Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/070517

Watermarking Generative Information Systems for Duplicate Traceability

Erik Sonnleitner[1,*] *and Josef Küng*[2]

Institute for Application-Oriented Knowledge Processing, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria

Received: 10 Jan. 2013, Revised: 13 May. 2013, Accepted: 14 May. 2013
Published online: 1 Sep. 2013

Abstract: This document outlines a watermarking scheme applicable to generative information systems in general, and dynamic web-page content delivered by a HTTP server in particular. The approach focuses on tracability of potential duplicates of previously served informational content. The suggested procedures are non-intrusive, resilient, blind, robust, reversible and suitable for large amounts of textual content. In order to do so, non- or semi-printable characters are inserted at syntactically and semantically valid places within the payload data.

Keywords: watermarking, digital steganography, web-page watermarking, generative information systems, tracability, copyright

1 Introduction

Steganography is the process of hiding secret information within other information, such that the secret's very existence is concealed [5]. The application of steganographic techniques in nowadays technological and economical landscape is primarily reflected by the emerging interests of the international entertainment industry, sustaining excessive research on the containment of copyright infringement on large scale. This particular steganographic application is commonly referred to as watermarking and follows similar, yet differently prioritized approaches in terms of technical theory and practical implementation.

Watermarking, as variant of steganography, outlines its main focus especially on robustness of the watermark itself, rather than the hiding aspect and the watermarking capacity [6] (the proportion of the secret bits which can be hidden within the bit stream of a given non-selective carrier medium). Within traditional steganography, it is attempted to maximize the capacity in order to hide as much information as possible within a relatively small data set of carrier bits. Moreover, it is utterly important that the steganographic message is subject to high concealment.

In terms of watermarking, the priorities shift towards high robustness of the watermark (the steganographic message), at the expense of hiddenness (although the latter remains highly attractive for most cases). A typical watermark consumes a rather low amount of informational bits, but is embedded more frequently, to ensure recovery and verifiability even if large portions of the carrier data are damaged, lost or inaccessible. While watermarking has undergone excessive research within the frame of multimedia data like images and videos, the scope of application within dynamic webpages constitutes a rather young scientific research field.

With respect to the dynamic nature of the content available in the World Wide Web, we can take advantage of the fact, that every document is separately requested by the client, wherein the server may introduce small, hardly noticeable changes within every document served. Moreover, in contrast to broadcasting an immutable static document (e.g. a newspaper), a HTTP server is normally aware of a considerable amount of information concerning its clients (e.g. originating IP address, date and time of request, requesting browser, operating system, rendering engine and granular versioning information), but static watermarking schemes do not allow to reproduce such information even if the existence of the watermark can be securely verified.

This paper is structured as follows:

– Section 1 shortly introduces to the subject of watermarking in general, and describes its motivation.

* Corresponding author e-mail: esonnleitner@faw.jku.at

© 2013 NSP
Natural Sciences Publishing Cor.

–Section 2 gives an outline of related work towards watermarking in information systems in general.
–Section 3 shows particular challanges regarding watermarking for back-tracing purposes.
–Section 4 gives an in-depth illustration of the presented watermarking proposals.
–Section 5 aims to deal with experimental results, based on prototype implementations of the proposed techniques.
–Section 6 discusses advantages, disadvantages and further considerations on the proposed algorithm.
–Section 7 draws a conclusive statement concerning the presented schemes.

2 Related work

2.1 General watermarking

During the past two decades, numerous research papers have been published on digital multimedia watermarking techniques, including image [7] [8] [9], audio [10] [11] and video [12] data as carrier. The watermarking technique we are proposing is closely related to the research field of relational database watermarking, which represents a rather young scientific research field.

2.2 Database watermarking

Distortion-based watermarking follows the traditional approach in information hiding, whereas certain attribute values are slightly modified in order to embed a watermark.

