CERIAS Tech Report 2002-28

WATERMARKING RELATIONAL DATABASES

by Radu Sion, Mikhail Atallah, Sunil Prabhakar

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

Watermarking Relational Databases

CERIAS TR 2002-28 *

Radu Sion, Mikhail Atallah, Sunil Prabhakar
Center for Education and Research in Information Assurance,
Computer Sciences, Purdue University
West Lafayette, IN, 47907, USA
http://www.cs.purdue.edu/homes/sion
[sion, mja, sunil]@cs.purdue.edu

Abstract

Digital Watermarking, in the traditional sense 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 rights over the content.

Recent research [29] of the authors introduces the issue of digital watermarking for generic number sets. In the present paper we expand on this foundation and introduce a solution for relational database content security through watermarking. To the best of our knowledge there is no research on this issue. Our solution addresses a series of important attacks, such as data re-sorting, subset selection (up to 30% and above data loss tolerance), linear data changes. Finally we present **dbwm.***, a proof-of-concept implementation of our algorithm and its application on real life data, namely in watermarking data from the outsourced Wal-Mart sales database of the years 1999-2000.

1 Introduction

Digital watermarking has traditionally [6] [8] [9] [10] [11] [12] [18] [23] [24] [34] [36] [37] relied upon the availability of a large noise domain within which the object can be altered while retaining its essential

*Portions of this work were supported by Grants EIA-9903545, IIS-9985019 and IIS-9972883 from the National Science Foundation, Contract N00014-02-1-0364 from the Office of Naval Research, and by sponsors of the Center for Education and Research in Information Assurance and Security.

properties. For example, the least significant bits of image pixels can be arbitrarily altered with little impact on the visual quality of the image (as perceived by a human). In fact, much of the "bandwidth" for inserting watermarks (such as in the least significant bits) is due to the inability of human sensory system (especially sight and hearing) to detect certain changes.

More recently, the focus of watermarking for digital rights protection is shifting towards data types such as text, software, and algorithms. Since these data types have very well defined semantics (as compared to those of images, video, or music) and may be designed for machine ingestion, the identification of the available "bandwidth" for watermarking is as important a challenge as the algorithms for inserting the watermarks themselves.

The goal of watermarking [30] is to insert an indelible mark in the object such that (i) the insertion of the mark does not destroy the value of the object (i.e. the object is still useful for the *intended purpose*); and (ii) it is difficult for an adversary to remove or alter the mark beyond detection without destroying the value of the object. Clearly, the notion of value or utility of the object is central to the watermarking process. This value is closely related to the type of data and its intended use. For example, in the case of software the value may be in ensuring equivalent computation, and for text it may be in conveying the same meaning (i.e. synonym substitution is acceptable). Similarly for a collection of numbers the value may lie in the actual values, in the relative values of the numbers, or in the distribution (e.g. normal with a certain mean).

Although extensive research has been conducted on the problem of watermarking multimedia data (images, video and audio), there is relatively little work on watermarking other types of data. Recent work has addressed the problems of software watermarking [5] [22] [33] and natural language watermarking [1]. Here we study the issue of watermarking numeric relational content.

Whereas extensive research has focused on various aspects of DBMS security, including access control techniques as well as data security issues [2] [3] [4] [13] [14] [15] [16] [19] [20] [21] [26] [28], to the best of our knowledge there is no previous work addressing this very problem of relational data security through watermarking.

Protecting rights over outsourced digital content is of ever increasing interest, especially considering areas where sensitive, valuable data is to be outsourced. A good example are data mining applications (e.g. Wal-Mart sales database, oil drilling data, financial data etc), where a set of data is usually produced/collected by a data collector and then sold in pieces to parties specialized in mining on that data. Given the nature of most of the data, it is hard to associate rights of the originator over it. Watermarking can be used to solve this issue.

An important point about watermarking should be noted. By its very nature, a watermark modifies the item being watermarked. If the object to be watermarked cannot be modified then a watermark cannot be inserted. The critical issue is not to avoid changing the data, but to be able to make minimal changes to the data while retaining its value. Clearly, one can always identify some use of the data that is affected by even a minor change to the any portion of the data. It is therefore necessary that the intended purpose of the data that should be preserved be identified during the watermarking process.

In this paper we explore the issue of securing valuable outsourced data through the process of watermarking, enabling future court-proofs assessing proper rights over the content. Thus, the main contributions of the present work consist of

- a resilient watermarking method
- a technique for enabling user-level run-time control over properties that are to be preserved as well as the degree of change introduced
- a proof-of-concept implementation for numeric relational data
- the deployment of the proof-of-concept implementation on real data, in watermarking the Wal-Mart Sales Database.

The algorithms introduced here proved to be resilient to various important classes of attacks, including data re-shuffling/sorting, massive subset selection, linear data changes, random item(s) alterations attacks etc.

The paper is structured as follows. Section 2 presents the considered framework issues and main associated challenges as well as a initial solution to a primitive problem (watermarking numeric collections) to be used later in the global algorithm. Section 3 links the generic solutions defined previously to the relational database framework. Section4 presents a practical example illustrating our algorithm. Section 5 presents main conclusions and defines our current ongoing efforts as part of a broader frame of future envisioned research.

2 Simplified Problem: Watermarking Numeric Collections

This section deals with the foundations of a primitive numeric collection watermarking procedure (an extension of our work in [29]) that will be later deployed as a sub-routine in the main watermarking algorithm. Thus here we are not actually entering into the relational database realm but rather keep the domain as unrestricted as possible.

Often it might be easier to consider the term "numeric relation attribute" instead of "number set" used here. Section 3 makes the connection to the database framework by integrating the primitive watermarking solution into a more generic relational data framework.

Let $\mathbb{S}$ be a set of n real numbers $\mathbb{S} = \{s_1, ..., s_n\} \subset \mathbb{R}$, Let $\mathbb{V}$ be the result of watermarking $\mathbb{S}$ by minor alterations to its content. For now we assume $\mathbb{V} = \{v_1, ..., v_n\} \subset \mathbb{R}$ is also of size n. Let a string of bits w of size $m << n$ be the desired watermark to be embedded into the data ($|w| = m$). We will use the notation w_i to denote the i-th bit of w.

