On Watermarking Numeric Sets CERIAS TR 2001-60 *

Radu Sion, Mikhail Atallah, Sunil Prabhakar

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

Abstract. *Digital Watermarking*, [3] [4] [5] [6] [7] [8] [9] [11] [12] [16] [17] [18] can be summarized the technique of embedding un-detectable (un-perceivable) hidden information into data objects (i.e. images, audio, video, text) mainly to protect the data from unauthorized duplication and distribution by enabling provable rights over the content.
In the present paper we address the issue of data security through resilient information hiding, in the framework of numeric data. We're looking into the fundamental problem of watermarking numeric collections and propose resilient algorithms. To the best of our knowledge there is no work specifically addressing the problem of watermarking this type of data. The wide area of applicability of the problem ranging from numeric database content to stock market analysis data, makes it especially intriguing when considering a generic solution and particularities of its various applications. Given a range of associated numeric constraints and assumptions we provide a solution and analyze associated attacks. Our solution is resilient to a multitude of attacks, including data re-sorting, subset selection (up to 30% data loss tolerance), linear data changes etc. Finally we present and discuss a proof-of-concept implementation of our algorithm.

1 Introduction

Extensive research has been conducted on the problem of watermarking multimedia data (images, video and audio), however there is relatively little work on watermarking other types of data. More recently, the focus of watermarking for digital rights protection is shifting toward data types such as text, software, and algorithms. Since these data types are designed for machine ingestion and have well defined semantics (as compared to those of images, video, or music), the identification of the available "bandwidth" for watermarking is as important

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

a challenge as the algorithms for inserting the watermarks themselves. Existing research has addressed the problems of software watermarking [2] [10] [15] and natural language watermarking [1]. Here we study the issue of watermarking numeric collections and propose algorithms for resilient watermarking.

Thus, in this paper we explore how Alice makes sure she can safely distribute a valuable numeric collection to Tim Mallory and others. Of course she also wants to be able to prove later on to Jared that the data in Tim's possession (possibly maliciously modified) is her creation and/or assert some other rights over it.

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

The paper is structured as follows. Section 2 presents the considered framework issues and main associated challenges. Section 3 presents our solution. Section 4 introduces a practical example illustrating our algorithm. Section 5 discusses conclusions and defines our current ongoing efforts as part of a broader frame of future envisioned research.

2 The Problem.

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

2.1 Data Usability

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

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

In other words we define the allowable distortion bounds for the given input data ($\mathbb{S}$) in terms of "usability" metrics (see "usability" in [14]) which are given at watermarking time and are defined by the actual purpose (see "usability domain" in [14]) of the data.

Example.

One simple but extremely relevant example is the *maximum allowable mean squared error* case, in which the usability metrics are defined in terms of mean squared error tolerances as follows:

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

where $\mathbb{T} = \{t_1, ..., t_n\} \subset \mathbb{R}$ and $t_{max} \in \mathbb{R}$ are defining the guaranteed error bounds at data distribution time. In other words $\mathbb{T}$ defines the individual elements allowable distortions in terms of mean squared error (MSE) and t_{max} the overall permissible MSE.

Many other semantic and numeric constraints can be imagined with respect to a given application and data set. This paper does not focus on any particular complex constraint case but rather on the overall concept. Analysis of different types of constraints is to make the object of a distinct, future research effort.

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

One interesting issue here is identifying the original set items *after* watermarking. That is, how do we "recognize" an item after it has been changed slightly. This is the subject of Section 3.2 where we devise a scheme for item labeling that works well enough to be suitable for our watermarking purposes.

2.2 Attacks

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

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

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

A3. Subset Alteration. Altering a subset of the items in the original data set such that there is still value associated with the resulting set. A special case needs to be outlined here, namely (A3.a) a linear transformation performed

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

uniformly to all of the items. This is of particular interest as such a transformation preserves many data-mining related properties of the data, while actually altering it considerably, making it necessary to provide resilience against it.

