CERIAS Tech Report 2001-54

ON WATERMARKING SEMI-STRUCTURES

by Radu Sion, Mikhail Atallah, Sunil Prabhakar

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907

On Watermarking Semi-Structures *

Radu Sion, Mikhail Atallah, Sunil Prabhakar
CERIAS & Computer Sciences, Purdue University
West Lafayette, IN, 47907, USA
http://www.cs.purdue.edu/homes/sion
[sion, mja, sunil]@cs.purdue.edu

Abstract

Watermarking, in the traditional sense [10] [11] [12] [14] [15] [16] [18] [23] [24] [29] [30] [31] is the technique of embedding un-detectable (un-perceivable) hidden information into multimedia objects (i.e. images, audio, video, text) mainly to protect the data from unauthorized duplication and distribution by enabling provable ownership over the content. Whereas considerable work has been invested in this topic, little has been done (with the notable exception of attempts in software watermarking [7] [8] [9] [22] and recent progress in the area of natural language processing [1]) to enable the same concept in the area of semi-structured non-media data such as XML, databases and non-multimedia repositories.

We believe that there is much to be gained from the ability to embed non-destructive hidden information in this kind of content, in particular considering current mainstream migration of business interactions towards distributed computing technologies using markup languages such as XML and underlying database storage.

Watermarking in the area of semi-structured data presents a whole new set of challenges and associated trade-offs. One characterizing main difference can be expressed simply as "lack of bandwidth", deriving from the inherent lack of a major noise component in that domain.

We present some of the issues encountered in the course of our ongoing work in watermarking XML and numeric database content. We define a preliminary model-level analysis of the new domain and corresponding transforms. We design a method for watermarking semistructures based on a novel canonical labeling algorithm that self-adjusts to the specifics of the content. Labeling is tolerant to a significant number of graph attacks ("surgeries") and relies on a complex "training" phase at watermarking time in which it reaches a optimal stability point with respect to the expected attacks. Watermark detection works without requiring the original un-marked object. We analyse how to perform efficient and useful generic node content summarisation, hashing. We treat the issue of graph partitioning in the framework of hierarchical watermarking and show how hierarchical watermarking effectively amplifies the power of weak marking algorithms leading to an ultimately more powerful and robust watermark. We perform experiments enforcing some of the introduced algorithms (e.g. labeling) under different attack conditions and present some of the conclusions. Future envisioned medium and long term research issues are outlined.

1 Introduction

Not enough attention has been focused on enabling the concept of Watermarking in the area of semi-structured data. Theoretical approaches [6] [20] [21] [26] explore the broader area of steganography and information hiding in a generic manner.

One fundamental difference between watermarking and steganography resides exactly in the main applicability and 0 descriptions of the two domains. Steganography's main concern lies in Alice and Bob being able to exchange messages [4] [13] [25] in a manner as resilient and hidden

*Portions of this work are supported by Grant EIA-9903545 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

as possible, through a medium controlled by malicious Wendy. On the other hand, digital watermarking is usually deployed by Alice to prove ownership over a piece of data, to Jared the Judge, usually in the case when Tim Mallory, the Thief benefits from using/selling that very same piece of data or maliciously modified versions of it.

Proof of ownership [1] usually is achievable by demonstrating that the particular piece of data exhibits a certain rare property (read "hidden message" or "watermark"), usually known only to Alice (with the aid of a "secret"), the property being so rare that if one considers any other random piece of data, even similar to the one in question, this property is very improbable to apply. It is to be stressed here that the main focus in watermarking is on 'detection' rather than 'extraction'. Extraction of a watermark (or bits of it) is usually a part of the detection process but just aids the process. If the watermark data in itself becomes more important than detecting the existence of it (aka. 'yes/no answer') then this is drifting towards covert communication and pure steganography. There is a threshold determining Jared's convince-ability related to the "very improbable" assessment. Nevertheless this defines a main difference from steganography: Jared doesn't care what the property is as long as Alice can prove it's she who embedded/induced it to the original (non-watermarked) data object.

Moreover, in many cases (semi-structures being one of them) we are faced with a very narrow bandwidth (watermarking capacity) characteristic, due to the very nature of the data in case.

Example: An excellent example is a small but important numeric database table, e.g. stock trend analysis information, where the allowable error tolerance is very low, often caused by associated high-level semantic constraints.

Lack of sufficient noise , a new transform domain as well as different value and usability models make the issue even more complex by posing new security threats related to even some simple attacks. For example, automatic translation of XML or the use of database views in re-generating parts of the data are simple but powerful security challenges the new algorithms have to face and resist.

Note: We define later on *usability* in theoretical terms, as a functional. Now we're referring to simply the fact that the very same data can exhibit different types and level of noise if being put to different uses. In the case of data mining, extreme care has to be taken on this account, because the watermark can reveal itself easily if a certain view (and it's associated statistics) has been overlooked at embedding time.

Semi-structures are characterized usually by value lying both in the structure ("graph") *and* in the non-structured content ("nodes").

Examples include XML documents, complex Web Content, Software, Natural Language, relational DBMS data, VRML and similar environmental representations, structured financial and B2B interaction data, workflow and planning descriptions etc.

In this case the available noise-band width is very low, making it necessary to discover new data properties that allow for a resilient, un-noticeable entropy increase, maintaining all other properties of interest in place, within usable boundaries.

In most media watermarking techniques, the noise-band is usually the main recipient for any watermarking techniques. The amount of noise in well defined structured data tends asymptotically to zero.

Discovering appropriate bandwidth channels (watermarking capacity) is one of the challenges. Using them in a most efficient and resilient way is another. Tradeoffs need to be identified and balanced properly so as to ensure a satisfying end result, according to the particular purpose of each application. We immediately envision the need for balancing a required level of mark resilience with the ability to maintain guaranteed error bounds, structural equivalence and higher level semantics. Apparently, bandwidth is available from capacities associated to properties of both the graph and the nodes. Further insight shows that many limitations apply and the task is not trivial.

In this paper we explore how Alice tries to make sure she can safely distribute newly created semi-structures, such as Web pages and XML meta-information, to Tim Mallory and others. Of course she also wants to be able to prove later on to

[1] Fingerprinting can be easily considered in this same category although it is at the boundary between watermarking and steganography, because we're usually also interested in the fingerprint identification information.

Jared that the data in Tim's possession (possibly maliciously modified) was her creation.

The paper is structured as follows. Section 2 presents a brief survey of known watermarking applicability areas and techniques. It can be skipped by the knowledgeable watermarking researcher. Section 3 determines major challenges and starts defining more theoretical concepts in the new domain, such as *watermark*, *usability*, *usability domains*, *watermarking algorithm*, *power*. Section 4 is dedicated to actually mapping our research domain onto the previously presented model. Here we discuss abstract constructs such as *structure*, *relation*, *equivalence*, *transforms*, *noise injection*. Different capacity sources are explored and trade-offs analyzed. A generic hierarchical watermarking algorithm is sketched. Section 5 presents some conclusions derived out of labeling experiments. Section 6 defines our current ongoing efforts as part of a broader frame of future envisioned research.