2.1 Data Usability

In the following we define a certain value (allowable "usability") metric of a data collection that will enable us to determine the watermarking result as being valuable and valid, within permitted/guarantee error bounds. Thus, our algorithm relies on the knowledge of the guaranteed (e.g. at watermarking/outsourcing time) restrictions/properties required of the actual data.

For each relevant subset $S_i \subset \mathbb{S}$ let $G_i = \{{G_1}^i, ..., {G_p}^i | {G_j}^i : subsets(\mathbb{S}) \to \mathbb{R}\}$ be a (possible empty) set of "usability metric functions", that S_i has to satisfy within a certain set of allowable (i.e. guaranteed result error bounds) usability intervals ${G_i}^\delta = \{(({g_1}^i)_{min}, ({g_1}^i)_{max}), ..., (({g_p}^i)_{min}, ({g_p}^i)_{max})\}$, such that the following "usability condition" holds: ${G_j}^i(S_i) \in (({g_j}^i)_{min}, ({g_j}^i)_{max}) \forall j \in (1, p)$.

In other words we define the allowable distortion bounds for the given input data ($\mathbb{S}$) in terms of "usability" metrics (see "usability" in [30]) which are given at watermarking time and are defined by the actual purpose (see "usability domain" in [30]) of the data.

Note: One simple but extremely relevant example is the *maximum allowable mean squared error* case, in which the usability metrics are defined in terms of mean squared error tolerances (see [29]).

We define the *general simplified problem* of watermarking the set $\mathbb{S}$ as the problem of finding a transformation from $\mathbb{S}$ to another item set $\mathbb{V}$, such that, given all possible imposed usability metrics sets $\mathbb{G} = \cup G_i$ for any and all subsets $S_i \subset \mathbb{S}$, that hold for $\mathbb{S}$, then, after the transformation yields $\mathbb{V}$, the metrics should hold also for $\mathbb{V}$ [1]. We call $\mathbb{V}$ the "watermarked" version of $\mathbb{S}$.

2.2 Discussion

Problem Formulation. For a numeric collection, a natural starting point for defining the allowed change is to specify an absolute (or relative) change in value. For example, each value may be altered by

[1]In other words, if $\mathbb{G}$ is given and holds for the initial input data, $\mathbb{S}$ then $\mathbb{G}$ should also hold for the resulting data $\mathbb{V}$.

no more than 0.0005 or 0.02%. Moreover a bound on the cummulative change may be specified. This is similar to the approach used for traditional multimedia watermarking. For example the Encyclopedia Britannica introduces small errors in the published figures (such as population, and surface area of countries). Small changes in these values do not significantly affect their use [2].

Also, an adversary attempting to destroy a watermark becomes much more effective if he can identify the values in which the watermark has been embedded. In addition to specifying properties of the data that should be preserved for usability [30], constraints can be used to prevent easy detection of watermark locations. For example a tuple with start time later than its corresponding end time, or a customer with age less than 12 years are very likely to be detected as resulting from watermarking.

Semantics. Specifying only allowable change limits on individual values and possibly an overall limit, fails to capture important semantic features associated with the data – especially if the data is structured. Consider for example, age data. While a small change to the age values may be acceptable, it may be critical that individuals that are younger than 21 remain so even after watermarking if the data will be used to determine behavior patterns for under-age drinking. Similarly, if the same data were to be used for identifying legal voters, the cut-off would be 18 years. Further still, for some other application it may be important that the relative ages (in terms of which one is younger) not change. Other examples of constraints include: (i) *uniqueness* – each value must be unique; (ii) *scale* – the ratio between any two number before and after the change must remain the same; and (iii) *classification* – the objects must remain in the same class (defined by a range of values) before and after the watermarking. As is clear from the above examples, simple bounds on the change of numerical values are not enough. Also, the allowable change depends upon the nature of the application in addition to the type of data.

Structured Data. Structured collections, for example a collection of relations presents further constraints that must be adhered to by the watermarking algorithm. For example, consider data warehouse data organized using a standard Star schema with a fact table and several dimension tables. It is important that the key relationships be preserved by the watermarking algorithm. This is similar to the "Cascade on update" option for foreign keys in SQL and ensures that tuples that join before watermarking also join after watermarking. Furthermore, we may want to ensure that only tuples that joined before watermarking join after watermarking. This requires that the new value for any attribute should be unique after the watermarking process. In other words, we want to preserve the relationship between the various tables. More generally, the relationship could be expressed in terms of an arbitrary join condition, not just a natural join. In addition to relationships between tuples,

[2]Note however that this primitive "watermark" is easy to defeat simply by making further small changes to the values.

relational data may have constraints within tuples. For example if a relation contains the start and end times of a web interaction, it is important that each tuple satisfies the condition that the end time be later than the start time.

2.3 Attacks

Having formulated the problem as above, before attempting any solution, let us first outline classes of envisioned attacks.

A1. Subset Selection. The attacker can randomly select and use a subset of the original data set, subset that might still provide value for its intended purpose (subtractive attack).

A2. Subset Addition. The attacker adds a set of numbers to the original set. This addition is not to significantly alter the "valuable" (from the attacker's perspective) properties of the initial set versus the resulting set.

A3. Subset Alteration. Altering a subset of the items in the original data set such that there is still value associated with the resulting set. A special case needs to be outlined here, namely (A3.a) a linear transformation performed uniformly to all of the items. This is of particular interest as such a transformation preserves many data-mining related properties of the data, while actually altering it considerably, making it necessary to provide resilience against it.

A4. Subset Re-sorting. If a certain order can be imposed on the data then watermark retrieval/detection should be resilient to re-sorting attacks and should not depend on this predefined ordering.

Given the attacks above, several properties of a successful solution surface. For immunity to **A1**, the watermark has to be embedded in overall collection properties that survive subset selection (e.g. numeric confidence intervals, see later).

If the assumption is made that the attack alterations are within problem distortion bounds, defined by the usability metric functions (otherwise the data might loose its associated value), then **A3** should be defeat-able by embedding the primitive mark in resilient global item set properties.

As a special case, **A3.a** can be defeated by a preliminary normalization step in which a common divider to all the items is first identified and divided by. For a given item X, for notation purposes we are going to denote this "normalized" version of it by $NORM(X)$.

Because it adds new data to the set, defeating **A2** seems to be the most difficult task, because it implies the discovery of the possible data usability domains [30] for the attacker. That is, we have to be able to pre-determine what the main uses (for the attacker) for the given data set/type are.

