Rights Protection for Relational Data

Radu Sion, Mikhail Atallah, *Fellow*, *IEEE*, and Sunil Prabhakar

Abstract—In this paper, we introduce a solution for relational database content rights protection through watermarking. Rights protection for 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 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 counterpart, for example, watermarking. While being able to handle higher level semantic constraints, such as classification preservation, our solution also addresses important attacks, such as subset selection and random and linear data changes. We introduce wmdb.*, a proof-of-concept implementation and its application to real-life data, namely, in watermarking the outsourced Wal-Mart sales data that we have available at our institute.

Index Terms—Rights protection, relational data, watermarking, information hiding.

1 Introduction

The main purpose of Digital Watermarking is to protect a certain content from unauthorized duplication and distribution by enabling provable ownership over the content. It has traditionally [6], [12], [18] 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 toward different 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.

A challenge of watermarking [21] is to insert an indelible mark in the object such that 1) 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 2) 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 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 or 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], [17] and natural language watermarking [2]. Here, we study the issue of watermarking numeric relational content. Protecting rights over outsourced relational data 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. 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 would be affected by even a minor change to any portion of it. It is therefore necessary that the intended purpose of the data to be preserved is 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], [14], [15], [16], [19], little has been done to secure proof of rights over relational data. Only one related simultaneously published effort is available for comparison [13]. Numerous fundamental differences distinguish our results from this effort. A comparative discussion can be found in Section 5.1.

In this paper, we explore the issue of securing valuable outsourced data through watermarking, enabling future

• *The authors are with the Department of Computer Sciences, 250 N. University Street, West Lafayette, Indiana, 47907-2066. E-mail: {sion, mja, sunil}@cs.purdue.edu.*

Manuscript received 5 Feb. 2003; revised 26 Aug. 2003; accepted 9 Dec. 2003. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 118247.

court proofs assessing proper rights over the content. Thus, the main contributions of the present work include:

1. a resilient watermarking method for relational data,
2. a technique for enabling user-level runtime control over properties that are to be preserved as well as the degree of change introduced,
3. a complete, user-friendly implementation for numeric relational data, and
4. the deployment of the implementation on real data, in watermarking the Wal-Mart Sales Database and the analysis thereof.

Our solution starts by receiving as user input a reference to the relational data to be rights-protected, a watermark to be embedded as a copyright proof, a secret key used to protect the embedding, and a set of data quality constraints to be preserved in the result. It then proceeds to watermark the data while continuously assessing data quality, potentially backtracking, and rolling back undesirable alterations that do not preserve data quality. Watermark embedding is composed of two main parts: In the first stage, the input data set is securely partitioned into subsets of items; the second stage then encodes one bit of the watermark into each subset. If more subsets (than watermark bits) are available, error correction is deployed to result in an increasingly resilient embedding. The algorithms introduced here prove to be resilient to important classes of attacks, including subset selection, linear data changes, and random item(s) alterations.

The paper is structured as follows: Section 2 discusses the main challenges for watermarking relational databases. Section 3 introduces an initial idea to a primitive problem (watermarking numeric collections) to be used later in the global algorithm. Section 4 constructs a solution for relational databases by building upon the primitive building block introduced earlier. Various issues, including related work, algorithm extensions, and challenges for watermarking nonnumeric relational data, are discussed in Section 5. Section 6 presents implementation details as well as experiments and evaluations of the proposed watermarking technique on real outsourced Wal-Mart warehouse data. Section 7 concludes.

2 Challenges

While research related to the issue of embedding information into a set of numbers [1] can be found (sometimes implicitly) in different frameworks, associated with various information hiding techniques (e.g., frequency domain embedding, DCT, and Wavelet watermarking [6]), relational data presents a different set of challenges and associated constraints. These challenges are novel and directly related to the specifics of the domain, namely, large sets of items organized in a relational framework, with associated semantics to be preserved. This is not the case for multimedia (mostly time-series type of) data, where semantics are associated with the data stream only at a much higher composite level. For example, in a multimegabit audio channel of news broadcast, the semantics to be protected are likely to be in the broadcast speech text rather than directly in the underlying audio stream bits; thus, a fundamentally different and broader noise band becomes available for watermark embedding, and with it different (possibly less accurate) encoding and evaluation methods. By contrast, the low noise bandwidth of major relational framework data uses (e.g., data mining) require a different approach, taking a more careful look at the actual tolerated changes on the given data.

Whereas in the multimedia case, the data quality model is usually at best fuzzy because of the relativity of any model of human perception, one solution here is to define the noise channel explicitly as part of the watermarking solution, in terms of required customer constraints to be preserved on the final data. At watermarking time, data quality can be continuously 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 percent 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, especially considering data mining issues, such as classification and JOIN results preservation (see Section 6.2).

Additionally, the watermark *encoding* method needs to feature a design 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. Our algorithm, deploying means for data distribution manipulation and encoding the actual information in distribution properties of the data rather than directly into the data itself, is best suited for its purpose, and almost optimally so. For, while allowing an adjustable degree of freedom in alteration points selection, it provides at the same time a surprisingly high level of resilience as evidenced by our extensive validation experiments (See Section 6.2).

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.

With respect to particular data uses and metrics of quality, it is of utmost importance that the watermarking process not interfere with the final data consumer requirements. This is why these requirements need to be considered as an integral part of the watermarking process, providing a feedback loop, in assessing the quality of the final result.

To the best of our knowledge, our solution is the first to recognize the importance of this essential desiderata and provide a direct algorithm for it.

In the following, we define a functionality 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 consumer-defined metrics. If the watermarked data satisfies the metrics, then the insertion of the

watermark is considered to be successful. This quality assessment mechanism is part of the marking process.

Example. One simple but 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

$$(s_i - v_i)^2 < t_i \quad \forall i = 1, \ldots, n$$

and $\sum (s_i - v_i)^2 < t_{max}$, where $\mathbb{S} = \{s_1, ..., s_n\} \subset \mathbb{R}$ is the data to be watermarked, $\mathbb{V} = \{v_1, \ldots, v_n\}$ is the result, $\mathbb{T} = \{t_1, \ldots, t_n\} \subset \mathbb{R}$, 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.

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 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. Furthermore, 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:

1. *uniqueness*—each value must be unique;
2. *scale*—the ratio between any two number before and after the change must remain the same; and
3. *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 often not enough.

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.

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 [21], constraints can be used to prevent easy detection of watermark locations. For example, a tuple with a start time later than its corresponding end time, or a customer with an age less than 12 years are very likely to be detected as resulting from watermarking.

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 (Mallory) 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. Mallory adds a set of numbers to the original set. This addition is not to significantly alter the useful (from the Mallory'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.

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 Mallory).

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?

3 Simplified Problem: Numeric Collections

This section deals with the foundations of a primitive numeric collection watermarking procedure that will be later deployed as a subroutine in the main watermarking algorithm. Section 4 evolves this building block into a complete solution in the relational framework.

Let $\mathbb{S}$ be a set of n real numbers $\mathbb{S} = \{s_1, \ldots, s_n\} \subset \mathbb{R}$. Then, the **general simplified problem** of watermarking the set $\mathbb{S}$ can be defined 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 $\mathbf{G} = \cup G_i$ *for any and all subsets* $S_i \subset \mathbf{S}$, *that hold for* $\mathbf{S}$, *then, after the transformation yields* $\mathbf{V}$, *the metrics should hold also for* $\mathbf{V}$.[1] we call $\mathbf{V}$ the "watermarked" version of $\mathbf{S}$.

