SELF-EMBEDDING AND RESTORATION ALGORITHMS FOR DOCUMENT WATERMARK

Anamitra Makur

School of Electrical & Electronic Engineering Nanyang Technological University, Singapore

ABSTRACT

When a document or a sequence of characters is tampered, watermarking should be able to not only detect the tampering but also restore it. Towards this goal, self-embedding is used where the watermark is a copy of the sequence itself. Restoration algorithms for substitution attack are proposed for two variants of self-embedding. These algorithms are analyzed to show where they fail and how to avoid these. Strategies to cope with deletion/insertion attacks are also mentioned.

1. INTRODUCTION

Watermarking is used on document and other media for diverse purposes such as copyright protection, annotation, authentication, fingerprinting, copy control and broadcast monitoring, and among others, *content protection*. The aim of contect protection is to protect the content of a text document, say, against editing attack. While authentication stops at detecting tampering, localization detects the parts of the document that have been modified. The ultimate aim of content protection is to be able to *restore* the original text.

Typical watermarks that are embedded in a media carry information unrelated or distantly related to the media itself. Examples are trademark, author and copyright information, etc. *Self-embedding* [1] refers to the watermark carrying the same (albeit partial) information as the media. For example, watermark based restoration/recovery for images [1,2] use an watermark which is a compressed or partial version of the image itself. For restoration purpose, information about a part A of the original media should be embedded either in a distributed fashion or localized in another part B of the media. Then, if A is tampered with, the information may be approximately restored from the entire media (distributed) or part B (localized) which is not tampered.

In this work we investigate self-embedding and restoration in a conceptual framework of text documents. We are not aware of any prior work on document watermark towards restoration. It is assumed that a printed or handwritten text document consists of a sequence of characters (from a finite set), and we are interested in restoring only the original character sequence and not its appearance such as font. Watermarking for printed or handwritten binary documents is done in two ways. First group of schemes such as line/word shifting, bounding box method, and hybrid method, change the space between text objects [3]. In second group of schemes, boundary pixels are flipped individually or in a group [4], or parameters such as thickness or curvature are modified. Since the capacity of second schemes is more than the first schemes, it is better suited for restoration. For example, 13 characters of 7 bits each are embedded in a signature in [5]. Our framework assumes use of a pixel-flip watermarking and detection scheme that (i) has enough capacity to watermark a character in a printed character, and (ii) detects the watermark blindly without error in absence of any attack. For example, since 7 bits are sufficient to represent the alphanumeric character set, an watermark of 7 bit capacity is embedded in each printed character. Editing of a text document may involve substitution, insertion and deletion. We consider two self-embedding schemes, and propose restoration schemes for these under certain assumptions. A text document with most of it edited is beyond restoration. We explore how much restoration is achievable and investigate under what condition the restoration fails.

2. SELF-EMBEDDING

Let the text sequence be $[C_0, C_1, ..., C_{n-1}]$ of length n, where C_i denotes a character and (C_i) denotes a watermark. Let $[p(0), p(1), ..., p(n-1)]$ be a permutation of $[0, 1, ..., n-1]$. Then watermark of each character $C_{p(i)}$ is embedded in another printed character C_i. Two permutation schemes are considered in this work. In *cyclic* self-embedding, $p(i) = (i - c)\%n$. The cyclic shift c constitutes the secret key. In *random* self-embedding, a random permutation $[p(i)]$ is obtained from a seed. The seed and n constitute the secret key.

Example: A sequence MR BUSH ON TV (without space) under cyclic self-embedding with $c = 7$ will be:

$$\mathrm{M}^{(\mathrm{U})}\mathrm{R}^{(\mathrm{S})}\mathrm{B}^{(\mathrm{H})}\mathrm{U}^{(\mathrm{O})}\mathrm{S}^{(\mathrm{N})}\mathrm{H}^{(\mathrm{T})}\mathrm{O}^{(\mathrm{V})}\mathrm{N}^{(\mathrm{M})}\mathrm{T}^{(\mathrm{R})}\mathrm{V}^{(\mathrm{B})} \qquad (1)$$