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

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

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

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

Because it adds new data to the set, defeating **A2** seems to be the most difficult task, because it implies the discovery of the possible data usability domains [14] for the attacker. That is, we have to be able to pre-determine what the main uses (for the attacker) for the given data set/type are.

3 A Solution.

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

3.1 Overview

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

Encoding

Step E.1. Select a maximal number of unique, non-intersecting (see below) subsets of the original set, as described in Section 3.3.

Step E.2. For each considered subset, (E.2.1) embed a watermark bit into it using the encoding convention in Section 3.4 and (E.2.2) check for data usability

bounds. If usability bounds exceeded, (E.2.3) retry different encoding parameter variations or, if still no success, (E.2.3a) try to mark the subset as invalid (i.e. see encoding convention in Section 3.4), 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 watermarking patterns embedded in the entire data collection.

Many other implementations of this solutions are possible. For example checking for data usability could be done at an even more atomic level, such as inside the bit-encoding procedure, selection of intersecting subsets could be allowed, etc.

Decoding

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

Step D.2. For each considered subset, using the encoding convention in Section 3.4, recover the embedded bit value and re-construct watermarks.

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

In the following we introduce more details on the actual building blocks of the overview above.

3.2 Item labeling for set elements

Watermarking a data collection requires the ability to "recognize" *most* of the collection items before and after watermarking and/or a security attack. This is especially important as a response to attacks of type **A4**. In other words if an item was accessed/modified before watermarking, e.g. being identified with a certain label L, then hopefully at watermark detection time the same item is still identify-able with the same label L or a known mapping to the new label.

Bringing this a little bit further, making some less restrictive assumptions, we would like to be able to identify *a majority of the initial elements of a subset* after watermarking and/or attacks. As we will see, "missing" a small number of items is not making it much worse as the marking method is resilient to that.

While research efforts of the authors include work in this area (see "tolerant canonical labeling" in [13]) we are going to present a simplified solution here, tailored to the particularities of the current problem.

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

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

the items. The secret one-way hashing ensures that an attacker cannot possibly determine what the actual item ordering is. Then, in the next step (see Section 3.3), subset "chunks" of the items are selected based on this secret ordering. This defeats **A4** as well as any attempts to actually deploy statistical analysis to determine the secret subsets.

More formally, given a collection of items as above, $\mathbb{S} = \{s_1, ..., s_n\} \subset \mathbb{R}$, and a secret "sorting key" k_s, we induce a secret ordering on it by sorting according to a cryptographic keyed hash of the most significant bits of the normalized items. Thus we have $index(s_i) = H(k_s, MSB(NORM(s_i)), k_s)$.

The MSB space here is assumed to be a domain where minor changes on the collection items (changes that still satisfy the given required usability metrics) have a minimal impact on the MSB labels. This is true in most cases (as usually the given 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.

3.3 Algorithm

Current watermarking algorithms draw most of their court-persuasion power [14] from a secret that controlled watermark embedding (i.e. watermarking key). Much of the attack immunity associated with an watermarking algorithm is based on this key and it's level of secrecy.

Given a weak partial marking technique (e.g. (re)setting a bit), a strong marking method can be derived by repeatedly applying the weak technique in a keyed fashion on different (secretly selected) parts of the object to be watermarked.

In this section we present the two main steps of our watermarking scheme that come in after labeling the data collection items (that is, we assume the ability to identify most of the items here).

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

Step One: Power Amplification

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

Generic Solution: We define $S_i = \{s_j \in \mathbb{S} | (k_i)_{bit\,j} = 1\}, i = 1, ..., m$. In other words each $S_i \subset \mathbb{S}$ is defined by selecting a subset of $\mathbb{S}$ fully determined by it's corresponding key $k_i \in \mathbb{K}$ (see Figure 1(a)).

The main purpose of this step is to amplify the power [14] 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 at least a m-bit (i.e. $i = 1, ..., m$) overall watermark bandwidth (unless multiple embeddings and majority voting are considered, see Section 3.5) in which each bit is embedded/hidden in each of the marked S_i.