2 Watermarking techniques for various content. Brief survey.

In the following we feature a brief survey of watermarking techniques and related research for different types of data. An exhaustive presentation is out of the scope here.

Image Watermarking. Being one of the favorite topics in the area, image watermarking algorithms cover a wide range of representative techniques.

Visible watermarks are made parts of the watermarked image. They are used in cases when resilience and secrecy are not paramount required features of the marking scheme. The watermarks can be observed easily and serve a more or less symbolic scope. Also, in most cases, visible watermarks damage the initial image more than any other techniques, making it often unusable for any other purpose than sampling. One of the first approaches to image watermarking, *Least Significant Bits marking*, uses parts of the image itself to store the watermark randomly (using a secret key) by altering the LSB (least significant bits) of each image pixel. Several variants exist, some of which do not use any kind of secrecy, others that go as far as to actually perform additional tests on the suitability (e.g. luminosity variance;) of certain pixels, before picking them to be part of the modified set. Although still widely used (in many modified variants), LSB marking features many undesired characteristics, enabling the easy removal or alteration of the embedded mark. Another interesting marking scheme is based on knowledge about the compression scheme deployed in the image (e.g. GIF palette encoding). Drawbacks of this method include the inability to support conversion to a different format, case in which the watermark would be lost.

One of the more powerful marking scheme is *frequency domain coefficients (discrete cosine transform) altering* watermarking. The main idea behind it is to avoid any destructive attack by embedding the mark into important parts of the image (high level coefficients in the frequency domain DCT transform), selected randomly by means of a key. JPEG is the favorite target of this scheme because of the inherent process of JPEG compression that requires computing and storing of the DCT transform coefficients anyway. Many other image watermarking schemes have been proposed, most of which can be related to one of the above mentioned techniques. Research has been conducted as to how masking properties of the HVS can be used in watermarking images.

Audio Watermarking. Existing audio watermarking techniques feature a high degree of similarity with image watermarking. This derives from the main characteristics of the data itself with higher noise to signal (useful data) ratio that most of the alternate usual data. *Echo Hiding* techniques, relying on masking properties of the HAS (Human Auditory System) insert small echo samples into the actual data. Another interesting idea relies on using statistical properties of the audio data in modeling the mark. One approach [28] is based on large sets theory, encoding 1 bit in every 1.2 seconds time-slice by changing the probability distribution function of the data.

Video Watermarking. Many video watermarking algorithms present a great deal of similarity with audio and image techniques. Well performing algorithms however use different ways of encoding the watermarks into the available data. Encoding-dependent watermarking schemes

use specific properties of the particular video encoding.

Text/Language Watermarking. Many challenges are associated with Natural Language (NL) and structured text Watermarking. The main issues stem from the complexity of the content as well as from the very low available bandwidth. The necessity of actually preserving semantic and syntactic constraints as well as the need to provide an acceptable level of resilience leads to low available watermark encoding bandwidth. Nevertheless, several attempts were made to provide viable NL watermarking techniques. Most of them are a combination of or include some of the following methods.

The use of *synonyms* seems to be a starting point in many of the existing algorithms. Preserving semantic coherency implies the use of specialized dictionaries and/or semantic networks. By replacing certain words with semantic equivalent synonyms, a scheme of attaching information to their existence within the content, can be designed and used for watermarking.

Defining a set of key words and/or canonical structure as well as associated modification metrics is another method allowing for information encoding.

Statistical properties of text items can also be used/modified in order to encode certain information. In case of the above mentioned use of synonyms a scheme favoring only certain synonyms (i.e. that feature a special statistical distribution) can be easily designed. The information can be encoded in the variation of the statistical distribution between the words or selected key words.

One extremely challenging issue is the ability to *semantically watermark* a certain NL content. One example would be embedding semantics into the content (without modifying the existing semantics) in a hard to remove manner (e.g. the name and the order in which cities are visited, in a travel description).

Other schemes include the use of *syntactical characterization* methods in encoding information (e.g. encoding the information in the number of existing words per class - substantives, verbs)

Software Watermarking. Watermarking techniques for software aim mainly at being able to prove ownership over algorithms. This main case is concerned with the actual intrinsic value attached to the algorithmic flow in itself and less with the code and executable binaries. There are also ongoing efforts to watermark compiled code. In this approach, value is also seen in the binary file (i.e. resulting from the compilation of a certain source file) itself. The two approaches are leading to different marking techniques. Algorithm watermarking uses the actual structure of the algorithm as a channel for the watermark information, and presents thus a lower bandwidth characteristic.

Binary code watermarking (given the nature of the binary data), usually encodes information mainly in the machine code content, having a slightly higher available bandwidth. Techniques for binary code watermarking may make use of some of the following ways of encoding hidden information: the initialization and use order of registers, the order of push/pop stack operations, hidden values in certain unused portions of registers and/or memory addresses, hidden values in numeric representations (e.g. exploiting overflow representation of numeric representations). In the same class of binary code related information hiding, worth mentioning are the techniques for Obfuscation/Runtime Tamperproofing. In this case, the hidden information is the algorithm itself, protected from unallowed decompilation/disassembly.

Algorithm watermarking methods often make use of some of the following: changing the runtime structure of an application by applying certain equivalence transforms

Palsberg et.al. in [22] and Collberg et. al. in [7] represent numbers/watermarks as graphs, and then as runtime structures, in some cases derived from the algorithmic flow and current program state, making the watermark part of the actual algorithmic behavior. Those ideas, coupled with the facility of runtime introspection and changeability becomes a very powerful tool in enabling software watermarking.

3 The Model

Let $\mathbb{D}$ be the domain of all possible data objects to be considered for watermarking.

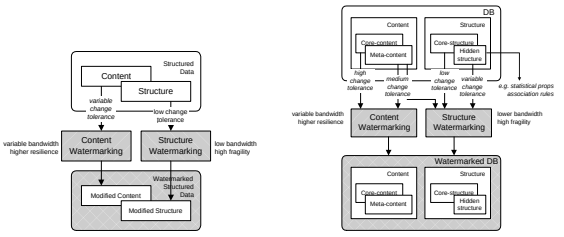

Figure 1: Watermarking semi-structured data.

Note: Most of the concepts here are introduced at intuitive level, for illustrative purposes only. More in depth definitions can be found in [27].

For example in case of digital media objects we can simply assume that $\mathbb{D}$ is the set of all variable sized bit strings over $\mathbb{B} = \{0, 1\}$.

Objects $d \in \mathbb{D}$ have associated value induced by the object creator. Watermarking tries to protect this association between the value carrying object and its creator.

Usability Domain: Complex objects can exhibit different value levels when put to different uses. We need a way to express the different associated values of objects, in different *usability domains*. Defined elsewhere [27], intuitively, a usability domain models different "uses" a certain object might be subjected to.

Usability: Defined elsewhere [27], intuitively, usability is a measure of how "useful" an object can be with respect to a given domain. The concept of usability enables the definition of a certain threshold below which the object is not "usable" anymore in the given domain. In other words, it "lost its value" to an unacceptable degree. The notion of usability is related to *distortion*. A highly distorted object (e.g. as result of watermark embedding or attacks) will likely suffer a drop in its distortion domain usability.

