Huajian Liu, Martin Steinebach, Marcel Schneider
Fraunhofer SIT, Darmstadt, Germany
{liu,steinebach}@sit.fraunhofer.de

EFFICIENT WATERMARK EMBEDDING FOR WEB IMAGES

Abstract: Digital watermarking is a promising solution for image piracy tracing, because it can generate unique individual image copies by embedding transaction watermarks. For web applications, however, it is difficult to integrate watermarking technique for real time embedding because the computational demand of the embedding process is usually too high for web servers. In this paper, we present a solution to this problem by using the watermarking container concept. The implementation problems and corresponding solutions for applying the container approach to web images are addressed. Experimental results demonstrate the efficiency of the proposed solution.

Keywords: digital watermarking, watermarking container, web images

1. Introduction

Nowadays digital content protection has become a pressing demand for various media published on the Internet, such as digital image, audio and video. For example, it is now very easy for a common user to download a copy of any image displayed on web pages. Therefore, after the images on web pages are transferred to the browser on client side, the protection of the image content is out of the owner’s control. These images can be easily copied and re-used by any recipient [1]. As each individual copy of digital image is identical, it is very difficult, if not impossible, for the content provider to identify the source of the pirate copies. Therefore, a solution to identify different copies or recipients is becoming of great concern.

Digital watermarking is more and more being accepted as a promising solution to this problem. It can make every delivered copy different from each other by embedding a unique ID into each copy, also known as digital fingerprinting or transaction watermark. The embedded ID represents a recipient’s identity. When a recipient reuses his unauthorized copy and the redistributed copy is acquired by the content provider, the source of the redistribution can be identified by examining which ID is contained in the suspicious copy. The fingerprinting process is done without changing the file format, file size and the perceived quality of the content. Each copy is still perceptually identical after watermarking. At the same time the embedding process is usually not reversible. It is hard or impossible to remove the embedded data without significantly degrading the quality of the cover.

Many different digital watermarking and fingerprinting algorithms have been proposed in the literature [2]. Most of attention has been paid to the improvement of the characteristics of the embedded watermark, especially transparency, robustness and security [3]. In order to achieve better transparency, higher robustness and security, more complex watermarking algorithms have been designed. For instance, frequency domain algorithms embedding watermarks by modifying image spectral components have been shown to be more robust and transparent that pixel domain ones. However, when embedding a watermark in a domain other than the pixel domain, transformation to and from the desired domain is necessary. In addition, complex perceptual model is usually involved for good transparency. All these improvements will inevitably increase the algorithm complexity and the computation cost.

Many web applications, however, require a low computational demand of watermarking. A good example is the common web page service on a Web Server. In order to deliver unique image copies on the web pages to every visitor, multiple copies of the original image file are generated upon request and each copy is individually watermarked. A recipient's ID (e.g. a visitor's IP address or a customer's ID) has to be embedded into the images as soon as a request is received. Any latency is not desired. For a busy server, there might be hundreds of concurrent requests. In this case, the computation cost of most common watermarking algorithms will be too high for a web server to keep timely response to every visitor.

In this paper, we introduce an efficient embedding strategy, watermarking container [4], for embedding individual watermarks into web images. In Section 2 and 3, the web application scenario and the watermark container concept will be introduced. Implementation problems and solutions will be addressed in Section 4. The experimental results will be presented in Section 5. In Section 6, we discuss a few traceability and security issues and the paper is concluded in Section 7.

2. Web Application Scenario

In the Web Server scenario as shown in Figure 1, a user visits the web pages on a server via a web browser. First, the browser client sends an http-request for a web image to the web server. The server then generates a unique ID for the user. The ID is stored in a database together with the collected information about the client, such as customer information if available, IP address, time stamp, and so forth. After that, an image copy with the user ID embedded will be produced and delivered to the browser client.

After receiving the watermarked image, the user can view and re-save the image on his computer. If the image is later found published anywhere, the user ID can be extracted from the suspicious image. With the user ID, the user information can be retrieved from the database to identify the source of piracy.