Finally, an extra step, in part defeating all of the above presented attacks, assures the ability to "recognize" *most* of the collection items before and after watermarking and/or a security attack. This is especially important as a response to attacks of type **A4**.In other words if an item was accessed/modified before watermarking, e.g. being identified with a certain label L, then hopefully at watermark detection time the same item is still identify-able with the same label L or a known mapping to the new label. Bringing this a little bit further, making some less restrictive assumptions, we would like to be able to identify *a majority of the initial elements of a subset* after watermarking and/or attacks. (As we will see later on, "missing" a small number of items is not making it much worse as the marking method is resilient to that.)

This final step lexicographically sorts the items in the collection, based on a one-way, secretly keyed, cryptographic hash of the set of most significant bits (MSB) of the normalized version of the items. The secret one-way hashing ensures that an attacker cannot possibly determine what the actual item ordering is. Then, in the next step (see Section 2.4.2), subset "chunks" of the items are selected based on this secret ordering. This defeats **A4** as well as any attempts to actually deploy statistical analysis to determine the secret subsets.

2.4 Solution

Our solution for the *simplified problem* consists of several steps. First, we deploy a resilient method for item labeling, enabling the required ability to "recognize" initial items at watermarking detection time (i.e. after watermarking and/or attacks). In the next step we ensure attack survivability by "amplifying" the power of a given primitive watermarking method. The amplification effect is achieved by the deployment of secrets in selecting collection subsets which will become input for the final stage, where a primitive watermarking method is deployed on selected secret subsets.

2.4.1 Overview

As an overview, the solution for the simplified problem reads as follows.

Encoding

Step E.1. Select a maximal number of unique, non-intersecting (see below) subsets of the original set, as described in Section 2.4.2.

Step E.2. For each considered subset, (E.2.1) embed a watermark bit into it using the encoding convention in Section 2.4.2 and (E.2.2) check for data usability bounds. If usability bounds exceeded, (E.2.3) retry different encoding parameter variations or, if still no success, (E.2.3a) try to mark the subset as invalid (i.e. see encoding convention in Section 2.4.2), or if still no success (E.2.4) ignore the current set [3] We repeat step E.2 until no more subsets are available for encoding. This results in multiple watermarking patterns embedded in the entire data collection.

Decoding

Step D.1. Using the keys and secrets from step E.1, recover a majority of the subsets in E.1, all of them if no attacks were performed on the data.

Step D.2. For each considered subset, using the encoding convention in Section 2.4.2, recover the embedded bit value and re-construct watermarks.

Step D.3. The result of D.2 is a set of copies of the same watermark with various potential errors. This last step uses a majority voting scheme to recover the highest likelihood initial mark.

2.4.2 Basic Algorithm

Current watermarking algorithms draw most of their court-persuasion power [30] from a secret that controlled watermark embedding (i.e. watermarking key). Much of the attack immunity associated with an watermarking algorithm is based on this key and its level of secrecy. Given a weak partial marking technique (e.g. (re)setting a bit), a strong marking method can be derived by a method of "mark amplification", repeatedly applying the weak technique in a keyed fashion on different parts of the object to be watermarked.

Step One: Power Amplification

This step performs exactly the *power amplification* role, as outlined above, namely selecting (based on set of selection secrets/keys) subsets of the initial data on which to apply a simple watermarking method later on, achieving overall a high degree of resilience and power.

More formally, given a collection of items as above, $\mathbb{S} = \{s_1, ..., s_n\} \subset \mathbb{R}$, and a secret 'sorting key" k_s, we first induce a secret ordering on it by sorting according to a cryptographic keyed hash of the most significant bits of the normalized items, e.g. $index(s_i) = H(k_s, MSB(NORM(s_i)), k_s)$. We then build the subsets S_i as "chunks" of items, a "chunk" being a set of adjacent items in the sorted version of the collection (see Figure 1).

This increases the ability to defeat different types of attacks including "cut" and/or "add" attacks (e.g. **A1** , **A2**), by "dispersing" their effect throughout the data, as a result of the secret ordering. Thus, if an attack removes 5% of the items this will result in each subset S_i being 5% smaller. If S_i is small enough and/or if the primitive watermarking method used to encode parts of the watermark

[3]This leaves an invalid watermark bit encoded in the data that will be corrected by majority voting at extraction time.

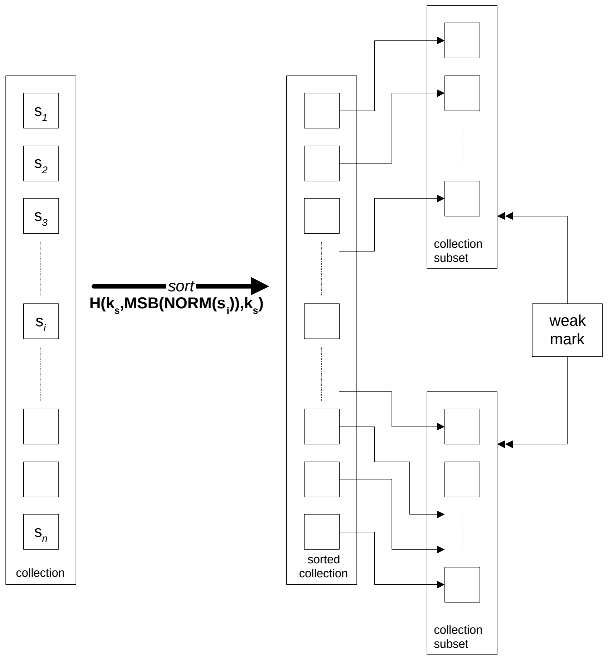

Figure 1: Primitive Mark Power Amplification. Subset selection after sorting on keyed hash of the most significant bits (MSB) of the normalized data items. This enables recovery after various attacks, including re-shuffling/sorting and linear changes.

(e.g. 1 bit) in S_i this is made resilient to these kind of minor transformations (See Figure 3) then the probability of survival of most of the embedded watermarks is higher.

Thus, the main purpose of this step is to amplify the power [30] of the general watermark. The next step will simply consider each S_i to be marked separately by building on a simple watermarking method. The result will be a m-bit (i.e. $i = 1, ..., m$) overall watermark bandwidth in which each bit is embedded/hidden in each of the marked S_i.