Usability Vicinity: Defined elsewhere [27], usability vicinity defines a set of objects that are not to far away (in terms of usability) from a given reference object. The *radius* of the vicinity is defined by the distance to the reference object of the "farthest" object within the vicinity.

Note that the usability vicinity of a certain object $d \in \mathbb{D}$ with respect to a considered set of usability domains $V \subset \mathbb{U}$ defines actually the set of possible watermarked versions of d with respect to V and Δu_{max}.

Watermark: Defined elsewhere [27], in plain words, a watermark can be defined as a special induced (through watermarking) property (w) of a certain watermarked object $d' \in \mathbb{D}$, so rare, that if we consider any other object $x \in \mathbb{D}$, with a "close-enough" usability level with the original object d, the probability that x exhibits the same property can be upper-bound.

Watermark Power: Defined elsewhere [27], the power of a certain watermark is directly related to it's convince-ability towards Jared the Judge. The weaker the watermark (higher the false hit probability upper bound) the less convincing it will be.

Algorithm: Defined elsewhere [27], a *watermarking algorithm* can be described as a functional $a : \mathbb{D} \times \mathbb{K} \rightarrow \mathbb{D} \times \mathbb{W}$.

Attack: Defined elsewhere [27], an attack simply tries to maintain the attacked watermarked object within the usability vicinity of the original non-watermarked one, while making it impossible to recover the watermark.

Main Challenge: Power and Usability. The main challenge of watermarking lies with keeping the required usability level of the object unchanged or close to its original value, while still featuring enough power. Thus, an appropriate algorithm will try to determine the main usability domains for a particular to-be-watermarked object and then preserve usability in those domains while maximizing the mark power.

4 The Structures

We consider structures that can be represented abstractly as simple connected graphs. Connectivity is assumed for the sake of simplicity only. An unconnected graph could be also handled by considering it's not-connected sub-graphs separately.

Most structures are represented also in reality as graphs. If not, a usability preserving mapping to graphs can be deployed. For now we are not concerned with this extension. For simplicity, directed graphs are not considered in this paper, although extrapolation is straightforward.

$\mathbb{D}$ becomes the set of all directed graphs with

data/value-carrying nodes. We consider graphs as being abstract objects characterized by a set of vertices (nodes) and directed edges.

Semi-structures. Nodes in semi-structures are value-carrying, and a watermarking algorithm can make use of their encoding capacity.

The nodes are considered as being non-structured for now and the available capacity can be used by deploying traditional (non-semistructure related) watermarking. In the following (*hierarchical watermarking*) we show how a hierarchical construction process can offer multiple granularity views on a given structure, in which case sometimes nodes become sub-structures themselves. Thus we make an important assumption about the nodes of the graph structure. The nodes allow full content read access and limited write access, regulated by the maximum allowable change in usability of the given graph and the node's impact in overall usability.

Let $content(n), n \in G$ where $G \in \mathbb{D}$, be the notation for the content of node $n \in G$.

The Labeling Issue. In the case of graphs one issue always arises, namely the ability to uniquely identify and reference nodes. For indistinguishable nodes, this is summarized in graph theory under the term *canonical labeling* [2] [3] [17] [19] and no solution has been provided with a high enough degree of generality. In our case, the value-carrying nodes are solving the labeling issue in quite a surprising manner.

Using content specific watermarking techniques as well as keyed content hashing can provide an resilient enough labeling scheme. The algorithm presented makes full use of content-specific watermarking (*content noise injection*, see bellow) and keyed hashing in node labeling.

Noise Injection. The concept of *noise injection* aims to basically increase the noise ratio of the to-be-watermarked data, while preserving usability levels. This is attainable by addition of new components to the data and/or modification of existing ones.

Given the bivalent nature of semi-structures, we distinguish among two types of *noise injection*. *Structural noise injection* works by removing insignificant (in terms of usability) structural elements (nodes or edges) or, more often, inserting fake new ones in the structure. This aims at increasing the available overall encoding bandwidth as well as shifting the usability level toward the vicinity limits in such a way as to ensure that any further un-knowledgeable modifications to the graph (e.g. attacks) will fail to produce an enough-usable structure.

Some of the requirements of structural noise injection include the following: Nodes need to be inserted in such a way as to affect existing labels in the labeling scheme as little as possible while preserving all usability levels in all considered domains. Content distribution but also semantics need to be considered when constructing the new nodes in such a way as to not reveal themselves by differing from the other nodes. In some cases, duplicating existing nodes in similar positions (*twins*), mainly for redundancy purposes, instead of creating new ones, provides a fair solution to this.

Content noise injection basically modifies existing nodes' content either by adding un-obtrusive information (if permitted by specific case) or by using content-related watermarking techniques to encode an usually small amount of information [2] (in the nodes' content), to be used later in the whole-structure watermarking scheme.

The design of correct structural and content noise injection techniques faces many challenges and no general method can be devised as it is very much linked to content type and distribution as well as to the considered usability domains. Preserving semantical coherency constitutes another hard issue to consider, making noise injection even more complex.

While, in the near future, it may not be possible to provide a general use noise injection technique, some related desiderata outline themselves, especially considering the framework of hierarchical watermarking (see bellow).

Semantic Classification. The possibility that certain graphs contain sub-partitions and even single nodes with associated higher-level semantics that make it hard to consistently noise inject (across semantic boundaries), makes it necessary to somehow differentiate between those parts of

[2] The use of the term *noise injection* in this later case is a bit forced but given that purpose-wise it serves the same final goal, we decided to use it also here.

the graphs and treat them separately, at least in terms of noise injection and content hashing.

This is attainable through a preliminary *semantic classification* and partitioning process, which has as result a set of different *semantic classes* in which we consider watermarking separately for each one.

It is hard to perform a *semantic classification* on a given semi-structure without knowledge of expected attacks and without properly identifying all corresponding usability domains for the object. This is subject to further research. In the following we assume semantic pruning and classification have been performed and we work on graphs that contain nodes in one single semantic class, allowing for consistent noise injection, as presented below.

Attacks. Given a certain value carrying semi-structure (value both in nodes and topology itself) several attack options present themselves, including: elimination of value-"insignificant" nodes (A1), elimination of inter node relations (A2), value preserving graph partitioning into independent usable partitions (A3), modification of node content, within usability vicinity (A4), addition of value insignifiant nodes aimed at destroying ulterior labeling attempts (A5). One has to keep in mind the ultimate goal of any attack, namely eliminating the watermark property, while preserving most of the attached value, within usability limits.

Note: Collusion attacks are not rellevant for now. Remember that collusion is most appealing when fingerprinting methods are deployed. Altough we envision extensions of current work for fingerprinting semi-structures, we are not going to discuss these here. We assume that a certain value-carrying semi-structure is watermarked with the same mark for its entire lifetime. This is not unreasonable to assume and extensions for treating the case of fingerprinting will be provided at some later point in research. For a discussion about collusion and on differences between fingerprinting and watermarking see [23] [24].