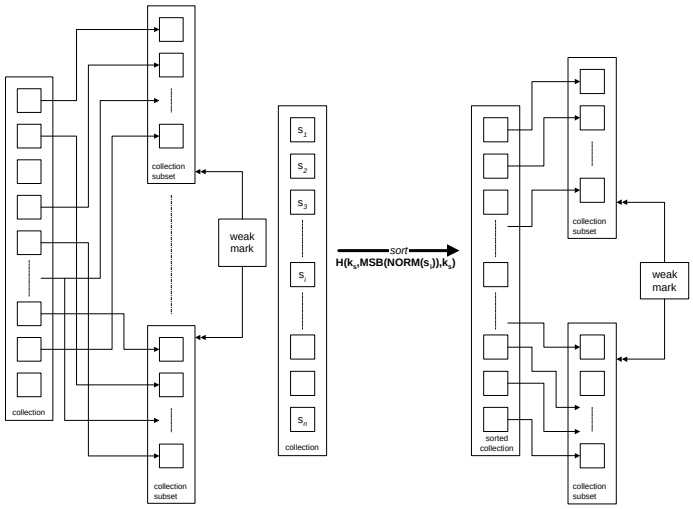

Fig. 1. Primitive Mark Power Amplification. (a) Keyed selection of subsets. This assumes the ability to uniquely and consistently identify items before and after watermarking and/or attacks. (b) Subset selection after sorting on keyed hash of the most significant bits (MSB) of the normalized data items. This enables recovery after various attacks, including re-shuffling/sorting and linear changes.

Note: In building the subsets S_i we do not consider elements s_j which are subject to very restrictive usability metrics (e.g. with corresponding $t_j = 0$, i.e. no available encoding bandwidth). If any of the considered keys are selecting one of those unalterable elements, we simply generate another key instead.

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: Considering the ability to induce a secret ordering on the collection items as presented in Section 3.2, we define an alternate method to select subsets as required by power amplification. Let k_s the sorting key as above. After inducing the secret ordering by sorting the collection $\mathbb{S} = \{s_1, ..., s_n\} \subset \mathbb{R}$ on $index(s_i) = H(k_s, MSB(NORM(s_i)), k_s)$, we build the subsets S_i as "chunks" of items from the collection, a "chunk" being a set of adjacent items in the sorted version of the collection (see Figure 1(b)).

This increases the ability to defeat different types of attacks including "cut" and/or "add" attacks (e.g. **A1** , **A2**), by "dispersing" their effect throughout the data, as a result of the secret ordering. Thus, if an attack removes 5% of the items this will result in each subset S_i being 5% smaller. If S_i is small enough and/or if the primitive watermarking method used to encode parts of the watermark (e.g. 1 bit) in S_i is made resilient to these kind of (minor) transformations (See

Figure 4) then the probability of survival of most of the embedded watermarks is higher.

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 guarantees a certain adaptability of our subset selection scheme, assuring later on the retrieval of the watermark even from, say, half of the original data.

For another discussion on the size of the subsets S_i see Section 3.5. For now it just suffices to say that all the subsets are equally sized and this size is to be considered a part of the overall watermark encoding secret.

Depending on the particularities of the desired result, as well as the data's usability domain, different keyed subset selection methods may be devised, more appropriate to the envisioned types of attacks.

Step Two: Watermarking

Once each of the to-be-watermarked secret (keyed) sets S_i is determined, the problem reduces itself to finding a reasonable, not-very-weak (i.e. better than "coin-flip", random occurrence) algorithm for watermarking a medium-small sized set of numbers. The problem here is to find a direct watermarking method that allows reliable encoding of a single bit value in this set of numbers.

3.4 Encoding. Primitive Watermarks.

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

One desired property of an encoding method is the ability to retrieve the encoded information without having the original data. This can be important especially in the case of very large dynamic databases (e.g. 4-5 TBytes of data) where data-mining portions were outsourced at various points in time. It is unreasonable to assume the requirement to store each outsourced copy of the original data. Our method satisfies this desiderata.