In [13], the least significant bits (LSB) of numerical attributes are used for hiding information. This technique is well-known from the field of image watermarking, but implies the serious disadvantage of numeric attribute modification which cannot be tolerated in numerous cases (e.g. timestamps, money quantities, military purposes, prescription provisions, etc.).

A different approach is shown in [14], where the authors embed a digital watermark into multi-word textual attributes by swapping the word order within given sentences, according to the watermark bits to be hidden. While providing considerable resistance against certain types of attacks, it is highly vulnerable to raising suspicion among third parties.

A mixed approach for hiding bits in numerical and textual data simultaniously is shown in [25], where Zhang et al propose a scheme which concurrently makes use of the LSB technique for numerical values, and insert carriage return and linefeed characters in textual values for bit hiding.

Our contribution is loosely based on [15], where a binary image is used as watermark, and hidden within multiple textual multi-word attributes of a database relation by inserting duplicate whitespaces between words, used to determine bit values of the watermark.

In addition, we introduce a tuple partitioning scheme which constitutes the basis for the introduced secret stegano-key, the security of the entire algorithm is based on. The tuple partitioning scheme embeds one bit of the watermark per partition, wherein every tuple contains this bit, as shown in [18].

3 Utilizing watermarks for back-tracing purposes

In order to trace back the origin of a particular portion of potentially duplicated data, it is essential to establish a scheme for distignuishing between differently watermarked data portions, even if its informational content is identical. Within the frame of a generative information system (like a World Wide Web service), every request served to a client has to be watermarked differently in order to create distinguishable data portions.

Conclusively, we have to design a watermarking algorithm whose extraction and verification processes do not rely on the knowledge of (all of) its secret parameters beforehand. Otherwise, we would not be able to initiate the extraction algorithm.

Although the subsequent approaches are similarly applicable to relational data, we will focus on the data as it is served by the HTTP server, as plain text data stream rather than focusing on the tuples of a database relation (although we will introduce a scheme for generating a virtual delimiter-based tuple set). This not only substantially enlarges the application areas for the proposed watermarking technique, but is also easily adaptable for modern non-relational database systems, commonly referred to as NoSQL (*Not-Only SQL*) databases which are especially used for data which does not rely on being stored in a relational manner on the one hand, and (partly) unstructured data whose applicability within traditional relational databases is limited.

For the algorithms and experiments of this proposal, we suppose that our carrier data is a reasonably sized, simple plain text document, constituted of a human-readable text written in English language. Furthermore, we assume the document to be served via HTTP and hence, is stateless and publicly available to an unrestricted number of semi-anonymous clients, whereas only information typically enclosed within a regular HTTP GET or HTTP POST request is available for further processing (e. g. target host, referer, browser type and version, et cetera).

© 2013 NSP
Natural Sciences Publishing Cor.

4 The proposed algorithm

4.1 Objectives

The main aims we have focussed on during development of the subsequently outlined watermarking technique are the following [16]:

–*Blindness* The watermarking process shall be *blind*, meaning that it's not necessarily required to have knowledge about the original, unwatermarked document data for watermark verification.
–*Robustness* The watermark shall be robust against various attack types, such as subset alteration- and subset deletion attacks. The experimental results on how different types of attacks affect the verifiability of the watermark are shown in Section 5.
–*Reversibility* It shall be possible to easily remove the watermark of a document, without corrupting or loosing original information. This prerequisite implies that all distortions made to the original document are lossless procedures.
–*Tracability of watermarked document* Within the extraction process of a watermarked document, the recovery of certain parameters from the (possibly duplicated) document should be easily possible. In this paper, this includes the IP address the original HTTP request was sent from, and the exact timestamp when the request has been initiated.
–*Security* Only the owner of the document himself should be able to extract and verify a given watermarked document. This is accomplished using a secret key in order to determine watermarked tuples, partitions and the watermark itself.

4.2 Notation used in this paper

The notation of variables, sets, parameters and operators used in this paper is listed in Table 1.

