Part II

Algorithm and System Designs
Chapter 5

Data Hiding in Binary Images

5.1 Introduction and prior art

An increasingly large number of digital binary images have been used in everyday life. Handwritten signatures captured by electronic signing pads are digitally stored and are being used as the records for credit card payment by many department stores in the U.S. and for parcel delivery by major courier services such as the United Parcel Service (UPS). Word processing software like Microsoft Word allows a user to store his/her signature in a binary image file for inclusion at specified locations of a document. The documents signed in such a way can be sent directly to a fax machine or be distributed across a network. The unauthorized use of a signature, such as copying it onto an unauthorized payment, is becoming a big concern. In addition, a variety of important documents, such as social security records, insurance information, and financial documents, have also been digitized and stored. Because it is easy to copy and edit digital images via software tools, the annotation and authentication of binary images as well as the detection of tampering are very important. This chapter discusses data hiding techniques for these purposes as an alternative to or in conjunction with the cryptographic authentication approach. While we expect the
embedded data to have some robustness against minor distortion and preferably to withstand printing and scanning, the robustness of embedded data against intentional removal or other obliteration is not a primary concern because there is little incentive to do so in the targeted applications of annotation and authentication.

Most prior works on image data hiding are for color or grayscale images in which the pixels may take on a wide range of values. For those images, changing pixel values by a small amount is generally unnoticeable under normal viewing conditions. This property of human visual system plays a key role in watermarking of perceptual media data [44, 46]. For images in which the pixels take on only a limited number of values, hiding data without causing visible artifacts becomes more difficult. In particular, flipping white or black pixels that are not on the boundary is likely to introduce visible artifacts in binary images. Before we present our solutions to the challenging issues of hiding data in binary images, we shall give a brief review of the prior art.

Several methods for hiding data in specific types of binary images have been proposed in literature. Matsui et al [106] embedded information in dithered images by manipulating the dithering patterns and in fax images by manipulating the run-lengths. Maxemchuk et al [108] changed line spacing and character spacing to embed information in textual images for bulk electronic publications. These approaches cannot be easily extended to other binary images and the amount of data that can be hidden is limited. In [107], Koch and Zhao proposed a data hiding algorithm which enforces the ratio of black vs. white pixels in a block to be larger or smaller than 1. Although the algorithm aims at robustly hiding information in binary image, it is not robust enough to tolerate many distortions/attacks, neither is it secure enough to be directly applied for authentication or other fragile use. Only a limited number
of bits can be embedded because the particular enforcing approach has difficulty in
dealing with blocks that have low or high percentage of black pixels. In spite of
these weaknesses, their idea of enforcing properties of a group of pixels via the local
manipulation of a small number of pixels can be extended as a general framework
of data embedding. Another approach of marking a binary document is proposed
in [110] by treating a binary image as a grayscale one and by manipulating the
luminance of dark pixels slightly so that the change is imperceptible to human eyes
yet detectable by scanners. This approach, targeted on intelligent copier systems, is
not applicable to bi-level images hence is beyond the scope of this paper. The bi-level
constraint also limits the extension of many approaches proposed for grayscale or color
images to binary images. For example, applying the spread spectrum embedding, a
transform-domain additive approach proposed by Cox et al [44], to binary image could
not only cause annoying noise on the black-white boundaries, but also have reduced
robustness hence limited embedding capacity due to the post-embedding binarization
that ensures the marked image is still a bi-level one [109]. For these Type-I additive
embeddings, hiding a large amount of data and detecting without the original binary
image is particularly difficult. In summary, these previously proposed approaches
either cannot be easily extended to other binary images, or can only embed a small
amount of data.

We propose a new approach that can hide a moderate amount of data in general
binary images, including scanned text, figures, and signatures. The hidden data can
be extracted without using the original unmarked image, and can also be extracted
after high quality printing and scanning with the help of a few registration marks.
The approach can be used to verify whether a binary document has been tampered
with or not, and to hide annotation labels or other side information.
We shall discuss three key issues of hiding data in binary image in Section 5.2, along with our proposed solutions. In Section 5.3, we demonstrate three applications and the experimental results of the proposed data hiding approach. A variety of discussions are given in Section 5.4, including robustness analysis and security considerations. Issues such as recovering hidden data from high quality printing-and-scaning is also addressed.

5.2 Proposed Scheme

There are two basic ways to manipulate binary images for the purpose of data hiding, namely, by changing the values of individual pixels and by changing a group of pixels. The first approach flips a black pixel to white or vice versa. The second approach modifies such features as the thickness of strokes, curvature, and relative positions, which generally depends more on the types of images (e.g., text, sketches, signatures, etc.). Since the number of parameters that can be changed by the second approach is limited, especially under the requirements of blind detection (i.e., without using the original image in detection) and invisibility, the amount of data that can be hidden is usually limited except for special types of images.

We focus in this paper on using the first approach. An image is partitioned into blocks and several bits are embedded in each block by changing some pixels in that block. For simplicity, we shall show how to embed one bit in each block. Three issues will be discussed below: (1) how to select pixels for modification so as to introduce as little visual artifacts as possible, (2) specific means to embed data in each block using these flippable pixels, and (3) why to embed the same number of bits in each block and how to enhance the efficiency. The entire process of embedding and extraction
is illustrated in Fig. 5.1.

Figure 5.1: Block diagram of the embedding and extraction process in binary images for authentication and/or annotation.

5.2.1 Flippable Pixels

While a human visual model is a key element in data hiding systems, there is little discussion on a human visual model for binary images. A simple criterion, proposed in [107], is to flip boundary pixels for high contrast image such as text image and to only create rather isolated pixels for dithered image. Our work takes the human perceptual factor into account by studying the flippability of each pixel. More specifically, we examine the pixel and its neighbors to establish a continuous score of how unnoticeable such a change will be. The score is from 0 to 1 with 0 meaning absolutely no flipping. Flipping pixels with higher scores generally introduces less artifacts than flipping a lower one.

