CERIAS Tech Report 2003-11

RIGHTS PROTECTION FOR RELATIONAL DATA

by Radu Sion, Mikhail Atallah, Sunil Prabhakar

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

Rights Protection for Relational Data *

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

ABSTRACT

Protecting rights over relational data is of ever increasing interest, especially considering areas where sensitive, valuable content is to be outsourced. A good example is a data mining application, where data is sold in pieces to parties specialized in mining it.

Different avenues for rights protection are available, each with its own advantages and drawbacks. Enforcement by legal means is usually ineffective in preventing theft of copyrighted works, *unless* augmented by a digital counter-part, for example watermarking.

Recent research 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 rights protection through watermarking.

Our solution addresses important attacks, such as data re-sorting, subset selection, linear data changes (applying a linear transformation on arbitrary subsets of the data). Our watermark also survives up to 50% and above data loss.

Finally we present wmdb.*, a proof-of-concept implementation of our algorithm and its application to real life data, namely in watermarking the outsourced Wal-Mart sales data that we have available at our institute.

1. INTRODUCTION

Digital Watermarking aims to protect a certain content from unauthorized duplication and distribution by enabling provable ownership over the content.

It has traditionally [6] [7] [14] relied upon the availability of a large noise domain within which the object can be altered while retaining its essential 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 [17] 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 utility of the data 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 a considerable amount of research effort has been invested in 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] [13] and natural language watermarking [2]. Here we study the issue of watermarking numeric relational content.

Protecting rights over outsourced digital content is of ever increasing interest, especially considering areas where sensitive, valuable data is to be outsourced. Good examples 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 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.

*Portions of this work were supported by Grants EIA-9903545, ISS-0219560, IIS-9985019 and IIS-9972883 from the National Science Foundation, 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.

SIGMOD 2003, June 9-12, San Diego, California USA

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 limit the change to acceptable levels with respect to the intended use of the data. 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.

Whereas extensive research has focused on various aspects of DBMS security, including access control techniques as well as data security issues [3] [4] [8] [9] [10] [11] [12] to the best of our knowledge only one simultaneous published related effort is available for comparison [**?**]. Numerous fundamental differences distinguish our results from this effort, including but not limited to, the actual method of watermarking, the fact that the method presented here is built around a framework considering higher level semantics to be preserved in the original data (an essential step, not addressed in [**?**]), the increased resulting level of resilience, the openess, portability and ease of use of our implementation. A more in depth discussion can be found in Section 4.1.

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 consists of

- a resilient watermarking method for relational data
- 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 complete, user-friendly implementation for numeric relational data
- the deployment of the implementation on real data, in watermarking the Wal-Mart Sales Database and the analysis thereof

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

The paper is structured as follows. Section 2 presents the main challenges for watermarking relational databases. Section 3 introduces an initial solution to a primitive problem (watermarking numeric collections) to be used later in the global algorithm. Section 4 develops a solution for relational databases by building upon the primitive solution developed earlier. Related work is discussed in Section 4.1. Section 5 presents implementation details as well as experiments and evaluations of the proposed watermarking technique on real outsourced Walmart warehouse data. Section 6 presents main conclusions of our work and discusses avenues for future research.

2. CHALLENGES FOR WATERMARKING RELATIONAL DATABASES

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 a simple numeric set watermarking algorithm (e.g. [15]) is applied. In order to avoid significant change, a mean square error bound could be specified for each attribute.

Unfortunately, this approach could significantly destroy the data even though individual values are modified only minimally, because it does not consider the final data consumer (i.e. 'usability") and the associated allowable distortion constraints.

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 our earlier algorithm directly, 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.

Thus it becomes clear that relational data presents a different set of challenges and associated constraints. The usually low noise bandwidth (i.e. low tolerance to extensive changes) of major relational framework data customers (e.g. data-mining) require a different approach, taking more carefully into account the actual tolerated changes.

2.1 Available Bandwidth

An important first step in inserting a watermark into a relational database (and thereby altering it), is to identify changes that are acceptable. As was mentioned earlier, the acceptable nature and level of change is dependent upon the application for which the data is to be used. In the following we define a metric that will enable us to determine the watermarking result as being valuable and valid, within permitted/guaranteed error bounds. The available "bandwidth" for inserting the bits of the watermark text is therefore not defined directly. Instead we define allowable distortion bounds for the input data in terms of metrics. If the watermarked data satisfies the metrics, then the insertion of the watermark is successful. We term such data to be usable for the intended purpose, or that it is within the "usability" bounds. Further details on the use of usability bounds and their definition can be found in [17].

Example 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 as follows:

$$(s_i - v_i)^2 < t_i \quad \forall i = 1, ..., n \quad (1)$$
$$\sum (s_i - v_i)^2 < t_{max} \quad (2)$$

where $\mathbb{S} = \{s_1, ..., s_n\} \subset \mathbb{R}$, is the data to be watermarked, $\mathbb{V} = \{v_1, ..., v_n\}$ is the result, $\mathbb{T} = \{t_1, ..., t_n\} \subset \mathbb{R}$ and $t_{max} \in \mathbb{R}$ define the guaranteed error bounds at data distribution time. In other words $\mathbb{T}$ defines the allowable distortions for individual elements in terms of mean squared error (MSE) and t_{max} the overall permissible MSE.