Furthermore we use CAPITALIZED words for variable names, double-slashes // for comments and square brackets [] for array indexes within the pseudo-code listings contained in this paper.

4.3 Document deconstruction

We are assuming a sufficiently large unstructured textual document Φ which is supposed to represent the plain source being included in a dynamic web-page. The document is split into tuple set $T = \{\tau_1, \tau_2, ..., \tau_n\}$ according to a delimiter δ which is a full stop period in our example (UTF-8 encoding `0x2e`). The tuple set T therefore represents the set of sentences within the text. Note that any τ_i may include multiple newlines or other non-word characters. The δ delimiter may be set to virtually any character which is repeatedly found within the source document, and may be adapted according to the source document's nature, structure and properties.

Table 1 Notation of symbols

Symbol	Description
Φ	Source document
$\|X\|$	Number of items in set or sequence X
T	Set of tuples (sentences) in document
τ_i	i^{th} tuple (sentence) in document
ω	Watermark
σ	Secret key
b_i^ω	i^{th} bit of watermark ω
Π	Set of partitions
π_i	i^{th} partition within Π
$\oplus$	Concatenation operator
h	Cryptographic hash function
δ	Delimiter character
α	Source IP address of HTTP request
β	Timestamp of HTTP request
θ	Word within a tuple/sentence
ζ	Function to calculate syllable positions

The process of document deconstruction is targeted on generating a virtual tuple set, which does not alter Φ, so we can easily reconstruct the original document from its tuples according to Equation (1).

$$\Phi = \tau_0 \oplus \tau_1 \oplus \ldots \oplus \tau_n \tag{1}$$

Therefore, the length $|\tau_i|$ of tuples $\tau \in T$ may variy greatly, depending on the statistical distribution characteristics of delimiter δ within Φ. Although the choice and definition of δ is not crucial for the proposed aglorithm to succeed, certain unfavorable properties regarding the delimiter may have a significant positive or negative impact on watermark extraction and verification:

–δ should split Φ into tuples, so that the statistical variance in length of each $\tau_i \in T$ regarding the standard deviation is reasonably small. Therefore, the more equally sized the tuples become, the better the embedding process can be performed.
–δ should be chosen, so that the tuples $\tau_i \in T$ are reasonably sized, since our approaches assumes every tuple to contain multiple textual words, whereas at least some of them should have more than two syllables.

Therefore, although the selection of δ remains unrestricted in theory, we will get better results, the closer we can stick to the guidelines above. For example, if we define $\delta :=' e'$, the average tuple size $|\tau_i|$ will be 7.8 characters assuming a source document in English language, whose relative occurrence frequency of the letter 'e' is roughly 12.7 % according to [20].

Accordingly, punctuation letters are probably most interesting for the definition of δ. The exact selection depends on structure and properties of Φ whatsoever, and can't be asserted in general.

© 2013 NSP
Natural Sciences Publishing Cor.

When the source document has been successfully deconstructed, the collective of tuples T is split into multiple equally sized partitions π_i of tuples. The process of partitioning is strongly dependent on a secret key σ, known solely to the creator of the watermark. Within the embedding process, for all $\tau_i \in \pi_j$ the very same particular bit b_i^ω is embedded.

4.4 Watermark generation

Our watermark ω is of dynamic nature, and changes on every HTTP request served. The particular watermark found within a previously watermarked document, in turn, enables us to exactly reproduce the circumstances in which this specific watermark has been used for embedding. This fact can be exploited within the frame of back-tracing details of the duplication.

In more detail, a document is watermarked with a dynamically calculated bit-string, which is generated upon every HTTP request based on a a three-tuple $\Sigma = \{\sigma, \alpha, \beta\}$, with σ being our secret key, α being the source IP address the HTTP request originates from, and β being the timestamp the HTTP request has been sent.

The generated watermark is called ω, and is persistently and securely stored on a server-side sequential list or database relation, together with the source parameters within Σ. The purpose of storing the mapping $\sigma \rightarrow \Sigma$ is to enable the document owner to lookup σ after the watermark extraction process in order to acquire the original values for Σ, and therefore the parameters which allow backtracing.