In order to prevent success for A5, we propose a preliminary step of **value pruning** in which all value-insignifiant nodes are marked as to-be-ignored in the ulterior watermarking steps.

A4 mandates the ability of the labeling scheme to depend as little as possible on node content or to provide for a mechanism of detecting altered-content nodes at extraction time. Another possibility of defending against A4 would be to actually noise-inject (using a private key, see bellow) the main considered nodes towards the usability vicinity limit, such that any further un-knowledgeable (without knowledge of the key used in the noise injection process) changes will fail to provide a usable result.

Attack A3 is one of the most powerful challenges. In order to survive it, meaning that the watermark has to be preserved (maybe in a weaker form) also in the resulting graph's partitions, the watermarking scheme has to consider some form of hierarchical embedding in such a way as to "touch" most of the potential partitions in the graph. Given a well known set of partitions as well as their attached value, it is not very hard to imagine a watermarking procedure that considers all of them in turn.

The issue becomes more complex if the usability domains of all possible graph partitions are unknown, making it difficult to envision the attacker's "cut". Fortunately, in many cases (see Scenarios) the number of available partitioning schemes that make sense and the associated usability domains are limited.

Example: A simple example would be a car consisting of an engine, transmission, wheels, body. An engine consist of a air-flow system, valve and cylinder train, ignition module, fuel supply. A pretty straight forward, intuitive, partitioning scheme would follow exactly the outlined components. It wouldn't make too much sense from most perspectives to separate the structure into partitions containing the ignition systems and wheels in one partition and the rest in the other partition. Intuitively any value preserving partitioning scheme would basically follow the graph link structure.

Cases A1 and A2 makes it necessary to devise a labeling scheme that tolerates node and edge elimination while preserving most of the other nodes' labels. This is a must because of the necessity to reference nodes at extraction time.

Even if there would exist a generic canonical graph labeling algorithm it would need to be heavily modified in order to provide for edge and node removal tolerance. We used the term "heavily" to outline the fact that canonical labeling has always been linked to proofs of graph isomorphism, whereas in this case the trend seems to be aimed exactly towards the opposite, namely

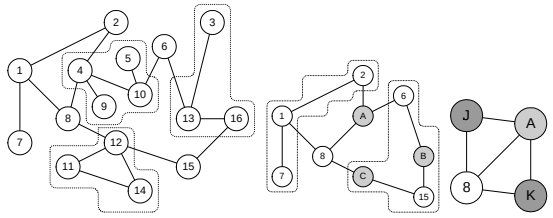

Figure 2: Hierarchical Watermarking. Partitioning Example on Connected Structure.

preserving node labels in the context of slight graph changes.

Hierarchical Watermarking. As described above, attack A3 mandates that the watermark encoding scheme used considers sub-partitions of the graph and ensures survivability of the watermark even in those resulting partitions from the attacked graph.

The idea behind *hierarchical watermarking* is to use a set of weaker marks in a hierarchical fashion so as to ensure survivability in subgraphs of the original object, even if providing less watermarking power. It will be harder to convince Jared regarding the sub-partitions but providing a combination of a set of weak watermarked graphs should increase the overall power level.

Labeling Solution: Tolerant Canonical Labeling. The labeling scheme is at the heart of watermarking semi-structures. The ability to identify and reference nodes within the to-be-watermarked structure is of paramount importance and the labeling scheme has to take into account the specifics of the case, in particular the requirement to be able to "recognize" all relevant nodes in an attacked version of the graph, based on labels issued on the original one.

Although canonical labeling was known for a long time to be a hard problem of graph theory, specific algorithms have been developed for some cases . In particular, reasonable solutions have been proposed for tree canonical labeling and apparently, many semi-structure watermarking applications (e.g. XML and HTML) would fit the assumption of tree structuring. An idea would be to even partition existing value-carrying semi-structures into a set of tree-shapes and remaining structural elements. Watermarking only those partitions might provide enough power and reduce the problem to tree shapes.

Unfortunately the requirement of being able to label nodes consistently before and after attacks, changes the entire perspective, rendering useless existing tree canonical labeling algorithms.

The dual nature of semi-structures enables a novel approach to labeling, the main idea being the use of a combination of structural and node content information.

On the one hand, content is combined in computing a node's label by using a special hash (i.e. a function of the content with specific properties, as defined bellow) of it. The assumption made is that content changes are small and that we are able to construct a hash function on the node content that will basically degrade gracefully with minor

On the other hand some node topology information is necessarily involved in the relative position of the node versus its neighbors and the entire graph. One simple solution that comes to mind is to use the neighbors' labels, which does capture the position of the current node in relationship to its neighbors, and through the entire labeling scheme, applied recursively, to the graph as a whole.

This labeling method is not trivial implementable, in extreme cases, for example, if the graph is loosely connected or entirely unconnected, neighboring information is not available. One might argue that in that case it wouldn't make sense anyway because a node's "locality" is undefined if there are no connecting edges, that is, no relations to other nodes. But this is another discussion altogether and we won't enter it here. We assume that in any case the graph is fully connected (e.g. in the directed case, each node has an incoming or outgoing edge).

Thus the algorithm can be summarized by the following iterative formula:

$$l(node) = \alpha * l(node) + \gamma * \sum_{nb \in neighbors(node)} l(nb).$$

Note: α determines the "weight" of the content in the labeling scheme. If essential content changes are unlikely in an attack, α is to be increased so as to provide labeling stability. On the other hand γ provides control over being able to more specifically localize the node with respect to the neighbors and also to the entire graph. If

structural changes are highly unlikely in the course of an attack an increased γ provides for stability. Trade-offs need to be made and each case will have its specifics.

The algorithm starts with the labels being the keyed content hash values $HASH(key, node)$ (key is our secret).

Step One. The first step performs a number of iterations i over the formula above (this number being kept as part of the watermark detection key and used later on in re-labeling the attacked graph), until the necessary labeling provisions are met. At this stage we are mainly concerned with a minimal number of identical labels.

Note: A number of iterations at least equal to the diameter of the graph are necessary in order to localize a given node with respect to the entire graph. But this is sometimes not desired nor required. The ability to set the number of performed iterations and make it part of the recovery key is another point of control over the labeling scheme.

Step Two. In order to provide resilience to a certain number of graph modifications ("surgery"), the next step is to artificially degrade the graph and re-perform step one again.

Intuitively (also confirmed by experimental results), removing and/or adding nodes and relations to the graph will result in changes in the initial labeling performed on an un-modified graph. A certain control over those changes is enabled by being able to specify α and γ values. Experiments show that for certain α and γ value bounds, labeling becomes controllable. In each step, all the resulting labels and the corresponding iteration number, α and γ values are kept. This enables later on, computing of the optimal values. This indeed leads to non-trivial storage requirements and a $O(n^2)$ complexity. But this only happens once in the lifetime of the watermarked object, that is, at graph labeling time.

The result of step two, for each node, is a range of values for the corresponding label, depending also on the three main control factors (step-one iteration number, α, γ). The actual label of the node will be defined by the lower and upper bounds of the resulting labeling range. This basically ensures that, when labeling the attacked/modified version of the graph (i.e. by performing step one of this same algorithm later on, in court), the resulting labels will fall within the corresponding node's label interval with a high likelihood.