Confidence Violators.

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

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

Definition: Let $avg(S_i) = \frac{\sum x_j}{|S_i|}$, $\delta(S_i) = \sqrt{\frac{\sum (avg(S_i) - x_j)^2}{|S_i|}} \forall x_j \in S_i$. Given S_i and the real number $c \in (0,1)$ as above, we define $v_c(S_i)$ to be the *number of*

items of S_i that are greater than $avg(S_i) + c \times \delta(S_i)$. We call $v_c(S_i)$ the number of positive "violators" of the c confidence over S_i, see Figure 2.

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

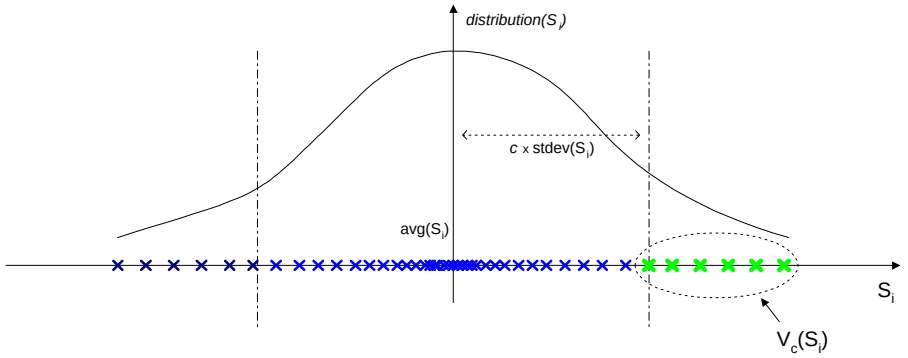

Fig. 2. Distribution of item set S_i. Encoding of the watermark bit relies on altering the size of the "positive violators" set, $v_c(S_i)$.

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

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

It is to be noted that many minor technical issues remain to be solved with the actual encoding method itself. For example, in order to maintain the actual mark reference (i.e. the mean of all values of S_i, $avg(S_i)$), items in S_i are to be altered in pairs. Other simple technical tweaks had to be taken into account.

Getting back to the encoding procedure, the question arises on how to perform the required item alterations so as to also satisfy the given "usability" metrics (i.e. $\mathbb{G}$) for the data in case. There are two possible approaches.

The first approach simply performs the primitive watermarking step (e.g. for S_i) and then checks for data usability with respect to $\mathbb{G}$. If watermarking altered main usability restrictions, simply ignore S_i and consider the next secretly selected subset to encode the rest of the watermark. This will result in errors in the encoded marks but by using majority voting (see Section 3.5) over

a large number of encoded mark copies in the data, the errors will (hopefuly) be eliminated in the result.

This approach is very attractive for dynamically generated/shaped data where the selected subsets S_i will have small sizes and the mark redundancy will be high even in small amounts of the original data.

The previous approach does not make optimal use of the existing bandwidth in the provided subset S_i. Another idea would be to check for data usability after *each* item alteration and adjust the encoding behavior accordingly. This can happen for example by choosing another item (if available) if the current considered one does not preserve data usability after alteration. While more optimal, this approach presents higher computing overhead at watermarking time.

The ideal approach is probably a combination of the previously presented ideas. Our implementation is based on the first approach although minor tweaks could bring it closer to a combination among the two.

3.5 Resilience. Vulnerability.

A decision needs to be made determining the size of the subsets selected in the amplification step (i.e. $|S_i|$). Given that our method embeds 1 bit per subset we have a total mark bandwidth of $\frac{|S|}{|S_i|}$ bits. Thus the size of the selected subsets determines directly the total mark encoding bandwidth.

This can and should be considered as a fine-tuning step for the particular data usability metrics provided. If those metrics are too restrictive, more items will be needed inside S_i to be able to encode one bit while still preserving required usability. On the other hand if the usability metrics are more on the relaxed side, S_i can be very small, sometimes even 10-15 items. This enables for more encoding bandwidth overall.