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 [1].

Database 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

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

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.

Structured Data Structured collections, for example a collection of relations, present further constraints that must be adhered to by the watermarking algorithm. Consider a data warehouse 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. 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, 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.

2.2 Model of the Adversary

In order to be effective, the watermarking technique must be able to survive a wide variety of attacks. These attacks may be malicious with the explicit intent of removing the watermark, or may be the result of normal use of the data by the intended user.

A1. Subset Selection The attacker can randomly select and use a subset of the original data set 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 useful (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 against **A1**, the watermark has to be embedded in overall collection properties that survive subset selection (e.g. confidence intervals).

If the assumption is made that the attack alterations do not destroy the value of the data, then **A3** should be defeatable by embedding the primitive mark in resilient global data 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 applied. For a given item X, for notation purposes we are going to denote this "normalized" version of it by $NORM(X)$.

Since it adds new data to the set, defeating **A2** seems to be the most difficult task, as it implies the ability to identify potential uses of the data (for the attacker).

Subset Recovery Another interesting requirement is the ability to "recognize" all (or at least *most*) of the collection items before and after watermarking and/or an attack. That is, how do we "recognize" an item and its corresponding subset after it has been changed slightly? While this requirement in part defeats all of the above-mentioned attacks, it is especially important as a response to attacks of type **A4**.

3. SIMPLIFIED PROBLEM: NUMERIC COLLECTIONS

This section deals with the foundations of a primitive numeric collection watermarking procedure (an extension of our work in [15]) that will be later deployed as a sub-routine in the main watermarking algorithm. Thus here we do not enter 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 4 makes the connection to the database framework by integrating the primitive solution into a generic relational 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.

Allowable change. 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 no more than 0.0005 or 0.02%. Moreover a bound on the cummulative change may be specified.

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 deploying secrets in the process of selecting the subsets to become input for the final stage, in which a primitive encoding method is deployed.

3.1 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 de-

scribed in Section 3.3.

Step E.2. For each considered subset, (E.2.1) embed a watermark bit into it using the encoding convention in Section 3.3 and (E.2.2) check for data usability bounds. If usability bounds are 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 3.3), or if still no success (E.2.4) ignore the current set. [2] We repeat step E.2 until no more subsets are available for encoding. This results in multiple embeddings in the data.

Note: Different levels of granularity are possible. For example checking for data usability could be done at an even more atomic level, such as inside the bit-encoding procedure.

Decoding

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

Step D.2. For each considered subset, using the encoding convention in Section 3.3, 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 set of error correcting mechanisms (e.g. majority voting schemes) to recover the highest likelihood initial mark.

3.2 Subset Selection

Watermarking a data collection requires 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 identified with the same label L or a known mapping to the new label.

More generally, 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, our technique is resilient to "missing" a small number of items.

While research efforts of the authors include work in this area (see "tolerant canonical labeling" in [16]) we are going to present a simplified tailored solution.

Our solution is based on lexicographically sorting the items in the collection, sorting occuring based on a one-way, secretly keyed, cryptographic hash of the set of most significant bits (MSB) of the normalized (see Section 2.2) version of the items. The secret one-way hashing ensures that an attacker cannot possibly determine the ordering. In the next step (see Section 3.3), subset "chunks" of the items are selected based on this secret ordering. This defeats **A4** as well as any attempts at statistical analysis to determine the secret subsets. Chunk-boundaries ("subset markers") are then computed and stored for detection time.

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 induce a secret ordering on it by sorting according to a cryptographic keyed hash of the most significant bits of the normalized items. Thus we have: $index(s_i) = H(k_s, MSB(NORM(s_i)), k_s)$.

The MSB space here is assumed to be a domain where minor changes on the collection items (changes that still satisfy the given required usability metrics) have a minimal impact on the MSB labels. This is true in many cases (as usually the usability metrics are related to preserving the "important" parts of the original data). If not suitable, a different labeling space can be envisioned, one where, as above, minor changes on the collection items have a minimal impact.

Note: In the relational data framework, the existence of a primary key associated with the given attribute to be watermarked can make it easier to impose a secret sorting. For more details see Section 4.

3.3 Power Amplification

Current watermarking algorithms draw most of their court-persuasion power [17] from a secret that controlled watermark embedding (i.e. watermarking key). Much of the attack immunity associated with a 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 data being watermarked.

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.

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 roughly 5% smaller. If S_i is small enough and/or if the primitive watermarking method used to encode parts of the watermark (i.e. 1 bit) in S_i is made resilient to these kind of minor transformations (see Section 5.2) then the probability of survival of most of the embedded watermarks is accordingly higher (see Section 3.4.2).

Additionally, in order to provide resilience to massive "cut" attacks, we will select the subset "chunks" to be of sizes equal to a given percent of the overall data set (i.e. not of fixed absolute sizes). This choice provides adaptability of our subset selection scheme to such attacks, assuring subsequent retrieval of the watermark even from, say, half of the original data.

3.4 Watermark Embedding

Once each of the to-be-watermarked secret (keyed) sets S_i are defined, the problem reduces 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 encodes the watermark bits into the provided subsets.