Now, for a given set of surgeries, performing the labeling algorithm in the space of (α, γ, i) results in a "bounded space of labeling points". The immediate next challenge is to identify an optimum in this "space", given a certain ability to compare two particular "points". Remember that a "point" corresponds to a labeled graph as a set of interval-labels for the graph's nodes, given the particular (α, γ, i) coordinates. Finding an optimum implies the ability to compare two "points". This is not trivial and needs further research. Our initial comparison formula for two different graph interval-label sets aims at capturing optimality in terms of both minimal number of label overlaps within each set as well as minimal potential for future overlap. If we write the two considered "points" as the actual interval-label sets $A = \{(a_{11}, a_{12}), ..., (a_{n1}, a_{n2})\}$ and $B = \{(b_{11}, b_{12}), ..., (b_{n1}, b_{n2})\}$ (i.e. (a_{i1}, a_{i2}) is the label-interval corresponding to node i in the graph labeling A) then the comparison formula reads

$$compare_1(A, B) = overlaps(B) * avg\_overlap\_size(B) - overlaps(A) * avg\_overlap\_size(A).$$

$$compare_2(A, B) = closest\_inter\_label\_size(A) - closest\_inter\_label\_size(B).$$

$$compare(A, B) = compare_1(A, B) + compare_2(A, B).$$

where $overlaps(X)$ is the number of overlapping interval-labels in labeling X, $avg\_overlap\_size(X)$ the average interval size of the overlapping portions and $closest\_inter\_label\_size(X)$ the size of the interval between the closest two interval-labels in X.

Intuitively, $compare_1()$ models and compares the current optimality of both labelings and $compare_2()$ captures the potential for future overlap (i.e. because having very "close" interval-labels hints to possible issues in labeling the graph in an attacked version of it).

Much research needs to be done on how this formula impacts the final resillience properties of the mark and what improvements can be brought to it.

Experiments show that, given reasonable experimental assumptions about the to-be-labeled

structure, the above labeling method produces fairly resilient labels. For large structures, the amount of work necessary is certainly high but every step can be automated and actual execution times are reasonable, even on lower end computers (PC, 500MHz, 128MB RAM, 100 nodes, tolerance to maximum 10 nodes removed, varied α, γ from 0 to .9 in .1 increments).

If overlapping intervals occur, some of the following actions are possible and should be performed:

1. If nodes are in identical positions (e.g. J and K in figure 2) and with similar content then this is normal. The nodes are marked as such and treated identical throughout the watermarking process.

2. If nodes differ in content but positions are similar, or content is close but positions are different, then variations in α, γ and the content hash key should be performed in such a way as to differentiate labels

3. If nodes differ in both content and position, changing also the iteration number in step one is required.

4. If everything has been done and label intervals are still overlapping one method simply "melts" the labels together and treats the nodes as in case 1 (i.e. identical).

In summary, the labeling process (i) collects all relevant labeling data over a number of iterations in which all of (α, γ, numbers of step-one iterations (i), content hash key and number of performed surgeries) are varied, and then (ii) decides upon a certain point in this space (defined by α, γ, i, content hash key and number of performed surgeries) which minimizes the number of overlapping label intervals and the potential for future overlaps.

Note: The algorithm presented below is simplified for illustration purposes in that it doesn't consider different values for $hash_{key}$ nor different iteration bounds (it does a fixed number of $diameter(G)$ iterations).

Graph Labeling:

LABEL(graph G):

10. for each node n let $label(n)$ = HASH($hash_{key}$,n)
20. for $\alpha = 0.1$ to 0.9
30. for $\gamma = 0.1$ to 0.9
40. forall artificial graph "surgeries"
50. perform surgery (remove node/relation(s))
60. for $iteration = 1$ to $diameter(G)$
70. foreach node n
80. $label(n) = \alpha * label(n) + \gamma * \sum_{neighbors(n)} label(nb)$
90. foreach node n store label(n)
100. foreach node n store $[min(label(n)), max(label(n))]$
110. choose (α,γ) minimizing the number of overlapping label intervals

By adapting to the given structure (i.e. through adjusting of α, γ, etc) , the labeling algorithm allows for a higher degree of control over the required trade-offs between label resilience and tolerated graph changes.

Note: Special consideration needs to be offered to the case of an attack modifying all existing nodes' content in a similar fashion. Alteration to the labeling scheme can be prevented in this case by introducing an additional final step of globally normalizing the labels (label intervals).

Content Hash Functionals. Finding an appropriate (set of) content hash function(s) that satisfy the requirements above is not trivial and strongly tied to the type of semi-structure node content and its associated transforms and envisioned attacks. The main two requirements of the content hash functions considered are the ability to be at the same time quite content specific while also degrading gracefully with minor changes in the content. While, domain specific solutions are probably more stable and better suited for each application, we experimented also with a set of generic hashes.

One of the most simple and trivial hash is defined by the number of bits in the content. Whereas this is obviously only partial content specific it degrades gracefully with minor changes in the content.

Counting the number of 1's in the content is also certainly specific but, given the assumption of randomness (i.e. unknown domain) it is not helping much. Variation on the same theme include the idea of using a certain XOR/hash/encrypt-ed version of the content and then performing parity and bit-set counts etc.

Another characteristic of the content that can be used in defining a hash as above, is the "compressibility" of the given content. That is, given a certain compression algorithm (e.g. Huffman), what is the maximum compression ratio we can get after a pre-determined number of rounds.

Interestingly enough, in trying to capture

something specific, associated to a certain given content, but also resilient enough so that minor changes in the content will not change it, we encountered the proven idea of mapping the content data into a new domain and trying to find some properties of the mapping result that satisfy the original requirements.

One simple mapping brings a one-dimensional data into a multiple dimensional space. For example it is possible to map the data to a 2 dimensional function defined by the following: starting in the origin of the coordinate system, if the next encountered bit is 1 advance 1 position on the oX scale and go "up", otherwise advance and go "down". The overall function shape can be integrated and the result is empirically proven to be quite content specific.

Another mapping to a two dimensional space can be defined by simply considering each pair of content bytes as a (x, y) "point" coordinate. After plotting all the "points", it is proven that a fairly resilient property of the numbers involved in defining the plot is determined by repeatedly "peeling-off" the convex hull until no more points remain on the plot [5]. The number of times we were able to perform the peel off as well as the series defined by the number of points peeled off in each round is proven to be very content specific and intuitively quite change tolerant.

Of much success in the image watermarking community, are transforms like the DCT (deployed mainly in JPEG watermarking) that map content into the frequency domain. The important transform coefficients in the new domain are then used for storing watermarks by various altering methods.

Whereas we could certainly use the same transform in the case of a known JPEG image content, by assuming generality this is certainly not possible in the given form.

But still the idea is very relevant to the case. The fact that minor changes in the DCT coefficients (in the transform domain) lead to minor, mostly un-noticeable changes in the result (i.e. back in the image domain) as well as the fact that DCT coefficients are quite content specific, lead to the idea that maybe using the inverse procedure will yield the desired results.