Here $p(0)$=$(0 - 7)\%10$=3, therefore the 3rd character C_3=U will be watermarked on the 0th character C_0=M. The watermark (U) embedded in the character M is denoted by $\mathrm{M}^{(\mathrm{U})}$. In general, ith character after self-embedding becomes $C_i^{(C_{p(i)})}$. For random case with a permutation of [2,5,7,8,6,0,3,1,9,4] (obtained from a seed 637 using Matlab function randperm), it is

$$\mathrm{M}^{(\mathrm{B})}\mathrm{R}^{(\mathrm{H})}\mathrm{B}^{(\mathrm{N})}\mathrm{U}^{(\mathrm{T})}\mathrm{S}^{(\mathrm{O})}\mathrm{H}^{(\mathrm{M})}\mathrm{O}^{(\mathrm{U})}\mathrm{N}^{(\mathrm{R})}\mathrm{T}^{(\mathrm{V})}\mathrm{V}^{(\mathrm{S})} \qquad (2)$$

The number of possible permutations of random self-embedding is $n!$. The number of possible shifts in cyclic self-embedding is $n - 1$ since $c = 0$ is not allowed. Since this number for random case is much larger than for cyclic case, random self-embedding is more secure.

3. RESTORING SUBSTITUTIONS

In *substitution*, one (or more) character C_i is replaced by another character $\hat{C}_i$. Originally C_i was embedded with the watermark $(C_{p(i)})$. Now, the watermark detection algorithm extracts some other watermark $(\#)$ from the substituted character $\hat{C}_i$. During

verification phase, each character is checked with its corresponding watermark. For example, in (1), 7th watermark (M) should match the $p(7)$=0th character M. One substitution, therefore, results in two *failures*. At location i, the watermark corresponding to C_i (the watermark is intact since it is in some other location) doesn't match the substituted character $\hat{C}_i$. At location $p(i)$, the watermark $(\#)$ from $\hat{C}_i$ doesn't match $C_{p(i)}$.

However, there is a distinction between these two failures. Consider location $p(i)$. Even though the character $C_{p(i)}$ in this location failed the verification, the watermark in this location $(C_{p(p(i))})$ is fine since it matches the corresponding character $C_{p(p(i))}$. (Notation $p(p(i))$ means permutation of permutation of i. For the cyclic example, for i=0, $p(0)$=3, and $p(p(0))$=$p(3)$=6.) Assuming that an accidental match is unlikely, we therefore conclude that location $p(i)$ is indeed untampered. Thus, the *untampered* locations are (i) locations whose characters pass the verification, or (ii) locations whose watermarks pass the verification.

3.1. Restoration algorithms

The proposed restoration algorithm for cyclic self-embedding is:

1. Find all *untampered* locations.
2. Declare the remaining locations as *tampered* locations. If location i is *tampered*, then its watermark is embedded in location $(i+c)\%n$. If location $(i+c)\%n$ is also *tampered*, declare failure to restore. Else, restore C_i from (C_i) found in location $(i+c)\%n$.

Example (contd.): Let the cyclic self-embedded sequence of (1) be tampered to MR BEAN ON TV, so that the received sequence is (with substituted characters shown in italics)

$$\mathrm{M}^{(\mathrm{U})}\mathrm{R}^{(\mathrm{S})}\mathrm{B}^{(\mathrm{H})}E^{(\#)}A^{(\$)}N^{(\%)}\mathrm{O}^{(\mathrm{V})}\mathrm{N}^{(\mathrm{M})}\mathrm{T}^{(\mathrm{R})}\mathrm{V}^{(\mathrm{B})}$$

Verification at location 0 is to check M=(M) (watermark of location (0+7)%10=7) or not, which passes. While characters in locations 0,1,2,9 pass the verification test, watermarks in locations 6,7,8 also pass the test. So *untampered* locations are 0,1,2,6,7,8,9. Therefore *tampered* locations are 3,4,5. Since corresponding watermarks in locations 0,1,2 are all *untampered*, all substitutions are restored successfully to the original sequence.