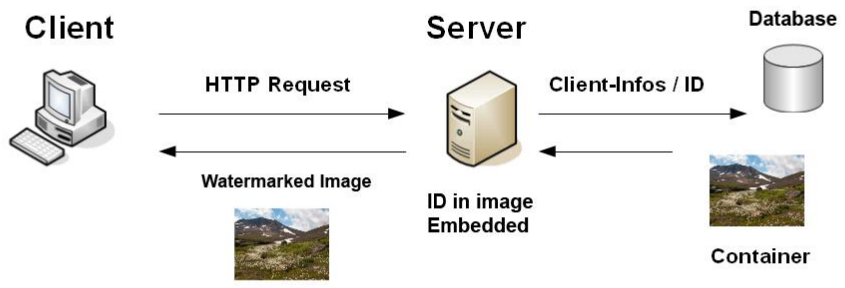

Figure 1. Transaction watermarking scenario for web images

3. Watermark Container Concept

The watermark container concept was first proposed in [4], where it was applied to audio data. The similar concept can also be applied to image data. Figure 2 illustrates the basic principle of watermarking container concept. The watermark container technique splits the watermark embedding into two stages: the watermark preprocessing and the watermark rendering.

In the first stage, the original cover data is divided into units (e.g. image blocks). In every unit of the original cover image, the watermarked results of embedding a bit ‘1’ or a ‘0’ are respectively calculated with the specified embedding parameters, such as embedding strength, secret key and so on. The two watermarked units are denoted as A(0) and A(1). Then we calculate the different signals from the original data as

$$DA(0) = A - A(0), \qquad DA(1) = A - A(1) \tag{1}$$

where A is the original image unit. Then the original unit A and the different results DA(0) and DA(1) are stored in a so-called watermarking container file. To save storage space, DA(0) and DA(1) can be first compressed before storing. Therefore, the container file contains the original image blocks and the two difference signals.

In the second stage, different copies marked with an arbitrary watermark sequence can be rendered efficiently from the container file. For example, when a bit ‘0’ should be embedded, the corresponding difference signal will be added on the original image block to generate the watermarked block. In this stage, no watermarking parameter is needed.

In web applications, the watermark preprocessing stage can be prepared in advance. Thereafter, the speed of this stage is not that crucial. The web server will only need to store the watermarking container file and perform the rendering stage. Since the most computation is finished by the preprocessing stage and only simple mathematic operation is needed for watermark rendering, the computation demand on the web server is significantly alleviated.

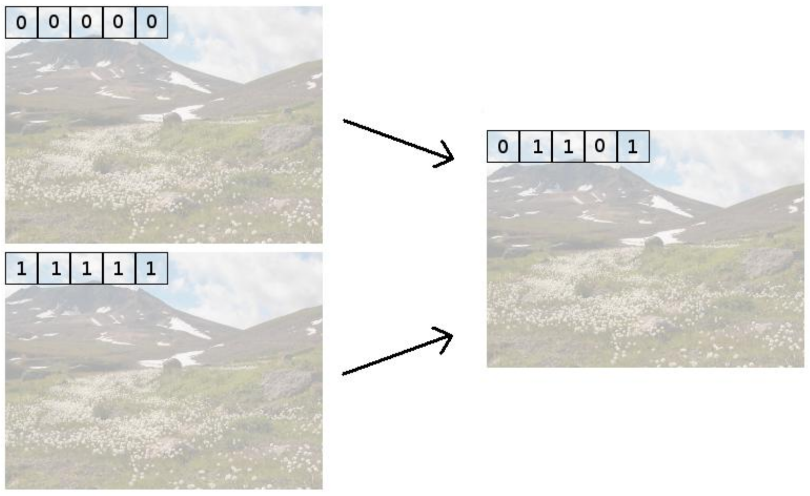

Figure 2. Illustration of the basic principle of watermarking container

4. Implementation Problems and Solutions

In this section, some implementation problems and their corresponding solutions are addressed. In order to apply the watermarking container concept, some assumptions must be met by the selected watermarking algorithm and image format.

In the watermarking container, watermarked copies are rendered by composing prepared image blocks according to the binary bit sequence of a given watermark ID. The prepared blocks must be embedded with ‘0’ or ‘1’ separately. Therefore, two prerequisites must be met in order to compose any arbitrary watermarked copy by creating a sequence of the pre-watermarked image blocks:

1. Individual watermark bits must be embedded in separate image blocks, and the embedded bit in each block must be independent of each other.
2. The image blocks used to generate the watermark container file must be of the same size and be independent of each other.