Let h be cryptographic hash function so that we can compute an HMAC[1] according to [3], and $\oplus$ be the concatenation operator. The calculation of the dynamic per-request watermark ω is subsequently shown in Equation (2).

$$\omega = h(\sigma \oplus h(\sigma \oplus \alpha \oplus \beta)) \bmod |\omega| \tag{2}$$

Once ω has been calculated, $\Sigma \cup \{\omega\}$ is stored on the server and kept private.

4.5 Data partitioning

The proposed partitioning process will split the total number of tuples $\tau_i \in T$ into (almost) equally sized tuple subsets (the *partitions*) $\pi_j \in \Pi$ for further processing while strictly maintaining the conditions shown in Equations (3) and (4).

$$\sum_{j}^{|\Pi|} |\pi_j| = |T| \tag{3}$$

$$\begin{aligned} \forall \pi_j \in \Pi: &\quad \pi_j \cap T = \pi_j \wedge \pi_j \cup T = T \\ \forall \pi_j \forall \pi_k \in \Pi: &\quad \pi_j \cap \pi_k = \emptyset \end{aligned} \tag{4}$$

The partitioning process offers two main benefits:

– Primarily, the partitioning scheme described below represents the most important factor of influence regarding accurate extractability of the watermark.
– Secondly, partitioning initially introduces the utilization of the secret key $\sigma \in \Sigma$, which is represented by a bit vector of arbitrary but fixed length $|\sigma|$. Therefore, extracting and/or verifying ω is not possible without σ, since knowledge about how data has been partitioned in the first place is strictly inevitable.

In order to assign all tuples $\tau_i \in T$ to disjoint partitions $\pi_j \in \Pi$, let $\oplus$ be the concatenation operator, τ_i be the textual sentence, wherein the watermark bits are embedded, and h be a cryptographic hash function, so that we can compute an HMAC for each tuple τ_i whose q least significant bits determine the number of the partition π_j the corresponding tuple is assigned to, so that $|\Pi| = 2^q$:

$$\forall \tau_i \in T: \begin{cases} \tau_i \in \pi_j & \text{if } h(\sigma \oplus h(\tau_i \oplus \sigma)) \bmod q \equiv j \\ \tau_i \notin \pi_j & \text{otherwise} \end{cases} \tag{5}$$

Implicitly, $q = \log_2 |\Pi|$ which should be carefully chosen according to the size of the tuple set $|T|$, and has been set to a value of 3 to 5 in our experiments, as described in Section 5.1. Therefore, Π consists between 8 and 32 tuple sub-sets, which are expected to contain roughly the same number of elements since h is supposed to provide uniformly distributed results, although this is not a strict constraint whatsoever. In our proof of concept implementation, we use MD5 for hashing.

4.6 Watermark insertion process

The watermark insertion process involves several steps:

– First, split the source document into tuples τ_i according to the pre-defined delimiter δ
– Second, partition the tuple set according to Equation (5)
– Third, iterate all partitions $\pi_i \in \Pi$ and embed the corresponding watermark bit multiple times within all $\tau_j \in \pi_i$, at the string positions dynamically calculated by the ζ function. Note, that all watermark bit positions within $\tau_j \in \pi_i$ contain the very same bit b_n^ω.

The watermark insertion process is done via character insertion at certain predefined places. Hereby, we take advantage of the fact, that most modern computer systems preferably use (and, if not, at least sufficiently decode) the Unicode text encoding, which offers multiple characters

[1] Hash-based message authentication code

© 2013 NSP
Natural Sciences Publishing Cor.

which are not directly visible in the text flow. We chose two characters representing the stegano-bits zero and one, which also have a well-defined and common HTML tag encoding, shown in Table 2.