The proposed restoration algorithm for random self-embedding is:

1. Find all *failure* locations. A verification failure in location i means $(C_i) \neq C_i$. Therefore, either the character C_i is tampered, or the character C_j that carries (C_i) is tampered (or both) where $p(j) = i$. For each *failure* location this pair (i, j) is stored in a *pair* array.
2. If a location of a pair in the *pair* list is in *untampered* list, then its partner must be tampered. Therefore, remove this pair from the *pair* list and insert its partner into the *tampered* list.
3. If a location of a pair in the *pair* list is in *tampered* list but its partner is not, then its partner must be fine. Therefore, remove this pair from the *pair* list.
4. Repeat steps 2 and 3 until: (i) no *pair* is left–then declare the *tampered* locations; or (ii) *pair* is non-empty but no further change occurs–then the algorithm fails to resolve which one is tampered with, and inserts both locations of the remaining pairs into the *tampered* list. Finally, the *tampered* locations are restored, if possible, as in step 3 of the cyclic restoration algorithm.

Example (contd.): For the random self-embedded sequence of (2) and the same tampering as before, the received sequence is

$$\mathrm{M}^{(\mathrm{B})}\mathrm{R}^{(\mathrm{H})}\mathrm{B}^{(\mathrm{N})}E^{(\#)}A^{(\$)}N^{(\%)}\mathrm{O}^{(\mathrm{U})}\mathrm{N}^{(\mathrm{R})}\mathrm{T}^{(\mathrm{V})}\mathrm{V}^{(\mathrm{S})}$$

After verification, locations 0,3,4,5,6,8 are *failure*. Since failure in location 0 means tampering in either location 0 or location 5, a pair (0,5) is stored. Continuing, the *pair* array becomes

[(0,5) (3,6) (4,9) (5,1) (6,4) (8,3)].

Since the *untampered* locations are 0,1,2,7,8,9, in step 2 the first pair (0,5) is removed from this array and location 5 is inserted into the *tampered* list. After step 2 the *pair* list is [(3,6) (6,4)] and the *tampered* list is 3,4,5. Note that unlike the cyclic case, checking the *untampered* list is not sufficient. In fact, location 6 does not appear in the *untampered* list, and if the algorithm stops here then location 6 will also be taken as tampered. This justifies the need for steps 3 and 4. In step 3, the pair (3,6) contains a tampered location 3, so it is removed. Similarly, (6,4) is also removed, and the algorithm terminates with substitution locations 3,4,5, all of which are restored successfully.

3.2. Analysis of the algorithms

The restoration algorithms may fail in two ways; *restoration failure* when it cannot restore a detected location, and *extraneous detection* when it detects an untampered location as tampered. While such failures are unavoidable for a large number of substitutions, they also occur for a few (even 1) substitutions, which is a more serious limitation of this kind of restoration. Below we analyze and quantify such cases for both self-embedding schemes.

A permutation $[p(i)]$ has a *fixed point* if $i = p(i)$ for some i. Then C_i is embedded with its own watermark. Therefore, C_i can not be restored if it is substituted. For cyclic self-embedding, since $c = 0$ is not used, there is no fixed point. For random self-embedding, let the number of permutations with exactly k fixed points be $F_k(n)$ for length n sequences. These permutations are obtained by choosing k locations out of n, placing $p(i) = i$ in these k locations, and choosing any permutation without fixed point for the remaining $n - k$ locations. Therefore, $F_k(n) = {}^nC_kF_0(n-k)$ for $0 \leq k \leq n$, where $F_0(0) = 1$ is assumed. Sum of the permutations with $k = 0, ..., n$ fixed points equals all possible permutation $n!$. Therefore,

$$n! = \sum_{k=0}^{n} F_k(n) = \sum_{k=0}^{n} {}^nC_kF_0(n-k). \tag{3}$$

Assuming all permutations are equally likely, the probability that a random permutation will be free of fixed point is $p_N(n) = F_0(n)/n!$. From (3) the following recursion is obtained.

$$\sum_{k=0}^{n} p_N(n-k)/k! = 1$$

If any random permutation is used, then for a single substitution, the probability of restoration failure is obtained by multiplying the probability of k fixed points with the probability of the substitution falling in one of these k locations, and summing for all $k \neq 0$.

$$\sum_{k=1}^{n} \frac{F_k(n)}{n!} \cdot \frac{k}{n} = \sum_{k=1}^{n} \frac{p_N(n-k)}{k!} \cdot \frac{k}{n}$$

Figure 1 (left) shows this probability which, unfortunately, is significant. Therefore, it may be prudent to use only those permutations that are free of fixed points. Unfortunately, $p_N(n)$ converges to a small value for large n as shown in figure 1 (right). Thus, only about 37% of the random permutations are free of fixed points.

