Available online at www.sciencedirect.com

ScienceDirect

Pattern Recognition 40 (2007) 3652–3666

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/pr

VCPSS: A two-in-one two-decoding-options image sharing method combining visual cryptography (VC) and polynomial-style sharing (PSS) approaches

Sian-Jheng Lin, Ja-Chen Lin*

Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, ROC

Received 17 January 2006; received in revised form 4 April 2007; accepted 5 April 2007

Abstract

This paper presents a novel method to combine two major branches of image sharing: VC and PSS. n transparencies are created for a given gray-valued secret image. If the decoding computer is temporarily not available at (or, not connected to) the decoding scene, we can still physically stack any t received transparencies ($t \leqslant n$ is a threshold value) to get a vague black-and-white view of the secret image immediately. On the other hand, when the decoding computer is finally available, then we can get a much finer gray-valued view of the secret image using the information hidden in the transparencies. In summary, each transparency is a two-in-one carrier of the information, and the decoding has two options.

© 2007 Pattern Recognition Society. Published by Elsevier Ltd. All rights reserved.

Keywords: Hiding using block-types; Image sharing; (t, n) threshold scheme; Halftone version; Compression

1. Introduction and goal

Image sharing can be used in a team when no member alone should be trusted. Visual cryptography (VC) [1–7] and polynomial-style sharing (PSS) [8–14] are both well-known branches to share images. Both can be designed as (t, n) schemes. (In this paper, we say that a sharing technique is (t, n) if and only if it shares a secret image $\boldsymbol{S}$ among n shadows so that any t of the n shadows ($n \geqslant t$) can unveil the secret image $\boldsymbol{S}$ (or a compressed version $\boldsymbol{S}^{(\mathrm{comp})}$ of $\boldsymbol{S}$), whereas less than t shadows cannot.) Although both VC and PSS can share images, they are quite different in many manners. Table 1 below compares VC and PSS.

In Table 1, if we temporarily ignore the final column (which is for the future comparison use in the experiment section, we list this column here just to save paper's space), we can see that VC is simple and fast, while PSS gives good image quality. A question arises naturally: "Can VC be combined with PSS?" To certain extent, the answer is positive, as is shown here. In this paper, we present a method to combine these two techniques and achieve a goal: if the decoding computer is temporarily not available in (or, not connected to) the decoding scene, we can still physically stack the t received shadows to get a vague black-and-white view of the secret image immediately; later, when the computer is finally available, we can get a much finer gray-valued view of the secret image using the information hidden earlier in the shadows by using PSS. (Hereinafter, the final output of the proposed method will be called as "transparencies" rather than "shadows" because, as mentioned above, one of the two decoding manners is that the shadows can be stacked physically for viewing, just like ordinary transparencies can be stacked and viewed.)

Below in Section 2 we first review some background knowledge used in this paper, and then in Section 3 we introduce our method. The experimental results are in Section 4, and the conclusions are in Section 5. Section 6 describes an application of this paper.

2. Background

Some background knowledge is reviewed in this section. Section 2.1 reviews the basis matrices roughly, which is a

* Corresponding author. Fax: +886 3 5721490.
E-mail address: jclin@cis.nctu.edu.tw (J.-C. Lin).

Table 1
A comparison between VC and PSS

	Visual cryptography (VC, see [1–7])	Polynomial-style sharing (PSS, see [8–14])	Ours (VC + PSS)
Usually, the input secret image ***S*** is	Black-and-white	Gray	Gray
Decoding speed (and decoding method)	Instant (by using eyes after stacking shadows)	Slow (by computation)	Instant in Layer 1; slow in Layer 2.
Is a computer needed in decoding?	No	Yes	"No" in Layer 1; "yes" in Layer 2.
Recovered image's perceptual quality	Vague	Fine	Vague in Layer 1; fine in Layer 2.
Size of each shadow	Larger than that of ***S***	Can be smaller than that of ***S***	Either the length or the width of a (binary) transparency is larger than that of ***S***, but the number of computer-storage bytes needed can be smaller than ***S***'s.

background knowledge well known in VC field; Section 2.2 reviews the PSS technologies, including Shamir's [14] and Thien and Lin's [8].

2.1. A review of the basis matrices [$\mathbf{B_0}$] and [$\mathbf{B_1}$] for VC

Below we review the two basis matrices [$\mathbf{B_0}$] and [$\mathbf{B_1}$] often mentioned in VC field (e.g. see Ref. [1]). The matrix $\mathbf{B_0}$ is called a "white matrix" because it is useful to produce blocks whose stacking result will represent white pixels of a black-and-white (e.g. halftone) image. Matrix [$\mathbf{B_1}$] is called a "black matrix" for analogous reason. Without the loss of generality, below we only show the case $(t, n) = (2, 4)$, i.e. only two out of four shares are needed in recovering. For a general pair of given values (t, n), the readers may either design their own [$\mathbf{B_0}$] and [$\mathbf{B_1}$], or use the Appendix to create some pairs of [$\mathbf{B_0}$] and [$\mathbf{B_1}$]. In fact, even if the values of t and n are fixed, the choice of [$\mathbf{B_0}$] and [$\mathbf{B_1}$] is still not unique. To apply the proposed VCPSS two-in-one sharing method, people can use any pair of [$\mathbf{B_0}$] and [$\mathbf{B_1}$] satisfying the requirements (i)–(iii) stated in next paragraph (these three requirements also appear in the Appendix). In summary, the pair [$\mathbf{B_0}$] and [$\mathbf{B_1}$] is not necessarily generated from the Appendix; the Appendix is just to let readers know that there always exists at least one solution to find out [$\mathbf{B_0}$] and [$\mathbf{B_1}$].

In the $(t, n) = (2, 4)$ case, one of the several possible choices for the white matrix [$\mathbf{B_0}$] and the black matrix [$\mathbf{B_1}$] is to use

$$[\mathbf{B_0}] = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \text{and} \quad [\mathbf{B_1}] = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}. \tag{1}$$

Both matrices have $n = 4$ rows. (In general, no matter how we assign the two matrices, each matrix must have n rows if n transparencies are to be created. This is the so-called requirement (i).) In both matrices, each 0 means that a white element is painted there, and each 1 means a black element is painted there. As we can see, both [$\mathbf{B_0}$] and [$\mathbf{B_1}$] have two black elements per row. (In general, the number of 1's appearing in each row of [$\mathbf{B_0}$] must be identical to that of [$\mathbf{B_1}$]. This is the so-called requirement (ii).) It is also obvious that if we stack any two (=t) rows of our [$\mathbf{B_0}$], the stacking result has two black elements and two white elements. On the other hand, if we stack any two (=t) rows of [$\mathbf{B_1}$], the stacking result has at least three black elements. (In general, no matter how we choose [$\mathbf{B_0}$] and [$\mathbf{B_1}$], the number of 1's contained in the result of stacking any t rows of [$\mathbf{B_1}$] must exceed that of stacking any t rows of [$\mathbf{B_0}$]. This is the so-called requirement (iii).)

Now, assume that we want to create 4(=n) blocks, each is 2×2 in size, so that stacking any 2(=t) of them will yield a 2×2 so-called "white block" (defined here as a block in which only two of the four elements are 1's (i.e. only two black elements)). All we have to do is to permute the columns of [$\mathbf{B_0}$] randomly, and then distribute the 4(=n) rows of the permuted [$\mathbf{B_0}$] to four customers. After that, each customer uses the first two elements as the first row of his block, and next two elements as the 2nd row of his block. As a result, each of the 4(=n) customers has his own 2×2 block, and any two of these four 2×2 blocks can be stacked to yield a 2×2 white block (only two of its $2 \times 2 = 4$ elements are 1's).

Similarly, if we want that any t(=2) of the n(=4) created blocks (each is still 2×2 in size) can be stacked to yield a so-called "black block" (defined as a 2×2) block in which at least three of the four elements are 1's (i.e. at least three black elements), then we only have to replace the role of [$\mathbf{B_0}$] by [$\mathbf{B_1}$] in the above argument and obtain four blocks corresponding to [$\mathbf{B_1}$]. Then distribute these four blocks arbitrarily to the four customers (one block per customer).

In the above example, each block has $w = 2$ white elements and $b = 2$ black elements, (or equivalently, each row of [$\mathbf{B_0}$] or [$\mathbf{B_1}$] has two white elements and two black elements), and the permutation of the columns of [$\mathbf{B_0}$] or [$\mathbf{B_1}$] will not affect the stacking result's brightness (i.e. number of black elements of the stacking result). Moreover, when we do column permutation of the matrix [$\mathbf{B_0}$] (or [$\mathbf{B_1}$]), if we concentrate on the first row, we can see that the first row can be either [0 0 1 1], or [0 1 0 1], or [1 0 0 1], or [0 1 1 0], or [1 0 1 0], or [1 1 0 0]. Therefore, all $\binom{2+2}{2} = 6$ types of row vectors can appear. The $\binom{w+b}{b} = \binom{4}{2} = 6$ types of blocks represented by these $\binom{w+b}{b} = \binom{4}{2} = 6$ types of row vectors will be called as fundamental blocks. (In general, if a VC system uses blocks of $(w + b)$ elements each, and each block of each transparency has w white elements and b black elements, then there will be $\binom{w+b}{b} = (w + b)!/(w!b!)$ types of fundamental blocks. For example, if 3×3 blocks are tobe used, and if each block of each transparency has five white elements and $4 = 3 \times 3$–5 black elements, then there will be $\binom{5+4}{5} = 126$ types of fundamental blocks. We can call them, respectively, blocks of type 0, type 1, type 2, . . ., and type $125 = \binom{5+4}{5} = 126$).

As a remark of this subsection and the Appendix, note that the Appendix is a self-explained appendix derived from Ref. [2] which is a graceful paper proposed by Ateniese et al. In order to reduce current paper length and concentrate on our topic, we did not intend to discuss in our appendix the many materials mentioned in Ref. [2]. Interested readers should refer to Ref. [2] for further details. As stated earlier, the Appendix here is just to show the readers that they can always create two basis matrices [$\mathbf{B_0}$] and [$\mathbf{B_1}$] for any pair of given t and n ($2 \leqslant t \leqslant n$).

2.2. A review of PSS

Below we roughly review Shamir's secret-value sharing scheme [14] and Thien and Lin's secret-image sharing scheme [8]. Both methods used polynomials to share a secret into n shadows, and the secret can be revealed by any t out of the n shadows.