A desired property of an encoding method is the ability to retrieve the encoded information without having the original data. This can be important especially in the case of very

[2] This leaves an invalid watermark bit encoded in the data that will be corrected by the deployed error correcting mechanisms (e.g. majority voting) at extraction time.

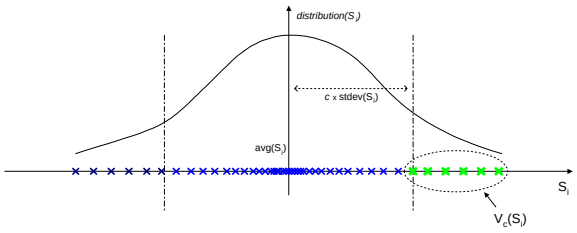

Figure 1: Distribution of item set S_i. Encoding of the watermark bit relies on altering the size of the "positive violators" set, $v_c(S_i)$.

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.

3.4.1 Encoding A Single Bit

We now discuss how a single bit is encoded into a selected subset of the data. 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. Let $\mathbb{G}$ represent the set of user specified change tolerance, or usability metrics.

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 $avg(S_i)$ and $\delta(S_i)$ be the average and standard deviation, respectively, of 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 1.

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}).

Encoding the single bit, b, into S_i is therefore achieved by making minor changes to some of the data values in S_i such that the number of postive violaters $(v_c(S_i))$is either (a) less than $v_{false} \times |S_i|$ if $b = 0$, or (b) more than $v_{true} \times |S_i|$ if $b = 1$. Of course the changes made to the data must not violate the change tolerances, $\mathbb{G}$, specified by the user.

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 Section 5.2) as compared to the fragility of direct data domain encoding.

Performing the required item alterations while satisfying the given "usability" metrics (i.e. $\mathbb{G}$) is one of the remaining challenges. To do this, the algorithm deploys the primitive watermarking step (e.g. for S_i) and then checks for data usability with respect to $\mathbb{G}$. If the tolerances are exceeded it simply ignores S_i and considers the next secretly selected subset to encode the rest of the watermark. This will result in errors (misses) in the encoded marks but by deploying error correcting techniques (e.g. majority voting, see Figure 2) the errors are mostly eliminated in the result. For more details see [15].

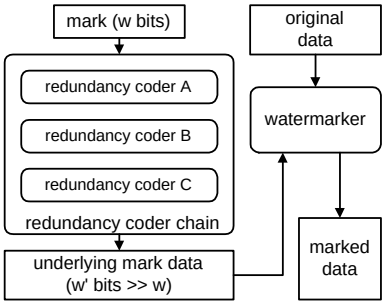

Figure 2: Different error correcting (wmdb.sys.RedundancyCoder) plugins can be added/removed at runtime in order to provide an increased level of resilience for the original watermark to be embeded.

Given that our method embeds 1 bit per subset, a trade-off is to be observed between larger sets (tolerant to more data alteration attacks) but a small bandwidth, and smaller sets (more "brittle" to attacks) but a larger encoding bandwidth. More details are to be found in [15].

At watermark detection time, after recovering all the watermark copies from the given data ***majority voting*** over all the recovered watermark bits (or any other error correcting method for that matter, see Figure 2) can be 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 [17], 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. An attacker will be faced with the problem of removing/altering the watermark and now any changes he performs with this intent have an increased likelihood of making the data invalid with respect to the guaranteed usability metrics [3], thus removing or at least diminishing its value.

3.4.2 Resilience Analysis

A detailed analysis of the encoding presented above can be found in our published results [15].

One of the main questions answered there was:

What is the probability of success of an attacker aiming at destroying at least one watermark bit, as a function of the amount of data damage (i.e. number of surgeries) ?

An answer to the above immediately enables the computation of resilience and attackability bounds of the watermarking algorithm by relating the required damage for a successful attack to the maximum permissible damage levels.

[3]Because the watermarking process already altered the data up to its usability metrics limits.

For example, it can be shown [15] that for a 10000 tuples item set and an encoding with subsets of size 50 and an average 1-bit subset encoding tolerance to 6% data item losses (experimental results show usually up to 25-30% loss tolerance, see Section 5.2), this probability is surprisingly low, virtually zero:

$$\frac{1}{200} \times \frac{1}{50} \times (\frac{49}{50})^{9999} \simeq 1.86 \times 10^{-92} \simeq 0$$

In Section 5.2 we present supporting experimental results.

4. RELATIONAL DATA

In the relational database setting it is important to preserve structural and semantic properties of the data (as discussed in Section 2).

Sometimes it is undesireable or even impossible to map higher level semantic constraints into low level (combined) change tolerances for individual tuples or attributes. It should be noted that not all constraints of the database need to be specified. A practical approach would be to begin by specifying a mean square error bound on individual items. Further semantic or structural constraints that the user would like to preserve can be added to these basic constraints.

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 (e.g. 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 their expression 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 below and used to watermark real Wal-mart data warehouse data.