4.1. Image Watermarking Algorithm

All of the image watermarking algorithms that can fulfill the first requirement mentioned above can be used in the watermarking container framework. Generally, most of block-based algorithms could be suitable as long as they are able to embed independent watermark bits into separate image blocks. A typical approach is to embed one bit in every block, so only two watermarked versions (‘0’ and ‘1’) of each block need to be prepared and stored in the preprocessing stage. If more than one bit is embedded in a block, it is still feasible to apply the watermarking container concept, but more watermarked versions have to be preprocessed and stored in the container file, e.g. ‘00’, ‘01’ ‘10’ and ‘11’ for two bits embedded into each block. This may significantly increase the effect of the preprocessing stage and the storage requirement.

In our implementation, we use a block-based watermarking algorithm that embeds watermarks in wavelet domain. One watermark bit is embedded in every 32×32 image block by modulating a random sequence on the corresponding wavelet coefficients as

$$\hat{c}(x,y)=\begin{cases} c(x,y)+\alpha\cdot T(x,y)\cdot s[j], & if\ \ w=1,\\ c(x,y)-\alpha\cdot T(x,y)\cdot s[j], & if\ \ w=0. \end{cases} \tag{2}$$

where c is the original wavelet coefficients and $\hat{c}$ is the watermarked ones. T is the local adaptive watermarking strength factor that is calculated from a perceptual model [5]. w is the desired watermark bit, α is a global controlling factor and s is a random *m*-sequence generated by a secret key. The watermark detection is done by calculating the cross correlation between $\hat{c}$ and s.

Thus, one single watermark bit is embedded in every 32×32 image block, which is independent of other watermark bits. It therefore meets the first requirement listed above.

4.2. JPEG Format Handling

As mentioned above, the image blocks used to compose watermarked copies must be independent of each other. Therefore, a suitable image format should have such a structure that it is possible to exchange image blocks in the decoding process as early as possible. The worst case is that it is not possible to do so until the image is represented in raw bitmap format. In addition, for Web application, in order to save bandwidth and transfer time, small file size is always desirable. Therefore, uncompressed image formats like BMP are not suitable in this application.

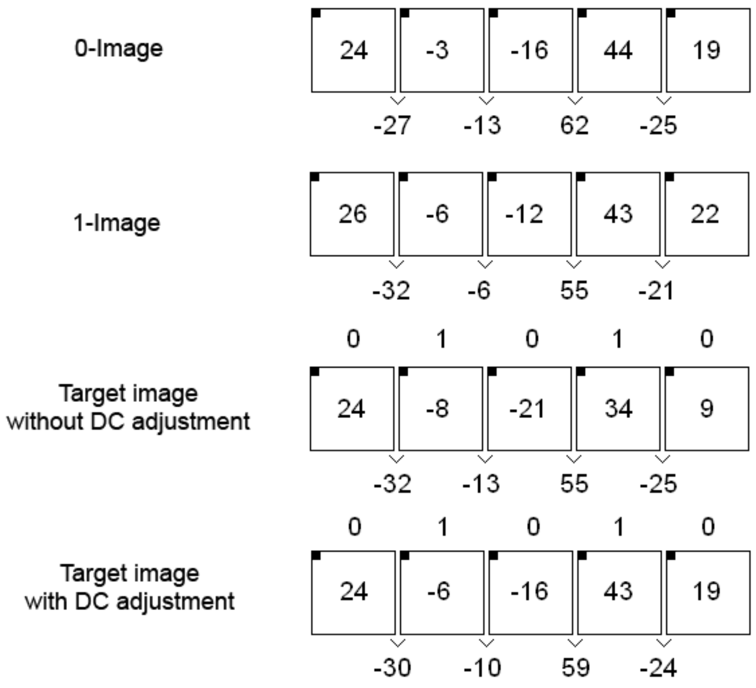

Figure 3. Luminance distortions in watermark rendering progress

Based on these considerations, JPEG format is a good candidate. First, image data in JPEG file is compressed in an 8×8 block based way. And JPEG files are usually of small size. Furthermore, JPEG images are the most widely used image format in Web applications. For these reasons, we use JPEG format in our implementation to realize the container concept.

JPEG standard encodes DC and AC image data in DCT (Discrete Cosine Transform) domain separately [6]. The required condition that each block should be independent of each other is unfortunately only true in the encoding of AC data. The encoding of the DC value of each 8×8 block is based on the DC value in the previous block. Only the difference instead of the absolute DC value will be encoded.