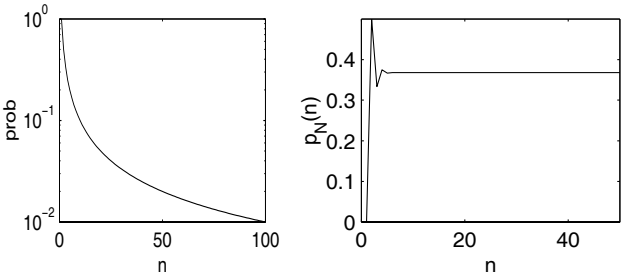

Fig. 1. Probability of restoration failure in random permutation for a single substitution (left) and probability of fixed point free random permutations (right) for different sequence length n

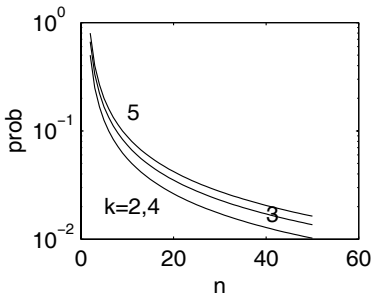

Fig. 2. Probability that a cyclic permutation has length k attractor cycle (for $k = 2$ to 5) for different sequence length n

A permutation has an attarctor cycle of periodicity 2 if $i = p(p(i))$ for some i. Let $j = p(i)$ in this case. Then two characters C_i and C_j are embedded with each other's watermark. If both of them are substituted, neither can be restored, and two restoration failures occur. If any one of them is substituted, both locations appear in the *failure* list, and there is no way to determine which one is untampered (for both cyclic and random cases). Therefore, both locations are declared *tampered* and cannot be restored. Thus, extraneous detection occurs along with restoration failure. Clearly, such cycles should be avoided. For cyclic self-embedding, a length 2 cycle exists for $(i, i + c)$ only when n is even and $c = n/2$. For random permutations the possibility is more, and as before, only about 23% permutations are free of length 2 cycles.

Larger length cycles affect the performance in a similar way. A length k cycle causes restoration failure if k substitutions occur at these locations. If $k/2$ or more substitutions occur at these locations such that no two substitutions are more than 2 positions apart on the cycle, then it causes extraneous detection and restoration failure. For example, for a $k = 4$ cycle p, q, r, s, substitutions at p, r will cause all four locations to be identified as *tampered*. For cyclic permutation, such cycle occurs for specific shifts. For example, length 3 cycle is present if n is divisible by 3 and $c = n/3$ or $2n/3$. Length 4 cycle occurs if n is divisible by 4 and $c = n/4$ or $3n/4$. The probability (fraction) of cyclic permutations having length k cycle(s) may be found as follows. Assume length n is locally equiprobable, so that probability of n divisible by k is $\frac{1}{k}$. Let D_k denote the set of divisors of k excluding 1 and k. Then length k cycles occur for shifts $c = \frac{ni}{k}$ where $i \in \{1, ..k-1\} \backslash D_k$ (where $\backslash$ denotes set exclusion). For example, for $k = 4$, $D_k = \{2\}$ and $c = \frac{ni}{4}$ gives length 4 cycles where $i \in \{1, 3\}$. So $k - 1 - |D_k|$ shifts out of possible $n - 1$ give length k cycle. Therefore, assuming equal likelihood of any shift, the probability of a cyclic permutation having length k cycle is $\frac{k-1-|D_k|}{k(n-1)}$. These probabilities for $k = 2$ to 5 are plotted for various length n in figure 2. If length k cycle is present, k substitutions in locations $i, i+c, ..., i+(k-1)c$ will result in restoration failure. The probability of such substitution is

$$1 \cdot \frac{k-1}{n-1} \cdot \frac{k-2}{n-2} \cdots \frac{1}{n-k-1} = 1/^{n-1}C_{k-1}.$$

Product of both these probabilities gives the probability of failure of k substitutions due to a length k cycle.