At watermarking time, data quality is continuosly assessed as an intrinsic part of the marking algorithm in itself. In this respect we can claim that, as opposed to other watermarking algorithms in various domains (e.g. image watermarking), we maintain 100% of the associated data value with respect to a set of given required data "goodness" constraints. We believe this is an essential part of any watermarking application in this low-noise, high fragility domain of relational data (e.g. financial data, oil-drilling coordinates).

The algorithm outline for watermarking relational data is as follows (see Figure 3):

- 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 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.

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).

4.1 Related Work

Research related to the issue of embedding information into a set of numbers [1], can be also found (sometimes implicitly) in various frameworks, associated with different information hiding techniques, e.g. frequency domain embedding, DCT and Wavelet watermarking [6].

Relational data presents a different set of challenges and associated constraints. The usually low noise bandwidth (i.e. low tolerance to extensive changes) of major relational framework data customers (e.g. data-mining) require a different approach, taking more carefully into account the actual tolerated changes on the given data.

Similarly, the watermark *encoding* method, presents specifics more suited to the new constraints, namely the ability to survive a maximum level of attacks and at the same time accommodate the existence of required data 'usability" conditions to be satisfied by the result.

One simultaneous published related effort is available for comparison [**?**]. Numerous fundamental differences distinguish our results from this effort, including but not limited to, the actual method of watermarking, the fact that the method presented here is built around a framework considering higher level semantics to be preserved in the original data (an essential required step, totally ignored by this effort), the increased resulting level of resilience, the openess, portability and ease of use of our implementation.

The requirement that the least significant bit (LSB) in any tuple can be altered has limited applicability. It cannot be applied for many important applications such as data mining that require the preservation of classification. We experimentally demonstrate the ability of our scheme to handle classification constraints in Section 5.2.

A sound and truly resilient watermarking method has to *start* by assesing the final purpose of the content to be watermarked, together with its associated allowable alteration limits. These limits are often times impossible to express as "least significant bits" and require a higher level semantic expression power such as offered by the data goodness plugins.

```
watermark(attribute, wm_key, mark_data[], goodness_plugin_handler, db_primary_key, subset_size, v_min, v_max, c)
    1. sorted_attribute ← sort_on_normalized_crypto_hash(wm_key,db_primary_key,wm_key)
    2. for (i=0; i < length(attribute)/subset_size;i++)
        a. subset_bin ← next subset_size elements from sorted_attribute
        b. compute rollback_data
        c. encode(mark_data[i % mark_data.length], subset_bin, v_min, v_max, c)
        d. propagate changes into attribute
        e. if (not goodness_plugin_handler.isSatisfied(new_data,changes)) then
            i. rollback rollback_data
            ii. continue
        f. else
            i. commit
            ii. embedding_map[i] = TRUE
            iii. subset_boundaries[i] = subset_bin[0]
    3. return embedding_map, subset_boundaries
```

Figure 3: Watermark Embedding Algorithm.

```
detect(attribute, wm_key, db_primary_key, subset_size, v_min, v_max, c, embedding_map[], subset_boundaries[])
    1. sorted_attribute ← sort_on_normalized_crypto_hash(wm_key,db_primary_key,wm_key)
    2. read_pipe ← null
    3. do { tuple ← next_tuple(sorted_attribute) }
    4. until (exists idx such that (subset_boundaries[idx] == tuple))
    5. current_subset ← idx
    6. while (not(sorted_attribute.empty())) do
        a. do {
            i. tuple ← next_tuple(sorted_attribute)
            ii. read_pipe = read_pipe.append(tuple)
        b. } until (exists idx such that (subset_boundaries[idx] == tuple))
        c. subset_bin ← (at most subset_size elements from read_pipe, not including last read element)
        d. read_pipe.remove_all_remaining_elements_but_last_read()
        e. if (embedding_map[current_subset]) then
            i. mark_data[current_subset] ← decode (subset_bin, v_min, v_max, confidence)
            ii. if (mark_data[current_subset] != DECODING_ERROR) then detection_map[current_subset] ← TRUE
        f. current_subset ← idx
    7. return mark_data, detection_map
```

Figure 4: Watermark Detection Algorithm.

Other vulnerabilities appear in [**?**]. True data alteration (e.g. linear changes to an arbitrary subset of the data, non-uniform scaling of all or part of the data, and epsilon-attacks) resilience is not analysed and the encoding method lacks fundamental provisions to resist such alterations. For example, even minor-level epsilon-attacks such as the ones illustrated in Section 5.2 (where our encoding survives up to 97%) would entirely remove the mark.

An analogy can be constructed in the image watermarking framework. There, the LSB aproach to watermarking was among of the inital attempts and immediately proved its limits. For example, one simple attack considers the assumed (also by watermarking party) fact that the LSB space is insignificant. Thus simply altering this space randomly or even more, zero-ing its entire content would immediately remove the watermark. Thus LSB information hiding was immediately discarded as an effective technique for resilient rights protection watermarking [6] [14].

5. IMPLEMENTATION AND EXPERIMENTS

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

5.1 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. [4] The software uses those metrics to re-evaluate data usability after each atomic watermarking step as explained in Sections 3 and 4.