2.2.1. To share a numerical value by Shamir's sharing scheme [14]

Below we review Shamir's sharing scheme [14]. To share a secret numerical value a_0 into n shadows, Shamir defined a polynomial

$$p(x) = a_0 + a_1x + \cdots + a_{t-1}x^{t-1}, \tag{2}$$

in which t was the threshold, the constant term a_0 was the secret value, and all other coefficients $\{a_i\}_{i=1,\dots,t-1}$ were random numbers for data-protection use. The n created shadows were $(1, p(1))$, $(2, p(1))$, . . . , $(n, p(n))$. Later, anyone who got any t of the n shadows could use Lagrange's interpolation to evaluate all coefficients, particularly, the coefficient a_0, of the polynomial $p(x)$; for passing through t points in two-dimensional plane uniquely defined the interpolation polynomial of degree not more than t. However, if a person only got $t - 1$ or fewer shadows, he could not unveil $\{a_i\}_{i=0,\dots,t-1}$; the secret number a_0 is thus still unknown.

In practice, people can use finite field arithmetic (e.g. modular arithmetic and Galois field arithmetic) to replace the arithmetic above. Also note that, if we repeatedly use Shamir's secret sharing scheme $|\boldsymbol{S}|$ times to share an image $\boldsymbol{S}$ (share one pixel value each time), then each shadow i receives a value $p(i)$ from each pixel. As a result, each shadow is as big as the image $\boldsymbol{S}$, for each shadow receives $|\boldsymbol{S}|$ values. This size is too large if we want to hide each shadow in a transparency later. In order to get smaller-size shadows, rather than using Shamir's scheme, we will use an approach similar to Thien and Lin's secret-image sharing scheme [8]. Although the shadow size is more economic, we will lose Shamir's data protection through the use of random numbers in Eq. (2). That is why we need to encrypt the image before sharing it, as will be seen in Section 3.2.3.

2.2.2. To share an image by Thien and Lin's sharing scheme [8]

In order to reduce the size of shadows, Thien and Lin [8] let all coefficients in $p(x)$ be data. No random numbers were used. They partitioned the given image $\boldsymbol{S}$ into $|\boldsymbol{S}|/t$ sectors, and each sector had t pixels. For each sector, they used the gray-values of its t pixels as the t coefficients $\{a_0, \dots, a_{t-1}\}$ in the sector-dependent polynomial

$$p(x) = (a_0 + a_1x + a_2x^2 + \cdots + a_{t-1}x^{t-1}) \bmod 251.$$

Then, for each sector, shadow s_i received a value $p(i)$, true for each $i = 1, \dots n$. As the sectors were processed sequentially, the data size of each of the n shadows also grew. Finally, when all $|\boldsymbol{S}|/t$ sectors were processed, there were n shadows. For each $i = 1, \dots, n$, shadow s_i received one value from each of the $|\boldsymbol{S}|/t$ sectors of image $\boldsymbol{S}$; so each shadow s_i has $|\boldsymbol{S}|/t$ values. Each shadow is therefore t times smaller than the image $\boldsymbol{S}$.

3. The proposed method

3.1. Main idea

Let image $\boldsymbol{H} = \boldsymbol{H_S}$ be a halftone binary version of the image $\boldsymbol{S}$. In general, any binary image can be shared using any (t, n)-threshold VC technology; so n transparencies $\{r_1, r_2, \dots, r_n\}$ can be created for the binary image $\boldsymbol{H}$. In this paper, however, we wish that the n created VC transparencies can also hide the gray-value information of $\boldsymbol{S}$, so that any t of the n created transparencies cannot only be stacked to "view" the superimposed black-and-white result as in ordinary VC, but also to extract the information of $\boldsymbol{S}$ hidden in the t received transparencies, and thus reconstruct the gray-value image $\boldsymbol{S}$. As a result, the design of the n transparencies $\{r_1, r_2, \dots, r_n\}$ needs special treatment. The above two-in-one goal can be achieved by first using (t, n)-threshold PSS to share $\boldsymbol{S}$ into n shadows $\{s_1, s_2, \dots, s_n\}$, and then, for each $i = 1, 2, \dots, n$, hide s_i in transparency r_i.

3.1.1. Hiding shadow s_i in (region i of) transparency r_i

Since $\boldsymbol{S}$ has $|\boldsymbol{S}|$ gray-value pixels, its binary version $\boldsymbol{H}$ also has $|\boldsymbol{S}|$ binary pixels. As a result, each transparency r_i has $|\boldsymbol{S}|$ blocks, for each pixel of $\boldsymbol{H}$ is mapped to a block of r_i. If we partition the $|\boldsymbol{S}|$ blocks into n equal-size regions, namely, regions 1–n, then each region has $|\boldsymbol{S}|/n$ blocks. Notably, the readers have the freedom to choose their own way to partition the $|\boldsymbol{S}|$ blocks into n regions, and the blocks in a region are not necessarily adjacent to each other. An example is to let the blocks in region i be chosen as blocks $\{i, i+n, i+2n, i+3n, \ldots\}$. Another example is to let the blocks in region i be chosen as the n consecutive blocks so that region 1 has blocks 1 to $|\boldsymbol{S}|/n$, and region 2 has blocks $\{1+(|\boldsymbol{S}|/n), 2+(|\boldsymbol{S}|/n), 3+(|\boldsymbol{S}|/n), \ldots, (|\boldsymbol{S}|/n)+|(\boldsymbol{S}|/n)\}$, and so on. The partition can be quite free since the only requirement is that each region has $|\boldsymbol{S}|/n$ blocks.

After the partition, let transparency r_1 use the $|\boldsymbol{S}|/n$ blocks in region 1 to hide the information of the shadow s_1, and let transparency r_2 use the $|\boldsymbol{S}|/n$ blocks in region 2 to hide information of the shadow s_2, etc. Assume that each shadow s_i has been transformed to a numerical file using digits in the range $\left\{0, 1, \ldots, \binom{w+b}{b}-1\right\}$; in other words, assume each s_i is a numerical file of base $\binom{w+b}{b}$. Also assume that each shadow s_i has at most $|\boldsymbol{S}|/n$ digits. Then, for each $i=1,2,\ldots,n$, we can hide the $|\boldsymbol{S}|/n$ digits of shadow s_i into the $|\boldsymbol{S}|/n$ blocks of region i of transparency r_i. Below we show how. Without the loss of generality, we show here how to hide the $|\boldsymbol{S}|/n$ digits of s_1 in region 1 of transparency r_1. For the jth block in region 1 of transparency r_1, where $1 \leqq j \leqq |\boldsymbol{S}|/n$, we can paint the block so that its block type is exactly the jth digit of s_1. We can do this because there are $\binom{w+b}{b} = (w+b)!/(w!b!)$ kinds of digits, and there are exactly $\binom{w+b}{b}$ kinds of fundamental blocks. For example, when $w=b=2$, the $\binom{w+b}{b}=\binom{4}{2}=6$ types of fundamental blocks are

$$[0\,0\,1\,1],\ [0\,1\,0\,1],\ [0\,1\,1\,0],\\ [1\,0\,0\,1],\ [1\,0\,1\,0] \text{ and } [1\,1\,0\,0]. \tag{3}$$

(For easier future reference, we have purposely arranged the blocks in an increasing order according to their binary values, i.e. according to the fact that $0011 < 0101 < 0110 < 1001 < 1010 < 1100$.) The above six blocks can be called as blocks of

$$\text{type } 0,\ \text{type } 1,\ \text{type } 2,\\ \text{type } 3,\ \text{type } 4 \text{ and type } 5, \tag{4}$$

respectively, if we want to use the simplest naming in which $0<1<2<3<4<5$ is also arranged in an increasing order. Then, to encode a digital string 5-5-2-3-0-4-4 in which each digit is in the range {0–5}, we can paint the corresponding sequence of blocks as [1 1 0 0]-[1 1 0 0]-[0 1 1 0]-[1 0 0 1]-[0 0 1 1]-[1 0 1 0]-[1 0 1 0]. Of course, there are many other ways to call the six types of the fundamental blocks. (For example, the above (0, 1, 2, 3, 4, 5) naming of the block types for ([0 0 1 1], [0 1 0 1], [0 1 1 0], [1 0 0 1], [1 0 1 0], [1 1 0 0]) can be replaced by another naming (3, 4, 0, 1, 5, 2) so that type 3 means [0 0 1 1], type 4 means [0 1 0 1], type 0 means [0 1 1 0], etc.) In general, there are $\binom{w+b}{b}! = [(w+b)!/(w!b!)]!$ possible ways of doing naming. The naming can be recorded by a mapping table $\boldsymbol{L}$ which has $\binom{w+b}{b}$ entries. For instance, in the above two examples, the corresponding mapping tables are the strings 012345 and 340152, respectively. The mapping table $\boldsymbol{L}$, which is a numerical string, can be either public to all n participants or shared by n participants using Shamir's sharing scheme [14].

When we embed shadow s_i in transparency r_i, some readers might wonder why a region is only used for a special pair of s_i and r_i, i.e. why we do not use blocks of the same region to do the embedding for another pair of shadow and transparency. The reason is explained below.

Once all blocks in region 1 of transparency r_1 have been painted according to the digits in the first shadow s_1 (so that the digits in s_1 can be hidden in these blocks), it is not suitable to paint again the blocks in region 1 of another transparency r_2 according to the digits in the second shadow s_2 (so that the digits in s_2 can be hidden in these blocks). Otherwise, the result of stacking the generated transparencies is not guaranteed to meet the black/white VC requirement (to look like the halftone image $\boldsymbol{H}$) in area corresponding to region 1. For example, assume $w=b=2$, (see Eqs. (3) and (4)) and blocks [1 1 0 0] is type 5. Then, to hide a digit value 5 of the shadow s_1 in a block of transparency r_1, we paint that block of transparency r_1 as [1 1 0 0]. Now, if we also want to hide a digit value 5 of the shadow s_2 in the same block position of transparency r_2, that very block of transparency r_2 will also be painted as [1 1 0 0]. Now, without the loss of generality, assume (t, n) is (2,4), i.e. collecting two transparencies is enough to stack and yield the halftone image $\boldsymbol{H}$. However, what happens when we stack the two transparencies r_1 and r_2 is that: the stacked result is again [1 1 0 0] on that very block. Since [1 1 0 0] is a white block according to the definitions given in the two paragraphs below Eq. (1), this means that the image $\boldsymbol{H}$ must be a white pixel at the position corresponding to the very block position. Of course this is a ridicules requirement, for $\boldsymbol{H}$ might be black there.