Table 2 Substitution characters

Character name	UCS entity	HTML entity	Bit value
Soft-hyphen	U+00AD	`&shy;`	0
Zero-width joiner	U+200D	`&zwj;`	1

The soft-hyphen is a character which is not visibly printed by default. It suggests a place where a word-break can happen, and is normally interpreted by the font rendering engine (e. g. by browsers). Only if the word is being broken, a hyphen is printed at the place where the soft-hyphen appears.

The zero-width joiner is a special character which represents a space with zero length. It is not used in the majority of Western languages, nor is it represented visually by a rendering engine. According to the HTML standard, the character is interpreted by rendering engines as a 0-em space, which therefore allows line- and word-wrapping at places where the character occurs.

In order to break words according to its syllables, we utilize the the same algorithm as LaTeX does for breaking words, which is based on Franklin Liang's pioneering dissertation *Word Hy-phen-a-tion by Com-put-er* [1] from 1983. This is done by the ζ function, which returns a list of character positions where the function's argument may be split for layouting purposes.

For the sake of inconspicuousness, we insert soft-hypens at every word position returned by ζ, except for two-syllable words: Hereby, we distinguish between dividing the word by a soft-hyphen character only, and by a soft-hyphen followed by a zero-width joiner character, according to the bit value we want to embed. Note: Although any $\pi_i \in \Pi$ is assumed to contain roughly the same number of tuples, the steganographic capacity of a particular π_i may vary greatly, due to the fact that sentences may be utterly different in size, and we are only hiding bits in two-syllable words.

The watermark insertion process is done using the algorithm shown in Listing 1.

Listing 1 Watermark insertion algorithm

```
1  Split Φ in tuples τ_i by delimiter δ
2  partitionize(Φ) //according to Equation
       (5)
3
4  ForAll π_i ∈ Π:
5    b := ω_{i mod |ω|}  //Current watermark bit
6    ForAll τ_j^i ∈ π_i:
7      ForAll θ_k^j ∈ τ_j^i:
8        Calculate ζ //according to [1]
9        if |ζ| = 1, then
10       Insert bit b_i^ω in position ζ within
             θ_k^j
11     EndFor
12   EndFor
13 EndFor
```

A more detailed explanation of the pseudo-code shown in Listing 1 is discussed the following listing:

1. Initially, the source document Φ is deconstructed to acquire the collective of virtual tuples T by splitting according to the delimiter δ.
2. Once the tuple set is generated, it is splitted into equally sized partitions as shown in Equation (5).
3. Next, all partitions $\pi_i \in \Pi$ are sequentially iterated.
4. For each partition, we pre-define the watermarking bit b which is subsequently embedded multiple times within π_i.
5. Furthermore, all tuples $\tau_j^i \in \pi_i$ are iterated for further processing.
6. Similarly, every token (word) $\theta_k^j \in \tau_j^i$ is iterated, and serves as basis for performing the actual embedding process.
7. For bit embedding, we initially calculate the ζ value according to [1], which represents the total number of syllables in the currently examined token.
8. If $|\zeta| = 1$, then the current bit b_i^ω is inserted at position ζ within θ_k^j.

Once the embedding process finished, the entire source document has been successfully watermarked and is ready for being transmitted to the requesting client.

4.7 Watermark extraction process

After document deconstruction and tuple set partitioning, the watermark bits can be extracted without the knowledge of further secret parameters, simply by iterating the tokens within tuples within partitions, and extracting the bit information of all two-syllable words as defined in Table 2.

Similar to our first proposal published in [19], majority voting is utilized for reconstructing the embedded watermark ω. Once the extraction has been accomplished, the private parameter set Σ is looked up by the extracted ω value.

Nevertheless, it has to be taken into account that the extracted value of ω may be slightly corrupted, depending on various factors concerning the watermark extraction success probability, like modifications to the duplicated source document, like size and detail of the investigated document excerpt.

The watermark extraction algorithm contains of several steps:

© 2013 NSP
Natural Sciences Publishing Cor.