S usually is large enough to provide for multiple embeddings of the original mark which is a requirement in order to survive subset "cut" attacks (i.e. to preserve the mark even inside a "cut" subset of data).

We can embed the main watermark no more than $\frac{|S|}{|S_i| \times m}$ times. At watermark detection time, after recovering all the watermark copies from the given data ***majority voting*** over all the recovered watermark bits is deployed in order to determine the most likely initial watermark bits, see Figure 3.

Another interesting point to be made here is considering the inherent attack-vulnerability of the watermarking scheme. As shown also in previous research [14], bringing the watermarked data as close as possible to the allowable distortion bounds ("usability vicinity" limits) is of definite benefit in making the result's data usability as fragile as possible to any attack.

An attacker will be faced with the problem of removing/altering the watermark and now any changes he performs with this intent have an increased

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. Majority voting over three recovered watermark copies for a 6 bit sized original watermark.

likelihood of making the data invalid with respect to the guaranteed usability metrics[3], thus removing or at least diminishing its value.

In effect we integrated this idea also in our primitive encoding implementation. Not only do we embed as many watermark copies as possible, but also, as watermark embedding progresses, a certain embedding aggressiveness factor increases, determining actual changes to the data to be performed more and more up to the permitted limit and not only as required.

Note: The *incremental* increase of the aggressiveness factor is needed so as to make sure that at least several copies of the mark were embedded successfully, before changing the data too aggressively might not allow for entire mark copies to be embedded (i.e. while maintaining data usability).

We performed various attack-related experiments on various data and determined very promising features of our embedding method. The "confidence-violators" primitive set encoding proves to be resilient to a considerable amount of randomly occurring uniformly distributed surgeries (i.e. attacker with no extra knowledge, in this case item removals) before watermark alterations occur. Even then, there exists the ability to "trace" or approximate the original watermark to a certain degree (i.e. by trying to infer the original mark value from an invalid set). Most of the considered data tolerated up to 20-25% data loss before mark alterations occurred, as illustrated also in Figure 4.

Thus, our watermarking scheme provides resilience for a set of attack types, including re-sorting (**A4**), subset selection (e.g. up to 25-30% data removal), massive subset selection (e.g. watermark preserved even in half of the data if data halved randomly), linear changes (e.g. adding/multiplication), random item(s) alterations attacks that preserve data usability.

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

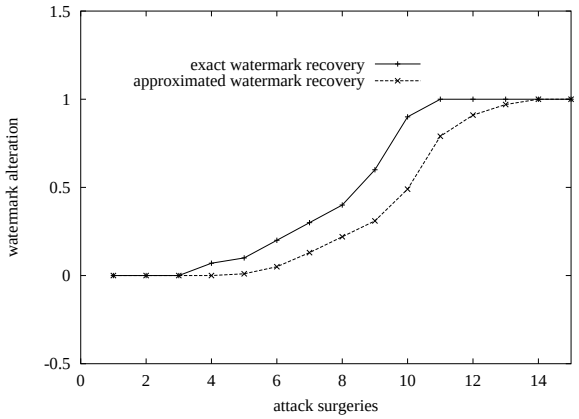

Fig. 4. Experiments on resilience to surgeries (data loss, "cut" attacks). The item set size considered was 35, experiments were performed on 10 different sets of close to normally distributed data. Various other parameters: $v_{false} = 5\%, v_{true} = 9\%, c = 88\%$. The average behavior is plotted here. Up to 25% data loss was tolerated gracefuly by most the tested data.

4 Implementation. Experiments.

This section presents aspects of our proof-of-concept implementation and some of the results obtained on various data input.

4.1 The nrx.* package

nrx.* is our test-bed implementation of the algorithms presented in this paper. It is written using the Java language. The package receives as input a watermark to be embedded, a secret key to be used for embedding, the input data (in a special file format) as well as a set of external *usability plug-in modules*. The role of the plug-in modules is to allow user defined usability metrics to be used at run-time without recompilation and/or software restart (We implemented the case of the maximum allowable mean squared error usability metric). The software then uses those metrics to re-evaluate data usability after each atomic watermarking step as explained in Sections 3.3 and 3.1.

