Chapter 3

Watermarking for Digital Images

In Sections 2.1 and 2.2 we pointed out the need to protect multimedia data against illegal redistribution and malicious alterations. In this chapter, we present our contributions to watermarking for digital images for copyright protection and authentication.

3.1 Image visual components for imperceptible mark embedding

Next, we propose an algorithm that computes pixel *visual components*, that is, the perceptual value of pixels. This value is an estimate of the maximum subperceptual increment/decrement that each pixel can accomodate without causing visual degradation.

The idea underlying Algorithm 1 is that dark pixels and those pixels in non-homogeneous regions are the ones that can best accomodate embedded information while minimizing the perceptual impact. This algorithm is used to provide imperceptibility to the new watermarking schemes proposed in Sections 3.2 and 3.3 which we have published in [SDH00, SD01] respectively.

Without loss of generality, we will assume a monochrome image in what follows; for RGB color images, watermarking is independently done for each color plane. Let the original image be $X = \{x_i : 1 \leq i \leq n\}$, where x_i is the color level of the i-th pixel and n is the number of pixels in the image. Let x_i take integer values between 0 and $MAXCOLOR$, so that the lower x_i, the darker is the color level. Let parameter $dt \in [0, MAXCOLOR]$ be a threshold such that all color levels x_i below dt visually appear as dark.
Let lb_1 and ub_1 be integer values $lb_1 < ub_1$ that are used as parameters to bound the variation of pixel color values. For a given $MAXCOLOR$, suitable values for dt, lb_1 and ub_1 are empirically chosen.

Algorithm 1 (Visual components(dt, lb_1, ub_1))

```
1. For $i = 1$ to $n$ do:
   (a) Compute $m_i := \max_j |x_i - x_j|/2$, for all pixels $j$ which are neighbors of pixel $i$ in the image (there are up to eight neighbors); $m_i$ can be regarded as a kind of discrete derivative at pixel $i$. To bound the value of $m_i$ between $lb_1$ and $ub_1$, perform the following corrections:
       i. If $m_i > ub_1$ then $m_i := ub_1$.
       ii. If $m_i < lb_1$ then $m_i := lb_1$.
   (b) Compute the darkness of the $i$-th pixel as $d_i := (dt - x_i) * ub_1/dt$ if $x_i < dt$ and $d_i := 0$ otherwise. A pixel is considered as dark if its color level is below $dt$. The value of $d_i$ lies between 0 and $ub_1$.
```

   (c) *Compute the preliminary visual component of the i-th pixel as* $v_i := \max(m_i, d_i)$.

2. *For* $i = 1$ *to* n *compute the final visual component of the i-th pixel as* $V_i := \max_j v_j$, *for all pixels* j *which are neighbors of* i *in the image plus the pixel* i *itself.*

The higher V_i for a pixel, the less perceptible are changes in that pixel.

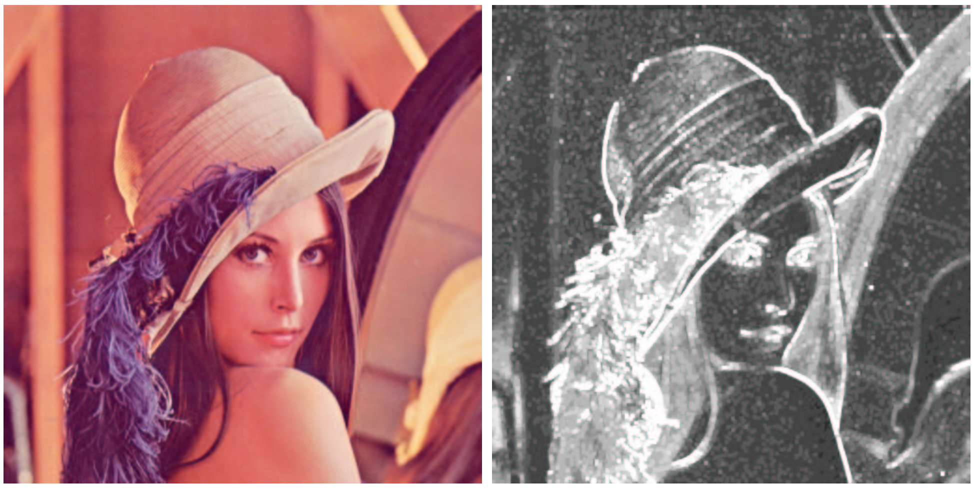

Figure 3.1: Visual components of the image Lena.

Figure 3.1 shows the result of applying the visual component algorithm to the image Lena (parameters are $dt = 70$, $lb_1 = 2$ and $ub_1 = 11$). The lighter a pixel, the larger is its visual component value. Note that large visual component values are located in non-homogeneous or dark areas where color level alterations will not be easily perceived.

3.2 Scale-proof semi-public image watermarking

This section describes a new robust semi-public[1] watermarking scheme for images which we have published in [SDH00]. Bits are embedded into the image using the same underlying idea of the scheme described in [Her00], but are distributed so as to replace robustness against cropping attacks by robustness against scaling attacks.
Imperceptibility is achieved by using the visual components algorithm described in previous section.

The scheme consists of the mark embedding and mark recovery algorithms.

3.2.1 Mark embedding

A set of parameters must be specified. These are:

- dt, lb_1, ub_1 are required by the visual components algorithm (see Section 3.1) run before mark embedding to compute values V_j;
- k is a secret key only known to the merchant and used to generate a pseudo-random bit sequence $\{s_i\}_{i\geq 1}$. This sequence is used to encrypt the mark bits before embedding;
- p and r are two parameters used to locate the pixels into which mark bits will be embedded.

[1]Knowledge of the original image is needed for mark recovery.

