TDDC03 Projects, Spring 2005

Winnowing, a Document Fingerprinting Algorithm

Norzima Elbegbayan
elbno400@student.liu.se

Supervisor: Tina Lindkvist

Winnowing, a Document Fingerprinting Algorithm

Norzima Elbegbayan
Department of Computer Science
Linkoping University
elbno400@student.liu.se

Abstract

Among digital data, documents are the easiest to copy and remove any signatures or fingerprints embedded, which make the pirating the hardest to detect. Anyone can just retype a document or copy a part of it. Document fingerprinting is concerned with accurately identifying and copying, including small partial copies, within large sets of documents.

We will make a literature study of Winnowing, a fingerprinting algorithm for documents. The Winnowing selects fingerprints from hashes of k-grams, a contiguous substring of length k. We will also show a document fingerprinting example to show the performance of the algorithm.

1. Introduction

The expansion of digital networks all over the world allows extensive access to and use of any digital material. Negative aspects of such access include unauthorized taping, reading, manipulating or removing of data, which might lead to financial loss or legal problems of the producers and creators. Therefore designers, producers and publishers of digital data like documents, images, video or multimedia are seeking technical solutions to these copyright protection problems.

Embedding of unique customer identification as a watermark into digital data, to identify illegal copies of the document and trace back *pirates*, is called *fingerprinting*. In fingerprinting, every copy of the document will get a unique *mark,* representing who is the owner of the document.

To keep track of copies distributed, we produce different copies for each customer. Drawbacks of this procedure are, some people don't like registering themselves in a database, attackers can compare several fingerprinted copies to find and change or destroy the embedded identification, and the more copies we have the bigger the mark gets making it difficult to embed. The more data the mark consists of the more robust it will be, but at the same time the embedding will be more difficult.

However, document fingerprinting is concerned with accurately identifying copying, including small partial copies, within large sets of documents. With pirated or copied documents, comparing whole document checksums is simple and suffices for reliably detecting exact copies; however, detecting partial copies is subtler.

In this report, we will discuss about fingerprinting, its example fingerprinting methods, and document fingerprinting algorithms. The Winnowing algorithm that uses hashing technique for document fingerprinting and its performance measures for detecting full and partial copies will be discussed in detail with an example.

1.1 Background and related work

Research in copyright protection and information hiding has grown recently and a large variety of techniques have been developed. To prevent illegal redistribution, the distributor can try to make it difficult to make a new copy from the distributed document. Usually this will only temporally stop the pirates since the copy protection will probably be cracked. Another way of limiting the illegal distribution is to embed a signature in the document. The signature can be divided into different classes.

The most common class of signatures in digital documents is *watermarking* [8]. In watermarking all distributed documents contain the same signature. For example, the signature can be a company logo or a copyright text. The watermark can also contain information about under which circumstances the document can be used.

Another class of signatures is *fingerprinting* [16], which we will discuss more precisely in this paper. In fingerprinting all distributed documents contain individual signatures. This way the distributor can bind an illegally distributed copy to the pirate.

Much has been published about how to embed signatures into images or video, for they are the most common types of data that deal with copyright protection. For multimedia files some transform domain is used for the embedding of the signature [4]. Well known coding and decoding algorithms can also be used by changing the parameters that control the compression. The company Digimark [15] has developed a search engine that can find registered watermarks in images and report back with the location to the distributor.

1.2 Problem

Among digital data, documents are the easiest to copy and to remove any signatures embedded in them, which makes copy detecting harder. There are various document fingerprinting algorithms, which seem to capture an essential property of any fingerprinting technique guaranteed to detect copies.

It is easy to detect exact copies between documents by just comparing a full document checksum. However, it is rarely the case that whole documents are copied. Generally, pirates use loads of tricks to avoid getting detected by a copy checking mechanism. Some of these are relocating parts of the text, changing document (file) names, rewriting parts of the text and/or changing words with their synonyms. Because of this, copy detection becomes quite challenging [16].

1.3 Method

There are various software and document copy detection mechanisms that detect whole and partial copies of a document, including Moss [12] and SandMark [3], which are widely used over Internet.