Step Two: Watermarking

Once each of the to-be-watermarked secret (keyed) sets S_i are defined, the problem reduces itself to finding a reasonable, not-very-weak (i.e. better than "coin-flip", random occurrence) algorithm for watermarking a medium-sized set of numbers.

Thus, the previous "amplification" provides most of the hiding power of our application (as happens in many current watermarking techniques where secrets are an important avenue for hiding as well as protecting the watermark). The next step actually encodes the watermark bits into the provided sub-sets.

One desired property of an encoding method is the ability to retrieve the encoded information without

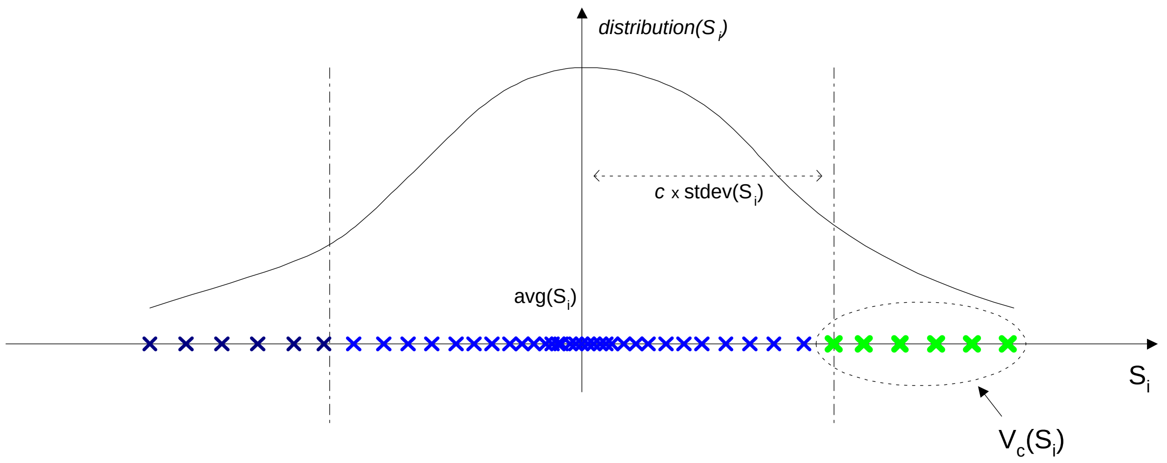

Figure 2: Distribution of item set S_i. Encoding of the watermark bit relies on altering the size of the "positive violators" set, $v_c(S_i)$.

having the original data. This can be important especially in the case of very large dynamic databases (e.g. 4-5 TBytes of data) where data-mining portions were outsourced at various points in time. It is unreasonable to assume the requirement to store each outsourced copy of the original data. Our method satisfies this desiderata.

Confidence Violators.

We are given S_i (i.e. one of the subsets secretly selected in the previous step) as well as the value of a watermark bit b that is to be encoded into S_i. The bit encoding procedure follows.

Let v_{false}, v_{true}, $c \in (0,1)$, $v_{false} < v_{true}$ be real numbers (e.g. $c = 90\%$, $v_{true} = 10\%$, $v_{false} = 7\%$). We call c a *confidence factor* and the interval (v_{false}, v_{true}) *confidence violators hysteresis*. These are values to be remembered also for watermark detection time. We can consider them as part of the encoding key.

Definition: Let $avg(S_i) = \frac{\sum x_j}{|S_i|}$, $\delta(S_i) = \sqrt{\frac{\sum (avg(S_i)-x_j)^2}{|S_i|}} \forall x_j \in S_i$. Given S_i and the real number $c \in (0,1)$ as above, we define $v_c(S_i)$ to be the *number of items of S_i that are greater than $avg(S_i) + c \times \delta(S_i)$*. We call $v_c(S_i)$ the number of positive "violators" of the c confidence over S_i, see Figure 2.

Mark encoding convention: **Given S_i, c, v_{false} and v_{true} as above, we define $mark(S_i) \in \{true, false, invalid\}$ to be *true* if $v_c(S_i) > (v_{true} \times |S_i|)$, *false* if $v_c(S_i) < v_{false} \times |S_i|$ and *invalid* if $v_c(S_i) \in (v_{false} \times |S_i|, v_{true} \times |S_i|)$.**

In other words, the watermark is modeled by the percentage of positive "confidence violators" present in S_i for a given confidence factor c and confidence violators hysteresis (v_{false}, v_{true}).

Actually watermarking the data (i.e. to the desired mark values) in this case resumes to modifying the component elements of S_i such as to yield the necessary value for $v_c(S_i)$, while satisfying all the given data "usability" constraints $\mathbb{G}$.

Note: Encoding the watermark bits into actual data distribution properties (as opposed to directly into the data itself) presents a set of advantages, the most important one being its increased resilience to various types of attacks and the tolerance of considerable data loss (see Figure 3) as compared to the fragility of direct data domain encoding.

It is to be noted that many minor technical issues remain to be solved with the actual encoding method itself. For example, in order to maintain the actual mark reference (i.e. the mean of all values of S_i, $avg(S_i)$), items in S_i are to be altered in pairs. Other simple technical tweaks had to be taken into account. We implemented a working encoding module (**wmdb.***, see Section 4.1) that was used also in the results presented in Section 4.2.

Getting back to the encoding procedure, the question arises on how to perform the required item alterations so as to also satisfy the given "usability" metrics (i.e. $\mathbb{G}$) for the data in case. There are two possible approaches. The first approach simply performs the primitive watermarking step (e.g. for S_i) and then checks for data usability with respect to $\mathbb{G}$. If watermarking altered main usability restrictions, simply ignore S_i and consider the next secretly selected subset to encode the rest of the watermark. This will result in errors in the encoded marks but by using majority voting (see Section 2.5) over a large number of encoded mark copies in the data, the errors will mostly be eliminated in the result. Another idea is to check for data usability after *each* item alteration and adjust the encoding behavior accordingly. This can happen for example by choosing another item (if available) if the current considered one does not preserve data usability after alteration. While more optimal, this approach presents higher computing overhead at watermarking time. For more details see [29].

Discussion: Other Primitive Mark Encodings.

The numeric encoding presented above is suitable for watermarking collections of items with an associated numeric value. It was designed to resist certain types of attacks including the ones presented above in Section 2.3.

