical.

Since the first and second rows are utilized to encode $t_1(i, j)$ and $t_1(i, wh-1-j)$ in the same transparency T_1, so if an intruder gets a single transparency T_1 and he finds in T_1 a pair of pixels $[t_1(i, j), t_1(i, wh-1-j)]=[0, 0]$, then he cannot know whether the four corresponding secret pixels in secret images S_1 and S_2 should be [*WWWW*] or [*WWWB*] or …. or [*BBBB*]. This is because each of the 2^4=16 basis matrices has the same number of columns ($a_0^{\mathrm{U}}\times r$ columns) read as $[0\ 0]^{\mathrm{T}}$ when the first two rows of the matrix is grabbed. Therefore, there are 1/16 chance that $[0\ 0]^{\mathrm{T}}$ was from secret pixels [*WWWW*]. Similarly, there are 1/16 chance that

$[0\ 0]^T$ was from secret pixels [*WWWB*]. Similarly, there are 1/16 chance that $[0\ 0]^T$ was from secret pixels [*WWBW*]. In fact, the same 1/16 chance holds for each of the sixteen basis matrices.

Therefore, the intruder cannot know whether the four corresponding secret pixels in secret images S_1 and S_2 should be [*WWWW*] or [*WWWB*] or …. or [*BBBB*]. The above analysis still holds if $[0\ 0]^T$ is replaced by $[0\ 1]^T$ or $[1\ 0]^T$ or $[1\ 1]^T$. Therefore, no matter what the contents of two secret images S_1 and S_2 are, the transparency T_1 is always of perfect security: no secret-leaking will occur. The perfect security of transparency T_2 can be proved likewise using the third and fourth rows of the 16 basis matrices, as defined in the second half of the security constraint.

ii) About the *Contrast constraint*, the definition is in Eq. (2.4) which reads $\alpha = w-b>0$. If the value of α is not positive, then there are two possible cases:

Case 1. ($\alpha=0$). In this case, we cannot see the information in the stacking result, because the luminance transmission of representing *W* and *B* are identical.

Case 2. ($\alpha<0$). In this case, the luminance transmission to represent *W* is smaller than the luminance transmission to represent *B*. Then we will see that *W* is darker than *B*, and the stacking result will look like the negative film of a photo, an inappropriate view. ▪

2.1.2 The 16 basis matrices of opaque-oriented FVC

In this section, we use 16 basis matrices of 6 columns each to encode the quadruple secret pixels $[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]$. Each generated transparency will be of perfect security by using the 16 basis matrices to encode. The word "opaque-oriented" means the stacking result representing black pixels in the secret image is restricted to be 100% opaque. The security and contrast are addressed below.

Property 2.1. The set of basis matrices shown in Table 2.1 is a valid FVC and it satisfies the security and the contrast of stacking result is 1/6.

Proof: Table 2.1 shows a set of 16 basis matrices mentioned below Definition 2.2. In the 1st and 2nd Rows of *each* basis matrix shown in Table 2.1, there are (1/6)×6=1 column of $[0\ 0]^T$, (1/6)×6=1 column of $[0\ 1]^T$, (1/6)×6=1 column of $[1\ 0]^T$, and (3/6)×6=3 columns of $[1\ 1]^T$. Hence, the cross-matrices constant-ratio $(a_0^U : a_1^U : a_2^U : a_3^U)$ requirement mentioned below Eq. (2.2) holds. In the 3rd and 4th Rows of each basis matrix, the cross-matrices constant-ratio

$(a_0^L : a_1^L : a_2^L : a_3^L)$ requirement mentioned below Eq. (2.3) also holds. The cross-matrices property required by the Security Constraint is thus satisfied. Moreover, after the computation stated below, it can be shown that w=1/6 and b=0, so the contrast α is 1/6–0=1/6 which is a positive number, and hence the Contrast Constraint is also satisfied by the Flip VC defined using Table 2.1.

The detail computation of w and b for Table 2.1 is as follows. First, statements 1-4 below are true for each basis matrix in Table 2.1. Therefore, every element of the Black-pool is 0, and each element of the White-pool is 1/6. Because the maximum element of the Black-pool (i.e. b) is 0 and the minimum element of the White-pool (i.e. w) is 1/6, contrast α is thus 1/6–0=1/6.

1. When the 1st and 3rd rows are stacked, if the first subscript in the matrix name is *B*, then the ratio of 0s in the stacking result is 0%; otherwise, the ratio is 1/6=16.7%.
2. When the 2nd and 4th rows are stacked, if the second subscript in the matrix name is *B*, then the ratio of 0s in the stacking result is 0%; otherwise, the ratio is 1/6=16.7%.
3. When the 2nd and 3rd rows are stacked, if the third subscript in the matrix name is *B*, then the ratio of 0s in the stacking result is 0%; otherwise, the ratio is 1/6=16.7%.
4. When the 1st and 4th rows are stacked, if the forth subscript in the matrix name is *B*, then the ratio of 0s in the stacking result is 0%; otherwise, the ratio is 1/6=16.7%.

■

We explain below in more detail what the two ratios 0% and 16.7% stand for. According to Fig. 2.2, each of the four secret-pixels in $[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]$ is recovered by tracing its two arrows in Fig. 2.2 back to two of the four transparency-pixels in $[t_1(i, j), t_1(i, wh-1-j), t_2(i, j), t_2(i, wh-1-j)]$. For example, the recovered version of secret pixel $s_1(i, j)$ is obtained by $s'_1(i, j)=t_1(i, j)\otimes t_2(i, j)$; whereas the recovered version of secret pixel $s_2(i, j)$ is obtained by $s'_2(i, j)=t_1(i, wh-1-j)\otimes t_2(i, j)$. As for the encoding to generate the two transparencies t_1 and t_2, note that each 4-by-6 basis matrix in Table 2.1 has 6 columns; so, in the encoding process, each time an input quadruple $[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]$ is given, there are 6 possible ways to encode this quadruple. For example, if the input secret quadruple $[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]$ is [*W*,*W*,*B*,*B*], then $[t_1(i, j), t_1(i, wh-1-j), t_2(i, j), t_2(i, wh-1-j)]$ is encoded as [1,0,1,0] if the third column of the basis matrix C_{WWWB} in Table 2.1 is selected. Likewise, $[t_1(i, j), t_1(i, wh-1-j), t_2(i, j), t_2(i, wh-1-j)]$ is

encoded as [1, 1, 0, 0] if the sixth column of matrix C_{WWWB} is selected in the random-selection process. Notably, the index *WWBB* means that the input quadruple secret pixels are $s_1(i, j)=0$, $s_1(i, wh-1-j)=0$, $s_2(i, j)=1$, $s_2(i, wh-1-j)=1$. Now, no matter which of the six columns of matrix C_{WWBB} is selected, the value $s'_2(i, wh-1-j) = t_1(i, j) \otimes t_2(i, wh-1-j)$ obtained by stacking is always 1, because (1^{st} Row)$\otimes$(4^{th} Row) = [1, 1, 1, 1, 1, 1] for matrix C_{WWWB} of Table 2.1, and so is $s'_2(i, j)$. However, the value $s'_1(i, j) = t_1(i, j) \otimes t_2(i, j)$ obtained by stacking is not always 0, because (1^{st} Row)$\otimes$(3^{rd} Row) = [1, 0, 1, 1, 1, 1] for that matrix C_{WWBB}. In other words, depending on which of the six columns is selected, the chance that $t_1(i, j) \otimes t_2(i, j) = 0$ is only 1/6=16.7%. Similar argument also shows that the chance that $t_1(i, wh-1-j) \otimes t_2(i, wh-1-j) = 0$ is only 1/6=16.7%, too. Moreover, for each of the 16 basis matrices in Table 2.1, the probability that the stacking result can recover a black secret pixel (i.e. a secret pixel with value 1) is always 100%; but the probability that the stacking result can recover a white secret pixel (i.e. a secret pixel with value 0) is always 1/6=16.7%, rather than 100%. As a result, the black area of the input secret images is still black after stacking the two transparencies; however, since the six columns of each basis matrix in Table 2.1 is randomly selected, the white area of the input secret images looks gray (rather than plain white). This is because in each white area, the area is formed of many pixels, and after stacking the two transparencies, 16.7% of these pixels are white while 83.3% of these pixels are black. From the view of human vision (recalling that the decoder is human eyes rather than computers), since 83.3% of the pixels in a white area is black (opaque) and 16.7% of the pixels in the same white area is white (transparent), the whole white area looks like dark-gray in brightness, rather than plain white. Therefore, the white area of the original input image looks brighter than the corresponding area of the stacked output. Notably, darker output in white area is a very common phenomenon for any VC approach. For example, in Fig. 1.1, which shows the stacking result of the VC method proposed by Noar and Shamir [1], the input image's white area also becomes darker after VC's encoding-then-stacking.

Table 2.1. The 16 basis matrices corresponding to the 2^4=16 combinations of [$s_1(i, j)$, $s_1(i, wh-1-j)$, $s_2(i, j)$, $s_2(i, wh-1-j)$], respectively. Some basis matrices (C_{WWWB}, C_{WWBW}, C_{WBWW}, and C_{BWWW}) have two forms, but only one form is needed in encoding. The user has freedom to choose the form he wants.

$$C_{WWWW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$C_{WWWB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}, \text{ or } \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$C_{WWBW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}, \text{ or } \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$C_{WWBB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$C_{WBWW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}, \text{ or } \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$C_{WBWB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$C_{WBBW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

$$C_{WBBB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$C_{BWWW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}, \text{ or } \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$C_{BWWB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$C_{BWBW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$C_{BWBB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$C_{BBWW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$C_{BBWB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$C_{BBBW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$C_{BBBB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

2.1.3 The 16 basis matrices of non-opaque-oriented FVC

This section presents the 16 basis matrices of 8 columns each (rather than 6 columns) to encode the quadruple secret pixels [$s_1(i, j)$, $s_1(i, wh-1-j)$, $s_2(i, j)$, $s_2(i, wh-1-j)$]. The word "non-opaque-oriented" means the stacking result representing black pixels in the secret image is not restricted to be 100% opaque. The security and contrast are addressed below.

Property 2.2. The set of basis matrices shown in Table 2.2 is a valid FVC and it satisfies the security and the contrast of stacking result is 1/4.

Proof: Table 2.2 shows a set of 16 basis matrices mentioned in Definition 2.2. In Rows 1 and 2 of *each* basis matrix shown in Table 2.2, there are (2/8)×8=2 column of $[0\ 0]^T$, (2/8)×8=2 column of $[0\ 1]^T$, (2/8)×8=2 column of $[1\ 0]^T$, and (2/8)×8=2 columns of $[1\ 1]^T$. Hence, the cross-matrices constant-ratio $(a_0^U : a_1^U : a_2^U : a_3^U)$ requirement mentioned below Eq. (2.2) holds. In Rows 3 and 4 of each basis matrix, the cross-matrices constant-ratio $(a_0^L : a_1^L : a_2^L : a_3^L)$ requirement mentioned below Eq. (2.3) also holds. The cross-matrices property required by the Security Constraint is thus satisfied. Moreover, after the computation stated below, it can be shown that w=3/8 and b=1/8, so the contrast α is 3/8－1/8=1/4 which is a positive number, and hence the Contrast Constraint is also satisfied by the FVC defined using Table 2.2.

The detail computation of w and b for Table 2.2 is as follows. First, statements 1-4 below are true for each basis matrix in Table 2.2. Therefore, every element of the Black-pool is 1/8, and each element of the White-pool is 3/8. So the maximum element of the Black-pool (i.e. b) is 1/8, and the minimum element of the White-pool (i.e. w) is 3/8, and contrast α is thus 3/8－1/8=1/4.

1. When the 1st and 3rd rows are stacked, if the first subscript in the matrix name is *B*, then the ratio of 0s in the stacking result is 1/8=12.5%; otherwise, the ratio is 3/8=37.5%.
2. When the 2nd and 4th rows are stacked, if the second subscript in the matrix name is *B*, then the ratio of 0s in the stacking result is 1/8=12.5%; otherwise, the ratio is 3/8=37.5%.
3. When the 2nd and 3rd rows are stacked, if the third subscript in the matrix name is *B*, then the ratio of 0s in the stacking result is 1/8=12.5%; otherwise, the ratio is 3/8=37.5%.

4. When the 1st and 4th rows are stacked, if the forth subscript in the matrix name is *B*, then the ratio of 0s in the stacking result is 1/8=12.5%; otherwise, the ratio is 3/8=37.5.

■

Table 2.2. Encoding matrices of all combinations of [$s_1(i, j)$, $s_1(i, wh-1-j)$, $s_2(i, j)$, $s_2(i, wh-1-j)$].

$C_{WWWW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$
$C_{WWWB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$
$C_{WWBW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$
$C_{WWBB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$
$C_{WBWW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$
$C_{WBWB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$
$C_{WBBW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$

$$C_{WBBB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$C_{BWWW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$C_{BWWB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$C_{BWBW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$C_{BWBB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$C_{BBWW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

$$C_{BBWB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

$$C_{BBBW} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$C_{BBBB} = \begin{matrix} t_1(i,j) \\ t_1(i,wh-1-j) \\ t_2(i,j) \\ t_2(i,wh-1-j) \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

2.2 Experimental results

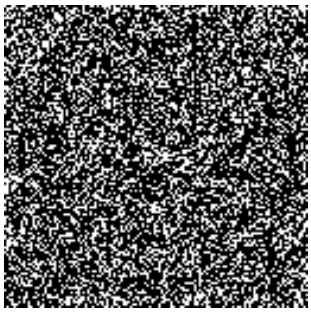
Experiments and comparisons are presented in this section. Sec. 2.2.1 presents the results of the proposed method. Sec. 2.2.2 gives the security testing of the transparencies. Sec. 2.2.3 shows the comparisons with other studies. Sec. 2.2.4 shows the expanded version of our method.

2.2.1 Experiments of proposed method

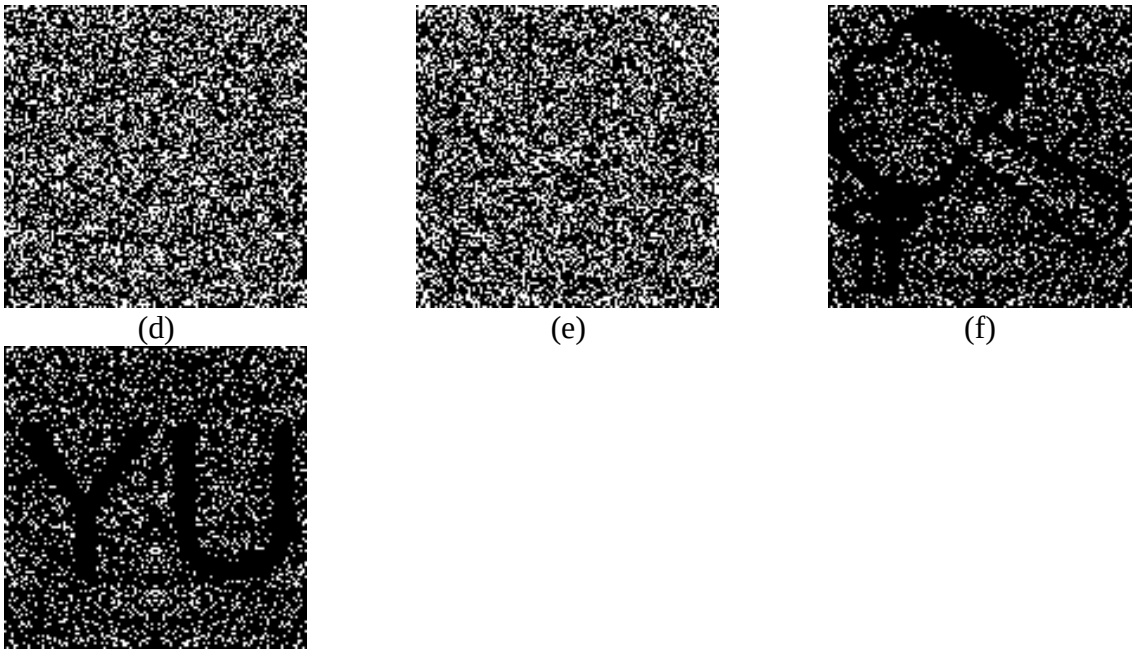
This subsection presents experimental results for the proposed scheme which can generate non-expanded transparencies with perfect security and can decode one more secret image by flipping one of the transparencies. The opaque-oriented FVC experiment is shown in Fig. 2.3. The two secret images are displayed in Figs. 2.3(a-b); and the two generated non-expanded transparencies are shown in Figs. 2.3(c-d). Fig. 2.3(e) shows the result of flipping Fig. 2.3(c) over. Fig. 2.3(f) shows the result of stacking Fig. 2.3(c) and Fig. 2.3(d) together; Fig. 2.3(g) shows the result of stacking 2.3(c) and 2.3(e) together. The non-opaque-oriented FVC experiment is shown in Fig. 2.4.

(a)

(b)



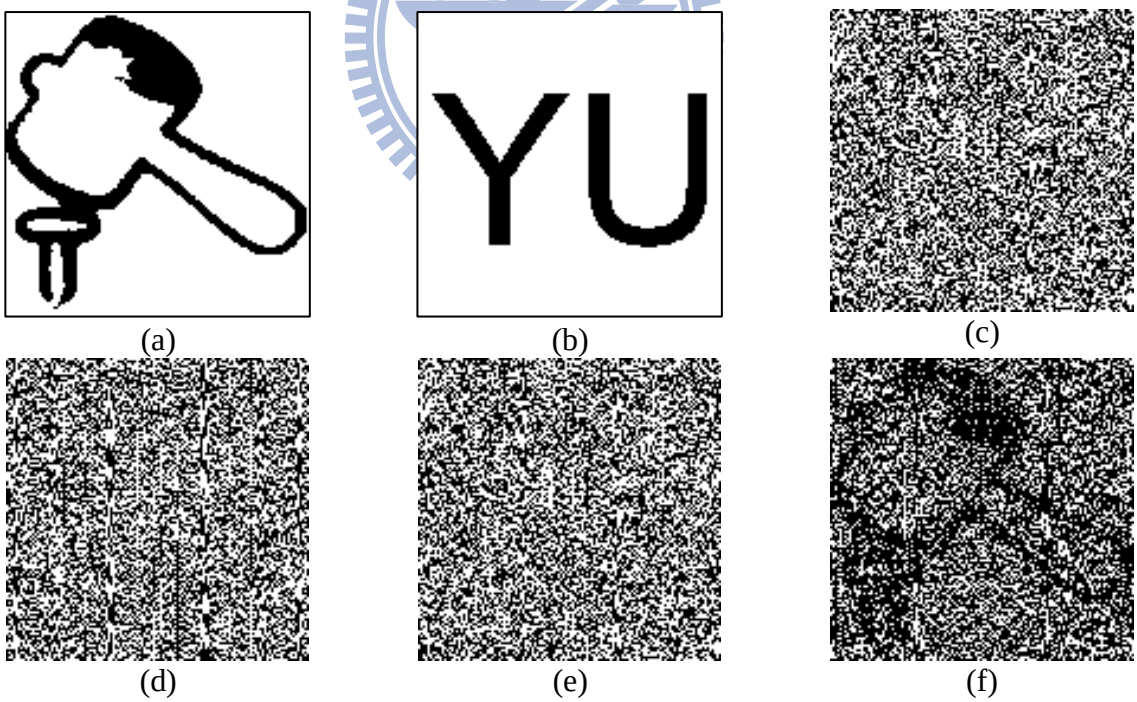
(c)



(d) (e) (f)

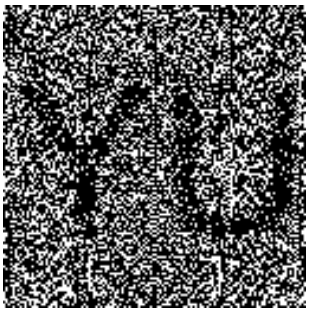
(g)

Fig. 2.3. The experimental result of the opaque-oriented FVC: (a-b): the secret images; (c-d): the two generated transparencies; (e): flipping (c) over; (f): the result of stacking (c) and (d) together; (g): the result of stacking (d) and (e) together.



(a) (b) (c)

(d) (e) (f)



(g)

Fig. 2.4. The experimental result of the non-opaque-oriented FVC: (a-b): the secret images; (c-d): the two generated transparencies; (e): flipping (c) over; (f): the result of stacking (c) and (d) together; (g): the result of stacking (d) and (e) together.

2.2.2 Security test of proposed method

In this subsection, we conduct two experiments for security testing. The first one is for scheme 1 and the second one is for scheme 2. Fig. 2.5 shows the first experiment. Figs. 2.5(a-b) illustrate two secret images S_1 and S_2, which consist of 16 sub-regions from top to down, and in each sub-region, $s_1(i, j)$ and $s_1(i, wh-1-j)$ are at the left-hand and right-hand sides of S_1, and $s_2(i, j)$ and $s_2(i, wh-1-j)$ are at the left-hand and right-hand sides of S_2. Then the four sections (the left-hand and right-hand sides of S_1 and the left-hand and right-hand sides of S_2) in each sub-region are painted using all possible colors {*WWWW, WWWB,…, BBBB*}. In other words, each sub-region is encoded with a basis matrix being referred to.

The generated transparencies T_1 and T_2 are shown in Figs. 2.5(c-d). The result of stacking T_1 and T_2 together is shown in Fig. 2.5(e). When T_1 is flipped and then stacked with T_2, the stacking result is shown in Fig. 2.5(f). To test security of T_1, in each sub-region of T_1, we count the probability distribution of symmetric pairs $[t_1(i, j), t_1(i, wh-1-j)] \in \{[0,0], [0,1], [1,0], [1,1]\}$. Fig. 2.5(g) shows the statistical result. The probability distributions are about [1/6, 1/6, 1/6, 3/6], no matter which basis matrix is used (the small variance is caused by randomly choosing a column in the basis matrix; so it is unrelated to the secret pixels, i.e. the intruder cannot judge the secret values by the small variance). Therefore, if an intruder only has T_1 (e.g. Fig. 2.5(c)), then no information about the secret image is unveiled. Fig. 2.5(h) shows statistical analysis of the second transparency. The result is similar to Fig. 2.5(g), so it is also secure. Fig. 2.6 shows the second experiment.

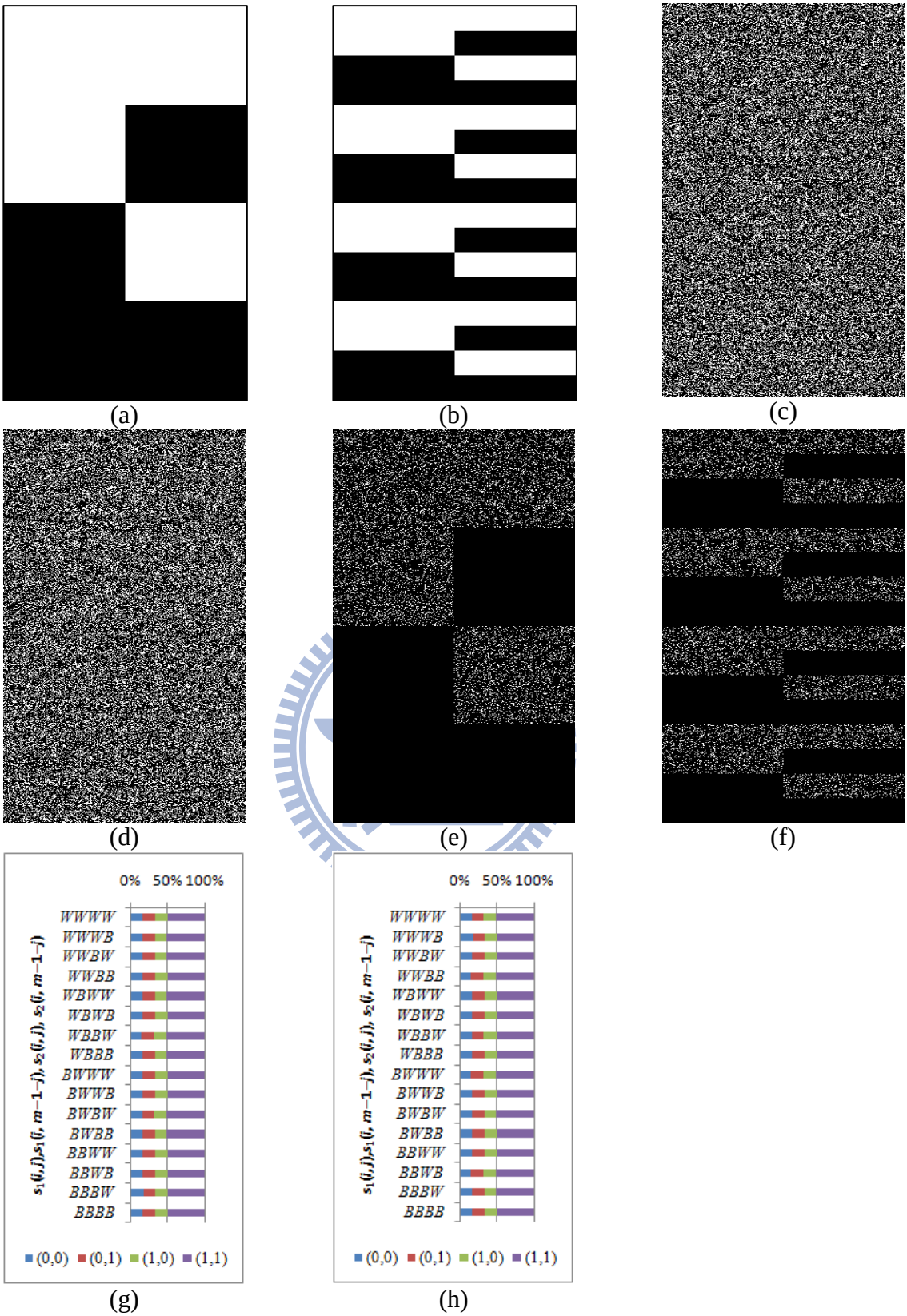


Fig. 2.5. Security test of scheme 1. (a-b): The two secret images; (c-d): The two generated transparencies; (e-f): The two stacking results; (g): Statistical result of (c); (h): Statistical result of (d).

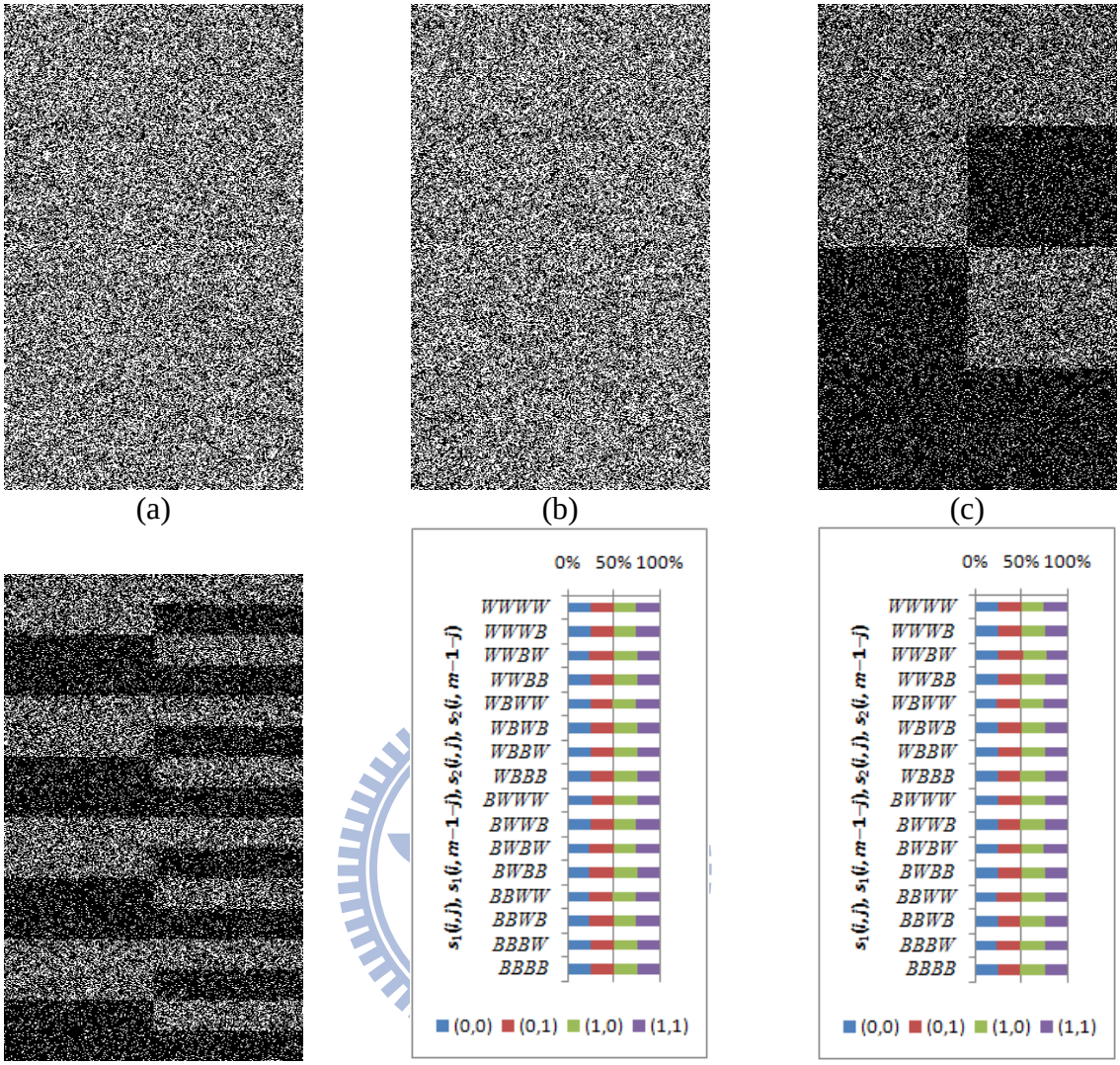


Fig. 2.6. Security test of scheme 2. (a-b): The two generated transparencies; (c-d): The two stacking results; (e): Statistical result of (a); (f): Statistical result of (b).

2.2.3 Comparison with other studies

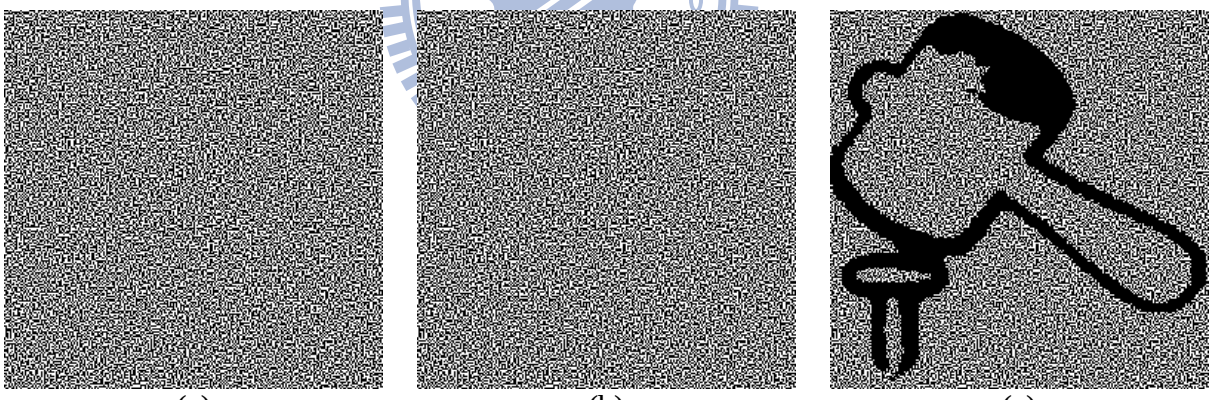
Table 2.3 lists the comparisons with previously reported VC methods [1, 3, 19, 20, 22]. Many reported methods had pixel expansion problem; and non-expanded methods often encoded only a single secret image. The proposed method encodes double secret images, and does not cause any pixel expansion.

Table 2.3. Characterization of VC methods

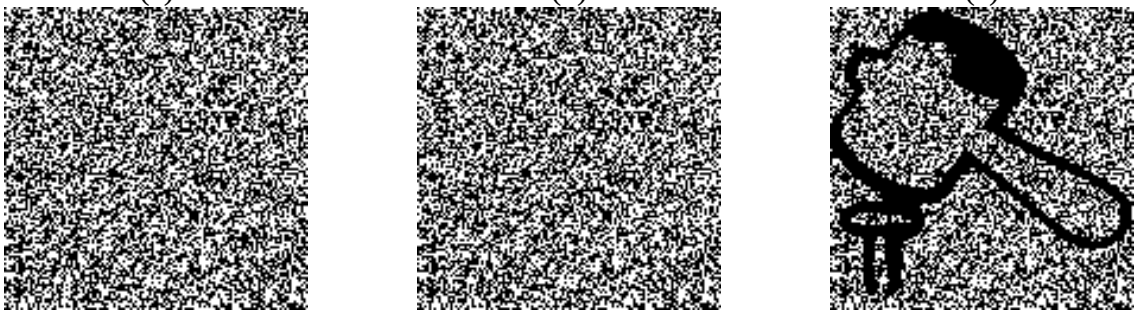
Methods	Pixel-expansion factor	Number of hidden secrets

Naor and Shamir [1]	4	Single
Yang's [20]	1	
Shyu [22]	1	
Wu and Chang [3]	4	double
Shyu et al [19]	4	
The proposed method	1	

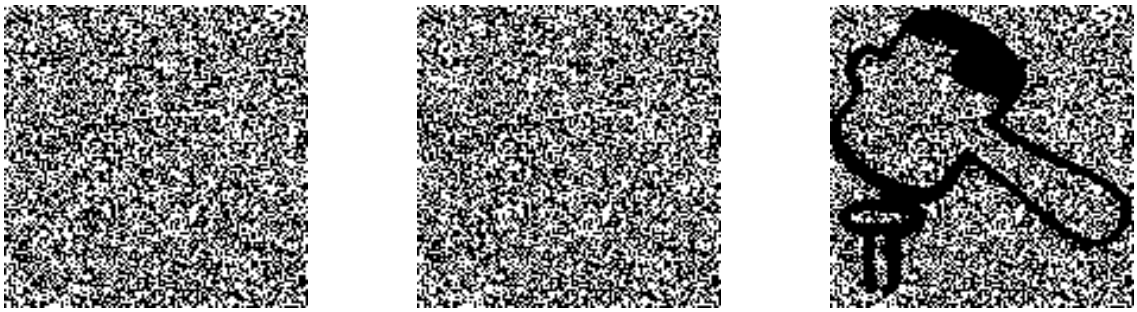
Previously reported VC methods [1, 3, 19, 20, 22] are implemented. First, single-secret VCs [1, 20, 22] are demonstrated in Fig. 2.7, and let the number of transparencies is two for each method. Figs. 2.7(a-c) show Naor and Shamir's method [1]. The expansion rate is 4 (ours is 1), and the contrast is 1/2 (ours is 1/6 or 1/4). Figs. 2.7(d-f) show Yang's method [20]. The contrast is 1/2, and the stacking result (f) is also tumultuous and hence not as good as Naor and Shamir's; but this is because there is no expansion (just like ours). Figs. 2.7(g-i) show Shyu's method [22], and the result is similar to Yang's. Notably, the three methods [1, 20, 22] only encode a single secret image in two transparencies, but the proposed method encode two secret images; so [1, 20, 22] has better visual quality than ours.



(a) (b) (c)



(d) (e) (f)



(g) (h) (i)

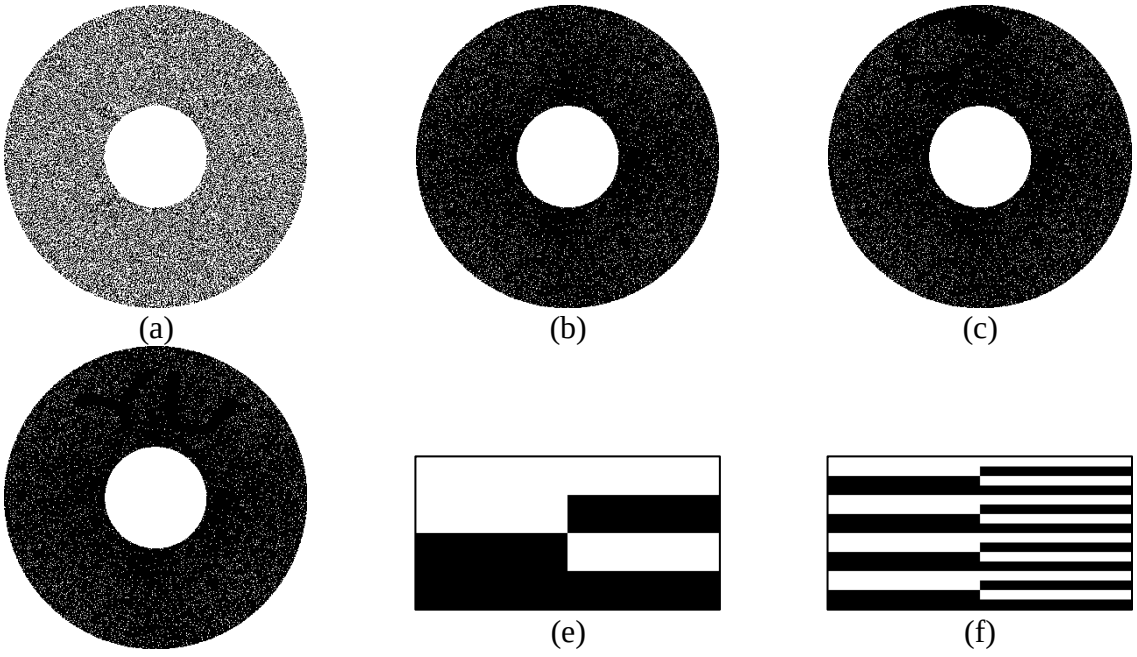
Fig. 2.7. Three “single-secret” (2, 2) VC methods. (a-c): Naor and Shamir’s method: (a-b) are the two generated transparencies, and (c) is the stacking result. (d-f): Yang’s method: (d-e) are the two generated transparencies, and (f) is the stacking result. (g-i): Shyu’s method: (g-h) are the two generated transparencies, and (i) is the stacking result.