Once usability metrics are defined and all other parameters are in place, the watermarking module (see Figure 5)

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

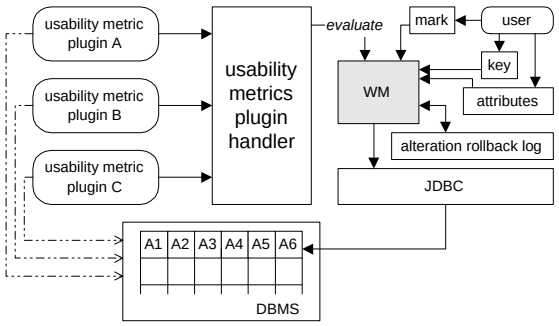

Figure 5: The wmdb.* package. Overview.

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 recovery takes as input the watermarking key used in embedding, the set of attributes known to contain the watermark as well as other various encoding specific parameters (see Figure 4). It recovers the set of watermark copies initially embedded. A final step of error correction (e.g. majority voting) over the recovered copies completes the recovery process.

Note: Various details need to be resolved in order to implement the encoding method. For example, in order to maintain actual mark references (e.g. mean of all values of S_i, $avg(S_i)$), items in S_i are to be altered in pairs. In order to more accurately recover subsets, markers had to be introduced and handled at detection time (see Figure 4).

5.2 Experiments

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 mining buying patterns and association rules. In the following we present some of our experiments using the wmdb.* package to watermark the Wal-Mart database.

Our 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. The amount of data available is enormous. For example, the *ItemScan* 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 a maximum of 141075 tuples for relation *UnivClassTables.StoreVisits*).

In the following we present experiments involving attacks (data loss, data alterations, linear changes, data resorting) as well as the evaluation of the available bandwidth in the presence of different data goodness metrics (tolerable absolute change, data classification preservation goodness).

5.2.1 Data Loss Attacks ("Surgeries")

In this attack scenario, we study the distortion of the watermark as the input data is subjected to gradually increasing levels of data loss.

In Figure 6 (c) the analysis is performed repeatedly for single bit encoding using the "confidence-violators" encoding method outlined in Section 3.4.1. The results are then averaged over multiple runs.

The "confidence-violators" primitive set encoding proves to be resilient to a considerable amount of randomly occurring uniformly distributed surgeries (i.e. item removals by an attacker with no extra knowledge) 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).

The set size considered was 35, experiments were performed on 30 different sets of close to normally distributed data. Other parameters for the experiment include: $v_{false} = 5\%, v_{true} = 9\%, c = 88\%$. The average behavior is plotted in the graphs. Up to 25% and above data loss was tolerated easily by the tested data, before mark alteration (i.e. bit-flip) occurred.

Figure 6 (a) and (b) depict more complex scenarios in which a real multi-bit watermark is embedded into a larger data set (both uniform (a) and normal distribution (b) were considered). The input data contained 8000 tuples, subset size was 30 and the considered watermark was 12 bits long. Other parameters: $v_{false} = 15\%, v_{true} = 35\%, c = 85\%$. This set is then subjected to various degrees of data loss and the watermark distortion is observed. The encoding method again proves to be surprisingly resilient by allowing up to 45-50% data loss while still 40-45% of the watermark survives. Also, in (a), as data alteration increases, the subset (i.e. secretly selected for encoding 1-bit, see 3.3) overlap (i.e. the "resemblance" to the original content) degrades.

5.2.2 Data Alteration Attacks (Epsilon-Attack)

Presented with the watermarked data an attacker is faced with two contradictory tasks: preserving the inherent data value while removing the hidden watermark. Given no knowledge of the secret watermarking key nor of the original data the only available choice is to attempt minor random data modifications in the hope that at some point the watermark will be destroyed. Because the original data is unknown (thus also the current watermark-related distortion is unknown) it is impossible for the attacker to determine the real "minority" of changes he/she performs. In other words, because of the goal of preserving the data value, the attacker cannot afford performing significant change to the data.

In this experiment we analyse the sensitivity of our watermarking scheme to randomly occuring changes, as a direct measure for watermark resilience.

To do this, we defined a transformation that modifies a certain percentage τ of the input data within certain bounds defined by two variables ϵ and μ. We called this transformation *epsilon-attack*. *Epsilon-attacks* can model any uninformed, random change transformation – the only available attack alternative.

A normal *epsilon-attack* modifies roughly $\frac{\tau}{2}$ percent of the input tuples by multiplication with $(1+\mu+\epsilon)$ and the other $\frac{\tau}{2}$ percent by multiplication with $(1+\mu-\epsilon)$.

A "uniform altering" *epsilon-attack* modifies τ percent of the input tuples by multiplication with a uniformly distributed value in the $(1+\mu-\epsilon, 1+\mu+\epsilon)$ interval.

In Figure 7, a comparison is made between the case of uniformly distributed (i.e. values are altered randomly between 100% and 120% of their original value) and fixed alterations (i.e. values are increased by exactly 20%).