Manual score assignment, though possible, has two weaknesses. First, except for small neighborhood such as $3 \times 3$, storing the score of every pattern involves a non-trivial amount of storage – storage on the order of mega bytes is needed for a $5 \times 5$ neighborhood and the storage increases exponentially as the neighborhood gets larger.
Second, as a pure subjective process, the outcomes of manual score assignment could vary significantly from person to person because different persons may emphasize different visual artifacts. This is especially the case when the score is preferably continuous or to involve many discrete levels.

To overcome the problems with manual score assignment, we look for causes that make some flipping more visible to human eyes and measure them in a subjective way. The continuous score can be obtained by applying a set of perceptual rules on these raw measures. In our work, the score is hierarchically determined and is arrived at by considering the change in smoothness and connectivity. The smoothness is measured by the horizontal, vertical, and diagonal transitions in a local window (e.g., $3 \times 3$), and the connectivity is measured by the number of the black and white clusters. For example, the flipping of the center pixel in Fig. 5.2(b) is more noticeable than that in Fig. 5.2(a) because the connectivity of (b) changes and human eyes are sensitive to this change. In this manner, we obtain a list of all $3 \times 3$ patterns ordered in terms of how unnoticeable the change of the center pixel will be. The list can be implemented using a look-up table. We then look at a larger neighborhood like $5 \times 5$ to refine the score. Special cases are also handled in such large neighborhood so as to avoid introducing noise on special patterns such as sharp corners. The details of our proposed score computation can be found in the appendix of this chapter (Sec. 5.6).

### 5.2.2 Mechanics of Embedding

When designing an embedding mechanism, it is important to consider how to extract the embedded data without the original image. Directly encoding the hidden information in flippable pixels (e.g., set to black if to embed a “0” and to white if to embed a “1”) may not work since the embedding process may change a flippable pixel
Figure 5.2: Two examples of $3 \times 3$ neighbourhood, for which flipping the center pixel to white in (a) is less noticeable than that in (b).

Figure 5.3: If assuming only black pixels that are directly adjacent to white pixels are considered as “flippable”, the boundary pixel indicated by an arrow becomes a “non-flippable” one after embedding. This simple example demonstrates that directly encoding the hidden information in flippable pixels may not work since the embedding process may change a flippable pixel in the original image to a pixel that may no longer be considered as flippable.

In the original image to a pixel that may no longer be considered as flippable. As a simple example, suppose only black pixels that are directly adjacent to white pixels are considered as “flippable”, and the flippable pixel marked by thick boundary in Fig. 5.3(a) is changed to white to carry a “1”. Fig. 5.3(b) shows that after embedding, this pixel is no longer considered as flippable if using the same rule. In this case, it is hard for detector to correctly identify which pixel carries what hidden information without knowing the original image.

Instead of directly encoding the hidden information in flippable pixels, we apply the Type-II embedding discussed in Chapter 3. That is, we embed the data by
manipulating flippable pixels so that a certain relationship on features of a group of
pixels is enforced. One possible feature is the total number of black pixels. To embed
a “0” in a block, we may change some pixels so that the total number of black pixels
in that block is an even number. Similarly, to embed a “1”, the number of black pixels
is enforced to an odd number. An alternative approach is to choose a “quantization”
step size $Q$ and to force the total number of black pixels in a block to be $2kQ$ (for some
integer $k$) in order to embed a “0”, and to be $(2k+1)Q$ to embed a “1”. As discussed
in Chapter 3, larger $Q$ gives higher robustness against noise because any perturbation
smaller than $Q/2$ will not affect the accuracy in decoding; however, as a tradeoff, the
changes introduced by the embedding process also increases and the image quality
may be reduced. The “odd-even” method can be viewed as a special case of the table
lookup approach which is similar to those in [78, 163] and in Chapter 7. These two
approaches are illustrated in Fig. 5.4, where each possible quantized number of black
pixels per block is mapped to 0 or 1. The marked image is generated by manipulating
pixels with high flippability score in such a way that the number of black pixels in
each block is enforced to match the bit to be embedded via a prescribed mapping.
That is,

$$v'_i = \arg\min_{x:T(x)=b_i,x=kQ} |x - v_i|$$

(5.1)

where $v_i$ is the $i^{th}$ feature to be enforced (in the above case, the total number of
black pixels of the $i^{th}$ block), $v'_i$ is the feature value after embedding, $b_i$ is the bit to
be embedded in $i^{th}$ feature, and $T(\cdot)$ is a prescribed mapping from feature values to
hidden data values $\{0,1\}$, which may be represented in closed-form or by a lookup
table. Detection is done by checking the enforced relationship - for the above cases,
to examine the odd/even properties, or to perform a table lookup. That is,

$$\hat{b}_i = T(v''_i)$$

(5.2)
where $v_i''$ is the feature extracted from the $i^{th}$ block of a test image, and $\hat{b}_i$ is the estimated value of the embedded bit in the $i^{th}$ block. If one bit is repeatedly embedded in more than one block, majority voting is performed to determine which bit has been hidden. More sophisticated coding than simple repetition may be used to enhance the performance.

Figure 5.4: Illustration of odd-even mapping and table lookup mapping from the quantized number of black pixels per block to the binary data to be embedded. One bit can be embedded in a block by enforcing the number of black pixels to a value that matches the bit to be embedded via a prescribed mapping. The enforcement involves the change of flippable pixels in the block, if necessary.

While other relationship enforcing techniques are certainly possible, we shall in this chapter, for simplicity of discussion, use the enforcing of odd or even number of black pixels.