In our report we will study algorithms for fingerprinting and detecting copies of a document using hashing techniques. To hash a document we will use the notion k-gram [15], a substring of length *k,* where *k* is chosen by the distributor. Documents will be divided into all possible k-grams, and then the k-grams will be hashed. The main problem is how to choose the fingerprint from the hashes. From several methods for selecting fingerprint we will study Winnowing algorithm which selects the smallest hash value from *w* window slides. With such hashed fingerprints, there are certain bounds for detecting similarity between copied and original documents.

2. Fingerprinting

Digital watermarking offers a supplemental form of protection that can extend into a broader scope. By watermarking, we mean inserting some information into a document, such as an audio track, still image, video stream or text, in such a way that the marked document differs imperceptibly from the original. The information contained in the mark can be recovered by authorized parties, and removal of the mark by modifying the marked document results in rendering the document useless [9].

One of such techniques of embedding a mark into a document for copyright protection is called fingerprinting. To fingerprint the data, unique information is inserted into each copy. This will enable the owner or the distributor to trace an unauthorized copy back to the source.

2.1 Fingerprinting methods

Most fingerprinting schemes are symmetric, which means that both the user and the distributor have access to the fingerprinted data. If an unauthorized copy of such data is found, one cannot assign responsibility to one of them with absolute certainty. Although symmetric fingerprinting algorithms are very helpful for eliminating possible sources, their findings alone cannot convict anybody in the court.

Asymmetric fingerprinting on the other hand creates a copy that only the user knows about [13]. The fingerprint is created by the distributor. First the distributor encrypts the object using the user's public key and than the user decrypts it using his secret key. After the deal is done the user have a uniquely, and tied to him, fingerprinted copy of the data. The distributor does not get this copy. From the public key gained in the key exchange, the distributor cannot create a copy identical to the one that the user has. But, if the user distributes illegal copies of the data, the distributor can identify them and trace them back to the user.

The traitor-tracing fingerprint schemes [2] differ from the other fingerprinting schemes mentioned above as they do not prevent or deter from redistribution of the data, but rather focus on prevention of decryption possibilities. The fingerprint is in the decryption keys, not in the actual data.

2.2 Fingerprinting example

Here we give an illustration to show the basic ideas of fingerprinting. The text we want to fingerprint looks like:

The match was cancelled due to a terrible storm.

The possible synonyms of words used in this sentence are:

due to \ owing to \ on account of the fact there was
match \ game \ contest \ competition
canceled \ called off
storm \ wind
terrible \ dreadful \ strong

From the text we can construct several different versions using word substitution, without the meaning being lost.

The game was cancelled due to a terrible storm.
The match was called off due to a terrible storm.
The match was cancelled owing to a terrible storm.
The match was cancelled due to a strong storm.

We can prevent from illegal distribution by keeping track of who got which copy. The fingerprint will be robust

against pirate attacks as long as the possible substitution word list and user database remain secret. However two or more users can collude and compare their fingerprints to detect the differences and by changing the words that were different can create a new one. If the created fingerprint matches one of the original fingerprints, an innocent user can be framed for illegal distribution.

If we have more copies than our fingerprint could handle, both the mark and user database needs to grow. Another potential problem is that people are not so keen to register themselves in a database due to confidentiality and integrity reasons.

3. Document fingerprinting

Document fingerprinting is a technique for accurate detection of full and partial copies between documents. Here, the idea is to store a small sketch (that is, a representative set of numbers) such that by comparing the sketches between two documents, we will be able to identify whether they have a substantial overlap [16].

According to [5], a digital document fingerprinting scheme consists of a number of marking positions in the document, a fingerprinting algorithm which selects the mark to be embedded for each marking position depending on the number of the copy and embeds it. Another possible content is a pirate tracing algorithm which, on input of a modified document, outputs at least one number of a copy that was used in constructing the modified document.