Algorithm 2 (Mark embedding(p,r))

1. *Divide the image into the maximum possible number of square tiles of p pixels side, so that there is a r pixels wide band between neighboring tiles (the band separates tiles). Let q be the number of resulting tiles. Each tile will be used to embed one bit, so q is the capacity of this watermarking scheme.*
2. *Call ε the mark to be embedded. Encode ε using an error-correcting code (ECC) to obtain the encoded mark E. If $|E|$ is the bit-length of E, we must have $|E| \leq q$. Replicate the mark E to obtain a sequence E' with q bits.*
3. *For $i = 1$ to q compute $s'_i = e'_i \oplus s_i$, where e'_i is the i-th bit of E'.*
4. *To embed the i-th encrypted mark bit s'_i into the i-th tile do:*
    (a) *If $s'_i = 0$ then $x'_j := x_j - V_j$ for all pixels x_j in the i-th tile.*
    (b) *If $s'_i = 1$ then $x'_j := x_j + V_j$ for all pixels x_j in the i-th tile.*
5. *For every pixel j not lying into any tile, $x'_j := x_j$.*

$X' = \{x'_i : 1 \leq i \leq w \times h\}$ is the marked image.

3.2.2 Mark recovery

The assumptions for mark recovery are knowledge of the original image X, the secret key k (in order to regenerate the pseudo-random sequence $\{s_i\}_{i\geq 1}$) and parameters p and r. The only required knowledge on the original mark ε is its length, so that the proposed scheme is also usable for fingerprinting. Let $\hat{X}$ be the redistributed image, and let $\hat{w}$ and $\hat{h}$ be its width and height.

Algorithm 3 (Mark recovery(p,r))

```
1. Let $ones[\cdot]$ and $zeroes[\cdot]$ be two vectors with $|E|$ integer positions
   initially all set to 0.
2. From the length $p$ of the tile side, the width $r$ of the intertile band and
   $X$, compute the number of tiles $q$.
3. For $t = 1$ to $q$ do:
   (a) Let $u := 1 + ((t-1) \bmod |E|)$
   (b) For each pixel in the $t$-th tile of the original image $X$ do:
       i. Let $i$ and $j$ be the row and column of the considered original
          pixel, which will be denoted by $x_{ij}$.
       ii. Locate the pixel $\hat{x}_{ab}$ in the marked image $\hat{X}$ corresponding to
           $x_{ij}$. To do this, let $a := i \times \hat{h}/h$ and $b := j \times \hat{w}/w$.
       iii. Compute $\hat{\delta}_{ij} := \hat{x}_{ab} - x_{ij}$.
       iv. If $\hat{\delta}_{ij} > 0$ then
           A. If $s_t = 0$ then $ones[u] := ones[u] + 1$.
           B. If $s_t = 1$ then $zeroes[u] := zeroes[u] + 1$.
       v. If $\hat{\delta}_{ij} < 0$ then
           A. If $s_t = 0$ then $zeroes[u] := zeroes[u] + 1$.
           B. If $s_t = 1$ then $ones[u] := ones[u] + 1$.
4. For $u = 1$ to $|E|$ do:
   (a) If $ones_u > zeroes_u$ then $\hat{e}_u := 1$, where $\hat{e}_u$ is the recovered version
       of the $u$-th embedded bit.
```

(b) *If* $ones_u < zeroes_u$ *then* $\hat{e}_u := 0$.

(c) *If* $ones_u = zeroes_u$ *then* $\hat{e}_u := \#$, *where* $\#$ *denotes erasure.*

5. *Decode* $\hat{E}$ *with the same ECC used for embedding to obtain* $\hat{\varepsilon}$.

3.2.3 Parameter choice

The scheme was implemented with parameter values $MAXCOLOR = 255$, $dt = 70$, $lb_1 = 1$, $ub_1 = 4$. These values were empirically chosen to achieve a satisfactory tradeoff between robustness and imperceptibility.

Regarding parameters p and r, we recommend to use $p = 5$ and $r = 3$ as a tradeoff between capacity —which would favor tiles as small as possible and intertile bands as narrow as possible—, robustness —the larger a tile, the more redundancy in bit embedding and the more likely is correct bit recovery— and imperceptibility —the wider a band, the less chances for artifacts. The band between tiles is never modified and it helps avoiding perceptual artifacts that could appear as a result of using two adjacent tiles to embed a 0 and a 1.

3.2.4 Robustness assessment

The scheme, with parameters described above, was implemented using a dual binary Hamming code $DH(31, 5)$ as ECC (which provides a correction capacity of 7 errors per codeword). The base test of the StirMark 3.1 benchmark [PAK98] was used to evaluate its robustness. The following images from [Bas] were tried: Lena, Bear, Baboon and Peppers. A 70-bit long mark ε was used (as stated in [PA99]), which resulted in an encoded E with $|E| = 434$. Figure 3.2 shows the original and the marked Lena after embedding a 70 bit length mark; the peak signal-to-noise ratio between both images is 41.13 dB.

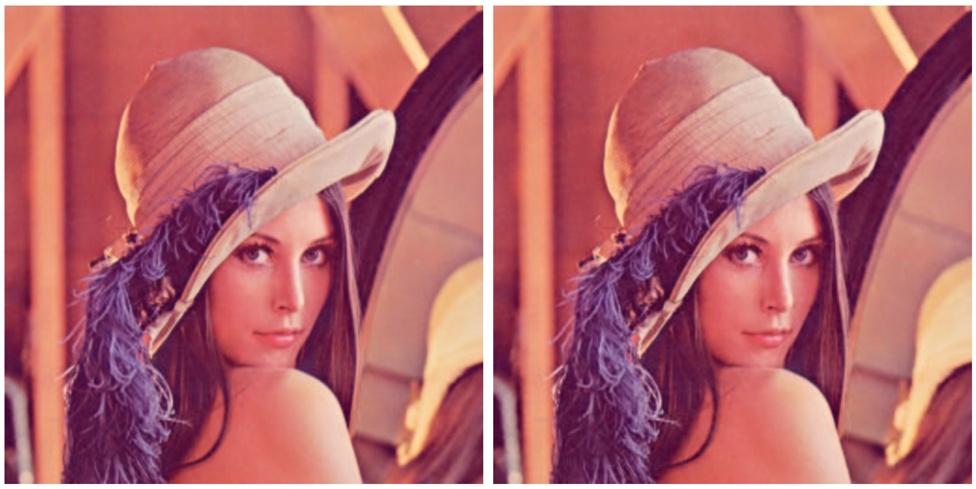

Figure 3.2: Semi-public scheme: Left, original Lena. Right, Lena after embedding a 70 bit long mark.