Next, in Figs. 2.8 and 2.9, we demonstrate two circular VC methods [3, 19]. Both methods encode multiple secret images in two circular transparencies, and each secret image is revealed by stacking the two transparencies with a rotation of the first transparency using a pre-defined degree. To facilitate the comparison, let the number of secret images be two, and the rotational degrees be 0 degree and 180 degree.

Figs. 2.8(a-b) are the two circular transparencies T_1 and T_2 generated by Wu and Chang [3] in which the expansion rate is 4 (each secret pixel is represented as a 2×2 block in two transparencies), and the contrast is 1/4. Fig. 2.8(c) is the results of stacking T_1 and T_2; and Fig. 2.8(d) is the results in which T_1 is rotated 180 degree and stacked with T_2. Let *wh* denote the width of secret image, two pixels $t_1(i, j)$ and $t_1(i, j+wh/2)$ are at two opposite positions in T_1, and so are $t_2(i, j)$ and $t_2(i, j+wh/2)$ in T_2. In the stacking, the secret S_1 is revealed by stacking $t_1(i, j)$ with $t_2(i, j)$ to decode $s_1(i, j)$, and stacking $t_1(i, j+wh/2)$ with $t_2(i, j+wh/2)$ to decode $s_1(i, j+wh/2)$; the second secret is revealed by stacking $t_1(i, j+wh/2)$ with $t_2(i, j)$ to decode $s_2(i, j)$, and stacking $t_1(i, j)$ with $t_2(i, j+wh/2)$ to decode $s_2(i, j+wh/2)$. Therefore, the two pixel values $(t_1(i, j), t_1(i, j+wh/2))$ form a symmetric pair, and so do $(t_2(i, j), t_2(i, j+wh/2))$. Since the four secret pixels $[s_1(i, j)$, $s_1(i, j+wh/2)$, $s_2(i, j)$, $s_2(i, j+wh/2)]$ have 2^4=16 possible colors {*WWWW*, *WWWB*,…, *BBBB*}, to test the security of 16 types of colors, an experiment is shown in Figs. 2.8(e-l). Figs. 2.8(e-f) illustrate two secret images S_1 and S_2, which consist of 16 equal sub-regions from top to bottom, and in each sub-region, $s_1(i, j)$ and $s_1(i, j+wh/2)$ are at the left and right sides of S_1, and $s_2(i, j)$ and $s_2(i, j+wh/2)$ are at the left and right sides of S_2. Then those four sections (the left and right sides of S_1 and the left and right sides of S_2) in each sub-region are painted using all possible colors {*WWWW*, *WWWB*,…, *BBBB*},

respectively. The generated transparencies are shown in Figs. 2.8(g-h). Fig. 2.8(i) shows the result of stacking T_1 and T_2, and Fig. 2.8(j) is the result of stacking the rotated T_1 with T_2.

To inspect the security issue of transparency T_1, Fig. 2.8(k) displays the probability distribution of symmetric pairs $[t_1(i, j), t_1(i, j+wh/2)] \in \{[0,0], [0,1], [1,0], [1,1]\}$ in each sub-region, where the probabilities are [1/4, 1/4, 1/4, 1/4] for all types of colors. Therefore, the first transparency is secure, because the intruder cannot judge the values of secret pixels $[s_1(i, j), s_1(i, j+wh/2), s_2(i, j), s_2(i, j+wh/2)]$ by observing the probability distribution of symmetric pairs. However, as shown in Fig. 2.8(l). The transparency T_2 leaks some information; because in S_1 and S_2, when the four secret pixels $[s_1(i, j), s_1(i, j+wh/2), s_2(i, j), s_2(i, j+wh/2)] \in \{WWWW, WBBW, BWWB, BBBB\}$, then the probability distribution of symmetric pairs $[t_2(i, j), t_2(i, j+wh/2)] \in \{[0,0], [0,1], [1,0], [1,1]\}$ is [1/4, 0, 0, 3/4]; when $[s_1(i, j), s_1(i, j+wh/2), s_2(i, j), s_2(i, j+wh/2)] \notin \{WWWW, WBBW, BWWB, BBBB\}$, then the probabilities are [0, 1/4, 1/4, 2/4], respectively. Hence, the intruder can judge whether the four secret pixels $[s_1(i, j), s_1(i, j+wh/2), s_2(i, j), s_2(i, j+wh/2)]$ are $\{WWWW, WBBW, BWWB, BBBB\}$ or not. In other words, if [0,0] pair or [1,1] pair appear in second transparency T_2, then we can claim that the corresponding position of secret images (S_1 and S_2) must be either [*WWWW*] or [*WBBW*] or [*BWWB*] or [*BBBB*]. Likewise, if [0,1] pair or [1,0] pair appear in second transparency T_2, then we can claim that the corresponding position of input images (S_1 and S_2) cannot be [*WWWW*] or [*WBBW*] or [*BWWB*] or [*BBBB*]. In summary, secret leaking occurs in T_2.



(a) (b) (c) (e) (f)

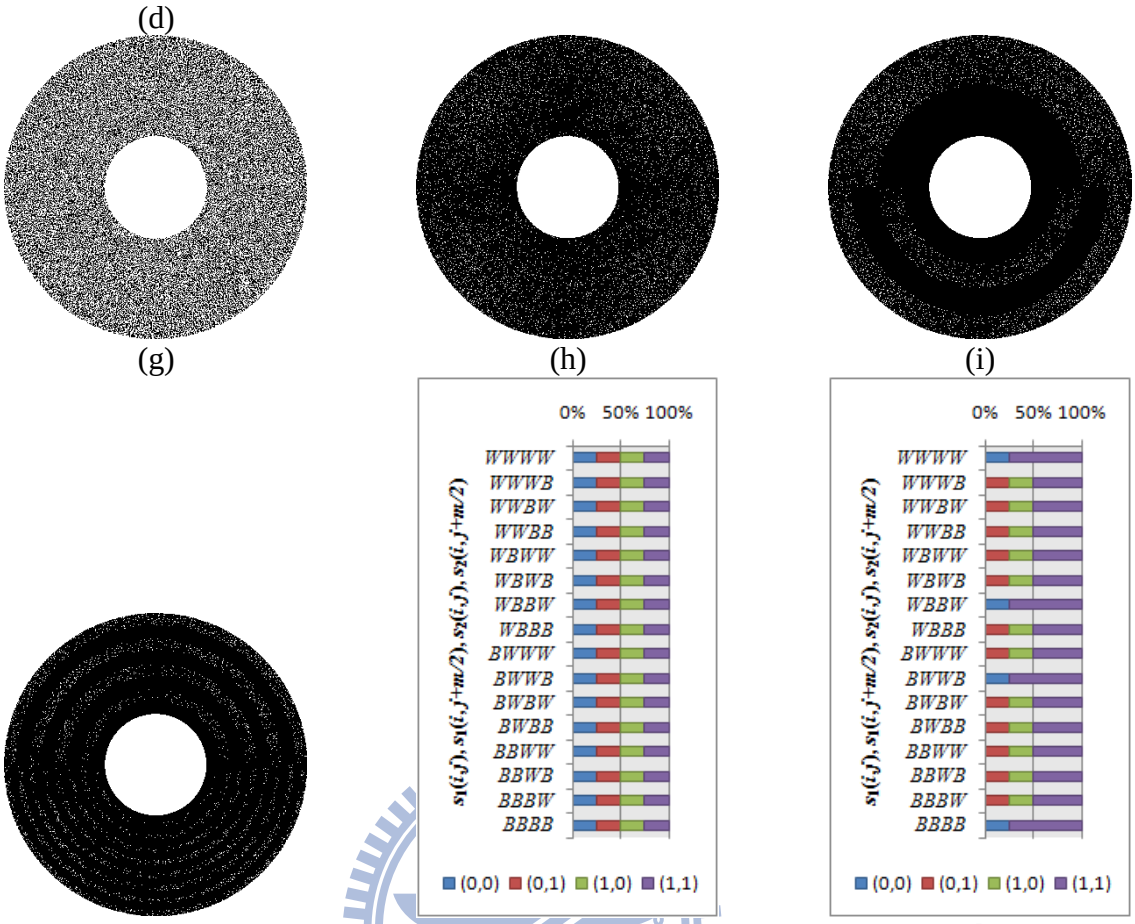
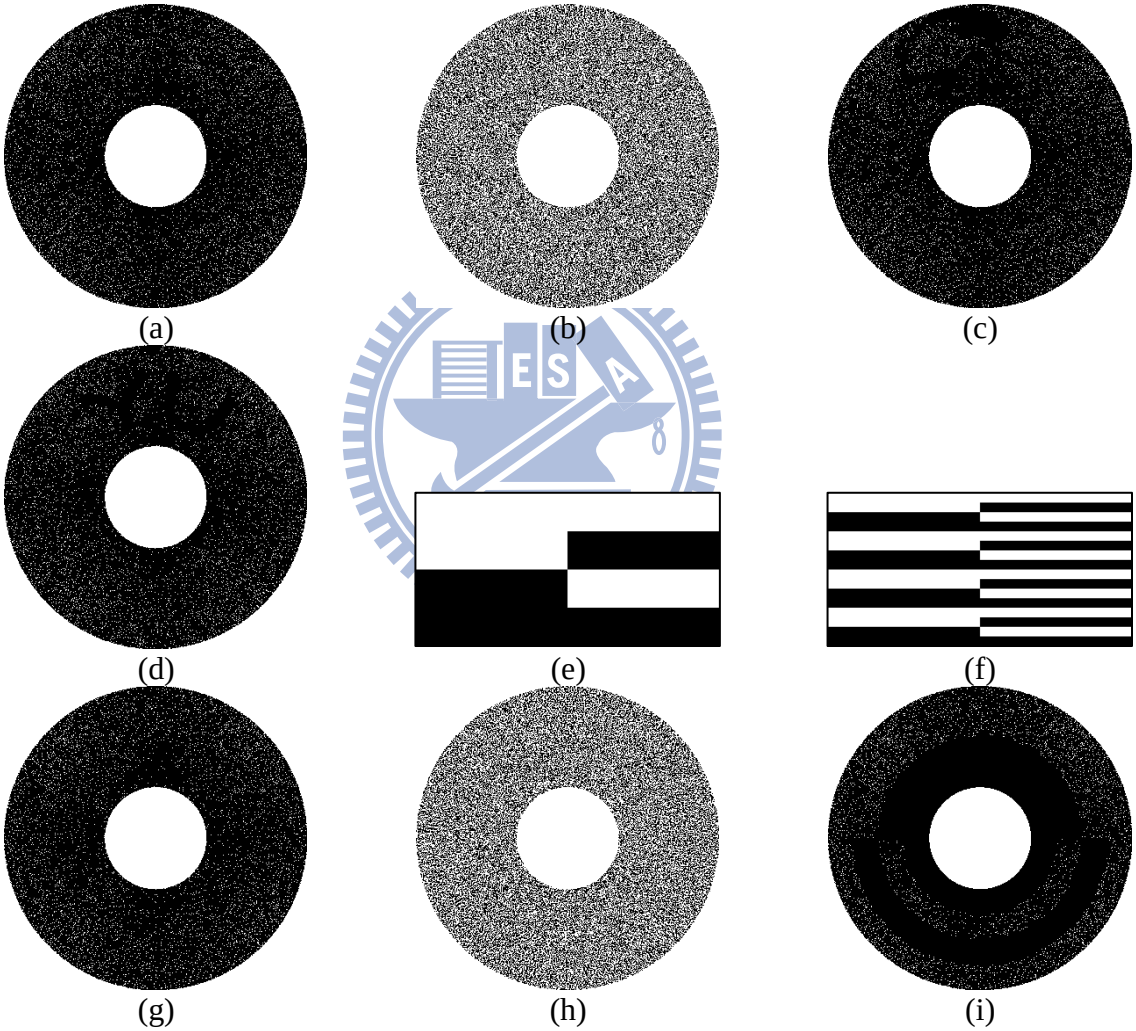


Fig. 2.8. About the method of Wu and Chang [3]. The expansion rate is 4. (a-b): Two generated circular transparencies T_1 and T_2. (c): The result of stacking T_1 and T_2. (d): The result of stacking rotational T_1 with T_2. (e-f): Two new secret images S_1 and S_2. (g-h): Two new transparencies T_1 and T_2 generated from S_1 and S_2. (i): The result of stacking T_1 with T_2. (j): The result of stacking rotational T_1 with T_2. (k): The probability distribution of symmetric pairs for all 16 sub-regions in T_1. (l): The probability distribution of symmetric pairs for all 16 sub-regions in T_2.

Fig. 2.9 is a demonstration about the method of Shyu et al. [19]. Figs. 2.9(a-b) are the two generated circular transparencies in which the expansion rate is 4, and the contrast is 1/4. Fig. 2.9(c) is the results of stacking (a) with (b), and Fig. 2.9(d) is the results of stacking rotated (a) with (b). The security test is shown in Figs. 2.9(e-l). Figs. 2.9(e-f) are the two new secret images which are the same as Figs. 2.8(e-f). Figs. 2.9(g-h) are the two generated transparencies, and the stacking results are Figs. 2.9(i-j). The security of T_1 is shown in Fig. 2.9(k), where the probability distribution of symmetric pairs $[t_1(i, j), t_1(i, j+wh/2)] \in \{(0,0), (0,1), (1,0), (1,1)\}$ is [0, 1/4, 1/4, 2/4] in all types of colors, so T_1 is secure. On the other hand,

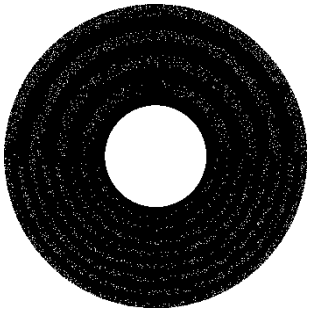
T_2 may leaks some information. The security of T_2 is shown in Fig. 2.9(l). When the four secret pixels $[s_1(i, j), s_1(i, j+wh/2), s_2(i, j), s_2(i, j+wh/2)] \in \{WWWW, WBBW, BWWB, BBBB\}$, the probability distribution of symmetric pairs is [1/2, 0, 0, 1/2]; when the four secret pixels are $\{WWBB, WBWB, BWBW, BBWW\}$, the probability distribution is [0, 1/2, 1/2, 0]. When the four secret pixels are in $\{BWWW, WBWW, WWBW, WWWB, WBBB, BWBB, BBWB, BBBW\}$, the probability distribution is [1/4, 1/4, 1/4, 1/4]. Therefore, the intruder can judge and divide the four secret pixels $[s_1(i, j), s_1(i, j+wh/2), s_2(i, j), s_2(i, j+wh/2)]$ to 3 sets by observing the probability distribution of symmetric pairs in transparency T_2.

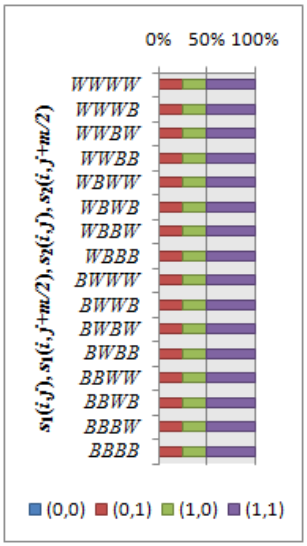


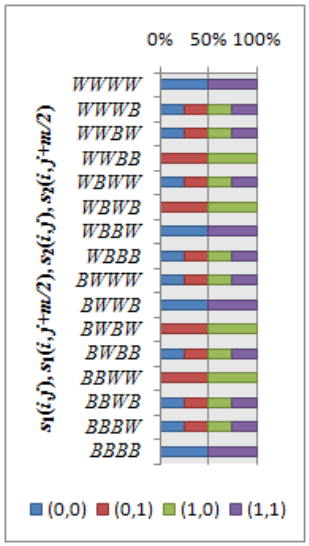
(a) (b) (c)

(d) (e) (f)

(g) (h) (i)







(j) (k) (l)

Fig. 2.9. About the method of Shyu et. al [19]. The expansion rate is 4. (a-b): Two generated circular transparencies T_1 and T_2. (c): The result of stacking T_1 and T_2. (d): The result of stacking rotational T_1 with T_2. (e-f): Two new secret images S_1 and S_2. (g-h): Two new transparencies T_1 and T_2 generated from S_1 and S_2. (i): The result of stacking T_1 with T_2. (j): The result of stacking rotational T_1 with T_2. (k): The probability distribution of symmetric pairs for all 16 sub-regions in T_1. (l): The probability distribution of symmetric pairs for all 16 sub-regions in T_2.

Figs. 2.8 and 2.9 show two well-known circular VCs [3, 19]. Stacking results of the two methods [3, 19] are 100% opaque both, and their contrast β=1/4 is better than our 1/6. But, as shown in Fig. 2.8(*l*) and Fig. 2.9(*l*), Methods [3, 19] are not of perfect security: the second transparency generated by [3, 19] have secret-leaking problem. In summary, under the constraint of avoiding secret-leaking (the fundamental requirement of VC), the best contrast value can be achieved is 1/6 (or 1/4, if the block pixels in stacking results are not restricted to 100% opaque), and ours already achieve this optimal contrast value 1/6 for scheme 1 (and 1/4 for Scheme 2). So we may say that ours are with conditionally optimal contrast under perfect-security requirement. As for others (e.g. [3, 19]), they might have contrast values better than ours, but it is because their methods did not meet perfect-security requirement.

2.2.4 The expanded version of our method

In order to yield no expansion, we use probability model to encode the shares. The price is that it may cause non-harmonic disarray of stacking result. If we are not constrained by the no-expansion rule, then all columns of basis matrix are used to encode the secret pixels [$s_1(i$,

j), $s_1(i, j+wh/2)$, $s_2(i, j)$, $s_2(i, j+wh/2)$], therefore, the expansion rate is the value of r. The results are as shown in Figs. 2.10 and 2.11. Fig. 2.10 shows the expanded version of flip scheme 1, with the expansion rate being 6=3×2. (Notably, r=6 is the minimal r we can have for Scheme 1. On the other hand, r will also be the expansion rate for our expanded version. So, in the expanded version, our minimal expansion rate will be 6 for Scheme 1 [8 for Scheme 2 because minimal r is 8 for scheme 2].) Fig. 2.10(a-b) shows the two generated transparencies, and (c-d) shows the stacking results. Fig. 2.11 shows the expanded version of flip scheme 2, with the expansion rate being 8=4×2. Fig. 2.11(a-b) shows the two generated transparencies and (c-d) show the stacking results. We can see that the visual quality is competitive again. In summary, the disarray of stacking result is due to the requirement of no-expansion, along with the perfect security for double secret; but the major weakness of pixel-expansion VC is that the size of transparencies will expand several times and waste space for carrying or storage.

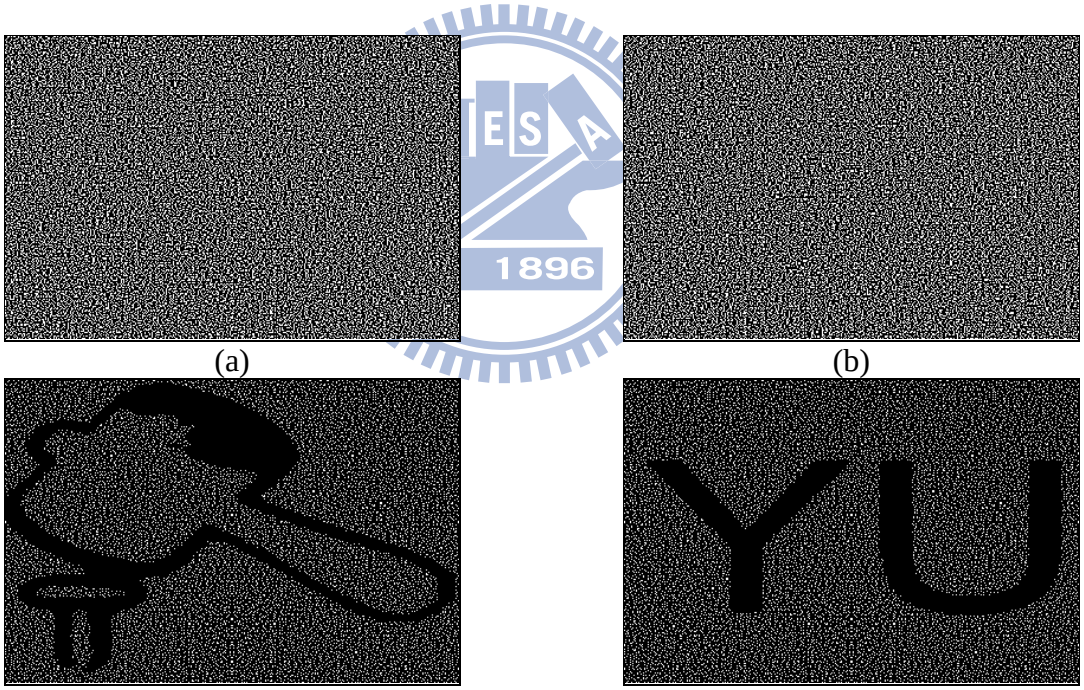
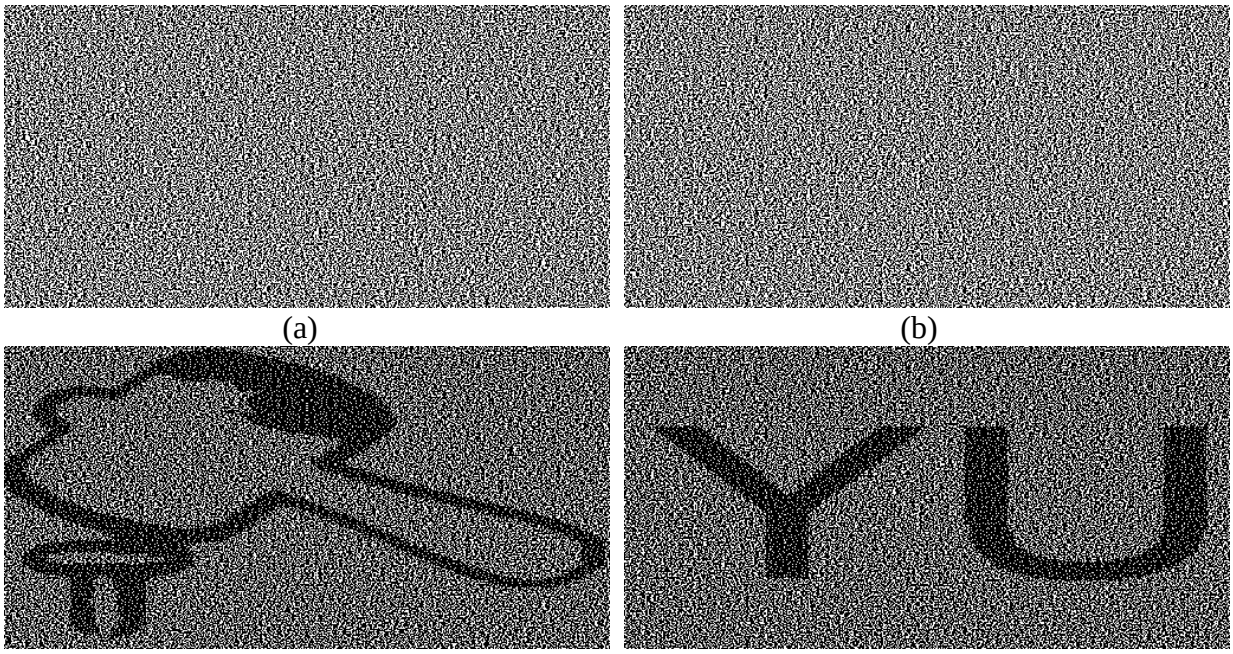


Fig. 2.10. The expanded version (block-based rather than pixel-based) of our scheme 1. (a-b): The two generated transparencies where the two secret images are Figs. 2.3(a-b). (c): The result of stacking (a) and (b). (d): The result of stacking (b) with the flipped version of (a).



(a) (b)

(c) (d)

Fig. 2.11. The expanded version (block-based rather than pixel-based) of our scheme 2. (a-b): The two generated transparencies where the two secret images are Figs. 2.3(a-b). (c): The result of stacking (a) and (b). (d): The result of stacking (b) with the flipped version of (a).

2.3 Discussions

In this section, some related topics are discussed in this section. Sec. 2.3.1 addresses the method of finding the basis matrices of FVC, and Sec. 2.3.2 shows the contrast values of the proposed method by other definition of contrast.

2.3.1 How to find the basis matrices of FVC

Basically, we may say that people can create these basis matrices by exhaustive search, as long as they meet the specified requirements. However, in reality, to save searching time, some basis matrices can be generated from others by exchanging rows. For instance, suppose the matrix C_{BWWW} of scheme 1 is set to

$$C_{BWWW} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix},$$

where the B is represented by stacking the 1st and 3rd rows to obtain six 1s, and the three W are represented by stacking the 2nd and 4th rows, the 2nd and 3th rows, and the 1st and 4th rows, respectively, to obtain one 0 and five 1s. Then the 1st and 2nd rows can be exchanged to get

$$C_{WWBW} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix},$$

where the B is by stacking the 2nd and 3rd rows, and the three W are by stacking the 1st and 3rd rows, the 2nd and 4th rows, and the 1st and 4th rows. Using this method, we can generate four basis matrices C_{BWWW} , C_{WBWW}, C_{WWBW} and C_{WWWB}, as long as one of the four matrices is found.

Actually, all the 16 basis matrices can be divided into 6 sets, namely, $\{C_{WWWW}\}$, $\{C_{BWWW}$, C_{WBWW}, C_{WWBW}, $C_{WWWB}\}$, $\{C_{WBBW}$, C_{BWWB}, C_{BWBW}, $C_{WBWB}\}$, $\{C_{WWBB}$, $C_{BBWW}\}$, $\{C_{BBBW}$, C_{BBWB}, C_{BWBB}, $C_{WBBB}\}$, and $\{C_{BBBB}\}$. In each set, only one matrix needs to be found, and the remaining is generated by exchanging the rows. Therefore, only 6 basis matrices are actually searched.

Next, to search the basis matrices, we need decide the value of r. The factor to determine the value r is the contrast of the constructed basis matrices. For basis matrices whose width is r, the possible contrast is $1/r$, $2/r$, $3/r$,..., r/r. In Sec. 2.1.2, we already proved that the upper bound of contrast of scheme 1 is 1/6. In symbols, the contrast is

$$\{i/r | r, i \in Z, 0 < i/r \le 1/6\}.$$

To reach 1/6 (the upper bound of contrast for Scheme 1), the value r must be a multiple of 6. If r is not a multiple of 6, then the possible contrast i/r cannot equal to 1/6, so the contrast will be less than 1/6.

Analogously, in Scheme 2, the width (i.e. value r) of basis matrices must be a multiple of 4, because in Sec. 2.2.2 we already proved that 1/4 is the upper bound of the contrast for Scheme 2. Unfortunately, when r=4, we could not find the basis matrices even after exhaustive search. So we tried r=8 and obtained the basis matrices shown in Table 2.2 whose contrast reached the upper bound 1/4.

2.3.2 Discussion about contrast values

In our method, the two definitions follow the basis matrices definitions which are given by Naor and Shamir[1], but some details are modified to conform to the structure of FVC. The (t, n) Visual cryptography, which is defined by Naor and Shamir [1], needs two basis matrices to encode a secret pixel which has only two values {W, B} in a secret image; however, our method needs consider four secret pixels simultaneously (two pixels in S_1 and

two pixels in S_2), so it needs 2^4=16 basis matrices to encode four pixels. However, the contrast evaluation in [1] did not consider the fact that, in darker image, human eyes have higher sensitiveness about (real-life-sense) contrast (This fact was mentioned in Ref. [69] by Liu et al.). To overcome the drawback, we bring up two schemes in the proposed method, where Scheme 1 set the black color of stacking result is 100% opaque, and scheme 2 do not set the constraint. We let the readers choose the one they like.

Liu et al. [69] gives a new definition of contrast for expanded VC (i.e. each secret pixel is encoded into many pixels in transparencies), but our design is a non-expanded VC (i.e. each secret pixel is encoded into a pixel in transparencies). For readers benefit, we also give the contrast value defined by Liu et al. [69], when the expanded VC version shown in Figs. 2.10 and 2.11 are used. The expanded Scheme 1 has contrast

$$\alpha^{\text{Liu}} = \frac{(h-l)m}{h(m-h)+l(m-l)+m^2} \overset{h=6,\,l=5,\,m=6}{=\!=\!=} \frac{6}{41} \approx 0.146,$$

and the expanded Scheme 2 has contrast

$$\alpha^{\text{Liu}} = \frac{(h-l)m}{h(m-h)+l(m-l)+m^2} \overset{h=7,\,l=5,\,m=8}{=\!=\!=} \frac{8}{43} \approx 0.186.$$

2.4 Conclusions

Opaque-oriented and non-opaque-oriented FVC schemes are both introduced in this chapter. We have proved that both schemes satisfy perfect security and they are conditionally optimal in contrast. The generated transparencies do not lead to any expansion of size. The experimental results show the revealing of double secrets via flipping and stacking the transparencies together.

Just like other VC methods, the whole decoding process uses no computer or any computation; so the decoding is very fast, and can be used in environment where computer is not stable or available. Due to the double-secrets feature of the proposed method, one of the applications is the double checking of ownership for personality identification. Since the size is non-expanded, the space needed to carry a transparency to a meeting is economic (size is the same as the space needed to carry an original image).

2.5 Appendix

In this section, the conditionally optimal contrast in opaque-oriented FVC and

non-opaque-oriented FVC are proven, respectively.

2.5.1. The proof of conditionally optimal contrast in opaque-oriented FVC

In this subsection, the contrast in opaque-oriented FVC, which is no more than 1/6, is proven. To satisfy the security constraint, the constant-ratios $(a_0^{\mathrm{U}} : a_1^{\mathrm{U}} : a_2^{\mathrm{U}} : a_3^{\mathrm{U}})$ and $(a_0^{\mathrm{L}} : a_1^{\mathrm{L}} : a_2^{\mathrm{L}} : a_3^{\mathrm{L}})$ in basis matrices $C[s_1(i, j), s_1(i, wh-1-j), s_2(i, j), s_2(i, wh-1-j)]$ must meet Eq. (2.2-2.3). Moreover, in the first and second rows of each basis matrix, the occurrence of $[0\ 0]^{\mathrm{T}}$, $[0\ 1]^{\mathrm{T}}$, $[1\ 0]^{\mathrm{T}}$, $[1\ 1]^{\mathrm{T}}$ must keep the constant ratio $(a_0^{\mathrm{U}} : a_1^{\mathrm{U}} : a_2^{\mathrm{U}} : a_3^{\mathrm{U}})$; and in the third and the fourth rows of each basis matrix, the occurrence of $[0\ 0]^{\mathrm{T}}$, $[0\ 1]^{\mathrm{T}}$, $[1\ 0]^{\mathrm{T}}$, $[1\ 1]^{\mathrm{T}}$ must keep the constant ratio $(a_0^{\mathrm{L}} : a_1^{\mathrm{L}} : a_2^{\mathrm{L}} : a_3^{\mathrm{L}})$. For each basis matrix, $c_{u,v} \geq 0$, where u =0,1,2,3 and v=0,1,2,3, is defined as the percentage of column $\left[\lfloor u/2 \rfloor \quad u \bmod 2 \quad \lfloor v/2 \rfloor \quad v \bmod 2\right]^{\mathrm{T}}$ which appears in columns of the basis matrix.

By the security constraint, we know

$$a_u^{\mathrm{U}} = c_{u,0} + c_{u,1} + c_{u,2} + c_{u,3} \quad \text{for } u = 0,1,2,3; \tag{2.5}$$

$$a_v^{\mathrm{L}} = c_{0,v} + c_{1,v} + c_{2,v} + c_{3,v} \quad \text{for } v = 0,1,2,3. \tag{2.6}$$

By the definition of *luminance transmission*, the four stacking results $s'_1(i, j)$, $s'_1(i, wh-1-j)$, $s'_2(i, j)$, $s'_2(i, wh-1-j)$ are represented by stacking two specific rows in basis matrix, which consists of 16 possible columns $\left[\lfloor u/2 \rfloor \quad u \bmod 2 \quad \lfloor v/2 \rfloor \quad v \bmod 2\right]^T$, where $u, v \in \{0,1,2,3\}$. Therefore, the *luminance transmission* of each stacking result $s'_1(i, j)$, $s'_1(i, wh-1-j)$, $s'_2(i, j)$, $s'_2(i, wh-1-j)$ can be represented by the sum of a subset $\{c_{u,v}\}$ which satisfies the result of stacking two specific rows defined in Definition 2.2.

1. The *luminance transmission* of stacking result $s_1(i, j)$ is

$$\sum_{u=0}^{3}\sum_{v=0}^{3} c_{u,v} \times \left(\overline{\lfloor u/2 \rfloor \otimes \lfloor v/2 \rfloor}\right) = c_{0,0} + c_{0,1} + c_{1,0} + c_{1,1}. \tag{2.7}$$

2. The *luminance transmission* of stacking result $s_1(i, wh-1-j)$ is

$$\sum_{u=0}^{3}\sum_{v=0}^{3} c_{u,v} \times \left(\overline{\lfloor u \bmod 2 \rfloor \otimes \lfloor v \bmod 2 \rfloor}\right) = c_{0,0} + c_{0,2} + c_{2,0} + c_{2,2}. \tag{2.8}$$

3. The *luminance transmission* of stacking result $s_2(i, j)$ is

$$\sum_{u=0}^{3}\sum_{v=0}^{3} c_{u,v} \times \left(\overline{\lfloor u \bmod 2 \rfloor \otimes \lfloor v/2 \rfloor}\right) = c_{0,0} + c_{0,1} + c_{2,0} + c_{2,1}. \tag{2.9}$$

4. The *luminance transmission* of stacking result $s_2(i, wh-1-j)$ is

$$\sum_{u=0}^{3}\sum_{v=0}^{3} c_{u,v} \times \overline{(\lfloor u/2 \rfloor \otimes \lfloor v \bmod 2 \rfloor)} = c_{0,0} + c_{0,2} + c_{1,0} + c_{1,2} \,. \tag{2.10}$$

where $\bar{\bullet}$ is the complement operator. The contrast α satisfies the Eq. (2.4). Notably, the *luminance transmission* of representing B is 0 and representing W is the contrast α by the definition of opaque-oriented FVC. Therefore, the complement operator is used in Eq. (2.7-2.10) for opposite definition between $B(1)/W(0)$ pixels and *luminance transmission.* Some basis matrices are considered below to gain the upper bound of contrast α.