That is, we estimate that minor content changes will have little effect on corresponding transform coefficients. Thus, given a certain one-dimensional content bit-string, the corresponding hash value will be composed of a weighted combination of the significant transform coefficients. Finding a transform suitable (e.g. similar to DCT) is still in the works. For illustrative purposes only, content hashing is outlined below.

Generic Node Content Hashing:

HASH(key k, node N):

10. compute $DCT(content(N))$

20. use k in computing a weighted combination of the most significant coefficients

30. return computed combination, globally normalized over all nodes

Whereas using a transform in computing node content hashes can be used for various content, it does not make use of any particularities of specific types of content. For example if the node content is an JPEG image (e.g. relational multimedia database), a generic transform applied to an one-dimensional data view might be sub-optimal in that it wouldn't capture image features which, if captured, would certainly increase the level of specificity and graceful degradation with minor changes. In that case, using feature extraction algorithms (e.g. property histograms) and/or DCT transforms will certainly yield better results.

In the case of natural language (NL) content, capturing much of the specifics can be done by translating syntax trees and semantic relationships into certain characteristic values (e.g. by using Planted Plane Cubic Tree [22]).

Each type of content will probably require a special round of research and insight before a good hash can be found. This is the case especially with XML where a high number of content types are involved.

Partitioning in particular has to tolerate minor changes to the original graph (structural and content). That is to say that given the original graph $G = \{nodes, edges\}$ and an attacked/modified version of it, $G' = \{nodes', edges'\}$, the partitioning scheme has to produce an equal number of partitions (p) of G and G' in such a way that if $\{G_1, ..., G_i, ...G_p\}$, $G_i = \{nodes_i, edges_i\}$ ($nodes = \cup_i nodes_i$ and $edges = \cup_i edges_i$) is the partition for G, and $\{G'_1, ..., G'_i, ...G'_p\}$, $G'_i = \{nodes'_i, edges'_i\}$ ($nodes' = \cup_i nodes'_i$ and $edges' =$

$\cup_i edges'_i$) the partition for G', then for most (all significant) nodes n that were in G and "remained" in G' after the attack/modification, if n was included in partition sub-graph G_i (in the initial partitioning of graph G) then n will be included in sub-graph G'_i (resulted from partitioning of attacked graph G').

We propose the following preliminary algorithm, summarized as follows: Find the p most connected nodes and "group" partitions around them in such a way as to create some closures that balance the sizes of the partitions, the final goal being to maximize the number of hierarchical watermarking recursion levels.

Graph Partitioning:

PARTITION(graph G, key k, number p):

10. if G contains less than N_{min} nodes return null

20. let $\{G_1 ... G_p\}$ as set of subgraphs, empty for now

30. foreach $node \in nodes$ compute the number of neighbors: $sizeof(neighbors(node))$

40. sort nodes in the descending order of $simple\_encryption\_hash(k, sizeof(neighbors(node)))$

50. remove first node from $nodes$, insert into G_1, second node into G_2, ..., p_{th} node into G_p

55. $i = 0$

60. while ($\exists node \in nodes$) do

70. $i = (i + 1) mod p$

80. consider "oldest" node in G_i that still has one of it's neighbors in $nodes$ and add that neighbor to G_i. if none found continue.

90. return $\{G_1 ... G_p\}$

Note: A more suited partitioning algorithm will require to consider some of the relevant points presented in attack A3 above, namely partitioning into value-preserving partitions that an attacker might later on make use of independently. Whereas this is certainly required in order to survive attack A3, it is also very case specific. Additional domain specific research should provide for this.

The Algorithm: Hierarchical Construction. Let G be the original semi-structure, M the watermark to be embedded, k_1 the key(s) used in partitioning the graph (use different key at each new "level"), k_2 the key(s) used in watermarking the graph (use different key at each new "level"), k_3 the key(s) used in selecting a different part of the watermark to be embedded in different partitions.

The final watermarking secret is a combination of all the k_1's, k_2's, k_3's. Note that as the algorithm moves on, the keys can also be modified by the algorithm itself as to better adapt to existing structure and content. Thus, the keys should be considered as passed by reference.

Let N_{min} be the minimum number of nodes for which we are still able to provide noise injection techniques. As the number of nodes decreases it becomes harder and harder to inject noise within semantic, data distribution and usability constrains. Given each case and associated usability domains we envision the existence of a minimal nodes number structure that still allows for noise injection.

Hierarchical construction:

hierarchical_wm(graph G, watermark M, key k_1, key k_2, key k_3)

00. if G contains less than N_{min} nodes return

10. PARTITION(G, k_1, p): partition G into G_1, G_2, G_3 ... G_p (p maximal with $p > N_{min}$)

20. consider $G_1 ... G_p$ as nodes with limited capacity and construct G' (meta)graph with them as nodes, ignoring their details (internal nodes of G_1 ... G_p)

30. for each $G_i \in \{G_1 ... G_p\}$: call hierarchical_wm(G_i, M, k_1', k_2', k_3')

40. WATERMARK(G', k_2, M, k_3) [3]

50. return

Meta-Nodes. As the hierarchical watermarking process proceeds, partitioning and considering resulting partition sub-graph as nodes in a new meta-graph, considered for watermarking, the issue arises on how to treat those *meta-nodes* as well as the regular nodes in a unified manner. Of special interest in this case is the significance of the notion of "node content" and its corresponding content hash, used in the labeling process.

When labeling a graph that contains meta-nodes, the following hold:

1. Meta-nodes are marked as such and are never content noise injected.

2. The content hash of a meta-node is a recursive computed weighted sum of the content hashes of the composing internal nodes. This relies on the fact that a weighted sum will preserve the required overall properties in the resulting value.

The Algorithm: Hierarchical Detection.

[3] We only watermark on **return** (stept 40) from the recursion in order to maintain retrieval correctness.

Let l be an importance/weight factor (adding towards the overall watermark) of detecting a watermark in a structure at a certain "level". The idea is to basically allow for detection failures in the attacked structure. This is why as the algorithm goes "deeper" (more specific sub-structures/partitions) the weight of detecting a part of the watermark there decreases according to a certain proportion value, currently set to $v = \frac{1}{2}$. The initial call (on the entire structure) will use a value of l=1.

That is, at each level, detecting a watermark in the current level is as important ($*v$) as detecting all the embedded/expected watermarks in the current sub-levels. Experimenting with different values of v for specific domains and associated attacks should yield a good trade-off between the importance of details versus the big picture.

Hierarchical detection:

hierarchical_detection(attacked_graph G, watermark M, key k_1, key k_2, key k_3, weight l)

00. if G contains less than N_{min} nodes return 0
10. PARTITION(G, k_1, p): partition G as above [4]
20. consider $G_1 ... G_p$ as above, construct G'
30. for each $G_i \in \{G_1 ... G_p\}$:
$ret\_val = ret\_val$ + hierarchical_detection(G_i, M, k_1', k_2', k_3', $\frac{l*(1-v)}{p}$)
40. $ret\_val = ret\_val + (l * v)*$ DETECT(G', k_2, M, k_3)
50. return ret_val

The Algorithm: Embeding and Detecting at Graph Level.