The following StirMark 3.1 manipulations were survived by the embedded mark:

1. Color quantization.
2. Most filtering manipulations. More specifically:
   (a) Gaussian filter (blur).
   (b) Median filter (2×2 and 3×3).
   (c) Frequency mode Laplacian removal [BP98].
   (d) Simple sharpening.
3. JPEG compression for qualities 90% down to 20% (for some images down to 10%).

4. Rotations with and without scaling of -0.25 up to 0.25 degrees.

5. Shearing up to 1% in the X and Y directions.

6. Cropping up to 1%.

7. Row and column removal.

8. All StirMark scaling attacks (scale factors from 0.5 to 2).

Scaling is resisted because a mark bit is embedded in each pixel of a tile; even if the tile becomes smaller or larger, the correct bit can still be recovered. Extreme compression and scaling attacks for which the mark still survives are presented in Figure 3.3.

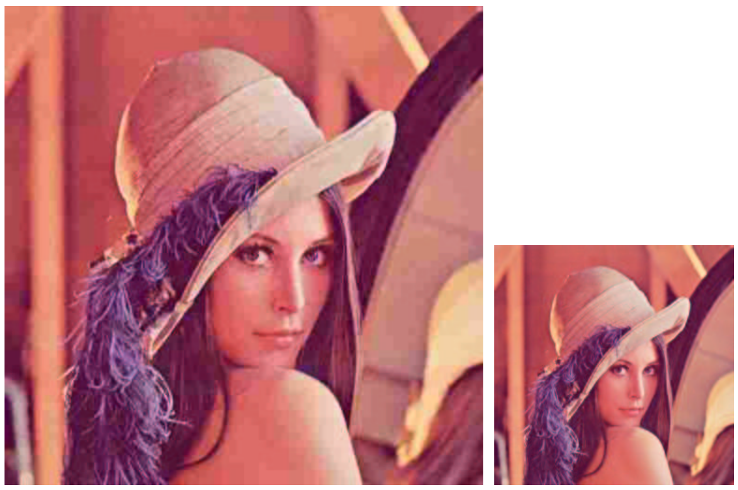

Figure 3.3: Semi-public scheme: Left, marked Lena after JPEG 15% compression. Right, marked Lena after 50% scaling.

3.3 Robust oblivious image watermarking

This section describes a new robust oblivious watermarking scheme for images which we have published in [SD01]. It overcomes the two main shortcomings of current oblivious schemes (see Section 2.1.1); it survives scaling and moderate geometric distortion attacks and does not need previous knowledge of the embedded mark for mark recovery.
Imperceptibility is achieved by using the visual components algorithm described in Section 3.1.

The scheme is composed of the mark embedding and mark recovery algorithms.

3.3.1 Mark embedding

A set of parameters must be specified. These are:

- dt, lb_1, ub_1 are required by the visual components algorithm (see Section 3.1) run before mark embedding to compute values V_j;
- lb_2, ub_2 are used when determining how color level encodes mark bits;
- k is a secret key only known to the merchant M and used to pseudo-randomly locate the pixels into which mark bits will be embedded and the way color levels determine the value of the embedded bits.

Algorithm 4 (Mark embedding(k, lb_2, ub_2))

1. *Let ε be the binary sequence to be embedded. Encode ε using an error-correcting code (ECC) to obtain the encoded mark E.*

2. *Using the key k as a seed, pseudo-randomly place $|E|$ non-overlapped square tiles R_i over the image, where $|E|$ is the bitlength of E. Tile size is also determined by k.*

3. *Using k as a seed, pseudo-randomly assign a value a_i between lb_2 and ub_2 to tile R_i, for $i = 1$ to $|E|$.*

4. *To embed the i-th bit e_i of the mark E in R_i:*

   (a) *Divide the color level interval $[0, MAXCOLOR]$ into subintervals of size a_i.*

   (b) *Label consecutive subintervals alternately as "0" or "1".*

   (c) *For each pixel x_j in R_i:*

      i. *If x_j lies in a subinterval labeled e_i, bring it as close as possible to the interval center by increasing or decreasing x_j no more than V_j.*

      ii. *If x_j lies in a subinterval labeled $\bar{e}_i$, bring it as close as possible to the nearest neighbor interval center (neighbor intervals are labeled e_i) by increasing or decreasing x_j no more than V_j.*

3.3.2 Mark recovery

Upon detecting a redistributed image $\hat{X}$, $\hat{\varepsilon}$ can be recovered as follows, provided that the length $|E|$ of the embedded mark and the secret key k used for embedding are known (the merchant should know these parameters).