### 5.2.3 Uneven Embedding Capacity and Shuffling

As outlined earlier, we embed multiple bits by dividing an image into blocks and hiding one bit in each block via the enforcement of the odd-even relationship. However, the distribution of flippable pixels may vary dramatically from block to block in a binary image. No data can be embedded in the white or black uniform regions, while regions with text, drawing, and dithered images may have quite a few flippable
pixels, especially on the non-smooth boundary. This uneven embedding capacity can be seen from Fig. 5.5 where the pixels with high flippability scores, indicated by black dots, are on the rugged boundaries.

Figure 5.5: A binary image (top) and its pixels with high flippability scores (bottom, shown in black).

General approaches to handling uneven embedding capacity have been discussed in Chapter 4. Regarding the uneven embedding capacity in a binary image, using variable embedding rate from block to block is not feasible because (1) a detector has to know exactly how many bits are embedded in each block, and any mistake in estimating the number of embedded bits is likely to cause errors in decoding the hidden data for the current block and the error may propagate to the other blocks, and (2) the overhead for conveying this side information via embedding is quite significant and could be even larger than the actual number of bits that can be hidden. For these reasons, we adopt constant embedding rate (i.e., to embed the same number of bits in each region) and use shuffling to equalize the uneven embedding capacity from region to region.

As shown in Fig. 5.6, the flippable pixels distribute more evenly after a random permutation of all pixels. This is also illustrated in the histogram of the number of flippable pixels in one $16 \times 16$-pixel block (Fig. 5.7). Before shuffling, the distribution
Figure 5.6: Distributions of flippable pixels per 16x16-pixel block of the binary image in Fig. 5.5, before shuffling (top) and after shuffling (bottom).

Figure 5.7: Histogram of flippable pixels per 16x16-pixel block of the binary image in Fig. 5.5, before shuffling (solid line) and after shuffling (dotted-dash line).
extends from 0 to 40 flippables per block and that about 20% of the blocks do not have any flippable pixels. This implies that either we embed nothing in those blocks or we have to introduce significant artifacts to hide data there. The distribution after shuffling, shown as the dotted line, concentrates from 10 to 20, and ALL shuffled blocks have flippable pixels. This equalization function of shuffling has been analyzed in Chapter 4. Plugging into Eq. 4.1 and Eq. 4.2 the parameters of the binary signature image of Fig. 5.5:

\[
\begin{align*}
\text{block size} & \quad q = 16 \times 16 \\
\text{image size} & \quad S = 288 \times 48 \\
\text{block number} & \quad N = S/q = 18 \times 3 \\
\text{flippable percentage} & \quad p = 5.45\%
\end{align*}
\]

we compute the mean and the standard deviation of the histogram. The analytic results are shown in Fig. 5.8, along with the simulation results from 1000 random shuffles. Table 5.2.3 also shows the behavior of blocks with no or few flippables, which are of most concern for data hiding problems. We can see that the analysis and simulation conform with each other very well, and the percentage of blocks with no or few flippables is extremely low. For the problem that we may encounter a small number of blocks with no flippable pixel to carry hidden information, applying error correction encoding with a little correction capability would be sufficient to handle this. As can be seen from the block diagram in Fig. 5.1, the embedding of one bit per block of fixed size described in Section 5.2.2 is performed in the shuffled domain, and inverse shuffling is performed to get a marked image.

We have also discussed in Chapter 4 that shuffling does not produce more flippable pixels. Instead, it dynamically assigns the flippable pixels around the active regions and rugged boundaries to carry more data than the less active regions, yet without the
Figure 5.8: Analysis and simulation of the statistical behavior of shuffling for the binary image in Fig. 5.5.

Table 5.1: Analysis and simulation of the blocks with no or few flippable pixels before and after shuffling for the binary image in Fig. 5.5.

<table>
<thead>
<tr>
<th></th>
<th>before shuffle</th>
<th>mean after shuffle</th>
<th>std after shuffle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>analysis</td>
<td>simulation</td>
</tr>
<tr>
<td>( m_0/N ) (0th bin)</td>
<td>20.37%</td>
<td>5.16x10^{-5} %</td>
<td>0 %</td>
</tr>
<tr>
<td>( m_1/N ) (1st bin)</td>
<td>1.85%</td>
<td>7.77x10^{-4} %</td>
<td>0 %</td>
</tr>
<tr>
<td>( m_2/N ) (2nd bin)</td>
<td>5.56%</td>
<td>5.81x10^{-3} %</td>
<td>5.56x10^{-3} %</td>
</tr>
</tbody>
</table>
need of specifying side information that is image dependent. Shuffling also enhances security since the shuffling table or a key for generating the table is needed to correctly extract the hidden data.

5.3 Applications and Experimental Results

In this section, we present three applications of the proposed data hiding method for binary images along with experimental results.

5.3.1 “Signature in Signature”

We mentioned before that unauthorized use is a potential concern for the increasingly popular use of digitized signature. “Signature in Signature” has been proposed as a tool for annotating the signer’s signature with the data that is related to the signed documents, so that the unauthorized use of a signature can be detected [111]. Here the second “signature” refers to the actual digital version of a person’s signature, while the first “signature” refers to a checksum related to the document content or other annotation information. The data hiding method proposed in this paper can be applied to annotating a signature in such applications as faxing signed documents and storing digitized signatures as transaction records. Compared with the traditional cryptographic authentication approach [11] that has been used in secure communication, the proposed data embedding based approach has the advantage of being user-friendly, easily visualized, and integrating the authentication data with the signature in a seamless way, hence is suitable for the general public.