3.1.2. *Painting region i for the remaining $(n-1)$ transparencies $\{r_k\}_{k\neq i}$*

For each $i=1,2,\ldots,n$, after hiding shadow s_i in region i of transparency r_i, the block type of each block in region i of r_i is already fixed. Below we discuss how to paint region i for the remaining $(n-1)$ transparencies $\{r_k\}_{k\neq i}$. Without the loss of generality, we show how to paint region 1 for the $(n-1)$ transparencies $\{r_k\}_{k\neq 1}$. Since each region has $|\boldsymbol{S}|/n$ blocks, for each block position j $(1 \leqslant j \leqslant |\boldsymbol{S}|/n)$ in region 1, we will use the block type of the (painted) jth block in region 1 of transparency r_1 to determine how to paint the jth block in region 1 of the remaining $(n-1)$ transparencies $\{r_2, \ldots, r_n\}$, as shown below. Recall that, in any VC system, each block of a transparency corresponds to a pixel of the binary image $\boldsymbol{H}$; so, for the jth block in region 1 of transparency r_1, we may locate the corresponding pixel position in the halftone image $\boldsymbol{H}$. If $\boldsymbol{H}$ is a white (black) pixel there, we certainly hope that the result of stacking the n generated transparencies is also a white (black) block there. Define a matrix $[\mathbf{B}]$ which is identical to $[\mathbf{B_0}]$ if the desired superimposed result is a white block; otherwise, set $[\mathbf{B}]$ to $[\mathbf{B_1}]$. Now, permute the columns of $[\mathbf{B}]$ so that the first row of [**B**] is identical to the painted jth block in region 1 of transparency r_1. (This can be done because both the first row of [**B**] and that painted block of transparency r_1 has w white elements (0's) and b black elements (1's).) Notably, if more than one permutation can make the first row of [**B**] identical to the painted jth block in region 1 of transparency r_1, then we "randomly" select one of those permutations to permute the matrix [**B**]. The term "randomly" means "uniformly" here, i.e. all qualified permutations should have equal chances of being selected. The randomness of the painted patterns is a commonly seen technique in VC to avoid the information being easier to guess by the hackers due to repeatedly used patterns.

After the permutation of columns, use the remaining $(n-1)$ rows of the matrix [**B**] to paint the remaining $(n-1)$ transparencies at the same block position (i.e. the jth block in region 1). This ensures the stacking result will be a white (black) block there if the matrix [**B**] is a white (black) matrix.

3.2. Some other details to keep the main idea working

Having introduced the main idea in Section 3.1, we discuss the details to implement it.

3.2.1. To share a gray-value image $\boldsymbol{S}$ into n shadows $\{s_1, s_2, \ldots, s_n\}$

In Section 3.1, we said that the desired two-in-one goal can be achieved by first using (t, n)-threshold PSS to share $\boldsymbol{S}$ into n shadows $\{s_1, s_2, \ldots, s_n\}$; and then, for each $i = 1, 2, \ldots, n$, hide s_i in transparency r_i. We also said that each shadow s_i should be transformed to a numerical file using digits in the range $\left\{0, 1, \ldots, \binom{w+b}{b} - 1\right\}$ before hiding. Below we discuss how to create shadows $\{s_1, s_2, \ldots, s_n\}$ and how to transform each file to a numerical file of base $\binom{w+b}{b}$.

To share the gray-value image $\boldsymbol{S}$ by a (t, n)-threshold PSS, we divide $\boldsymbol{S}$ into $|\boldsymbol{S}|/t$ sectors of t pixels each. Then, for each sector, use the gray-values of its t pixels as the t coefficients $\{a_0, \ldots, a_{t-1}\}$ in the sector-dependent polynomial

$$p(x) = (a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}) \bmod 257. \tag{5}$$

Then, for each sector, shadow s_i receives a value $p(i)$, true for each $i = 1, \ldots, n$. As we process the sectors sequentially, the data size of each of the n shadows also grows. Finally, when we finish all $|\boldsymbol{S}|/t$ sectors, we obtain n shadows. s_i receives one value from each sector, and there are $|\boldsymbol{S}|/t$ sectors in image $\boldsymbol{S}$; so each shadow s_i has $|\boldsymbol{S}|/t$ values. Since each value is in the range $\{0, 1, \ldots, 256\}$ due to the use of mod 257, each shadow s_i can be treated as a digit string of numerical base 257. To avoid misunderstanding, notice that the temporary output value 256 in the set $\{0, 1, \ldots, 256\}$ is just one of the intermediate values in our method; it is a numerical value to be hidden later in a transparency, and this value is not a gray value of any image.

In order to hide each shadow s_i in transparency r_i, we need to transform s_i from the digit string using base 257 to a new digit using base $\binom{w+b}{b}$; for we use block types to hide digits, and each transparency only uses $\binom{w+b}{b}$ kinds of fundamental blocks, as mentioned earlier in the third paragraph of Section 3.1. The base switching from base 257 to base $\binom{w+b}{b}$ is an obvious job. There are many ways to do it. One of the possible ways is to switch first from base 257 to binary, and then switch the binary sequence to the desired base. Another way is to switch from 257 to $\binom{w+b}{b}$ directly, which is particularly easy if $(w+b)!/(w!b!) > 257$; but we will not use this easier and direct approach because it is not economic in reducing the length of the final string. In general, no matter how a reader designs his own way to do base switching, try to keep the length of the final string (of base $\binom{w+b}{b}$) as compact as possible; for smaller size of the final string means hiding the string later will be easier.

If a reader wants to switch first from base 257 to binary, in order to make the length of the binary string short, he may proceed as follows. If a base 257 digit is in the set $\{0, 1, 2, \ldots, 254\}$, encode the digit as its traditional 8-bit binary counterpart; for example, 0 is 0000-0000, and 254 is 1111-1110. However, if the base 257 digit is 255 (or 256), then encode it as the 9-bit code 1111-1111-0 (or 1111-1111-1). In other words, later in the decoding, the decoder takes 8 bits each time. If these 8 bits are 1111-1111, then take one more bit to determine whether the number is 255 or 256. However, if these 8 bits are not 1111-1111, then just decode these 8 bits to a decimal equivalent in the range $\{0, 1, 2, \ldots, 254\}$ as in the traditional 8-bit decoder.

Notably, in Ref. [8], Thien and Lin used mod 251 rather than mod 257. This might cause some encoding–decoding trouble should the input image $\boldsymbol{S}$ owned some input-gray-values exceeding 250. (To avoid such overflow problem, Ref. [8] had to split an input pixel of this kind into two pixels. For example, splited a gray value 253 as {249 and 4}, for $253 = 249 + 4$.) Therefore, in the current paper, we use mod 257 rather than mod 251. Also note that mod 256 is not suitable, for 256 is not a prime, and this kind of function requires a prime number be used in the mod function. However, if a reader insists to use 256, then he will have to use all operations in terms of Galois field rather than ordinary arithmetic; the detail is omitted to save the paper length.

3.2.2. Compression of the gray-value image $\boldsymbol{S}$ might be needed

As stated in Section 3.1, for each $i = 1, 2, \ldots, n$, in order that we can hide the $|\boldsymbol{S}|/n$ digits of s_i in the $|\boldsymbol{S}|/n$ blocks in region i of transparency r_i, we have assumed that: each shadow s_i can be transformed to numerical files using digits in the range $\left\{0, 1, \ldots, \binom{w+b}{b} - 1\right\}$, and each shadow s_i has at most $|\boldsymbol{S}|/n$ digits. Unfortunately, if $\binom{w+b}{b}$ is too small, then each shadow s_i might have more than $|\boldsymbol{S}|/n$ digits when s_i is transformed to a digit–string using digits in the range $\left\{0, 1, \ldots, \binom{w+b}{b} - 1\right\}$. To make the system still work, it might be necessary to reduce the length of s_i, which can be achieved by compressing the gray-value image $\boldsymbol{S}$ before sharing $\boldsymbol{S}$ into the n shadows $\{s_1, s_2, \ldots, s_n\}$.

When the image $\boldsymbol{S}$ is compressed by Jpeg or any other compression tool, a compact version of $\boldsymbol{S}$, called $\boldsymbol{S}^{(\mathrm{comp})}$, is produced. The value

$$ratio = \frac{\text{number of bits in } \boldsymbol{S}}{\text{number of bits in } \boldsymbol{S}^{(\text{comp})}} \geqslant 1$$

will affect the quality of the recovered gray-value image $\boldsymbol{S}^{(\text{comp})}$ when we collect later any t of the n-produced shadows and extract $\boldsymbol{S}^{(\text{comp})}$ from the shadows. If the *ratio* is too large, the recovered gray-value image quality will be quite poor. On the other hand, if the *ratio* is too small, the size of $\boldsymbol{S}^{(\text{comp})}$, and hence, the size of each shadow s_i, will be too big, thus making the hiding of s_i in transparency r_i become impossible. Below we derive a formula to estimate the suitable value for *ratio*.

As mentioned in Section 3.1, we use block types to hide digits. Since the number of possible block types is $\binom{w+b}{b}$, the possible digits are $\left\{0, 1, \ldots, \binom{w+b}{b} - 1\right\}$. Therefore, each block can hide about $\log_2 \binom{w+b}{b}$ bits. From Section 3.1, we know that there are $|\boldsymbol{S}|/n$ blocks in region i of transparency r_i; therefore, when we use region i of transparency r_i to hide shadow s_i, the shadow s_i can have at most $|\boldsymbol{S}|/n$ digits, or, in terms of binary bits, $\left(\log_2 \binom{w+b}{b}\right) \times |\boldsymbol{S}|/n$ bits.

On the other hand, when we compress image $\boldsymbol{S}$ before sharing, the number of bits contained in the compressed version $\boldsymbol{S}^{(\text{comp})}$ of image $\boldsymbol{S}$ is $8 \times |\boldsymbol{S}|/ratio$ (assuming that each pixel has 8 bits, and the term *ratio* denotes the compression ratio). After sharing the $\boldsymbol{S}^{(\text{comp})}$ by (t, n)-threshold sharing, due to the fact that we use a sharing similar to Thien and Lin's sharing, the size of each shadow is t times smaller (see Section 3.2.1 or the final sentence of Section 2.2.2). This means that each shadow s_i has about $(8 \times |\boldsymbol{S}|/ratio)/t$ bits. As a result, the value of *ratio* cannot be too small because we require that $\left(\log_2 \binom{w+b}{b}\right) \times |\boldsymbol{S}|/n \geqq (8 \times |\boldsymbol{S}|/ratio)/t$; in other words, we have the requirement

$$ratio \geqslant (8n) \Big/ \left(t \times \log_2 \binom{w+b}{b}\right). \tag{6}$$

Notably, if the right-hand side of the above equation is less than 1, which happens when $(w+b)!/(w!b!)$ is large, then the compression is not necessary, for we can let *ratio* be 1 to satisfy Eq. (6), and a compression with $ratio = 1$ means no compression at all.