This block dependence problem may be negligible for watermarking algorithms that embed a watermark only by modifying AC coefficients in 8×8 block DCT blocks. But some algorithms also change the DC value during embedding when it is performed in other domains or the used block size is different from 8×8. For example, in our wavelet-based watermarking algorithm, though the embedding is done in the middle frequency band in the wavelet domain, the DC values in DCT blocks are still slightly changed. If the DC values are not adjusted in the composition of image blocks during the watermark rendering stage, luminance and color distortions will accumulate progressively. As illustrated in Figure 3, the two watermarked versions have slightly different absolute DC values in five consecutive blocks in luminance channel. If the corresponding blocks (with bit ‘0’ or ‘1’ embedded) are simply copied to compose the final watermarked version without DC value adjustment, the DC values of the rendered watermarked copy, as shown in the third row, could be significantly shifted after decoding. The shift of the DC values may subsequently cause visible artifacts in the rendered watermarked image. Therefore, a solution to this block dependence problem is essentially necessary.

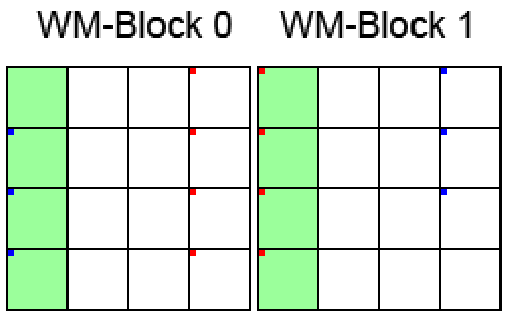

Figure 4. Block crossover of watermark bit transition

In our implementation, we store the necessary DC values in the container file for DC coefficient adjustment in the rendering stage. The absolute DC values of both JPEG blocks that are located on the border of a watermark bit transition need to be stored. For example, a sample image of 64×32 pixels is shown in Figure 4. Since the watermark block used in our wavelet-based watermarking algorithm is 32×32, two watermark bits will be embedded in this sample image, one ‘0’ and one ‘1’. The DC values of all the green blocks except the first one should be corrected during the rendering stage. Therefore, the absolute DC values in the blocks marked with a red or blue dot in Figure 4 should be stored in the container file in the preprocessing stage. Using this additional information, the correct DC values in the green blocks can be quickly calculated.

For the corrections in JPEG bit stream, the old DC bits (both the length and the value) must be first removed, so the length of the DC bit streams of all the green blocks should also be stored. As only one DC value need to be encoded in a block, only a few bits are involved. The new calculated difference DC value (from the stored absolute values) can then be encoded and inserted into the bit stream. In order to do so, the relevant DC Huffman Table for the possible lengths of the involved DC values should be stored in the container file. After re-encoding the DC values in the relevant blocks, the absolute DC values after decoding will be kept correct, as illustrated in the fourth row in Figure 3.

5. Experimental Results

In this section, we present some selected test results, especially comparing the efficiency of the watermarking container and the original embedding algorithm. In order to keep the test results as accurate as possible, all the tests in the following sections have been run three times and then the average values are taken. The first two tests have been performed on a computer with Intel Core 2 Duo T8100@2.1 GHz processor, 3 GB memory and 5,400 RPM hard disk.

5.1. Test on Watermarking Duration and Image Quality

In the first test, we compare the watermarking duration and image quality with/without watermarking container. A test image of size 1024×768 pixels is chosen for the test and stored in JPEG format with a quality factor 90 meeting the quality requirements of most web images. The original file size is 176,111 bytes. A binary sequence of ‘01101101’ is embedded into the image. The test results are shown in Table 1.

From the values listed in Table 1, we can see that the container scheme significantly speeds up the watermarking process. The container method can render the watermarked image approximately 30 times faster than the original embedding process. It takes about 2 seconds to mark the test image without applying the watermarking container, which is evidently inacceptable for a busy web server. When the server gets many concurrent requests and must deliver a lot of different images per second, the watermarking latency of 2 seconds may lead to a few minutes waiting time for an image to be loaded on the client side. Two different embedding strengths have been used in the test. As expected, different embedding strengths won’t affect the computation cost but will change the output file size because the introduced noise-like watermark patterns will decrease the JPEG compression rate. With the same embedding strength, the embedding processes with/without container achieve approximately the same image quality. That means that applying watermarking container won't degrade the quality of final watermarked images.