An example is demonstrated in Fig. 5.9, in which up to 7 characters (approximately 50 bits) can be embedded in a \(287 \times 61\) signature of Fig. 5.9(top). The
embedding rate is 1 bit per block of 320 pixels. Fig. 5.9(middle), which has 7 letters embedded, differs very little from the original one, as indicated by black pixels in Fig. 5.9(bottom) \(^1\).

\[\text{Figure 5.9: “Signature in Signature”. (top) the original image, (middle) a marked copy with 7 letters (approximately 50 bits) embedded in, (bottom) the difference between the original and the marked (shown in black).}\]

\(^1\)The gray areas in Fig. 5.9(bottom) and Fig. 5.11(d), visualizing the strokes and the background, respectively, are for assisting viewers to associate the difference between the original and the marked image with their precise location in the images. They don’t indicate the pixelwise differences that are indicated by black pixels.
5.3.2 Invisible Annotation for Line Drawings

In artistic applications, one would usually prefer to annotate the artwork with information like the creation date and location in such a way that the annotation data interfere with perceptual appreciation to the minimal extent. Our proposed approach can be used to invisibly annotating artistic line drawings like the binary comic picture (120 × 150) shown in Fig. 5.10. In this example, a character string of date information “01/01/2000” is embedded in Fig. 5.10(middle). We can see that the annotation does not interfere with perceptual appreciation in any perceivable way.

Figure 5.10: Invisible annotation for line drawings: (left) the original image, (middle) a marked copy with 10-letter date information (70 bits) embedded in, (right) the difference between the original and the marked (shown in black).

5.3.3 Tamper Detection for Binary Document

As we have mentioned, a large number of important documents have been digitized and stored for records. Because it is easy to edit digital images, the authentication of these digital documents as well as the detection of possible tampering is a very important concern. The data hiding techniques proposed in this paper can be applied
for such purposes, as an alternative to or in conjunction with the cryptographic
authentication approach.

The basic idea of authentication is the same as that for grayscale and color im-
age [163]. Data is embedded in an image in such a fragile way that it will be obliterated
if the image is altered and/or it no longer matches some properties of the image. The
hidden data may be an easily recognized pattern and/or some features/digest related
to the content of host image. Shown in Fig. 5.11(a) is a part of a U.S. Patent, con-
sisting of $1000 \times 1000$ pixels. This binary image contains a variety of patterns such
as texts, drawings, lines, and bar codes. Fig. 5.11(b) is a visually identical figure, but
with 976 bits embedded in it using the proposed techniques. In this particular exam-
ple, 800 bits of the embedded data forms a “PUEE” pattern shown in Fig. 5.11(g).
If the date “1998” on the top is changed to “1999”, the extracted data will be the
random pattern shown in Fig. 5.11(g) and significantly different from the originally
embedded one. This is an indication that alteration was made on the document.

5.4 Discussions

In this section, we will discuss the robustness and security issues of the proposed
scheme. Other considerations associated with shuffling, such as the methods for
handling bad shuffles and for adaptively choosing the block size, can be found in
Chapter 4.

5.4.1 Analysis and Enhancement of Robustness

In Section 5.2.2, we discussed possible ways of embedding secondary data via manip-
ulating pixel values to enforce certain relationships (i.e., Type-II embedding). The
Figure 5.11: Data hiding in binary document image. (a) Original copy, (b) a marked copy with 976-bit embedded in, (c) magnified original image, (d) difference between original and marked (shown in black), (e) magnified marked image, (f) a portion of the image where alteration is done (on the marked image) by changing “1998” to “1999”, (g) among the 976-bit hidden data, 800 bits forms a “PUEE” pattern; the 800-bit data patterns extracted after alteration is visually random and significantly different from the embedded “PUEE”.

5,825,892
Oct. 20, 1998
alter
5,825,892
Oct. 20, 1999
alteration
PUEE
robustness against noise is quite limited, and generally depends on whether and how much quantization or tolerance zone we applied. Let us consider the simple odd-even case with no quantization, i.e., the total number of black pixels is enforced to an even number to embed a “0”, and to an odd number to embed a “1”. When a single pixel gets flipped due to noise, the bit embedded in the block to which the pixel belongs to will be wrongly decoded. When several pixels in an embedding block are subject to be changed, whether or not the bit can be decoded correctly depends on how many pixels being flipped; if the change is independent from pixel to pixel and is with probability $p$ for each of $n$ pixels where $n \geq 1$, the probability of getting a wrongly decoded bit is

$$P_{e_1} = \sum_{k=1, k \text{ odd}}^{n} \binom{n}{k} p^k (1-p)^{n-k} = \frac{1 - (1 - 2p)^n}{2}. \quad (5.3)$$

The error probability $P_{e_1}$ is small for small $p$ and small $n$. In this case, error correction encoding can be applied to correct errors if accurate decoding of hidden data is preferred. When $p$ is close to 0.5, so is $P_{e_1}$, implying the difficulty in embedding and extracting data reliably. Notice that because of shuffling, the assumption of independent change is likely to hold even if the noise involves nearby pixels since adjacent pixels in the original image will be distributed to several blocks. If the total number of changed pixels in the whole image is small (no matter whether they are close to each other in the original image or far away), it is likely that most of those pixels are involved in different embedding blocks hence the extracted bits from those blocks will be wrong; on the other hand, if many pixels have been changed, each embedding block may include several of these pixels and the decoded bit from each block is wrong with approximately 0.5 probability. This implies that the decoded data are rather random, like what we have seen in Fig. 5.11(g). The case of incorporating quantization
or tolerance zone can be analyzed similarly, combining the above strategies and those in Chapter 3.