1. Split the source document into tuples τ_i according to the pre-defined delimiter δ.
2. Partition the watermarked document according to Equation (5).
3. On per-partition basis, deconstruct the tuples into words, which are used as argument for the ζ function. If ζ returns only one element (which suggests a two-syllable word), go to next step.
4. Extract bit b_i^{ω} according to Table 2.
5. Perform majority voting for bits of ω.

The algorithmic watermark extraction is shown in Listing 2.

Listing 2 Watermark extraction algorithm

```
split Φ in tuples τ_i by delimiter δ
partitionize(Φ) //according to Equation (5)

//(1) extract watermark bits
define EXT[|Π|] //array for extracted bits
ForAll π_i ∈ Π:
  ForAll τ_j ∈ π_i:
    ForAll θ_k^j ∈ τ_j^i:
      Calculate ζ //according to [1]
      if |ζ| = 1, then set CHAR := θ_k^j[ζ]
      if CHAR = U+00AD then EXT[i] := EXT[i] ⊕ 0
      if CHAR = U+200D then EXT[i] := EXT[i] ⊕ 1
    EndFor
  EndFor
EndFor

//(2) perform majority voting on elements in EXT
set WM := ""
For i := 0..|Π|:
  set ONES := ZEROES := 0
  For j := 0..|EXT[i]|:
    if EXT[i][j] = U+00AD then ONES := ONES + 1
    if EXT[i][j] = U+200D then ZEROES := ZEROES + 1
  EndFor
  if ONES > ZEROES
    then WM = WM ⊕ 1
    else WM = WM ⊕ 0
EndFor

//(3) lookup Σ by WM from server-side list
```

If the lookup procedure, fetching Σ from the securely stored server-side parameter list fails, we slightly untighten the conditional search parametrization by using a string similarity measurement metric.

The larger the co-domain of ω has been initially set to, the lower the chance for identifying a Σ record as false positive. For example, if $|\Sigma| = 32$, its co-domain is $2^{32} = 4294967296$. Assuming that the source document has been served several thousand times within a specific timeframe, the probability for watermark collisions remains sufficiently low, even if the watermark similarity coefficient has been untightend to drop below 100 %, in case ω isn't fully extractable.

4.8 Reversibility

In order to fully delete an embedded watermark from a document, we simply process the document character-wise and delete all occurrences of characters within Table 2, assuming that none of them occur in the original, unwatermarked source document. This restriction is strongly dependent on the particular insertion characters used for watermark embedding.

Listing 3 shows the Bash[2] command for entirely removing the watermark, without taking care of characters belonging to the original document.

Listing 3 Watermark removal for proposal 2

```
DOCUMENT="~/carrier.txt"
ZERO="\x00\xad"
ONE="\x20\xad"
sed -i 's/$ZERO//g' $DOCUMENT     # in-place substitution of zeroes
sed -i 's/$ONE//g' $DOCUMENT      # in-place substitution of ones
```

If the characters used for watermark insertion are known to be contained in the original document, the preceeding process is insufficient since original data will get lost. Alternatively, the watermark insertion process could create a insertion character map, which accurately keeps track of all byte offsets within the file stream of the document where watermark-related characters are inserted.

Another way which doesn't directly interfere with the watermarking algorithm itself, is to automatically create a binary[3] change set immediately upon finishing the insertion process. This technique requires a temporary backup of the original, unwatermarked carrier document for computing the changes. For this purpose, standard Unix tools like `diff` or `cmp` can be used.

[2] The *Bourne-again Shell*, the default shell environment on most Linux distributions.

[3] Depending on the characters used for representing bits of the watermark, also string-based change sets may be used.

© 2013 NSP
Natural Sciences Publishing Cor.

Both methods must keep the change set mapping up to date on every modification of the carrier. For security purposes, calculated change sets should be stored securely, e. g. by utilizing cryptographic routines.

Reversibility itself is a contrary topic among watermarking, whereas certain application scenarios, like for medical [23] or military purposes, strictly require the possibility of reverting changes introduced by watermark insertion [21].