Thus, $\mathbf{V} = \{v_1, \ldots, v_n\} \subset \mathbb{R}$ is the result of watermarking $\mathbf{S}$ by minor alterations to its content. 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 ith bit of w.

But, how much of a change is to be allowed to the content? 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 percent. Morever, 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 Solution Summary

A summary of the solution for the simplified problem reads as follows:

Encoding Phase: (E.1). Select a maximal number of unique, nonintersecting (see below) subsets of the original set, using a set of secrets, as described in Section 3.3. **(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.

Decoding Phase: (D.1). Using the 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). **(D.2)** For each considered subset, using the encoding convention in Section 3.3, recover the embedded bit value and reconstruct watermarks. **(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 Selecting Subsets

Watermarking a collection of data items requires the ability to "recognize" (i.e., rediscover, at detection time) *most* of the items before and after watermarking and/or a security attack. 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. For more details, see Section 3.4.2.

Our solution is based on lexicographically sorting the items in the collection, sorting occurring 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 Mallory 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. Chunk-boundaries ("subset markers") are then computed and stored for detection time (for a more in-depth discussion of subset markers see Section 4).

More formally, given a collection of items as above, $\mathbf{S} = \{s_1, \ldots, s_n\} \subset \mathbb{R}$, 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 Amplifying Watermark Power

Current watermarking algorithms draw most of their court-persuasion power [21] 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.

Generic Solution. Let $\mathbb{K} = \{k_1, \ldots, k_m\}$ be a set of m keys of n bits each. We define

$$S_i = \{s_j \in \mathbf{S} | (k_i)_{bitj} = 1\}, i = 1, \ldots, m.$$

In other words, each $S_i \subset \mathbf{S}$ is defined by selecting a subset of $\mathbf{S}$ fully determined by its corresponding key $k_i \in \mathbb{K}$.

The main purpose of this step is to amplify the power [21] of the general watermark. The next step will simply consider each S_i to be marked separately by building on a

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

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.

simple watermarking method. The result will be at least an m-bit (i.e., $i = 1, \ldots, m$) overall watermark bandwidth (unless we consider multiple embeddings and majority voting, for error correcting purposes) in which each bit is embedded/hidden in each of the marked S_i.

We presented the generic solution above for illustrative purposes. It works well for cases when exact item labeling is available and there are no concerns of attacks of the types **A2** and **A1** (i.e., subset addition, selection). The following idea takes also into account these concerns.

Actual Solution. Given a collection of items as above, $\mathbb{S} = \{s_1, \ldots, s_n\} \subset \mathbb{R}$, 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 percent of the items, this will result in each subset S_i being roughly 5 percent 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 6.2), 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. Thus, the main purpose of this step is to amplify the power [21] 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. $i = 1, \ldots, m$) overall watermark bandwidth in which each bit is embedded/hidden in each of the marked S_i.

3.4 Embedding the Watermark

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.

A desired property of an encoding method is the ability to retrieve the encoded information ("blindly") without 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.

3.4.1 Single Bit Encoding

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 Fig. 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}). Encoding the single bit (see Fig. 1), 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 positive violators ($v_c(S_i)$) is either 1) less than $v_{false} \times |S_i|$ if $b = 0$, or 2) 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 numeric attacks (see Section 6.2) 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 Fig. 3b), the errors are mostly eliminated in the result. For more details, see [20].

A decision needs to be made regarding the size of the subsets selected in the amplification step (i.e., $|S_i|$). 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. 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 on the relaxed side, S_i can be very small,

encode(bit, set, v_{false}, v_{true}, c)
- **compute** $avg(set)$, $\delta(set)$
- **compute** $v_c(set)$
- **if** $v_c(set)$ **satisfies** desired bit value **return** *true*
- **if** (bit)
  - **compute** $v_* \leftarrow v_{true} - v_c(set)$
  - **alter** v_* items close to the stddev boundary so that they become $> v_{true}$
- **else**
  - (!bit) case is similar
- **compute** $v_c(set)$
- **if** $v_c(set)$ **satisfies** desired bit value **return** *true*
- **else rollback** alterations (distribution shifted too much ?)
- **return** *false*

Fig. 1. Single Bit Encoding Algorithm (illustrative overview).

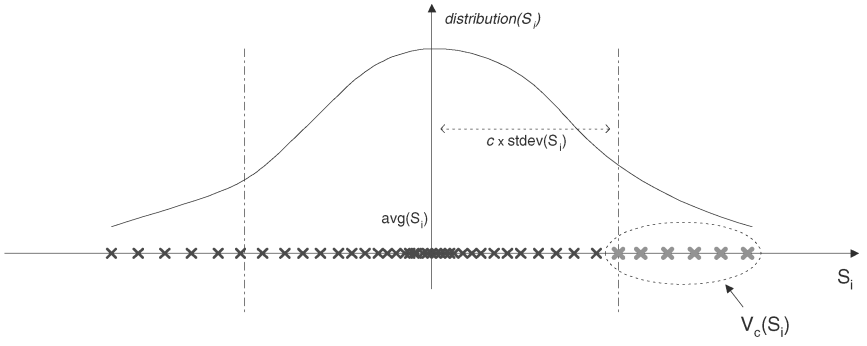
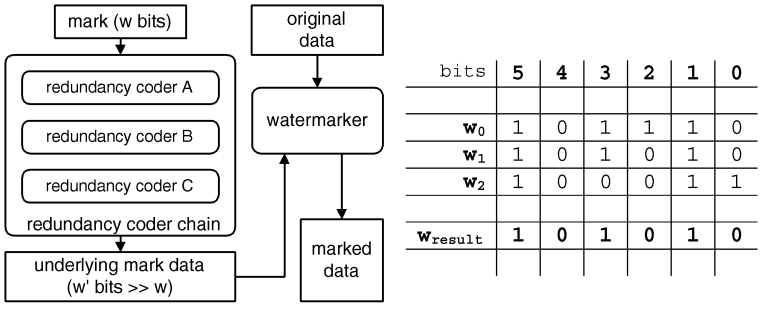


Fig. 2. Distribution of item set S_i. Encoding of the watermark bit relies on altering the size of the "positive violators" set, $v_c(S_i)$.



bits	5	4	3	2	1	0
w_0	1	0	1	1	1	0
w_1	1	0	1	0	1	0
w_2	1	0	0	0	1	1
w_{result}	**1**	**0**	**1**	**0**	**1**	**0**

Fig. 3. (a) 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 embedded. (b) Example of majority voting over three recovered watermark copies for a 6 bit sized

sometimes even 10-15 items. This enables for more encoding bandwidth overall.

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 Fig. 3a) can be employed in order to determine the most likely initial watermark bits.

Another interesting point to be made here is that (as shown in [21]) bringing the watermarked data as close as possible to the allowable distortion bounds ("usability vicinity" limits) is of definite benefit in making the usability of the watermarked data as fragile as possible to any attack. An attacker with the intent of removing/altering the watermark is now faced with the fact that any further alterations performed have an increased likelihood of making the data invalid with respect to the guaranteed usability metrics,[3] thus potentially removing its value. We integrated this idea also in our implementation. As watermark embedding progresses, a certain embedding aggres-

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

siveness factor increases, resulting in actual changes to the data to be performed more and more up to the permitted limit and not only as required.