Besides the noise involving flipping of individual pixels, misalignment is another cause of decoding errors. For this matter, using shuffling has the disadvantage of increasing the sensitivity against geometric distortion such as translation. This is due to the shift-variant property of the shuffling operation, i.e., the shuffling result of a shifted image is very different from that of the non-shifted one. To alleviate the sensitivity with respect of translation, we can hide secondary data in a cropped part of the image, as shown in Fig. 5.12. Without loss of generality, we consider the case of black foreground and white background. The upper-left point of the data hiding region is determined by the uppermost and leftmost black pixel, and the lower-right point is by the lowermost and rightmost black pixel. The data hiding region therefore covers all black pixels. This approach can reduce the shifting sensitivity as long as both embedding and detection system agree on the protocol and no cropping or addition of the outermost black pixels is involved.

Figure 5.12: Achieving robustness against small translation. Here we use the outermost black pixel to determine a data hiding region (indicated by a dash box) covering all black pixels.
In addition to the above approach, adding registration marks helps to survive high-resolution printing and scanning. Recovering the image from printing and scanning with precision as high as one pixel is a challenging task, because this D/A-A/D process may result in small rotation, up-scaling of an unknown factor, and noisy boundary. If one original pixel in the image corresponds to a very small number of pixels in the scanned version (e.g., corresponding to one or less than one pixel), it will be very difficult to combat the distortion introduced by the D/A-A/D process. On the other hand, if significant oversampling is performed so that one original pixel corresponds to a large number of pixels in the scanned version, it would be possible to sample at the center of each “original” pixel, averaging out the noise introduced on the boundary and/or by the rounding errors in de-skewing. The registration marks help to identify the boundary and the size of the original image as well as to correct skewing. We noted that while the size of one original pixel represented in the scanned image may be estimated from a well-designed registration mark (e.g., we may estimate that one original pixel corresponds to $8 \times 8$ pixels in a scanned image), minor errors in such estimation could be accumulated when determining the width and height of the original image up to single pixel precision. For this reason, we impose constraints on the width and height of original images, for example, to be multiples of 50. As shown in Fig. 5.13(a), one possibility is to add cross-shape marks at four corners and at four sides at an interval of 50 pixels horizontally and of 25 pixels vertically, serving as a ruler. In our experiment, we printed out the signature image via the Microsoft Word program (with default image importing resolution 72dpi) using a HP 2100TN laser printer, and scan back with 600dpi precision and 256 gray levels using a Microtek 3600 scanner. The image is binarized using the mean of the maximum and minimum of scanned luminance value. We use the registration marks to determine the
image boundary, to perform de-skewing, and to compute the proper scaling factor. The estimated centers of each original pixel on the scanned version are shown in Fig. 5.13(b). Sampling at those pixels can recover the original digital image perfectly from the scanned one hence allow the embedded data to be extracted correctly. In the Appendix Section 5.7, we shall present more detailed discussion on the recovery of binary image from printing and scanning.

Figure 5.13: Recovering binary image from high quality printing and scanning. (a) Cross-shape marks are added at four corners and at four sides at an interval of 50 pixels horizontally and of 25 pixels vertically, helping to determine the boundary, the scale, and the skewing angle of a scanned image; in addition, the width and height of original images are constrained to be multiples of 50; the image is imported to Microsoft Word at 72dpi, printed out via a laser printer, and scanned in with 600dpi and 256 grey levels; the size of the scanned image is 2028x444. (b) The estimated centers of each original pixel are shown in light color; sampling at those centers can recover the original binary image perfectly from a scanned one.
5.4.2 Security Considerations

We have demonstrated how to embed data in a binary image and illustrated two possible applications in Section 5.3. Drawing an analogy between data hiding and communication, the embedding methods serve as physical communication layer, on top of which other functionalities and features can be built. For instance, security issues may be handled by top layers in authentication applications. Under this scenario, the major objective of an adversary is to forge authentication data so that an altered document can still pass the authentication test. One could use traditional cryptography-based authentication to produce a cryptographical digital signature and to embed it in the binary image. This traditional approach relies on a cryptographically strong hash function to produce a digest of the document to be signed as well as on public-key encryption to enable verification without giving up the encryption keys, hence only authorized person can produce a correctly encrypted signature [11]. By using embedding, we not only save room that is needed for storing and/or displaying the cryptographical data separately, but also obtain additional capability to associate the authentication data with the media source in a seamless way.

Although a cryptographical signature can be adopted as (part of) the embedded data, the embedding approach proposed in this paper has the potential of allowing plain text to be embedded since secret information such as keys/seeds have already been incorporated via shuffling and/or lookup table. However, envisioning potential attacks [139], the following security issues have to be considered. More specifically, for the application of authentication, it is important to study the following two problems, assuming that the attacker has no knowledge about any secret keys: (1) the probability of making content alterations while preserving the $m$-bit embedded authentication data, and (2) the possibility for an adversary to hide specific data in an
CHAPTER 5. DATA HIDING IN BINARY IMAGES

image, assuming he/she has no knowledge about any secret keys.

For the first problem, we have discussed in Section 5.4.1 that an \( n \)-pixel alteration on a marked image would change the decoded data. If \( n \) is small compared to the total number of blocks \( m \), there are approximately \( n \) bits in the decoded data that will be different from the originally embedded one; if \( n \) is large, the probability of getting the decoded data to be exactly the same as the originally embedded one is approximately \( 2^{-m} \), which is very small as long as \( m \) is reasonably large. Therefore, the threat of making content alterations while preserving the \( m \)-bit embedded authentication data is weak.