I. Consider the basis matrix C_{BBBB}. By Eq. (2.7-2.10), the *luminance transmission* of the four secret pixels $s_1(i, j)$, $s_1(i, wh-1-j)$, $s_2(i, j)$, $s_2(i, wh-1-j)$ are

$$c_{0,0} + c_{0,1} + c_{1,0} + c_{1,1} = 0;$$

$$c_{0,0} + c_{0,2} + c_{2,0} + c_{2,2} = 0;$$

$$c_{0,0} + c_{0,1} + c_{2,0} + c_{2,1} = 0;$$

$$c_{0,0} + c_{0,2} + c_{1,0} + c_{1,2} = 0.$$

Due to $c_{u,v} \geq 0$, for all u, v. Therefore, $c_{0,0}=c_{0,1}=c_{0,2}=c_{1,0}=c_{1,1}=c_{1,2}=c_{2,0}=c_{2,1}=c_{2,2}=0$. By Eq. (2.5),

$$a_0^{\mathrm{U}} = c_{0,0} + c_{0,1} + c_{0,2} + c_{0,3} = c_{0,3};$$

$$a_1^{\mathrm{U}} = c_{1,0} + c_{1,1} + c_{1,2} + c_{1,3} = c_{1,3};$$

$$a_2^{\mathrm{U}} = c_{2,0} + c_{2,1} + c_{2,2} + c_{2,3} = c_{2,3}.$$

By Eq. (2.6),

$$a_0^{\mathrm{L}} = c_{0,0} + c_{1,0} + c_{2,0} + c_{3,0} = c_{3,0};$$

$$a_1^{\mathrm{L}} = c_{0,1} + c_{1,1} + c_{2,1} + c_{3,1} = c_{3,1};$$

$$a_2^{\mathrm{L}} = c_{0,2} + c_{1,2} + c_{2,2} + c_{3,2} = c_{3,2}.$$

By Eq. (2.5), $a_3^{\mathrm{U}} = c_{3,0} + c_{3,1} + c_{3,2} + c_{3,3} \geq c_{3,0} + c_{3,1} + c_{3,2} = a_0^{\mathrm{L}} + a_1^{\mathrm{L}} + a_2^{\mathrm{L}}$. Therefore,

$$\begin{aligned} & a_3^{\mathrm{U}} \geq a_0^{\mathrm{L}} + a_1^{\mathrm{L}} + a_2^{\mathrm{L}} \\ \Rightarrow\ & 1 - a_0^{\mathrm{U}} - a_1^{\mathrm{U}} - a_2^{\mathrm{U}} \geq a_0^{\mathrm{L}} + a_1^{\mathrm{L}} + a_2^{\mathrm{L}} \\ \Rightarrow\ & (a_0^{\mathrm{U}} + a_1^{\mathrm{U}} + a_2^{\mathrm{U}}) + (a_0^{\mathrm{L}} + a_1^{\mathrm{L}} + a_2^{\mathrm{L}}) \leq 1 \\ \Rightarrow\ & (1 - a_3^{\mathrm{U}}) + (1 - a_3^{\mathrm{L}}) \leq 1 \\ \Rightarrow\ & a_3^{\mathrm{U}} + a_3^{\mathrm{L}} \geq 1. \end{aligned} \tag{2.11}$$

II. Consider the basis matrix C_{BWBB}. By Eq. (2.7-2.10), the *luminance transmission* of the four secret pixels $s_1(i, j)$, $s_1(i, wh-1-j)$, $s_2(i, j)$, $s_2(i, wh-1-j)$ are

$$c_{0,0}+c_{0,1}+c_{1,0}+c_{1,1}=0;$$

$$c_{0,0}+c_{0,2}+c_{2,0}+c_{2,2}=\alpha;$$

$$c_{0,0}+c_{0,1}+c_{2,0}+c_{2,1}=0;$$

$$c_{0,0}+c_{0,2}+c_{1,0}+c_{1,2}=0.$$

Therefore, $c_{0,0}$=$c_{0,1}$=$c_{0,2}$=$c_{1,0}$=$c_{1,1}$=$c_{1,2}$=$c_{2,0}$=$c_{2,1}$=0, and $c_{2,2}$=α. By Eq. (2.5) and Eq. (2.6), $a_2^{\mathrm{U}}=c_{2,0}+c_{2,1}+c_{2,2}+c_{2,3}\geq c_{2,2}$, and $a_2^{\mathrm{L}}=c_{0,2}+c_{1,2}+c_{2,2}+c_{3,2}\geq c_{2,2}$, so

$$\alpha\leq a_2^{\mathrm{U}} \text{ and } \alpha\leq a_2^{\mathrm{L}}. \tag{2.12}$$

III. Consider the basis matrix C_{WBBW} and C_{WBWB}. When the basis matrices is C_{WBBW}, by Eq. (2.7-2.10), the *luminance transmission* of the four secret pixels $s_1(i, j)$, $s_1(i, wh-1-j)$, $s_2(i, j)$, $s_2(i, wh-1-j)$ are

$$c_{0,0}+c_{0,1}+c_{1,0}+c_{1,1}=\alpha;$$

$$c_{0,0}+c_{0,2}+c_{2,0}+c_{2,2}=0;$$

$$c_{0,0}+c_{0,1}+c_{2,0}+c_{2,1}=0;$$

$$c_{0,0}+c_{0,2}+c_{1,0}+c_{1,2}=\alpha.$$

Therefore, $c_{0,0}$=$c_{0,1}$=$c_{0,2}$=$c_{2,0}$=$c_{2,1}$= $c_{2,2}$=0, and

$$\alpha=c_{1,0}+c_{1,1}=c_{1,0}+c_{1,2}$$
$$\Rightarrow\alpha=(c_{1,0}+c_{1,1}+c_{1,0}+c_{1,2})/2=[(c_{1,0}+c_{1,1}+c_{1,2})+c_{1,0}]/2$$

Because, by Eq. (2.5),

$$a_1^{\mathrm{U}}=c_{1,0}+c_{1,1}+c_{1,2}+c_{1,3}\geq c_{1,0}+c_{1,1}+c_{1,2},$$

and by Eq. (2.6),

$$a_0^{\mathrm{L}}=c_{0,0}+c_{1,0}+c_{2,0}+c_{3,0}\geq c_{1,0},$$

so the contrast

$$\alpha=[(c_{1,0}+c_{1,1}+c_{1,2})+c_{1,0}]/2$$
$$\leq(a_1^{\mathrm{U}}+a_0^{\mathrm{L}})/2. \tag{2.13}$$

When stacking result is C_{WBWB}, by Eq. (2.6), the *average luminance transmission* of the four secret pixels $s_1(i, j)$, $s_1(i, wh-1-j)$, $s_2(i, j)$, $s_2(i, wh-1-j)$ are

$$c_{0,0}+c_{0,1}+c_{1,0}+c_{1,1}=\alpha;$$

$$c_{0,0}+c_{0,2}+c_{2,0}+c_{2,2}=0;$$

$$c_{0,0}+c_{0,1}+c_{2,0}+c_{2,1}=\alpha;$$

$$c_{0,0} + c_{0,2} + c_{1,0} + c_{1,2} = 0.$$

Therefore, $c_{0,0}$=$c_{0,2}$=$c_{1,0}$=$c_{1,2}$=$c_{2,0}$= $c_{2,2}$=0, and

$$\alpha = c_{0,1} + c_{1,1} = c_{0,1} + c_{2,1}$$
$$\Rightarrow \alpha = (c_{0,1} + c_{1,1} + c_{0,1} + c_{2,1})/2 = [(c_{0,1} + c_{1,1} + c_{2,1}) + c_{0,1}]/2$$

Because, by Eq. (2.5),

$a_0^{\mathrm{U}} = c_{0,0} + c_{0,1} + c_{0,2} + c_{0,3} \geq c_{0,1}$, and by Eq. (2.6),

$a_1^{\mathrm{L}} = c_{0,1} + c_{1,1} + c_{2,1} + c_{3,1} \geq c_{0,1}$, so the contrast

$$\alpha = [(c_{0,1} + c_{1,1} + c_{2,1}) + c_{0,1}]/2$$
$$\leq (a_1^{\mathrm{L}} + a_0^{\mathrm{U}})/2. \quad (2.14)$$

By Eq. (2.13) and (2.14), $\alpha \leq (a_1^{\mathrm{U}} + a_0^{\mathrm{L}})/2$, and $\alpha \leq (a_1^{\mathrm{L}} + a_0^{\mathrm{U}})/2$, we have $2\alpha \leq (a_1^{\mathrm{U}} + a_0^{\mathrm{L}})/2 + (a_1^{\mathrm{L}} + a_0^{\mathrm{U}})/2$, so

$$\alpha \leq (a_1^{\mathrm{U}} + a_0^{\mathrm{L}} + a_1^{\mathrm{L}} + a_0^{\mathrm{U}})/4. \quad (2.15)$$

$$\alpha \leq (a_1^{\mathrm{U}} + a_0^{\mathrm{L}} + a_1^{\mathrm{L}} + a_0^{\mathrm{U}})/4 \quad (\text{By Eq. (2.15)})$$
$$= [1 - (a_2^{\mathrm{U}} + a_3^{\mathrm{U}} + a_2^{\mathrm{L}} + a_3^{\mathrm{L}})]/4 \ (\text{By Eqs. (2.2) and (2.3)})$$
$$\leq [1 - (a_2^{\mathrm{U}} + a_2^{\mathrm{L}})]/4 \quad (a_3 \geq 0, b_3 \geq 0)$$
$$\leq (1 - 2\alpha)/4 \quad (\text{By Eq. (2.12)})$$
$$\Rightarrow \alpha \leq 1/6.$$

Therefore, the contrast of opaque-oriented FVC is no more than 1/6 if perfect security is required. The result also means that the encoding matrices shown in Table 2.1 are the optimal solution.

2.5.2 The proof of conditionally optimal contrast in non-opaque-oriented FVC

Non-opaque-oriented FVC also satisfies Eq. (2.2-2.10)**.** In the following, *w* is the *luminance transmission* of stacking result to represent white pixel *W*, *b* is the *luminance transmission* of stacking result to represent black pixel *B*, and α=*w*–*b* is the contrast. Some basis matrices are considered below to gain the upper bound of contrast α.

I. Consider the basis matrix C_{WWBB}. By Eq. (2.7-2.10), the *luminance transmission* of the four secret pixels $s_1(i, j)$, $s_1(i, wh-1-j)$, $s_2(i, j)$, $s_2(i, wh-1-j)$ are

$$c_{0,0} + c_{0,1} + c_{1,0} + c_{1,1} = w;$$

$$c_{0,0} + c_{0,2} + c_{2,0} + c_{2,2} = w;$$

$$c_{0,0}+c_{0,1}+c_{2,0}+c_{2,1}=b;$$

$$c_{0,0}+c_{0,2}+c_{1,0}+c_{1,2}=b.$$

By Eq. (2.5-2.6), $a_1^{\mathrm{L}}=c_{0,1}+c_{1,1}+c_{2,1}+c_{3,1}\geq c_{1,1}$, and $a_2^{\mathrm{L}}=c_{0,2}+c_{1,2}+c_{2,2}+c_{3,2}\geq c_{2,2}$. Therefore,

$$\begin{aligned}\alpha &= w-b=[w+w-b-b]/2\\ &=[(c_{0,0}+c_{0,1}+c_{1,0}+c_{1,1})+(c_{0,0}+c_{0,2}+c_{2,0}+c_{2,2})-\\ &(c_{0,0}+c_{0,1}+c_{2,0}+c_{2,1})-(c_{0,0}+c_{0,2}+c_{1,0}+c_{1,2})]/2\\ &=[(c_{1,1}+c_{2,2})-(c_{1,2}+c_{2,1})]/2\leq(c_{1,1}+c_{2,2})/2\leq(a_1^{\mathrm{L}}+a_2^{\mathrm{L}})/2.\end{aligned} \tag{2.16}$$

II. Consider the basis matrix C_{WWWW} and C_{BBBB}. For the basis matrix $C[W, W, W, W]$, by Eq. (2.7-2.10),

$$c_{0,0}+c_{0,1}+c_{1,0}+c_{1,1}=w;$$

$$c_{0,0}+c_{0,2}+c_{2,0}+c_{2,2}=w;$$

$$c_{0,0}+c_{0,1}+c_{2,0}+c_{2,1}=w;$$

$$c_{0,0}+c_{0,2}+c_{1,0}+c_{1,2}=w.$$

By Eq. (2.5), $a_0=c_{0,0}+c_{0,1}+c_{0,2}+c_{0,3}\geq c_{0,0}+c_{0,1}+c_{0,2}$, and $a_1=c_{1,0}+c_{1,1}+c_{1,2}+c_{1,3}\geq c_{1,0}+c_{1,1}+c_{1,2}$.

By Eq. (2.6), $b_0=c_{0,0}+c_{1,0}+c_{2,0}+c_{3,0}\geq c_{0,0}+c_{1,0}$. Therefore,

$$\begin{aligned}w&=[(c_{0,0}+c_{0,1}+c_{1,0}+c_{1,1})+(c_{0,0}+c_{0,2}+c_{1,0}+c_{1,2})]/2\\ &=[(c_{0,0}+c_{0,1}+c_{0,2})+(c_{1,0}+c_{1,1}+c_{1,2})+(c_{0,0}+c_{1,0})]/2\\ &\leq(a_0^{\mathrm{U}}+a_1^{\mathrm{U}}+a_0^{\mathrm{L}})/2\end{aligned} \tag{2.17}$$

For the basis matrix C_{BBBB}, by Eq. (2.7-2.10),

$$c_{0,0}+c_{0,1}+c_{1,0}+c_{1,1}=b;$$

$$c_{0,0}+c_{0,2}+c_{2,0}+c_{2,2}=b;$$

$$c_{0,0}+c_{0,1}+c_{2,0}+c_{2,1}=b;$$

$$c_{0,0}+c_{0,2}+c_{1,0}+c_{1,2}=b.$$

By Eq. (2.12), $a_3^{\mathrm{L}}=c_{0,3}+c_{1,3}+c_{2,3}+c_{3,3}\geq c_{0,3}+c_{1,3}$. Therefore,

$$
\begin{aligned}
b &= [(c_{0,0} + c_{0,1} + c_{1,0} + c_{1,1}) + (c_{0,0} + c_{0,2} + c_{1,0} + c_{1,2})]/2 \\
&\geq [(c_{0,0} + c_{0,1} + c_{0,2}) + (c_{1,0} + c_{1,1} + c_{1,2})]/2 \\
&= [(c_{0,0} + c_{0,1} + c_{0,2} + c_{0,3}) + (c_{1,0} + c_{1,1} + c_{1,2} + c_{1,3}) - (c_{0,3} + c_{1,3})]/2 \\
&\geq (a_0^{\mathrm{U}} + a_1^{\mathrm{U}} - a_3^{\mathrm{L}})/2
\end{aligned} \tag{2.18}
$$

By Eq. (2.4), Eq.(2.17) and Eq.(2.18),

$$
\begin{aligned}
\alpha &= w - b \\
&\leq (a_0^{\mathrm{U}} + a_1^{\mathrm{U}} + a_0^{\mathrm{L}})/2 - (a_0^{\mathrm{U}} + a_1^{\mathrm{U}} - a_3^{\mathrm{L}})/2 \\
&= (a_0^{\mathrm{L}} + a_3^{\mathrm{L}})/2
\end{aligned} \tag{2.19}
$$

By Eq. (2.16) and Eq.(2.19),

$$
\begin{aligned}
\alpha &\leq [(a_1^{\mathrm{L}} + a_2^{\mathrm{L}})/2 + (a_0^{\mathrm{L}} + a_3^{\mathrm{L}})/2]/2 \\
&= (a_0^{\mathrm{L}} + a_1^{\mathrm{L}} + a_2^{\mathrm{L}} + a_3^{\mathrm{L}})/4 \\
&= 1/4 \quad (\text{By Eq. (2.6)}).
\end{aligned}
$$

Therefore, if non-opaque-oriented FVC is used, then the conditionally-optimal contrast is 1/4. The result is better than the conditionally-optimal contrast value 1/6 of the opaque-oriented FVC. The encoding matrices shown in Table 2.2 are conditionally optimal, because a) Property 2.2 shows that, for these matrices, the contrast of stacking result is 1/4; and b) the proof given above indicates that: under the perfect security constraint, no basis matrices can yield a contrast larger than 1/4.

Chapter 3

Fast Weighted Secret Image Sharing

Thien and Lin [Computers and Graphics 26(5), 2002, pp. 765–770] proposed a threshold scheme to share a secret image among n shadows: any t of the n shadows can recover the secret, whereas t–1 or fewer shadows cannot. However, in real life, certain managers probably play key roles to run a company, and thus need special authority to recover the secret in managers' meeting. (Each manager's shadow should be more powerful than an ordinary employee's shadow.) In Thien and Lin's scheme, if a company has less than t managers, then manager's meeting cannot recover the secret, unless some managers were given multiple shadows in advance. But this compromise causes managers inconvenience because too many shadows to be kept daily and carried to the meeting. To solve this dilemma, a weighted sharing method is proposed: each of our shadow has a weight. Secret is recovered if and only if the total weights (rather than the number) of received shadows is at least t. To accelerate sharing speed, properties of GF(2^k) are utilized. Time-saving is shown. Besides, the method is also a more general approach to polynomial-based sharing. Moreover, for convenience, each person keeps only one shadow and only one shadow-index.

The rest of the chapter is organized as follows. Sec. 3.1 reviews the related works. Sec. 3.2 describes the details of the proposed fast weighted secret image sharing method. Sec. 3.3 shows the experimental results, comparisons and security analysis. Finally, Sec. 3.4 draws the conclusions.

Notations of this chapter:

t	The threshold of secret sharing.
n	The number of shadows of secret sharing.
$f(x)$	The sharing polynomial where $f(x)=a_0+a_1+\ldots+a_{t-1}x^{t-1} \pmod p$.
p	A prime number.
w_i	The weight of the shadow h_i.
h_i	The i^{th} generated shadow.
$g_i^{w_i}(x)$	$g_i^{w_i}(x) = f(x) \pmod{(x-i)^{w_i}}$ is the i^{th} shadow.
GF(p^k)	Galois field which contains p^k elements.

$\oplus$	XOR operator

3.1 Related works

Sec. 3.1.1 introduces the Thien and Lin's sharing method, and Section 3.1.2 introduces the Galois field which will be utilized in this chapter.

3.1.1 Thien and Lin's secret image sharing method[7]

In the sharing phase of Thien and Lin's (t, n) threshold method, for each non-overlapping t pixels of the secret image, the corresponding polynomial is defined as

$$f(x) = a_0 + a_1 \times x + \cdots + a_{t-1} \times x^{t-1} \pmod{p} \tag{3.1}$$

where a_0, a_1,..., a_{t-1} are the gray values of each t pixels, and p is a prime number. Then

$$f(1), f(2),\ldots, f(n) \tag{3.2}$$

are evaluated and assigned to the n shadows sequentially. After processing all pixels in the secret image, the n shadows are thus generated. Since each t pixels in secret image only contributes one pixel to each generated shadow, the size of which is $1/t$ of the secret image.

As for the revealing phase, when any t of the n shadows are received, the first not-yet-used pixel from each of the t shadows is taken, and these t pixels can be used to solve the coefficients a_0, a_1, ..., a_{t-1} in Eq. (3.1) by using Lagrange's polynomial. After sequentially processing all pixels of the t shadows, the secret image can be obtained.

3.1.2 Galois field

Galois Field GF(p^k) is a finite filed that contains p^k elements, where p is a prime number, and k is a positive integer. (Thien and Lin used p=251 and k=1, but we use p = 2 and k=8 in our method.) A finite field also equips with two operators: addition(+) and multiplication(•). Both operators must satisfy the commutative, associative, and distributive laws. The manipulation of addition and multiplication under GF(2^k) are introduced below. Before doing GF(2^k) arithmetic, an k-degree binary-coefficient polynomial $m(X)$, called primitive polynomial, have to be defined first. Primitive means that $m(X)$ has a root α, and {0, 1, α, α^2, ..., α^{2^k-2}} is the all elements in GF(2^k) (the multiplication operator is defined below). About more details of the primitive polynomial $m(X)$ and the root α, please see S. Lin and D. J. Costello[70]. Here, we will use k=8 and $m(X)$=1+X^2+X^3+X^4+X^8 in our experiments.

Let A^0 and A^1 be any two elements in GF(2^k). Then define the addition operator as

$$A^0+A^1= A^0 \oplus A^1,$$

where $\oplus$ is the XOR operator. The multiplication operator $\times$ is somewhat more complicated. Before doing multiplication, convert the two elements A^0 and A^1 to two binary polynomials

$$A^0 = (a_{k-1}^0 ... a_1^0 a_0^0)_2 \rightarrow a_0^0 + a_1^0 x + ... + a_{k-1}^0 X^{k-1},$$

$$A^1 = (a_{k-1}^1 ... a_1^1 a_0^1)_2 \rightarrow a_0^1 + a_1^1 x + ... + a_{k-1}^1 X^{k-1}.$$

Then do the following polynomial multiplication and modulus operations

$$(a_0^0 + a_1^0 x + ... + a_{k-1}^0 X^{k-1})(a_0^1 + a_1^1 x + ... + a_{k-1}^1 X^{k-1}) (\mathrm{mod}\, m(X))$$
$$\Rightarrow (a_0^0 \wedge a_0^1) + (a_0^0 \wedge a_1^1 \oplus a_1^0 \wedge a_0^1) X + ... + (a_{k-1}^0 \wedge a_{k-1}^1) X^{2k-2} (\mathrm{mod}\, m(X))$$
$$\Rightarrow a_0^2 + a_1^2 x + ... + a_{k-1}^2 X^{k-1},$$

where $\wedge$ is the AND operator. Finally, the result for $A^0 \times A^1$ can be obtained by

$$A^0 \times A^1 = A^2 = (a_{k-1}^2 a_1^2 ... a_0^2)_2.$$

Remark: In general, there exist other definitions for addition and multiplication operators. (The details about GF(2^k) can see be found in [70].) But we will use the above definition for addition and multiplication throughout the chapter.

3.2 The proposed method

This section has three subsections: 3.2.1 is for weighted secret image sharing; 3.2.2 is for weighted secret image revealing; 3.2.3 shows the improved weighted secret image sharing algorithm based on GF(2^k).

3.2.1 The weighted secret image sharing phase

According to the Chinese Remainder Theorem for polynomials, when we divide

$$f(x)=a_0+a_1x+\ldots+a_{t-1}x^{t-1}$$

by a factor $(x-i)$, the remainder is $h(i)$. In symbols,

$$h(i)= h(x) \bmod (x-i).$$

Now, when we apply mod p on both sides, we have

$$f(i) = h(i) \bmod p = [h(x) \bmod (x-i)] \pmod p$$

where $f(i) = h(i) \pmod p$ is due to the equation $f(x) = a_0+a_1x+\ldots+a_{t-1}x^{t-1} \pmod p = h(x) \pmod p$ defined in Eq. (3.2). Therefore, in Galois Field GF(p), i.e. in the field of mod p, we may say that $f(i)$ and $[h(x) \bmod (x-i)]$ are equal. In symbols,

$$f(i) = h(x) \bmod (x-i) = a_0+a_1x+\ldots+a_{t-1}x^{t-1} \pmod{(x-i)} \qquad (3.3)$$

in Galois Field GF(p). That is to say, if we divide that polynomial $a_0+a_1x+\ldots+a_{t-1}x^{t-1}$ by $(x-i)$, then the remainder is a number. If we divide this number by p further, then we obtain $f(i)$. In this chapter, to define our own formula of the weighted secret image sharing with the (t, n) threshold scheme, we extend Eq. (3.3) as

$$g_i^{w_i}(x) = a_0+a_1x+\ldots+a_{t-1}x^{t-1} \pmod{(x-i)^{w_i}}, \tag{3.4}$$

where w_i is the shadow weight and i=1, 2, ..., n. Also, rather than explaining Eq. (3.4) in the GF(251) that Thien and Lin used, we explain Eq. (3.4) in the Galois Field GF(2^k). As stated in Sec. 3.1.2, k is a positive integer, and we will use GF(2^8) in our experiments.

Before sharing each non-overlapping t pixels of the secret image using weighted secret image sharing with (t, n) threshold scheme, the secret image is encrypted first. Next,

$$g_1^{w_1}(x), g_2^{w_2}(x),..., g_n^{w_n}(x) \tag{3.5}$$

are computed using Eq. (3.4). Then, the w_i coefficients of the polynomial $g_i^{w_i}(x)$ in order of decreasing power of x are sequentially assigned to the corresponding shadow h_i. After processing all pixels in the secret image, the n shadows $\{(h_1, w_1), (h_2, w_2), ..., (h_n, w_n)\}$ are generated. Since t pixels in secret image contribute w_i pixels to the generated shadow h_i, the size of which is w_i/t of the secret image.

In the proposed (t, n) threshold weighted secret image sharing scheme, the two values index i and weight w_i of the generated shadow h_i are needed for revealing the secret image where $1 \le i \le n$. Like Thien and Lin's method[7], the value i can be attached to the head of the shadow h_i. As for the value w_i, it can be either simply attached to the head of the shadow h_i or calculated by the size of the shadow. Let the size of secret image be $|S|$ and the size of shadow be $|h_i|$. Then, the weight w_i can be calculated by the formula

$$w_i = \frac{|h_i|}{\lceil |S|/t \rceil} \tag{3.6}$$

3.2.2 The weighted secret image revealing phase

If someone gets any t' of the n shadows and the sum of the weights of the m shadows is greater than or equal to the threshold t, then the secret image can be recovered. Without loss of generality, let these t' shadows be $\{(h_{k[1]}, w_{k[1]}), (h_{k[2]}, w_{k[2]}), ..., (h_{k[t']}, w_{k[t']})\}$ and $\sum_{j=1}^{t'} w_{k[j]} \ge t$. Then for each shadow $h_{k[j]}$, the first $w_{k[j]}$ not-yet-used pixels are sequentially

taken and then assigned to the coefficients of polynomial $g_{k[j]}^{w_{k[j]}}(x)$ in order of decreasing power of *x*. After obtaining $g_{k[j]}^{w_{k[j]}}(x)$, we have the following equation

$$g_{k[j]}^{w_{k[j]}}(x) = f(x) \bmod (x - k[j])^{w_{k[j]}} \tag{3.7}$$

where *j*=1, 2, ..., *t′*. Because the *t′* divisors $(x-k[1])^{w_{k[1]}}$, $(x-k[2])^{w_{k[2]}}$, ..., $(x-k[m])^{w_{k[m]}}$ in Eq. (3.7) are pair wised relatively prime, as stated in Sec. 1.4 of [71], $f(x)$ can be solved using extended Lagrange polynomial as

$$f(x) = \sum_{j=1}^{t'} \left[g_{k[j]}^{w_{k[j]}}(x) u_j(x) \quad \left(\bmod (x-k[j])^{w_{k[j]}} \right) \times \prod_{\substack{l=1 \\ l \neq j}}^{t'} (x-k[l])^{w_{k[l]}} \right],$$

$$\text{where } u_j(x) = \left[\prod_{\substack{l=1 \\ l \neq j}}^{t'} (x-k[l])^{w_{k[l]}} \right]^{-1} \quad \left(\bmod (x-k[j])^{w_{k[j]}} \right). \tag{3.8}$$

In addition, according to Chinese Remainder Theorem for polynomials, the decoded *f*(*x*) is a unique polynomial with degree is less than $\sum_{j=1}^{t'} w_{k[j]}$. Because $\sum_{j=1}^{t'} w_{k[j]} \geq t$, the polynomial *f*(*x*) is identical to the original polynomial, where the degree of *f*(*x*) is less than *t*. In other words, the *t* coefficients $a_0, a_1, ..., a_{t-1}$ in Eq. (3.4) can be obtained.

After sequentially processing all pixels of the *t′* shadows, the encrypted secret image can be reconstructed. The encrypted secret image is then decrypted to obtain the secret image.

3.2.3 The fast weighted secret image sharing algorithm

The computing time of Eq. (3.5) is improved by using the properties of GF(2^k). The utilized property is that the additive inverse of an element under GF(2^k) is the element. In other words,

$$x = -x. \tag{3.9}$$

By Eq. (3.9), the following equation is derived:

$$(x-u)^2 = (x+u)^2 = x^2 + xu + xu + u^2 = x^2 + xu - xu + u^2 = x^2 + u^2. \tag{3.10}$$

Then the Eq. (3.10) can be extended as

$$(x-u)^{2^q} = (x+u)^{2^q} = (x^2+u^2)^{2^{q-1}} = (x^4+u^4)^{2^{q-2}} = ... = x^{2^q} + u^{2^q}. \tag{3.11}$$

where *q* is a positive integer, and *u* is an element in GF(2^k). A generalized form is addressed

in Sec. 2.3 of [70]. Let $\sum_{j=0}^{2^q-1} a_j x^j$ and $(x+u)^{2^{q-1}}$ be two polynomials. Then $\sum_{j=0}^{2^q-1} a_j x^j$ is divided by $(x+u)^{2^{q-1}}$ under GF(2^k) to get the quotient $a_{2^q-1}x^{2^{q-1}-1} + a_{2^q-2}x^{2^{q-1}-2} + \ldots + a_{2^{q-1}}$ and the remainder $\left(a_{2^{q-1}-1} + a_{2^q-1}u^{2^{q-1}}\right)x^{2^{q-1}-1} + \left(a_{2^{q-1}-2} + a_{2^q-2}u^{2^{q-1}}\right)x^{2^{q-1}-2} + \ldots + \left(a_0 + a_{2^{q-1}}u^{2^{q-1}}\right)$. By Eq. (3.11), we have $(x+u)^{2^{q-1}} = x^{2^{q-1}} + u^{2^{q-1}}$. Therefore, $\sum_{j=0}^{2^q-1} a_j x^j$ can be expressed as

$$\begin{aligned}\sum_{j=0}^{2^q-1} a_j x^j = &\left(a_{2^q-1}x^{2^{q-1}-1} + a_{2^q-2}x^{2^{q-1}-2} + \ldots + a_{2^{q-1}}\right)(x+u)^{2^{q-1}} + \\ &\left(a_{2^{q-1}-1} + a_{2^q-1}u^{2^{q-1}}\right)x^{2^{q-1}-1} + \left(a_{2^{q-1}-2} + a_{2^q-2}u^{2^{q-1}}\right)x^{2^{q-1}-2} + \ldots + \left(a_0 + a_{2^{q-1}}u^{2^{q-1}}\right)\end{aligned} \quad . \tag{3.12}$$

However, if uses Eq. (3.12) for sharing directly, the weight w_i is restricted as power of two (2^{q-1}). In order to achieving a generalized version, a recursive algorithm is proposed below. Let $\hat{i} \leftarrow i$, $\hat{w}_i \leftarrow w_i$, $\hat{t} \leftarrow 2^{\lceil \log_2 t \rceil}$, and $\hat{f}(x) \leftarrow f(x)$. Now, $\hat{f}(x) \bmod (x+\hat{i})^{\hat{w}_i}$ is solved using the following recursive algorithm $A(\hat{f}(x), \hat{i}, \hat{w}_i, \hat{t})$.

Algorithm 3.1. Fast weighted secret image sharing algorithm $A(\hat{f}(x), \hat{i}, \hat{w}_i, \hat{t})$.

Input: a polynomial $\hat{f}(x)$, three positive integers $\hat{i}$ (index), $\hat{w}_i$ (weight), and $\hat{t}$ (a value in {1, 2, 2^2, 2^3,…}, and $\hat{t}$ is the number of polynomial coefficients for $\hat{f}(x)$).

Output: The shadow values with index $\hat{i}$ and weight $\hat{w}_i$ (The coefficients of $\hat{f}(x) \bmod (x+\hat{i})^{\hat{w}_i}$).

1. According to Eq. (3.12), rewrite $\hat{f}(x)$ as $\hat{f}(x) = \hat{Q}(x)(x+\hat{i})^{\hat{t}/2} + \hat{R}(x)$, where $\hat{Q}(x)$ and $\hat{R}(x)$ are, respectively, the quotient and the remainder on dividing $\hat{f}(x)$ by $(x+\hat{i})^{\hat{t}/2} = x^{\hat{t}/2} + \hat{i}^{\hat{t}/2}$ over $GF(2^{\hat{t}})$.

2. Compare $\hat{w}_i$ with $\hat{t}/2$:

Case 1: If $\hat{w}_i = \hat{t}/2$, then

$$\begin{aligned}&\hat{f}(x) \bmod (x+\hat{i})^{\hat{t}/2} = \hat{Q}(x)(x+\hat{i})^{\hat{t}/2} + \hat{R}(x) \\ &= \hat{Q}(x)(x+\hat{i})^{\hat{t}/2} + \hat{R}(x) = \hat{R}(x) \quad \left(\bmod (x+\hat{i})^{\hat{t}/2}\right).\end{aligned}$$

Therefore, return $\hat{R}(x)$.

Case 2: If $\hat{w}_i < \hat{t}/2$, then

$\hat{f}(x) \quad (\mathrm{mod}(x+\hat{i})^{\hat{w}_i}) = \hat{Q}(x)(x+\hat{i})^{\hat{t}/2} + \hat{R}(x) = \hat{R}(x) \quad (\mathrm{mod}(x+\hat{i})^{\hat{w}_i})$.

Then, $\hat{R}(x) \bmod (x+\hat{i})^{\hat{w}_i}$ is recursively computed by $A(\hat{R}(x), \hat{i}, \hat{w}_i, \hat{t}/2)$.

Finally, return $\hat{R}(x) \bmod (x+\hat{i})^{\hat{w}_i}$.

Case 3: If $\hat{w}_i > \hat{t}/2$, then

$$\begin{aligned}
&\hat{f}(x) \bmod (x+\hat{i})^{\hat{w}_i} \\
&= \hat{Q}(x)(x+\hat{i})^{\hat{t}/2} + \hat{R}(x) \quad (\mathrm{mod}(x+\hat{i})^{\hat{w}_i}) \\
&= (\hat{Q}(x) \bmod (x+\hat{i})^{\hat{w}_i - \hat{t}/2})(x+\hat{i})^{\hat{t}/2} + \hat{R}(x) \\
&= (\hat{Q}(x) \bmod (x+\hat{i})^{\hat{w}_i - \hat{t}/2})(x^{\hat{t}/2} + \hat{i}^{\hat{t}/2}) + \hat{R}(x) \quad (\because \text{By Eq. (3.11)})
\end{aligned}$$

Then, $\hat{Q}(x) \bmod (x+\hat{i})^{\hat{w}_i - \hat{t}/2}$ is recursively computed by $A(\hat{Q}(x), \hat{i}, \hat{w}_i - \hat{t}/2, \hat{t}/2)$. Finally, return $(\hat{Q}(x) \bmod (x+\hat{i})^{\hat{w}_i - \hat{t}/2})(x^{\hat{t}/2} + \hat{i}^{\hat{t}/2}) + \hat{R}(x)$.

Notably, the above algorithm can be abbreviated as a recursive function. Let $\hat{i} = i, \hat{w}_i = w_i, \hat{t} = 2^{\lceil \log t \rceil}$, and $\hat{f}(x) = \sum_{i=t}^{\hat{t}} 0x^i + f(x)$. Then,

$$\begin{aligned}
&A(\hat{f}(x), \hat{i}, \hat{w}_i, \hat{t}) \\
&= A(\hat{Q}(x)(x+\hat{i})^{\hat{t}/2} + \hat{R}(x), \hat{i}, \hat{w}_i, \hat{t}) \\
&= \begin{cases}
\text{Case 1}: \hat{R}(x), & \text{if } \hat{w}_i = \hat{t}/2 \\
\text{Case 2}: A(\hat{R}(x), \hat{i}, \hat{w}_i, \hat{t}/2), & \text{if } \hat{w}_i < \hat{t}/2 \\
\text{Case 3}: A(\hat{Q}(x), \hat{i}, \hat{w}_i - \hat{t}/2, \hat{t}/2)(x^{\hat{t}/2} + \hat{i}^{\hat{t}/2}) + \hat{R}(x), & \text{if } \hat{w}_i > \hat{t}/2
\end{cases}
\end{aligned}$$

Now, for the recursive function above, an example is given below.

Example 3.1. An demonstration of fast weighted secret image sharing:

Input of the demonstration:

i) A polynomial $f(x)=2x^5+5x^4+2x^3+6x^2+3x+1$ whose coefficients are all in GF(2^3=8), i.e. all in the range {0,1,2,3,4,5,6,7}.

ii) A shadow index i=1, a shadow weight w_i=5, and a threshold t=6.

Demonstration purpose:

Show how to compute the corresponding shadow value $g_1^5(x) = \hat{f}(x) \bmod (x+\hat{i})^{\hat{w}_i} = \left(0x^7 + 0x^6 + 2x^5 + 5x^4 + 2x^3 + 6x^2 + 3x + 1\right) \bmod (x+1)^5$ where $\hat{i} = i = 1$, $\hat{w}_i = w_i = 5$, and $\hat{f}(x) = \sum_{i=t}^{\hat{t}} 0x^i + f(x) = 0x^7+0x^6+2x^5+5x^4+2x^3+6x^2+3x+1$ is the whole-power-of-two version of f (by adding the missing zero coefficients to f so that all $\hat{t} = 2^{\lceil \log_2 t \rceil} = 2^{\lceil \log_2 6 \rceil} = 8$ coefficients appear.)