Algorithm 5 (Mark recovery(k, lb_2, ub_2))

1. *Using the key k as a seed, pseudo-randomly place $|E|$ non-overlapped square tiles R_i over the image (again, tile size is also determined by k). Also using k as a seed, pseudo-randomly assign a value a_i between lb_2 and ub_2 to tile R_i, for $i = 1$ to $|E|$. (Tiling done in this step is analogous to tiling done during mark embedding).*
2. *To recover the i-th bit $\hat{e}_i$ of $\hat{E}$ from R_i:*
   (a) *Divide the color level interval $[0, MAXCOLOR]$ into subintervals of size a_i.*
   (b) *Label consecutive subintervals alternately as "0" or "1".*
   (c) *Let $ones := 0$ and $zeroes := 0$.*
   (d) *For each pixel x_j in R_i:*
      i. *If x_j lies in a subinterval labeled "1", then $ones := ones + 1$*
      ii. *If x_j lies in a subinterval labeled "0", then $zeroes := zeroes + 1$*
   (e) *If $ones > zeroes$ then $\hat{e}_i := 1$; if $ones < zeroes$ then $\hat{e}_i := 0$; otherwise $\hat{e}_i$ is an erasure.*
3. *Decode $\hat{E}$ with the same ECC used for embedding to obtain $\hat{\varepsilon}$.*

3.3.3 Parameter choice

In Algorithm 1 (visual components), suitable values for parameters dt, lb_1 and ub_1 should be empirically chosen for a given $MAXCOLOR$. Suitability depends on visual perception and robustness considerations. We have suggested a good choice for $MAXCOLOR = 255$, namely $dt = 70$, $lb_1 = 2$

and $ub_1 = 11$; for other values of $MAXCOLOR$, a rule of thumb of is to scale that choice by $MAXCOLOR/255$. We discuss below other parameters related to Algorithm 4 (embedding) and Algorithm 5 (recovery).

On the size of tiles

At Step 2 of the mark embedding algorithm, $|E|$ square tiles are randomly placed over the image. From the point of view of robustness, the tile size must be large enough so that each bit is embedded in a sufficient number of pixels. However, the requirement that all $|E|$ tiles should not overlap limits the maximum tile size, which decreases as $|E|$ increases.

An additional consideration is imperceptibility. Better imperceptibility is gained if neighboring tiles are separated by a band of unmodified pixels. This further limits the tile size.

On the width of color level subintervals

The size a_i in which we divide the color level interval is a tradeoff between robustness and imperceptibility:

- Making such intervals narrow means that, during the mark embedding algorithm, the variation applied to pixels to mark them is low, which leads to better imperceptibility. The drawback of narrow intervals is a loss of robustness, because even small noise can easily shift a color level to a neighboring subinterval, which can cause an incorrect bit to be recovered.

- Larger values a_i yield higher robustness, but moving the color level of a pixel to a neighboring subinterval is more perceptible and sometimes it cannot be achieved as the maximum increment/decrement of a pixel

x_j is limited by its visual component value V_j.

Thus, given $MAXCOLOR$, the interval $[lb_2, ub_2]$ where a_i randomly takes values has its lower bound lb_2 limited by robustness and its upper bound ub_2 limited by imperceptibility.

3.3.4 Robustness assessment

The scheme was implemented and tested using parameter values suggested in Section 3.3.3. The error-correcting code used was a dual binary Hamming code $DH(31, 5)$. The following images from [Bas] were tried: Lena, Bear, Baboon and Peppers. A 30-bit long mark ε was used, which needed six codewords of the dual Hamming code and resulted in an encoded E with $|E| = 31 \times 6 = 186$ bits. For Lena, a version of size 512×512 pixels was used and the length of the tile side was randomly chosen between 11 and 31; for the other images, a similar proportion between image size and tile size was maintained. For all images, tiles were placed so that neighboring tiles were separated by a band of unmodified pixels at least one pixel wide.

Figure 3.4 shows the original and the marked Lena after embedding a 30 bit long mark; the peak signal-to-noise ratio (PSNR) between both images is as high as 41.16 dB.

Some general considerations follow regarding robustness in front of the various kinds of attacks:

- After an attack, a bit is correctly recovered if a majority of correct mark bits are still inside the corresponding tile.

- Scaling attacks are survived by placing tiles in positions relative to the image size. In this way, even if the image size varies, each tile still contains original mark bits.

Figure 3.4: Oblivious scheme: Left, original Lena. Right, Lena after embedding a 30 bit long mark.

- Unless tiles are very small, other attacks like row and column removal, shearing, cropping and rotation will only succeed if most pixels in the tile suffer a variation so large that it leads to visual degradation.

The base test of the StirMark 3.1 benchmark [Sti] was used to evaluate robustness on the marked versions of the four test images. The following manipulations were survived:

1. Color quantization
2. Most filtering manipulations. More specifically:
   (a) Gaussian filter (blur)
   (b) Median filter (2×2, 3×3 and 4×4).
   (c) Linear filter
3. JPEG compression for qualities 90 down to 30.

4. All StirMark scaling attacks (scale factors from 0.5 to 2).
5. All StirMark aspect ratio modification attacks.
6. All StirMark row and column removal attacks.
7. All StirMark shearing attacks.
8. Small rotations with and without scaling from -2 to 2 degrees.
9. Small cropping up to 2%.
10. StirMark random bend.

3.3.5 Multiple marking

A useful feature of the presented algorithm is that multiple marking is supported. Up to three consecutive markings on the same image are possible without substantial perceptual degradation nor loss of robustness. For example, the content creator M_1 can mark an image and sell the marked image to a distributing company M_2 which re-marks the image with its own mark, re-sells it to a retailer M_3, who re-marks the image again before selling it to the end consumer. Each of M_1, M_2, M_3 can recover their embedded watermark by using the same key they used at embedding time.

To illustrate the effects on imperceptibility, Figure 3.5 shows Lena after three consecutive markings.

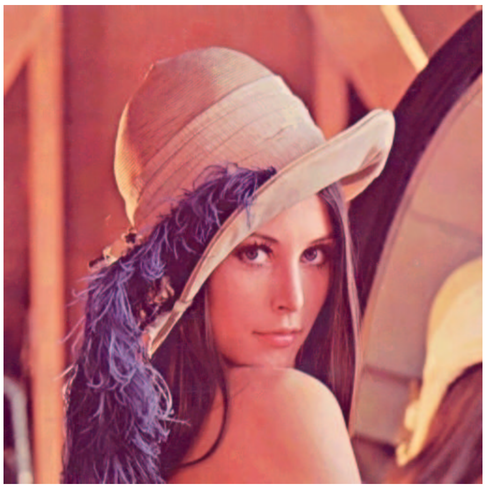

Figure 3.5: Oblivious scheme: Lena after three consecutive markings.