Table 1. Test results for comparison of watermarking time and image quality

	File Size (Byte)	**PSNR (dB)**	**Trial 1**	**Trial 2**	**Trial 3**	**Avg.**
Original (α=0.02)	147,321	49.34	1.85 s	1.82 s	1.86 s	1.84 s
Container (α=0.02)	171,853	48.65	0.06 s	0.06 s	0.06 s	0.06 s
Improvement	-	-	3083%	3033%	3100%	3066%
Original (α=0.05)	174,165	37.66	1.86 s	1,86 s	1,84 s	1.85 s
Container (α=0.05)	198,627	37.60	0.06 s	0,06 s	0,06 s	0.06 s
Improvement	-	-	3100%	3100%	3066%	3083%

5.2. Test on Watermarking a Group of Images

In this test, we compare the watermarking process with/without watermarking container on a group of images. This test simulates a practical scenario that different images will be watermarked on request. Eight JPEG images are used in this test, which are stored with JPEG quality factors from 90 to 100. The image sizes vary from 1024×768 to 1632×1232. The total size of eight JPEG files is 4 MB. The used binary watermark sequence is '0110110100110100'. The test results are shown in Table 2.

Table 2. Test results of watermarking a group of images

	Trial 1	**Trial 2**	**Trial 3**	**Average**
Original – 1 pass	24.17 s	24.47 s	24.31 s	24.32 s
Container – 1 pass	0.52 s	0.50 s	0.48 s	0.50 s
Improvement	4648 %	4894 %	5065 %	4864 %
Original – 5 passes	121.97 s	121.48 s	120.68 s	121.38 s
Container – 5 passes	1.86 s	1.89 s	2.03 s	1.93 s
Improvement	6558 %	6428 %	5945 %	6289 %

As listed in Table 2, the original embedding process takes about 24 seconds to watermark all the eight images, while the watermarking container can render the eight watermarked images in 0.5 second. The efficiency is therefore improved about 50 times. We also test to watermark all the eight images five times in a row. In this case, about 20% more improvement has been achieved by applying the watermarking container. This is because all necessary components have already been cached after the first run. Similar situation would occur on a web server when different users are attempting to download the same images.

5.3. Test on Watermarking Bit-rate for Various Systems

In this test, we demostrate how many bytes of watermarked images the watermarking container can render per second, from which we can estimate whether the algorithm can fulfill the speed requirement of a specific server. Three computers with different hardware configurations are used in this test as listed in Table 3. System 1 is a Desktop PC that has two hard disks (Raid 1). System 2 is quite an aged computer from the year of 2000 and System 3 is the laptop used in the previous two tests. Since the processor in System 3 has two cores, we have also run a test with two parallel watermarking processes. By testing on these quite different system, we will demonstrate the scalability of the watermarking container.

To calculate the watermarking bit-rate, 50.76 MB image data are watermarked with/without the watermarking contariner on the three systems respectively. The test results are listed in Table 4, where we can see that on System 1 and 3 the watermarking bit-rate is over 15 MB per second. When using two theads in parallel, the bit-rate increases to 35.6 MB per second on System 3. Note that this performance is achieved on a common laptop. On a powerful server, the performance process will be further improved as the watermarking bit-rate is approximately proportional to the speed of the used processor. The speed improvement achieved by the watermarking container is illustrated in Figure 5, comparing the results with/without using the container.

Table 3. List of test systems

	System 1	**System 2**	**System 3**
Processor	AMD Sempron 3000+, 2.0 GHZ	Intel Pentium III 533 MHZ	Intel Core 2 Duo T8100, 2.1 GHZ
Memory	1024 MB	512 MB	3072 MB
Hard Disk	2×3.5’ @7.200 RPM	3.5’ @7.200 RPM	2.5’ @5.400 RPM
Operating System	Windows XP Pro	Windows XP Home	Windows XP Pro