3.4.2 Resilience Analysis

Maybe the most important resilience-revealing questions in evaluating watermarking algorithms can be formulated as follows: *What is the probability of success of Mallory aiming at destroying at least one watermark bit, as a function of the amount of data damage (i.e., number of surgeries)?* The importance of an answer to the above stems from the immediate ability to compute resilience and attack-ability bounds of the watermarking algorithm by relating the required damage for a successful attack to the maximum permissible damage levels.

Let us naturally assume a primitive 1-bit encoding method for subsets (of subset size denoted by s) that is resilient to a minimum $s \times l$ random "surgeries" (data item removals and/or alterations), $l \in \mathbb{R}$. This resilience can be guaranteed by varying the encoding parameters presented in Section 3.3. For now, we are going to assume a most general scenario by not assigning values for l. Also, let us consider an error correcting mechanism (e.g., majority voting) able to correct $e \times \frac{n}{s} \times \frac{1}{m}$ bit errors where, as above, n is the number of total items in the input set, m is the bit-size of the watermark to be embedded, and $e \in \mathbb{R}$, $e \geq 2$. In other words, e naturally models the error correcting power proportional to the ratio of total available bandwidth to watermark size. In order to keep a maximum degree of generality, we are not assigning values for e at this point.

Let $P(s, a'')$ be the average success probability (i.e., actual bit-flip) of a random, a'' sized (i.e., a'' surgeries) attack on a 1-bit encoding subset of size s. The assumption of resilience to l surgeries of the subset encoding can be thus also expressed as $P(s, x) = 0, \forall x \leq l$. First, let us compute the local (i.e., at subset level) amount of surgeries required in the case of an a-sized (i.e., a surgeries) global attack on the entire marking scheme. Because of the additional sorting and one-way hashing step (see Section 3.3), for illustrative purposes, we introduce a simplifying assumption, namely, that of a uniform distribution of all the surgeries among the individual subsets. That is, in the following, $a'' = a \times \frac{s}{n}$. The probability of an a-sized attack affecting (e.g., flipping) exactly t bits in the underlying data bandwidth, before error correction, $P_t(s, a)$ is:

$$P_t(s, a) = C^t_{\frac{n}{s}} \times P(s, a'')^t \times (1 - P(s, a''))^{n-t}.$$

Given our e-bit error correction ability, the probability that one watermark bit is altered by an a-sized attack becomes: $P_1(a) = 1 - \sum_{i=1}^{e}(P_i(s, a))$. Getting back to $P(s, a'')$ let us recollect the fact that it actually represents the average success probability (i.e., actual bit-flip) of a random, a'' sized (i.e., a'' surgeries) attack on a 1-bit encoding subset of size s. There also exists the assumption of resilience to l surgeries of the subset encoding.

Because the bit-encoding method is highly dependent on input data, its distribution, and actual values of the involved encoding parameters (e.g., c, v_{false}, and v_{true}), it becomes impossible to provide an exact, in-depth analysis of the actual value of $P(s, a'')$ for arbitrary input data. Given a certain fixed data set, it might be possible to actually exactly determine the value of $P(s, a'')$ but to no useful effect, as much of the power of the encoding lies in its ability to watermark arbitrary input.

Another method of analyzing $P(s, a'')$ could take the form of an experiment, sampling its value over a large number of potential different data inputs. We are proposing to pursue this avenue in future research. For the scope of the current analysis, given also associated space constraints, we are going to reasonably approximate $P(s, a'')$.

Remember that we introduced the assumed average l tolerated surgeries per 1-bit encoding. We know that, on average, $P(s, x) = 0, \forall x \leq l$. Let us assume that $a'' > l$. Then, we approximate $P(s, a'') = q \times \frac{a''-l}{s}, \forall a'' \in (l, s)$, where $q \in \mathbb{R}, q \geq 1$ is a input data characteristic normalization constant. Now, we can write $P_t(s, a) = C^t_{\frac{n}{s}} \times (q \times \frac{a''-l}{s}))^{n-t}$ For illustration purposes, by substituting

$$t = 1,\ n = 10,000,\ s = 50,\ a = 1,000,\ l = 4, q = 1,$$

and continuing the computation, we obtain,

$$P_1(50,\ 1,000) = \frac{1}{200} \times \frac{1}{50} \times \left(\frac{49}{50}\right)^{9,999} \simeq 1.86 \times 10^{-92} \simeq 0,$$

namely, that there is a surprisingly low probability of destroying one bit in the underlying data by a 1,000-sized attack on an input set of 10,000 where the subsets are of size 50 and subset encoding is tolerant to at least four item-surgeries. In other words, for a 10,000 tuples item set, an encoding with subsets of size 50 and an average 1-bit subset encoding tolerance to 6 percent data item losses (experimental results show much higher loss tolerance, see Section 6.2), this probability is surprisingly low, virtually zero. In Section 6.2, we present supporting experimental results.

4 The Relational Database

As discussed in Section 2, in the relational database setting, it is essential to preserve structural and semantic properties of the data. Sometimes, it is undesirable 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. For example, in certain scenarios, a practical approach would be to begin by specifying a mean square error bound on individual items. Further semantic or structural constraints that the final data consumer (user) would like to preserve can be added to these basic ones. The practically infinite nature of the set of these potential constraints that can be desired/imposed on a given data set makes it such that a different, more versatile, "data goodness" (i.e., semantically) assessment method is required. We propose a solution that handles each of these constraints that need to be preserved as an inherent component for the watermarking algorithm.

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

```
watermark(attribute, wm_key, mark_data[], plugin_handler, db_primary_key, subset_size, v_false, v_true, c)
    sorted_attribute ← sort_on_normalized_crypto_hash(wm_key,db_primary_key,wm_key)
    for (i=0; i < length(attribute)/subset_size ;i++)
        subset_bin ← next subset_size elements from sorted_attribute
        compute rollback_data
        encode(mark_data[i % mark_data.length], subset_bin, v_false, v_true, c)
        propagate changes into attribute
        if (not goodness_plugin_handler.isSatisfied(new_data,changes)) then
            rollback rollback_data
            continue
        else
            commit
            embedding_map[i] = true
            subset_boundaries[i] = subset_bin[0]
    return embedding_map, subset_boundaries
```

Fig. 4. Watermark Embedding Algorithm (version using subset markers and detection maps shown).

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, our solutions allows various forms of expression, e.g., 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 percent can be specified. Thus, we can ensure that any changes made by the watermarking algorithm do not violate the required properties.

4.1 Algorithm

The algorithm outline for watermarking relational data proceeds as follows (see Fig. 4):

1. User-defined queries and associated guaranteed query usability metrics and bounds are specified with respect to the given database.
2. User input determines a set of attributes in the database considered for watermarking, possibly all.
3. For each selected attribute, we then deploy the simplified algorithm where, 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).

4.1.1 Subset Selection

Subset selection proceeds as follows: The input data tuples are sorted (lexicographically) on a secret keyed cryptographic hash of the primary key attribute K. Based on this hash of the primary key attribute value in each tuple, compose a criteria for selecting a set of "special" tuples such that they are uniformly distributed and average a total number of $e = \frac{length(attribute)}{subset\_size}$. For example, this criteria could be $H(K, key)) \bmod e = 0$. These special tuples are going to be used as subset "markers." Each subset is defined as the elements between two adjacent markers, having on average $subset\_size$ elements. The detection phase will then rely on this construction criteria to rediscover the subset markers.