Different copies of a document containing digital fingerprints differ at most at these marking positions. A powerful attack to remove a fingerprint therefore consists of comparing two or more fingerprinted documents and to alter these documents randomly in those places where a difference was detected. If three or more documents are compared, a majority decision can be applied to improve this kind of attack; for the area where the documents differ, choose the value that is present in most of the documents.

The only marking positions the pirates can not detect are those positions which contain the same letter in all the compared documents. We call the set of these marking positions the intersection of the different fingerprints.

3.1 Document fingerprinting methods

Most of the existing techniques for document fingerprinting or copy detection use a notion of hashing of k-grams [15] or w-shingles [1]. Each document may be considered as a sequence of words in a canonical form (stripped of formatting, capitalization, punctuation etc.). The algorithm divides a document into k-grams or w-shingles, where *k* and *w* are parameters chosen by the user.

The first one, which is considered being very efficient for detecting full and partial copies between documents is hashing of k-grams. A k-gram is a contiguous substring of length *k* [15]. For example, the sequence of 5-grams of the phrase "*A do run run run, a do run run*" is:

adoru dorun orunr runru unrun nrunr runru unrun nruna runad unado nador adoru dorun orunr runru unrun

On the other hand, a contiguous subsequence of words is called a shingle, and specifically a contiguous subsequence of *w* words is a w-shingle [1]. For example, the set of 4-shingles of the phrase *"one two three one two three one two three"* is:

{(one, two, three, one), (two, three, one, two), (three, one, two, three)}

Then hash each k-gram/w-shingle and select a subset of these hashes to be the document's fingerprints. How hashing and selecting of the subset are done will be discussed in Section 3.2. An example algorithm using k-gram is the Winnowing algorithm [15] that selects the fingerprints from a sequence of hashes that guarantees that at least part of any sufficiently long match is detected. We will discuss this algorithm in detail later in Section 4.

[16] introduces a new randomized algorithm that provides a guarantee that with very high probability, any match of greater than or equal to *W* characters (an input parameter) will be detected. This algorithm has small deterministic bounds on the amount of space needed for the algorithm.

The Google search engine uses the technique outlined by Broder [1] to detect copies of web pages while crawling the web. It then tries to display only unique results (removing mirror sites etc.) so that the user has a better selection.

3.2 Document fingerprinting algorithm using hashes of k-grams

Many copy detecting approaches may rely upon gross similarities between documents. One such technique derives a document hash (checksum) for all items in a comparison set. Thereby, identical checksums characterize identical documents. This is a viable means of building a set of document comparators over time. Whenever a new set of documents is checked the historical collection of checksums can be invoked to provide an extended comparator set.

Clearly, such techniques may be refined. For instance, using document sections as a basis for fingerprinting, rather than complete documents affords a finer grain comparison [18]. Such document sections could be either k-grams [15] or w-shingles [1], that are described in above section.

Let us look at an example from [15] to see how hashing and selecting fingerprints from the hashes are done. In the example, 5 is parameter k.

A do run run run, a do run run
(a) Some text.

adorunrunrunadorunrun
(b) The text with irrelevant features removed.

adoru dorun orunr runru unrun nrunr runru unrun nruna runad unado nador adoru dorun orunr runru unrun
(c) The sequence of 5-grams derived from the text.

Now hash each k-gram and select some subset of these hashes to be the document's fingerprints. In all practical approaches, the set of fingerprints is a small subset of the set of all k-gram hashes. If the hash function is chosen so that the probability of collisions is very small, then whenever two documents share one or more fingerprints, it is extremely likely that they share a k-gram as well.

77 72 42 17 98 50 17 98 8 88 67 39 77 72 42 17 98
(d) A hypothetical sequence of hashes of the 5-grams.

For efficiency, only a subset of the hashes should be retained as the document's fingerprints. But which hashes should be selected as fingerprints?

Karp and Rabin's algorithm [7] for fast substring matching is apparently the earliest version for selecting fingerprints based on k-grams. A simple but incorrect strategy is to select every i^{th} hash of a document, but this is not robust against reordering, insertions and deletions. In fact, pre-pending one character to a file shifts the positions of all k-grams by one, which means the modified file shares none of its fingerprints with the original.