Table 4. Watermarking bit-rate of different systems

Image Data Size: 50.76 MB	Original	Container	Improvement
System 1 – Duration (seconds)	264.48	3.07	8615 %
System 1 – Bit-rate (MB/sec)	0.19	16.5	8615 %
System 2 – Duration (seconds)	704.39	11.89	5924 %
System 2 – Bit-rate (MB/sec)	0.07	4.3	5924 %
System 3 – Duration (seconds)	198.25	2.67	7425 %
System 3 – Bit-rate (MB/sec)	0.26	19.0	7425 %
System 3 (2 threads) – Duration (seconds)	208.90	2.85	7330 %
System 3 (2 threads) – Bit-rate (MB/sec)	0.49	35.6	7330 %

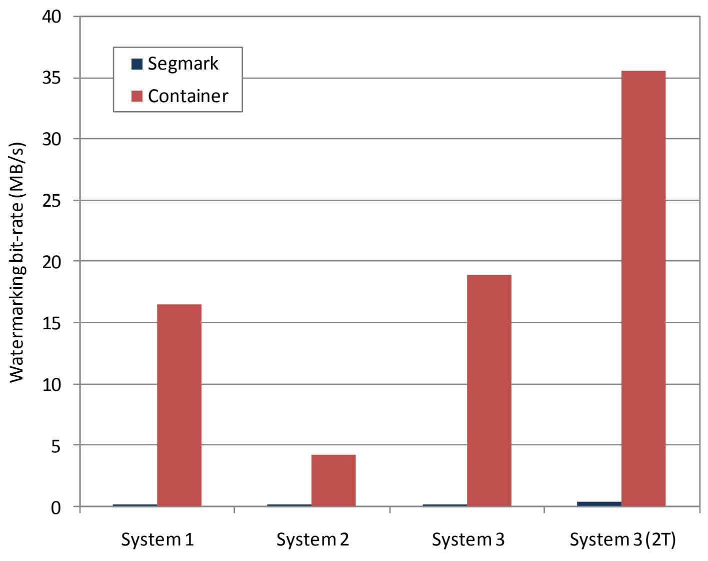

Figure 5. Comparison of the watermarking bit-rate

5.4. Test on Different Watermark Sequences

In this test, we examine the effect on watermarking duration by using different watermark sequences. Because the DC values of JPEG blocks need to be adjusted only at the positions where a bit transition occurs, the duration of image rendering will vary for different watermark sequences. In this test, the same image set as in Section 5.2 is used. Each used watermark sequence is 16-bit long as listed in Table 5.

The bit transition rate decrease from the first sequence to the last one gradually. The first sequence denotes the worst case in which every two adjacent bits are different. The last one gives the other extreme case: no bit transition. From the test results listed in Table 5, we can see that the watermarking duration for the last sequence is approximately 20% shorter than the first one. Figure 6 illustrates visually the watermarking duration trend from the worst to the best case.

Table 5. Test results for different watermark sequences

	Trial 1	**Trial 2**	**Trial 3**	**Average**
1) 1010101010101010	0.56 s	0.50 s	0.49 s	0.52 s
2) 1100110011001100	0.47 s	0.53 s	0.48 s	0.49 s
3) 0000111100001111	0.52 s	0.42 s	0.41 s	0.45 s
4) 1111111100000000	0.39 s	0.41 s	0.47 s	0.43 s
5) 0000000000000000	0.39 s	0.39 s	0.43 s	0.40 s

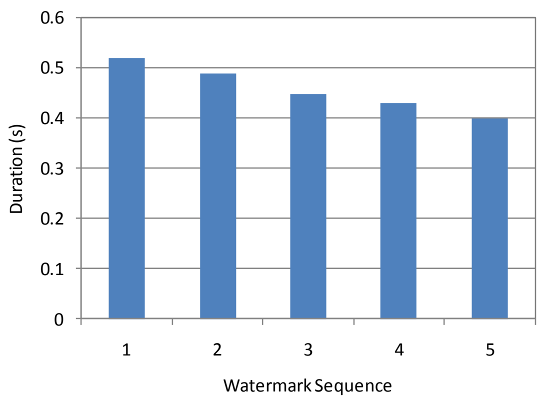

Figure 6. Comparison of watermarking duration for different sequences

6. Discussion

In this section, we discuss several issues regarding the image traceability by embedding available information as user identifiers in different web environments, and the security of digital fingerprinting against collusion attacks.

In order to trace the source of unauthorized copies, the embedded watermark must be able to identify the image recipient. On the public Internet, only the available web-based information, e.g. IP address, may not be enough to exactly identify the recipient. Such information, however, still provides the source of piracy somehow. For instance, although the individual pirate can not be exactly identified by a single IP address, the IP address can still indicate the county, the city, or even the organization where the piracy copies come from.