Nevertheless to be noted is the fact that an extension to any type of non-numeric data is possible by providing an encoding method with similar characteristics suited for the particular data domain. For example inside a multimedia database, where much of the data are images/audio-clips/video-clips, existing media marking methods [7] [17] [25] [27] [31] [32] [35] could be deployed.

2.5 Resilience.

Given that our method embeds 1 bit per subset, a decision needs to be made determining the size of the subsets selected in the amplification step (i.e. $|S_i|$), as we have an upper bound on the total available mark bandwidth of $\frac{|S|}{|S_i|}$ bits. Thus the size of the selected subsets determines directly the total mark encoding bandwidth. This can and should be considered as a fine-tuning step for the particular data usability metrics provided. If those metrics are too restrictive, more items will be needed inside S_i to be able to encode one bit while still preserving required usability. On the other hand if the usability metrics are more relaxed, S_i can be very small, sometimes even 10-15 items. This enables for more encoding bandwidth overall.

S usually is large enough to provide for multiple embeddings of the original mark which is a requirement in order to survive subset "cut" attacks (i.e. to preserve the mark even inside a "cut" subset of data). We can embed the main watermark no more than $\frac{|S|}{|S_i| \times m}$ times. At watermark detection time, after recovering all the watermark copies from the given data ***majority voting*** over all the recovered watermark bits is deployed in order to determine the most likely initial watermark bits.

Another interesting point to be made here is considering the inherent attack-vulnerability of the watermarking scheme. As shown also in previous research [30], bringing the watermarked data as close as possible to the allowable distortion bounds ("usability vicinity" limits) is of definite benefit in making the result's data usability as fragile as possible to any attack. In effect we integrated this idea also in our primitive encoding implementation. Not only do we embed as many watermark copies as possible, but also, as watermark embedding progresses, a certain embedding aggressiveness factor increases, determining actual changes to the data to be performed more and more up to the permitted limit and not only as required.

We performed various attack-related experiments on real data and determined very promising features of our embedding method. The "confidence-violators" primitive set encoding proves to be resilient to a considerable amount of randomly occurring uniformly distributed surgeries (i.e. attacker with no extra knowledge, in this case item removals) before watermark alterations occur. Even then, there exists the ability to "trace" or approximate the original watermark to a certain degree (i.e. by trying to infer the original mark value from an invalid set). Most of the considered data tolerated up to 20-25% data loss before mark alterations occurred, as illustrated also in Figure 3.

Thus, our watermarking scheme provides resilience for a set of attack types, including re-sorting (**A4**), subset selection (e.g. up to 25-30% tuples removal), massive subset selection (e.g. watermark

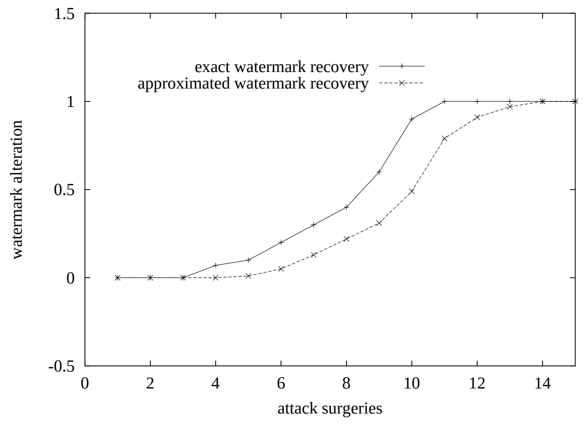

Figure 3: Experiments on resilience to surgeries (data loss, "cut" attacks). The item set size considered was 35, experiments were performed on 10 different sets of close to normally distributed data. Various other parameters: $v_{false} = 5\%, v_{true} = 9\%, c = 88\%$. The average behavior is plotted here. Up to 25% data loss was tolerated easily by the tested data.

preserved even in half of the data if data halved randomly), linear changes (e.g. adding/multiplication), random item(s) alterations attacks that preserve data usability.

3 The relational database

Given the basic solution presented above, this section introduces a solution deployment in the actual relational database framework. We define an associated formalism, describing (e.g. by user at run-time) primitive (e.g. database association rules) and higher level constraints (e.g. "guaranteed" bounds for SQL query behavior) in the database domain. We then show how the previously presented algorithm can be used inside a database, after constraint definition and propagation takes place.

The major distinction between a relational database and a bag of numbers lies in the structure that is imposed by the relational schema as well as the semantics of the data. A naive approach to watermarking a relational database would be to treat each numeric attribute as a set on which the algorithm that we have just described is applied. To avoid significant change, a mean square error bound could be specified for each attribute.

Unfortunately, this approach could destroy the data stored significantly even though individual values are modified only minimally. To see why this is the case, consider two of the most common properties of relational data: primary keys and foreign keys. If we simply apply the earlier algorithm, primary key attributes in a relation may no longer be unique. Similarly, since attributes are watermarked independently, changes to foreign key attributes would change the result of the join of the two tables.

In general, for the relational database setting it is important to preserve structural and semantic properties of the data (as discussed in Section 2.2). We propose to solve this problem by treating each of these properties as a constraint on the usability of the data as described in Section 2.1. Each property of the database that needs to be preserved is written as a constraint on the allowable change to the dataset. The watermarking algorithm presented earlier is then applied with these constraints as input.

Constraints that arise from the schema (chiefly key constraints), can easily be specified in a form similar to (or derived from) SQL *create table* statements. In addition, integrity constraints such as end time being greater than begin time can be expressed. A tolerance (or usability metric) is specified for each constraint. The tolerance is the amount of change or violation of the constraint that is acceptable. This is an important parameter since it can be used to tailor the quality of the watermark (at the expense of greater change in the data). As mentioned earlier, if the tolerances are too low, it may not be possible to insert a watermark in the data.

In order to handle a very wide variety of constraints, we allow constraints to be expressed in terms of arbitrary SQL queries over the relations, with associated requirements (usability metric functions). For example, the requirement that the result of the join (natural or otherwise) of two relations does not change by more than 3% can be specified. Using this approach we can ensure that any changes made by the watermarking algorithm do not violate the required properties. Representative examples of constraints are presented in Section 4 and used to watermark real Wal-mart data warehouse data.