For the second problem, it depends on whether watermarked versions of the same image with different data embedded are available to an adversary. For convenience, we shall call these as “multiple copies”. When multiple copies are not available, it is extremely hard for an adversary to embed specific data in an image, even if he/she knows the algorithm. This is due to the secrecy in the shuffling table. However, in applications such as “signature in signature”, an adversary may be able to obtain multiple copies, for example, signatures with different signing date or payment amount embedded. This is similar to the scenario of plaintext attack in cryptography [11]. We would like to know whether he/she can derive information regarding which pixels carrying which bit by studying the difference between those copies hence create new images with specific data embedded (e.g., specific date or payment amount). If the embedding imposes the minimal necessary changes to enforce a desirable relationship (for example, in the odd-even case, at most one pixel will be flipped in each embedding block), the pixels that differ among the multiple copies are those used for embedding hidden information. Assuming an adversary collects sufficiently many copies and knows what data is embedded in each copy, he/she will be able to identify which pixels
carrying which bit and to hide his/her desired data by manipulating the corresponding pixels.

To prevent the above-mentioned attack, we have to introduce more uncertainty. One approach is to use a different shuffling table, for example, choose one table from $K$ candidate ones, similar to the approach used for handling bad shuffles in Section 4.2.3. Another approach is that instead of making minimal changes for hiding one bit in each embedding block, we also flip, with probability of 0.5 in each block, an additional pair of flippable pixels. More specifically, to embed a “0”, if the number of black pixels in a shuffled block is already an even number, with probability of 0.5 we flip an additional pair of pixels selected arbitrarily from 3 highly flippable pixels; if the number of black pixels is an odd number, with probability of 0.5:0.5 we flip all 3 pixels or flip one pixels selected arbitrarily from the 3 ones. When more than three highly flippable pixels are available, we may make the above selection from a larger pool. Now if we look at two image copies whose hidden data differ in just one bit, the difference between the two images via minimal-change embedding is just at one pixel, while that via the above-mentioned randomization involves many other pixels randomly. In the latter case, if a total of $N$ bits are embedded, on average there will be $(4N + 1)/3$ pixels being different. The computation is sketched as follows: we first consider the embedding of the different bit between the two images, i.e., in one image, 0 or 2 pixels are flipped while in the other image, 1 or 3 pixels are flipped. The four combinations (0-1, 2-1, 0-3, 2-3) are equally likely, giving the average number of different pixels as a result for embedding this bit as $5/3$. Similarly, the average number of different pixels for embedding the rest $(N − 1)$ identical bits is $4(N − 1)/3$, giving the overall average $(4N + 1)/3$. When $N$ is sufficiently large, it is difficult for an adversary to identify which pixels are associated with which bits. As a tradeoff, the randomization requires
three flippable pixels to be available for many shuffled blocks and introduces more pixel changes at the embedding step. Note that both countermeasures assume that for any given hidden data, only one copy of a marked image is available to an attacker, otherwise he/she may be able to average out the randomization and to compromise our solutions.

5.5 Chapter Summary

This chapter addresses the problem of data hiding for binary images. Technical challenges in such embedding are discussed with solutions proposed. In particular, we propose a new data hiding method for binary images. The method manipulates “flippable” pixels to enforce a specific block-based relationship in order to embed a significant amount of data without causing noticeable artifacts. Shuffling is applied before embedding to equalize the uneven embedding capacity. The hidden data can be extracted without using the original image, and with the help of a few registration marks, they can also be accurately extracted after high quality printing and scanning. The algorithm can be applied to detect unauthorized use of signatures in binary image format and to detect alterations on documents.

In terms of future work, the flippability model can be refined for different types of binary images such as texts, figures/drawings, and dithered images. The approach for recovering binary image from high quality printing and scanning may be improved by using the grayscale information from the scanned image. Comparative study of various embedding mechanisms will also provide insights on the improvement of data hiding in binary images.
Acknowledgement

The connectivity criterion for generating flippability scores was revised from a proposal by Ed Tang of Princeton Summer Institute ’99. The application of “signature in signature” was proposed by Computer Science Prof. Adam Finkelstein.

5.6 Appendix - Details of Determining Flippability Scores

In this appendix section, we describe the details of a 5-step procedure for computing flippability scores. For simplicity, we shall illustrate the evaluation method for non-dithered binary image. The scores will be used to determine which pixel will be flipped with high priority during the embedding process.

Step-1 Compute smoothness and connectivity of 3 × 3 pattern.

The smoothness of the neighborhood around pixel \( (i, j) \) is measured by the total number of horizontal, vertical, diagonal, and anti-diagonal transitions in the 3x3 window, respectively, using a differential operator along the corresponding directions:

\[
\begin{align*}
\text{horizontal} \quad N_h(i, j) &= \sum_{k=-1}^{1} \sum_{l=-1}^{0} I(p_{i+k,j+l} \neq p_{i+k,j+l+1}),
\text{vertical} \quad N_v(i, j) &= \sum_{k=-1}^{1} \sum_{l=-1}^{0} I(p_{i+l,j+k} \neq p_{i+l+1,j+k}),
\text{diagonal} \quad N_d_1(i, j) &= \sum_{k,l \in \{-1,0\}} I(p_{i+k,j+l} \neq p_{i+k+1,j+l+1}),
\text{anti-diagonal} \quad N_d_2(i, j) &= \sum_{k \in \{0,1\}, l \in \{-1,0\}} I(p_{i+k,j+l} \neq p_{i+k-1,j+l+1}).
\end{align*}
\]

where \( I(\cdot) \) is the indicator function taking value from \( \{0, 1\} \), and \( p_{i,j} \) denotes the pixel value of the \( i^{th} \) row and \( j^{th} \) column of the whole image. These computations are also
illustrated in Fig. 5.14. Note that regular patterns such as straight lines, have zero transition along at least one direction, as shown in Fig. 5.15.

Figure 5.14: Illustration of transitions in four directions, namely, horizontal, vertical, diagonal, and anti-diagonal. The number of transitions is used to measure the smoothness of the $3 \times 3$ neighbourhood.