This scheme presents a drawback, namely, the fact that alterations to the data (such as attacks) have a nonzero probability of destroying one of the markers, thus making recovery of the corresponding subset impossible. This will result in a one bit loss in the underlying data (before error correction), hopefully corrected by the error correction mechanisms. For a more in depth discussion, see Section 5.

4.2 Embedding Optimizations

The embedding optimality of the solution presented above is dependent on a set of parameters such as c, $subset\_size$, v_{false}, v_{true}, etc. These parameters define a space in which each point corresponds to a different embedding. Intuitively, the solution would benefit from a fine-tuning step in which a certain optimum can be identified in this space, for example, a set of values for which the encoding bandwidth is maximized. Subject to further research is determining potential shapes defined by optimal or close to optimal points in this space. What are some criteria that could help

in determining such a point? Furthermore, another optimization, time and storage permitting, would start by training the watermarking process to be resilient to a set of transformations expected from any potential attacker. The training process would first watermark the input data only to attack it afterward. If the postattack recovered watermark is not of a satisfactory level, change the input parameters and restart the process. While this might yield considerably better embeddings, it is obviously time-consuming by nature and can probably only be applied if time constraints are not an issue.

4.3 On-the-Fly Update-Ability

In most scenarios, watermarking outsourced relational 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 seems to be 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.[4]

Nevertheless, our solution naturally supports on-the-fly watermarking, especially in the presence of updates. Let us analyze several different update scenarios:

1. updates that add fresh tuples to the already watermarked data set,
2. updates that remove tuples from the already watermarked data, and
3. updates that alter existing tuples.

In each of the cases, we assume that the watermarking mechanism runs continuously as a dormant process and is notified for each update, having full control over the watermarked version of the relational data.

Item 3 is naturally handled. As altering updates come in, the marking process lets all updates go through that are not altering the watermark. In other words, if an update is to alter a value that belongs to a certain subset $set \subset \mathbf{S}$, it first verifies if the alteration is going to alter the $v_c(set)$ value. If this is the case, the marking process automatically reconstructs the set from the original data (also updating the new value), reembeds the corresponding watermark bit and then updates the subset values in the watermarked version. Otherwise, simply let the update go through. Item 1 is similarly handled. If the added value is within a subset $set \subset \mathbf{S}$, the process verifies if this addition (which increases the set size) alters the $v_c(set)$ value. Then, proceed as above. Item 1 becomes more challenging if we consider the insertion of new tuples containing primary key values that would qualify them as subset markers. This can be handled as follows: Before inserting, a simple check is performed if indeed the new tuple could be mistaken for a marker. If this is the case, there are two options available. In option (A), the subset in which the tuple is to be inserted is split into two parts, each part being used independently as a subset in encoding one bit of the watermark. Another option (B) is to simply keep a list of such fake markers (together with the detection maps) at the detector's site, awaiting the detection process. At detection time, such markers are simply ignored. While (A) would result in producing a cleaner output, it presents the drawback of requiring a more complex management of bit embeddings. More specifically, these newly available, "'out of bound," bit "slots" (split subsets) need to be managed in such a way as to not interfere with the already embedded bits (for efficiency purposes). This can be done by keeping a mapping between each subset marker and it's corresponding underlying watermarking data bit index. This scheme allows for more flexibility as subsequent data bits are to not be embedded in subsequent subsets anymore. On the other hand, (B) is more straightforward, but does not make use of the newly available bandwidth.

Item 2 is also challenging. Its main difficulty derives from the fact that some removed tuples could be actually subset markers. If a subset marker is removed, then in the detection phase, one bit in the underlying embedding bandwidth will be destroyed. This does not necessarily result in a watermark deterioration as this could hopefully be recovered in the error correction phase. Furthermore, an improvement dealing with this scenario is the use of an embedding map, remembering exactly which subsets contain a watermark bit and which not (see Section 5.3). In the case of a marker removal, the embedding map bit for the subset corresponding to this marker can be reset to signal any future detection process that the marker was lost. Yet, another idea would be to simply add a fake marker tuple (satisfying the marker criteria) at the right point in the data.

5 Discussion

5.1 Related Work

With respect to directly related work in the relational data framework, one simultaneous published related effort [13] is available for comparison. Its main algorithm proceeds as follows: A subset of the initial data tuples are selected based on a secret criteria; for each tuple, a secret attribute and corresponding least significant (ξ) bit position are chosen. This bit position is then altered according to yet another secret criteria. The main assumption is that changes can be made to any attribute in a tuple at any least significant ξ bit positions. At watermark detection time, the process will rediscover the watermarked tuples and, for each detected accurate encoding, become more "confident" of a true-positive detection.

There are many fundamental differences between this effort and our work, including:

1. In [13], there is no provision for multibit watermarks.
2. Because the watermark is embedded in multiple attributes at the same time, vertical partitioning attacks become of concern. In a schema with a primary key and two attributes, removing one of the attributes will weaken the watermark embedding 50 percent (i.e., an amount proportional to the number of total attributes used in embedding).

4. After all, how could it generate the outsourced version at the time of outsourcing?

3. The actual bit encoding of the watermark is naturally vulnerable to an entire set of trivial attacks in the numeric domain (e.g., linear changes, see below).
4. Maybe the most important difference is the fact that our solution is built around a framework considering higher-level semantics to be preserved in the original data. Kiernan and Agrawal [13] only honor limits to fractional change in individual attribute values and there are no provisions for imposing any constraints specific to relational databases, such as preserving JOIN results, classifications, the relative values of attributes, and other requirements such as outlined in Section 2. Our solution was designed around the concept of preservation of such higher level semantic constraints. All of these and ad hoc specified SQL constraints can be preserved and honored in the result.

This is so because we believe that a sound and truly resilient watermarking method has to *start* by assessing 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 bit" constraints and require a higher-level semantic expression power such as offered by the data goodness plugins. As outlined in Section 2.1, one of the main challenges of watermarking is the ability of the encoding method to not interfere with the final data uses. This is why

5. the assumption that the least significant ξ bits in any tuple can be altered, has limited applicability and is often plain wrong.

It cannot be considered in many important applications, such as data mining, that require the preservation of classification. Consider a simple application where a relational data set is used in conjunction with a classifier clustering individuals into several categories based on age, e.g., "preschool" (0-6 years), "child" (7-13), "teenager" (14-18), "young male" (19-21), "adult" (22+). Naturally, there are likely many scenarios in which any minor alteration to the data should *not* change a person's class, for various reasons (e.g., adult movie rentals). If the rights protection method deployed is not able to handle this semantic constraint, using the watermarked result can lead to potentially illegal situations in which a minor is able to purchase alcohol and/or rent adult movies. By randomly modifying least significant bits, changing an age of 20 into 21 becomes quite likely and produces a highly undesirable result. A higher-level analogy can be constructed with the image watermarking framework, where the LSB approach to watermarking was among of the initial attempts and immediately proved its limits. LSB information hiding was immediately discarded as an effective technique for resilient watermarking [6], [18]. Modern media (e.g., sound, image) watermarking algorithms start from human perceptual models of (assuming this is the final consumer of the Works) and build on it by mapping the model into allowable tolerances for data changes. The watermark encoding needs to preserve data quality. Cross-domain experience is required to deploy the same paradigm also in the relational framework. Our approach builds on this experience.