More formally, the constraints can be expressed in terms of query usability. Let $\mathbb{A}$ be a set of considered attributes (e.g. all attributes appearing in a certain database), let $\mathbb{A}_N \subset \mathbb{A}$ be the subset of numeric attributes. Let $\mathbb{R}el(\mathbb{A}) = \{$ set of all possible relations, given the set of attributes $\mathbb{A}$ $\}$. Let $D \subset \mathbb{R}el(\mathbb{A})$ be a considered relational database (e.g. schema and content).

Let $\mathbb{Q} = \{Q_i | Q_i : D \to \mathbb{R}el(\mathbb{A})\}$ be a set of queries over the considered sets of relations. For each query Q_i let $G_i = \{{G_1}^i, ..., {G_p}^i | {G_j}^i : \mathbb{R}el(\mathbb{A}) \to \mathbb{R}\}$ be a set of "usability metric functions", that the result of applying Q_i to D (i.e. $Q_i(D)$) has to satisfy within a certain set of allowable (i.e. guaranteed result error bounds) usability intervals [4], ${G_i}^\delta = \{(({g_1}^i)_{min}, ({g_1}^i)_{max}), ..., (({g_p}^i)_{min}, ({g_p}^i)_{max})\}$, such that the following holds ${G_j}^i(Q_i(D)) \in (({g_j}^i)_{min}, ({g_j}^i)_{max}) \forall j \in (1, p)$.

The proposed algorithm for watermarking relational databases is as follows:

[4]The concept of usability metrics can be taken much further, e.g. defining a complete formalism of constraints and associated data domains etc.

- User-defined queries and associated guaranteed query usability metrics and bounds are specified with respect to the given database.
- User input determines a set of numeric attributes in the database considered for watermarking, possibly all.
- For each selected attribute we then deploy the simplified algorithm with the mention that in step E.2.2 instead of checking for local data usability the algorithm simply checks all global user-defined queries and usability bounds by execution.

Note: It is to be noted that computing times are of no concern at this time as this procedure is performed only once, at embedding time.

An additional benefit of operating in the relational data domain is the ability to use the actual relation key in the secret subset selection procedure, instead of the proposed most significant bits of the data (i.e. watermarked attribute data). It is highly unlikely that an attack will entirely change the database schema and replace the key attribute. Thus for most applications it might be a safe idea to use it (or it's MSB space), especially in cases where the actual data is subject to lax usability metrics (i.e. making the data MSB domain less reliable).

The long-term goal of our research is a solution to the relational database watermarking problem that takes the constraints specified in SQL and automatically generates a watermarked version of the data. In this preliminary step, we focus on the watermarking technique itself. Subject to follow-up research could be a more elegant theoretically optimal solution that proceeds along the following steps:

1. Prove that a given *query* usability formalism and instance as above for the entire database D can result into separate independent collection set usability metrics and bounds over all the numeric attributes in D, $\mathbb{A}_N$.
2. Construct an automatic transformation mechanism for 1.
3. Deploy the simplified algorithm in each of the collection sets defined by the elements in $\mathbb{A}_N$, effectively watermarking all feasible numeric attribute columns.
4. Techniques for efficient testing of constraints (e.g. local testing only) and smart rollback (e.g. Cascading modified key values to foreign keys).

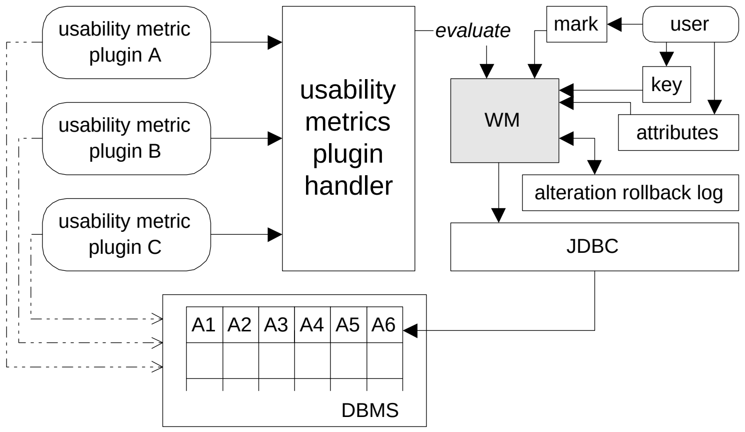
usability metric plugin A
usability metric plugin B
usability metric plugin C
usability metrics plugin handler
evaluate
mark
user
key
WM
attributes
alteration rollback log
JDBC
A1
A2
A3
A4
A5
A6
DBMS

Figure 4: The **wmdb.*** package. Overview.

4 Implementation and Application

This section presents aspects of our implementation and some of the results of watermarking real-life, commercial, data, namely the Wal-Mart Sales relational database.

4.1 The wmdb.* package

wmdb.* is our test-bed implementation of the algorithms presented in this paper. It is written using the Java language and uses the JDBC API in accessing the data.

The package receives as input a watermark to be embedded, a secret key to be used for embedding, a set of relations/attributes to consider in watermarking as well as a set of external *usability plugin modules*. The role of the plugin modules is to allow user defined query metrics to be used at run-time without recompilation and/or software restart [5]. The software then uses those metrics to re-evaluate data usability after each atomic watermarking step as explained in Sections 2.4.2, 2.4.1 and 3.

Once usability metrics are defined and all other parameters are in place, the watermarking module (see Figure 4) starts the process of watermarking. A rollback log is kept for each atomic step performed (i.e. 1-bit encoding) until data usability is assessed and confirmed. This allows for "rollbacks" in case usability is not preserved by the current atomic operation.

Watermark embedding continues until a maximal number of watermark copies have been embedded into the data, while preserving the guaranteed usability metrics. Watermark recovery takes as input the key used in embedding, the set of attributes known to contain the watermark and recovers the

[5]Usability metrics can be thus specified either as SQL queries, stored procedures or simple Java code inside the plug-in modules.

set of watermark copies initially embedded. A final step of majority voting over the recovered copies completes the recovery process.

4.2 Example: Wal-Mart Sales Database

The Wal-Mart Sales Database contains most of the information regarding item sales in Wal-Mart stores nationwide. Its main value lies in the huge commercial potential deriving from data mining discovering buying patterns and association rules. In the following we present some notes regarding our experiments using the wmdb.* package to watermark the Wal-Mart database.