Note: In the case of fixed alterations the behavior demon-

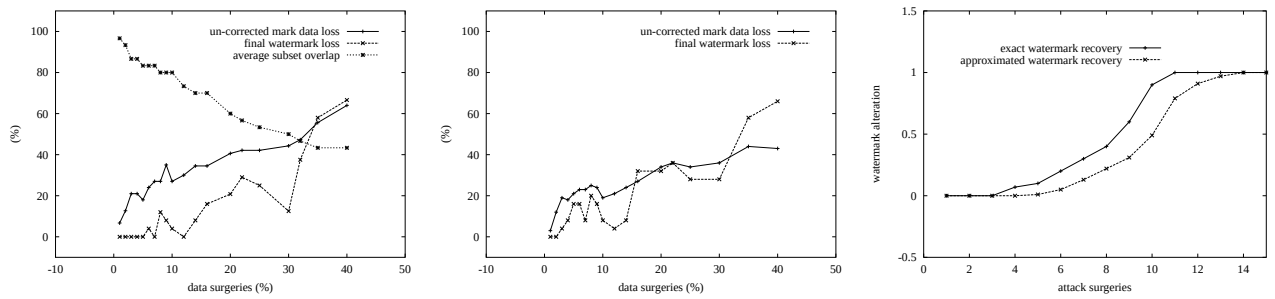

Figure 6: Experiments on resilience to data surgeries (a) uniform distribution, (b) normal distribution, (c) single subset (1-bit) encoding

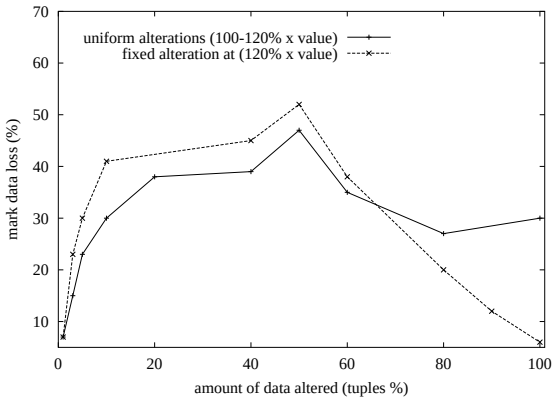

Figure 7: Epsilon-attack (non-zero average) on a normally distributed data set.

strates the effectiveness of the encoding convention: as more and more of the tuples are altered linearly, the data distribution comes increasingly closer to the original shape. For example when 100% of the data is modified consistently and linearly the mark data suffers only 6% alterations.

A peak around 50% data alterations can be observed indicating that an attack changing roughly 50% of the data might have a greater chance of success. This is also intuitively so (in the case of randomly distributed alterations) as a maximal change in distribution is expected naturally when close to half of the data set is skewed in the same "direction" (by addition or subtraction).

Parameter μ models the average of the data alteration distribution while ϵ controls its width. Naturally, a "zero-average epsilon-attack" ($\mu = 0$) is a transformation that modifies roughly $\frac{\tau}{2}$ percent of the input tuples by multiplication with $(1+\epsilon)$ and the other $\frac{\tau}{2}$ percent by multiplication with $(1-\epsilon)$.

Figure 8 presents the behaviour of our encoding algorithm to this type of attack. This is particularly intriguing as it reveals clearly a special feature of the watermarking method: since the bit-encoding convention relies on altering the actual *distribution* of the data, it survives gracefully to any distribution-preserving transformation. Randomly changing the data, while it can definitely damage the watermark (e.g. especially when altering around 50% of the data, see Figure 7), proves to be, to a certain extent, distribution-preserving.

A *zero-average epsilon-attack* is survived very well. For example, altering 80% of the input data within 20% of the original values still yields over 70% of the watermark.

Note: One could argue that, after all, if the watermark encoding relies too much on the distribution of the data, one successful attack could be the one that alters exactly this distribution. But this is not possible, as the power of the watermarking scheme lies not only in the distribution itself but also in the secrecy of the encoding subsets. In other words, *where* the bits are encoded (i.e. subsets, see Section 3) is as important as *how*. Altering global data characteristics would not only destroy probably much of the value of the data but, as shown above, achieve little in destroying the watermark.

In Figure 8 (a), as the percentage of tuples altered and the alteration factor goes up, so does the watermark distortion. Nevertheless, it turns out to be surprisingly resilient. For example, altering 100% of the data within 1% of the original values can yield a distortion as low as 3-4% in the resulting watermark. The watermark distortion increases with increasing alteration factor (b) or percentage of data (c). Figure 8 (b) presents a comparison between the curves corresponding to the alteration of 80% of the tuples versus 100% of the tuples. Naturally the curve for the higher tuples percentage appears "above". In (c) a comparison is made between curves for the alteration factor 1% or 5%. The higher alteration curve is intuitively "above". Note that the curves are not very steep: mark alteration is less dependent on the percentage of data altered than on the alteration factor (as seen in (b)).

Thus the watermarking scheme is also naturally resilient to un-informed attacks (modeled by epsilon-attack transformations).

5.2.3 Data Quality (Goodness) Metrics

We analysed the impact of data goodness preservation on the ability to provide watermark encoding bandwidth. Intuitively the more restrictive data constraints one imposes, the less available bandwidth there is as allowable data changes are directly impacted.

In the following we present two results. The first analysed goodness metric is a commonly considered one, namely upper bounds imposed on the total and local tolerable absolute change (i.e. of the new data with respect to the original).

Note: A practically identical experimental result was obtained for a related metric, the maximum allowable mean squared error.