6. In [13], resilience to true data alterations (e.g., linear changes to an arbitrary subset of the data, nonuniform scaling of all or part of the data, and epsilon-attacks) is not analyzed and the encoding method lacks fundamental provisions to resist such alterations, many of which would certainly preserve value in the result. Even minor-level epsilon-attacks, such as the ones illustrated in Section 6.2 (where our encoding survives up to 97 percent), would entirely remove the mark. Consider, for example, the case of a data mining application aiming to discover association rules from a data set with a schema composed of a primary key and two numeric attributes. By multiplying all numeric values with e.g., two, the resulting value bit strings are effectively shifted, resulting in a total loss of the original watermark. If the association rules are preserved in the result, the data is still valuable and Mallory can simply perform this attack on any and all suspected watermarked data sets and completely remove the watermark. Arguably, a majority of associations are preserved if linear data changes are performed to the underlying data consistently.

Aside from the issues outlined above, [13] also features one difference that results in desirable properties of clear benefit. Partly because of the assumption of single bit sized watermarks, the encoding is independent of tuple ordering (if the primary key attribute is preserved and unchanged). The detection process is not required to reconstruct an actual watermark string, but rather relies on the detection of a statistical improbability in the result to return the one-bit watermark; thus, data location-awareness is not necessary, a partial reason for tuple ordering independency. This independence results in two advantages of using an encoding such as in [13]: the ability

7. to detect a watermark bias in small amounts of the data and, in certain scenarios,
8. to handle multiple source data merging.[5]

There surfaces a trade off to be observed here. Handling multiple source data merges and resisting massive data loss attacks are desirable important properties. Preserving higher level semantics in the result and surviving value preserving data alteration attacks (e.g., classification preserving linear changes, random epsilon-attacks) are equally or even more important. Our solution balances the trade off between the ability to resist data loss up to 60-70 percent, major value-preserving numeric attacks as well as preserve guaranteed levels of data quality in the result. It naturally handles a certain level of data merging (similar to data addition). Nevertheless, an ability to handle increased merging levels could be obtained by designing an encoding scheme with all the advantages of our solution and independent of tuple ordering. We are currently investigating this issue.

Another interesting related research effort is to be found in [7] where the authors discuss theoretical links between query result preservation and associated allowable input data alterations.

5. This is the case if several watermarked data sources are combined, and the primary key attribute that was used in the embeddings is preserved (e.g., as the join attribute) in the result.

5.2 Trade Off between Resilience and Data Quality Preservation

Watermarking, in general, (and in the relational framework, in particular) features a trade off between the desired level of marking resilience and resistance to attacks, and the ability to preserve data quality in the result (with respect to the original). Intuitively, to one extreme, if the embedded watermark is to be very "strong" one can simply modify the entire data set such that a majority of items feature the watermark, probably also destroying the actual data value. At the other extreme, a disproportioned concern with data quality will hinder most of the watermarking alterations, resulting in a weak, possibly nonexistent embedding.

If the process of watermarking can be expressed metaphorically as a game between the watermarker and Mallory, then this is not how it is played. In this game, the watermarker and Mallory play against each other within subtle trade off rules aimed at keeping the quality of the result withing acceptable bounds. It is as if there exists an impartial referee (the data itself) moderating each and every "move" (this is why we are making this "referee" an explicit part of the marking process).

The concept of value of the resulting data, is necessarily relative and largely influenced by each semantic context it appears in. For example, while a statistical analyst would be satisfied with a set of feature summarizations (e.g., average, higher-level moments) of the numeric data set, a data mining application may need a majority of the data items, for example, to validate a classification hypothesis.

We believe it is important that watermarking-related alterations to not interfere with known customer requirements (or other advertised guaranteed properties of the result). For a watermarking solution to be sound, it has to operate inside these boundaries. This quality guarantee principle will also impact the watermarking process in itself.

5.3 Detection Maps

Storage space permitting, it might be helpful to store some information about the validity of subsets embeddings. In the detection phase, this information can be used to eliminate unusable bit encodings, in the case of invalid or unmarked subsets, thus increasing detection accuracy. At watermarking time (including on-the-fly phase), a bit for each encoding subset is maintained and updated by the marking process. This map of bits is then used in the detection process to avoid invalid subsets. If the bit is set, the subset is signaled as being valid; otherwise, the detection process ignores the corresponding subset.

In the case of small to medium data sizes, and arguably for larger data sizes too, the detection map is not hard to store and "remember" (at the detector's site) for detection time. An example of an embedding map used was only 2,000 bits long, barely 1.5KBytes of data, certainly small enough to be stored together with the embedding key and additional watermarking parameters awaiting detection time. As a general rule, the detection map will be (naturally) $\frac{length(attribute)}{subset\_size}$ bits long.

5.4 Subset Markers

The use of subset markers (both stored and criteria-based) increases the ability to accurately reconstruct the underlying partitions corresponding to the individual bit embeddings.

While experimental results show the real-life resilience of the embedding method, from a more theoretical viewpoint we would like to ask: 1) What is the likelihood of Mallory removing a subset marker? 2) What is the impact of this likelihood on the final resulting watermark?

Let $P_m(s, a)$ be the probability that an a-sized removal attack eliminates a subset marker in the resulting data. Naturally, for each subset (s items), there exists one subset marker. Thus, each individual data removal has a probability of $\frac{1}{s}$ of succeeding in removing an actual subset marker; for $a < s$, we have $P_m(s, a) = \frac{a}{s}$. In other words, for each "subset-worth" of data (s tuples) removed from the data, on average, one subset marker is destroyed. An eliminated subset marker directly results in the inability to detect the corresponding subset (unless the adjacent markers are preserved in which case something can still be done), thus resulting in a lost bit in the underlying embedding (before error correction). This result is intuitive and encouraging because it shows a direct and linear behavior between data value degradation (tuples removal) and subset marker loss. Mallory has to remove at least this much data to be able to eliminate a marker. In the experiments, we used an implementation deploying markers; thus, the results include this probability of marker loss.

Another interesting issue related to using subset markers is that apparently the embedding method can get to a point where it is difficult to find subset marker selection criteria that would yield evenly sized sets in the data partitioning phase. This can happen when the selection criteria do not uniformly spread the selected markers over the input tuples. In the case of $H(K, key) \bmod e = 0$, this is the case iff. the $H(K, key)$ values are not uniformly distributed, which, in turn, can happen (because of the one-way randomized nature of the cryptographic hash) only if the K values are identical or partitioned in large chunks of values. But, the K values are by nature (primary key attribute) different for each data tuple; thus, the marker selection is naturally uniformly distributed and results in almost identical subset sizes.

5.5 Primary Key Dependence

In Section 3.3, we presented a subset selection idea for the case of the simplified problem. The solution was performing a lexicographical, secret, one-way sort on the most significant bit (MSB) space in the considered items, in order to then enable the selection of subsets. While this idea provides a certain level of self-containment and is well suited for the problem it was formulated in, where there exist no external aids in defining an ordering on the data, in the relational data framework, we chose to investigate the use of the primary key as such an aide.