The experimental setup included access to the 4 TBytes Wal-mart data, hosted on a NCR Teradata machine, one 1GHz CPU Linux box with Sun JDK 1.3.1 and 384MB RAM. A subset of the relational schema of the data is presented in the Appendix. The amount of data available is enormous, for example the *Item_Scan* relation contains over 840 million tuples.

For testing purposes we deployed our algorithm on a randomly selected subset of size equal to a small percentage of the original data size (e.g. just 141075 tuples for relation *Item_Scan*). The watermark considered for embedding was the string "(C)walmart" (80 bits in 8bit/character encoding). Algorithm parameters were adjusted repeatedly in an attempt to maximize the number of embedded copies, finally establishing them as $c = 88\%, v_{false} = 5\%, v_{true} = 9\%$. The considered attributes for watermarking were *Item_Scan.Total_Scan_Amount, Item_Scan.Unit_Cost_Amount, Store_Visits.Tender_Amt, Store_Visits.Total_Visit_Amt, Store_Visits.Sales_Tax_Amt.* The size of the subsets considered was roughly 70 for a total of around 2000 available encoding bits in the *Item_Scan.Total_Scan_Amount* attribute for example.

For demonstration purposes, different usability metrics and associated queries were guaranteed, including the following:

(a) Intra-relational Consistency: e.g. Item_Scan.Unit_Cost_Amount x Item_Scan.Item_Quantity = Item_Scan.Total_Scan_Amount

(b) Inter-relational Consistency: e.g. Store_Visits.Total_Visit_Amt ¡ SUM(Item_Scan.Total_Scan_Amount)

(c) General Attribute Constraints: e.g. MSE constraints for attribute Store_Visits.Total_Visit_Amt - introduced normalized mean squared error should not exceed 1%.

(d) General SQL Constraints: e.g. (d.1) for each store and date, the number of sales after watermarking should not deviate more that 2% from the original data, (d.2) for the join between *Store_Visits* and *Item_Scan* on the *Visit_Nbr* attribute, a maximum number of 5% of the elements should be disturbed after watermarking.

For example, the actual numeric value in (d.2) can be formulated as follows:

```
SELECT * AS J1 FROM Item_Scan_orig, Store_Visits_orig
WHERE Item_Scan_orig.Visit_Nbr = Store_Visits_orig.Visit_Nbr;
SELECT * AS J1 FROM Item_Scan_wm, Store_Visits_wm
WHERE Item_Scan_wm.Visit_Nbr = Store_Visits_wm.Visit_Nbr;
SELECT COUNT(*) FROM (
(SELECT * FROM J1 EXCEPT SELECT * FROM J2)
UNION
(SELECT * FROM J2 EXCEPT SELECT * FROM J1))
```

In the working system, each of these metrics is represented by a separate usability metric plug-in, used in evaluating data usability after each atomic watermarking step (see Figure 4).

For example, the usability metric module for (d.2) executes the above query and if the result exceeds a certain threshold, it simply returns false, denying the watermarking module the proposed modifications to the data. The watermarking module then performs a rollback on those modifications and continues on to the next subset as presented in Section 2.4.1.

Deployment Issues

Some of the usability constraints above present a set of deployment challenges especially when implemented as usability plug-in modules. Whereas (a), (b) might be straight-forward to code, (c) presented some complications because of the need of maintaining original reference data in order to be able to compute MSE values. This was solved by creating an additional relation at run-time, used by the plug-in to keep original data that was altered in the watermarked version.

Maybe the most challenging proved to be (d), particularly (d.2) because of the requirement to always compare JOINS on the original data to joins on the resulting data. We tried two approaches. In the first approach, the entire original data was duplicated temporarily [6] and JOINS were dynamically performed at run-time. This soon proved to be un-feasible, a time-hog for computation, often causing JDBC buffer-related crashes and taking up long times to execute.

The next approach optimized the idea by keeping just a record of watermark-related alterations and then directly assessing their impact in the data JOIN result (i.e. determining whether a change in tuple X in table *Store_Visits* will affect a yet unaccessed tuple in the JOIN result with table *Item_Scan*). This second approach, requiring definitely less space and computation power, proved to be working well.

[6]Again, space and computation constraints are not of concern here if within reach, as this is done only once in the lifetime of the outsourced data.

Algorithm deployment times cannot be pre-assessed as much of the time is spent in evaluating usability metrics, and not in the actual encoding algorithm itself. Thus this is to be considered as depending heavily on the optimality of the usability metrics plug-in modules implementations.

Using the attribute *Item_Scan.Total_Scan_Amount*, the watermark was embedded successfully roughly 21 times, leading to a good utilization (84%) of the potential encoding bandwidth of 2000 bits (see above). This allows for a highly accurate final majority voting step at mark retrieval/detection time.

We successfully deployed wmdb.* in watermarking the Wal-Mart Sales Database. As outlined before in Section 2.5 we then performed some attack experiments on parts of the data in order to experimentally assess encoding resilience and obtained encouraging results, some presented in Figure 3. Pre-processed parts of the original data as well as their watermarked version are soon to be found at **http://www.cs.purdue.edu/homes/sion/wm/wmdb**.

5 Conclusions. Future Research.

In the present paper we introduced the problem of data security through watermarking in the framework of numeric relational data.

We (a) design a solution to a simplified version of our problem, namely watermarking a numeric collection by (a.i) defining a new suitable mark encoding method for numeric sets and (a.ii) designing an algorithmic securing mapping (i.e. mark amplification) from a simple encoding method to a more complex watermarking algorithm, and (b) applied the concept to numeric relational databases, thus providing a solution for resiliently watermarking relational databases.

We also developed a proof of concept implementation of our algorithms under the form of a Java software package, **wmdb.*** which we then used to watermark a commercial database extensively used for data-mining in the area of customer trends and buying patterns (i.e. Wal-mart sales database).

Further research should investigate new, non-numeric encoding domains. A model of attacks in this new domain needs to be devised and a more detailed attack-ability analysis performed. A full-fledged commercial watermarking application could be derived from our proof-of-concept software. Finally, different applications for our numeric collection marking method could be envisioned and pursued.