Figure 5.15: Regular patterns such as straight lines have zero transition along at least one direction. Showing here is part of a horizontal line with zero horizontal transition.

The connectivity is measured by the number of the black and white clusters. For this purpose, the connectivity criterion between two pixels needs to be prescribed. A commonly used criterion, illustrated in Fig. 5.16, considers the pixels that touch each others by 90-degree (i.e., $(i, j \pm 1)$ or $(i \pm 1, j)$) or by 45-degree (i.e., $(i + 1, j \pm 1)$
or \((i - 1, j \pm 1)\) and that have the same pixel value as connected. Depending on the specific constraints of visual artifacts, 45-degree touching may not always be considered as connected. Using the criterion, we can build a graph for black (or white) pixels. In the graph, each vertex represents a black (or white) pixel, and there is an edge between two vertices if and only if the two corresponding pixels are connected. An example is shown in Fig. 5.17 with five black pixels forming two clusters and four white pixels forming one cluster. The number of clusters can be automatically identified by traversing the graph using depth-first search strategy. Here we present a stack-based implementation of non-recursive depth-first search algorithm, adapted from [4]. We assume that there are \(M\) pixels in total (counting both white and black), and the final value of “counter” indicates the number of clusters.

1. Initialization: let \(p[k]\) store the value of \(k\)th pixel and \(q\) be the pixel value of interest (i.e., \(q\) is black if to find black clusters, and vice versa); set up an empty stack and an \(M\)-element array \(\text{label}[\cdot]\) for storing the index of the cluster that each pixel belongs to; set \(\text{label}[k] = 0\) for all \(k = 1, \ldots, M\); \(i = 1;\) counter = 0.

2. If \(\text{label}[i] \neq 0\) (i.e., it has already been visited) or \(p[i] \neq q\), go to (7).

3. \(\text{counter} = \text{counter} + 1;\) push node-\(i\) into the stack.

4. If the stack is empty, go to (7).

5. \(k = \text{pop( ) from stack}\); \(\text{label}[k] = \text{counter} \).

6. Find all pixels directly connected with \(k\). For each connected pixel \(j\), if \(\text{label}[j] = 0\) (i.e., it has not yet been visited or pushed into stack), assign \(\text{label}[j] = -1\),

\[\text{In some references, 90-degree touching is known as four-connectivity, and 90-degree or 45-degree touching is known as eight-connectivity [17].}\]
and push node-$j$ into stack $^3$. Go back to (4).

$^7$ $i = i + 1$; if $i > M$, stop, otherwise go to (2).

Figure 5.16: The pixels that touch each others by $90^\circ$ (i.e., $(i, j \pm 1)$ or $(i \pm 1, j)$) or by $45^\circ$ (i.e., $(i + 1, j \pm 1)$ or $(i - 1, j \pm 1)$ ) and that have the same pixel value as connected. The lightly shaded pixels in this figure are considered as touching the center pixel by $90^\circ$, while those with stripes touch the center by $45^\circ$.

Figure 5.17: Graph representation of the connectivity for black and white pixels. Showing here is an example of 3x3 pattern with five black pixels forming two clusters and with four white pixels forming one cluster. Only 90-degree touching is considered as connected in this example.

**Step-2 Compute flippability score.**

The smoothness and connectivity measures are passed into a decision module to come up with a flippability score. Main considerations when designing this module

$^3$Note that by the definition of connected, $p[j] = q$. 
are: (1) whether the original pattern is a very smooth pattern, (2) whether flipping will increase non-smoothness by a large amount, (3) whether flipping will cause the change of connectivity. These changes or the artifacts on these patterns are generally more significant. For each pixel, the followings are the major rules used in our decision module:

(1) the lowest score (i.e., not flippable) is assigned to uniform white or black regions as well as to the isolated single white or black pixels. These trivial cases are handled first.

(2) if the number of transitions along horizontal or vertical direction is zero, i.e., the pattern is very smooth and regular, assign zero flippability as a final score for the current pixel. Otherwise, assign to the pixel a base flippability score $S_B$ and proceed to the next rule.

(3) if the number of transitions along diagonal or anti-diagonal direction is zero, reduce the flippability. Otherwise, if the minimum transition point along any one of the four directions is below a given threshold $T_1$, which means the pattern is rather smooth, reduce flippability by a small amount. Note that we treat smooth horizontal/vertical patterns and diagonal/anti-diagonal patterns differently because the artifacts along the horizontal/vertical patterns are likely to attract more attention from viewers.

(4) if flipping the center pixel does not change the transition points, increase the flippability. Otherwise, if flipping results in the increase of transition points (i.e., reduces smoothness and makes the pattern noisy), decrease flippability.

(5) if flipping changes the number of black clusters or white clusters, reduce the flippability.
After going through these rules, a lookup table of all $3 \times 3$ patterns can be obtained and ordered in terms of how unnoticeable the change of the center pixel will cause. For small neighborhood such as $3 \times 3$, this table has a small number of entries ($2^{3 \times 3} = 512$) hence can be off-line computed. The flippability score of every pattern in an image can then be determined by looking up the stored table. When larger neighborhood is involved, for example, $5 \times 5$ neighborhood, the table size increases exponentially ($2^{5 \times 5} = 2^{25} \approx 32\text{mega}$), possibly exceeding the available memory size for particular applications. This problem can be solved by online computing the flippability for each pattern. More efficiently, we may adopt hierarchical approach, namely, obtaining preliminary flippability measure based on a small neighborhood (e.g., $3 \times 3$) by table lookup, then if necessary, refining the measure by on-line computing based on a larger neighborhood.