Thus, the solution as such features a certain dependency on the primary database key. If attacks on the primary key occur, there are two potential options: 1) the use of the initial MSB space sort idea and/or 2) an initial normalizing step that brings the primary key within a predefined range in which the subset selection step in Section 4.1 (i.e.,

```
detect(attribute, wm_key, db_primary_key, subset_size, v_false, v_true, c, embedding_map[], subset_boundaries[])
    sorted_attribute ← sort_on_normalized_crypto_hash(wm_key,db_primary_key,wm_key)
    read_pipe ← null
    do { tuple ← next_tuple(sorted_attribute) }
    until (exists idx such that (subset_boundaries[idx] == tuple))
    current_subset ← idx
    while (not(sorted_attribute.empty())) do
        do {
            tuple ← next_tuple(sorted_attribute)
            read_pipe = read_pipe.append(tuple)
        } until (exists idx such that (subset_boundaries[idx] == tuple))
        subset_bin ← (at most subset_size elements from read_pipe, not including last read element)
        read_pipe.remove_all_remaining_elements_but_last_read()
        if (embedding_map[current_subset]) then
            mark_data[current_subset] ← decode (subset_bin, v_false, v_true, confidence)
            if (mark_data[current_subset] != DECODING_ERROR)
                    then detection_map[current_subset] ← true
        current_subset ← idx
    return mark_data, detection_map
```

Fig. 5. Watermark detection algorithm (version using subset markers and detection maps shown).

$H(K', key))$ mod $e = 0$) is performed on a MSB portion of the primary key K, i.e., $K' = MSB(K)$. This is to be subject to further investigation, hopefully resulting in primary key independence.

6 Experimental Results

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

6.1 Implementation: wmdb.*

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 deployed and queried at runtime without recompilation and/or software restart.[6]

Once usability metrics are defined and all other parameters are in place, the watermarking module (see Fig. 6) initiates the process of watermarking. An undo/rollback log is kept for each atomic step performed (i.e., 1-bit encoding) until data usability is assessed and confirmed (by querying the currently active usability plugins). This allows for rollbacks in the case when data quality 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 various other encoding specific parameters (see Fig. 5). 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.

6.2 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

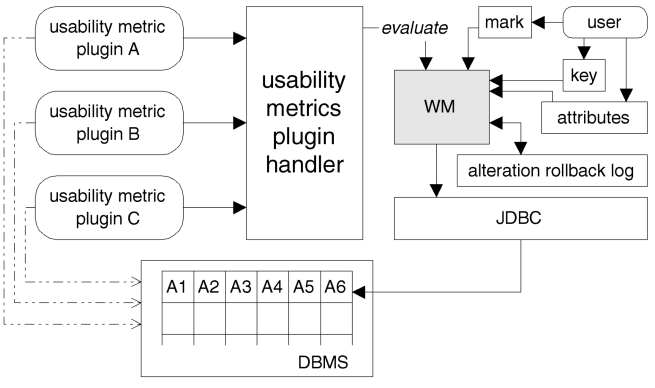


Fig. 6. The **wmdb.*** package. Overview.

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

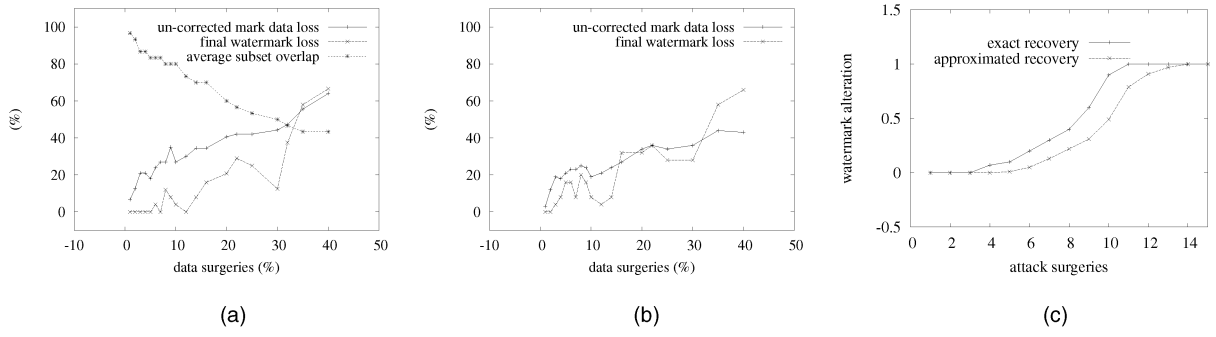


Fig. 7. Resilience to data surgeries: (a) uniform distribution, (b) normal distribution, and (c) single subset (1-bit) encoding.

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, (formerly) hosted on a NCR Teradata machine, one 1.8GHz CPU Linux box with Sun JDK 1.4 and 384MB RAM. The amount of data available is enormous. For example, the *ItemScan* relation contains more than 840 million tuples.

For testing purposes, we deployed our algorithm on a randomly selected subset of the original data (e.g., just a maximum of 141,075 tuples for relation *UnivClassTables.StoreVisits*).

We 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 the setup described above, in single-user mode, with a local database we obtained an average of around 350-400 tuples/second for watermark embedding, while detection turned out to be approximatively twice as fast. This occurs in the nonoptimized, interpreted Java proof-of-concept implementation. We expect major speedups (orders of magnitude) in a real-life deployment version.

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 and data classification preservation).

6.2.1 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 Fig. 7c, 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 Mallory, 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 percent and above data loss was tolerated easily by the tested data, before mark alteration (i.e., bit-flip) occurred.

Figs. 7a and 7b depict more complex scenarios in which a real multibit watermark is embedded into a larger data set (both (a) uniform and (b) normal distributions in Fig. 7 were considered). The input data contained 8,000 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 percent data loss while still 40-45 percent of the watermark survives. Also, in Fig. 7a, as data alteration increases, the subset (i.e., secretly selected for encoding 1-bit, see Section 3.3) overlap (i.e., the "resemblance" to the original content, the number of same elements in resulting subsets) degrades.

Note on Data Dependency in Figures. Some of the figures presented in this section feature "spikes." This is a result of the adaptive data-dependent nature of the encoding. Different input data reacts differently to data surgeries (for example) and feature slightly varying behavior at distinct points. Averaging over multiple inputs provides a solution for this issue. Nevertheless, we believe that, while it might soften the spikes, it would also (arguably) tone down distinct features for a given data set, features that interrelate figures. Instead of focusing on local variations, the figures should be interpreted as an illustrative sample of the global governing trends.

6.2.2 Data Alteration Attacks (Epsilon-Attack)

Presented with the watermarked data Mallory is faced with two contradictory tasks: preserving the inherent value of the data while, at the same time, 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 Mallory to determine the real "minority" of changes he/she performs.

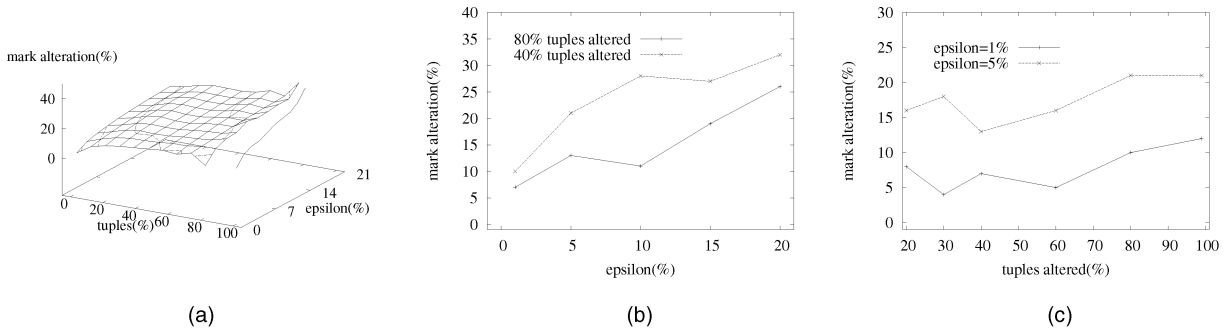


Fig. 8. Epsilon-attack (zero-average) on normally distributed data.