Thus, any effective algorithm for choosing the fingerprints to represent a document cannot rely on the position of the fingerprints within the document.

The scheme Manber [11] chose is to select all hashes that are *0 mod p,* for some fixed *p*. In this way fingerprints are chosen independent of their position, and if two documents share a hash that is *0 mod p*, it is selected in both documents. This approach is easy to implement and retains only *1/p* of all hashes as fingerprints.

72 8 88 72
(e) The sequence of hashes selected using 0 mod 4.

A disadvantage of this method is that it gives no guarantee that matches between documents are detected: a k-gram shared between documents is detected only if its hash is *0 mod p*. If the hash function is chosen so that the probability of collisions is very small, then whenever two documents share one or more fingerprints, it is extremely likely that they share a k-gram as well [15]. Consider the sequence of hashes generated by hashing all k-grams of a file in order. Call the distance between consecutive selected fingerprints the *gap* between them [15]. If fingerprints are selected *0 mod p*, the maximum gap between two fingerprints is unbounded and any matches inside a gap are not detected. In Section 4, the Winnowing algorithm [15] for selecting the fingerprints from a sequence of hashes will be discussed. This algorithm guarantees that at least part of any sufficiently long match is detected.

In [6], Heintze proposed choosing the *n* smallest hashes of all k-grams of a document as the fingerprints of that document. By fixing the number of hashes per document, the system would be more scalable as large documents have the same number of fingerprints as small documents.

The price for a fixed-size fingerprint set is that only near-copies of entire documents could be detected. Documents of vastly different size could not be meaningfully compared; for example, the fingerprints of a paragraph would probably contain no fingerprints of the book that the paragraph came from. Choosing hashes *0 mod p*, on the other hand, generates variable size sets of fingerprints for documents but guarantees that all representative fingerprints for a paragraph would also be selected for the book. Broder [1] classifies these two different approaches to fingerprinting as being able to detect only *resemblance* between documents or also being able to detect *containment* between documents.

A fingerprint can also contain positional information, which we do not show, describing the document and the location within that document that the fingerprint came from.

3.3 Detecting resemblance and containment of documents from their hashed k-grams

As mentioned above in Section 1, document fingerprinting is concerned with accurately identifying copying, including small partial copies, within large sets of documents. Therefore, the most probable pirate attack for documents is that either all or part of the document can be copied. Computing the resemblance and containment between the copy and the original could be sufficient for detecting such pirating.

One measure of the resemblance of two text files *A* and *B* is the resemblance of their corresponding sets of k-grams. We therefore define the resemblance *r(A,B)* as

$$r(A,B) = |S(A,k) \cap S(B,k)| / |S(A,k) \cup S(B,k)|$$

Here *S(A,k)* is a set of hashes of k-grams of the document *A.* The resemblance is implicitly dependent on *k*, a pre-chosen fixed parameter. The resemblance is a number between 0 and 1, with a value of 1 meaning that the two documents have the same set of k-grams. Small changes in a large document can only affect the resemblance slightly, since each word change can affect at most *k* distinct grams. Similarly, resemblance is resilient to changes such as swapping the order of paragraphs.

For a fingerprinted document, we store only k-grams that are *0 mod p* for some suitable *p*. Let L(A) be the fingerprint or hashes that are *0 mod p* for the document *A.* Then the estimated value of the resemblance r_e is given by

$$r_e(A,B) = |L(A) \cap L(B)| / |L(A) \cup L(B)|$$

This is an unbiased estimator for the actual resemblance *r(A,B)*. By choosing *p* appropriately, we can reduce the amount of storage for *L(A)*, at the expense of obtaining possibly less accurate estimates of the resemblance. As *L(A)* is a smaller set of k-grams derived from the original set, we call it a sketch of the document *A*. Given sketches for two files *A* and *B* we can compute text resemblance.