3.4 Enhancing watermark robustness through mixture of watermarked digital objects

Coming up with a watermarking method surviving all conceivable attacks may indeed be a difficult task. We explore in this section ways to obtain increased robustness by mixing the outputs of several watermarking methods. We have published the material in this section in [DS02a].

We will first discuss prior mixture, whereby a digital object is watermarked with different methods and a mixture of the watermarked objects is released. Posterior mixture will then be presented, which consists of mixing several attacked versions of the same watermarked digital object. It will be shown that prior mixture may result in a combination of the robustness properties of the watermarking methods being used. It will also be shown that posterior mixture may allow recovery of the embedded watermark, even if this watermark can no longer be recovered from each individual attacked version of the watermarked object. Note that prior or posterior mixtures are non-exclusive.

3.4.1 Prior mixture

Prior mixture is a general technique that allows a watermarked object to be obtained that combines the robustness properties of several watermarking schemes. No knowledge on the specific embedding and recovery algorithms is needed as they are used as a black box.

Mark embedding and prior mixture

Let $E_1, \cdots, E_n$ be n different watermark embedding algorithms which can be used to embed a watermark M into the original digital object X. It is assumed in what follows that M contains some kind of redundancy (checksum, cyclic redundancy check, etc.), that allows its correctness or integrity to be checked. We then proceed as follows:

Algorithm 6 (Prior mixed embedding)

1. *The watermark M is embedded into X using algorithms $E_1, \cdots, E_n$ to obtain $X'_1, \ldots, X'_n$, where X'_i is the output of E_i.*

2. *A weight α_i is selected for each object X'_i, such that $0 \leq \alpha_i \leq 1$ and $\sum \alpha_i = 1$.*

3. *The watermarked mixed object is computed as*

$$X'_{premix} = f(\alpha_1, \cdots, \alpha_n, X'_1, \cdots, X'_n) \tag{3.1}$$

where f is a mixture function (see below).

Figure 3.6 illustrates Algorithm 6.

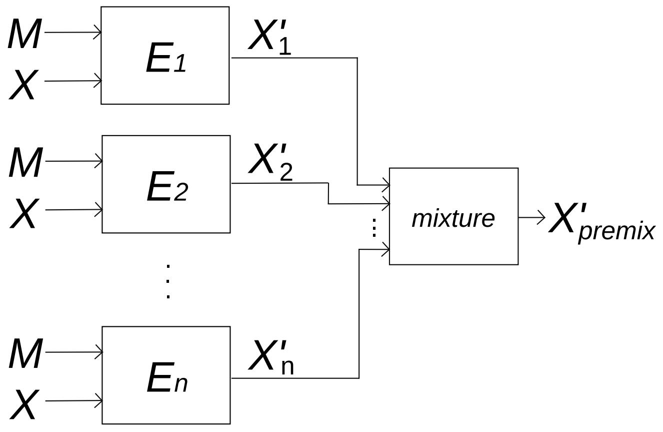

Figure 3.6: Prior mixture mark embedding procedure.

Any mixture function can be used in Algorithm 6. However, sensible choices are an additive mixture

$$f(\alpha_1, \cdots, \alpha_n, X'_1, \cdots, X'_n) = \alpha_1 X'_1 + \cdots + \alpha_n X'_n$$

or a multiplicative mixture

$$f(\alpha_1, \cdots, \alpha_n, X'_1, \cdots, X'_n) = X_1'^{\alpha_1} X_2'^{\alpha_2} \cdots X_n'^{\alpha_n}$$

The above mixtures are componentwise between the semantically corresponding components of objects: for example, if the object is an image, components are pixels and the mixture amounts to averaging the color levels of corresponding pixels.

Mark recovery from a mixed object

Denote by $R_1, \cdots, R_n$ the watermark recovery algorithms corresponding to embedding algorithms $E_1, \cdots, E_n$ respectively. Let $\hat{X}$ be the object we want to recover the watermark from; if it has been attacked, $\hat{X}$ will not exactly match any watermarked object X'. The recovery procedure is as follows:

Algorithm 7 (Recovery from a mixed object)

1. *Run algorithms $R_1, \cdots, R_n$ on $\hat{X}$ and record the output of those algorithms, if any. Depending on the attacks suffered by $\hat{X}$ some algorithms may give no output.*
2. *Look for a correct watermark among the outputs of the recovery algorithms (the redundancy included in marks is checked for correctness). If all correct watermarks found have the same value, then recovery is successful. If there is no correct watermark or if there are several correct watermarks with different values, recovery fails.*

Figure 3.7 illustrates Algorithm 7.

Note that mixing watermarked objects entails some amount of noise for each invidual watermarking method (E_i, R_i). In other words, when running recovery algorithm R_i, the effect of embedding algorithms E_j for $j \neq i$ is perceived as noise. Therefore, for prior mixture to be practical noise-robust watermarking methods must be used.

Applying prior mixture

Next, prior mixture is demonstrated for combining the crop-proof [Her00] and the scale-proof [SDH00] schemes for image watermarking. The resulting mixture stands both cropping and scaling attacks.

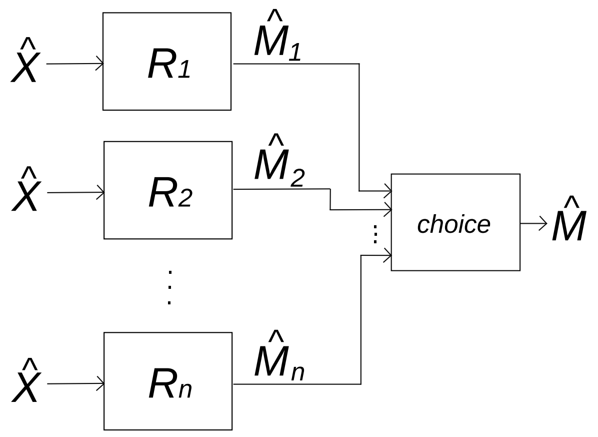

Figure 3.7: Prior mixture mark recovery procedure.