Demonstration detail:

According to the recursive function of our sharing algorithm, we have

$A(\hat{f}(x), \hat{i}, \hat{w}_i, \hat{t})$

$= A\left(0x^7 + 0x^6 + 2x^5 + 5x^4 + 2x^3 + 6x^2 + 3x + 1, 1, 5, 8\right)$

$= A\left((0x^3 + 0x^2 + 2x + 5)(x+1)^4 + (2x^3 + 6x^2 + 1x + 4), 1, 5, 8\right)$ ($\because$ Eq. (3.12))

$= A\left(0x^3 + 0x^2 + 2x + 5, 1, 1, 4\right)(x^4 + 1^4) + (2x^3 + 6x^2 + 1x + 4)$ ($\because \hat{w}_i = 5 > \hat{t}/2 = 4$, $\therefore$Case 3)

$= A\left((0x+0)(x+1)^2 + (2x+5), 1, 1, 4\right)(x^4 + 1^4) + (2x^3 + 6x^2 + 1x + 4)$ ($\because$ Eq. (3.12))

$= A\left(2x + 5, 1, 1, 2\right)(x^4 + 1^4) + (2x^3 + 6x^2 + 1x + 4)$ ($\because \hat{w}_i = 1 < \hat{t}/2 = 2$, $\therefore$Case 2)

$= A\left(2(x+1)^1 + 7, 1, 1, 2\right)(x^4 + 1^4) + (2x^3 + 6x^2 + 1x + 4)$ ($\because$ Eq. (3.12))

$= 7(x^4 + 1^4) + (2x^3 + 6x^2 + 1x + 4)$ ($\because \hat{w}_i = 1 = \hat{t}/2 = 1$, $\therefore$Case 1)

$= 7x^4 + 2x^3 + 6x^2 + 1x + 3$.

Therefore, $g_1^5(x) = \left(2x^5 + 5x^4 + 2x^3 + 6x^2 + 3x + 1\right) \bmod (x+1)^5$

$= A\left(0x^7 + 0x^6 + 2x^5 + 5x^4 + 2x^3 + 6x^2 + 3x + 1, 1, 5, 8\right) = 7x^4 + 2x^3 + 6x^2 + 1x + 3$.

3.3 Experimental results, comparisons, and security analysis

Sec. 3.3.1 shows the experimental results. Sec. 3.3.2 compares our method with Thien and Lin's method. Sec. 3.3.3 is the discussion about the security of our method.

3.3.1 Experimental results

The standard 512×512 gray-level image Lena is shown in Fig. 3.1, which is used as the secret image in the experiments. Fig. 3.2 shows the encrypted image of Fig. 3.1; the encryption uses exclusive-OR operation between a random sequence and the gray values of the secret image. GF(2^8) is used in the sharing scheme. Then, the proposed fast weighted

secret image sharing with (t=256, n=7) threshold scheme is used to share the encrypted secret image Fig. 3.2, and n=7 shadows are thus generated shown in Figs. 3.3(a-g) with shadow weight 160, 64, 24, 8,134, 12, 3, respectively. Fig. 3.4 is the image revealed by Figs. 3.3(a-d), and the revealed image is identical to Fig. 3.1.

Fig. 3.5 compares the execution time in the weighted secret image sharing phase using Thien and Lin's (t, n)=(256, w_i) threshold scheme[7] and our (t, n)=(256, 1) threshold scheme. Notably, the execution time of our sharing algorithm is 7±3 mille-seconds for each of these 255 sets of weights; whereas the execution time increases linearly as the weight value increases in Thien and Lin's direct and repeated application (using multiple shadows to simulate weighted feature).

Fig. 3.1. The 512×512 secret image Lena.

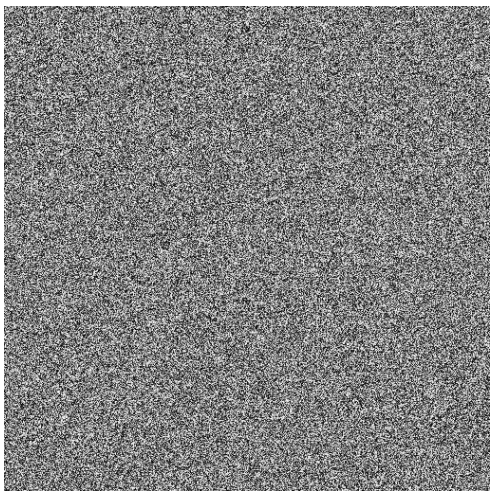
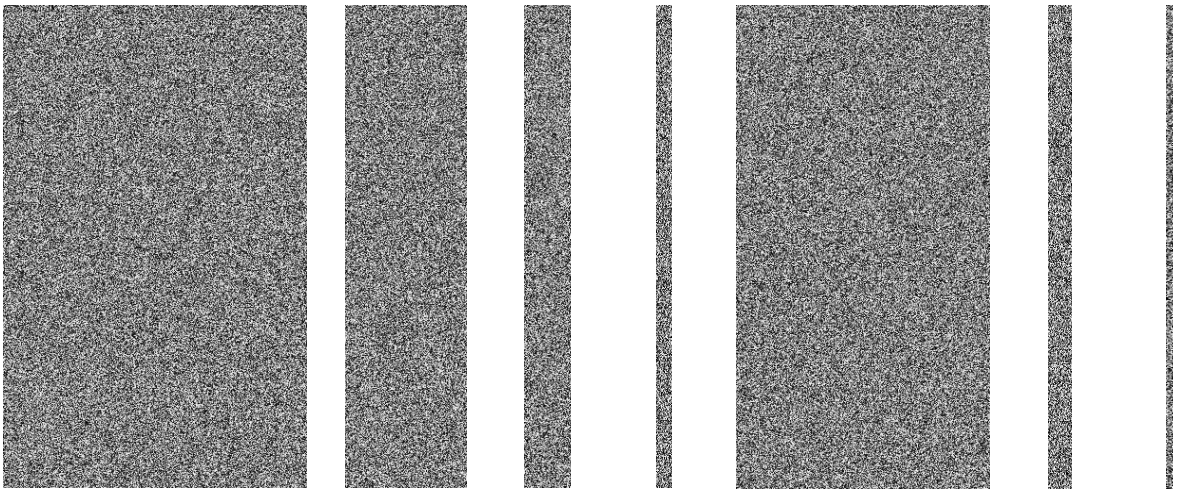


Fig. 3.2. The encrypted image of Lena.



(a) (b) (c) (d) (e) (f) (g)

Fig. 3.3. The (*t*, *n*)=(256, 7) secret sharing scheme in GF(2^8). Each of the 7 shadow weight is (a) 160; (b) 64; (c) 24; (d) 8; (e) 134; (f) 12; (g) 3.

Fig. 3.4. The image revealed from Figs. 3(a-d).

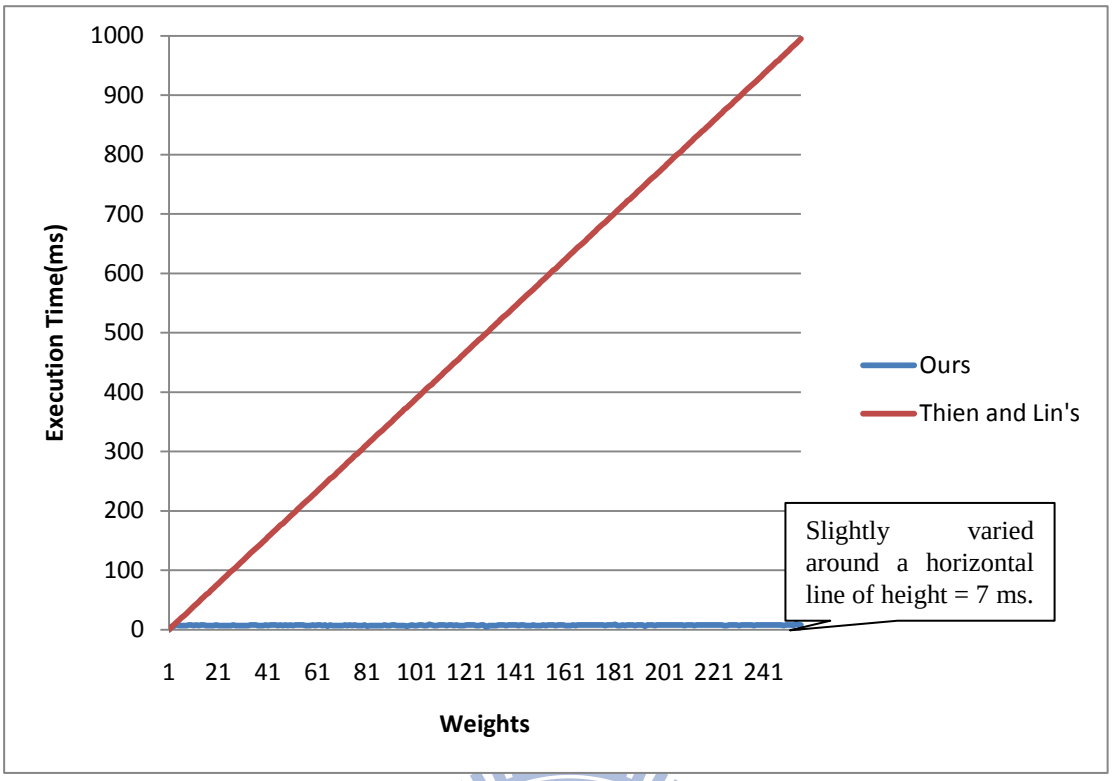


Fig. 3.5. The execution time in the weighted secret image sharing using Thien and Lin's (t=256, n= w_i) threshold scheme[3] and our (t=256, n=1) threshold scheme.

3.3.2 Comparisons with Thien and Lin's scheme[7]

Some advantages of our method are listed below (compared with Thien and Lin's method).

Time complexity:

The time complexity of the weighted secret image sharing using Thien and Lin's scheme[3] and our scheme is analyzed as follows. Let $|S|$ denotes the size of the secret image. For Thien and Lin's (t, n) threshold scheme[3], when sharing each non-overlapping t pixels of the secret image to $\sum_{i=1}^{n} w_i$ shadows, based on Shamir's seminal work, $f(1)$, $f(2)$, …, $f(\sum_{i=1}^{n} w_i)$ are computed using Eq. (3.1). Because there are t multiplications and $(t-1)$ additions in Eq. (3.1), the time complexity of sharing secret image with size $|S|$ to $\sum_{i=1}^{n} w_i$ shadows using Thien and Lin's scheme is $\theta(\sum_{i=1}^{n} w_i) \times \theta(t) \times \theta(|S|/t) = \theta(|S| \times \sum_{i=1}^{n} w_i)$. Since $\sum_{i=1}^{n} w_i < nt$, we have

$\theta(|S| \times \sum_{i=1}^{p} w_i) = O(|S| nt)$.

As for our scheme, when sharing each non-overlapping t pixels of the secret image to n shadows, $f(x) = a_0 + a_1x + \ldots + a_{t-1}x^{t-1}$ in Eq. (3.4) is expanded to $f(x) = a_0 + a_1x + \ldots + a_{t-1}x^{t-1} + 0x^t + \ldots + 0\, x^{2^{\lceil \log_2 t \rceil}}$ if the value of t is not power of two. Then, $g_1^{w_1}(x), g_2^{w_2}(x), \ldots, g_n^{w_n}(x)$ in Eq. (3.5) are computed using our fast weighted secret image sharing algorithm. Suppose the time complexity of computing each $g_i^{w_i}(x)$ in Eq. (3.5) is $T(\hat{t})$ where i=1, 2, …, n. Because the concept of the recursive function is applied in our algorithm, and there are $\hat{t}/2$ multiplications and $\hat{t}/2$ additions in the step 1 of Algorithm 1, the recurrence relation $T(\hat{t}) = T(\hat{t}/2) + \theta(\hat{t})$ can be derived. The recurrence relation is then solved by the substitution method to obtain $T(\hat{t}) = \theta(\hat{t})$. Because $\hat{t} \leftarrow 2^{\lceil \log_2 t \rceil}$, the value of $\hat{t}$ is at most two times of t. Therefore, we have $T(\hat{t}) = \theta(\hat{t}) = \theta(t)$. So, the time complexity of sharing secret image with size $|S|$ to n shadows using our scheme is $\theta(n) \times \theta(t) \times \theta(|S|/t) = \theta(|S|n)$.

A more general scheme for polynomial-based sharing:

In our weighted sharing scheme, according to the Chinese Remainder Theorem for polynomials, the n polynomials $x-1$, $x-2$, …, $x-n$ in Eq. (3.3) can be replaced by n other sharing polynomials such as x^2+x+1, x^2+x+2, …, x^2+x+n, as long as these n polynomials are pair-wise prime (i.e. no pair of polynomials has a non-trivial common factor). Notably, Thien and Lin' method is only a special case of this generalized scheme (i.e. the n shadows of Thien and Lin's are evaluated by $f(i) = f(x) \bmod (x-i)$, for i=1, 2, ..., n. In other words, only $\{x-1, x-2, \ldots, x-n\}$ were used by Thien and Lin; whereas we can use all sharing polynomials which are pair-wise prime).

Better performance when pixel values are larger than 250:

The computations in Thien and Lin's sharing process are in the field GF(251). All gray values 251~255 of the gray-level secret image have to be truncated to 250. Therefore, the recovered secret image is lossy. To recover the secret image without any loss, Thien and Lin introduce a pre-processing to decompose the gray value larger than 250, for example, 253 is separated as a pair of pixels {250 and 3}. This pre-processing will waste time and slightly increase the size of their shadows. However, since we use GF(256) in our weighted sharing

procedure, the secret image can be lossless reconstructed without additional post-processing.

Each participant keeps only one index and only one shadow (hence more convenient and space-saving):

If a company wants to apply Thien and Lin's (t, n) scheme directly to achieve the goal of weighted participants, then the company can let some participants hold multiple shadows. To be more specific, because each shadow generated by Thien and Lin's scheme has weight 1, the participant i ($1 \le i \le n$) whose weight is w_i should be assigned w_i shadows, and each of these w_i shadows will be attached with an index value for the secret-recovery meeting in the future. The w_i indices for the participant i will cause inconvenience than single index does, and the w_i shadows (rather than a single shadow) also waste storage space of participant i. Moreover, if there are three participants whose weights are, respectively, 128, 122, and 99; then, Thien and Lin's method will be in trouble. This is because the first participant will obtain 128 shadows with the 128 indices values being 1, 2, ..., 128; and the second participant will obtain 122 shadows with the indices values being 129, 130, ..., 250. As for the third participant, there is "no" shadow left for him because GF(251) restricts the input index value be less than 251; so the system cannot generate more than 250 shadows. However, by using our method, the first participant will obtain only a shadow with the index value 1 and the weight value 128; the second participant will obtain a shadow with the index value 2 and the weight value 122; and the third participant will obtain a shadow with the index value 1 and the weight value 99. Hence, besides giving convenience to each participant, the proposed method also keeps storage space of each participant much more economic.

3.3.3 Security analysis

The security analysis is divided into two parts: 1) a group of shadows with total weights t-1 cannot reveal the secret image and 2) shadows of different weights are not equally secure.

Firstly, suppose that the t'' obtained shadows are $\{(\tilde{h}_{k[1]}, w_{k[1]}), (\tilde{h}_{k[2]}, w_{k[2]}), ..., (\tilde{h}_{k[t'']}, w_{k[t'']})\}$ and the sum of their weights is $t''-1$ (i.e. $\sum_{j=1}^{t''} w_{k[j]} = t-1$), then we analyze the probability of obtaining the secret image by guessing. According the Chinese Remainder Theorem for polynomials, we can construct a unique polynomial $\tilde{f}'(x)$ with degree is less

than $\sum_{j=1}^{t''} w_{k[j]} = t-1$ from these m' shadows. After obtaining $\tilde{f}'(x)$, to reveal the $\tilde{f}(x)$ in Eq. (3.7) by $\tilde{f}'(x)$, we have

$$\tilde{f}(x) = \alpha \prod_{j=1}^{t''} (x - k[j])^{w_{k[j]}} + \tilde{f}'(x)$$

where α is a non-negative integer less than 2^8 =256 (because GF(2^8) is used in our experiments). Since there are 2^8 =256 possible values of α, the possibility of guessing the right solution $\tilde{f}(x)$ is 1/256. For a 512×512 secret image, because there are 512×512/t polynomials, the possibility of obtaining the right secret image is $256^{-(512\times512/t)}$, which is a form similar to the $251^{-(512\times512/t)}$ given in Thien and Lin's chapter.

Secondly, we analyze below the probability of obtaining the secret image by using only one shadow. Given a shadow h_{w_i} of weight w_i, then the polynomial $g_i^{w_i}(x)$ can be obtained using the shadow h_{w_i}. Now, to use $g_i^{w_i}(x)$ to reveal the $\tilde{f}(x)$ in Eq. (3.7), we have $\tilde{f}(x) = Q'(x)(x-i)^{w_i} + g_i^{w_i}(x)$ where $Q'(x)$ is an unknown polynomial with degree is less than $t-w_i$. Therefore, there is $1/256^{t-w_i}$ chance to find out the polynomial $Q'(x)$ by guessing. On the other hand, there are 512×512/t polynomials for a given 512×512 secret image, so the possibility of obtaining finding out the secret image is $256^{-(512\times512\times(1-w_i/t))}$. This shows that shadows of different weights are not equally secure, for the security of each shadow is weight-dependent. To find out the secret image by guessing, the owner of a larger-weight shadow has more chance than the owner of a smaller-weight has. This agrees with our daily-life experience: a higher-rank manager (having heavier weight) has more chance to uncover the company's secret than a lower-rank employee has.

3.4 Conclusions

In this chapter, a fast weighted secret image sharing with (t, n) threshold method is proposed. The method shares the secret image among the weighted participants, and the secret image can be losslessly recovered if the sum of the weights of the participants is greater than or equal to the threshold t. Besides, the execution time in the weighted secret image sharing phase is improved by using the properties of GF(2^k). As shown in Fig. 3.5, our execution time is better than that of Thien and Lin when $w_i > 1$. The executives of a company can use our

method to share the secret image.

Chapter 4

Weighted-Sum Function (WSF) – A Gray-scale Image Hiding Method with Competitive PSNR over a Wide Range of Embedding Rates

This chapter proposes an embedding method based on a weighted-sum function. A gray-scale host image is divided into blocks of n pixels, and each block embeds m secret bits in it. The stego-pixel values in each block are obtained by calculating the weighted-sum function with minimal distortions. The advantages of this method include: (1) Wide range of embedding rate (such as 0.5 to 4 bits per pixel), (2) Competitive image quality over the whole wide range, (3) Once the embedding rate (bits per pixel) is given, our look-up table can predict the PSNR value, even before the actual embedding.

The remainder of the chapter is organized as follows. The method is introduced in Sec. 4.1. Experimental results are presented in Sec. 4.2. Sec. 4.3 compares our method with previous works. Sec. 4.4 provides a discussion, and. Sec. 4.5 is the summary.

Notations in this chapter:

H	The gray-scale host image (After embedding, we obtain its stego-image H'.).
z	Number of pixels in each block of the host image.
m	Number of secret bits to be embedded in a block.
B_m	An m-bits binary value to be embedded in a block.
p_i	The i-th pixel value in a block of host image H (After embedding, we obtain its stego-pixel p_i'.).
p^{max}	The upper bound of host pixels and stego-pixels (each pixel value is between 0 and $p^{max}-1$).
Δp_i	The distortion between the stego-pixel p_i' and the host pixel p_i.
$(1=c_0,c_1,...,c_{z-1})$	The z weights repeatedly used by all blocks to extract their data B_m by the weighted-sum function

	$f(p'_0, p'_1,..., p'_{n-1}) = 1p'_0 + c_1 p'_1 + ... + c_{n-1} p'_{n-1} \pmod{2^m}$.
R_m	Weighted error sum $R_m = 1\Delta p_0 + c_1 \Delta p_1 + ... + c_{n-1} \Delta p_{n-1} \pmod{2^m}$ which is required to have the value $B_m - (1p_0 + c_1 p_1 + ... + c_{n-1} p_{n-1}) \pmod{2^m}$ (the so-called R_m-constraint in optimization).
$(\Delta p_0^*, \Delta p_1^*,..., \Delta p_{z-1}^*)_{R_m}$	The vector which has minimal sum of squares under the constraint $\sum_{i=0}^{z-1} c_i \Delta p_i = R_m \pmod{2^m}$.
T	The vector which has minimal sum of squares under the constraint $\sum_{i=0}^{z-1} c_i \Delta p_i = R_m \pmod{2^m}$.

4.1 The proposed method

Divide the gray-scale host image H into blocks of z pixels each. Also divide the data to be embedded into sectors of m bits each. Without a loss of generality, focus on one block and one sector. In other words, we show below how to embed an m-bits binary value B_m in an z-pixels block $(p_0, p_1,..., p_{z-1})$ of H. Let

$$p'_i = p_i + \Delta p_i \,;\, i = 0, 1, \ldots, z-1,$$

be the values of z stego-pixels, i.e. the pixel values after embedding B_m. Also assume that pixel values must be in the gray value range $[0, p^{max})$ where $p^{max} - 1$ is the maximal possible gray value. For example, if each gray has 8 bits, then $p^{max} = 2^8 = 256$. In the future, we want to extract B_m from the stego-pixels simply by a weighted-sum function

$$f(p'_0, p'_1,..., p'_{z-1}) = 1p'_0 + c_1 p'_1 + ... + c_{z-1} p'_{z-1} = B_m \pmod{2^m}, \tag{4.1}$$

where the given positive parameters $(c_0, c_1,..., c_{z-1}) = (1, c_1,..., c_{z-1})$ are called the weights of $f(p'_0, p'_1,..., p'_{z-1})$. In some studies for embedding, such as [8, 38], the base of modulus function is not necessarily a power of two. However, since the embedded data is often a binary stream, we set the modulus value to 2^m. Now, Eq. (4.1) reads

$$\begin{aligned} B_m &= 1p'_0 + c_1 p'_1 + ... + c_{z-1} p'_{z-1} \\ &= 1(p_0 + \Delta p_0) + c_1 (p_1 + \Delta p_1) + ... + c_{z-1} (p_{z-1} + \Delta p_{z-1}) \pmod{2^m}. \end{aligned} \tag{4.2}$$

So, the weighted error sum

$$R_m = 1\Delta p_0 + c_1\Delta p_1 + ... + c_{z-1}\Delta p_{z-1} \pmod{2^m} \tag{4.3}$$

is

$$R_m = B_m - (1p_0 + c_1 p_1 + ... + c_{z-1} p_{z-1}) \pmod{2^m}. \tag{4.4}$$

Notably, R_m can be evaluated by Eq. (4.4) in the embedding phase whenever the weights $(1,c_1,...,c_{z-1})$ are given or determined, since the secret data B_m and host pixel values $\{p_0, p_1,..., p_{z-1}\}$ are both known.

For n given weights $(1, c_1,..., c_{z-1})$, our purpose is to find optimal $(\Delta p_0^*, \Delta p_1^*,..., \Delta p_{n-1}^*)_{R_m}$ which minimizes the sum of squares under constraint (4.3). In symbols,

$$(\Delta p_0^*, \Delta p_1^*,..., \Delta p_{z-1}^*)_{R_m} = \arg \min_{\Delta p_0,...,\Delta p_{z-1}} \left\{ \sum_{i=0}^{z-1} (\Delta p_i)^2 \middle| \sum_{i=0}^{z-1} c_i \Delta p_i = R_m \pmod{2^m} \right\}. \tag{4.5}$$

This is a time-consuming combinatorial problem, and we use a dynamic programming skill to obtain a solution quickly. First, for the given $(1, c_1,..., c_{z-1})$, we generate table T based on Eq. (4.5). In other words, table T should list the suitable $(\Delta p_0^*, \Delta p_1^*,..., \Delta p_{z-1}^*)_{R_m}$ corresponding to each $R_m \in [0,2^m)$ (An example is given in Example 1 later). Notably, table T is only used in encoding. (As for decoding, the z stego-pixels $(p_0', p_1',..., p_{z-1}')$ are grabbed from the stego-image H' and then $B_m = f(p_0',..., p_{z-1}')$ is calculated by Eq. (4.1) to get the embedded m-bits secret value B_m).

Some recursive formulas are used in the dynamic programming (Algorithm 4.1). For $0 \le k < 2^m$ and $0 \le j < n$, define $Q[k, j]$ as the minimum of the partial sum of squares $\sum_{i=0}^{j} (\Delta p_i)^2$ obtained under the $(j+1)$-terms constraint

$$1\Delta p_0 + c_1\Delta p_1 + ... + c_j\Delta p_j = k \pmod{2^m}. \tag{4.6}$$

In symbols,

$$Q[k, j] = \min_{\Delta p_0,...,\Delta p_j} \left\{ \sum_{i=0}^{j} (\Delta p_i)^2 \middle| k = \sum_{i=0}^{j} c_i \Delta p_i \pmod{2^m} \right\}. \tag{4.7}$$

Eq. (4.7) implies that our original minimization goal is equivalent to getting $Q[k, z-1]$ for each $k \in [0,2^m)$. (The parameter values $(\Delta p_0^*, \Delta p_1^*,..., \Delta p_{n-1}^*)_{R_m}$ utilized to get $Q[k, z-1]$ are also the parameter values utilized to solve Eq. (4.5).) Now, in order to get $Q[k, z-1]$, we start from the 0th column $Q[\bullet,0]$. Then we get the 1st column $Q[\bullet, 1]$, and then the 2nd column $Q[\bullet, 2]$; ….; until we get the $(z-1)$th column $Q[\bullet, z-1]$. In other words, we need a recursive

formula to evaluate $Q[k, j]$. By observation, the value of $Q[k, 0]$ is

$$Q[k,0] = \min\left\{(\Delta p_0)^2 \middle| k = \Delta p_0 \pmod{2^m}\right\}$$
$$= \min\{k^2, (k-2^m)^2\} = \begin{cases} k^2 & \text{if } 0 \le k \le 2^{m-1} \\ (k-2^m)^2 & \text{otherwise} \end{cases}, \tag{4.8}$$

where we have used the $(j+1)=(0+1)=1$ term constraint $1\Delta p_0 = i \pmod{2^m}$ to know that either $\Delta p_0 = k$ or $\Delta p_0 = k - 2^m$. (It is impossible to have $\Delta p_0 = k + 2^m$ because $k^2 < (k+2^m)^2$.)

When $j \ge 1$, the recursive formula for $Q[k, j]$ can be derived as follows. Let $l=\Delta p_j$, then the $(j+1)$-terms constraint Eq.(6) becomes $k = 1\Delta p_0 + c_1\Delta p_1 + ... + c_j l \pmod{2^m}$, so

$$k - c_j l = 1\Delta p_0 + c_1\Delta p_1 + ... + c_{j-1}\Delta p_{j-1} \pmod{2^m} \tag{4.9}$$

is equivalent to the $(j+1)$-terms constraint Eq. (4.6).

By definition Eq. (4.7), $Q[\, k - c_j l \pmod{2^m}, j-1\,]$ is the minimal sum of squares $\sum_{i=0}^{j} (\Delta p_i)^2$ under the constraint Eq. (4.6), i.e.

$$Q[\, k - c_j l \pmod{2^m}, j-1\,] = \min_{\Delta p_0 \ldots \Delta p_{j-1}} \left\{ \sum_{i=0}^{j-1} (\Delta p_i)^2 \middle| k - c_j l = \sum_{i=0}^{j-1} c_i \Delta p_i \pmod{2^m} \right\}.$$

From this, we can express $Q[k, j]$ as

$$Q[k, j] = \min_{\Delta p_0 \ldots \Delta p_j} \left\{ \sum_{i=0}^{j} (\Delta p_i)^2 \middle| k = \sum_{i=0}^{j} c_i \Delta p_i \pmod{2^m} \right\}$$
$$= \min_{\Delta p_0 \ldots \Delta p_j} \left\{ (\Delta p_j)^2 + \sum_{i=0}^{j-1} (\Delta p_i)^2 \middle| k - c_j \Delta p_j = \sum_{i=0}^{j-1} c_i \Delta p_i \pmod{2^m} \right\}$$
$$= \min_{\Delta p_0 \ldots \Delta p_j, l} \left\{ l^2 + \sum_{i=0}^{j-1} (\Delta p_i)^2 \middle| k - c_j l = \sum_{i=0}^{j-1} c_i \Delta p_i \pmod{2^m} \right\} .$$
$$= \min_{l} \left\{ l^2 + \min_{\Delta p_0 \ldots \Delta p_j} \left\{ \sum_{i=0}^{j-1} (\Delta p_i)^2 \middle| k - c_j l = \sum_{i=0}^{j-1} c_i \Delta p_i \pmod{2^m} \right\} \right\}$$
$$= \min_{l} \left\{ l^2 + Q[\, k - c_j l \pmod{2^m}, j-1\,] \right\}$$

More precisely, $Q[k, j]$ can be obtained from

$$Q[k, j] = \min\left\{ Q[k - c_j l \pmod{2^m}, j-1] + l^2 \middle| l = 0, \pm 1, ..., \pm(2^m - 1) \right\} \tag{4.10}$$

by inspecting all l throughout the range $l \in \{0, \pm 1, \pm 2, \ldots, \pm(2^m - 1)\}$. Notably, there is no need to consider those l not in the range $(-2^m, 2^m)$, because $l=\Delta p_j$ and we want to minimize the sum

of squares $\sum_{i=0}^{j}(\Delta p_i)^2$ subject to constraint $k=\sum_{i=0}^{j}c_i\Delta p_i \pmod{2^m}$. If $(\Delta p_0,\Delta p_1,...,\Delta p_{j-1},\Delta p_j)$ satisfies constraint $k=\sum_{i=0}^{j}c_i\Delta p_i \pmod{2^m}$, and Δp_j is larger than the 2^m-1, then the constraint is still satisfied by $(\Delta p_0,\Delta p_1,...,\Delta p_{j-1},(\Delta p_j-2^m))$ because $(\Delta p_0+c_1\Delta p_1+...+c_{j-1}\Delta p_{j-1}+c_j(\Delta p_j-2^m)=k \pmod{2^m}$ is also true. However, $(\Delta p_0)^2+...+(\Delta p_{j-1})^2+(\Delta p_j-2^m)^2$ is smaller than $(\Delta p_0)^2+...+(\Delta p_{j-1})^2+(\Delta p_j)^2$.

Define table T as a matrix whose entries are

$$T[k,\,j]=\arg\min_{\Delta p_j}\left\{\sum_{i=0}^{z-1}(\Delta p_i)^2\,\middle|\,k=\sum_{i=0}^{z-1}c_i\Delta p_i \pmod{2^m}\right\}. \tag{4.11}$$

To obtain T from table Q, the selected value l of Eq. (4.10) is recorded in an auxiliary table L. In symbols, record

$$\begin{aligned}L[k,\,j]&=\arg\min_{\Delta p_j}\left\{\sum_{i=0}^{j}(\Delta p_i)^2\,\middle|\,k=\sum_{i=0}^{j}c_i\Delta p_i \pmod{2^m}\right\}\\&=\arg\min_{l}\left\{Q[k-c_j l \pmod{2^m},\,j-1]+l^2\,\middle|\,l=0,\pm1,...,\pm(2^m-1)\right\}\end{aligned}. \tag{4.12}$$

The value of $L[k,\,j]$ is simultaneously updated with $Q[k,\,j]$, so table L is obtained once table Q is obtained. Then, table T is constructed by a loop function. Firstly, because of Eq. (4.11) and Eq. (4.12), we have

$$T[k,\,z-1]=L[k,\,z-1] \text{ for each } k\in[0,2^m).$$

Secondly, $T[k,\,z-2]=L[k-c_{z-1}T[k,\,z-1] \pmod{2^m},\,z-2]$ for each $k\in[0,2^m)$, because

$$\begin{aligned}T[\,k,z-2\,]&=\arg\min_{\Delta p_{z-2}}\left\{\sum_{i=0}^{z-1}(\Delta p_i)^2\,\middle|\,k=\sum_{i=0}^{z-1}c_i\Delta p_i \pmod{2^m}\right\}\\&=\arg\min_{\Delta p_{z-2}}\left\{(\Delta p_{z-1})^2+\sum_{i=0}^{z-2}(\Delta p_i)^2\,\middle|\,k-c_{z-1}\Delta p_{z-1}=\sum_{i=0}^{z-2}c_i\Delta p_i \pmod{2^m}\right\}\\&=\arg\min_{\Delta p_{z-2}}\left\{T[k,z-1]^2+\sum_{i=0}^{z-2}(\Delta p_i)^2\,\middle|\,k-c_{z-1}T[k,z-1]=\sum_{i=0}^{z-2}c_i\Delta p_i \pmod{2^m}\right\}\quad(\because \Delta p_{z-1}=T[k,z-1])\\&=\arg\min_{\Delta p_{z-2}}\left\{\sum_{i=0}^{z-2}(\Delta p_i)^2\,\middle|\,k-c_{z-1}T[k,z-1]=\sum_{i=0}^{z-2}c_i\Delta p_i \pmod{2^m}\right\}\\&=L[\,k-c_{z-1}T[k,z-1] \pmod{2^m},z-2\,].\end{aligned}$$

Thirdly, $T[k,\,z-3]=L[k-c_{z-1}T[k,\,z-1]-c_{z-2}T[k,\,z-2] \pmod{2^m},\,z-3]$ for each $k\in[0,2^m)$, and the proof is similar, and hence, it is omitted. These steps are repeated until table T is complete.

Algorithm 4.1 below describes the details.

Algorithm 4.1. An auxiliary algorithm to construct table T by dynamic-programming.

Input: Two positive integers m and z, and z weights $(1, c_1,..., c_{z-1})$.

Output: A table T containing 2^m vectors (each vector is of the form $(\Delta p_0^*, \Delta p_1^*, ..., \Delta p_{n-1}^*)$), and an predicted *PSNR* value $PSNR_{est}$.

Remark: The Q, L and T in the algorithm are tables of size $2^m \times z$ each.

```
1   for k = 0 to 2^m–1 do
2     if  k^2 ≤ (k – 2^m)^2  then
3       L[k,0] ⇐ k
4       Q[k,0] ⇐ k^2
5     else
6       L[k,0] ⇐ k – 2^m
7       Q[k,0] ⇐ (k – 2^m)^2
8     end if
9   end for
10  for j = 1 to z–1 do
11    for k = 0 to 2^m–1 do
12      Q[k, j] ⇐ Q[k, j –1]
13      L[k, j] ⇐ 0
14      l ⇐ 1
15      while  l^2 < Q[k, j]  do
16        if  Q[k, j] ≥ Q[k + c_j l  (mod 2^m), j –1] + l^2  do
17          Q[k, j] ⇐ Q[k + c_j l  (mod 2^m), j –1] + l^2
18          L[k, j] ⇐ –l
19        end if
20        if  Q[k, j] ≥ Q[k – c_j l  (mod 2^m), j –1] + l^2  do
21          Q[k, j] ⇐ Q[k – c_j l  (mod 2^m), j –1] + l^2
22          L[k, j] ⇐ l
23        end if
24        l ⇐ l + 1
25      end while
26    end for
```

27	end for
28	for k = 0 to 2^m–1 do
29	$l \Leftarrow k$
30	for j = z–1 to 0 do
31	$T[k, j] \Leftarrow L[l, j]$
32	$l \Leftarrow l - c_j L[l, j] \pmod{2^m}$
33	end for
34	end for
35	Calculate MSEest (the expected value of MSE) and PSNRest (the predicted PSNR) by $$MSE_{est} = \frac{1}{n2^m} \sum_{k=0}^{2^m-1} Q[k, z-1], \quad (4.13)$$ $$PSNR_{est} = 10 \log_{10} (X-1)^2 / MSE_{est} . \quad (4.14)$$

Remarks: In Lines 15–25 above, the condition of the *while* loop is $l^2 < Q[k, j]$, rather than $l < 2^m$, because if $l^2 \geq Q[k, j]$, then the two "*if*" conditions in Lines 16–19 and Lines 20–23, i.e. $Q[k, j] \geq Q[k + c_j l \pmod{2^m}, j-1] + l^2$ and $Q[k, j] \geq Q[k - c_j l \pmod{2^m}, j-1] + l^2$, will never be satisfied.

Example 4.1: Assume (m, z)= (4, 3), and the z weights are (1, c_1, c_2)=(1, 2, 6). Table 4.1 is the three tables Q, L and the table T generated by Algorithm 4.1 above. The predicted PSNR provided by Algorithm 4.1 is $PSNR_{est} = 10 \log 255^2 / MSE_{est} = 51.14$ dB where $MSE_{est} = \frac{1}{n2^m} \sum_{k=0}^{2^m-1} Q[k, z-1] = 0.5$ is evaluated at Step 4 of Algorithm 4.1.

Table 4.1. The tables Q, L, and T for (m, z)=(4, 3) when z given weights are (1, c_1, c_2)=(1, 2, 6). (a): Table Q generated in the intermediate process of Algorithm 1. (b): Table L generated in the intermediate process of Algorithm 1. (c): The final output table T of Algorithm 1.