Similarly we may define the containment of *A* by *B*, or *c(A,B)* by

$$c(A,B) = |S(A,k) \cap S(B,k)| / S(A,k)$$

Again containment is a value between 0 and 1, with value near 1 meaning that most of the shingles of *A* are also shingles of *B*. In the text setting, a containment score near 1 suggests that the text of *A* is somewhere contained in the text of *B*. We may estimate containment by

$$c_e(A,B) = |L(A) \cap L(B)| / L(A)$$

Again this is an unbiased estimator.

4. Winnowing algorithm

In this section we describe and analyze the Winnowing algorithm, which is taken from [15], for selecting fingerprints from hashes of k-grams. We give an upper bound on the performance of Winnowing, expressed as a trade-off between the number of fingerprints that must be selected and the shortest match that we are guaranteed to detect.

Given a set of documents, we want to find substring matches between them that satisfy two properties:

1. If there is a substring match at least as long as the *guarantee threshold*, *t*, then this match is detected, and
2. We do not detect any matches shorter than the *noise threshold*, *k.*

The constants t and $k \leq t$ are chosen by the user. We avoid matching strings below the noise threshold by considering only hashes of *k*-grams. The larger k is, the more confident we can be that matches between documents are not coincidental. On the other hand, larger values of k also limit the sensitivity to reordering of document contents, as we cannot detect the relocation of any substring of length less than k. Thus, it is important to choose k to be the minimum value that eliminates coincidental matches. We will continue with the previous example text, with the parameter k of 5.

A do run run run, a do run run
(a) Some text.

adorunrunrunadorunrun
(b) The text with irrelevant features removed.

adoru dorun orunr runru unrun nrunr runru unrun nruna runad unado nador adoru dorun orunr runru unrun
(c) The sequence of 5-grams derived from the text.

77 72 42 17 98 50 17 98 8 88 67 39 77 72 42 17 98
(d) A hypothetical sequence of hashes of the 5-grams.

Define a window of size w to be w consecutive hashes of k-grams in a document (w is a parameter set by the user). By selecting at least one fingerprint from every window the algorithm limits the maximum gap between fingerprints. In fact, the algorithm is guaranteed to detect at least one k-gram in any shared substring of length at least $w + k - 1$.

Given a sequence of hashes $h_1 \ldots h_n$, if $n > t - k$, then at least one of the h_i must be chosen to guarantee detection of all matches of length at least t. This suggests the following simple approach. Let the *window size* be $w = t - k + 1$. Consider the sequence of hashes $h_1 h_2 \ldots h_n$ that represents a document. Each position $1 \leq i \leq n - w + 1$ in this sequence defines a *window* of hashes $h_i \ldots h_{i+w-1}$.

(77, 72, 42, 17) (72, 42, 17, 98)
(42, 17, 98, 50) (17, 98, 50, 17)
(98, 50, 17, 98) (50, 17, 98, 8)
(17, 98, 8, 88) (98, 8, 88, 67)
(8, 88, 67, 39) (88, 67, 39, 77)
(67, 39, 77, 72) (39, 77, 72, 42)
(77, 72, 42, 17) (72, 42, 17, 98)
(e) Windows of hashes of length 4.

To maintain the guarantee it is necessary to select one hash value from every window to be a fingerprint of the document. In each window select the minimum hash value. If there is more than one hash with the minimum value, select the rightmost occurrence. Now save all selected hashes as the fingerprints of the document.

*(77, 72, 42, **17**) (72, 42, 17, 98)*
*(42, 17, 98, 50) (17, 98, 50, **17**)*
*(98, 50, 17, 98) (50, 17, 98, **8**)*
(17, 98, 8, 88) (98, 8, 88, 67)
*(8, 88, 67, **39**) (88, 67, 39, 77)*
(67, 39, 77, 72) (39, 77, 72, 42)
*(77, 72, 42, **17**) (72, 42, 17, 98)*
(f) Minimum hash values are selected from each window.

In our example, each hash that is selected is shown in boldface (but only once, in the window that first selects that hash) in *(f)*. The intuition behind choosing the minimum hash is that the minimum hash in one window is very likely

to remain the minimum hash in adjacent windows, since the odds are that the minimum of w random numbers is smaller than one additional random number. Thus, many overlapping windows select the same hash, and the number of fingerprints selected is far smaller than the number of windows while still maintaining the guarantee. *(g)* shows the set of fingerprints selected by Winnowing.