The benchmark image Lena [Bas] was watermarked using the two aforementioned schemes, so that two watermarked versions were obtained. In both cases, the embedded watermark was the same 45-bit binary sequence. Prior image mixture was applied to mix the two watermarked versions of Lena. Additive and multiplicative mixtures with weights $\alpha_1 = \alpha_2 = 0.5$ were tried; in what follows, we report results only for additive mixture, which turned out to outperform multiplicative mixture for this particular example. Additive mixture with the above weights is actually the arithmetic average of color levels of pairs of pixels in the same position within images to be mixed.

The error correcting code (ECC) used in this experiment was a $(31, 5)$ dual Hamming binary code (with correcting capacity 7 errors). When attempting mark recovery from an attacked watermarked image, the average number of corrected errors per codeword at the decoding stage gives an indication of the vulnerability of the scheme against the attack. If the number of errors that must be corrected to reconstruct the watermark is low, then the scheme easily survives the attack; the higher the number of corrected errors after an

attack	crop-proof	mix crop-proof
JPEG 30	2.1	not survived
gaussian	0	2.3
sharpening	0	0
FMLR	1.9	not survived
median 3x3	0	1
cropping	0	0

Table 3.1: Average no. of corrected errors at mark recovery for Lena (crop-proof method).

attack	scale-proof	mix scale-proof
JPEG 30	0	2.8
gaussian	0	1
sharpening	0	3.2
FMLR	5.5	not survived
median 3x3	1.7	not survived
scaling	0	1.5

Table 3.2: Average no. of corrected errors at mark recovery for Lena (scale-proof method).

attack, the more vulnerable is the scheme against the attack.

The following tables show the average number of errors corrected when recovering the watermark from the image Lena. Table 3.1 shows the average number errors corrected by the crop-proof recovery algorithm: the second column accounts for recovery from the crop-proof watermarked Lena (before mixing), while the third column refers to recovery from the mixed crop-proof and scale-proof Lena. Table 3.2 corresponds to errors corrected by the scale-proof recovery algorithm: its second column displays the average number of errors corrected when recovering a mark from the scale-proof watermarked Lena (before mixing); the third column refers to corrected errors in the recovery from the mixed crop-proof and scale-proof Lena.

It can be seen from Tables 3.1 and 3.2 that the result of mixing both schemes is a semi-public image watermarking scheme robust against color quantization, filtering, JPEG compression, cropping and scaling. Thus, we have succeeded in combining resistance against cropping attacks with resistance against scaling attacks. Of course, the amount of noise tolerated by the mixture of both schemes is lower than the amount that would be tolerated by each scheme individually and some filters like FMLR are no longer survived.

The experiment above was repeated with other benchmark images in [Bas], and the results were similar to those obtained with Lena.

Imperceptibility is a very important feature of a watermarking scheme. It refers to the extent to which the image quality is preserved after the mark has been embedded. The Peak Signal-to-Noise Ratio (PSNR) between the original and the watermarked images is one common way to measure imperceptibility.

Table 3.3 shows how, after mixture, image quality does not decrease but stays similar or even higher than quality of watermarked images input to mixture. Table rows correspond to images Lena and Baboon [Bas]. Table columns correspond to the three watermarking possibilities: crop-proof only, scale-proof only or additive mixture of both methods. For each image, the PSNR of the three watermarked versions vs the original image is given. It is noteworthy that the PSNR of the mixed image can even be higher than the PSNR of images watermarked with a single method.

	crop-proof	scale-proof	mixed
Lena	38	41.12	40.59
Baboon	36.7	36.52	36.76

Table 3.3: PSNR of watermarked vs original images

3.4.2 Posterior mixture

Posterior mixture is a technique usable if the following assumptions hold:

A1. Several attacked versions $\hat{X}_1, \cdots, \hat{X}_m$ originating from the same watermarked digital object X' are available, where the watermarking method used and the embedded watermark are the same for all attacked versions. The difference between versions is only caused by the attacks they have undergone.

A2. None of $\hat{X}_1, \cdots, \hat{X}_m$ separately allows recovery of the common embedded watermark.

A3. It must be possible to find a one-to-one mapping between semantically corresponding components of $\hat{X}_i$ and $\hat{X}_{i+1}$, for $i = 1$ to $m - 1$. Note that some attacks may render fulfilling this assumption difficult or even infeasible. For example, let objects be images; then components are pixels and mapping semantically equivalent pixels may require undoing scaling attacks, rotation attacks, mapping cropped images with the corresponding parts of uncropped images, etc.

The recovery procedure based on a posterior mixture can be illustrated as shown in Figure 3.8 and be described as follows:

Algorithm 8 (Posterior mixed recovery)

1. *Mix the attacked watermarked objects, by computing*

$$\hat{X}_{postmix} = f(\beta_1, \cdots, \beta_m, \hat{X}_1, \cdots, \hat{X}_m)$$

   where f is a componentwise mixture function (mixing semantically corresponding components, see Assumption A3 above) and β_j, for $j = 1, \cdots, m$ are weights such that $0 \leq \beta_j \leq 1$ and $\sum \beta_j = 1$.