Once usability metrics are defined and all other parameters are in place, the watermarking module (see Figure 5) starts the process of watermarking. A rollback log is kept for each atomic step performed (i.e. 1-bit encoding) until data usability is assessed and confirmed. This allows for "rollbacks" in case usability is not preserved by the current atomic operation.

Watermark embedding continues until a maximal number of watermark copies have been embedded into the data, while preserving the guaranteed usability metrics.

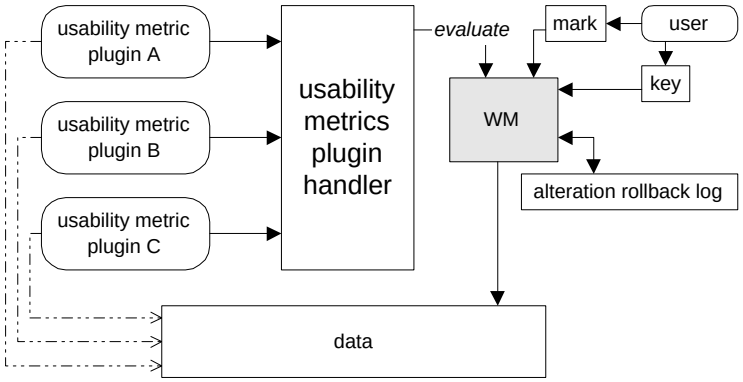

Fig. 5. The **nrx.*** package. Overview.

Watermark recovery takes as input the key used in embedding, the input data known to contain the watermark and recovers the set of watermark copies initially embedded. A final step of majority voting over the recovered copies completes the recovery process.

4.2 Experiments

For exemplification purposes we generated synthetic input data for nrwm.*. The size of the generated set was 10000, the set item values being uniformly distributed in the interval $(1, 1000)$. We considered the MSE usability metric case and conditioned our generated data to a maximum of 2% overall and 1% individual allowable MSE change (See Section 2.1). The experimental setup hardware consisted of a 1GHz CPU linux box with Sun JDK 1.3.1 and 384MB RAM. Algorithm parameters were adjusted repeatedly in an attempt to maximize the number of embedded copies, establishing them around $c = 88\%, v_{false} = 5\%, v_{true} = 9\%$. Using the confidence violators encoding method with the above specified parameters we were able to repeatedly (for different generated input data) embed a total of over watermark 120-130 bits in the set of size 10000. The considered watermark consisted of 24 bits. We were able to resiliently embed at least 5 copies of the watermark each time, allowing for a final enforcing step of majority voting, as discussed in Sections 3.1 and 3.5. We then performed attack experiments on parts of the data in order to experimentally assess encoding resilience and obtained encouraging results, some presented in Figure 4. Pre-processed parts of the original data as well as their watermarked version are soon to be found at **http://www.cs.purdue.edu/homes/sion/wm/nrwm**.

5 Conclusions. Future Research.

In the present paper we introduced a solution to the problem of watermarking a numeric collection by (i) defining a new suitable mark encoding method for

numeric sets and (ii) designing an algorithmic securing mapping (i.e. mark amplification) from a simple encoding method to a more complex watermarking algorithm. We also developed a proof of concept implementation of our algorithms under the form of a Java software package, **nrx.***.