17 17 8 39 17
(g) Fingerprints selected by Winnowing.

In many applications it is useful to record not only the fingerprints of a document, but also the position of the fingerprints in the document. For example, we need positional information to show the matching substrings in a user interface. An efficient implementation of Winnowing also needs to retain the position of the most recently selected fingerprint. *(h)* shows the set of [*fingerprint, position*] pairs for this example (the first position is numbered 0).

[17,3] [17,6] [8,8] [39,11] [17,15]
(h) Fingerprints paired with 0-base positional information.

To avoid the notational complexity of indexing all hashes with their position in the global sequence of hashes of k-grams of a document, we suppress most explicit references to the position of k-grams in documents in our presentation.

4.1 Queries

This section, also taken from [15], is about how to choose hashes well and how hashes can be used once selected. In a typical application, one first builds a database of fingerprints and then later queries the fingerprints of individual documents against this database. Winnowing gives us some flexibility to treat the two fingerprinting times (database-build time and query time) differently.

Consider a database of fingerprints (obtained from k-grams) generated by Winnowing documents with window size w. Now, query documents can be fingerprinted using a different window size. Let F_w be the set of fingerprints chosen for a document by Winnowing with window size w. The advantage of Winnowing query documents with a window size $w' \geq w$ is that $F_{w'} \subseteq F_w$, which means fewer memory or disk accesses to look up fingerprints. This may be useful if, for example, the system is heavily loaded and we wish to reduce the work per query, or if we are just interested in obtaining a faster but coarser estimate of the matching in a document.

We can extend this idea one step further. Fingerprint a query document with the same window w used to generate the database, and then sort all of the selected fingerprints in ascending order. Next, look up some number of the fingerprints in the database, starting with the smallest. If we stop after a few, fixed number of hashes, we have realized Broder's approach [1] for testing document resemblance. If we use all of the hashes as fingerprints, we realize the standard notion of testing for document containment.

There is also a spectrum where we stop anywhere in between these two extremes. Broder's paper [1] on resemblance and containment gives distinct algorithms to compute these two properties; Winnowing naturally realizes both.

4.2 Chaffing and Winnowing

Another slightly changed version of the Winnowing algorithm is Chaffing and Winnowing [14]. However, this algorithm is used for providing confidentiality or information hiding not for copy detecting. Therefore it deals with terminologies like sender and receiver, encryption and decryption, key, authentication, MAC and packets etc. The rest of the description in this section is from [14].

The sender breaks the message into packets, and authenticates each packet using a secret authentication key. That is, the sender appends to each packet a "message authentication code" or MAC computed as a function of the packet contents and the secret authentication key, using some standard MAC algorithm [10].

The packet is still "in the clear'"; no encryption has been performed. We note that software that merely authenticates messages by adding MACs is automatically approved for export, as it is deemed not to encrypt.

There is a secret key shared by the sender and the receiver to authenticate the origin and contents of each packet. The legitimate receiver, knowing the secret authentication key, can determine that a packet is authentic by recomputing the MAC and comparing it to the received MAC. If the comparison fails, the packet and its MAC are automatically discarded. The sender and the receiver can initially create and agree upon the secret authentication key with any standard technique, such as authenticated Diffie-Hellman.

We note that it is typical for each packet to contain a serial number as well. For example, when a long file is transmitted it is broken up into smaller packets, and each packet carries a unique serial number. The serial numbers help the receiver to remove duplicate packets, identify missing packets, and to correctly order the received packets when reassembling the file. The MAC for a packet is computed as a function of the serial number of the packet as well as of the packet contents and the secret authentication key.

As an example, we might have a sequence of the form:

(1,Hi Bob,465231)
(2,Meet me at,782290)

(3,7PM,344287)
(4,Love-Alice,312265)

of triples of *sequence number, message,* and *MAC*.