2. *Use the recovery algorithm of the common watermarking method to recover the embedded watermark from $\hat{X}_{postmix}$.*

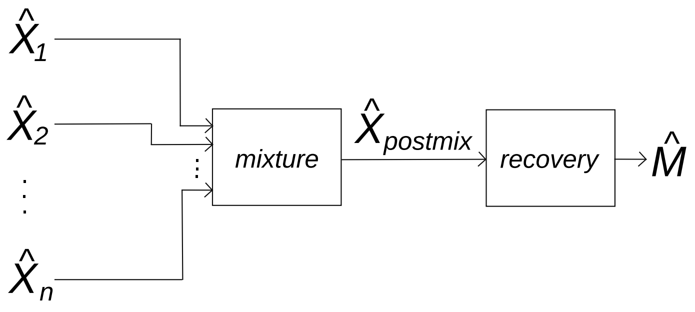

Figure 3.8: Posterior mixture mark recovery procedure.

Algorithm 8 must be regarded as a last chance to repair an otherwise unrecoverable attacked watermark. Posterior mixture can be used as a second line of defense in combination with prior mixture, *i.e.* prior mixture can be used before the attacks happen and posterior mixture after the attacks have happened: in this case, the attacked $\hat{X}_1, \cdots, \hat{X}_m$ would originate from the same prior mixed object X'_{premix} (see Expression (3.1)).

Applying posterior mixture

Using the oblivious scheme described in Section 3.3 a sequence of 35 bits was embedded into the benchmark image Lena. The embedded sequence was encoded using the $(31,5)$ dual binary Hamming code.

Two attacks were performed on the watermarked image: the first one consisted of JPEG compression with quality 20, and the second one was a sharpening filter. From none of both attacked images could the watermark be recovered.

Posterior additive mixture with weights $\beta_1 = \beta_2 = 0.5$ was used to mix both attacked images. In other words, the arithmetic average of color levels for semantically corresponding pixels in the attacked images was computed; since neither compression nor sharpening attacks alter the size nor the orientation of images, semantically corresponding pixels are those occupying the same position in both images. The watermark was recoverable from the posterior mixed image, with an average number of 2.5 corrected errors per codeword, well below the correcting capacity of the $(31,5)$ dual binary Hamming code (7 errors).

Exactly the same experiment was successfully repeated with other benchmark images, like Skyline_arch and Bear [Bas]. For those images, the average number of corrected errors per codeword were, respectively, 6 and 5.7, which are already closer to the correcting capacity of the code. Thus, the effectiveness of posterior mixture depends on the particular image, method and attacks being dealt with.

3.5 Invertible spread-spectrum watermarking for image authentication

In Section 2.2.2 invertible watermarking for image authentication is introduced. Next, we show how to invert under certain conditions one of the most widely known robust oblivious watermarking methods, namely the Hartung-Girod [HG98] spread-spectrum spatial-domain watermarking algorithm. This invertibility can be used to construct an image authentication scheme, as shown in this section. Results presented here have been published in [DS02b].

3.5.1 Hartung-Girod spread-spectrum watermarking

In [HG98], a spread-spectrum technique is used to obtain an oblivious watermarking method in the spatial domain. Oblivious watermarking does not require the original image to recover the watermark embedded in the watermarked image. We will first recall the fundamentals of this method and then we will show that, under certain conditions, this kind of watermarking is invertible.

The embedding and recovery procedures of [HG98] are as follows:

Embedding The copyright information to be embedded is a binary sequence $a_j, \quad a_j \in \{-1, 1\}$. This discrete signal is spread by a large factor cr, called chip-rate, to obtain the sequence $b_i = a_j, \quad j \cdot cr \leq i < (j+1) \cdot cr$. The spread sequence b_i is amplified by a locally adjustable amplitude factor $\alpha_i \geq 0$ and is then modulated by a binary pseudo-noise sequence $p_i, \quad p_i \in \{-1, 1\}$ generated from a seed s (which acts as the secret key). Let x_i be the original signal to be marked. The resulting watermarked signal is

$$x_i' = x_i + \alpha_i \cdot b_i \cdot p_i \tag{3.2}$$

Recovery Mark recovery is performed by demodulating the watermarked signal with the same pseudo-noise signal p_i that was used for embedding, followed by summation over the window for each embedded bit, which yields the correlation sum c_j for the j-th information bit $c_j = \sum_{i=j\cdot cr}^{(j+1)\cdot cr-1} p_i \cdot x_i' \approx \sum_{i=j\cdot cr}^{(j+1)\cdot cr-1} p_i^2 \cdot \alpha_i \cdot b_i$. The sign of $c_j \approx cr \cdot \overline{\alpha}_j \cdot b_i = cr \cdot \overline{\alpha}_j \cdot a_j$, where $\overline{\alpha}_j = \sum_{i=j\cdot cr}^{(j+1)\cdot cr-1} \alpha_i / cr$, is interpreted as the embedded bit $\hat{a}_j$.

With the above method, several watermarks can be superimposed (multiple marking) if different pseudo-noise sequences are used for modulation. This is due to the fact that different pseudo-noise sequences are in general orthogonal to each other and do not significantly interfere [Nic88].

3.5.2 Inverting spread-spectrum watermarks

In order for the above watermarking scheme to be totally invertible, the following three conditions must be met:

1. The seed s used to generate the pseudo-noise signal p_i must be known. Being able to re-create p_i is needed to recover the embedded bits (see Algorithm 9 below).

2. The locally adjustable amplitude factor α_i used at each sample of the watermarked signal during the embedding phase must be known. α_i is needed to invert Equation (3.2) as shown in Equation (3.3). This requirement can be easily met by using a constant value α for all samples.

3. For every sample x_i to be modulated, its modulated value $x'_i = x_i + \alpha_i \cdot b_i \cdot p_i$ must fall within the same range of the original values x_i (otherwise truncation would be needed, which would hamper invertibility).