Therefore, for composite n, shifts which are large factors of n or their multiples should be avoided. Note that as k becomes large, the probability of failure from a length k cycle becomes small. Therefore, only smaller length cycles are cause for concern. For random permutations the situation is same as before. The probability of a random permutation having m cycles of length k each is independent of n (for large n). These probabilities for $k = 1$ to 4 (length 1 attractor is a fixed point) are plotted in figure 3 for small values of m. It is indeed advisable to choose a random permutation that is free of all small length cycles.

While above failures occur due to the permutation and can be avoided, failures happen due to the restoration algorithms, too. For example, let there be two substitutions at locations i and $p(i)$. Since location i is *tampered*, watermark of $C_{p(i)}$ is no longer available, and location $p(i)$ cannot be restored. This occurs independent of any (cyclic/random) permutation. As another example, consider two substitutions at locations i and $p(p(i))$. After verification, *failure* locations are i, $p(i)$, $p(p(i))$, $p(p(p(i)))$. While location $p(p(p(i)))$ is eventually declared *untampered* by the algorithms, location $p(i)$ remains *tampered* since its watermark $(C_{p(p(i))})$ cannot be verified. Thus, extraneous detection occurs resulting in restoration failure for location $p(p(i))$. This is a serious algorithm failure, since the watermark of $C_{p(p(i))}$ is actually present and untampered in the received sequence.

Figure 4 shows simulation results on a sequence of length 100. Random and cyclic permutations are chosen to avoid known failures. Random substitutions are made with varying probabilities of 0.01 to 0.3, and the restoration algorithms are applied to estimate the probabilities of restoration failure and extraneous detection.

To compare cyclic and random cases, the number of cyclic permutations is much smaller than the number of random permutations. However, cyclic permutation is free of fixed point. The fraction (probability) of cyclic permutations having length k cycle(s) is small and becomes smaller with larger length n. The permutations (shifts) having such cycles follow certain conditions and hence are easy to detect. On the other hand, the number of random permutations is large. Random permutations may have fixed point and length k cycles. The probability of having such cycles is larger than the cyclic case, and is constant for large n. Further, detection of permutations with cycles is by exhaustive checking. Restoration algorithm for cyclic self-embedding is simpler than that for random self-embedding. Results of figure 4 illustrate the superior performance of random self-embedding.

4. RESTORING INSERTIONS AND DELETIONS

Insertion or deletion of characters pose a much more difficult challenge. In presence of insertion/deletion, the location of the wa-

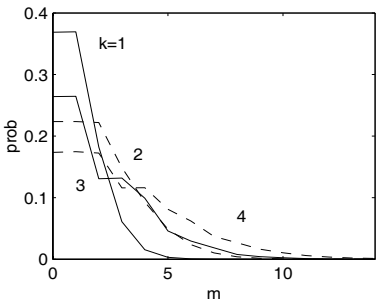

Fig. 3. Probability that a random permutation of large sequence length n has exactly m attractor cycles of length k (k = 1,2,3,4)

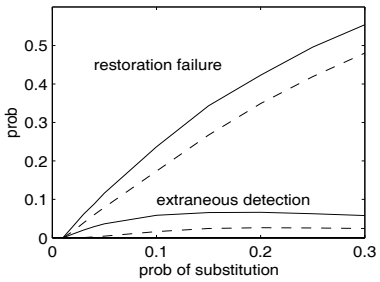

Fig. 4. Simulation results showing both types of failure probabilities for cyclic (solid) and random (dashed) permutations with $n = 100$ for different probabilities of random substitution

termark (C_i) of a character C_i is no longer known, and has to be searched. Since all characters are not distinct, the search may result in multiple or wrong match. We present here some ideas on restoration with the assumption of small length insertion/deletion (reasonable) and locally distinct sequence (not so reasonable), so that multiple/wrong match is ruled out. (Another option is to use sequence number in the embedding to make all characters distinct.)