3.2.3. Encryption of $\boldsymbol{S}^{(\text{comp})}$ before sharing

After obtaining the compressed version $\boldsymbol{S}^{(\text{comp})}$ of $\boldsymbol{S}$, we should encrypt $\boldsymbol{S}^{(\text{comp})}$ before sharing $\boldsymbol{S}^{(\text{comp})}$. In the special case when there is no compression, there is no $\boldsymbol{S}^{(\text{comp})}$; in that case, the encryption before sharing is on $\boldsymbol{S}$.

Below we explain why encryption is needed. In Shamir's approach [14] to sharing a secret number a_0, the $t-1$ coefficients $\{a_i\}_{i=1,\ldots,t-1}$ in the polynomial $p(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$ are random numbers, i.e. only the coefficient a_0 is the data. However, in our approach dealing with large-size image data, we let all t coefficients be gray-valued data for economic reason. Therefore, although we get the benefit that our shadow size is t times smaller than Shamir's, we no longer have the protection of using random numbers to protect the data. The encryption of the data before sharing is thus needed.

To encrypt $\boldsymbol{S}^{(\text{comp})}$ or $\boldsymbol{S}$ to get an encrypted bit stream, we may use a security key, or use some very simple functions. For example, we may just use XOR function in a bit-by-bit manner on the two available bit streams: the bit stream representing $\boldsymbol{S}^{(\text{comp})}$, and the bit stream of the binary halftone image $\boldsymbol{H}$. (Re-read the image $\boldsymbol{H}$ several times if the size of $\boldsymbol{S}^{(\text{comp})}$ is larger than that of $\boldsymbol{H}$.) This kind of encryption uses no key, and the binary halftone image $\boldsymbol{H}$ needed in decryption can be extracted when the decoder stacks t of the n transparencies (see Step 1 of Section 3.4); so there is no problem such as "who keeps the key?" However, if people want to use a security key rather than this approach of using XOR and $\boldsymbol{H}$, then the key can be kept by all participants, or be shared in a (t, n) manner by all n participants using Shamir's scheme [14], or, just be kept by the company boss or the team organizer who insists that the decoding meeting must have his attendance, even though he may keep none of the n transparencies.

3.3. A flowchart to summarize the idea of encoding

Fig. 1 is a flowchart that summarizes the ideas in Sections 3.1 and 3.2 for encoding.

3.4. Decoding

Once we collect any t of the n transparencies $\{r_1, r_2, \ldots, r_n\}$, we may stack them to view immediately an enlarged version of the halftone image $\boldsymbol{H}$ without any computation. Then, according to the following decoding algorithm, we may obtain precisely the original-size halftone image $\boldsymbol{H}$ using Step 1. Besides that, we may use the remaining steps of the decoding algorithm to extract the information hidden in the t transparencies, and thus reconstruct a gray-value image $\boldsymbol{S}^{(\text{comp})}$, which is the Jpeg-compressed version of $\boldsymbol{S}$. The flowchart showing the components of the decoding is in Fig. 2. The decoding algorithm is also given below.

Computer-aided decoding algorithm.

Input: (I1). Any t of the n transparencies $\{r_1, r_2, \ldots, r_n\}$.

(I2). The one-to-one and onto mapping table L that maps the $\binom{w+b}{b}$ possible block types to the set $\left\{0, 1, \ldots, \binom{w+b}{b} - 1\right\}$. (The pair of Eqs. (3) and (4) gives an example of such mapping, which is so simple that it might not even need be stored or shared due to plainness.)

(I3). The security key used in the encryption process of Section 3.2.3. (No such key exists if Section 3.2.3 used a key-less XOR encryption.)

Output: The halftone image H, and a gray-value image $\boldsymbol{S}^{(\text{comp})}$ which is a compressed version of the gray-value image $\boldsymbol{S}$.

Steps: 1. Stack all t collected transparencies. Then, for each block of the stacked image, counting the black (white) elements in the block, this can determine whether the stacked block is a black block or a white block. If the block is black (white), then the pixel at the corresponding position of the halftone image $\boldsymbol{H}$

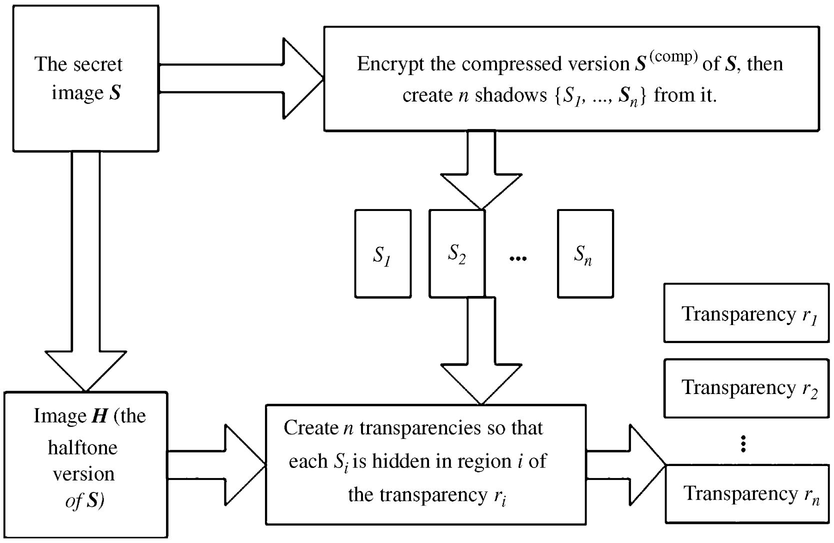

The secret image *S*
Encrypt the compressed version $S^{(\text{comp})}$ of *S*, then create *n* shadows $\{S_1, \ldots, S_n\}$ from it.
S_1
S_2
...
S_n
Image *H* (the halftone version of *S*)
Create *n* transparencies so that each S_i is hidden in region *i* of the transparency r_i
Transparency r_1
Transparency r_2
Transparency r_n

Fig. 1. The flowchart to summarize the idea of encoding.

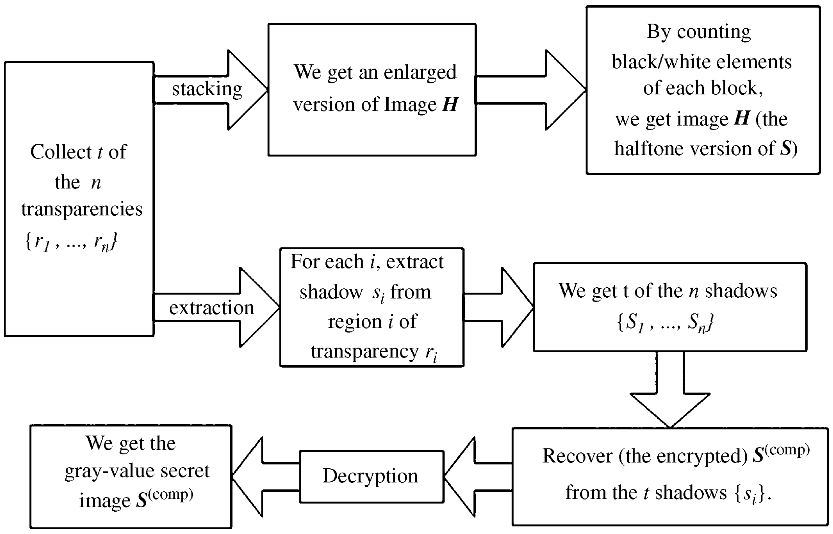

Collect *t* of the *n* transparencies $\{r_1, \ldots, r_n\}$
stacking
We get an enlarged version of Image *H*
By counting black/white elements of each block, we get image *H* (the halftone version of *S*)
extraction
For each *i*, extract shadow s_i from region *i* of transparency r_i
We get t of the *n* shadows $\{S_1, \ldots, S_n\}$
Recover (the encrypted) $S^{(\text{comp})}$ from the *t* shadows $\{s_i\}$.
Decryption
We get the gray-value secret image $S^{(\text{comp})}$

Fig. 2. The flowchart of the decoding algorithm.

is black (white). Therefore, all pixels of the halftone image $\boldsymbol{H}$ can be reconstructed one by one. This recovers $\boldsymbol{H}$ without any loss.

2. For each of the t collected transparencies, extract the $|\boldsymbol{S}|/n$ digits of s_i hidden in region i of the transparency r_i. This can be done by inspecting the block type (it is a value in the range $\left\{0, 1, \ldots, \binom{w+b}{b} - 1\right\}$) for each of the $|\boldsymbol{S}|/n$ blocks in region i of the transparency r_i.

3. Each s_i is now a digit string of $|\boldsymbol{S}|/n$ digits, and each digit is in the range $\left\{0, 1, \ldots, \binom{w+b}{b} - 1\right\}$. Switch each digit string from base $\binom{w+b}{b}$ back to base 257. Each base 257 string is a shadow now.

4. Recover (the encrypted) $\boldsymbol{S}^{(\text{comp})}$ from the t shadows $\{s_i\}$ by using the inverse processing of the PSS (the inverse processing of Eq. (5) to recover the t coefficients $\{a_0, \ldots, a_{t-1}\}$ of a sector k from the t values $\{p(i)\}$, which are formed of the kth digit of the t shadows $\{s_i\}$). Notably, the inverse processing of the PSS (Eq. (5)) can be done by an interpolation method using linear combination of Lagrange polynomials, which is a very common method in numerical analysis field for finding the interpolation polynomial $p(x)$ passing through the t received points $\{(i, p(i))\}$. Interested readers can refer to Ref. [12, Eq. (3)].

5. Decrypt $\boldsymbol{S}^{(\text{comp})}$.

6. Now, the decrypted $\boldsymbol{S}^{(\text{comp})}$ is the desired output. As a remark, the output is just a compressed version of $\boldsymbol{S}$. In general, unless the compression in the encoding phase was a lossless compression, we cannot recover $\boldsymbol{S}$ in an error-free manner.