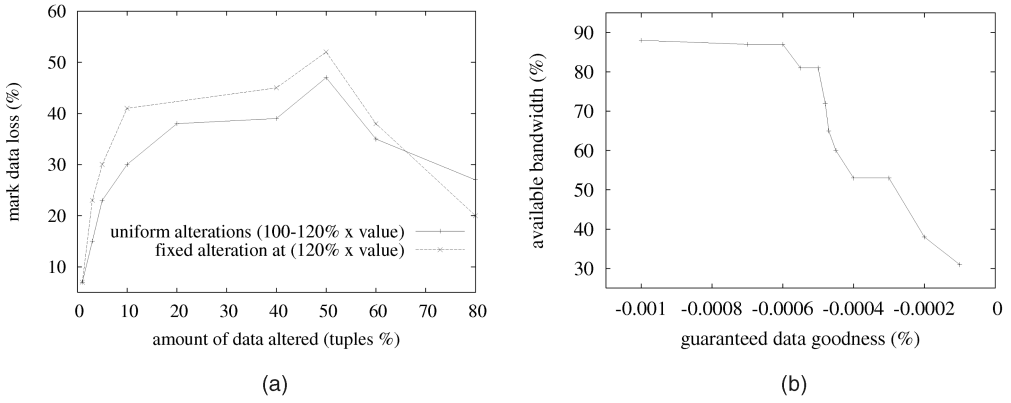


Fig. 9. (a) Epsilon-attack (nonzero average) on a normally distributed data set. (b) Impact of guaranteeing a maximum allowable absolute change on the available watermarking bandwidth.

In other words, because of the goal of preserving the data value, Mallory cannot afford to perform significant change to the data.

In this experiment, we analyze the sensitivity of our watermarking scheme to randomly occurring changes, as a direct measure for watermark resilience. To do this, we define a transformation that modifies a 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 alteration—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 Fig. 9a, a comparison is made between the case of uniformly distributed (i.e., values are altered randomly between 100 and 120 percent of their original value) and fixed alterations (i.e., values are increased by exactly 20 percent). In the case of fixed alterations, the behavior demonstrates 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 percent of the data is modified consistently and linearly, the mark data suffers only 6 percent alterations. A peak around 50 percent data alterations can be observed indicating that an attack changing roughly 50 percent 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 $(1-\epsilon)$.

Fig. 8 presents the behavior of our encoding algorithm to this type of attack. This is particularly intriguing as it clearly reveals 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 percent of the data, see Fig. 9a), proves to be, to a certain extent, distribution-preserving. A *zero-average epsilon-attack* is survived very well. For example, altering 80 percent of the input data within 20 percent of the original values still yields over 70 percent 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)

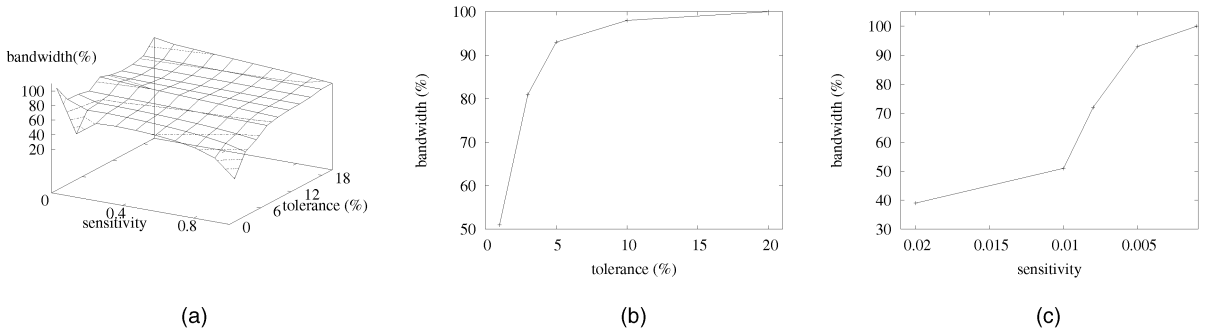


Fig. 10. Impact of a classification preservation on the available watermarking bandwidth.

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 Fig. 8a, 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 percent of the data within 1 percent of the original values can yield a distortion as low as 5-6 percent in the resulting watermark. The watermark distortion increases with increasing (b) alteration factor or (c) percentage of data in Fig. 8. Fig. 8b presents a comparison between the curves corresponding to the alteration of 40 percent of the tuples versus 80 percent of the tuples. Naturally, the curve for the higher tuples percentage appears "above." In Fig. 7c, a comparison is made between curves for the alteration factor 1 or 5 percent. The higher alteration curve is intuitively "above." Note that the curves are slightly increasing but not very steep: Mark alteration is less dependent on the percentage of data altered than on the alteration factor (as seen in Fig. 8b). Thus, the watermarking scheme proves a natural resilience to uninformed attacks (modeled by epsilon-attack transformations).

6.2.3 Data Quality (Goodness) Metrics

Here, we analyze the impact of data goodness preservation on the available 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 analyzed 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: An identical experimental result was obtained for a related metric, the maximum allowable mean squared error.

In Fig. 9b, as data goodness metrics are increasingly restrictive, 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 percent) and the decrease in available encoding bandwidth is observed (depicted as a percent of total potential bandwidth). The upper limit (approximately 90 percent) is inherently data imposed and cannot be exceeded due to original data characteristics, making it effectively the maximum attainable bandwidth.

Another important experiment analyzes 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 occurring in the original (input) data in the watermarked (output) version of the data. Thus, it provides 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 classification selectivity. The more selective a classification is, the more *sensitive* its behavior.

The *tolerance* factor in Fig. 10 represents the maximum tolerated classification distortion (i.e., percentage of class violators with respect to the original). In Fig. 10a, as the classification tolerance and sensitivity go up, so does the available bandwidth. Fig. 10b shows how the watermarking algorithm adapts to an increasing data goodness tolerance (classification sensitivity 0.01). Fig. 10c depicts how for classification tolerance fixed at 1 percent, the sensitivity of the classification impacts directly the available bandwidth.

Depending on classification sensitivity (e.g., 0.01 in Fig. 10b), up to 90 percent of the underlying bandwidth can become available for watermark encoding with a restrictive 6 percent 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. This also shows 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.

7 Conclusions and Future Research

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

1. designed a solution to a simplified version of our problem, namely, watermarking a numeric collection by
   a. defining a new suitable mark encoding method for numeric sets and
   b. building an algorithmic secure mapping (i.e., mark amplification) from a simple encoding method to a more complex watermarking algorithm, and
2. applied the concept to numeric relational databases.

We thus provided 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. In upcoming research, we are investigating new, nonnumeric encoding domains. Furthermore, a model of attacks in this new domain needs to be devised and a more detailed attack analysis performed. A full-fledged commercial watermarking application could be derived from our proof-of-concept software.