(a)

i	table Q		
0	0	0	0
1	1	1	1
2	4	1	1
3	9	2	2
4	16	4	2
5	25	5	2
6	36	8	1
7	49	10	2
8	64	13	2
9	49	10	2
10	36	8	1
11	25	5	2
12	16	4	2
13	9	2	2
14	4	1	1
15	1	1	1

(b)

i	table L		
0	0	0	0
1	1	0	0
2	2	1	0
3	3	1	0
4	4	2	1
5	5	2	1
6	6	2	1
7	7	3	1
8	8	3	1
9	−7	−3	−1
10	−6	−2	−1
11	−5	−2	−1
12	−4	−2	−1
13	−3	−1	0
14	−2	−1	0
15	−1	0	0

(c) table T

R_m	Δp_0^*	Δp_1^*	Δp_2^*
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	−1	1
5	−1	0	1
6	0	0	1
7	1	0	1
8	0	1	1
9	−1	0	−1
10	0	0	−1
11	1	0	−1
12	0	1	−1
13	−1	−1	0
14	0	−1	0
15	−1	0	0

Table 4.2. Suggested weights (1, c_1,..., c_{z-1}) for certain embedding rate values. For the listed (m, z), the estimated PSNR (i.e. value of Eq. (4.14)) is optimal if users adopt these suggested weights.

No.	**Embedding rate**	***m*, *z***	**Estimated PSNR**	**1, c_1,..., c_{z-1}**
0	0.500 bpp	4, 8	57.44 dB	1, 2, 3, 4, 5, 6, 7, 8
1	0.571	4, 7	56.58	1, 2, 3, 4, 5, 6, 7
2	0.667	4, 6	55.40	1, 2, 3, 4, 5, 6
3	0.750	6, 8	54.81	1, 2, 3, 4, 5, 6, 13, 26
4	0.875	7, 8	54.25	1, 2, 8, 12, 24, 29, 47, 62
5	1.000	6, 6	53.33	1, 2, 5, 12, 20, 28
6	1.167	7, 6	52.26	1, 3, 8, 18, 42, 54
7	1.200	6, 5	52.04	1, 6, 10, 18, 31
8	1.250	5, 4	51.64	1, 2, 6, 11
9	1.333	8, 6	51.40	1, 3, 9, 27, 50, 93
10	1.400	7, 5	50.97	1, 3, 9, 28, 52
11	1.500	6, 4	50.34	1, 3, 8, 22
12	1.600	8, 5	49.75	1, 3, 58, 87, 124
13	1.667	5, 3	49.09	1, 4, 10
14	1.750	7, 4	48.65	1, 4, 40, 58
15	1.800	9, 5	48.46	1, 36, 86, 146, 215
16	2.000	10, 5	47.31	1, 9, 23, 243, 324
17	2.250	9, 4	45.73	1, 13, 149, 232
18	2.500	10, 4	44.23	1, 26, 33, 221
19	2.750	11, 4	42.72	1, 364, 559, 986
20	3.000	12, 4	41.22	1, 9, 350, 491
21	3.333	10, 3	39.10	1, 20, 195
22	3.500	7, 2	38.00	1, 12
23	3.667	11, 3	37.10	1, 61, 597
24	4.000	12, 3	35.10	1, 1210, 2026

Algorithm 4.1 above needs the user to provide z weights (1, c_1,..., c_{z-1}). For the reader's benefit, some suggested weights for different combinations of (m, z) are provided in Table 4.2.

Notably, the value of bpp (bits per pixel) is always

$$\text{bpp}= m/z,$$

which is the number of embedded bits in each pixel of each block. The weights shown in Table 4.2 yield the optimal expected value of MSE for each pair of (m, z) specified there. These weights are found by an exhaustive search (i.e. all possible (1, c_1,..., c_{z-1}) in the searching domain (each $c_i \in [0,2^m)$) has been tested for the specified (m, z)). Notably, there is no need to test other values $c_i \notin [0,2^m)$, because they will be normalized to the scope $[0,2^m)$ by the modulus operator (mod 2^m).

Table 4.2 enables the readers to easily decide the weights (1, c_1,..., c_{z-1}). For example, if the host image is 512×512, and the size of embedded data is 500000 bits, then the embedding rate is 500000/(512×512)=1.91 bpp, which is between the 1.8 bpp and 2.0 bpp of Table 5.2. To get enough embedding space, the embedding rate cannot be smaller than 1.91, so we use 2.0 bpp. Therefore, from Table 4.2, use (m, z)=(10, 5), and choose $f(p'_0, p'_1, p'_2, p'_3, p'_4) = 1p'_0 + 9p'_1 + 23p'_2 + 243p'_3 + 324p'_4 \pmod{2^{10}}$ as the desired weighted-sum function Eq. (4.1) to extract embedded data from each stego-block $(p'_0, p'_1,..., p'_4)$. Of course, since (1, c_1,...,c_{z-1})=(1, 9, 23, 243, 324) are known, the corresponding table T can be constructed by Algorithm 4.1, and the data-embedding can be done by Algorithm 4.2 below. The optimal weights {1, c_1,...,c_{z-1}} in Table 4.2 may not be unique for each pair of m and z. For example, when m=z=6, to obtain PSNR = 53.33 dB, the readers can either use the weights (1, 2, 4, 12, 21, 28) or (1, 2, 5, 12, 20, 28) or (1, 3, 6, 12, 20, 28) or …; these optimal weights all give the predicted PSNR = 53.33 dB. Notably, there are two layers of optimization to construct Table 4.2, as listed below.

1. Inner layer: Given the z weights (c_0=1, c_1,…,c_{z-1}); then, for each integer $R_m \in [0,2^m)$, find the vector $(\Delta p_0^*, \Delta p_1^*,..., \Delta p_{z-1}^*)_{R_m}$ that yields the minimal sum of squares $\sum_{i=0}^{n-1} (\Delta p_i^*)^2$ under the constraint $\sum_{i=0}^{n-1} c_i \Delta p_i = R_m \pmod{2^m}$ (see Eq. (4.5)). This is resolved by dynamic programming (Algorithm 4.1).
2. Outer layer: Given a pair of values (m, z), find the z weights (1, c_1,…,c_{z-1}) which have a minimal expected value of MSE. This is resolved by an exhaustive search, and the results are listed in Table 4.2.

The details of the embedding algorithm are listed in Algorithm 4.2, which uses Algorithms

4.1 and 4.3 (Algorithm 4.3 is an auxiliary algorithm to handle the overflow/underflow case).

Algorithm 4.2. *Main embedding algorithm.*

Input: The embedded data S and host image H.

Output: A gray stego-image H'; the two values m, z and the z weights $(1, c_1,..., c_{z-1})$.

1. Calculate the embedding rate $er = |S|/|H|$, where $|S|$ is the bit-length of S and $|H|$ is the number of pixels of H.
2. Use the embedding rate er to look up the "Embedding rate" column in Table 4.2. Find an embedding rate er' which is the one closest to er, but still not less than er. For er', grab its corresponding (m, z) and the corresponding z weights $(1, c_1,..., c_{z-1})$ from Table 4.2.
3. For the z weights $(1, c_1,..., c_{z-1})$, if its table T was recorded earlier in the off-line process when Table 4.2 was constructed, then go to Step 4. Otherwise construct table T by Algorithm 4.1.
4. Let the z pixels $(p_0,..., p_{z-1})$ be the z not-yet-processed pixels taken from the host image H. Then let the m-bits value B_m be the m not-yet-processed bits taken from the bit stream S.
5. Calculate $R_m = B_m - (1p_0 + c_1 p_1 + ... + c_{z-1} p_{z-1}) \pmod{2^m}$.
6. Grab the R_m-th row $(\Delta p_0^*, \Delta p_1^*,..., \Delta p_{z-1}^*)_{R_m}$ of the table T. Calculate the z stego-pixels of the stego-image H' by $(p_0', p_1',..., p_{z-1}') = (p_0 + \Delta p_0^*, p_1 + \Delta p_1^*,..., p_{z-1} + \Delta p_{z-1}^*)$.
7. If any stego-pixel p_i' is out of boundary (<0 or $\geq p^{max}$), then re-calculate the z stego-pixels by calling the out-of-bound algorithm (Algorithm 4.3).
8. Go to step 4 if unprocessed data bits remain. Otherwise, output image H', the values of m, z, and the z weights $(1, c_1,..., c_{z-1})$.

In Algorithm 4.2 above, should some of the z generated stego-pixels $(p_0', p_1',..., p_{z-1}')$ in Step 6 be out of bound (i.e. <0 or $\geq p^{max}$.), then Algorithm 4.3 below is called in Step 7 of Algorithm 4.2 to re-generate $(p_0', p_1',..., p_{z-1}')$ which always stay within the gray-level range $[0, p^{max})$. Here, for each pixel p_i, to ensure $p_i + \Delta p_i \in [0, p^{\max})$ after embedding; the formula $Q[k, j] = \min\left\{Q[k - c_j l \pmod{2^m}, j-1] + l^2 \middle| l = 0, \pm 1,..., \pm(2^m - 1)\right\}$ in Eq. (4.10) is rewritten as

$$Q[k, j] = \min\left\{Q[k - c_j l \pmod{2^m}, j-1] + l^2 \middle| l = 0, \pm 1,..., \pm(2^m - 1) \text{ with } 0 \leq l + p_j < p^{\max}\right\}$$

so that the new stego-values created by Algorithm 4.3 can stay in bound by dynamical modification according to the given host pixel values $(p_0, p_1,..., p_{n-1})$. (In opposition, Table 4.2

and Algorithm 4.1 do not need any host pixel value.)

Algorithm 4.3. The out-of-bound algorithm to deal with the case when the gray value generated in Step 6 of Embedding Algorithm 5.2 is <0 or $\geq p^{max}$.

Input: Two positive integers m and z, the z weights $(1, c_1,..., c_{z-1})$, the z host pixels $(p_0, p_1,..., p_{z-1})$, the value R_m, and an integer p^{max} indicating that all host pixels are less than p^{max}.

Output: z stego-pixels $(p'_0, p'_1,..., p'_{n-1})$ of which each pixel is in the gray-level range $[0, p^{max})$.

Remark: The Q and L in the algorithm are tables of size $2^m \times z$ each.

1 for $k = 0$ to $2^m - 1$ do
2 $Q[k,0] \Leftarrow MAX$, where MAX is a very big number such as $(11\ldots1)_2$.
3 if $\boxed{p_0 + k < p^{\max}}$ and $k^2 \leq Q[k,0]$ then
4 $L[k,0] \Leftarrow k$
5 $Q[k,0] \Leftarrow k^2$
6 end if
7 if $\boxed{p_0 + (k - 2^m) \geq 0}$ and $(k - 2^m)^2 < Q[k,0]$ then
8 $L[k,0] \Leftarrow k - 2^m$
9 $Q[k,0] \Leftarrow (k - 2^m)^2$
10 end if
11 end for
12 for $j = 1$ to $z-1$ do
13 for $k = 0$ to $2^m - 1$ do
14 $Q[k, j] \Leftarrow Q[k, j-1]$
15 $L[k, j] \Leftarrow 0$
16 $l \Leftarrow 1$
17 while $l^2 < Q[k, j]$ do
18 if $\boxed{p_j - l \geq 0}$ and $Q[k, j] \geq Q[k + c_j l \pmod{2^m}, j-1] + l^2$ do
19 $Q[k, j] \Leftarrow Q[k + c_j l \pmod{2^m}, j-1] + l^2$
20 $L[k, j] \Leftarrow -l$
21 end if
22 if $\boxed{p_j + l < p^{\max}}$ and $Q[k, j] \geq Q[k - c_j l \pmod{2^m}, j-1] + l^2$ do
23 $Q[k, j] \Leftarrow Q[k - c_j l \pmod{2^m}, j-1] + l^2$
24 $L[k, j] \Leftarrow l$
25 end if

26	$l \Leftarrow l+1$
27	end while
28	end for
29	end for
30	$l \Leftarrow R_m$
31	for $j = z-1$ to 0 do
32	$p'_j \Leftarrow p_j + L[l, j]$
33	$l \Leftarrow l - c_j L[l, j] \pmod{2^m}$
34	end for

The extraction of the hidden data is easy; just use Algorithm 4.4 below.

Algorithm 4.4. *Extraction algorithm.*

Input: The stego-image H', and the value d which indices the d-th row in Table 4.2.

Output: The embedded data S.

1. Grab z not-yet-processed pixels of the stego-image H', denote the z pixel values as $(p'_0, p'_1, ..., p'_{z-1})$.
2. Calculate $B_m = 1p'_0 + c_1 p'_1 + ... + c_{z-1} p'_{z-1} \pmod{2^m}$, and append the m-bits value B_m to the tail of data S.
3. Go to step 1 until all pixels in stego-image H' are processed.

Example 4.2: If $(m, z) = (4, 3)$, and the weights are $(1, c_1, c_2) = (1, 2, 6)$, then use Algorithm 4.1 to establish table T, which is shown in Table 4.1. Then use Table 4.1 throughout the embedding phase. Assume the z=3 pixels of a host block are $(p_0, p_1, p_2) = (128, 127, 125)$, and an m-bits secret value is $B_m=(0111)_2=7$. Below we show how to embed 7 in the block. Firstly, calculate $R_m = 7 - (1\times128 + 2\times127 + 6\times125) = 11 \pmod{2^4}$ by Step 5 of Algorithm 4.2. Since R_m=11, take from Table 4.1 the row with R_m=11, which reads $(\Delta p_0^*, \Delta p_1^*, \Delta p_2^*)_{R_m} = (1,0,-1)$. Finally, calculate the z=3 stego-pixels values as (p'_0, p'_1, p'_2) =(128+1, 127+0, 125–1)=(129, 127, 124). In the future, secret data B_m is extracted by calculating $B_m = 1\times129+2\times127+6\times124 = 7 \pmod{16}$.

4.2 Experimental results

This section presents the experimental results. When data is embedded in a $ht\times wh$ host

image H to get its stego-image H', a PSNR value is computed to measure the quality of H'. The definitions is

$$PSNR = 10\log {(p^{\max}-1)^2}/{MSE} \quad \text{where} \quad MSE = \frac{1}{wh \times ht}\sum_{i=0}^{wh-1}\sum_{j=0}^{ht-1}[H(i,j)-H'(i,j)]^2 .$$

In our first experiment, let the host image be the 512×512 gray image Lena. Let the data to be embedded be a string generated by a random number generator. Fig. 4.1 shows six stego-versions of Lena with various embedding rates. The PSNR of the experimental results is extremely close to the theoretically-predicted $PSNR_{est}$ values (listed in Table 4.2), which are predicted by Algorithm 4.1. Thus, the PSNR value before embedding can really be predicted by Table 4.2. The actual PSNR values are still very close to the predicted PSNR values (Table 4.2) when the image Baboon in Fig. 4.2 is used as the host image.

Table 4.3 compares ours with other papers [8-10, 35, 38, 48, 49, 51, 52]. The host image is Lena, which is a common image found in all referenced experiments. It can be seen from Table 4.3 that our method achieves competitive PSNR values for each embedding rate. When the image is Baboon, Table 4.4 again shows that ours are very competitive. Among many embedding methods, [8, 9, 35] are very competitive to ours. A comparison of the three methods [8, 9, 35] is discussed below.

The LSB Matching method given by Li et al. [9] has very good stego-image quality, but the choice of the embedding rate (bits per pixel, i.e. bpp) is very limited. [9] uses 1 bpp as the embedding rate, and it has no algorithm or experiment to deal with the case when bpp≠1 (for example, when the embedding rate is 3.5 bpp). On the contrary, our new product is an all-in-one method with competitive quality everywhere over a wide range of embedding rates, for example, from 0.5 bpp to 4 bpp. (Our embedding rates include, but are not limited to, rates which are non-integer or smaller than one). In the embedding method given by Lin et al. [35], as shown in Tables 4.3 and 4.4, although [35] can almost keep up with ours when the embedding rate is 2, 3 or 4, the method cannot compete with ours when the bpp is 1 or non-integer. For example, we lead by a difference of about 2.2 dB when the embedding rate is 1.5, and we lead by a difference of about 3.3 dB when the embedding rate is 0.5. (In fact, the method in [35] only deals with the integer embedding rate. Hence, if the specified embedding rate is a non-integer, for example, 1.5 bpp, then embed 1 bits per pixel in one half of the image, and then 2 bits per pixel in the remaining half of the image). In the embedding method of Thein and Lin [8], as shown in Tables 4.3 and 4.4, although [8] can almost keep up with ours when the embedding rate is larger than 1 bpp, the method cannot compete with ours

when the embedding rate is 1 or less than 1 (for example, we win by 2.2 dB when the embedding rate is 1, and we win by 3.3 dB when the bpp is 0.5). In summary, the embedding rates of our method can have a wide range of embedding rate (from 4.0 to, say, 0.5 bpp), including rates which are non-integer or even smaller than 1 bpp. For the whole range of embedding rates, our method provides competitive PSNR values. In the less-than-1-bpp case, our PSNR values are very competitive (see Tables 4.3 and 4.4).

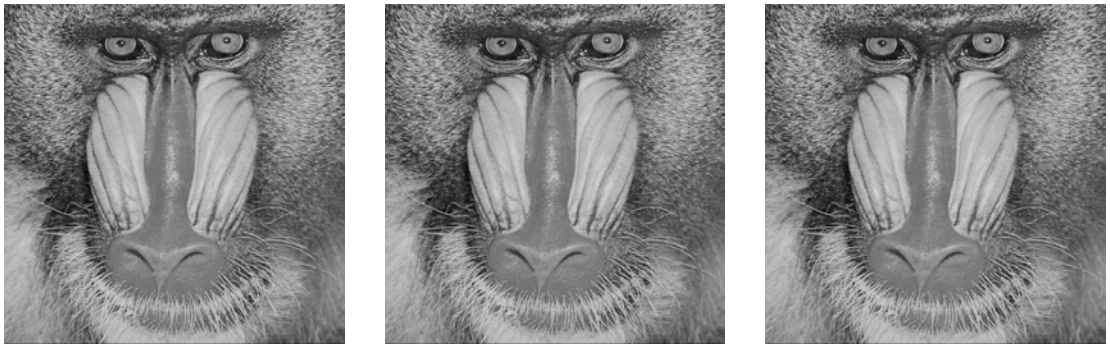
In Tables 4.5 and 4.6, some experiments are conducted to determine the PSNR values when the embedded data is real data rather than random. The host images are {Lena, Baboon, Jet, Sailboat, Peppers, Boat, Elaine, House}, all of which are 512×512, and the embedded data is the resized images decided by the embedding rate. In Tables 4.5-4.6, the real PSNRs (using real data) and the predicted values ($PSNR_{est}$) are still very close; the difference is at most 0.02 dB.

The following two experiments were also conducted to test the PSNR values of all possible weights (1, c_1,…, c_{z-1}). The host image is Lena, and the embedded data is random data. Firstly, we selected (m, z)=(5, 3) and tested the weights (1, c_1, c_2) for $c_1 \in [1,2^m)=[1,32)$ and $c_2 \in [1,2^m)=[1,32)$. The other values of c_1 and c_2 need not be tested because they would be normalized to the range [0, 2^m) by the modulus operation (mod 2^m). The cases c_1=0 or c_2=0 need not be considered, because they are equal to embedding m bits in z–1 pixels rather than z pixels (see Eq. (4.1), and the corresponding pixels p_1 (when c_1=0) or p_2 (when c_2=0) can be removed from Eq. (4.1)). The 31×31 $PSNR_{est}$ values (the 31×31 predicted *PSNR* values evaluated by Algorithm 4.1) were inspected for these 31×31 sets of (1, c_1, c_2), and the global maximum was found to have occurred eight times. More precisely, when (c_1,c_2) are respectively, (4,10), (4,22), (10,4), (10,28), (22,4), (22,28), (28,10), (28,22), the eight corresponding $PSNR_{est}$ values are all 49.087, and 49.087 is the maximum among all 31×31 $PSNR_{est}$ values given by Algorithm 4.1 (it should be noted that 49.087 is also the 49.09 listed in Table 4.2 if it is rounded to 4 significant digits.) Then, in Fig. 4.3, the 31×31=961 "real" PSNR values were sketched when the random data was embedded in Lena. The maximum of the 31^2 =991 real PSNR values was 49.094, which occurred at the (1,c_1,c_2)= (1,4, 10). It should be noted that 49.094 is very close to the predicted value of 49.087. It should also be noted that (1,c_1,c_2)= (1,4, 10) happens to be the weights listed in Table 4.2. For the reader's benefit, in Fig. 4.3, eight blue points are marked where (c_1,c_2) are, respectively, (4,10), (4,22), (10,4), (10,28), (22,4), (22,28), (28,10), (28,22). It can be seen that the real *PSNR* values at these eight theoretically-optimal points are also very high. Their actual *PSNR* values are at

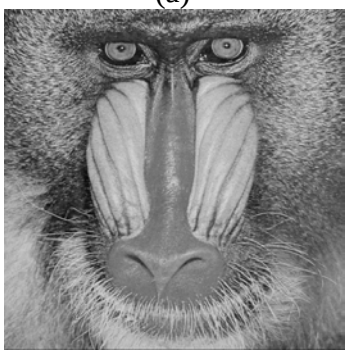
least 49.081.

In the other experiment, let $(m, z)=(7, 2)$ and test the weights $(1, c_1)$ for $c_1 \in [1,2^m)= [1,128)$. The maximum of the 127 $PSNR_{est}$ values evaluated by Algorithm 4.1 is 37.999, which is also the 38.00 listed in Table 4.2 if it is rounded to 4 significant digits. Then, in Fig. 4.4, the 127 "real" *PSNR* values were sketched when random data was embedded in Lena. The maximum of these 127 real *PSNR* values was 38.002, which occurred at the weight $c_1=116$. Again, 38.002 is very close to 37.999. For the reader's benefit, the two blue points ($c_1=12$ and 116) were also marked in Fig. 4.4 which are the places which generate 37.999 (the maximum of $PSNR_{est}$). It can be seen that the real *PSNR* values at these two points are also very high (one is 38.002 and the other is 37.995) From the two experiments above for real embedding, it was observed that the suggested weights listed in Table 4.2 can provide very high *PSNR* values (optimal or nearly optimal *PSNR* values, if they are compared with other weights). The real *PSNR* values are also very close to the predicted *PSNR* values listed in Table 4.2.

Fig. 4.1. The six Lena stego-images with various embedding rates. The embedding rates and the values of *PSNR* of the stego-images are (a): 0.5 bpp, 57.45 dB. (b): 1.0 bpp, 53.33dB. (c): 2.0 bpp, 47.30dB. (d): 3.0 bpp, 41.22 dB. (e): 3.33 bpp, 39.11dB. (f): 4.0 bpp, 35.10 dB.



(a) (b) (c)



(d)

Fig. 4.2. The Baboon stego-images with various embedding rates. The embedding rates and the values of *PSNR* of the stego-images are (a): 1.0 bpp, 53.33 dB. (b): 2.0 bpp, 47.30 dB. (c): 3.0 bpp, 41.21 dB. (d): 4.0 bpp, 35.11dB.

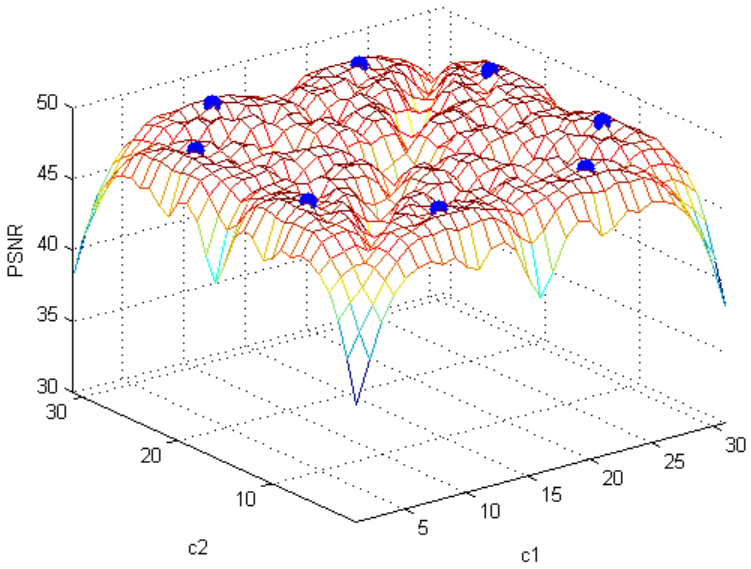


Fig. 4.3. The *PSNR* of embedding random data in Lena, for (m, z)=(5, 3), $c_1 \in [1,2^m)$= [1,32), and $c_2 \in [1,2^m)$ = [1,32). The real maximal *PSNR* for all possible combination of weights is

49.094 which is very close to 49.087. (The eight blue points are the places that generate 49.087 [the maximum of $PSNR_{est}$, if Algorithm 4.1 is executed for each combination of weights].)

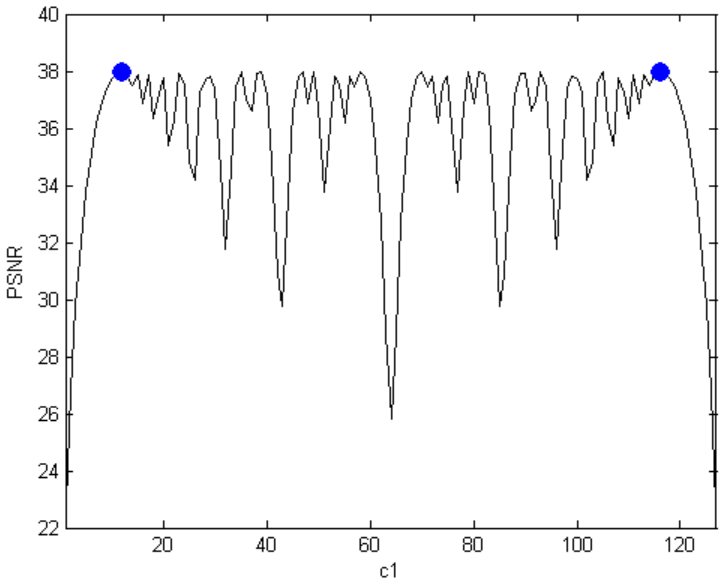


Fig. 4.4. The *PSNR* of embedding random data in Lena, for $(m, z)=(7,2)$ and $c_1 \in [1,2^m)=[1,128)$. The maximum of the real *PSNR* of all 127 possible weights is 38.002, very close to 37.999. (The two blue points are the places that generate 37.999 [the maximum of $PSNR_{est}$].)

Table 4.3. Comparison with other papers. Host image is Lena for all methods, and the embedded data are random numbers.

Embedding rate	**Methods**	***PSNR***
0.50 bpp	[8, 35]	54.14 dB
0.50	ours	57.44
0.75	[8, 35]	52.38
0.75	ours	54.82
1.00	[52]	51.14
1.00	[8, 35]	51.14
1.00	[10] $_{(z=2)}$	52.39
1.00	[9] $_{(z=6)}$	53.33
1.00	ours$_{(m=z=6)}$	53.33
1.16	[38]	52.11
1.17	ours	52.26
1.50	[35]	48.12
1.50	[8] $_{(\text{mod } 3)}$	49.89
1.50	ours	50.34

Embedding rate	**Methods**	***PSNR***
2.19 bpp	[51]	43.95 dB
2.25	ours	45.73
2.39	[51]	36.96
2.50	[35]	42.69
2.50	[8] $_{(\text{mod } 6)}$	43.12
2.50	ours	44.23
2.89	[52]	39.31
3.00	[8, 35]	40.73
3.00	ours	41.22
3.19	[51]	36.28
3.33	ours	39.11
3.50	[35]	36.82
3.50	[8] $_{(\text{mod } 12)}$	37.29
3.50	ours	38.00

1.56	[48]	41.79
1.56	[49]	44.10
1.99	[52]	45.14
2.00	[8, 35]	46.37
2.00	ours	47.30

3.53	[52]	34.54
3.67	ours	37.10
4.00	[8, 35]	34.80
4.00	ours	35.10

[9] did not have algorithm or experiment for bpp≠1.

Table 4.4. Comparison with other papers. Host image is Baboon for all methods, and the embedded data are random numbers.

Embedding rate	**Methods**	***PSNR***
0.5 bpp	[8, 35]	54.15 dB
0.5	ours	57.45
0.75	[8, 35]	52.41
0.75	ours	54.81
1.00	[8, 35]	51.13
1.00	[10] $_{(z=2)}$	52.39
1.00	[9] $_{(z=6)}$	53.33
1.00	ours$_{(m=z=6)}$	53.33
1.10	[48]	44.10
1.16	[38]	52.11
1.17	ours	52.26
1.50	[35]	48.12
1.50	[8] $_{(\text{mod } 3)}$	49.89
1.50	ours	50.34
1.74	[49]	40.3
2.00	[8, 35]	46.38
2.00	ours	47.30

Embedding rate	**Methods**	***PSNR***
2.49	[51]	42.08
2.50	[35]	42.68
2.50	[8] $_{(\text{mod } 6)}$	43.13
2.50	ours	44.23
2.99	[51]	34.20
3.00	[52]	39.16
3.00	[8, 35]	40.72
3.00	ours	41.21
3.49	[51]	33.01
3.50	[35]	36.82
3.50	[8] $_{(\text{mod } 12)}$	37.28
3.50	ours	38.00
4.00	[8, 35]	34.80
4.00	ours	35.11

[9] did not have algorithm or experiment for bpp≠1.

Table 4.5. *PSNR* values when secret data are also images. Each host image is 512×512, but each secret image is resized to be 234×234. Here, (m, z)=(5,3), (1,c_1,c_2)=(1,4,10), so the estimated *PSNR* is 49.09 dB according to Table 4.2.

secret \ host	Lena	Baboon	Jet	Sailboat	Peppers	Boat	Elaine	House
Lena	49.09	49.09	49.09	49.08	49.09	49.10	49.09	49.08
Baboon	49.09	49.08	49.09	49.09	49.09	49.09	49.10	49.09
Jet	49.10	49.09	49.10	49.08	49.08	49.08	49.09	49.09
Sailboat	49.07	49.09	49.08	49.09	49.09	49.09	49.10	49.09
Peppers	49.08	49.09	49.09	49.09	49.09	49.10	49.08	49.08
Boat	49.10	49.09	49.09	49.09	49.09	49.10	49.09	49.10
Elaine	49.08	49.08	49.08	49.09	49.10	49.08	49.08	49.09
House	49.08	49.09	49.09	49.10	49.08	49.08	49.09	49.09

Table 4.6. *PSNR* values when secret data are also images. Each host image is 512×512, but each secret image is resized to be 339×339. Here, (m, z)=(7,2), (1,c_1)=(1,12), so the estimated

PSNR is 38.00 dB according to Table 4.2.

secret \ host	Lena	Baboon	Jet	Sailboat	Peppers	Boat	Elaine	House
Lena	37.99	38.00	38.00	37.99	38.00	37.99	37.99	38.01
Baboon	38.00	38.00	38.00	37.99	37.99	38.00	38.01	38.00
Jet	38.00	37.99	38.00	38.01	38.01	38.00	37.99	38.01
Sailboat	38.01	38.00	38.00	38.01	38.00	37.99	38.00	38.00
Peppers	37.99	37.99	38.01	37.99	37.99	37.99	38.00	38.01
Boat	37.99	38.00	38.00	37.99	38.00	38.00	38.01	38.00
Elaine	37.98	37.98	37.99	38.00	38.00	38.01	38.00	37.99
House	38.00	37.99	38.02	38.01	37.99	38.01	38.00	38.02

4.3 Comparison with previous works

Property 4.1 below indicates that the modulus-based method [8] coincides with our z=1 case. The embedding scheme $f(p'_0, p'_1,..., p'_{z-1}) = 1p'_0 + 2p'_1 + ... + zp'_{z-1} \pmod{2z+1}$ proposed by Zhang and Wang [38] can relate to ours if we set our (1, c_1, c_2,..., c_{z-1}) to (1, 2, 3,…, z), and $m = \lfloor \log_2(2z+1) \rfloor$. Of course, in this particular (1, c_1, c_2,..., c_{z-1})=(1, 2,…, z) situation, their embedding capacity and *PSNR* of [38] will be a little different from ours because their module base is of the form (mod 2z+1) while ours is of the form (mod 2^m) where $m = \lfloor \log_2(2z+1) \rfloor$. However, as stated earlier in a paragraph below Eq. (4.1) of Sec. 4.1, the modulus base was set to 2^m to reduce computation time, since the embedded data is often a binary stream. In a certain sense, [38] has more freedom to choose a module base, while ours has more freedom to choose weights (1, c_1, c_2,..., c_{z-1}).

Property 4.1. If z is set to 1, i.e. only one weight is used in the extraction function f (which means $B_m = f(p'_0) = p'_0 \pmod{2^m}$), then our method coincides with the modulus-based embedding method $p'_0 = round\left(\frac{p_0 - B_m}{2^m}\right) \times 2^m + B_m$ of [1].

Proof: In [8], the embedding equation is $p'_0 = round\left(\frac{p_0 - B_m}{2^m}\right) \times 2^m + B_m$ where *round*(•) rounds the value • to the nearest integer. Their extraction equation is $B_m = p'_0 \pmod{2^m}$, and it is proved [8] that their distortion $\Delta p_0 = p'_0 - p_0$ is between $-(2^{m-1}-1)$ and 2^{m-1}. As for ours, if we set z as 1 in our method, then our extraction function Eq. (4.1) also reads $B_m = f(p'_0) = 1p'_0 \pmod{2^m}$. Since the given secret data B_m is fixed, the identical extraction formula means that the stego-values of the two methods are either identical or differ by a

whole multiple of 2^m. However, by letting z=1 in Algorithm 1, the Δp_0 in table T becomes

$$\Delta p_0 = \begin{cases} R_m & \text{if } 0 \le R_m \le 2^{m-1} \\ R_m - 2^m & \text{otherwise} \end{cases},$$

which means our $\Delta p_0 = p_0' - p_0$ is also between $-(2^{m-1}-1)$ and 2^{m-1}. Therefore, with the same given pixel value p_0, it is impossible that our stego-value and their stego-value differ by a nonzero integer multiple of 2^m. Therefore, the two stego-values must be the same. ∎

Property 4.2. If we let $(1, c_1, c_2, ..., c_{z-1})=(1, 2, 3, \ldots, z)$ and $m = \lfloor \log_2(2z+1) \rfloor$, then at most one of the z pixels in the block is distorted, and the distortion at that pixel is, at most, 1. (Zhang and Wang's method [38] neatly found this good property for distortion when module base is $2z+1$ rather than 2^m. Our proof below should be able to let the readers understand clearly why Property 4.2 holds.)

Proof: We show below that for R_m from 0 through $2^m - 1 = 2^{\lfloor \log_2(2z+1) \rfloor} - 1 \le 2n$, only a distortion in $\{-1, 0, 1\}$ is added to a pixel value p_i of the block to get its stego-pixel p_i'. Firstly, we evaluated the weighted error sum $R_m = B_m - (1p_0 + 2p_1 + ... + zp_{z-1}) \pmod{2^m}$. To minimize the sum of squares $\left\{ \sum_{i=0}^{z-1} \Delta p_i^2 \right\}$ under the constraint $1p_0 + 2p_1 + ... + zp_{z-1} \pmod{2^m} = B_m - (1p_0 + 2p_1 + ... + zp_{z-1}) \pmod{2^m} = R_m$, we analyzed the three cases as follows. It should be noted that the distortion is within ±1 in all three cases. Case a): If $R_m = 0$, then the optimal solution is $\Delta p_i = 0$, for i=0,1,…, $z-1$. Case b): If $R_m = j$ where $0 < j \le z$, then the optimal solution is $\Delta p_{j-1} = 1$ and $\Delta p_i = 0$, for any $i \ne j-1$. Because $1\Delta p_0 + 2\Delta p_1 + \ldots + z\Delta p_{z-1} \pmod{2^m} = j$ and the sum of squares $\sum_{i=0}^{z-1} (\Delta p_i)^2$ is 1 which is the minimal possible value. Case c). If $R_m = j = z+1, z+2, \ldots, 2^m - 1$, the optimal solution is $\Delta p_{2^m - j - 1} = -1$ and $\Delta p_i = 0$, for any $i \ne 2^m - j - 1$. Because $1\Delta p_0 + 2\Delta p_1 + \ldots + z\Delta p_{z-1} \pmod{2^m} = 2^m - j \pmod{2^m} = j$ and $\sum_{i=0}^{z-1} (\Delta p_i)^2$ is 1 which is the minimal possible value. ∎

Below we relate and compare ours with LSB matching methods [9, 10]. In [9], Li et al. elegantly generalize Mielikainen's gorgeous scheme [10] (a LSB matching scheme using $z=2$) to get a so-called generalized LSB matching (G-LSB-M) scheme. As shown in Property 4, when the embedding ratio is 1 bpp, if z is small enough, then the optimal weights $(1, c_1, c_2, ...,$

c_{z-1}) found by ours also solves the minimization problem of the LSB matching scheme of [9] (and [10], if z is 2), although our optimization goal (each Δp_i term is squared before summing up) is a little different from theirs. Notably, the two elegant LSB matching methods [9, 10] and ours are different in that (i). [9] has no algorithm or experiment to deal with the case when bpp≠1; but our embedding rates have a wide range from 0.5 bpp to 4.0 bpp. (ii). The distortion Δp_i of each pixel is required to be ±1 or 0 in LSB matching, but ours does not use this requirement because we need to embed m-bits data in z pixels, rather than embedding z bits in z pixels. (iii). The goal of LSB matching is $\min\left\{\sum_{i=0}^{z-1}|\Delta p_i|\right\}$; whereas our goal is $\min\left\{\sum_{i=0}^{z-1}(\Delta p_i)^2\right\}$.