4. Experiments and comparisons

In our first experiment, the secret image is the image 512 × 512 Lena shown in Fig. 3(a). Its JPEG-compressed version, the so-called image $\boldsymbol{S}^{(\mathrm{comp})}$ in Section 3.2.2, is the 512 × 512 Lena$^{(\mathrm{comp})}$ shown in Fig. 3(b), and the PSNR of Lena$^{(\mathrm{comp})}$ is 39.31 dB (the compression ratio is 6.9). The 512 × 512 halftone image $\boldsymbol{H}$ of Lena is shown in Fig. 4. Assume $(t, n) = (2, 4)$. In other words, the target is that any two of the four generated transparencies can be used for image reconstruction. The basis matrices are the same as in Eq. (1). According to formula (6), the compression ratio must satisfy

$$\mathit{ratio} \geqslant (8n) \Big/ \left(t \times \log_2 \binom{w+b}{b}\right)$$
$$= (8 \times 4)/(2 \times \log_2[4!/(2! \times 2!)]) = 6.2,$$

and our compression ratio 6.9 in Fig. 3(b) of course meets this requirement. Then, using Figs. 3(b) and 4, the $n = 4$ transparencies are created, as shown in Fig. 5. Each 1024 × 1024 transparency is 2 × 2 times bigger than each 512 × 512 image in Figs. 3 and 4, for each fundamental block used to expand a pixel (see Section 2) is 2 × 2 in this experiment. Now, if we stack any two of the four generated 1024 × 1024 black-and-white transparencies, we get the 1024 × 1024 black-and-white image shown in Fig. 6, which looks like a 2 × 2-times enlarged version of the halftone image $\boldsymbol{H}$ shown in Fig. 4. This 1024 × 1024 enlarged version can be utilized to recover the 512 × 512 halftone image $\boldsymbol{H}$, by using Step 1 of the computer-aided decoding algorithm in Section 3.4. On the other hand, if we use the two received transparencies to extract the information hidden in them, we can recover exactly the 512 × 512 gray-value compressed image Lena$^{(\mathrm{comp})}$ shown in Fig. 3(b), as shown in Fig. 7.

In the second experiment, the secret image is the 512 × 512 image "Jet" shown in Fig. 8(a). Assume $(t, n) = (3, 4)$. In other words, the goal is that any three of the four generated transparencies can be used for image reconstruction. The corresponding experimental results are shown in the remaining parts of Fig. 8. By formula (6), the compression ratio must satisfy

$$\mathit{ratio} \geqslant (8n) \Big/ \left(t \times \log_2 \binom{w+b}{b}\right)$$
$$= (8 \times 4)/(3 \times \log_2[6!/(3! \times 3!)]) = 2.5,$$

and our compression ratio 2.5 in Fig. 8(b) of course meets this requirement. Note that Fig. 8(e) is 2 × 3 times larger than the halftone image H because we use 2 × 3 blocks to expand pixels of the halftone image H there. Also note that any two transparencies together $(2 < 3 = t)$ cannot reveal the JPEG version Jet$^{(\mathrm{comp})}$ or the halftone version H.

Notably, if we prefer a lossless recovery, then some lossless image compression algorithms, such as PNG (portable

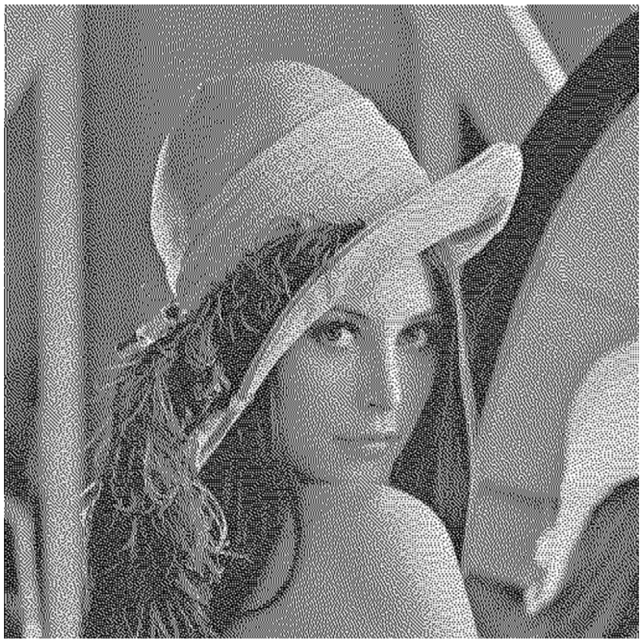


Fig. 4. The halftone version (i.e. the binary image H) of Fig. 3(a).

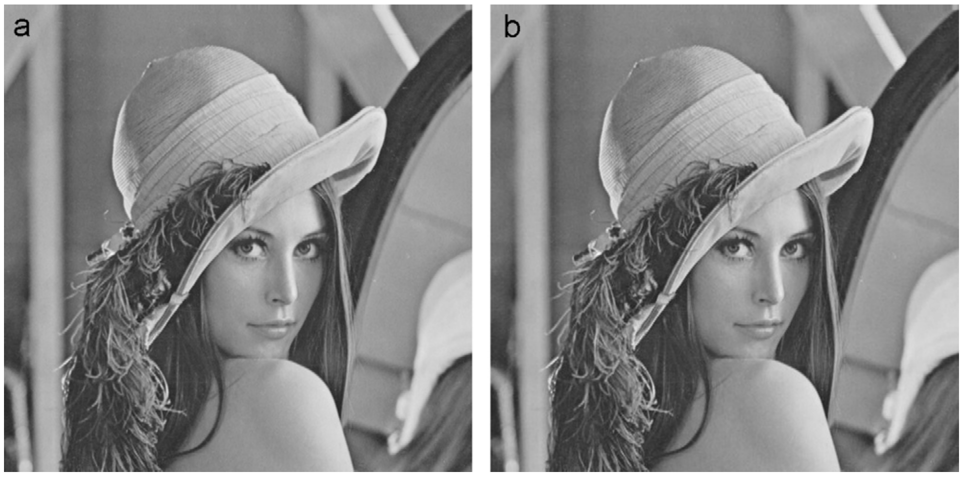


Fig. 3. A gray-value 512 × 512 image Lena (shown in (a)), and its JPEG-compressed version Lena$^{(\mathrm{comp})}$ whose PSNR is 39.31 dB (shown in (b)). The compression ratio between (a) and (b) is $\mathit{ratio} = 6.9$.

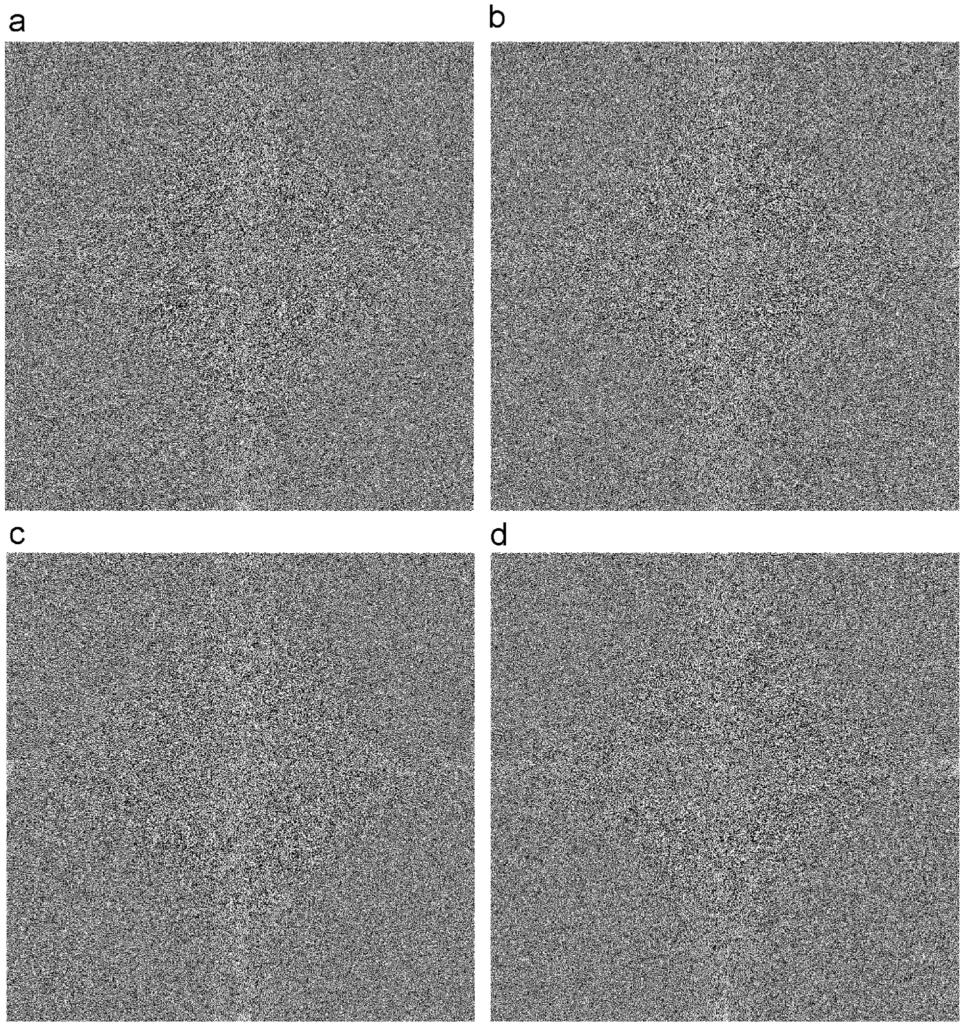


Fig. 5. The $n = 4$ transparencies T0–T3 generated from the pair (Figs. 3(b), 4) in our ($t = 2$, $n = 4$) threshold scheme.

network graphics, see http://www.libpng.org/pub/png/), or lossless JPEG2000, or Maniccam and Bourbakis's method [19], should be applied to obtain a lossless $\boldsymbol{S}^{(\text{comp})}$. Then use our method to share the compressed file (while Ref. [15] was to hide the compressed file in a cover image).

Table 2 lists a comparison between our approach and some other hiding methods reported in recent years (2004–2007). In 2004, Maniccam and Bourbakis [15] introduced an elegant hiding method, which used a SCAN-based image compression and encryption algorithm. The main algorithm consists of two major parts: the compression part and hiding part. In the compression part, the secret image was compressed by using a SCAN-based lossless image compression technology, which will generate a sequence of bit stream. In the hiding part, a matrix called "complexity matrix" was generated; its main purpose is to evaluate the capacity of each pixel (i.e. the number of LSB) of the cover image, then, the compressed file (obtained by the compression part) was embedded in the cover image by looking up the "complexity matrix". Besides this, a SCAN-based encryption algorithm was also utilized to rearrange the cover image and the "complex matrix", in order to enhance the security of the hiding algorithm.