In Figure 9, as data goodness metrics are increasingly re-

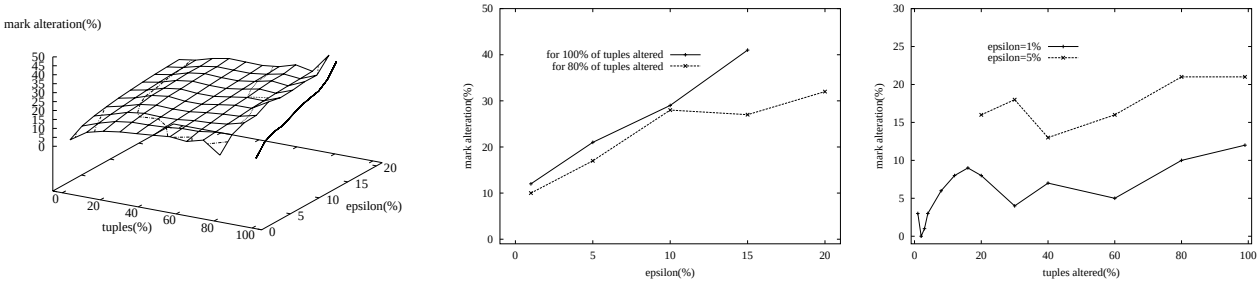

Figure 8: Epsilon-attack (zero-average) on normally distributed data.

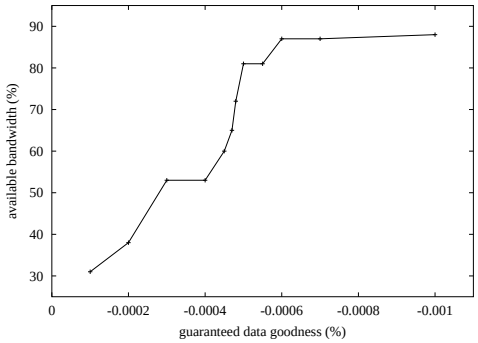

Figure 9: Impact of the Maximum Allowable Absolute Change Goodness Metric on the available watermarking bandwidth.

strictive, the available bandwidth (guaranteeing higher resilience) decreases. In the illustrated experiment, the allowed absolute change in the watermarked data (i.e. from the original) is decreased gradually (from 0.1% to 0.02%) and the decrease in available encoding bandwidth is observed (depicted as a percent of total potential bandwidth). The upper limit (approx. 90%) is inherently data-imposed and cannot be exceeded due to original data characteristics, making it effectively the maximum possibly attainable data bandwidth.

Another important experiment analyses a classification-preserving data goodness metric. Classification is extremely relevant in areas such as data mining and we envision that many of the actual deployment scenarios for our relational watermarking application will require classification preservation.

Classification preservation deals with the problem of propagation of the classes occuring in the original (input) data in the watermarked (output) version of the data. It provides thus the assurance that the watermarked version still contains most (or within a certain allowed percentage) of the original classes.

To perform the experiment, we designed and implemented a data classifier which allows for runtime fine-tuning of several important classification parameters such as the number of (synthetic) classes to be associated with a certain data set as well as the *sensitivity* of these classes. The *sensitivity* parameter can be illustrated best by example. Given a certain data set to be altered (e.g. watermarked) and an item X in this data set, the classification sensitivity models the amount of alterations X tolerates before it "jumps" out of its original class.

Note: One different perspective on *sensitivity* can be obtained by linking it to the notion of "selectivity". The more "selective" a classification is, the more *sensitive* it behaves.

The *tolerance* factor in Figure 10 represents the maximum tolerated classification distortion (i.e. percentage of class violators with respect to the original). In (a), as the classification tolerance and sensitivity go up, so does the available bandwidth. Figure 10 (b) – a section through (a) for different classification sensitivities – shows how the watermarking algorithm adapts to more and more available tolerance in data goodness. Figure 10 (c) depicts how for classification tolerance fixed at 1%, the sensitivity of the classification impacts directly the available bandwidth.

Depending on classification sensitivity (e.g. 0.01 in (b)), up to 90% of the bandwidth can become available for watermark encoding with a quite restrictive 6% classification preservation goodness.

These results confirm the adaptability of our watermarking algorithm. As classification tolerance is increased, the application adapts and makes use of an increased available bandwidth for watermark encoding. The results also show that classification preservation is compatible with our distribution-based encoding method, an important point to be made, considering the wide range of data-mining applications that could naturally benefit from watermarking ability.

5.2.4 Discussion

Watermarking outsourced content happens only once, at outsourcing time. The main purpose of watermarking in this framework is rights-protection and/or traitor tracing through fingerprinting. Thus, there is little to be gained from an ability to watermark at runtime, in the presence of updates. Moreover, because watermarking inherently alters the data, it is unreasonable to assume that a certain party would keep an altered (i.e. watermarked) copy of the data as replacement for the original (after all, how could it generate the outsourced version at the time of outsourcing ?).