Watermark embedding:

WATERMARK(graph G, key k, mark M, key $mark\_key$)

00. if G contains less than N_{min} nodes return
10. structural noise inject G in a maximal number of injection points (maximal = until the usability limit of the structure is reached)
20. select set of nodes $N_1 ... N_w$
30. test $N_1 ... N_w$ for content noise injection suitability. redo step 30 (changing k if necessary) until enough suitable nodes found.
40. if step 40 fails to produce a sufficient number of nodes change $mark_key$ to reflect the (smaller size) watermark portion that we *managed* to embed in this graph
50. content noise inject nodes $N_1 ... N_w$ with selected (by $mark\_key$) parts of mark M (here we have to ensure a corresponding degree of redundancy, embed several times)
60. LABEL(G)
70. store/keep labels and content hash of injection points
80. modify key k to point to $N_1 ... N_w$
90. return

Watermark confidence detection:

DETECT(graph G, key k, mark M, key $mark\_key$)

00. if G contains less than N_{min} nodes return 0
10. LABEL(G)
20. confirm injection points (using remembered labels and content hashes from step 30 above)
30. if not "enough" injection points match, return 0
40. select content noise injection points using previous remembered key k
50. consider selected (by $mark\_key$) parts of mark M and try to confirm each bit of it (by recovering content noise injection watermark and using redundant copies)
60. return level of confidence (percentage of mark detected)

By encoding even 1 bit per each node at each level of the hierarchical watermarking procedure, it can lead to a watermark of a high overall power even though each noise injection procedure in itself is very weak.

The Algorithm: Power. If value is attached to the considered semi-structure, an attack is likely to maintain it. Any reasonable attack will have to detect noise injection points and try to alter remove/them, while preserving overall usability bounds.

Assuming the most simple case of content noise injection, a trivial analysis determines the probability of detecting a number of j injection points ($j < n$) as being:

P(detect j injection points, given G with n nodes) $= (\frac{1}{n}) * (\frac{1}{n-1}) * ... * (\frac{1}{n-j}) = \frac{(j-1)!}{n!} < (\frac{1}{n-j})^j$

If the number of actual injection points is $i < n$ and we assume that the labeling scheme is tolerant (roughly) to altering of a maximum of l injection points (before degrading unacceptably) we have:

For $n > i > j > l$: Attack destroyed labeling scheme. Hopefully, depending on the application domain, also Δu_{max} was exceeded and result rendered unusable. The probability of this happening is less than $(\frac{1}{n-l})^l$. Considering $n = 26$ and $l = 6$ this becomes approximatively $0.25*10^{-9}$, highly unlikely (two in ten billion).

[4] Here the partitioning scheme has to be able to detect the previously inserted noise injection nodes and ignore them such as to produce a partition close/identical to the one produced in step 10 of the embedding procedure.

For $n > i > l > j$: Attack partially destroyed watermark. Labeling scheme still holds. The probability of this happening for $n = 26$, $l = 6$ and $j = 4$ is approximatively $4.3 * 10^{-6}$, again highly unlikely (four in a million).

Even in this case, if we assume, for simplicity purposes, uniform weights in the hierarchical watermarking process, a certain amount of the original watermark is still retrievable, yielding an additional positive confidence level of roughly 33% (i.e. approx. $1 - \frac{j}{l}$).

More sophisticated attacks need to be envisioned and researched, the impact on the watermarking algorithm to be assessed and the adjusted accordingly.

5 Experiments.

In order to reach a production level implementation of the presented method, various experiments and associated software have to be developed. Given it's importance to the overall algorithm, we first implemented a test suite that allows experiments with the presented labeling method on abstract, dynamically redefineable structures composed of a customizable number of nodes with associated random generated (or predefined) content. We performed mainly experiments on structures with around 25 nodes and various level of connectedness. The computations were conducted on a 500Mhz PC with 128MB RAM running Linux. Code was written mainly in Java and compiled with the Sun JDK. Even so, none of the encountered computations took more than 3-4 seconds at most. Labeling an entire graph of 26 nodes and around 60 edges, with around 6 iterations per labeling cycle, took around 35 seconds. Actual summarized results and examples should become available online (January 2002) at (removed for reviewing purposes).

Labeling experiments

As experiments went along we changed the labeling algorithm accordingly, reaching the current presented stage. For example, we realized early on that a certain level of adjustability has to be reached that allows for customization according to the specifics of each structure. Some excerpts of the working conclusions are presented bellow:

Depending on the localization of certain nodes, removing an incident edge or neighboring node changes the labels pretty fast, although in the first roughly $\frac{diameter}{2}$ iterations they look pretty much the same (good news). After that, as expected the label values start to diverge into something not recognizable from the original unchanged graph. Doing some more tests turned out that removing a node is bad but not too bad ... and basically the main affected labels are only the ones of the former neighbors (as expected) ... while the rest of the labels are pretty consistent. Thus our scheme proves to be quite ok for minor node content modification, graph edge removal, if not essential, but the attacker will probably not remove important edges because of value. A limited number of node removals is still controllable and the label intervals can be computed fairly ok. The immunity of this labeling scheme is then related to how much we 'toiled' in trying to purposefully alter things around and recompute the labeling interval. It is interesting to adjust the weights in such a way as to minimize the number of overlapping labels/intervals. Remember that overlapping intervals are unfolding possible weakness in case of a specific attack (because in that case the labels of the two nodes might be the same). The number of iterations performed is also part of the key. The fact that more iterations put the node more in 'perspective' with regard to the ***entire*** graph (not just it's close neighborhood) has to be ballanced with the degrading labeling intervals (as no. of iterations goes up) in case of heavily attacked graphs.

Applicability

We developed a preliminary version of an algorithm aimed at watermarking numeric database content, such as petroleum-digging statistics, customer buying patterns etc. We are currently working on quickly deploying the above presented algorithm in the simple case of HTML. We hope to be able to publish results in the very near future. Other envisioned applicability domains we're looking into are: Bussiness Application Description (XML, e.g. IBM WebSphere, BEA WebLogic application descriptions) Financial/Stock Market published B2B data (XML), Music Layout

definition (XML), SOAP enabled communication of original content.

6 Conclusions

We presented ongoing research in the area of watermarking semi-structured content. Various challenges are associated with this new domain, outlining it as different from any previous research, especially research in the area of media and signal watermarking. Lack of noise and a new transform domain present a whole set of problems and trade-offs. A different domain leads to a different approach and associated algorithm. Benefiting from the dual nature of semi-structures, our algorithm makes use of both the available node content as well as of the value-carying structure, through the processes of partitioning, canonical labeling and noise injection bound together by hierarchical watermarking. The idea behind noise injection is to use traditional, known, content watermarking techniques, in encoding parts of the watermark in the node content, while hierarchical watermarking effectively "amplifies" the power of each encoded sub-part. Providing a canonical labeling scheme, "trained" to tolerancy for a set of graph modifications was essential in being able to later-on identify noise injection points. Also, worth mentioning is the fact that our algorithm does not require the original un-watermarked object in order to perform mark detection.