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

References

1. M.J. Atallah, V. Raskin (with M. Crogan, C. Hempelmann, F. Kerschbaum, D. Mohamed, and S. Naik). Natural language watermarking: Design, analysis, and a proof-of-concept implementation. In *Lecture Notes in Computer Science, Proc. 4th International Information Hiding Workshop, Pittsburgh, Pennsylvania, April 2001.* Springer Verlag, 2001.
2. Christian Collberg and Clark Thomborson. On the limits of software watermarking, August 1998.
3. Ingemar Cox, Jeffrey Bloom, and Matthew Miller. Digital watermarking. In *Digital Watermarking.* Morgan Kaufmann, 2001.
4. Ingemar J. Cox and Jean-Paul M. G. Linnartz. Public watermarks and resistance to tampering. In *International Conference on Image Processing (ICIP'97)*, Santa Barbara, California, U.S.A., 26–29 October 1997. IEEE.
5. Ingemar J. Cox, Matt L. Miller, and A. L. McKellips. Watermarking as communications with side information. *Proceedings of the IEEE (USA)*, 87(7):1127–1141, July 1999.
6. Edward J. Delp. Watermarking: Who cares? does it work? In Jana Dittmann, Petra Wohlmacher, Patrick Horster, and Ralf Steinmetz, editors, *Multimedia and Security – Workshop at ACM Multimedia'98*, volume 41 of *GMD Report*, pages 123–137, Bristol, United Kingdom, September 1998. ACM, GMD – Forschungszentrum Informationstechnik GmbH, Darmstadt, Germany.
7. Stefan Katzenbeisser (editor) and Fabien Petitcolas (editor). Information hiding techniques for steganography and digital watermarking. In *Information Hiding Techniques for Steganography and Digital Watermarking.* Artech House, 2001.
8. Bob Ellis. Public policy: New on-line surveys: Digital watermarking. *Computer Graphics*, 33(1):39–39, February 1999.
9. M. Kobayashi. Digital watermarking: Historical roots. IBM Research Report RT0199, IBM Japan, Tokyo, Japan, April 1997.
10. J. Palsberg, S. Krishnaswamy, M. Kwon, D. Ma, Q. Shao, and Y. Zhang. Experience with software watermarking. In *Proceedings of ACSAC, 16th Annual Computer Security Applications Conference*, pages 308–316, 2000.
11. Fabien A. P. Petitcolas, Ross J. Anderson, and Markus G. Kuhn. Attacks on copyright marking systems. In David Aucsmith, editor, *Information Hiding: Second International Workshop*, volume 1525 of *Lecture Notes in Computer Science*, pages 218–238, Portland, Oregon, U.S.A., 1998. Springer-Verlag, Berlin, Germany.

12. Fabien A. P. Petitcolas, Ross J. Anderson, and Markus G. Kuhn. Information hiding - a survey. *Proceedings of the IEEE*, 87(7):1062–1078, July 1999. Special issue on protection of multimedia content.
13. Radu Sion, Mikhail Atallah, and Sunil Prabhakar. On watermarking semistructures. In *(submission for review), CERIAS TR 2001-54*, 2002.
14. Radu Sion, Mikhail Atallah, and Sunil Prabhakar. Power: Metrics for evaluating watermarking algorithms. In *Proceedings of IEEE ITCC02, CERIAS TR 2001-55.* IEEE Computer Society Press, 2002.
15. R. Venkatesan, V. Vazirani, and S. Sinha. A graph theoretic approach to software watermarking. In *Proceedings of the Fourth International Hiding Workshop, IH01*, 2001.
16. G. Voyatzis, N. Nikolaidis, and I. Pitas. Digital watermarking: an overview. In S. Theodoridis et al., editors, *Signal processing IX, theories and applications: proceedings of Eusipco-98, Ninth European Signal Processing Conference, Rhodes, Greece, 8–11 September 1998*, pages 9–12, Patras, Greece, 1998. Typorama Editions.
17. Jian Zhao and Eckhard Koch. A generic digital watermarking model. *Computers and Graphics*, 22(4):397–403, August 1998.
18. Jian Zhao, Eckhard Koch, Joe O'Ruanaidh, and Minerva M. Yeung. Digital watermarking: what will it do for me? and what it won't! In ACM, editor, *SIGGRAPH 99. Proceedings of the 1999 SIGGRAPH annual conference: Conference abstracts and applications*, Computer Graphics, pages 153–155, New York, NY 10036, USA, 1999. ACM Press.