Besides [15], we also compare in Table 2 some other image hiding methods [16–18] reported in 2005–2007. We bypass the detail introduction of [16–18] to save the paper length. In Tests 1 and 2, since Ref. [16] is the only one of which the secret image is as large as the cover image (both 512×512), we particularly do Tests 1′ and 2′ in order to compare with Ref. [16]. Note that our Test 2′ is in fact the experiment described in Figs. 3–7. In Test 2′, we need $t = 2$ shadows to recover Lena (or Jet), and each shadow is a $(512 \times 2) \times (512 \times 2)$ "binary" file. As a result, the $t = 2$ shadows together use $(512 \times 2 \times 512 \times 2) \times \frac{2}{8} = 512 \times 512$ bytes, just identical to the storage space needed for each 512×512 "stego" gray-level image in each test of Refs. [15–18]. Note particularly, in Test 2 of Ref. [16], whose secret image Lena is also 512×512 (as large as secret image), they obtain 35.73 dB when the 512×512 secret Lena is recovered, while ours is 39.3 dB. Notably, Test 1′ is identical to Test 2′, except that we replace the 512×512 secret Lena in Figs. 3–7 by the 512×512 secret Jet. Since we still use $(t, n) = (2, 4)$ and 1024×1024 binary shadows (transparencies),

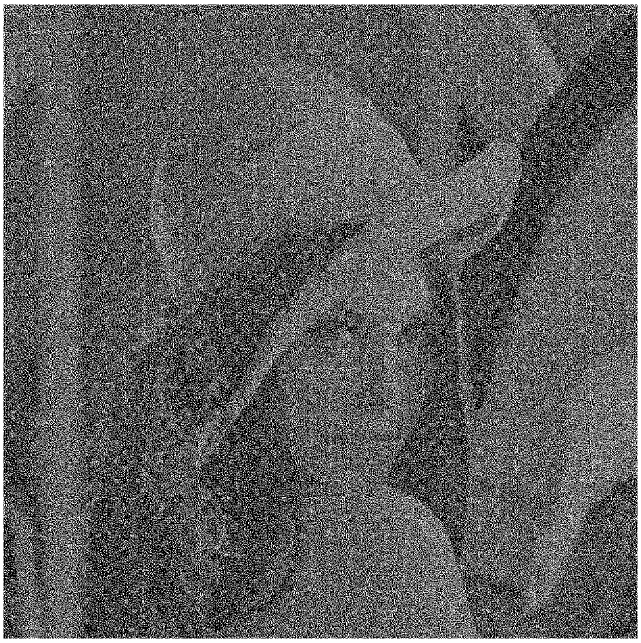


Fig. 6. Stacking "any" two transparencies (e.g. 1st and 3rd transparencies here) yields an enlarged binary image of Lena.

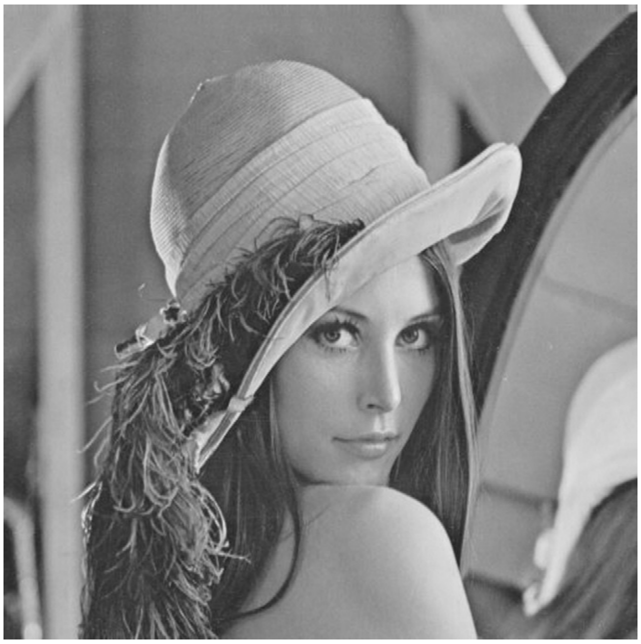


Fig. 7. The gray-value image Lena$^{(\mathrm{comp})}$ (identical to Fig. 3(b)) reconstructed using the information embedded in any two transparencies (the 1st and 3rd transparencies here).

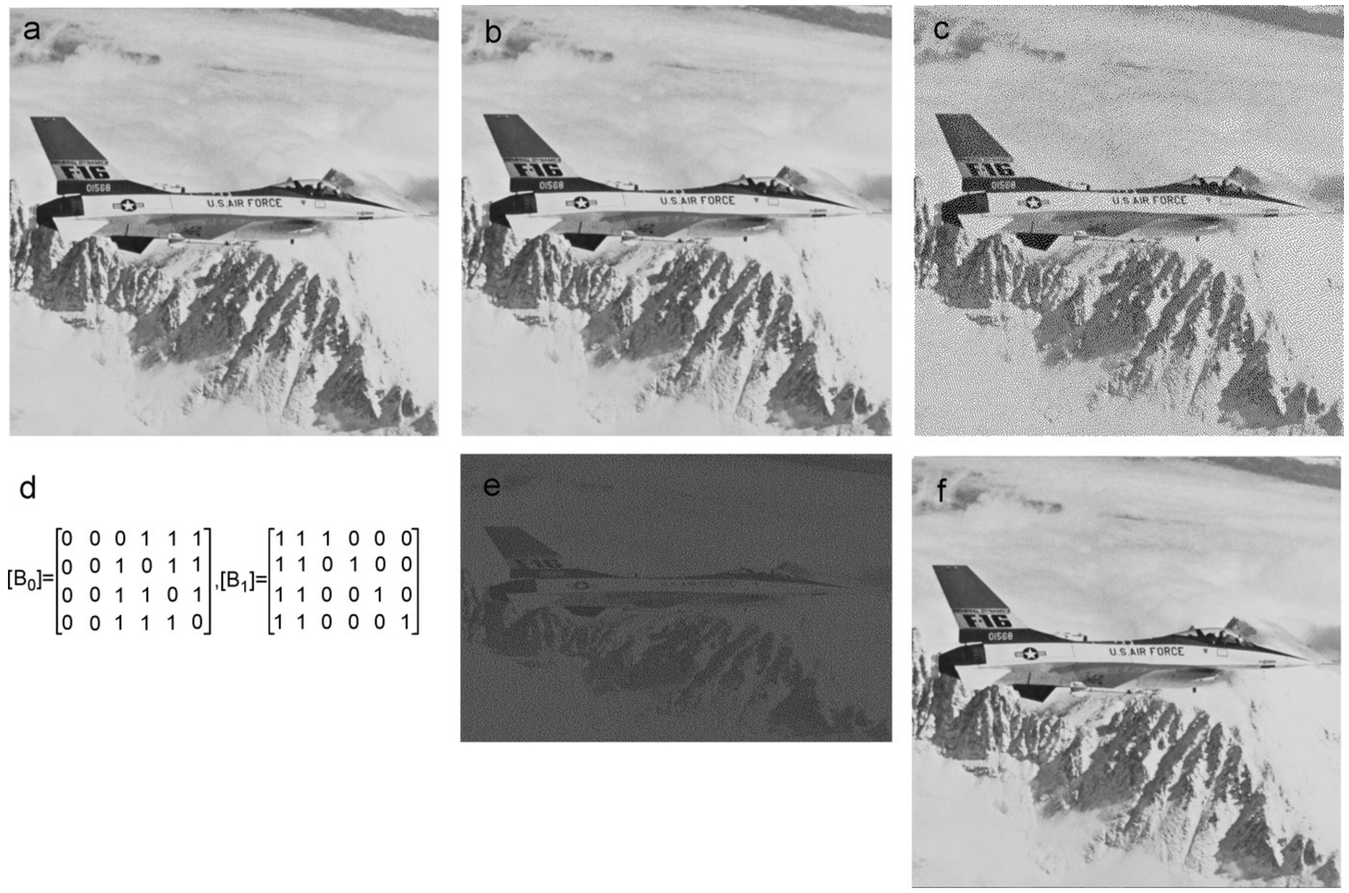
the $t = 2$ shadows needed to recover secret still use together $(512 \times 2 \times 512 \times 2) \times \frac{2}{8} = 512 \times 512$ bytes, and the recovered 512×512 secret Jet is 39.5 dB, still better than the 38.14 dB of Ref. [16]. Of course, some readers might say that, when we compared, why we used t shadows, rather than n shadows, to evaluate the space needed. There are two reasons. The first reason is that we only need t shadows to recover the secret. The second reason is that other methods using a single stego image do not have missing–allowable property (once the only stego image is gone, the secret is gone, but we allow up to $n - t$ shadows to disappear). If we also turn off (do not care) the missing–allowable property, then we can store only the first t (out of the n generated) shadows in t distinct channels and destroy the remaining $n - t$ shadows. Later, just using these t shadows still can obtain both the binary stacking result and recover the gray-level image. Based on these two reasons, we only count the space of t shadows for comparison.

In Table 2, we also do a test called Test 3′ using a smaller size 256×256 secret image Barb (and a 256×256 Lena); the (t, n) is $(4, 4)$, and each binary shadow is $(256 \times 4) \times (256 \times 2)$. Then, the recovered 256×256 secret image is lossless, as in Tests 1 and 2 of Ref. [15], and as in Test 2 of Ref. [18]. The four shadows needed in recovery together use $(256 \times 4) \times (256 \times 2) \times \frac{4}{8} = 512 \times 512$ bytes, as in the cover image's 512×512 bytes of Refs. [15,18]. Finally, when the secret is a middle size 512×256 Jet, our Test 4′ in Table 2 uses $(t, n) = (2, 2)$, the recovered secret is again lossless, and the two binary shadows, each is $512 \times (256 \times 7)$ in size, together use $512 \times 256 \times (1 \times 7) \times \frac{2}{8} = (512)^2 \times 0.875$ bytes for Jet's shadows. The needed space is $(512)^2 \times 0.875$ bytes, which of course can still compete with the $(512)^2$ bytes used in Tests 1 and 2 of Ref. [17] for stego image.

From the above analysis, it can be seen that we can compete with these hiding methods, namely, using similar space to obtain competitive recovery quality. Of course, if we let $n > t$, and store more than t shadows, for example, store $t + 1$ shadows, then we might waste space (the total space becomes $1/t$ times larger in this example), but we can get the missing–allowable advantage. (For those cover-image-based hiding methods, the space will be doubled if they store the stego image twice followed by storing these two copies in two distinct places, in order to be missing–allowable.)