**Step-3 Handle special cases.**

We handle some special cases that involve larger neighborhood. For example, we detect particular patterns such as sharp corners to avoid introducing annoying artifacts on them.

**Step-4 Impose minimum distance constraint between two flippable pixels.**

Up to now, the flippability evaluation is done independently for the pattern revealed in a moving window centered at each pixel, assuming that any pixels other than the center one will not be flipped. Pixels that are close to each others may be considered flippable by this independent study, but simultaneously flipping them could cause artifacts. We handle this problem by imposing constraints on the minimum distance of two pixels that can be flipped and pruning the pixels with relatively low flippability in its neighborhood.
Step-5 Assign a predetermined score to the remaining boundary points (optional).

Edge pixels that have not yet been assigned non-zero flippability will be given a small flippability value. These pixels serve as a bottom line for hiding a particular bit when there is no pixel with higher flippability available to carry the hidden data. Adding this step helps to achieve a high embedding rate while keeping visual quality reasonably good for the extreme cases.

The above procedures can be further refined by studying a larger neighborhood and by using more extensive analysis, especially for Step-2. Shown in Fig. 5.18 is one possible lookup table for $3 \times 3$ patterns, excluding the symmetric cases of rotation, mirroring, and complement. Here we set the threshold $T_1 = 3$, the base flippability score $S_B = 0.5$, and the flippability adjustments in Step-2 are multiples of 0.125.

For dithered image, some criterions and parameters need to be revised, for example, a pixel is given high flippability if its flipping does not cause larger relative change in local intensity, and the connectivity is given less consideration. The techniques in lossy bi-level image compression like those in JBIG2 activities [20] may provide further insights to data hiding, and the methods used in data hiding may also contribute to compression.

5.7 Appendix - Details on Recovering Binary Images After Printing and Scanning

In Section 5.4.1, we described adding cross-shape marks at four corners and at four sides to serve as a ruler for registration purpose. Identifying the cross points of these
Figure 5.18: One possible flippability lookup table for $3 \times 3$ pattern, excluding symmetric cases of rotation, mirroring, and complement. Larger value indicates that the change of center pixel is less noticeable hence the change is more likely to be made for hiding information.

marks in a scanned image is the first step in recovering binary image from high quality printing and scanning. Here we propose a projection based approach under the assumption that the approximate region of the mark to be recovered has already been specified. For white background, the range should include the entire mark and preferably no other black pixels. A white outer layer of fixed width is added to the source image to facilitate the identification of the mark regions, as shown in Fig. 5.19. The approximate region containing the mark can be either manually specified via an interactive interface or automatically determined via pattern matching. For simplicity, the manual approach is used in our experiment, and reasonable effort is made
during scanning so that the skewing of each mark is negligible.

![Illustration of registration marks](image)

Figure 5.19: Illustration of registration marks. A white outer layer of fixed width is added to facilitate the identification of the approximate region of each mark during print-and-scan.

To determine the cross point of a mark, we perform horizontal and vertical projections and get two profiles each of which has a unique “plateau” corresponding to the horizontal and vertical stroke, respectively. As illustrated in Fig. 5.20, the centers of the two plateaus determine the y- and x- coordinates of the cross point.

Using the identified cross points of registration marks, we can determine the skewing angle $\alpha$ of the entire scanned image, as illustrated in Fig. 5.21. The scaling factors can be estimated as follows: assuming the original image size has been determined as $N_w \times N_h$ and the scanned image size is $W \times H$, all measured in pixels. We further assume the coordinate of the upper-left pixel in both the scanned image and

---

$^4$Recall that we have impose constraints that the width and height of original binary image has to be multiples of 50. The actual multiplication factor can be determined by registration marks that serve as a ruler. Alternatively, the multiplication factor can be determined by estimating from the width of registration marks how many pixels in the scanned image correspond to one pixel in the original. Any additions such as the white outer layer in Fig. 5.19 need also be counted.
Figure 5.20: Determining the cross point of a registration mark by performing horizontal and vertical projection. The centers of the two projection plateaus are used as y- and x-coordinates of the cross point.
the original image is (0, 0). Considering a pixel \((x', y')\) in the original image, we would like to find the center of this pixel in the scanned version. We first perform a scaling operation:

\[
\begin{bmatrix}
  x_1 \\
  y_1
\end{bmatrix} = \begin{bmatrix}
  \frac{W-1}{N_w-1} & 0 \\
  0 & \frac{H-1}{N_h-1}
\end{bmatrix} \begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  \frac{(W-1)x'}{N_w-1} \\
  \frac{(H-1)y'}{N_h-1}
\end{bmatrix},
\]

(5.8)

where \((W - 1), (H - 1), (N_w - 1)\) and \((N_h - 1)\) are used because the coordinate of the first pixel starts from \((0, 0)\). We then perform rotation of \(-\alpha\) degree and get the coordinate \((x, y)\) of the estimated pixel center:

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = \begin{bmatrix}
  \cos \alpha & -\sin \alpha \\
  \sin \alpha & \cos \alpha
\end{bmatrix} \begin{bmatrix}
  x_1 \\
  y_1
\end{bmatrix}.
\]

(5.9)

If the estimation is well centered in the pixel and the scanning resolution is sufficiently high so that one original pixel corresponds to many scanned pixels (like those shown in Fig. 5.13), sampling at the estimated centers will recover the original image. Improvement may be done by considering the surrounding pixels as well as the grayscale information obtained from scanning, especially when a printed image has noisy boundaries and/or slightly blurred.
Figure 5.21: Using coordinate conversion to perform scaling and de-skewing. Coordinate x-y is for the scanned image, and Coordinate x’-y’ is for the original image. Skewing angle between the two coordinates is represented by $\alpha$; the lightly dotted squares represent original pixels, and the round dots are the corresponding centers.