Consider cyclic self-embedding with m characters deleted from a single location. The verification stage now involves searching the watermark in a local neighborhood. Characters which are located on the same side of the deletion as their watermarks will pass the verification. Let these locations be termed *same*. Characters which are located on one side but their watermarks on another side of the deletion, will show a smaller shift $c - m$. Let these locations be *change*. Characters whose watermarks were embedded in the deleted locations will fail the verification. Each element i of *same* signifies that no deletion has occured between $\{i, ..., i + c\}$ (with modulo operation where appropriate). Overlap (union) of these segments gives the segment without deletion. Similarly, each element i of *change* signifies that deletion has occured between $\{i, ..., i + c\}$. Intersection of these segments gives the segment where deletion took place. From these two hypotheses, the exact deletion location can be determined. Due to cyclic nature, deletion at the beginning is not distinguishable from deletion at the end. The deleted characters can be restored from their watermarks. Note that if $c < m$ then complete restoration is not possible. So a large c is preferred.

Example: Consider the sequence I DONT SAY with $c = 5$ cyclic self-embedding. If locations 3 and 4 are deleted, the received sequence is

$$\mathrm{I}^{(\mathrm{N})}\mathrm{D}^{(\mathrm{T})}\mathrm{O}^{(\mathrm{S})}\mathrm{S}^{(\mathrm{I})}\mathrm{A}^{(\mathrm{D})}\mathrm{Y}^{(\mathrm{O})}$$

During search and verification, watermark (I) of character I (location 0) is found in location 3. Therefore the shift is 3, and location 0 is placed in *change* list. Finally, only location 3 is in *same* since its watermark is 5 characters apart. Locations 0,1,2 are in *change* since their watermarks have shift of 3. Locations 4,5 do not pass the verification. Now, from *same* list we know that locations {3,...,2}, or in between 3–4, 4–5, 5–0, 0–1, 1–2, have no deletion. Also, from *change* list we find 3 segments where deletion occured: {0,...,3}, {1,...,4}, and {2,...,5}. The intersection is {2,3} which declares that $5-3 = 2$ characters are deleted between locations 2 and 3. These are then restored successfully from their watermarks.

When insertion occurs, if the watermarks of the inserted characters are superfluous, then it is easy to detect these locations. If not, then a strategy similar to deletion should be applied. *same* list will contain locations with correctly shifted watermarks. *change* list will contain locations with shift $c + m$. Remaining characters may match some watermark, but likely with a shift other than c and $c + m$, so they are not put in any list. The algorithm then proceeds the same way as deletion.

For random self-embedding, the above strategy does not work. This is because, unless we know the actual position of a character, we do not know where its watermark is expected to be. However, under certain assumptions it is possible to rule out other possibilities and find the actual tampering. More research is needed before a general strategy may be found.

5. CONCLUSION

The idea of self-embedding for document watermarking and restoration is forwarded. Cyclic and random permutations are considered as variants, and each has its advantages. Restoration algorithms for substitution are proposed and analyzed to show where restoration fails. Restoration strategies for single location deletion and insertion with cyclic self-embedding is also proposed. Future work towards multiple deletion/insertion and for random permutation is being pursued. Simultaneous detection (and restoration) of substitution, insertion and deletion will be the ultimate goal towards making document watermarking a powerful tool.

6. REFERENCES

[1] J. Fridrich, M. Goljan, "Images with self-correcting capabilities," Proc IEEE ICIP 1999, vol 3, pp 792–796.

[2] C-Y. Lin, S-F. Chang, "Semi-fragile watermarking for authenticating JPEG visual content," SPIE Proc Security and Watermarking of Multimedia Contents II (vol 3971 no 13), Jan 2000, San Jose.

[3] J. T. Brassil, S. Low, N. F. Maxemchuk, "Copyright protection for the electronic distribution of text documents," Proc IEEE, 87(7), July 1999, pp 1181–1196.

[4] M. Wu, E. Tang, B. Liu, "Data hiding in digital binary images," Proc IEEE Intl Conf on Multimedia and Expo, Jul 31-Aug 2, 2000, New York.

[5] Q. Mei, E. K. Wong, N. Memon, "Data hiding in binary text documents," SPIE Proc Security and Watermarking of Multimedia Contents III, Jan 2001, San Jose.