Property 4.3. When the embedding rate is as in [9, 10] (i.e. when m/z=1 bpp), if our optimal weights (1, c_1, c_2,..., c_{z-1}) also yield (or require) $\Delta p_i \in \{0, \pm 1\}$ for all i=0,1,…, z–1, then our optimal weights also resolve the optimization problem of LSB matching methods [9, 10].

Proof: If the distortion of each pixel Δp_i is only {0, ±1}, then $\sum_{i=0}^{z-1}(\Delta p_i)^2 = \sum_{i=0}^{z-1}|\Delta p_i|$. Therefore, the goal of LSB matching $\min\left\{\sum_{i=0}^{z-1}|\Delta p_i|\right\}$ is the same as our $\min\left\{\sum_{i=0}^{z-1}(\Delta p_i)^2\right\}$, so the weights (1, c_1,..., c_{z-1}), which are found by our exhaustive search are also a solution to LSB matching. ∎

Property 4.4. When the embedding rate m/z=1 is bpp, if each block unit contains z<4 pixels, then our optimal weights also resolve the optimization problem of LSBM (LSB matching) method [9] (and [10], if z=2.)

Proof: z<4 means that the optimal solution (1, c_1, c_2,..., c_{z-1})$^{\text{LSBM}}$ given by the LSB matching method will yield $(\sum_{i=0}^{z-1}(\Delta p_i)^2)^{\text{LSBM}} = (\sum_{i=0}^{z-1}|\Delta p_i|)^{\text{LSBM}} \le 1 \times z < 1+1+1+1=4$ due to item (ii) above. As a result, the $(\sum_{i=0}^{z-1}(\Delta p_i)^2)^{\text{ours}}$ obtained using our optimal weights (1, c_1, c_2,..., c_{z-1})$^{\text{ours}}$ also yields $(\sum_{i=0}^{z-1}(\Delta p_i)^2)^{\text{ours}} \le (\sum_{i=0}^{z-1}(\Delta p_i)^2)^{\text{LSBM}} < 1+1+1+1=4$, since our optimization goal is to

minimize $\sum_{i=0}^{z-1}(\Delta p_i)^2$. Therefore, in our optimal solution, $\Delta p_i \in \{0, \pm 1\}$ for all i. (None of our $|\Delta p_i|$ can be 2 or more; otherwise, $(\sum_{i=0}^{z-1}(\Delta p_i)^2)^{\text{ours}} \geq 2^2=4$, a contradiction.) Property 4.3 above then ensures that our optimal weights $(1, c_1, c_2,..., c_{z-1})^{\text{ours}}$ also resolve the optimization problem of LSB matching methods [9, 10]. ∎

4.4 Analyses

Some factors about the proposed method are analyzed in this section.

4.4.1 Running time of Algorithm 1.

Lines 1–9 of Algorithm 1 compute the 0^{th} column Q[•, 0] of table Q, its time complexity is $\Theta(2^m)$. Lines 10–27 compute the j-th columns Q[•, j] for each $j \in [1, n)$; Lines 11–26 calculates $Q[k, j]$, for each $k \in [0, 2^m)$; and Lines 15–25 repeat (e.g. e times) until $l^2 \geq Q[k, j]$. Therefore, Lines 10–27 needs $\Theta((z-1)\,2^m e)$ seconds. Lines 28–34 generate table T; and it needs $\Theta(z2^m)$ seconds. Line 35 calculates the predicted $PSNR_{est}$ and MSE_{est}; and it needs $\Theta(2^m)$ seconds. Therefore, the total running time of Algorithm 1 is $\Theta(2^m) + \Theta((z-1)2^m e) + \Theta(z2^m) + \Theta(2^m) = \Theta(z2^m e)$. The lower bound of running time is $\Omega(z2^m)$ by plugging in $e = 1$. As for the upper bound, the loop condition of Lines 15–25 is $l^2 < Q[k, j]$, and the maximum of $Q[k, j]$ is $Q[2^{m-1}, 0] = 2^{m-1}$, so $l^2 < Q[k, j] \leq 2^{m-1}$, which implies that $l \leq 2^{(m-1)/2}$. Hence $e \leq 2^{(m-1)/2}$; so the upper bound of running time is $O(z2^m 2^{(m-1)/2}) = O(z2^{(3m-1)/2})$.

Although the time complexity $O(z2^{(3m-1)/2})$ increases exponentially as m increases (partially because the size of table T is $2^m \times z$), the time is still acceptable, because the values of m listed in Table 4.2 are, at most, 12, and Table 4.2 has covered a practical range of embedding rates from 0.5 bpp to 4 bpp. To verify this, the running time of all embedding rates listed in Table 4.2 were tested. Our personal computer uses a Pentium D 2.80 GHz CPU, and the programming language is JAVA. As indicated in the fourth column of Table 4.7, the maximum CPU time to run Algorithm 1 was less than 0.016 seconds.

Table 4.7. The running time for various $(1, c_1,..., c_{z-1})$.

No.	bpp	m, z	1, c_1,..., c_{z-1}	The CPU	Total CPU seconds

				seconds to generate table *T* by dynamic programming.	**to embed random data in 512×512 Lena (including the time to generate table *T*)**
0	0.500	4, 8	1, 2, 3, 4, 5, 6, 7, 8	0.000032 seconds	0.438 seconds
1	0.571	4, 7	1, 2, 3, 4, 5, 6, 7	0.000032	0.438
2	0.667	4, 6	1, 2, 3, 4, 5, 6	0.000031	0.36
3	0.750	6, 8	1, 2, 3, 4, 5, 6, 13, 26	0.000172	0.359
4	0.875	7, 8	1, 2, 8, 12, 24, 29, 47, 62	0.000390	0.375
5	1.000	6, 6	1, 2, 5, 12, 20, 28	0.000141	0.359
6	1.167	7, 6	1, 3, 8, 18, 42, 54	0.000297	0.359
7	1.200	6, 5	1, 6, 10, 18, 31	0.000093	0.359
8	1.250	5, 4	1, 2, 6, 11	0.000047	0.375
9	1.333	8, 6	1, 3, 9, 27, 50, 93	0.000907	0.375
10	1.400	7, 5	1, 3, 9, 28, 52	0.000281	0.422
11	1.500	6, 4	1, 3, 8, 22	0.000094	0.375
12	1.600	8, 5	1, 3, 58, 87, 124	0.000797	0.375
13	1.667	5, 3	1, 4, 10	0.000031	0.391
14	1.750	7, 4	1, 4, 40, 58	0.000219	0.375
15	1.800	9, 5	1, 36, 86, 146, 215	0.001219	0.375
16	2.000	10, 5	1, 9, 23, 243, 324	0.004594	0.39
17	2.250	9, 4	1, 13, 149, 232	0.001094	0.39
18	2.500	10, 4	1, 26, 33, 221	0.002688	0.406
19	2.750	11, 4	1, 364, 559, 986	0.006453	0.485
20	3.000	12, 4	1, 9, 350, 491	0.015536	0.515
21	3.333	10, 3	1, 20, 195	0.00225	0.406
22	3.500	7, 2	1, 12	0.000125	0.406
23	3.667	11, 3	1, 61, 597	0.005312	0.406
24	4.000	12, 3	1, 1210, 2026	0.01486	0.5

4.4.2 Running time of main embedding algorithm (Algorithm 4.2).

In terms of the running time of Algorithm 4.2, the three major parts are: to generate table T (Step 3); to embed m bits data in n pixels (Step 4–6); and to process the overflow/underflow case (Step 7). The running time of generating table T is discussed in topic I. The embedding steps (Steps 4–6) need $\Theta(z)$ seconds to calculate the value of R_m, and $\Theta(z)$ seconds to calculate the z stego-pixels $(p'_0, p'_1, ..., p'_{z-1})$, so processing the whole image of $|H|$ pixels will need $\Theta(|H|)$ seconds. Should a stego-pixel p'_i be out of boundary, then the z stego-pixels are re-calculated by Algorithm 4.3. The time complexity of Algorithm 4.3 is the same as Algorithm 4.1. If a pixel value of the host image is close to 0 or p^{max}, then it has a higher probability to be an overflow or underflow. Suppose the probability is α, i.e., in the $|H|/z$ blocks of the host image H, $\alpha|H|/z$ blocks have underflow/overflow, and hence, they need to

call Algorithm 4.3, then the total time needed by Algorithm 4.2 is between $\Omega(z2^m)+\Theta(|H|)+\Omega(\alpha|H|/z\times z2^m)$ = $\Theta(|H|)+\Omega((z+\alpha|H|)2^m)$ and $O(z2^{(3m-1)/2})+\Theta(|H|)+O(\alpha|H|/z\times z2^{(3m-1)/2}) = \Theta(|H|)+O((z+\alpha|H|)2^{(3m-1)/2})$.

We also tested the running time for each embedding rate listed in Table 4.2. The host image is the 512×512 Lena, and the embedding data is random. The results are listed in the final column of Table 4.7. For each bpp listed in Table 4.2, the total CPU time to embed random data in Lena was between 0.359 seconds and 0.515 seconds.

4.4.3 Expected value of MSE for our method

Theorem 1 below states that the MSE_{est} evaluated by Eq. (4.13) of Algorithm 4.1 is, in fact, the expected value of MSE, as long as the embedded data is random, i.e. as long as the data can satisfy the following criteria:

i. For each embedded digit B_m, we have $P(B_m=0)=P(B_m=1)=\ldots=P(B_m=2^m-1)=1/2^m$. Here, $P(\bullet)$ is the probability of an event.

ii. The value of embedded digit B_m and the z values of the host pixels $\{p_0, p_1,\ldots,p_{z-1}\}$are independent.

Generally, the embedded data cannot completely satisfy (i), so the actual *PSNR* is only close to (but not identical to) the $PSNR_{est}$.

Theorem 4.1. Given the z weights $(1, c_1,\ldots, c_{z-1})$, if the embedded data are random, then the expected value of MSE is

$$MSE_{est} = \frac{1}{2^m z}\sum_{k=0}^{z-1}\min_{\Delta p_0,\ldots,\Delta p_{z-1}}\left\{\sum_{i=0}^{z-1}(\Delta p_i)^2 \middle| k=\sum_{i=0}^{z-1}c_i\Delta p_i \pmod{2^m}\right\}. \quad (4.15)$$

This value equals to the MSE_{est} obtained by Eq. (4.13) of the Algorithm 4.1.

Proof: Consider z pixels $\{p_0, p_1,\ldots, p_{z-1}\}$ in an image block. The embedding equations are Eq. (4.3) and (4.4); and for each $R_m \in [0,2^m)$, our goal in Eq. (4.5) is to find the 2^m vectors $(\Delta p_0^*, \Delta p_1^*,\ldots,\Delta p_{z-1}^*)_{R_m}$ satisfying

$$(\Delta p_0^*, \Delta p_1^*,\ldots,\Delta p_{z-1}^*)_{R_m} = \arg\min_{\Delta p_0,\ldots,\Delta p_{z-1}}\left\{\sum_{i=0}^{z-1}(\Delta p_i)^2 \middle| \sum_{i=0}^{z-1}c_i\Delta p_i = R_m \pmod{2^m}\right\}$$

. Since each vector $(\Delta p_0^*, \Delta p_1^*,\ldots,\Delta p_{z-1}^*)_{R_m}$ has minimal sum of squares, to minimize *MSE*, there is no need to consider other vectors $(\Delta p_0, \Delta p_1,\ldots,\Delta p_{z-1})$ that have larger sum of squares.

Given a value $k \in [0,2^m)$, the probability for the event R_m=k is

$$P(R_m = k) = P(B_m - \sum_{i=0}^{z-1} c_i p_i = k \pmod{2^m})$$

$$= \sum_{i=0}^{2^m-1} P((B_m = i) \cap (\sum_{i=0}^{z-1} c_i p_i = i - k \pmod{2^m}))$$

$$= \sum_{i=0}^{2^m-1} \left[P(B_m = i) \times P(\sum_{i=0}^{z-1} c_i p_i = i - k \pmod{2^m}) \right] \quad (\because \text{By the property } (ii) \text{ of the random data})$$

$$= \sum_{i=0}^{2^m-1} \left[1/2^m \times P(\sum_{i=0}^{z-1} c_i p_i = i - k \pmod{2^m}) \right] \quad (\because \text{By the property } (i) \text{ of the random data})$$

$$= 1/2^m \sum_{i=0}^{2^m-1} P(\sum_{i=0}^{z-1} c_i p_i = i - k \pmod{2^m})$$

$$= 1/2^m.$$

Therefore, in the embedding process, the 2^m events R_m=0, R_m=1,…, R_m=2^m–1 have uniform probability:

$$P(R_m = 0) = P(R_m = 1) = \ldots = P(R_m = 2^m - 1) = 1/2^m. \tag{4.16}$$

Therefore, the expected value of MSE is the average of all mean-squares of $(\Delta p_0^*, \Delta p_1^*, ..., \Delta p_{z-1}^*)_{R_m}$. By Eq. (4.5), the mean-squares of $(\Delta p_0^*, \Delta p_1^*, ..., \Delta p_{z-1}^*)_{R_m}$ is

$$\frac{1}{z} \sum_{i=0}^{z-1} (\Delta p_i^*)^2 = \frac{1}{z} \min_{\Delta p_0, ..., \Delta p_{z-1}} \left\{ \sum_{i=0}^{z-1} (\Delta p_i)^2 \middle| R_m = \sum_{i=0}^{z-1} c_i \Delta p_i \pmod{2^m} \right\},$$

so the expected value of *MSE* is

$$MSE_{est} = \sum_{k=0}^{z-1} \left\{ P(R_m = k) \times \frac{1}{z} \min_{\Delta p_0, ..., \Delta p_{z-1}} \left\{ \sum_{i=0}^{z-1} (\Delta p_i)^2 \middle| k = \sum_{i=0}^{z-1} c_i \Delta p_i \pmod{2^m} \right\} \right\}$$

$$= \sum_{q=0}^{z-1} \left\{ \frac{1}{2^m} \times \frac{1}{z} \min_{\Delta p_0, ..., \Delta p_{z-1}} \left\{ \sum_{i=0}^{z-1} (\Delta p_i)^2 \middle| k = \sum_{i=0}^{z-1} c_i \Delta p_i \pmod{2^m} \right\} \right\}$$

$$= \frac{1}{2^m z} \sum_{q=0}^{z-1} \min_{\Delta p_0, ..., \Delta p_{z-1}} \left\{ \sum_{i=0}^{z-1} (\Delta p_i)^2 \middle| k = \sum_{i=0}^{z-1} c_i \Delta p_i \pmod{2^m} \right\}.$$

This value can be rewritten as $MSE_{est} = \frac{1}{2^m z} \sum_{k=0}^{z-1} Q[k, z-1]$, which is Eq. (4.13) of Algorithm 4.1, because Eq. (4.7) reads that $Q[k, z-1] = \min_{\Delta p_0, ..., \Delta p_{z-1}} \left\{ \sum_{i=0}^{z-1} (\Delta p_i)^2 \middle| k = \sum_{i=0}^{z-1} c_i \Delta p_i \pmod{2^m} \right\}$. ∎

4.4.4 Application of the predicted *PSNR*

Given the desired embedding rate, our *PSNR* can be predicted in advance by Table 4.2. It

is observed that, to embed random or ordinary data, the predicted *PSNR* values are very close to the actual *PSNR* values (see Tables 4.5 and 4.6 for example). Thus, the embedding impact on the host image can be predicted even before the actual embedding. With this property of *PSNR*-prediction, if a customer tells us the minimal *PSNR* value he can tolerate, then we can look up Table 4.2 and find the embedding rate he needs. Then, for each natural or random secret data the customer gives us, we can use this embedding rate and data size to find the required size of host image. This determines a group of host images which are usually suitable for the secret data given by that customer. Without such an ability to predict *PSNR*, the above case must repeatedly be tested on several host images of different sizes. For example, firstly the customer's secret data is embedded in a 256×256 host image. If the resulting *PSNR* value is too low, then a 300×300 host image is tried, etc. This embedding process should be repeated several times until the *PSNR* quality of the stego-image meets the customer's requirement. Of course, if some people try to cut the trial-and-error time by using a super-large host image, the price they will pay is that the size of the stego-image is probably way too large for further processing, transmission, storage, or carriage.

4.4.5 Worst case *PSNR*

If people consider the worst case *PSNR* before using an embedding method (i.e. for that specified method, how bad can the resulting *PSNR* be if the user embeds a data set which is completely unsuitable for the host image), then our embedding method also performs better than the modulus-based embedding methods [8, 35], as indicated by Table 4.8, in which our gains are 2 dB or more. The worst case host image happens when the embedded data is specially designed according to the given host image, and the design of this data is in a pixel-by-pixel manner so that it gives the worst possible pixel-distortion at every pixel of the given host image. In our experiments for each method, since the theoretically worst possible pixel-distortion that the specified method could yield at a pixel did occur at every pixel of the stego-image we used, we already achieved the worst case *PSNR*, so there was no need to check all possible images. In other words, Table 8 not only gives the theoretically worst possible *PSNR* values for each method, but also the worst possible *PSNR* values in the real world for each method in the table.

Table 4.8. The worst-case *PSNR* values. ([6] did not have algorithm or experiment for bpp≠1. The worst-case *PSNR* value for [6] is also 51.14 dB if bpp=1.)

Embedding rate	Methods	Embedded data type	*PSNR*	dB difference from ours
1.00 bpp	[1,2]	Artificial data set 1a	48.13 dB	3.01
1.00	ours	Artificial data set 1b	51.14	-
2.00	[1,2]	Artificial data set 2a	42.11	2.6
2.00	ours	Artificial data set 2b	44.71	-
3.00	[1,2]	Artificial data set 3a	36.09	2.04
3.00	ours	Artificial data set 3b	38.13	-
4.00	[1,2]	Artificial data set 4a	30.07	2.00
4.00	ours	Artificial data set 4b	32.07	-

4.5 Conclusion

This chapter proposes an embedding method based on the weighted sum function. As shown in the figures and tables, our method has a wide range of embedding rates (0.5–4.0 bpp), and has competitive *PSNR* over the whole range. The predicted *PSNR* values ($PSNR_{est}$ by Eq. (4.14)) are also extremely close to the actual *PSNR* values. Therefore, the embedding error can be predicted even before the actual embedding. With this *PSNR*-prediction property (Table 4.2), for each secret data the customer gives us, we can determine the necessary size of host image if the customer also specifies the minimal *PSNR* value he can tolerate. This determines a set of host images for that secret data. Sec. 4.3 proved that Modulus-based method [8] and LSB matching methods [9, 10] are special cases of ours. The worst-case *PSNR* discussed in Item V of Sec. 4.4 also shows that, even if some very strange data (data artificially made by picky users and quite unnatural) was to be embedded, our method can still compete.

Chapter 5
Authentication and Recovery of an Image by Sharing and Lattice-embedding

Based on sharing and lattice-embedding techniques, this chapter presents an authentication-recovery method for an image. The recovery data are shared among many shadows, then lattice-embedding is utilized to embed each shadow in the DCT domain of an 8×8 block, respectively. The proposed method can resist certain content-preserving operations such as JPEG compression, Gaussian noise, and brightness adjustment, up to a tolerance level which is controlled by a quantization parameter value. The method can also resist certain security attacks such as cut-and-paste attack, collage attack, and VQ attack. Comparing with previous works, the proposed method have following major advantages and novelty: i) the method has no need to predict the trace of tampering, and the tampered blocks are always recovered as long as the number of valid blocks reaches a threshold; ii) Lattice-embedding yields smaller distortion than parity-check quantization does, and the latter was often used in reported works.

5.1 Introduction

It is easy for attackers to modify public digital media, and many authentication techniques, called watermarking, have been reported in recent years to protect the digital media. These techniques can check the correctness of the media content. Recently, some fragile watermarking schemes for tampered-region's detection and recovery have also been introduced. Based on Pascal transform, Varsaki et al.[72] proposed a semi-fragile watermarking with the recovery ability to deal with color images. The host is a color image, but the recovery data is a 1/16-size gray-level version of the host. The data is embedded in the Pascal domain of R, G, and B color components, respectively. As a result, the recovered region of the tampered color image is gray-leveled, rather than colored. However, as demonstrated in Fig. 5.7 of Sec. 5.4.3, the recovery ability is not so good after cropping in the central area of the image. Tsai and Chien [16, 17] proposed a method based on discrete

wavelet transform. The verification data and the recovery data is generated from low-frequency bands and then embedded in high-frequency bands. This method can resist JPEG compression and Gaussian noise.

Based on (t, n) sharing and lattice embedding, a novel semi-fragile watermarking method with recovery ability is proposed. The motivation of the proposed method is based on the three observations as follows. First, the (t, n) two-layer sharing, which will be introduced in Sec. 5.2.2, can recover the embedded data, as long as no more than $\lfloor (n-t)/2 \rfloor$ of the n created shadows are damaged shadows. This is an important property for our design to recover the image. Since each of our generated shadows is embedded in a block, the watermarked image can still extract useful recovery data after tampering, as long as the percentage of validity blocks reaches a pre-defined threshold β. If sharing-related techniques were not used, and if, for example, traditional block-mapping sequence techniques were used instead, the recovery data of some tampered blocks would have been lost because it was hard to predict in advance which blocks would be tampered (More details are addressed in Part (a) of Sec. 5.5). Second, the generated shadows E_i have the ability to both verify and recover. Because both of these abilities are tied together in E_i, we only need to embed one shadow E_i (rather than two data sets) in an 8×8 host block. This simplifies the design. Finally, We use lattice embedding to replace the so-called parity-check embedding used by many methods[13, 14, 16, 17] when data is to be embedded. The major reason for this is that lattice embedding has a smaller impact on the host image (More details are addressed in Part (b) of Sec. 5).

The remainder is organized as follows. Sec. 5.2 introduces the (t, n) two-layer sharing and lattice embedding which are utilized in our design. Sec. 5.3 describes our method, including a mathematical property. Sec. 5.4 shows the experimental results. Sec. 5.5 discusses the difference between the proposed method and other works, and Sec. 5.6 is the conclusion.

Notations in this chapter:

n	Number of created shadows in a (t, n) secret sharing.
t	The threshold of (t, n) secret sharing.
P_i	The i^{th} shadow of (t, n) two layer sharing.
c	The size of P_i.
E_i	The generated shadow which is attached to P_i by (t, n) two layer sharing.
M	The step size of lattice embedding.
d_i	The i^{th} DCT value of a 8×8 image block.

ID	The image's identification number.
Key	A secret key.
β	A specified threshold for the percentage of valid blocks in a tampered image.

5.2 Related works

Secret image sharing[7] and RS code technique[34] are briefly reviewed in Sec. 5.2.1; the two-layer sharing technique is introduced in Sec. 5.2.2, and lattice embedding is reviewed in Sec. 5.2.3. These techniques will be used by the proposed method in Sec. 5.3.

5.2.1 Secret image sharing[7] and RS code technique[34]

In the sharing phase of Thien and Lin's (t, n) threshold method[7], for each non-overlapping t pixel values of the secret image (secret message), a related polynomial is defined as

$$f(x) = a_0 + a_1 \times x + \cdots + a_{t-1} \times x^{t-1} \pmod{p} \qquad (5.1)$$

where a_0, a_1,..., a_{t-1} are the gray values of the t pixels, and p is a prime constant. Then $f(1)$, $f(2)$,…, $f(n)$ is evaluated and sequentially attached to the n shadows. Having processed all of the pixels in the secret image, the n shadows are generated. Since each t pixels in the secret image only contributes one pixel to each generated shadow, the size of each shadow is $1/t$ of the secret image.

As indicated by Preparata[33], from the mathematical viewpoint, the encoding phase of using the sharing equation (1) is isomorphic to the creation of a Reed-Solomon code (RS code)[34]. Therefore, by the error-correction property of the RS code, if $\lfloor (n-t)/2 \rfloor$ of the n received shadows are contaminated and become malign shadows, people can still utilize the RS code decoder to decode the n received shadows, locate the malign shadows, and then extract the whole secret data correctly. Two commonly-used RS code decoders are the Berlekamp-Massey decoder[73] and the Euclidean algorithm[74]. In our method, we use Euclidean algorithm[74] to locate the position of the malign shadows, or the tampered blocks, because all of the shadows are embedded in image blocks.

5.2.2 A (t, n) two-layer sharing technique modified from Chang et.al[68]

This technique can share n given data sets $\{P_i \| P_i| = c, i = 1,2,...,n\}$ to produce n shadows $\{(P_i, E_i) | i = 1,2,...,n\}$ where constant c is the constant size of each P_i. The created file E_i is

attached to P_i, where the size of each E_i is $c \times (n/t - 1)$ (the analysis is addressed later). In the original design by Chang et.al[68], the n data sets $\{P_i \| P_i| = c, i = 1,2,...,n\}$ can be decoded using any t received shadows. However, in our method, we always get all n shadows (but some of them may have been tampered). Therefore, certain steps of the algorithm in Chang et.al[68] are modified so that, for all n shadows, if the number of error shadows is not more than $\lfloor (n-t)/2 \rfloor$, then all n data sets $\{P_i \| P_i| = c, i = 1,2,...,n\}$ can be decoded by the modified decoder. Notably, in our method, the n data sets $\{P_i\}$ are the DCT coefficients which will be used to recover the tampered blocks. Moreover, the location of the error shadows are also the location of error blocks in the tampered watermarked image. The encoder (sharing) and the decoder (inverse-sharing) are shown below. Notably, in our method, the calculations of Algorithms 5.1 and 5.2 are under GF(2^{12}).

Algorithm 5.1: (t, n) two-layer sharing encoder

Input: n data sets $\{P_i | i = 1,2,...,n\}$ in which each P_i has constant size c.

Output: n shadows $\{(P_i, E_i) | i = 1,2,...,n\}$.

1. *First-layer sharing*: For each P_i, we calculate B_i (each P_i and B_i is treated as a vector, and $|P_i|=|B_i|= c$ for each i) by the following equation:

$$B_i = P_1 + 2^i P_2 + 3^i P_3 + ... + n^i P_n .$$

Then we collect and store the $n-t$ vectors $\{B_1, B_2, \ldots, B_{n-t}\}$ in file B.

2. *Second-layer sharing*: For each t digits in file B, we encode the t digits by (t, n) secret image sharing[7] (Sec. 5.2.1) to get an n-digits codeword. The n digits are dispersed, respectively, to n shadows $\{E_i | i = 1,2,...,n\}$. The construction of $\{E_i\}$ is thus done when all of the digits of file B are shared.

Algorithm 5.2: (t, n) two-layer sharing decoder

Input: n received shadows $\{(\tilde{P}_i, \tilde{E}_i) | i = 1,2,...,n\}$, but some shadows within them may have been tampered.

Output: If the number of tampered shadows is no more than $\lfloor (n-t)/2 \rfloor$, then we output the n error-corrected data sets $\{P_i | i = 1,2,...,n\}$ and the verification result.

1. Reconstruct the file B from $\{\tilde{E}_i | i=1,2,...,n\}$. Here, if the number of tampered $\tilde{E}_i$ is no more than $\lfloor (n-t)/2 \rfloor$, then the file B can be reconstructed by the RS code decoder, and we can also identify locations of the tampered shadows $\{(\tilde{P}_{\tilde{j}[i]}, \tilde{E}_{\tilde{j}[i]}) | i=1,2,...,v\}$. We can mark the location of these tampered shadows as "tampered", then output the verification result. (However, if the number of tampered $\tilde{E}_i$ is more than $\lfloor (n-t)/2 \rfloor$, the file B cannot be reconstructed, so we would stop the procedure in this case.)
2. Let the un-tampered shadows be $\{(\tilde{P}_{j[i]}, \tilde{E}_{j[i]}) | i=1,2,...,n-v\}$ where $\{j[i] | i=1,2,...,n-v\}$ are indices of the un-tampered shadows. In other words, $\{j[i] | i=1,2,...,n-v\} \cup \{\tilde{j}[i] | i=1,2,...,v\} = \{1,2,...,n\}$ and $\{j[i] | i=1,2,...,n-v\} \cap \{\tilde{j}[i] | i=1,2,...,v\} = \Phi$. The tampered data $\{\tilde{P}_{\tilde{j}[i]} | i=1,2,...,v\}$ can be recovered by the equation

$$\begin{bmatrix} \tilde{P}_{\tilde{j}[1]} \\ \tilde{P}_{\tilde{j}[2]} \\ \vdots \\ \tilde{P}_{\tilde{j}[v]} \end{bmatrix} = \begin{bmatrix} \tilde{j}[1] & \tilde{j}[2] & \cdots & \tilde{j}[v] \\ \tilde{j}[1]^2 & \tilde{j}[2]^2 & \cdots & \tilde{j}[v]^2 \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{j}[1]^v & \tilde{j}[2]^v & \cdots & \tilde{j}[v]^v \end{bmatrix}^{-1} \times \left(\begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_v \end{bmatrix} - \begin{bmatrix} j[1] & j[2] & \cdots & j[n-v] \\ j[1]^2 & j[2]^2 & \cdots & j[n-v]^2 \\ \vdots & \vdots & \ddots & \vdots \\ j[1]^v & j[2]^v & \cdots & j[n-v]^v \end{bmatrix} \begin{bmatrix} \tilde{P}_{j[1]} \\ \tilde{P}_{j[2]} \\ \vdots \\ \tilde{P}_{j[n-v]} \end{bmatrix} \right).$$

3. Output $\{P_i | i=1,2,...,n\} = \{\tilde{P}_{\tilde{j}[i]} | i=1,2,...,v\} \cup \{\tilde{P}_{j[i]} | i=1,2,...,n-v\}$.

Two size and time issues about the generated shadow E_i are discussed below.

i). Size of E_i.

We consider the two-layer sharing encoder, and in the *First-layer sharing*, since the P_i contains c digits, and the matrix in the multiplication has $(n–t)$ rows, the generated matrix $[B_1 \quad B_2 \quad \dots \quad B_{n-t}]^T$ contains $(n–t)c$ digits, which is the size of file B. In the *Second-layer sharing*, the file B is divided into $\lceil (n-t)c/t \rceil$ sectors of t digits each, and each t digits are encoded into n digits by (n, t) RS code; and then a digit is assigned to each shadow E_i. So the size of each shadow E_i is $\lceil (n-t)c/t \rceil \approx (n/t-1)c$.

ii). Running time of generating E_i.

In the *First-layer sharing*, the size of the two matrices at the equation right are $(n-t)n$ and nc, so it needs $(n–t)nc$ operations to calculate $[B_1 \quad B_2 \quad \dots \quad B_{n-t}]^T$. In the *Second-layer*

sharing, the file *B* is divided into $\lceil (n-t)c/t \rceil$ sectors of *t* digits each, and each *t* digits needs *nt* operations to generate the code, so it needs $\lceil (n-t)c/t \rceil \times nt \approx (n-t)nc$ operations. Therefore, all of the operations needed are $2(n-t)nc = \theta((n-t)nc)$.

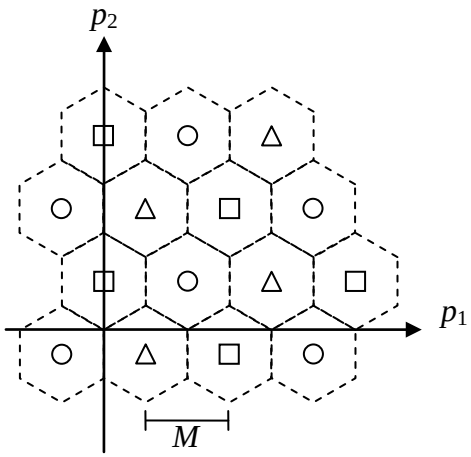


Fig. 5.1. Diagram of lattice embedding.

5.2.3 Lattice embedding[75]

Lattice embedding[75] is an embedding method in which a secret digit can be embedded into many signals. Fig. 5.1 is an example, in which a ternary value {0, 1, 2} is embedded in a pair of signals (p_1, p_2). As shown in Fig. 5.1, the space of host pair-values (p_1, p_2) is divided into many hexagonal regions, and each region corresponds to a ternary value 0, 1 or 2 (squares, circles, and triangles are used to represent the three values). As shown in Fig. 5.1, in each one of the horizontal and vertical directions, the distance of contiguous hexagonal region's centers is set to our step size *M*. If a ternary value is to be embedded in the pair (p_1, p_2), we find the nearest region's center to (p_1, p_2) so that the nearest region corresponds to the ternary value to be embedded. Then the coordinate of the region's center is output. Later, to decode the embedded value, the region which contains the stego-pair (p_1, p_2) is found, and the corresponding region-value 0, 1, or 2 of the region is output. In general, if the value of *M* is larger, then the image quality after embedding is lower, but the hidden data is more robust. The embedding and extracting algorithms are shown below.

Algorithm 5.3: Embed a ternary value in a pair of signals
Input: a ternary value *s*, a pair of host signals (p_1, p_2), and step size *M*.

Output: a pair of stego-signals (p_1', p_2').

$$\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} M^{-1} & -(\sqrt{3}M)^{-1} \\ 0 & 2(\sqrt{3}M)^{-1} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 + M/\sqrt{3} \end{bmatrix}$$

$$(lq_1, lq_2) = (\lfloor q_1 \rfloor, \lfloor q_2 \rfloor)$$

$$t = s - (lq_1 + 2lq_2) \pmod 3$$

if t=1 or 2 **then**

$$lq_t = lq_t + 1$$

else if $q_1 - lq_1 + q_2 - lq_2 > 1$

$$lq_1 = lq_1 + 1$$

$$lq_2 = lq_2 + 1$$

end if

end if

$$\begin{bmatrix} p_1' \\ p_2' \end{bmatrix} = \begin{bmatrix} M & M/2 \\ 0 & \sqrt{3}M/2 \end{bmatrix} \begin{bmatrix} lq_1 \\ lq_2 \end{bmatrix} - \begin{bmatrix} 0 \\ M/\sqrt{3} \end{bmatrix}$$

Algorithm 5.4: Extract a ternary value from a pair of signals

Input: a pair of stego-signals (p_1', p_2'), and step size M.

Output: the ternary value s hidden in (p_1', p_2').

$$\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} M^{-1} & -(\sqrt{3}M)^{-1} \\ 0 & 2(\sqrt{3}M)^{-1} \end{bmatrix} \begin{bmatrix} p_1' \\ p_2' + M/\sqrt{3} \end{bmatrix}$$

$$(lq_1, lq_2) = (\lfloor q_1 \rfloor, \lfloor q_2 \rfloor)$$

$$\begin{bmatrix} \Delta p_1' \\ \Delta p_2' \end{bmatrix} = \begin{bmatrix} M & M/2 \\ 0 & \sqrt{3}M/2 \end{bmatrix} \begin{bmatrix} lq_1 \\ lq_2 \end{bmatrix} - \begin{bmatrix} p_1' \\ p_2' + M/\sqrt{3} \end{bmatrix}$$

$$min = (\Delta p_1')^2 + (\Delta p_2')^2$$

$$b = lq_1 + 2lq_2 \pmod 3$$

if $min > (\Delta p_1' + M)^2 + (\Delta p_2')^2$ **then**

$$min = (\Delta p_1' + M)^2 + (\Delta p_2')^2$$

$$s = b + 1 \pmod 3$$

end if

if $min > (\Delta p_1' + M/2)^2 + (\Delta p_2' + \sqrt{3}M/2)^2$ **then**

$min = (\Delta p_1' + M/2)^2 + (\Delta p_2' + \sqrt{3}M/2)^2$

$s = b + 2 (\mathrm{mod}\ 3)$

end if

if $min > (\Delta p_1' + 3M/2)^2 + (\Delta p_2' + \sqrt{3}M/2)^2$ **then**

$min = (\Delta p_1' + 3M/2)^2 + (\Delta p_2' + \sqrt{3}M/2)^2$

$s = b$

end if

5.3 The proposed method

The proposed method consists of two phases. 1) Watermark generation which generates the shadows $\{E_i\}$, followed by embedding $\{E_i\}$ in the DCT domain of the host image. 2) Tampered block detection, together with the recovery achieved by a two-layer sharing decoder (then the decoded data set $\{P_i\}$ is utilized to recover the tampered blocks).