Given the above, as it is to be a performed once per transaction, computation overhead is naturally not an issue (unless exorbitantly time-consuming). Nevertheless we also assessed computation times and observed an intuitively (according to the O(n) nature of the algorithm) linear behavior, directly proportional with the input data size. Given

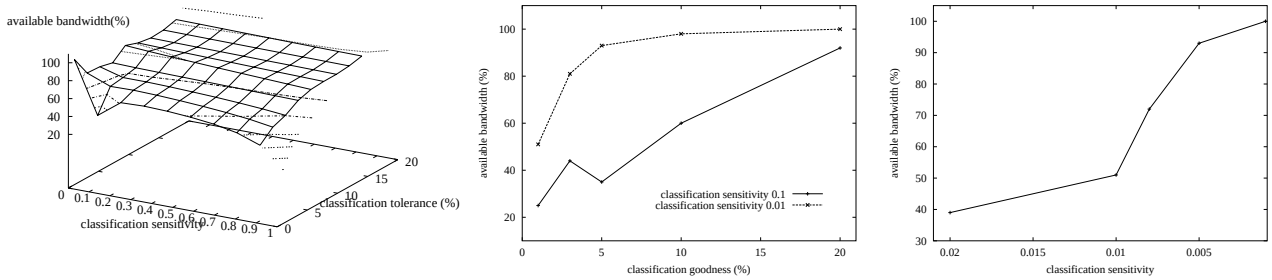

Figure 10: Impact of a classification preserving goodness metric on the available watermarking bandwidth.

the setup described above, in single-user mode, with a local database we obtained an average of around 200 tuples/second for watermark embedding, while detection turned out to be approximatively twice as fast. The results are considering the case of no data goodness metrics, thus measuring the core performance of the system (goodness metric evaluation is mostly user and data dependent). This occurs in the non-optimized, interpreted Java proof-of-concept implementation. We expect major speedups (one order of magnitude) in a real life deployment version.

5.3 Example: Watermarking the Wal-Mart Sales Database

A set of more complex experiments performed are outlined in Section 5.2. Here we present one actual application of our watermarking software. The watermark considered to be hidden 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 $c = 85\%, v_{false} = 15\%, v_{true} = 30\%$.

Notation: For formatting purposes in the following we abbreviated "StoreVisits" by "SV" and "ItemScan" by "IS".

The size of the subsets considered was roughly 70 for a total of around 2000 available encoding bits in the TotalScanAmount attribute for example.

We considered a set of usability metrics and associated queries, including the following:

(a) Intra-relational Consistency:

```
IS.UnitCostAmount x IS.ItemQuantity = IS.TotalScanAmount
```

(b) Inter-relational Consistency:

```
SV.TotalVisitAmt < SUM(IS.TotalScanAmount)
```

(c) General Attribute Constraints: MSE constraints for attribute SV.TotalVisitAmt: 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 *SV* and *IS* on the *VisitNbr* 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 ItemScanOrig, StoreVisitsOrig
WHERE ItemScanOrig.VisitNbr = StoreVisitsOrig.VisitNbr;
SELECT * AS J1 FROM ItemScanWM, StoreVisitsWM
WHERE ItemScanWM.VisitNbr = StoreVisitsWM.VisitNbr;
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 was represented by a separate usability metric plug-in, used in evaluating data usability after each atomic watermarking step (see Figure 5).

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 3.1.

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 straightforward 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.

Step (d) proved to be the most challenging, 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 [5] and JOINS were dynamically performed at run-time. This soon proved to be infeasible, and computation-intensive, often causing JDBC buffer related crashes and taking 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 *SV* will affect a yet unaccessed tuple in the JOIN result with table *IS*). This second approach, requiring definitely less space and computation power, proved to be working well.

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

[5] 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.

is to be considered as depending heavily on the optimality of the usability metrics plug-in modules implementations.

Using the attribute *IS.TotalScanAmount*, 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 (see Section 5.2 for attack and resilience experiments).

Thus, we successfully deployed wmdb.* in watermarking the Wal-Mart Sales Database. Pre-processed parts of the original data as well as their watermarked version will be available over the web (details ommitted to facilitate blind review).

6. 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 secure 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.

Recently we started investigating new, non-numeric relational data encoding domains. A proof-of-concept implementation and associated experimental results for generic data types are in the works.

Another fascinating application that we plan on tackling is in the framework of streaming data, especially of interest in scenarios such as boundary protection and massive remote sensor data processing. A model of attacks in this new domain needs to be devised and a more detailed attack-ability analysis performed.

Additionaly, a full-fledged commercial watermarking application could be derived from our proof-of-concept software and various other applications for our numeric collection marking method could be envisioned and pursued.

7. REFERENCES

[1] M. J. Atallah and Jr. S. S. Wagstaff. Watermarking with quadratic residues. In *Proc. of IS-T/SPIE Conf. on Security and Watermarking of Multimedia Contents, SPIE Vol. 3657, pp. 283–288.*, 1999.

[2] 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*. Springer Verlag, 2001.

[3] 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.

[4] Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. A flexible authorization mechanism for relational data management systems. *ACM Trans. on Information Systems*, 17(2), 1999.

[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] 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.

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

[9] 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.

[10] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD*, 1997.

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

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

[13] 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.

[14] 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.

[15] 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.

[16] Radu Sion, Mikhail Atallah, and Sunil Prabhakar. On watermarking semistructures. In *(submission for review), CERIAS TR 2001-54*, 2002.

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