Assuming that the above three conditions are met, Algorithm 9 shows how the original unwatermarked signal x_i can be recovered from x'_i:

Algorithm 9 (Spread-spectrum watermark inversion)

1. *Recover all embedded bits, where the j-th embedded bit $\hat{a}_j \in \{-1, 1\}$ is obtained as the sign of the correlation sum $c_j = \sum_{i=j\cdot cr}^{(j+1)\cdot cr-1} p_i \cdot x'_i$.*
2. *Spread the recovered sequence $\hat{a}_j$ by the chip-rate cr value, to obtain the sequence $\hat{b}_i = \hat{a}_j, \quad j \cdot cr \leq i < (j+1) \cdot cr$.*
3. *Recover the original $\hat{x}_i$ sequence by computing*

$$\hat{x}_i = x'_i - \alpha_i \cdot \hat{b}_i \cdot p_i \tag{3.3}$$

Note that $\hat{x}_i = x_i, \ \forall i$ if $\hat{a}_j = a_j, \ \forall j$, *i.e.* if the embedded bits are correctly recovered, the unwatermarked image will match the original one. This can be readily seen by comparing Equations (3.2) and (3.3).

3.5.3 Image authentication using invertible spread-spectrum spatial-domain watermarking

Based on the spatial-domain spread-spectrum watermarking described above, we next adapt the ideas of [FGD01a] (reproduced in Section 2.2.2) to give a construction that, given an image, allows the hash of the image to be embedded in its pixels; the hash is used as a MAC (Message Authentication

Code). Anyone knowing the embedding key and the amplitude factor α is able to recover the embedded MAC, undo the watermark, get the original image and check for MAC validity.

Without loss of generality, we will assume a monochrome image in what follows. Let the original image be $X = \{x_i : 1 \leq i \leq n\}$, where x_i is the color level of the i-th pixel and n is the number of pixels in the image. Let x_i be the grayscale level of the pixel, which is assumed to take integer values between 0 and $MAXCOLOR$.

Invertible addition

One of the three conditions stated above for a watermark to be invertible is that the value of a modulated pixel must fall into the grayscale range of original pixels. In [HJRS99], modular addition modulo $MAXCOLOR$ is proposed as another way to keep modulated pixel values within $[0, MAXCOLOR]$. In [FGD01a], this operation is criticized because of possible visual artifacts in the watermarked image resulting from grayscale values close to 0 being flipped to grayscale values close to $MAXCOLOR$, and grayscale values close to $MAXCOLOR$ being flipped to values close to 0 (nearly white pixels become nearly black and conversely). We claim that, in addition to visual artifacts, modular addition may lead to incorrect watermark recovery when inverting the Hartung-Girod watermarking. This is illustrated by the following example.

Example 1 *In the watermarking procedure described in Section 3.5.1, assume that* $MAXCOLOR = 255$, $\alpha = 3$ *and that we want to embed* $a = 1$ *in the first four pixels of the original image. If the values of those four pixels are* $(v_0, v_1, v_2, v_3) = (1, 2, 3, 2)$ *and the first four bits of the pseudorandom sequence are* $(p_0, p_1, p_2, p_3) = (-1, 1, 1, -1)$, *we spread* a *over four pixels to* *obtain* $(b_0, b_1, b_2, b_3) = (1, 1, 1, 1)$ *and compute*

$$x'_i = x_i + \alpha \cdot b_i \cdot p_i, \quad \textit{for } i = 0 \textit{ to } 3$$

This yields $(x'_0, x'_1, x'_2, x'_3) = (254, 5, 6, 255)$. *Values 254 and 255 result from modular reduction of* -2 *and* -1 *(which were out of range). Now, when trying to recover the embedded bit, we compute*

$$c = \sum_{i=0}^{3} p_i \cdot x'_i = -254 + 5 + 6 - 255 = -498$$

Since the sign of c *is negative, we reach the erroneous conclusion that the embedded bit was* $\hat{a} = -1$.

A better way to keep modulated pixels within range is to pre-process the image in the following simple way:

Algorithm 10 (Gray-level pre-processing(α))

```
For $i = 1$ to $n$ do:
    1. If $x_i < \alpha$ then $x_i := \alpha$
    2. If $x_i > MAXCOLOR - \alpha$ then $x_i := MAXCOLOR - \alpha$
```

Algorithm 10 does indeed result in some non-invertible distortion, so when watermarking is inverted, there may be some slight difference between the grayscale values of some pixels of the original and the watermarked images. However, the advantages over modular addition are clear:

- There are no visual artifacts in the watermarked image, because the magnitude of grayscale changes is at most α levels.
- Erroneous bit recovery illustrated in Example 1 is avoided.

Hash embedding and verification

As shown in Section 2.2.2, invertible watermarking for image authentication consists of computing a hash of the original image and embedding the hash bits in the image. Our embedding algorithm takes as input the pre-processed image resulting from Algorithm 10 and depends on two parameters: a seed s for pseudo-random number generation and the amplitude factor α.

Algorithm 11 (Hash embedding(s,α))

1. *Compute the hash $\mathcal{H}$ of the pre-processed image X.*
2. *Construct the sequence a_i to be embedded by doing, for $i = 1$ to $|\mathcal{H}|$:*
   - *If $\mathcal{H}_i = 0$ then $a_i := -1$*
   - *If $\mathcal{H}_i = 1$ then $a_i := 1$*
3. *Using the spread-spectrum Hartung-Girod technique with parameter α and seed s, embed the sequence $\{a_i : i = 1, \cdots, |\mathcal{H}|\}$ into X. Let X' be the resulting watermarked image.*

The algorithm for image authentication, *i.e.* for verification of image integrity, is now straightforward:

Algorithm 12 (Integrity verification(s,α))

1. *Use Algorithm 9 to recover the embedded sequence $\hat{\mathcal{H}}$ and the unwatermarked image $\hat{X}$ from X'.*
2. *Compute the hash of $\hat{X}$, $\mathcal{H}(\hat{X})$*

3. *Compare* $\mathcal{H}(\hat{X})$ *with the hash* $\hat{\mathcal{H}}$. *If they agree, then* $\hat{X} = X$ *and the image is deemed authentic. If they do not,* $\hat{X}$ *is deemed non-authentic.*