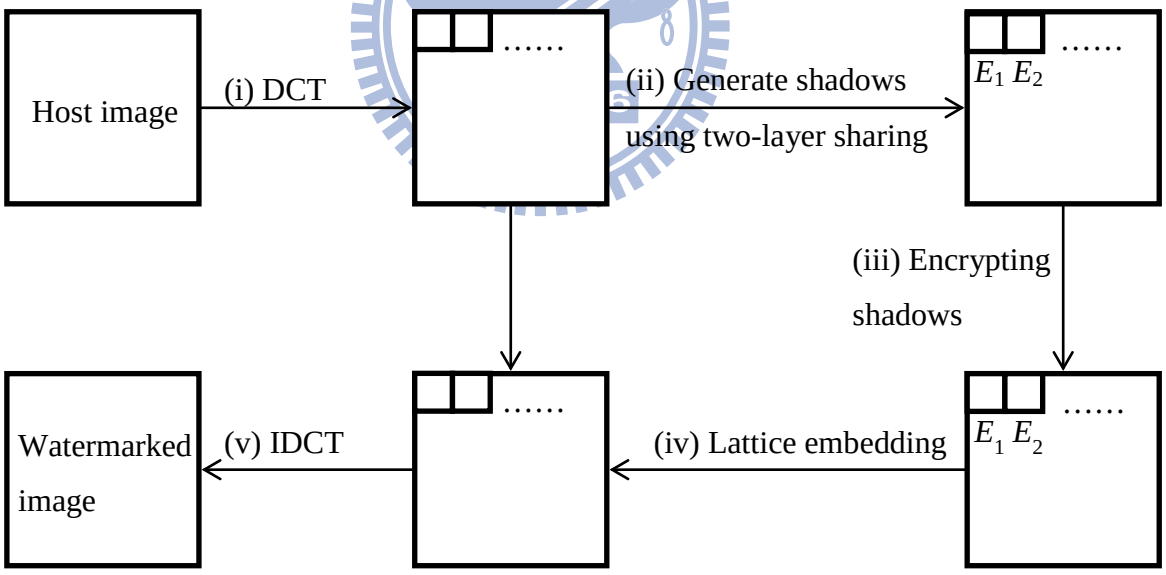


Fig. 5.2. The diagram of watermarking steps.

5.3.1 Watermark generation

Without the loss of generality, assuming that the host image is 512×512, so there are 4096 blocks of 8×8 pixels each. Then the following four steps are taken. i) Data sets $\{P_i | i = 1,2,...,4096\}$ are generated for all 4096 blocks in the host image. ii) Then these data sets

are used to generate 4096 shadows $\{E_i | i = 1,2,...,4096\}$. iii) To increase the security and protection, a 12-bit hashing code is generated for each block *i*, and then the shadow E_i is encrypted by an Exclusive-OR with the 12-bit code. iv) Finally, the (encrypted) shadow E_i is embedded in each block, and v) the DCT coefficients are converted into spatial domain to obtain the watermarked image. Fig. 5.2 shows the watermarking steps. The details of the steps i)-iv) are explained below.

d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7
d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}
d_{16}	d_{17}	d_{18}	d_{19}	d_{20}	d_{21}	d_{22}	d_{23}
d_{24}	d_{25}	d_{26}	d_{27}	d_{28}	d_{29}	d_{30}	d_{31}
d_{32}	d_{33}	d_{34}	d_{35}	d_{36}	d_{37}	d_{38}	d_{39}
d_{40}	d_{41}	d_{42}	d_{43}	d_{44}	d_{45}	d_{46}	d_{47}
d_{48}	d_{49}	d_{50}	d_{51}	d_{52}	d_{53}	d_{54}	d_{55}
d_{56}	d_{57}	d_{58}	d_{59}	d_{60}	d_{61}	d_{62}	d_{63}

Fig. 5.3. DCT coefficients which are selected as data P_i (dark gray) and embedded locations of E_i (light gray).

5.3.1.1 Generating Data sets $\{P_i | i = 1,2,...,4096\}$

Each 8×8 block of the host image is converted to DCT domain, and then four low-frequency DCT coefficients are quantized by step size *M* (the value of *M* influences the robustness of the watermarked image). Fig. 5.3 shows the positions of the four DCT coefficients $\{d_0, d_1, d_8, d_9\}$, which are painted dark gray. Then for each DC coefficient d_0 in 8×8 block, the difference value Δd_0 is calculated by subtracting the DC coefficient of the previous block from d_0. This step has two advantages. First, the contiguous blocks have similar DC values, so the bit length of d_0 can be reduced by storing the difference Δd_0. Second, when the brightness of the watermarked image is adjusted, all DC values will increase/decrease a constant value, but the difference of two DC values is unchanged, so the

integrity of difference Δd_0 is preserved after the brightness is adjusted. Finally, for each 8×8 block, the data P_i is calculated by

$$P_i = [(\Delta d_0 \times d_1^{\text{MAX}} + d_1) d_8^{\text{MAX}} + d_8] d_9^{\text{MAX}} + d_9 .$$

where d_i^{MAX} is the maximal d_i of all blocks, and the three values $\{ d_1^{\text{MAX}}, d_8^{\text{MAX}}, d_9^{\text{MAX}} \}$ are sent to the decoding side for the purpose of recovery. The size of P_i is $c = \log_2(\Delta d_0^{\text{MAX}} d_1^{\text{MAX}} d_8^{\text{MAX}} d_9^{\text{MAX}})$ which is determined by the maximal coefficients in $\{\Delta d_0, d_1, d_8, d_9\}$ among all blocks.

5.3.1.2 Generating recovery data

Generate 4096 couples $\{(P_i, E_i) \mid |P_i| = c, i = 1, \ldots, 4096\}$ by the $(t, n) = (4096(2\beta - 1), 4096)$ two-layer sharing encoder (Algorithm 5.1). Here, β is a specified threshold for the percentage of valid blocks in a tampered image. Due to embedding capacity limitation, each generated shadow E_i should have 12 bits at most. This will make $\alpha \geq 6/(c+12)$ a requirement when percentage value β is specified. The proof is given in Sec. 5.3.3. On the other hand, 100%=1>β is a natural requirement. Together, the percentage value β must satisfy $1 > \beta \geq 6/(c+12)$.

5.3.1.3 Generating the hashing code of a block

First, a 128-bit MD5 code of the block is calculated, and the formula is

MD5(P_i, i, *ID*, *Key*),

where *ID* is the image's identification number, and *Key* is a secret key. Then the generated 128-bit MD5 code is divided into 10 sectors of 12 bits each (the final 8 bits of the MD5 code are dropped), and the Exclusive-OR on the 10 sectors is used to get a 12-bit hashing code.

5.3.1.4 Embedding shadow E_i in a block

For each 8×8 block, the shadow E_i is converted into 8 ternary digits, and each digit is embedded in two DCT coefficients with lattice embedding (Algorithm 5.3). Fig. 5.3 shows all 2×8 DCT coefficients which are painted light gray. A coefficient in the middle frequency and a coefficient in the low frequency are selected to form a pair of values (p_1, p_2). Here the eight pairs of values being used for (p_1, p_2) are {(d_{12}, d_2), (d_{40}, d_{17}), (d_{19}, d_{10}), (d_4, d_{24}), (d_{33}, d_{16}), (d_{26}, d_3), (d_{11}, d_{18}), (d_{32}, d_{25})}. Then a ternary digit (0 or 1 or 2) grabbed from the shadow E_i is embedded in each (p_1, p_2) pair.

5.3.2 Tampered image verification and recovery

When a tampered watermarked image is received, the following steps can generate the verification result and the recovered image.

(1) Data sets $\{P_i | i = 1,2,...,4096\}$ are generated for all blocks in the tampered watermark image (this step is the same as Sec. 3.1.1).

(2) All shadows $\{E_i' | i = 1,2,...,4096\}$ are extracted from all blocks with Algorithm 5.4.

(3) A 12-bit hashing code is generated from each block (this step is the same as Sec. 3.1.3). Then E_i is decrypted by applying Exclusive-OR to E_i and the 12-bit code.

(4) Detect the tampered blocks of the coupled-shadows $\{(P_i', E_i') | i = 1,2,...,4096\}$ with (t, n) = $(4096(2\beta-1), 4096)$ two-layer sharing decoder (Algorithm 5.2). If the percentage of tampered blocks is less than $1-\beta$, the data sets $\{P_i | i = 1,2,...,4096\}$ are decoded. Then output the decoded data sets $\{P_i\}$ and the verification result indicating locations of the tampered blocks. (If the percentage of tampered blocks is more than $1-\beta$, the warning-message is output: "Too many blocks are tampered, and hence no recovery can be done", then the procedure should be stopped without going to step (5).)

(5) For the data sets $\{P_i | i = 1,2,...,4096\}$, decode the four values $\{\Delta d_0, d_1, d_8, d_9\}$ by division. The step needs the three values $\{d_1^{MAX}, d_8^{MAX}, d_9^{MAX}\}$. Then the DC value d_0 is obtained by calculating the sum of Δd_0 and the DC coefficient of the previous block.

(6) In all tampered blocks, the four DCT coefficients $\{d_0, d_1, d_8, d_9\}$ should be replaced with the de-quantized values in P_i.

(7) For each block which passes the authentication tests, if its DCT coefficient d_0 is too far away from the de-quantized value d_0 in P_i (up to a threshold), then still replace the DCT coefficient d_0 by the de-quantized value d_0 extracted from P_i; this is to recover back the whole image after global adjustment of brightness.

(8) The DCT coefficients in each block should be converted to the spatial domain.

5.3.3 The value of α

The value of β used in Sec. 5.3.1 is discussed here. In Sec. 5.3.1.2, we set (t, n) = $(4096(2\beta-1), 4096)$ where 4096 is the number of 8×8 blocks in a 512×512 image. Due to the use of the RS code, which can correct $\lfloor (n-t)/2 \rfloor$ error shadows in all n received shadows in

general applications, the tolerable percentage of tampered blocks equals

$$\frac{\left\lfloor \frac{n-t}{2} \right\rfloor}{n} = \frac{\left\lfloor \frac{4096-4096(2\beta-1)}{2} \right\rfloor}{4096} \approx \frac{\frac{4096-4096(2\beta-1)}{2}}{4096} = 1-\beta .$$

In other words, if the percentage of tampered blocks exceeds $1-\beta$, then our method loses the ability to recover tampered blocks. Analogously, in a tampered image, β is the minimal percentage of valid blocks needed to keep the recovery ability active. In Sec. 5.3.1, the size of each P_i is c. On the other hand, Sec. 5.3.1.4 states that shadow E_i is converted into 8 ternary digits. Hence, each E_i has $|E_i| = \lg(3^8) = 8\lg 3 \geq 12$ bits. So, by the equation $|E_i| \approx (n/t-1)c$ of Sec. 2.2, it can be derived that $12 \leq |E_i| \leq c \times \left(\frac{4096}{4096(2\beta-1)} - 1 \right)$. Hence, $\beta \geq \frac{6}{c+12}$. Notably, $\beta<1$ is required for a recovery system to be meaningful ($\beta=1$ would require all blocks to be valid blocks, which means that the recovery system is too weak to tolerate even just one block being altered). Having combined the two inequalities, a more integrated inequity should be

$$1 > \beta \geq 6/(c+12).$$

5.4 Experimental results

In this section, some experiments are undertaken to check the performance of our watermarked images.

5.4.1 Robustness test

With the step value M being 20, four watermarked images {Lena, Peppers, Jet, Scenery} are generated and shown in Fig. 5.4(a-d). From (a) to (d), the PSNR values are 34.75 dB, 34.70 dB, 34.80 dB, and 34.65 dB, respectively. Then a cropping attack is applied to the four watermarked images (we crop the central 192×192 pixels of each 512×512 image). Moreover, some further distortion is made to the cropped images, as illustrated below. The cropped-Lena shown in (e) is compressed by a JPEG with QF=65 and the compression ratio is 8.57. The brightness of the cropped-Peppers shown in (f) is increased by 30; Gaussian noises (σ^2=6) are added to the cropped-Jet shown in (g); an 8-pixels-wide “white” horizontal bar is inserted to the cropped-Scenery image shown in (h). Fig. 5.4(i-l) is the verification result of the four tampered images. White blocks are the places which are marked as “tampered” blocks. (m-p)

are the four recovered images, and (q-t) are the close-up versions of the recovered images. The PSNR values of four recovered images are, from left to right, 30.16 dB, 30.96 dB, 29.83 dB, and 31.75 dB, respectively. More experiments for various values of *M* are shown in Table 5.1. In each row, column 1 is the value of the user-specified step value *M*; and column 2 is the PSNR of the corresponding watermarked image. The remaining columns are for different kinds of attacks. Column 3 is the allowed tampered ratio of watermarked image; i.e. if the tampered ratio is larger than the specified value, our method cannot recover the tampered region. Column 4 is the tolerable bound of the qualify factor (QF) of JPEG compression; i.e., if the compression uses a QF below this bound, our method cannot recover the tampered region. Column 5 is for Gaussian noise, it shows the maximal tolerable variance σ^2 of Gaussian noise. Column 6 is the tolerable range of brightness adjustment. In summary, if the attack uses a parameter value worse than the threshold values specified in Tables 5.1, then our method cannot recover the tampered region. Should this happen, switching to a larger value of *M* in advance is necessary. (In general, using a larger value of *M* in advance can increase the tolerable range of attack; but the watermarked image's quality degenerates.) From Table 5.1, we see that our method can resist certain levels of area-tampering (e.g. cropping or replacement of some areas of image), JPEG compression, Gaussian noise, and brightness adjustment.

5.4.2 Security test

Three kinds of attack are tested on our watermarked images, which are cut-and-paste attack[76], collage attack[77], and VQ attack[78].

Firstly, the so-called cut-and-paste attack [76] is tested. Fig. 5.5(a) is our watermarked image and the PSNR value is 48.13 dB. Then a small-size pepper is copied and pasted to the lower-left corner as shown in (b); (c) is the verification result, and (d) is the recovered image, the PSNR value of which is 40.54 dB. Secondly, the so-called collage attack [77] is tested. Fig. 5.6(a-b) shows our two watermarked images, Boat and House, and the PSNR values are 34.63 dB and 34.76 dB, respectively. Then the car in (b) is copied-and-pasted to the same place in (a). This yields the image shown in (c). (d) is the verification result, and (e) is the recovered image, the PSNR value of which is 31.88 dB. Finally, the so-called VQ (Vector Quantization) attack is tested[78]. Twelve aerial images are downloaded from the USC-SIPI Image Database[79]. Each image is 512×512, and is converted to a grayscale image. Then one of them is assigned as the test image (the one shown in Fig. 5.7(a)); and the remaining eleven

images are watermarked by the proposed method (using M=4). Then, since our method is based on 8×8 blocks, the eleven watermarked images are partitioned to get blocks of 8×8 each. These blocks are collected together and treated as a VQ codebook with many code words. With this VQ codebook, the VQ-compression-decompression version of Fig. 5.7(a) is shown in (b). (c) is the verified result which shows that the whole (b) is a fake.

5.4.3 Image quality and our advantage

Firstly, the method of Varsaki et al.[72] was implemented, and the results are shown in Fig. 5.8. Fig. 5.8(a) is the 512×512 watermarked color image Lena which is 40.88 dB. As shown in (b) of the figure, the embedded recovery data is the size-reduced 128×128 *gray-level* version of the *rotated* host, the clockwise rotation is 180 degree, as suggested by Varsaki et al.[72]. This 128×128 *gray-level* rotated version is embedded in the 128×128 blocks of the host image, and each block is 4×4. So, the recovery data of the rightmost bottom 4×4 block is embedded in the leftmost top 4×4 block, and the recovery data of the Southwest quadrant is embedded in the Northeast quadrant, and so on. Unfortunately, when the central 192×192 pixels of the watermarked Lena are cropped (the tampered image is shown in (c)), the recovery data extracted from the non-cropped area is as shown in (d). It can be seen that the tampered region still cannot be recovered because the recovery data of the central 192×192-pixels box was embedded earlier in the box itself. In other words, the recovery data of the cropped box is also cropped. This is very different from ours. As shown in Fig. 5.4, when the central 192×192 pixels of our watermarked Lena are cropped, the cropped region can still be recovered. This should come as no surprise because, according to Table 5.1, when M=20, a moderate-size-area's tampering can be tolerated (up to 16.7% of the whole image's blocks can be tampered).

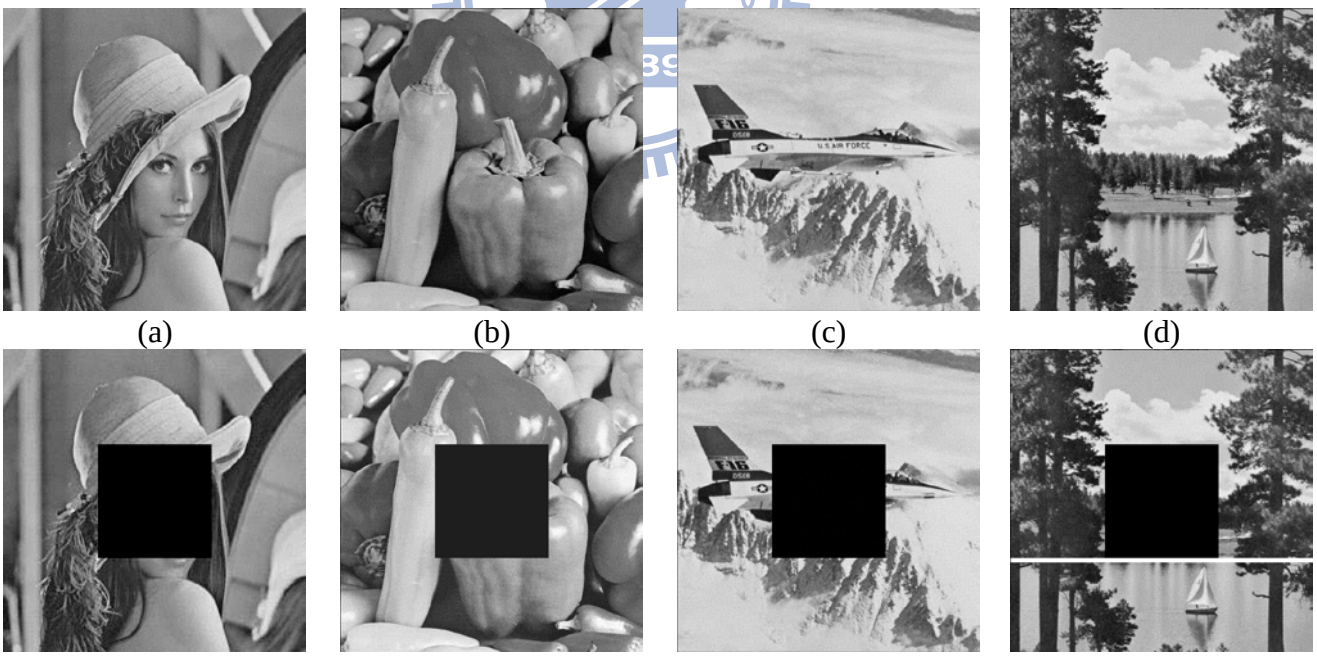
Next, because Tsai and Chien's method[16] used scaled versions of the originals as messages, then embedded the messages in the frequency domain, and also because they provided experimental results about the resistance to JPEG compression and Gaussian noise, we compare their method with ours.

Fig. 5.9 shows the experiment. The results of Tsai and Chien[16] are shown in (a-d), and ours are shown in (e-h). Notably, the PSNR value of their watermarked image Jet in (a) is 30.8 dB while ours in (e) is 32.29 dB. The PSNR value of the recovered image is 29.3 dB (Fig. 5.9(d)) for theirs, and 31.89dB (Fig. 5.9(h)) for ours. Notably, (d′) and (h′) show the details of (d) and (h), respectively. It is observed that, between the lower-middle and lower-left of the

image, there are some artifacts in their recovered snow area below the mountains, as shown in (d′). Therefore, our recovered image is better.

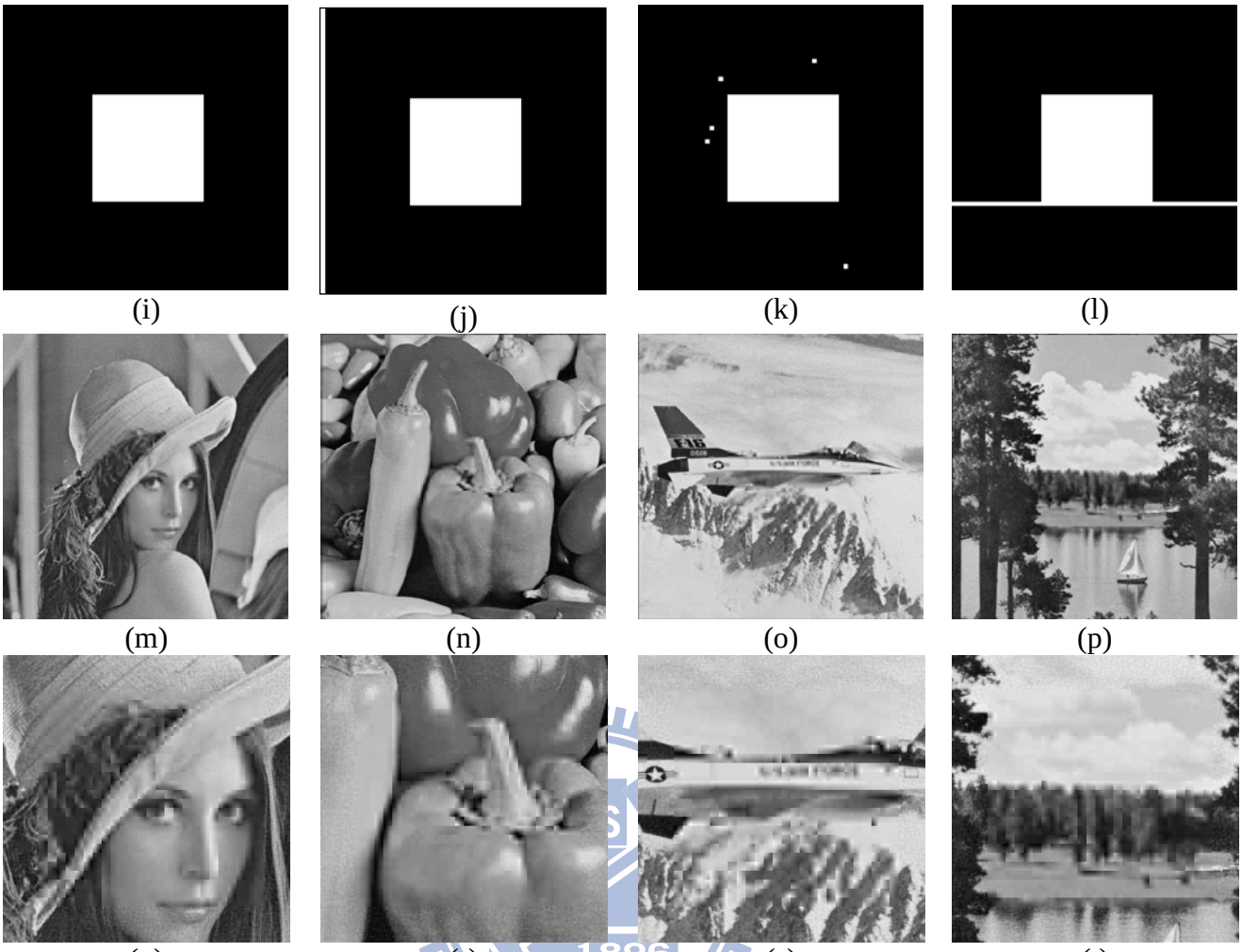
Fig. 5.10 shows the second experiment. The watermarked image Peppers are under both tampered attack and JPEG compression. The results of Tsai and Chien's[16] method are shown in (a-d). Ours are shown in (e-h). (a) is their 30.6 dB watermarked image. (b) is their tampered image (inserting a sub-image compressed-decompressed by JPEG (QF=80, and the compression ratio is 4.3). (e) is our 32.24 dB watermarked image. (f) is our tampered image (inserting a sub-image, then use JPEG (QF=80, and the compression ratio is 5.4) to compress/decompress the mixed image). Having compared the two recovered images (d) and (h), it can be seen that ours (Fig. 5.10(h)) has better visual quality (because Fig. 5.10(d) has some noisy dots). Details are shown in (d′) and (h′).

Fig. 5.11 shows the third experiment. The watermarked image Peppers is tampered with; and then the damaged image is attacked by adding Gaussian noises. The results of Tsai and Chien[16] are shown in (a-d), and ours are shown in (e-h). Having compared the two recovered images (d) and (h), again, it can be seen that ours (Fig. 5.11(h)) still has a better visual quality (because Fig. 5.11(d) has some noisy dots). Details are shown in (d′) and (h′).



(a) (b) (c) (d)

(e) (f) (g) (h)

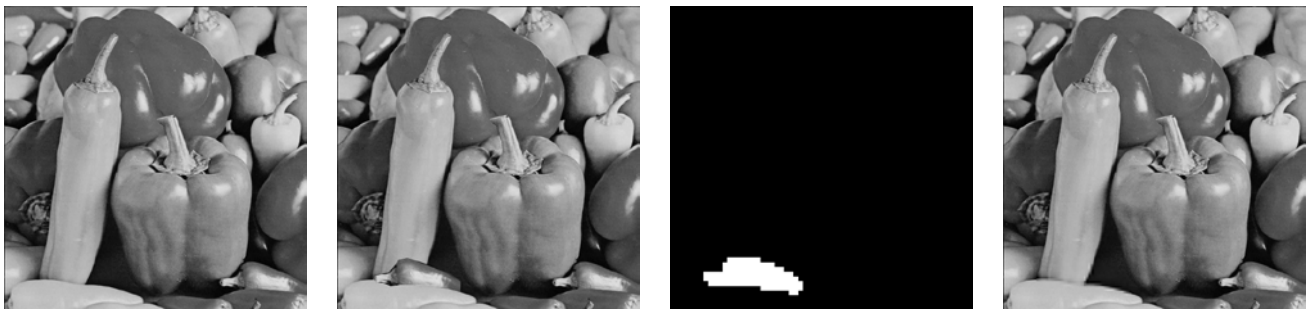


(i) (j) (k) (l)

(m) (n) (o) (p)

(q) (r) (s) (t)

Fig. 5.4. Robustness test of the proposed method. (a-d): Our four watermarked images Lena, Peppers, Jet, and Scenery. (e-f): The four cropped images. (i-l): The corresponding verification results, after doing a JPEG compression on (e), adjusting brightness of (f), adding noise to (g), and adding white bar to (h). (m-p): The recovered images. (q-t): Close-up versions around the recover area of the recovered images (m-p).



(a) (b) (c) (d)

Fig. 5.5. Cut-and-paste attack. (a): Watermarked image, (b):Tampered image, (c):Verification result, (d): Recovered image.

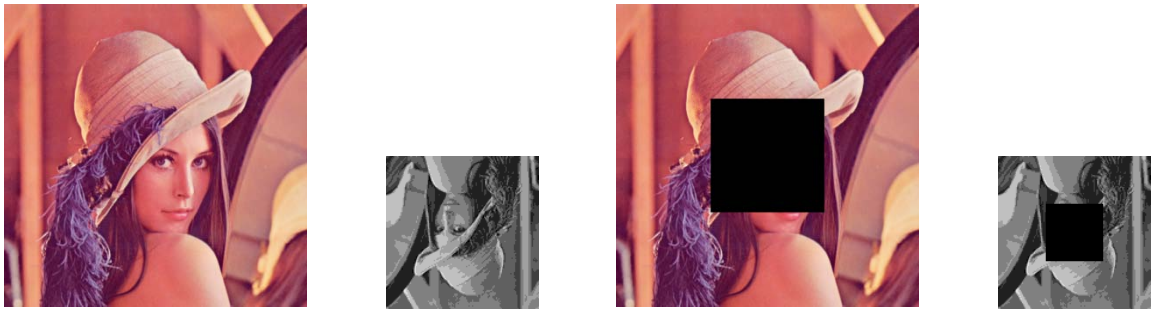
(a) (b) (c) (d)

(e)

Fig. 5.6. Collage attack. (a): First watermarked image Boat, (b): Second watermarked image House, (c):Collaged image in which the car in (b) is copied-and-pasted to the same place as (a), (d): Verification result, (e): Recovered image.

(a) (b) (c)

Fig. 5.7. Vector quantization (VQ) attack. (a): Original image, (b): VQ-attack result of (a), (c): Verification result indicates that the *whole* image (b) is fake everywhere.



(a) (b) (c) (d)

Fig. 5.8. Cropping test for Varsaki et al.'s[72] method. (a): Watermarked image Lena, (b): Recovery data embedded in (a), (c): When (a) is cropped, (d): Recovery data extracted from the support of the non-cropped area.

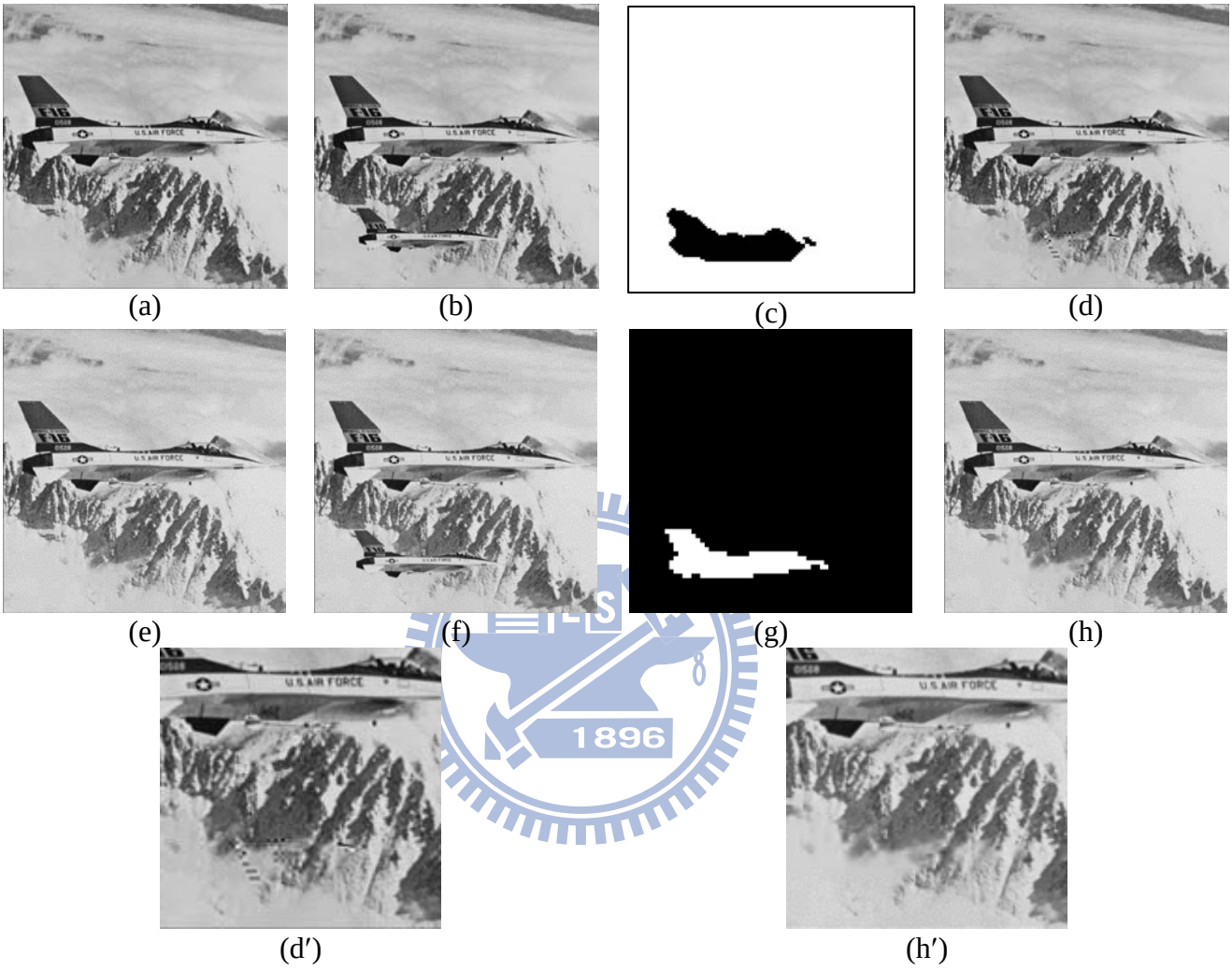


Fig. 5.9. An experiment to compare our method with that of Tsai and Chien[16]. (a): Their 30.8 dB watermarked image Jet, (b): Their tampered image, (c): Their verification result, (d): Their 29.3 dB recovered image, (e): Our 32.29 dB watermarked image, (f): Our tampered image, (g): Our verification result, (h): Our 31.89dB recovered image, (Notably, (d′) and (h′) show the details of (d) and (h) respectively. There are some artifacts in (d′) on the recovered snow.)