Another property that we have is the stacking-and-see ability if we receive t shadows. Of course, this stacking-and-see function can be turned-off for security reason, if the designers in some applications do not allow the t participants to use "stack-and-see" decoding option. To turn off this option, the designers only have to permute the order of the image pixels before generating transparencies (also see the scan-based encryption introduced in Refs. [15,19]).

The final property is that our security is based on sharing: to obtain the secret, at least t of the n channels (each channel has one transparency) have to be intercepted. To the contrast, the cover-image-based hiding methods' fundamental defence is to use a cover image as a trick to avoid curious attack from hackers. Of course, to increase the security level, additional encryption techniques, such as security key, XOR, or scan-based encryption [19], might be used in both ours and others, as shown in the bottom of Table 2.



$$[B_0]=\begin{bmatrix}0&0&0&1&1&1\\0&0&1&0&1&1\\0&0&1&1&0&1\\0&0&1&1&1&0\end{bmatrix},[B_1]=\begin{bmatrix}1&1&1&0&0&0\\1&1&0&1&0&0\\1&1&0&0&1&0\\1&1&0&0&0&1\end{bmatrix}$$

Fig. 8. The second experiment with $(t,n)=(3,4)$, and the experiment uses 2×3 blocks. (a) is the gray-value 512×512 image Jet; (b) is the JPEG-compressed version Jet$^{(\text{comp})}$ whose PSNR is 47.6 dB (the compression ratio is *ratio* $=2.5$); (c) is the halftone version H of (a); (d) shows the basis matrices being used in the second experiment; (e) is the stacking result using "any three" of the four generated transparencies (here, 1st, 2nd, and 4th transparencies); and (f) is the 47.6 dB Jet$^{(\text{comp})}$ (identical to Fig. (b)) reconstructed using the information embedded earlier in the three transparencies mentioned in (e).

Finally, we add the final column to Table 1, in order to briefly summarize our method when it is compared with two major branches of sharing: the VC approaches [1–7] and PSS approaches [8–14]. We can see from this column that, as expected in our goal and design, the two-layer property is obvious in the proposed VCPSS method. This column also shows that: we can have a much faster decoding (using Layer 1) that the PSS approaches (for example, Refs. [8–14]) do not have; on the other hand, we can also get a better-quality recovered image (obtained in Layer 2) that the VC approaches (for example, Refs. [8–14]) do not have. As for the final row called "size of each shadow", we write that either the length or the width of each "binary" shadow is larger than that of $\boldsymbol{S}$, because each binary shadow is in fact a transparency created using VC on the halftone version $\boldsymbol{H}$ of image $\boldsymbol{S}$. Hence, as in ordinary VC, either the length or the width increases. However, when the binary pixel of a transparency is stored in a computer, only 1 bit (rather than 1 byte) is needed, that is why the number of computer bytes needed can be smaller than that of $\boldsymbol{S}$, because each pixel of the gray-value image $\boldsymbol{S}$ needs 8 bits.

5. Concluding discussions

In this paper, we have proposed a new method which combines two major branches of image sharing: Visual Cryptography (VC) and polynomial-style sharing (PSS). In the decoding issue, this new method is more flexible than applying VC or PSS independently, since our method provides a "two-options" decoding.

In Figs. 3(b) and 7, the PSNR of Lena$^{(\text{comp})}$ is 39.31 dB. If the readers would like to have an image Lena$^{(\text{comp})}$ of better PSNR, then they should use larger blocks (say, 2×3 or 3×3 blocks) to replace the 2×2 blocks here for each transparency.

To explain why using large blocks usually implies a better PSNR, we may analyze the relation between the image compression ratio and the size of the block. In general, the compressed image $\boldsymbol{S}^{(\text{comp})}$ will have a better PSNR if the image compression ratio (a value not less than 1) is smaller (closer to 1). On the other hand, the compression ratio in this paper is constrained by Eq. (6), i.e. $\textit{ratio} \geqslant (8n)/\left(t\times\log_2\binom{w+b}{b}\right)$. Therefore, in order to get a better PSNR, which means a smaller

Table 2
A comparison between ours and some image hiding technologies

	Our method	Ref. [15]	Ref. [16]	Ref. [17]	Ref. [18]
Purpose	To share the secret	To hide the secret in a cover image			
Cover image (Input)	No cover image	Test 1: use 512×512 Lena as the cover Test 2: use 512×512 Baboon as the cover			
Secret image (Input)	Test 1′: to share 512×512 Jet Test 2′: to share 512×512 Lena Test 3′: to share 256×256 Barb (or 256×256 Lena) Test 4′: to share 512×256 Jet	Test 1: hide 256×256 Barb Test 2: hide 256×256 Barb	Test 1: hide 512×512 Jet Test 2: hide 512×512 Lena	Test 1: hide 512×256 Jet Test 2: hide 512×256 Jet	Test 1: hide 256×512 Jet Test 2: hide 256×256 Lena
Output	Several noisy transparencies	A gray value stego image, which looks just like the cover image			
Total space needed in order to recover the secret	Test 1′ (with $(t, n) = (2, 4)$): $(512)^2 \times (2 \times 2) \times 2/8 = (512)^2$ bytes for Jet's shadows Test 2′ (Fig. 5): $(512)^2 \times (2 \times 2) \times \frac{2}{8} = (512)^2$ bytes for Lena's shadows Test 3′ (with $(t, n) = (4, 4)$): $(256)^2 \times (4 \times 2) \times \frac{4}{8} = (512)^2$ bytes for Barb's shadows. So is Lena's Test 4′ (with $(t, n) = (2, 2)$): $512 \times 256 \times (7 \times 1) \times \frac{2}{8} = (512)^2 \times 0.875$ bytes for Jet's shadows	512×512 bytes, i.e. the size of the output stego image, which is identical to the size of the input cover image			
Stego image's PSNR	No PSNR value for the t noisy transparencies because no cover image	Test 1: 43.47 dB Test 2: 39.12 dB	Test 1: 44.14 dB Test 2: 44.16 dB	Test 1: 41.20 dB Test 2: 39.62 dB	Test 1: 31.05 dB Test 2: 44.07 dB
Recovered gray-value secret image quality	Test 1′: 39.5 dB 512×512 Jet Test 2′ (Fig. 7): 39.3 dB 512×512 Lena Test 3′: lossless (either 256×256 Barb or 256×256 Lena) Test 4′: lossless 512×256 Jet	Test 1: lossless 256×256 Barb Test 2: lossless 256×256 Barb.	Test 1: 38.14 dB 512×512 Jet Test 2: 35.73 dB 512×512 Lena	Test 1: lossless 512×256 Jet Test 2: lossless 512×256 Jet	Test 1: lossless 256×512 Jet Test 2: lossless 256×256 Lena
Also has stacking-to-see ability?	Yes	No			
Allow the missing of stego image or some shadows?	Yes, if store more than t of the n generated shadows (total pace is then at least $1/t$ times larger)	Yes, if duplicate the stego image (Total space is then at least two times larger)			
The manners to achieve security	Each channel holds only one shadow. Need t shadows to recover	Hide the secret in the cover image	Hide the secret in a cover image	Hide the secret in a cover image	Hide the secret in a cover image

Table 2 (*Continued*).

	Our method	Ref. [15]	Ref. [16]	Ref. [17]	Ref. [18]
	A key or XOR or other simple functions to encrypt ***S*** (comp)	Use a SCAN-based encryption algorithm [19] Use security key and XOR	Use a security key to select the hiding place in the cover image	Use a security key to encrypt the header file	Use data encryption standard (DES) Use a security key to select hiding place in cover

value of *ratio* is needed, we should try to make the value of the lower-bound term $(8n)/\left(t \times \log_2 \binom{w+b}{b}\right)$ as small as possible. Without the loss of generality, take a look the experiment in Fig. 8, in which $(t, n) = (3, 4)$, and the basis matrices of the experiment were

$$[\mathbf{B_0}] = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \quad \text{and}$$

$$[\mathbf{B_1}] = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Therefore, the compression ratio was constrained by

$$ratio \geqslant (8n) \Big/ \left(t \times \log_2 \binom{w+b}{b}\right)$$
$$= (8 \times 4) \Big/ \left(3 \times \log_2 \binom{3+3}{3}\right) = 2.4680. \qquad (7)$$

That was why the compression ratio was taken as 2.5 in the experiment in Fig. 8.

Now, if we try to redo this $(t, n) = (3, 4)$ experiment by using larger blocks; for instance, by using 4×3 blocks to replace the 2×3 blocks in Fig. 8, then each block has 12 elements instead of six elements. As a result, the basis matrices should have 12 columns, rather than six columns. (As for the number of rows, it is still of $n = 4$ rows, for we did not change the value of n.) Let the two new basis matrices be

$$[\mathbf{B}_0^{(\text{new})}] = \left[\begin{array}{cccccc|cccccc} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right]$$

and

$$[\mathbf{B}_1^{(\text{new})}] = \left[\begin{array}{cccccc|cccccc} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right].$$

The width of the new basis matrices is two times larger than that of the original basis matrices. Moreover, in the old pair of basis matrices, each row has $w = 3$ zeros and $b = 3$ ones. In the new pair of basis matrices, each row has $w = 6$ zeros and $b = 6$ ones. Therefore, with these two new basis matrices, the new compression ratio is constrained by

$$ratio \geqslant (8n) \Big/ \left(t \times \log_2 \binom{w+b}{b}\right)$$
$$= (8 \times 4) \Big/ \left(3 \times \log_2 \binom{6+6}{6}\right) = 1.0827. \qquad (8)$$

This lower bound for the compression ratio is so low that we can take *ratio* as 1.09 and make the new compressed image $\boldsymbol{S}^{(\text{comp})}$ identical to the original image $\boldsymbol{S}$, for a compression ratio so close to 1 means the compression can be lossless.

From the above analysis, we can see that the key factor to get a high-quality compressed image $\boldsymbol{S}^{(\text{comp})}$ before sharing is to make the value of the term $\log_2 \binom{w+b}{b}$ in the denominator as large as possible. To achieve this, there are two things we should know. First, for a given value of the sum $(w + b)$, the value of $\binom{w+b}{b}$ is larger if we let w and b be equal (or, almost equal, if $(w + b)$ is an odd number). Second, if we let $w = b$ (without the loss of generality, we only discuss the case when $w+b$ is an even value), then $\binom{w+b}{b} = \binom{b+b}{b} = (2b)!/(b!)(b!) = (b + b)(b + b - 1) \ldots (b + 1)/(b)(b - 1) \ldots (1)$, which is an increasing function of b (the larger the value of b, the larger the value of $\binom{2b}{b}$). Therefore, if we can use larger blocks, then the sum value $(w + b)$ is larger because $(w + b)$ is the total number of elements contained in a block. As a result, if we let w and b be (almost) equal, then using larger blocks means larger value of $\binom{w+b}{b}$, which in turn means smaller value of the term *ratio*, because the term $\log_2 \binom{w+b}{b}$ is in the denominators of Eqs. (6)–(8). And finally, as stated earlier, smaller values of *ratio* means better image quality of the compressed version $\boldsymbol{S}^{(\text{comp})}$ of $\boldsymbol{S}$.