The second process involved in sending a message is *adding chaff*: adding fake packets with bogus MACs. The chaff packets have the correct overall format, have reasonable serial numbers and reasonable message contents, but have MACs that are not valid. The chaff packets may be randomly intermingled with the good (wheat) packets to form the transmitted packet sequence. Extending the preceding example, chaff packets might make the received sequence look like:

(1,Hi Larry,532105)
(1,Hi Bob,465231)
(2,Meet me at,782290)
(2,I'll call you at,793122)
(3,6PM,891231)
(3,7PM,344287)
(4,Yours-Susan,553419)
(4,Love-Alice,312265)

In this case, for each serial number, one packet is good (wheat) and one is bad (chaff). Instead of randomly intermingling the chaff with the wheat, the packets can also be output in sorted order, sorting first by serial number, and then by message contents.

To obtain the correct message, the receiver merely discards all of the chaff packets, and retains the wheat packets. But this is what the receiver does anyway. In a typical packet-based communication system the receiver will automatically discard all packets with bad MACs. So the "winnowing" process is a normal part of such a system.

5. Conclusion

In this report we have discussed about fingerprinting, a copyright protection technique and its example methods with their advantages and disadvantages. We also took document fingerprinting as a case and have shown that though document plagiarism is the most difficult to detect, there are certain bounds when using hashes of k-grams for selecting document fingerprints. Finally, we have presented the Winnowing, a document fingerprinting algorithm that is both efficient and guarantees that matches of a certain length are detected in documents.

6. References

[1] Broder A. Z., *On the resemblance and containment of documents*, Proceedings of the Compression and Complexity of Sequences 1997, 1997.

[2] Chor B., Fiat A., Naor M. and Pinkas B., *Tracing Traitors,* Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology, 1994.

[3] Collberg Christian, *SANDMARK*, http://sandmark.cs.arizona.edu.

[4] Cox I. J., Killian J., Leighton F. T., Shamoon T., *Secure Spread Spectrum Watermarking for Multimedia*, IEEE Transactions on Image Processing, 1997.

[5] Dittmann Jana, Behr Alexander, Stabenau Mark, Schmitt Peter, Schwenk Jörg, Ueberberg Johannes, *Combining digital Watermarks and collusion secure Fingerprints for digital Images*, German National Research Center for Information Technology, Darmstadt, Germany, 1999.

[6] Heintze Nevin, *Scalable document fingerprinting,* In 1996 USENIX Workshop on Electronic Commerce, 1996.

[7] Karp Richard M. and Rabin Michael O., *Pattern-matching algorithms*. IBM Journal of Research and Development, 1987.

[8] Katzenbeisser S., and Petitcolas F.A.P., *Information hiding techniques for steganography and digital watermarking*, Artech House, 2000.

[9] Kilian Joe, *Resistance of Digital Watermarks to Collusive Attacks*, 1998.

[10] Krawczyk, H., Bellare M., and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, 1997.

[11] Manber Udi, *Finding similar files in a large file system*, In Proceedings of the USENIX Winter 1994 Technical Conference, 1994.

[12] *Moss – a system for detecting software plagiarism,* http://www.cs.berkeley.edu/ aiken/moss.html.

[13] Pfitzmann B. and Schunter M., *Asymmetric fingerprinting*, Springer-Verlag Berlin Heidelberg, 1996.

[14] Rivest Ronald L., *Chaffing and Winnowing: Confidentiality without Encryption*, MIT Lab for Computer Science, 1998.

[15] Schleimer Saul, *Winnowing: Local Algorithms for Document Fingerprinting*, University of Illinois, 2003.

[16] Sumit Ganguly and Gaurav Veda, *A new randomized algorithm for Document Fingerprinting*, Indian Institute of Technology, 2000.

[17] Wang Yiwei, Doherty John F., Dyck Robert E. Van, *A Watermarking Algorithm for Fingerprinting Intelligence Images*, Conference on Information Sciences and Systems, The Johns Hopkins University, 2001.

[18] Weir George R S, Gordon Margaret Anne and McGregor Grant, *Work in Progress – Technology in plagiarism detection and management*, 4th ASEE/IEEE Frontiers in Education Conference, 2004.