Appendix

The following is an excerpt of the schema of the Wal-Mart Sales Database.

```
(842,556,378 tuples)
```

```
Table Univ_Class_Tables.Item_Scan
  Visit_Nbr Integer Not Null
, Store_Nbr Smallint Not Null
, Item_Nbr Integer Not Null
, Item_Quantity Decimal(9,2) Not Null
, Total_Scan_Amount Decimal(9,2) Not Null
, transaction_Date Date Format 'YYYYMMDD' Not Null
, Unit_Cost_Amount Decimal(9,4) Not Null
, Unit_Retail_Amount Decimal(9,2) Not Null
, Tax_Collect_Code Char(1) Not Null Compress '1'
     Primary Index (Visit_Nbr);

(136,410,119 tuples)
Table Univ_Class_Tables.Store_Visits
          Visit_Nbr          Integer Not Null
        , Store_Nbr          Smallint Not Null
        , Register_Nbr       Byteint Not Null
        , Card_Holder_Nbr    Byteint Not Null
        , Membership_Nbr     Integer Not Null
        , Member_Code        Char(1) Not Null
        , Tender_Type        Byteint Not Null
        , Tender_Amt         Decimal(9,2) Not Null
        , Sales_Tax_Amt      Decimal(9,2) Not Null
        , Total_Visit_Amt    Decimal(9,2) Not Null
        , Transaction_Date   Date Format 'YYYYMMDD' Not Null
        , Transaction_Time   Integer Not Null
        , Refund_Code        Char(1) Compress '0'
        , Upcharge_Code      Char(1) Not Null Compress '0'
        , Tot_Unit_Cost      Decimal(9,2) NOT NULL Default  0.00
        , Tot_Unique_Itm_Cnt Smallint NOT NULL Default  0
        , Tot_Scan_Cnt       Integer NOT NULL Default  0
        , Operator_nbr       Smallint
Unique Primary Index (Visit_Nbr);

(835,760,698 tuples)
Table Univ_Class_Tables.Visit_Scan_lookup
  Store_nbr           SMALLINT  NOT NULL,
, Transaction_date    DATE Format 'YYYYMMDD'NOT NULL,
, Item_Nbr          INTEGER   NOT NULL,
, Visit_nbr           INTEGER   NOT NULL
Primary Index (Store_Nbr, Transaction_Date, Item_Nbr);
```

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] Elisa Bertino, M. Braun, Silvana Castano, Elena Ferrari, and Marco Mesiti. Author-x: A java-based system for XML data protection. In *IFIP Workshop on Database Security*, pages 15–26, 2000.

[3] Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. A flexible authorization mechanism for relational data management systems. *ACM Transactions on Information Systems*, 17(2):101–140, 1999.

[4] Chris Clifton and Don Marks. Security and privacy implications of data mining. In *Workshop on Data Mining and Knowledge Discovery*, pages 15–19, Montreal, Canada, 1996. University of British Columbia Department of Computer Science.

[5] Christian Collberg and Clark Thomborson. On the limits of software watermarking, August 1998.

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

[7] Ingemar J. Cox, Joe Kilian, Tom Leighton, and Talal Shamoon. Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing*, 6(12):1673–1687, 1997.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] J. Hale, J. Threet, and S. Shenoi. A framework for high assurance security of distributed objects, 1997.

[14] E. Hildebrandt and G. Saake. User Authentication in Multidatabase Systems. In R. R. Wagner, editor, *Proc. Ninth Int. Workshop on Database and Expert Systems Applications, August 26–28, 1998, Vienna, Austria*, pages 281–286, Los Alamitos, CA, 1998. IEEE Computer Society Press.

[15] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *IEEE Symposium on Security and Privacy. Oakland, CA*, pages 31–42, 1997.

[16] Sushil Jajodia, Pierangela Samarati, V. S. Subrahmanian, and Eliza Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD 1997*, pages 474–485, 1997.

[17] Neil F. Johnson and Sushil Jajodia. Steganalysis of images created using current steganography software. In *Information Hiding*, pages 273–289, 1998.

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

[19] Li, Feigenbaum, and Grosof. A logic-based knowledge representation for authorization with delegation. In *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop.* IEEE Computer Society Press, 1999.

[20] Matunda Nyanchama and Sylvia L. Osborn. Access rights administration in role-based security systems. In *IFIP Workshop on Database Security*, pages 37–56, 1994.

[21] Sylvia L. Osborn. Database security integration using role-based access control. In *IFIP Workshop on Database Security*, pages 245–258, 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] C. I. Podilchuk and W. Zeng. Image-adaptive watermarking using visual models. *IEEE Journal on Special Areas in Communications*, 16(4):525–539, 1998.

[26] David Rasikan, Sang H. Son, and Ravi Mukkamala. Supporting security requirements in multilevel real-time databases tr-95-21, 1995.

[27] Joseph J. K. Ó Ruanaidh, W. J. Dowling, and F. M. Boland. Watermarking digital images for copyright protection. *I.E.E. Proceedings on Vision, Signal and Image Processing*, 143(4):250–256, 1996.

[28] Ravi S. Sandhu. On five definitions of data integrity. In *IFIP Workshop on Database Security*, pages 257–267, 1993.

[29] Radu Sion, Mikhail Atallah, and Sunil Prabhakar. On watermarking numeric sets. In *Proceedings of IWDW 2002, Lecture Notes in Computer Science, CERIAS TR 2001-60*. Springer-Verlag, 2002.

[30] Radu Sion, Mikhail Atallah, and Sunil Prabhakar. Power: Metrics for evaluating watermarking algorithms. In *Proceedings of IEEE ITCC02, CERIAS TR 2001-55*. IEEE Computer Society Press, 2002.

[31] P. Su, C. Jay, and K. Wang. Blind digital watermarking for cartoon and map images. In *Proc. of SPIE International Symposium Electronic Imaging*, 1999.

[32] Mitchell D. Swanson, Bin Zhu, and Ahmed H. Tewfik. Transparent robust image watermarking. In *1996 SPIE Conf. on Visual Communications and Image Proc.*, volume III, pages 211–214, 1996.

[33] R. Venkatesan, V. Vazirani, and S. Sinha. A graph theoretic approach to software watermarking. In *Proceedings of the Fourth International Hiding Workshop, IH01*, 2001.

[34] 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.

[35] M. Wu and B. Liu. Watermarking for image authentication. In *Proceedings of ICIP*, 1998.

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

[37] 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.