Further work is required in semantical graph partitioning, content hashing and labeling. Different application domains will require specific approaches. Alternative ideas include using bandwidth available in the specifications of inter-node relations, Effective implementation of an XML bussiness application description watermarking application is imminent and to be presented in a near-future systems paper. With the advent of technologies such as SOAP and other XML based communication protocols, we envision a broadening of the applicability of our algorithm in the near future.

Acknowledgements: We thank Professors John R. Rice and Robert E. Lynch for the useful comments.

References

[1] M.J. Atallah, V. Raskin (with M. Crogan, C. Hempelmann, F. Kerschbaum, D. Mohamed, and S. Naik). Natural language watermarking: Design, analysis, and a proof-of-concept implementation. In *Lecture Notes in Computer Science, Proc. 4th International Information Hiding Workshop, Pittsburgh, Pennsylvania, April 2001.* Springer Verlag, 2001.

[2] L. Babai and L. Kucera. Canonical labeling of graphs in linear average time. In *Proceedings 20th IEEE Symposium on Foundations of Computer Science (1979), 39-46.*, 1979.

[3] L. Babai and E. Luks. Canonical labeling of graphs. In *Fifteenth Annual ACM Symposium on Theory of Computing, pages 171–183. ACM, 1983.*, 1983.

[4] W. Bender, D. Gruhl, N. Morimoto, and A. Lu. Techniques for data hiding. *IBM Systems Journal*, 35(3–4):313–336, 1996.

[5] B. Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31:509–517, 1985.

[6] G. Cohen, S. Encheva, and G. Zemor. Copyright protection for digital data. In *Workshop on Watermarking and Copyright Enforcement*, Paris France, 1999.

[7] Christian Collberg and Clark Thomborson. Software watermarking: Models and dynamic embeddings. In *Principles of Programming Languages 1999, POPL'99*, San Antonio, TX, January 1999.

[8] Christian Collberg, Clark Thomborson, and Douglas Low. Breaking abstractions and unstructuring data structures (manuscript). October 1997.

[9] Christian Collberg, Clark Thomborson, and Douglas Low. Manufacturing cheap, resilient, and stealthy opaque constructs. In *Principles of Programming Languages 1998, POPL'98*, San Diego, CA, January 1998.

[10] Ingemar Cox, Jeffrey Bloom, and Matthew Miller. Digital watermarking. In *Digital Watermarking.* Morgan Kaufmann, 2001.

[11] Ingemar J. Cox and Jean-Paul M. G. Linnartz. Public watermarks and resistance to tampering. In *International Conference on Image Processing (ICIP'97)*, Santa Barbara, California, U.S.A., 26–29 October 1997. IEEE.

[12] Ingemar J. Cox, Matt L. Miller, and A. L. McKellips. Watermarking as communications with side information. *Proceedings of the IEEE (USA)*, 87(7):1127–1141, July 1999.

[13] Scott Craver. On public-key steganography in the presence of an active warden. In *Information Hiding*, pages 355–368, 1998.

[14] Edward J. Delp. Watermarking: Who cares? does it work? In Jana Dittmann, Petra Wohlmacher, Patrick Horster, and Ralf Steinmetz, editors, *Multimedia and Security – Workshop at ACM Multimedia'98*, volume 41 of *GMD Report*, pages 123–137, Bristol,

United Kingdom, September 1998. ACM, GMD – Forschungszentrum Informationstechnik GmbH, Darmstadt, Germany.

[15] Stefan Katzenbeisser (editor) and Fabien Petitcolas (editor). Information hiding techniques for steganography and digital watermarking. In *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House, 2001.

[16] Bob Ellis. Public policy: New on-line surveys: Digital watermarking. *Computer Graphics*, 33(1):39–39, February 1999.

[17] Faulon J. Automorphism partitioning and canonical labeling can be solved in polynomial time for molecular graphs. In *J. Chem. Inf. Comput. Sci. 38, 1998, 432-444.*, 1998.

[18] M. Kobayashi. Digital watermarking: Historical roots. IBM Research Report RT0199, IBM Japan, Tokyo, Japan, April 1997.

[19] Ludek Kucera. Canonical labeling of regular graphs in linear average time. In *IEEE Symposium on Foundations of Computer Science*, pages 271–279, 1987.

[20] P. Moulin and J. O'Sullivan. Information-theoretic analysis of information hiding. In *P. Moulin and J. A. O'Sullivan, Information-Theoretic Analysis of Information Hiding*, 1999.

[21] Pierre Moulin, M. K. Mihcak, and Gen-Iu (Alan) Lin. An information–theoretic model for image watermarking and data hiding (manuscript). 2000.

[22] J. Palsberg, S. Krishnaswamy, M. Kwon, D. Ma, Q. Shao, and Y. Zhang. Experience with software watermarking. In *Proceedings of ACSAC, 16th Annual Computer Security Applications Conference*, pages 308–316, 2000.

[23] Fabien A. P. Petitcolas, Ross J. Anderson, and Markus G. Kuhn. Attacks on copyright marking systems. In David Aucsmith, editor, *Information Hiding: Second International Workshop*, volume 1525 of *Lecture Notes in Computer Science*, pages 218–238, Portland, Oregon, U.S.A., 1998. Springer-Verlag, Berlin, Germany.

[24] Fabien A. P. Petitcolas, Ross J. Anderson, and Markus G. Kuhn. Information hiding - a survey. *Proceedings of the IEEE*, 87(7):1062–1078, July 1999. Special issue on protection of multimedia content.

[25] B. Pfitzmann. Information hiding terminology. In Ross Anderson, editor, *Information hiding: first international workshop, Cambridge, U.K., May 30–June 1, 1996: proceedings*, volume 1174 of *Lecture Notes in Computer Science*, pages 347–350, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1996. Springer-Verlag.

[26] Claude Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, 1949.

[27] Radu Sion, Mikhail Atallah, and Sunil Prabhakar. Power: Metrics for evaluating watermarking algorithms. In *Proceedings of ITCC02*, 2002.

[28] M. D. Swanson, B. Zhu, A. H. Tewfik, and L. Boney. Robust audio watermarking using perceptual masking. *Signal Processing*, 66(3):337–355, May 1998.

[29] G. Voyatzis, N. Nikolaidis, and I. Pitas. Digital watermarking: an overview. In S. Theodoridis et al., editors, *Signal processing IX, theories and applications: proceedings of Eusipco-98, Ninth European Signal Processing Conference, Rhodes, Greece, 8–11 September 1998*, pages 9–12, Patras, Greece, 1998. Typorama Editions.

[30] Jian Zhao and Eckhard Koch. A generic digital watermarking model. *Computers and Graphics*, 22(4):397–403, August 1998.

[31] Jian Zhao, Eckhard Koch, Joe O'Ruanaidh, and Minerva M. Yeung. Digital watermarking: what will it do for me? and what it won't! In ACM, editor, *SIGGRAPH 99. Proceedings of the 1999 SIGGRAPH annual conference: Conference abstracts and applications*, Computer Graphics, pages 153–155, New York, NY 10036, USA, 1999. ACM Press.