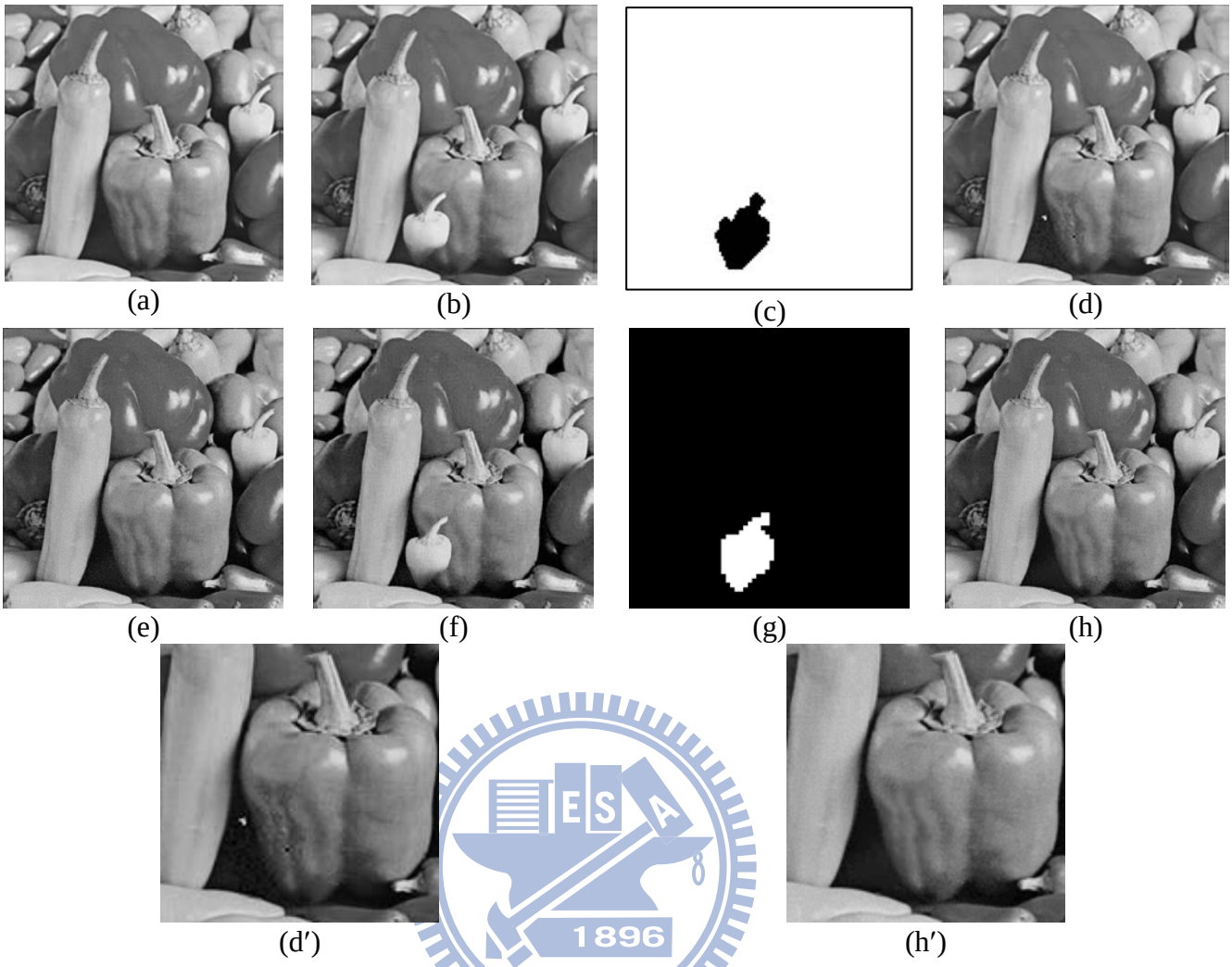
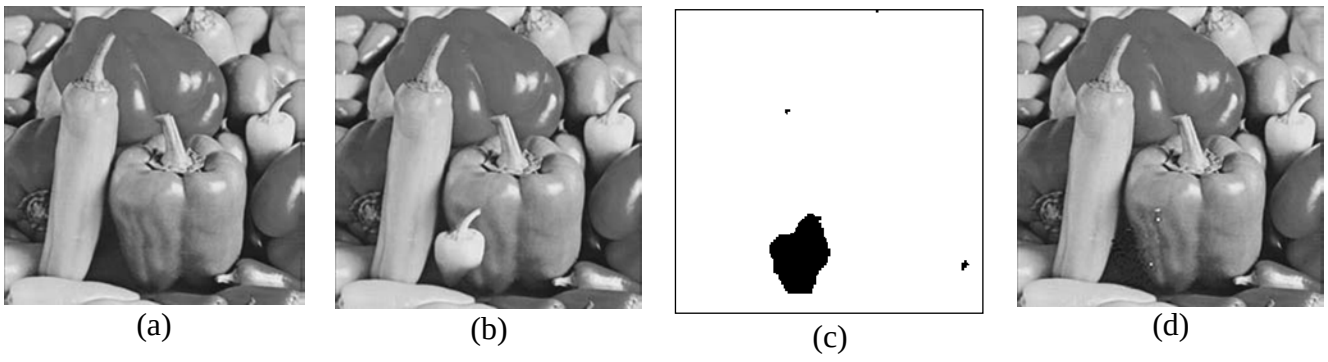
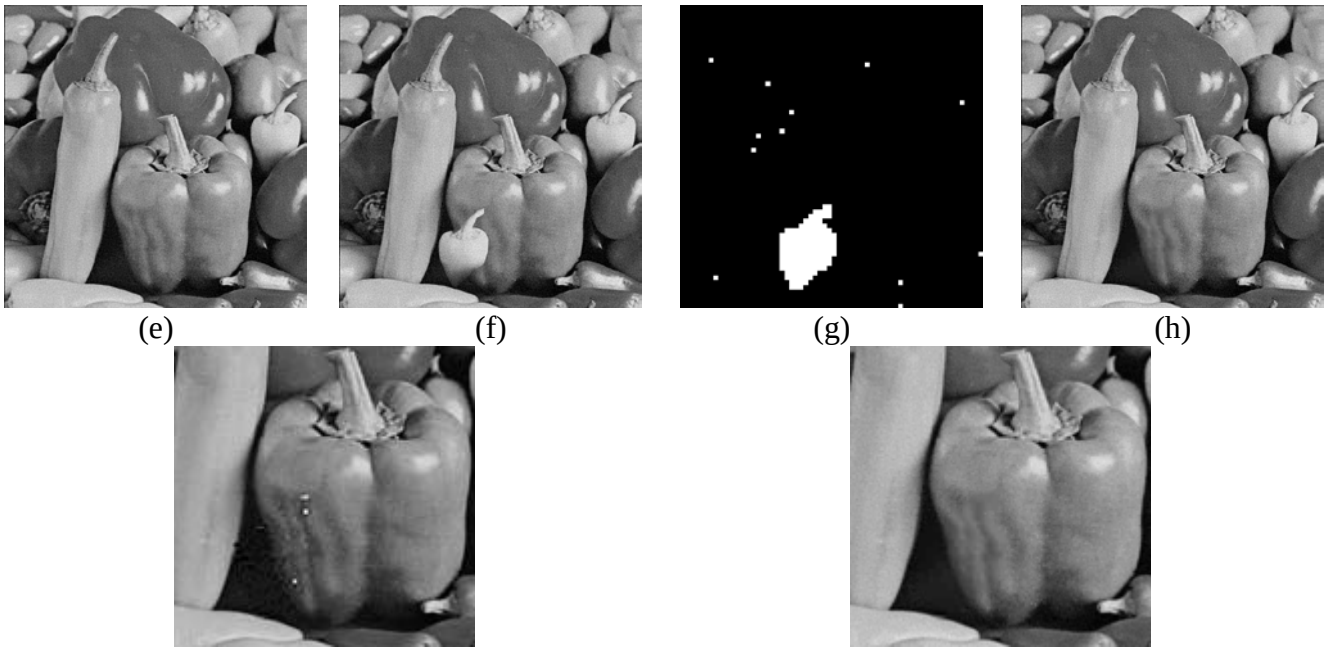


Fig. 5.10. Second experiment to compare our method with that of Tsai and Chien[16]. (a): Their 30.6 dB watermarked image Peppers, (b): Tampering with (a), followed by JPEG compression with QF=80, (c): Their verification result, (d): Their recovered image, (e): Our 32.24 dB watermarked image Peppers, (f): Tampering with (e), followed by a JPEG compression with QF=80, (g): Our verification result, (h): Our recovered image. (Notably, (d′) and (h′) show the details of (d) and (h), respectively.)





(e) (f) (g) (h)

(d′) (h′)

Fig. 5.11. The other experiment to compare our method with that of Tsai and Chien[16]. (a): Their 30.6 dB watermarked image Peppers, (b): Tampering with (a), followed by adding Gaussian noises with σ^2=12, (c): Their verification result, (d): Their recovered image, (e): Our 32.24 dB watermarked image Peppers, (f): Tampering with (e), followed by adding Gaussian noises with σ^2=12, (g): Our verification result, (h): Our recovered image. (Notably, (d′) and (h′) show the details of (d) and (h), respectively.).

Table 5.1. PSNR quality of watermarked image and attack-tolerance (for various quantization step value *M*). The host images are Lena (L), Peppers (P), Jet (J), and Scenery (S).

Watermarked image		**Attack-tolerance**				
Quantiz. step value (*M*) used in our algorithm	PSNR (dB) of watermarked image	$(1-\beta)$, i.e. percentage of area can be cropped or replaced	JPEG with Quality Factor (*QF*) not less than thresholds shown here	Gaussian noise (σ^2)	Range of "brightness adjustment"	
					Lena; Pepper	Jet; Scene
2	53.37-53.40	1/8=12.5%	100	0	[-35,40]; [-10,40]	[-55,30]; [-20,25]
4	48.13-48.15	1/8	94	0	[-40,50]; [-15,50]	[-70,35]; [-30,30]

6	44.83-44.85	1/8	90	0	[-40,50]; [-15,55]	[-70,35]; [-30,30]
8	42.40-42.48	1/8	86	0-1 (0 for J; 1 for L&P&S)	[-40,50]; [-15,55]	[-70,35]; [-30,30]
10	40.52-40.54	1/8	82	2-3 (2 for J&P&S; 3 for L)	[-40,50]; [-15,55]	[-70,35]; [-30,30]
12	38.95-39.01	1/8	78	4	[-40,50]; [-20,55]	[-75,35]; [-30,30]
14	37.67-37.72	1/8	75	5-6 (5 for L&S; 6 for J&P)	[-40,55]; [-20,55]	[-75,35]; [-30,30]
16	36.56-36.61	1/8	71	7	[-40,55]; [-20,55]	[-75,35]; [-30,30]
18	35.55-35.62	1/8	67	10	[-40,55]; [-25,55]	[-75,35]; [-30,30]
20	34.63-34.80	1/6=16.7%	63	13	[-45,60]; [-25,60]	[-85,35]; [-35,35]
22	33.84-33.97	1/6	59	16	[-45,60]; [-25,60]	[-85,35]; [-35,35]
24	33.10-33.24	1/6	55	19	[-45,60]; [-25,60]	[-85,40]; [-40,40]
26	32.43-32.60	1/6	51	22	[-45,60]; [-25,65]	[-85,40]; [-40,40]
28	31.80-32.00	1/6	48	26	[-45,60]; [-25,65]	[-85,40]; [-40,40]
30	31.24-31.42	1/6	45	30	[-45,65]; [-25,65]	[-85,40]; [-40,45]
32	30.70-30.87	1/6	42	34	[-45,65]; [-25,65]	[-85,40]; [-40,45]

5.5 Comparison with other studies

In this section, the proposed method is compared with other studies. Firstly, the two studies[11, 12] are authentication methods without considering the issue of recovery, but ours is equipped with both authentication and recovery abilities. As for other semi-fragile watermarking methods[13-17] with both authentication and recovery abilities, to describe the difference between ours and those methods, each watermarking algorithm is divided into major sub-steps (from the perspective of methodology and system design), and then a comparison is made. The differences are described below.

a). Embedding location (i.e. where to embed?) and our advantage

In methods[13-15], the recovery data of each block is embedded in another block. For example, the recovery data of block A_0 is embedded in block A_1, the recovery data of block A_1 is embedded in block A_2, and so on. In the verification and recovery phase, if block A_0 is judged as "tampered", then the recovery data in block A_1 is extracted to recover block A_0, and so on. In this example, if blocks A_0 and A_1 are both tampered, then block A_0 cannot be recovered. In Tsai and Chien's method[16, 17], although the processing domain is the discrete wavelet domain, the recovery data in low-frequency bands still needs to find some other location in high-frequency domain to undertake embedding. Therefore, if both locations are attacked, a similar recovery-disabled problem exists, although it is less severe.

However, in our method, as long as the number of valid blocks reaches a threshold, our inverse operation of the two-layer sharing can always decode the recovery data, so there is no need to consider the case that a block A_0 and the block A_1 storing its recovery data are simultaneously tampered. We only have to consider the percentage of the damaged area occupied in the whole image. As long as the damaged blocks occupy less than, say, 1/6=16.7% of the whole image's blocks, recovery can always be undertaken. In general, it is hard to predict in advance which part would be tampered. There is no way to predict the trace of tampering, and worrying about "the percentage of blocks (in the whole image) being tampered" is simpler than worrying about "how to predict the actual location of the tampered area".

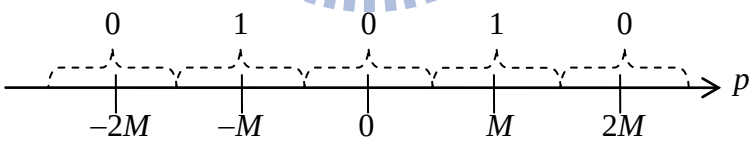


Fig.5.12. Diagram of the 1-deimensional parity-check quantization used in many research works.

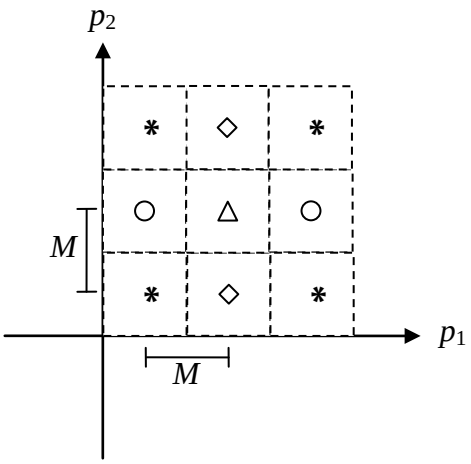


Fig. 5.13. Diagram to explain a two-dimensional case of parity-check quantization. Here, two host pixel values (p_1, p_2) are replaced by one of the centers for the purpose of embedding a two-bit data.

b). Embedding method (i.e. how to embed?) and our advantage

Parity-check quantization (or similar works) is used in a great many research works[13, 14, 16, 17]. (As for Ref. [15], it embeds the data in Least Significant Bits (LSB) of the host image. Notably, 1-bit LSB embeds one bit per pixel, so two bits are embedded in two pixels, and the largest distortion for a pair of stego-pixels is $\sqrt{2} \approx 1.414$, but in our lattice embedding, when M=2 in Fig.5.1, our largest distortion for two host pixels is $M/\sqrt{3} = 2/\sqrt{3} \approx 1.155$; hence smaller.) As shown in Fig. 5.12, in 1-dimensional parity-check quantization, the host values are divided into many regions like [–3M/2, –M/2), [–M/2, M/2), [M/2, 3M/2), [3M/2, 5M/2), and so on, with M is called the quantization level or step size. Each region corresponds to a binary value 0 or 1. If a bit is to be embedded in a host value p, then just find the nearest region of p so that the nearest region corresponds to the bit value to be embedded. Then output the central coordinate (0, or ±M, or ±2M, or …) of the picked region as the stego-value that replaces p. To extract the hidden secret bit, just locate the region which contains the stego-value, then output the corresponding bit value 0 or 1. Our method is different, since we use lattice embedding as the embedding method. As shown in Fig. 5.1, the space of the host pair-values (p_1, p_2) is divided into many hexagonal regions, and the center of each region corresponds to a ternary value 0, 1 or 2. (In Fig. 5.1, small rectangles, triangles, and circles are used to represent the three values, respectively.) If a ternary value is to be embedded in the host pair (p_1, p_2), the nearest hexagon-center is found (i.e. small rectangle,

triangle, or circle) which corresponds to the ternary value to be embedded. Then the 2-dimensional coordinate of the hexagon-center is output. Later, to decode the embedded value, just locate the hexagonal region which contains the stego-pair (p_1, p_2), then output the corresponding value 0, 1, or 2 of the region. The major reason for using lattice embedding is that the distortion of the embedding is smaller, because the hexagon-centers are denser in the plane than the square-centers. To see this, first let us inspect Fig. 5.13, which explains the two-dimensional case of parity-check quantization, i.e. it explains what happens when two pixels (p_1, p_2) are modified by parity-check quantization in order to embed two bits (00=0, 01=1, 10=2, or 11=3) of the data. According to the location of (p_1, p_2), the nearest square-center, whose class-reading in {0, 1, 2, 3} must coincide with the given two-bit data, is picked and the value of (p_1, p_2) is replaced by the coordinate of the square-center. The quantization step size M in Fig. 5.13 is the same as in Fig. 5.1. Having compared Figs. 5.1 and 5.12, it can be seen that the distance between the hexagon-centers is smaller than the distance between the square-centers (each hexagon can be contained by a square box of size M-by-M). Therefore, when two host pixel values (p_1, p_2) are modified to embed the data, the distortion of the Lattice embedding is smaller. (As shown in Figs. 5.9-5.11, our watermarked images have a better image quality.)

Of course, when Fig. 5.1 is adopted to replace Fig. 5.13, the price is that the data embedded in a hexagon can only have a data value of 0, 1 or 2, but not 3. In other words, we sacrifice the size of the embedding to get a smaller distortion. However, this difficulty is overcome because a sharing technique is used in the proposed method to reduce the amount of data to be embedded, along with the second benefit of increasing the recovery ability from scattered large–area tampering. In general, the size of each share is only a small portion of the original size of the data. (As shown in Figs. 5.9-5.11, although our watermarked images have a better image quality, our recovery ability is still very competitive.)

5.6 Conclusions

This chapter proposes an authentication-recovery method. The watermarked image can be moderately altered by doing a JPEG compression, adding a Gaussian noise, or adjusting the brightness. Certain security tests, such as a cut-and-paste attack, a collage attack, and a VQ attack, are also tested. In our design, the recovery data is embedded in DCT coefficients using lattice embedding to reduce distortion. The recovery data is dispersed into many blocks by two-layer sharing. Compared with previously reported methods, our specialty is that the

tampered region can be recovered as long as the percentage of the tampered blocks does not exceed a pre-defined threshold, say, 16.66%. Notably, as stated in Part (a) of Sec. 5.5, it is hard to predict in advance which part of a watermarked image will be cropped or replaced by attackers. The traditional mapping-sequence strategy for finding locations to hide recovery data is not a suitable strategy. This dilemma is avoided in the proposed method by using sharing. After all, worrying about “the percentage of blocks being tampered” is simpler than worrying about “how to predict the actual locations of tampered area”.

Chapter 6
Conclusions and Future works

6.1 Conclusions

In this dissertation, some technologies are proposed to protect digital images. The technologies are Flip Visual Cryptography (Ch. 2), weighted secret image sharing (Ch. 3), data hiding (Ch. 4), and semi-fragile watermarking (Ch. 5).

In Chapter 2, Opaque-oriented FVC and non-opaque-oriented FVC schemes were introduced. We proved that both schemes satisfy perfect security and they are conditionally optimal in contrast. The generated transparencies do not lead to any expansion in size. The experimental results show the revealing of double secrets via flipping and stacking the transparencies together. Due to the double secrets feature of the proposed method, one of the applications is the double checking of ownership for personality identification. Since the size is non-expanded, the space needed to carry a transparency to a meeting is economical (the size is the same as the space needed to carry an original image).

In Chapter 3, a fast weighted secret image sharing method with a (*t*, *n*) threshold was proposed. This method shares the secret image among weighted participants, and the secret image can be losslessly recovered if the sum of the weights of the participants is greater than or equal to the threshold *t*. Additionally, the execution time in the weighted secret image sharing phase is improved by using properties of GF(2^r). As shown in Fig. 2.5, our execution time is better than that of Thien and Lin when $w_i > 1$. The executives of a company can use our method to share secret images.

In Chapter 4, an embedding method based on a weighted sum function was proposed. As shown in our figures and tables, this method has a wide range of embedding rates (0.5–4.0 bpp), and has a competitive PSNR over the entire range. The predicted PSNR values ($PSNR_{est}$ by Eq. (3.14)) are also extremely close to the actual PSNR values. Therefore, embedding errors can be predicted even before the actual embedding. With this PSNR-prediction property (Table 3.2), for each secret data the customer gives us, we can determine the necessary size of a host image if the customer also specifies the minimal PSNR value he can tolerate. This determines a set of host images for that secret data. Sec. 3.4 proved that

Modulus-based method [8] and LSB matching methods [9, 10] are special cases for us. The worst-case PSNR discussed in Item V of Sec. 3.5 also shows that, even if some very strange data (data artificially made by picky users and quite unnatural) was to be embedded, our method is still competitive with others.

In Chapter 5, a semi-fragile method with recovery ability was proposed. The watermarked image can be moderately altered by JPEG compression, adding Gaussian noise, or adjusting the brightness. Certain security tests, such as a cut-and-paste attack, a collage attack, and a VQ attack, were also tested. In our method, the recovery data is embedded in DCT coefficients using lattice embedding to reduce distortion. The recovery data is dispersed into many blocks by two-layer sharing. The defining characteristic of our method is that unlike previously reported methods, tampered regions can be recovered as long as the percentage of the tampered blocks does not exceed a pre-defined threshold, say, 16.66%.

6.2 Future works

Based on the proposed methodologies in this dissertation, some further works can be studied.

1. Visual Cryptography with multiple secrets is an interesting study issue (e.g. circular VC methods [3, 19]). Based on the method proposed in Chapter 2, in the future we plan to design a circular VC method for multiple secrets (the number of secrets can be larger than 2), using perfect security (and optimal contrast, if possible).
2. A fast sharing algorithm under GF(2^k) is proposed in Chapter 3. However, in our method, the calculation in the decoding uses an extended Lagrange polynomial equation (2.8), which involves matrix multiplication. Therefore, it needs $\Theta(t)$ to decode a secret digit a_i. Creation of a fast decoding algorithm is one of our future works.
3. The method proposed in Chapter 5 is processed under the DCT domain. In recent years, wavelet transform has been widely used in image compression. In the future, we plan to design a semi-fragile method in the wavelet domain.

References

[1] M. Naor and A. Shamir, "Visual Cryptography," in *Proceedings of Advances in Cryptology — EUROCRYPT'94*, Perugia, Italy, vol. 950, pp. 1-12, May 1994.

[2] C. N. Yang, "New visual secret sharing schemes using probabilistic method," *Pattern Recognition Letters*, vol. 25, no. 4, pp. 481-494, 2004.

[3] S. H. Shyu, "Image encryption by random grids," *Pattern Recognition*, vol. 40, no. 3, pp. 1014-1031, 2007.

[4] A. Shamir, "How to Share a Secret," *Communications of the Acm*, vol. 22, no. 11, pp. 612-613, 1979.

[5] G. R. Blakley, "Safeguarding cryptographic keys," in *Proceedings of the National Computer Conference*, New York, vol. 48, pp. 313-317, 1979.

[6] M. O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," *Journal of the ACM*, vol. 36, no. 2, pp. 335-348, 1989.

[7] C. C. Thien and J. C. Lin, "Secret image sharing," *Computers & Graphics*, vol. 26, no. 5, pp. 766-770, 2002.

[8] C. C. Thien and J. C. Lin, "A simple and high-hiding capacity method for hiding digit-by-digit data in images based on modulus function," *Pattern Recognition*, vol. 36, no. 12, pp. 2875-2881, 2003.

[9] X. L. Li, B. Yang, D. F. Cheng, and T. Y. Zeng, "A Generalization of LSB Matching," *IEEE Signal Processing Letters*, vol. 16, no. 1-3, pp. 69-72, 2009.

[10] J. Mielikainen, "LSB matching revisited," *IEEE Signal Processing Letters*, vol. 13, no. 5, pp. 285-287, 2006.

[11] C. K. Ho and C. T. Li, "Semi-Fragile Watermarking Scheme for Authentication of JPEG Images," in *Proceedings of the International Conference on Information Technology: Coding and Computing*, Las Vegas, Nevada, vol. 1, pp. 7-11, April 2004.

[12] C. H. Lin, T. S. Su, and W. S. Hsieh, "Semi-Fragile Watermarking Scheme for Authentication of JPEG Images," *Tamkang Journal of Science and Engineering*, vol. 10, no. 1, pp. 57-66, 2007.

[13] C. Y. Lin and S. F. Chang, "Semi-fragile watermarking for authenticating JPEG visual content," in *SPIE International Conference on Security and Watermarking of Multimedia Contents II*, San Jose CA, USA, vol. 3971, pp. 140-151, Jan 2000.

[14] S. L. Hsieh, P. D. Wu, I. J. Tsai, and B. Y. Huang, "A Recoverable Semi-fragile Watermarking Scheme Using Cosine Transform and Adaptive Median Filter," in *Proceedings of the 5th international conference on Autonomic and Trusted Computing*, Oslo, Norway, pp. 629-640, Jun 2008.

[15] X. Jiang and Q. Liu, "Semi-fragile watermarking algorithm for image tampers localization and recovery," *Journal of Electronics*, no. pp. 343-51, 2008.

[16] M. J. Tsai and C. C. Chien, "Authentication and recovery for wavelet-based semifragile watermarking," *Optical Engineering*, vol. 47, no. 6, p. 067005, 2008.

[17] M. J. Tsai and C. C. Chien, "A wavelet-based semi-fragile watermarking with recovery mechanism," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Seattle, WA, vol. 1-10, pp. 3033-3036, May 2008.

[18] S. K. Chen and S. J. Lin, "Non-expansible Flip-flop Visual Cryptography with Perfect Security," in *Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Kyoto, pp. 949-952, Sep 2009.

[19] H. C. Wu and C. C. Chang, "Sharing visual multi-secrets using circle shares," *Computer Standards & Interfaces*, vol. 28, no. 1, pp. 123-135, 2005.
[20] W. P. Fang and J. C. Lin, "Visual cryptography with extra ability of hiding confidential data," *Journal of Electronic Imaging*, vol. 15, no. 2, p. 023020, 2006.
[21] S. J. Shyu, S. Y. Huang, Y. K. Lee, R. Z. Wang, and K. Chen, "Sharing multiple secrets in visual cryptography," *Pattern Recognition*, vol. 40, no. 12, pp. 3633-3651, 2007.
[22] A. De Bonis and A. De Santis, "Randomness in secret sharing and visual cryptography schemes," *Theoretical Computer Science*, vol. 314, no. 3, pp. 351-374, 2004.
[23] T. H. Chen and K. H. Tsao, "Visual secret sharing by random grids revisited," *Pattern Recognition*, vol. 42, no. 9, pp. 2203-2217, 2009.
[24] Y. F. Chen, Y. K. Chan, C. C. Huang, M. H. Tsai, and Y. P. Chu, "A multiple-level visual secret-sharing scheme without image size expansion," *Information Sciences*, vol. 177, no. 21, pp. 4696-4710, 2007.
[25] R. Ito, H. Kuwakado, and H. Tanaka, "Image size invariant visual cryptography," *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, vol. E82a, no. 10, pp. 2172-2177, 1999.
[26] R. Z. Wang and S. J. Shyu, "Scalable secret image sharing," *Signal Processing: Image Communication*, vol. 22, no. 4, pp. 363-373, 2007.
[27] W. P. Fang, "Friendly progressive visual secret sharing," *Pattern Recognition*, vol. 41, no. 4, pp. 1410-1414, 2008.
[28] S. J. Lin and J. C. Lin, "VCPSS: A two-in-one two-decoding-options image sharing method combining visual cryptography (VC) and polynomial-style sharing (PSS) approaches," *Pattern Recognition*, vol. 40, no. 12, pp. 3652-3666, 2007.
[29] C. N. Yang and T. S. Chen, "Aspect ratio invariant visual secret sharing schemes with minimum pixel expansion," *Pattern Recognition Letters*, vol. 26, no. 2, pp. 193-206, 2005.
[30] H. K. Tso, "Sharing secret images using Blakley's concept," *Optical Engineering*, vol. 47, no. 7, p. 077001, 2008.
[31] S. K. Chen and J. C. Lin, "Fault-tolerant and progressive transmission of images," *Pattern Recognition*, vol. 38, no. 12, pp. 2466-2471, 2005.
[32] P. Beguin and A. Cresti, "General information dispersal algorithms," *Theoretical Computer Science*, vol. 209, no. 1-2, pp. 87-105, 1998.
[33] F. P. Preparata, "Holographic Dispersal and Recovery of Information," *IEEE Transactions on Information Theory*, vol. 35, no. 5, pp. 1123-1124, 1989.
[34] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300-304, 1960.
[35] I. C. Lin, Y. B. Lin, and C. M. Wang, "Hiding data in spatial domain images with distortion tolerance," *Computer Standards & Interfaces*, vol. 31, no. 2, pp. 458-464, 2009.
[36] C. H. Yang, "Inverted pattern approach to improve image quality of information hiding by LSB substitution," *Pattern Recognition*, vol. 41, no. 8, pp. 2674-2683, 2008.
[37] S. J. Wang, "Steganography of capacity required using modulo operator for embedding secret image," *Applied Mathematics and Computation*, vol. 164, no. 1, pp. 99-116, 2005.
[38] X. P. Zhang and S. Z. Wang, "Efficient steganographic embedding by exploiting modification direction," *IEEE Communications Letters*, vol. 10, no. 11, pp. 781-783, 2006.
[39] S. L. Li, K. C. Leung, L. M. Cheng, and C. K. Chan, "A novel image-hiding scheme

based on block difference," *Pattern Recognition,* vol. 39, no. 6, pp. 1168-1176, 2006.
[40] S. S. Maniccam and N. Bourbakis, "Lossless compression and information hiding in images," *Pattern Recognition,* vol. 37, no. 3, pp. 475-486, 2004.
[41] R. Z. Wang and Y. S. Chen, "High-payload image steganography using two-way block matching," *IEEE Signal Processing Letters,* vol. 13, no. 3, pp. 161-164, 2006.
[42] Y. C. Tseng, Y. Y. Chen, and H. K. Pan, "A secure data hiding scheme for binary images," *IEEE Transactions on Communications,* vol. 50, no. 8, pp. 1227-1231, 2002.
[43] M. Y. Wu, Y. K. Ho, and H. H. Lee, "An iterative method of palette-based image steganography," *Pattern Recognition Letters,* vol. 25, no. 3, pp. 301-309, 2004.
[44] C. L. Liu and S. R. Liao, "High-performance JPEG steganography using complementary embedding strategy," *Pattern Recognition,* vol. 41, no. 9, pp. 2945-2955, 2008.
[45] Y. Lee, H. Kim, and Y. Park, "A new data hiding scheme for binary image authentication with small image distortion," *Information Sciences,* vol. 179, no. 22, pp. 3866-3884, 2009.
[46] J. X. Wang and Z. M. Lu, "A path optional lossless data hiding scheme based on VQ joint neighboring coding," *Information Sciences,* vol. 179, no. 19, pp. 3332-3348, 2009.
[47] H. W. Tseng and C. P. Hsieh, "Prediction-based reversible data hiding," *Information Sciences,* vol. 179, no. 14, pp. 2460-2469, 2009.
[48] D. C. Wu and W. H. Tsai, "A steganographic method for images by pixel-value differencing," *Pattern Recognition Letters,* vol. 24, no. 9-10, pp. 1613-1626, 2003.
[49] C. M. Wang, N. I. Wu, C. S. Tsai, and M. S. Hwang, "A high quality steganographic method with pixel-value differencing and modulus function," *Journal of Systems and Software,* vol. 81, no. 1, pp. 150-158, 2008.
[50] X. P. Zhang and S. Z. Wang, "Steganography using multiple-base notational system and human vision sensitivity," *IEEE Signal Processing Letters,* vol. 12, no. 1, pp. 67-70, 2005.
[51] C. H. Yang, C. Y. Weng, S. J. Wang, and H. M. Sun, "Adaptive data hiding in edge areas of images with spatial LSB domain systems," *IEEE Transactions on Information Forensics and Security,* vol. 3, no. 3, pp. 488-497, Sep 2008.
[52] H. F. Yang, X. M. Sun, and G. Sun, "A High-Capacity Image Data Hiding Scheme Using Adaptive LSB Substitution," *Radioengineering,* vol. 18, no. 4, pp. 509-516, 2009.
[53] C. S. Lu and H. Y. M. Liao, "Structural digital signature for image authentication: An incidental distortion resistant scheme," *IEEE Transactions on Multimedia,* vol. 5, no. 2, pp. 161-173, 2003.
[54] D. C. Lou and J. L. Liu, "Fault resilient and compression tolerant digital signature for image authentication," *IEEE Transactions on Consumer Electronics,* vol. 46, no. 1, pp. 31-39, 2000.
[55] C. T. Hsu and J. L. Wu, "Hidden digital watermarks in images," *IEEE Transactions on Image Processing,* vol. 8, no. 1, pp. 58-68, 1999.
[56] P. W. Wong and N. Memon, "Secret and public key image watermarking schemes for image authentication and ownership verification," *IEEE Transactions on Image Processing,* vol. 10, no. 10, pp. 1593-1601, 2001.
[57] M. Wu and B. D. Liu, "Data hiding in binary image for authentication and annotation," *IEEE Transactions on Multimedia,* vol. 6, no. 4, pp. 528-538, 2004.
[58] H. Luo, Z. M. Lu, S. C. Chu, and J. S. Pan, "Self embedding watermarking scheme using halftone image," *IEICE Transactions on Information and Systems,* vol. E91d, no.

1, pp. 148-152, 2008.
[59] P. L. Lin, C. K. Hsieh, and P. W. Huang, "A hierarchical digital watermarking method for image tamper detection and recovery," *Pattern Recognition*, vol. 38, no. 12, pp. 2519-2529, 2005.
[60] F. H. Yeh and G. C. Lee, "Content-based watermarking in image authentication allowing remedying of tampered images," *Optical Engineering*, vol. 45, no. 7, p. 077004, 2006.
[61] M. S. Wang and W. C. Chen, "A majority-voting based watermarking scheme for color image tamper detection and recovery," *Computer Standards & Interfaces*, vol. 29, no. 5, pp. 561-570, 2007.
[62] Y. Park, H. Kang, K. Yamaguchi, and K. Kobayashi, "Watermarking for tamper detection and recovery," *IEICE Electronics Express*, vol. 5, no. 17, pp. 689-696, 2008.
[63] T. Y. Lee and S. F. D. Lin, "Dual watermark for image tamper detection and recovery," *Pattern Recognition*, vol. 41, no. 11, pp. 3497-3506, 2008.
[64] S. S. Wang and S. L. Tsai, "Automatic image authentication and recovery using fractal code embedding and image inpainting," *Pattern Recognition*, vol. 41, no. 2, pp. 701-712, 2008.
[65] Y. J. Chang, Z. Wang, and J. C. Lin, "A Sharing-Based Fragile Watermarking Method for Authentication and Self-Recovery of Image Tampering," *EURASIP Journal on Advances in Signal Processing*, vol. 2008, no. 200, 2008.
[66] C. S. Chan and C. C. Chang, "An efficient image authentication method based on Hamming code," *Pattern Recognition*, vol. 40, no. 2, pp. 681-690, 2007.
[67] X. P. Zhang and S. Z. Wang, "Fragile Watermarking With Error-Free Restoration Capability," *IEEE Transactions on Multimedia*, vol. 10, no. 8, pp. 1490-1499, 2008.
[68] Y. J. Chang, S. J. Lin, and J. C. Lin, "Authentication and cross-recovery for multiple images," *Journal of Electronic Imaging*, vol. 17, no. 4, p. 043007, 2008.
[69] F. Liu, C. K. Wu, and X. J. Lin, "A new definition of the contrast of visual cryptography scheme," *Information Processing Letters*, vol. 110, no. 7, pp. 241-246, 2010.
[70] S. Lin and D. J. C. Jr., *Error Control Coding*, 2 ed.: Prentice Hall, 2004.
[71] D. Bini and V. Y. Pan, *Polynomial and matrix computations (vol. 1): fundamental algorithms*: Birkhauser Verlag, 1994.
[72] E. E. Varsaki, V. Fotopoulos, and A. N. Skodras, "Self-authentication of natural color images in Pascal Transform domain," in *16th International Conference on Digital Signal Processing*, Santorini, Hellas, pp. 1-6, Jul 2009.
[73] J. Massey, "Shift-register synthesis and BCH decoding," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 122-127, 1969.
[74] T. K. Truong, I. S. Hsu, W. L. Eastman, and I. S. Reed, "Simplified procedure for correcting both errors and erasures of Reed-Solomon code using Euclidean algorithm," in *IEE Proceedings-E of Computers and Digital Techniques*, vol. 135, pp. 318-324, Nov 1988.
[75] P. Moulin and R. Koetter, "Data-hiding codes," in *Proceedings of the IEEE*, vol. 93, pp. 2083-2126, Dec 2005.
[76] P. S. L. M. Barreto, H. Y. Kim, and V. Rijmen, "Toward a secure public-key blockwise fragile authentication watermarking," in *IEEE International Conference on Image Processing*, Thessaloniki, Greece, vol. 2, pp. 494-497, 2001.
[77] J. Fridrich, M. Goljan, and N. D. Memon, "Further attacks on Yeung-Mintzer fragile watermarking scheme," in *Proceedings of the SPIE Security and Watermarking of Multimedia Contents II*, vol. 3971, pp. 428-437, 2000.

[78] M. Holliman and N. Memon, "Counterfeiting attacks on oblivious block-wise independent invisible watermarking schemes," *IEEE Transactions on Image Processing*, vol. 9, no. 3, pp. 432-441, 2000.
[79] *The USC-SIPI Image Database*. Available: http://sipi.usc.edu/database/

Publication list

Journal papers

1. S. J. Lin, S. K. Chen, and J. C. Lin, "Flip Visual Cryptography (FVC) with perfect security, conditionally optimal contrast, and no expansion", *Journal of Visual Communication and Image Representation*, article in press. [SCI, EI] (Ch. 2).
2. S. J. Lin, L. S. T. Chen and J. C. Lin, "Fast weighted secret image sharing", *Optical Engineering*, vol. 48(7), Jul. 2009, pp. 077008. [SCI, EI] (Ch. 3).
3. S. J. Lin and J. C. Lin, "Authentication and Recovery of an Image by sharing and lattice-embedding", *Journal of Electronic Imaging*, vol. 19(4), Oct. 2010, article in press. [SCI, EI] (Ch. 5).
4. S. J. Lin and J. C. Lin, "VCPSS: a two-in-one two-decoding-options image sharing method combining visual cryptography (VC) and polynomial-style sharing (PSS) approaches", *Pattern Recognition*, vol. 40(12), Dec. 2007, pp. 3652-3666.[SCI, EI].
5. L. S. T. Chen, S. J. Lin and J. C. Lin, “Reversible JPEG-based hiding method with high hiding-ratio”, *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 24(3), pp. 433-456, 2010. [SCI, EI]
6. Y. J. Chang, S. J. Lin and J. C. Lin, "Authentication and cross-recovery for multiple images", *Journal of Electronic Imaging*, vol. 17(4), Oct. 2008, pp. 043007. [SCI, EI]

Conference papers

1. S. J. Lin, J. C. Lin and W. P. Fang, "Visual cryptography (VC) with non-expanded shadow images: a Hilbert-curve approach", *Proceedings on IEEE International Conference on Intelligence and Security Informatics (ISI2008)*, 2008.[EI].
2. W. P. Fang and S. J. Lin, "Fast Secret Image Sharing Scheme in HPC, "*Proceedings on the 10th International Conference on High-Performance Computing in Asia-Pacific Region(HPC ASIA 2009) joint WorkShop on PC-Grid*, 2009.
3. S. K. Chen and S. J. Lin, "Non-expansible Flip-flop Visual Cryptography with perfect security", *Proceedings on the Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2009.

Under Review

1. S. J. Lin and J. C. Lin, "Weighted-Sum Function (WSF) – A Gray-scale Image Hiding Method with Competitive PSNR over a Wide Range of Embedding Rates“, *Information Sciences*. (Ch. 4).

Vita

Sian-Jheng LIN was born in Taiwan, Republic of China. He received his B.S., M.S. and Ph.D. degree in Computer Science from National Chiao Tung University in 2004, 2006 and 2010, respectively. His recent research interests include data hiding and secret sharing.