Furthermore, if the access to the images is somehow limited in a controlled environment, it will be much more effective and efficient to use digital fingerprinting for traitor tracing. For example, in an online shop and online gallery, users have to register first before they can view and download the images. In this case, the registered user ID can be used as a unique identifier and the traceability will be significantly improved.

As well-known, collusion attack is a fundamental security problem for digital fingerprinting. Since every image copy becomes different from each other after embedding a unique identifier, one or many adversaries may collude together to use multiple copies to compose a new copy with no valid identifier embedded to avoid being traced, or even to create a new identifier to frame an innocent user. Block-based watermarking algorithms, which are most suitable for the watermark container framework, make such attacks even easier since each image block and each embedded watermark bit are independent of each other as we mentioned in Section 4.

The problem of collusion attack and corresponding solutions has already been studied in many fingerprinting papers [7][8]. Here we only give a short discussion about the possible countermeasures in the case of web images. One solution is to use the so-called collusion-resistant fingerprint codes [9], which are able to identify the colluders. Unfortunately, such codes are usually too long to be embedded into web images, because they are commonly of small sizes and the watermark capacity is therefore quite low. Another possible countermeasure is the prewarping technique [10]. Before an image is delivered to the recipient, it is first be prewarped in a random way. The prewarping process introduces such slight distortion that the perception quality of the image is not degraded. Since each delivered copy is prewarped in different random ways, when multiple copies are used to make a collage or averaged, visible artifacts will appear and the quality of the composed image will be significantly degraded.

7. Conclusion

The protection of digital images presented on the Internet has become an important challenge where still no solutions have been identified. One approach is transaction watermarking. But the computational cost of watermarking is usually too high for a web server to embed individual ID for every user without significant delays during web page access. In our work, we provide tests results of the implementation of a solution to this problem. We introduce the watermarking container concept, dividing the watermarking process into two stages. Implementation challenges and solutions, regarding suitable watermarking algorithms and JPEG format handling, are addressed. Test results demonstrate that very efficient embedding can be achieved by the proposed solution.

Acknowledgements

This work was supported by CASED (www.cased.de).

References

[1] The Guardian Online, Digital thieves swipe your photos - and profit from them, http://www.guardian.co.uk/technology/2008/jun/18/news.internet.

[2] I. J. Cox, M. L. Miller, J. A. Bloom, *Digital Watermarking*. San Mateo, CA: Morgan Kaufmann, ISBN: 1-55860-714-5, 2001.

[3] LANGELAAR, G. C., I. STEYAWAN, R.L. LAGENDIJK, *Watermarking Digital Image and Video*, IEEE Signal Processing Magazine, September 2000, vol. 17(5), pp. 20-46.

[4] STEINEBACH, M., S. ZMUDZINSKI, F. CHEN, *The Digital Watermarking Container: Secure and Efficient Embedding*, Proceedings of the 6th ACM Multimedia & Security Workshop, MM&Sec 2004, Magdeburg, Germany, September, 2004, pp. 199-205.

[5] LEWIS,A.S., G. KNOWLES, *Image Compression Using the 2-D Wavelet Transform*, IEEE Trans. Image Processing, Apr. 1992, vol. 1, pp. 244-250.

[6] Joint Photographic Experts Group. JPEG File Interchange Format Version 1.02, September 1992, available at http://www.jpeg.org/public/jfif.pdf.

[7] BONEH, D., SHAW, J., *Collusion-Secure Fingerprinting for Digital Data*, IEEE Transactions on Information Theory, 1998, vol. 44, No. 5, pp. 1897-1905.

[8] STEINEBACH M., ZMUDZINSKI S., *Countermeasure for Collusion Attacks against Digital Watermarking*, In Proc. of SPIE Conf. on Security, Steganography and Watermarking of Multimedia Contents VIII, 2006, vol. 6072, pp. 60720D.1-60720D.9.

[9] TARDOS G., *Optimal Probabilistic Fingerprint Codes*, In the Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), 2003, pp. 116–125.

[10] CELIK, M. U., SHARMA, G., TEKALP, A. M., *Collusion-resilient Fingerprinting Using Random Prewarping*, Proceedings of International Conference on Image Processing, 2003, vol. 1, pp. 14-17.