Acknowledgments

Portions of this work were supported by Grants EIA-9903545, IIS-0325345, IIS-0219560, IIS-0312357, IIS-9985019, IIS-9972883, and IIS-0242421 from the US National Science Foundation, Contract N00014-02-1-0364 from the US Office of Naval Research, by sponsors of the Center for Education and Research in Information Assurance and Security, and by Purdue Discovery Park's e-enterprise Center.

References

[1] M.J. Atallah and S.S. Wagstaff Jr., "Watermarking with Quadratic Residues," *Proc. IS-T/SPIE Conf. Security and Watermarking of Multimedia Contents, SPIE,* vol. 3657, pp. 283-288, 1999.

[2] M.J. Atallah, V. Raskin, C.F. Hempelmann, M. Karahan, R. Sion, K.E. Triezenberg, and U. Topkara, "Natural Language Watermarking and Tamperproofing," *Proc. Fifth Int'l Information Hiding Workshop,* 2002.

[3] E. Bertino, M. Braun, S. Castano, E. Ferrari, and M. Mesiti, "Author-x: A Java-Based System for XML Data Protection," *Proc. IFIP Workshop Database Security,* pp. 15-26, 2000.

[4] E. Bertino, S. Jajodia, and P. Samarati, "A Flexible Authorization Mechanism for Relational Data Management Systems," *ACM Trans. Information Systems,* vol. 17, no. 2, 1999.

[5] C. Collberg and C. Thomborson, "On the Limits of Software Watermarking," Technical Report #164, Dept. of Computer Science, The Univ. of Auckland, http://citeseer.ist.psu.edu/collberg98limits.html, Aug. 1998.

[6] I. Cox, J. Bloom, and M. Miller, "Digital Watermarking," *Digital Watermarking,* Morgan Kaufmann, 2001.

[7] D. Gross-Amblard, "Query-Preserving Watermarking of Relational Databases and XML Documents," *Proc. 19th ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems,* 2003.

[8] J. Hale, J. Threet, and S. Shenoi, "A Framework for High Assurance Security of Distributed Objects," *IFIP Conference Proceedings,* 1997.

[9] E. Hildebrandt and G. Saake, "User Authentication in Multidatabase Systems," *Proc. Ninth Int'l Workshop Database and Expert Systems Applications,* R.R. Wagner, ed., pp. 281-286, 1998.

[10] S. Jajodia, P. Samarati, and V.S. Subrahmanian, "A Logical Language for Expressing Authorizations," *Proc. IEEE Symp. Security and Privacy,* pp. 31-42, 1997.

[11] S. Jajodia, P. Samarati, V.S. Subrahmanian, and E. Bertino, "A Unified Framework for Enforcing Multiple Access Control Policies," *Proc. SIGMOD,* 1997.

[12] *Information Hiding Techniques for Steganography and Digital Watermarking,* S. Katzenbeisser and F. Petitcolas, eds. Artech House, 2001.

[13] J. Kiernan and R. Agrawal, "Watermarking Relational Databases," *Proc. 28th Int'l Conf. Very Large Databases VLDB,* 2002.

[14] N Li, J. Feigenbaum, and B.N. Grosof, "A Logic-Based Knowledge Representation for Authorization with Delegation," *PCSFW: Proc. 12th Computer Security Foundations Workshop,* 1999.

[15] M. Nyanchama and S.L. Osborn, "Access Rights Administration in Role-Based Security Systems,," *Proc. IFIP Workshop Database Security,* pp. 37-56, 1994.

[16] S.L. Osborn, "Database Security Integration Using Role-Based Access Control," *Proc. IFIP Workshop Database Security,* pp. 245-258, 2000.

[17] J. Palsberg, S. Krishnaswamy, M. Kwon, D. Ma, Q. Shao, and Y. Zhang, "Experience with Software Watermarking," *Proc. ACSAC, 16th Ann. Computer Security Applications Conf.,* pp. 308-316, 2000.

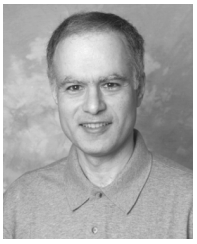
[18] F.A.P. Petitcolas, R.J. Anderson, and M.G. Kuhn, "Attacks on Copyright Marking Systems," *Proc. Information Hiding: Second Int'l Workshop,* D. Aucsmith, ed., pp. 218-238, 1998.

[19] D. Rasikan, S.H. Son, and R. Mukkamala, "Supporting Security Requirements in Multilevel Real-Time Databases," tr-95-21, 1995.

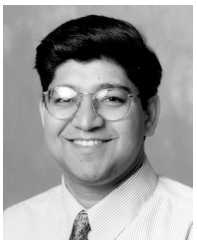
[20] R. Sion, M. Atallah, and S. Prabhakar, "On Watermarking Numeric Sets," *Proc. Int'l Workshop Digital Watermaking,* 2002.

[21] R. Sion, M. Atallah, and S. Prabhakar, "Power: Metrics for Evaluating Watermarking Algorithms," *Proc. IEEE Int'l Conf. Information Technology: Coding and Computing ,* 2002.

Radu Sion received the BSc degree from Politehnica University of Bucharest in 1998 and the MSc degree Politehnica University of Bucharest, Romania, in 1999, and Purdue University in 2000. He is a PhD student in computer sciences at Purdue University. His main interests lie in information security with applications in relational databases, rights protection, and trust and purpose management. Recently, he has been also working on various topics in the areas of privacy preserving computing, hippocratic databases, and Web service workflows. His PhD dissertation analyzes rights protection for generalized data objects and introduces novel ideas in the area of rights protection for streams, structures, and relational data with applications ranging from database watermarking to stream fingerprinting.



Mikhail ("Mike") Atallah received the PhD degree from the Johns Hopkins University in 1982, and has been on the faculty of the Purdue University in the Computer Science Department since then. His current research interests are in information security (in particular, software security, secure protocols, and watermarking). A fellow of the IEEE, he has served on the editorial boards of *SIAM Journal on Computing*, the *IEEE Transactions on Computers*, the *Journal of Parallel and Distributed Computing*, *Information Processing Letters*, *Computational Geometry: Theory and Applications*, the *International Journal of Computational Geometry and Applications*, *Parallel Processing Letters*, and *Methods of Logic in Computer Science*. He was guest editor for a special issue of algorithmica on computational geometry, has served as editor of the *Handbook of Parallel and Distributed Computing* (McGraw-Hill), as editorial advisor for the *Handbook of Computer Science and Engineering*, (CRC Press), and as editor-in-chief for *Handbook of Algorithms and Theory of Computation* (CRC Press).



Sunil Prabhakar's research focuses on performance and security issues in large-scale, modern database applications such as multimedia, moving-object, and sensor databases. The efficient execution of I/O is a critical problem for these applications. The scope of this research ranges from main memory to disks and tertiary storage devices. Sensor and moving object applications are also faced with the need to process large volumes of data in an online manner. His current research effort addresses efficient continuous query evaluation and novel techniques for managing the inherent lack of accuracy for these applications. Dr. Prabhakar's interest also lies in the design and development of digital watermarking techniques for structured (e.g., relational databases) and semistructured (e.g., XML) data. Prior to joining Purdue, Dr. Prabhakar held a position with Tata Unisys Ltd. from 1990 to 1994.