6. An application

Our two-layer decoding system can also be used in business in the following way that balances between convenience and security: (i) any t of the n lower-rank employees can gather together to unveil a vague black-and-white version of the image; while (ii) the manager of these employees can unveil further a fine gray-value version of the image using the encryption key that only he knows. In the above, (i) is for the convenience of the daily meeting between the employees (the meeting does not need the attendance of the manager, although a meeting with less than t employees cannot reveal anything). Meanwhile, (ii) is for the company owner to prevent a high-quality image from being sold in a black market (neither the t lower-rank employees (each of them holds a shadow) nor the manager alone (who holds the encryption key rather than a shadow) can obtain the high-quality version, unless the two sides cooperate).

Acknowledgments

The anonymous reviewers are highly appreciated for the valuable suggestions to make this paper more readable. The work was supported by National Science Council, Taiwan, under project NSC 95-2221-E-009-256.

Appendix

A possible way to generate a pair of basis matrices $[\mathbf{B_0}]$ and $[\mathbf{B_1}]$ for an (t, n) threshold system. (This self-explained appendix is a modified part derived from Ref. [2].)

Input: An integer threshold t, and an integer n indicating the number of produced transparencies ($t \leqslant n$).

Output: A pair of basis matrices $[\mathbf{B_0}]$ and $[\mathbf{B_1}]$ (both matrices are formed of 0's and 1's) satisfying the following requirements:

(i) Each matrix has n rows.
(ii) Stacking any k ($k < t$) of the n rows of the matrix $[\mathbf{B_0}]$ (or $[\mathbf{B_1}]$, respectively) gets a row vector called the White-Row-Vector (the Black-Row-Vector, respectively). For a given $k < t$, the number of 1's in any Black-Row-Vector is identical to that in any White-Row-Vector.
(iii) Stacking any t of the n rows of the matrix $[\mathbf{B_0}]$ (or $[\mathbf{B_1}]$, respectively) gets a row vector called the White-Row-Vector (the Black-Row-Vector, respectively). Any Black-Row-Vector contains more 1's than any White-Row-Vector does.

Steps: 1. Arbitrarily choose $(n - t)$ integers $\{h_j | 0 \leqslant j \leqslant n - t - 1\}$ and an integer $h_{n-t} > 0$.

2. Compute the integers $\{a_i | 0 \leqslant i \leqslant n\}$ by the formula.

$$a_i = \sum_{j=0}^{\min\{i, n-t\}} (-1)^{i+j} h_j \frac{(i+j)!}{(i!)(j!)}. \tag{A.1}$$

3. Create $\{G_0, \ldots, G_n\}$. Here, each G_i is a matrix having n rows and $\binom{n}{i}$ columns; no two columns of G_i are identical, and each of these $\binom{n}{i}$ columns is exactly one of the $\binom{n}{i}$ permutations of the n elements of the first column of G_i. (The first column of G_i is formed of i consecutive 1's followed by $(n - i)$ consecutive 0's.)

4. Initially, set both $[\mathbf{B_0}]$ and $[\mathbf{B_1}]$ to empty set; also set $a_i = a_0$. Then,

(i) If ($a_i = 0$), then do nothing for this a_i, just go to (iv).
(ii) If ($a_i > 0$), then repeatedly append G_i into $[\mathbf{B_0}]$ (repeat $|a_i|$ times).
(iii) If ($a_i < 0$), then repeatedly append G_i into $[\mathbf{B_1}]$ (repeat $|a_i|$ times).
(iv) If $i = n$, go to Step 5, else, increment i by 1 and go to (i).

5. $[\mathbf{B_0}]$ and $[\mathbf{B_1}]$ are now the desired output.

Example 1. In the ($t = 2$, $n = 4$) case, if a reader uses $\{h_0, h_1, h_2\}=\{3, 3, 2\}$ as his arbitrary setting for the $\{h_0, h_1, h_2\}$ in Step 1 above, then he will proceed as follows:

Step 1: Let $\{h_0, h_1, h_2\} = \{3, 3, 2\}$.

Step 2: By formula (A.1), compute and obtain $\{a_0, a_1, a_2, a_3, a_4\} = \{3, 0, -1, 0, 3\}$.

Step 3: $G_0, \ldots, G_4$ are, respectively,

$$G_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad G_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$G_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix},$$

$$G_3 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}, \quad G_4 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Step 4: Since $\{a_0, a_1, a_2, a_3, a_4\}=\{3, 0, \underline{-1}, 0, 3\}$; from those positive a_i, we get

$$[\mathbf{B_0}] = (G_0 \text{ repeats three times } (\text{for } a_0 = 3), \text{ followed by repeating } G_4 \text{ three times (for } a_4 = 3)), \text{ i.e.}$$

$$[\mathbf{B_0}] = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix},$$

similarly, from those negative a_i, we get

$$[\mathbf{B_1}] = (G_2 \text{ appears } |a_2| = |-1| = 1 \text{ time}), \text{ i.e.}$$

$$[\mathbf{B_1}] = G_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Step 5: $[\mathbf{B_0}]$ and $[\mathbf{B_1}]$ are now the desired output.

Remark. In Example 1 above, stacking any two rows of $[\mathbf{B_0}]$ yields a row having three 1's, while stacking any two rows of $[\mathbf{B_1}]$ yields a row having five 1's (and hence, "darker" than stacking any two rows of $[\mathbf{B_0}]$). Also note that each row of each matrix has six elements; therefore, each block is 2×3 (or 3×2) rather than the 2×2 used in Section 2 and in the experiment for Lena (Figs. 3–7). Now, since $\binom{b+w}{b} = \binom{6}{3} = 20 > 6 = \binom{4}{2}$, the JPEG compression ratio mentioned in Section 3.2.2 is thus smaller (less strict) now, and the quality of the JPEG image Lena$^{(\text{comp})}$ in Figs. 3(b) and 7 will thus becomes better (has higher PSNR value). The price of getting this better-quality gray-value image Lena$^{(\mathrm{comp})}$ is that the stacked black-and-white image in Figs. 6 will become larger in width (just like Fig. 8(e) does) because of the use of 2×3 blocks.

References

[1] M. Naor, A. Shamir, Visual cryptography, Cryptology-Eurocrypt '94, Lecture Notes in Computer Science, vol. 950, Springer, Berlin, 1995, pp. 1–12.

[2] G. Ateniese, C. Blundo, A. De Santis, D.R. Stinson, Contrast optimal threshold visual cryptography schemes, SIAM J. Discrete Math. 16 (2003) 224–226.

[3] C.N. Yang, New visual secret sharing schemes using probabilistic method, Pattern Recognition Lett. 25 (4) (2004) 481–494.

[4] C.N. Yang, T.S. Chen, Size-adjustable visual secret sharing schemes, IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E88-A (9) (2005) 2471–2474.

[5] C.C. Lin, W.H. Tsai, Visual cryptography by dithering techniques, Pattern Recognition Lett. 24 (2003) 349–358.

[6] R. Lukac, K.N. Plataniotis, Bit-level based secret sharing for image encryption, Pattern Recognition 38 (5) (2005) 767–772.

[7] Y.C. Hou, Visual cryptography for color images, Pattern Recognition 36 (2003) 1619–1629.

[8] C.C. Thien, J.C. Lin, Secret image sharing, Comput. Graphics 26 (5) (2002) 765–770.

[9] C.C. Thien, J.C. Lin, An image-sharing method with user-friendly shadow images, IEEE Trans. Circuits Syst. Video Technol. 13 (12) (2003) 1161–1169.

[10] Y.S. Wu, C.C. Thien, J.C. Lin, Sharing and hiding secret images with size constraint, Pattern Recognition 37 (2004) 1377–1385.

[11] J.B. Feng, H.C. Wu, C.S. Tsai, Y.P. Chu, A new multi-secret images sharing scheme using Lagrange's interpolation, J. Syst. Software 76 (2005) 327–339.

[12] C.C. Lin, W.H. Tsai, Secret image sharing with steganography and authentication, J. Syst. Software 73 (2004) 405–414.

[13] S.K. Chen, J.C. Lin, Fault-tolerant and progressive transmission of images, Pattern Recognition 38 (2005) 2466–2471.

[14] A. Shamir, How to share a secret, Commun. ACM 1979 22 (11) (1979) 612–613.

[15] S.S. Maniccam, N.G Bourbakis, Lossless compression and information hiding in images, Pattern Recognition 37 (3) (2004) 475–486.

[16] R.Z. Wang, Y.D. Tsai, An image-hiding method with high hiding capacity based on best-block matching and k-means clustering, Pattern Recognition 40 (2) (2007) 398–409.

[17] S.L. Li, K.C. Leung, L.M. Cheng, C.K. Chan, A novel image-hiding scheme based on block difference, Pattern Recognition 39 (6) (2006) 1168–1176.

[18] S.J. Wang, Steganography of capacity required using modulo operator for embedding secret image, Appl. Math. Comput. 164 (2005) 99–116.

[19] S.S. Maniccam, N.G. Bourbakis, Lossless image compression and encryption using SCAN, Pattern Recognition 34 (6) (2001) 1229–1245.

About the Author—SIAN-JHENG LIN was born in Taiwan, Republic of China. He received his B.S. and M.S. degree in Computer Science from the National Chiao Tung University in 2004 and 2006, respectively. He is currently a Ph.D. candidate in Computer Science Department of the National Chiao Tung University. His recent research interests include pattern recognition and image processing.

About the Author—JA-CHEN LIN was born in Taiwan, Republic of China. He received his B.S. degree in computer science in 1977 and M.S. degree in applied mathematics in 1979, both from the National Chiao Tung University, Taiwan. In 1988, he received his Ph.D. degree in mathematics from the Purdue University, USA. In 1981–1982, he was an instructor at the National Chiao Tung University. From 1984 to 1988, he was a graduate instructor at the Purdue University. He joined the Department of Computer Science at the National Chiao Tung University in August 1988 and is currently a professor there. His recent research interests include pattern recognition and image processing. Dr. Lin is a member of the Phi-Tau-Phi Scholastic Honor Society.