4.9 Additionally encoding watermark identifiers

An interesting adjustment of the proposed algorithm is the utilization of *magic numbers*. A magic number is a plain random integer number of arbitrary but reasonable and fixed size, which serves as sole reference between the embedded watermark and the necessary private parameters used for watermark generation.

Hereby, the server-side list does not contain the generated watermark at all, but adds the magic number to the parameter set Σ. The watermark can subsequently be recalculated, if the magic number has been successfully extracted from a watermarked document. In order to do so, the magic number has to be embedded within the target carrier, in addition to the actual watermark.

This procedure adds a new layer of abstraction between σ and $\Sigma \setminus \{\sigma\}$, which may be useful for some corner cases, e. g. if the dynamic watermarks must not be stored on the server. Section 6 shows several ways of how to additionally embed a magic number, while focusing on high-grade extractability. Subsequent experiments with magic numbers, however, show a significant decrease in robustness and extractability and is therefore not discussed further.

4.10 Alternate definitions of the private parameter set

The set of private parameters Σ has been declared to contain a private key σ, and two variable parameters α and β. While σ is of high importance to the overall security of the proposal and is therefore necessarily needed, Σ may contain a theoretically unlimited number of additional parameters which will subsequently influence the generation of the actual watermark ω.

Aside from its previous declaration of α being the source IP of the incoming request and β being the corresponding timestamp, any other values may also be chosen.

Within the frame of the use-case referring to a HTTP service, especially the user-agent string can be of significant importance in order to increase the degree of uniqueness of certain parameters and therefore ease the mapping of a server-side list entry of Σ to the actual client. The HTTP user-agent usually contains:

–The web-browser type (e. g. *Mozilla* or *MSIE*),
–the web-browser engine version (e. g. *5.0*),
–detais about the platform in use (e. g. *X11* or *iPhone*),
–the Operating System (e. g. *Ubuntu* or *Windows NT 6.1*),
–the system's locale definitions (e. g. *en_US* or *de_AT*),
–the rendering engine the browser uses (e. g. *AppleWebKit* or *Gecko*),
–web-browser version details (e. g. *Firefox/18.0*)

The user agent is an independent key-value string in the HTTP request header, which is normally crafted and sent automatically by the web-browser. It is, however, very easy to spoof user-agent strings either by setting the corresponding browser preferences or by utilizing particular add-ons or browser extensions which will do so. The user agent is therefore not to be seen as trustworthy.

Despite the fact that user-agent string can easily be manipulated, it can be assumed that the vast majority of user-agent strings processed by a web-server are valid, due to the fact that many web-pages will modify the contents of the delivered web-pages according to certain fields within the user-agent string, like the web-browser (e. g. for different CSS style options to be compatible with legacy browsers) or the platform (e. g. to deliver mobile web-pages optimized for smaller screens and/or touch input devices like tablets and smart-phones).

5 Experimental results

5.1 Content & parametrization

Since our proposal has been designed with respect to watermarking non-relational data (or more precisely, textual data which has already been fetched from a database and pre-processed for being served to a client), we have chosen one chapter from a well-known mundane prose text document, namely *A Christmas Carol* by Charles Dickens, first published in 1843 and freely available without copyright restrictions. The selected excerpt of the document which is used for watermarking contains 613 sentences (12624 words).

Although the original document is of non-relational structure, our algorithm introduces a document deconstruction scheme, which results in the generation of pseudo-tuples. This approach enables us to simulate the very same tuple-based attacks as for attack scenarios towards database watermarking.

All conducted experiments have been made with various different parametrizations of the proposed algorithm, especially regarding the number of partitions and the size of the embedded watermark, with

–the partition exponent q set to 3, 4 and 5 (generating 8, 16 and 32 partitions, respectively), and
–the size of the watermark $|\omega|$ set to 8, 16 and 32 bits.

© 2013 NSP
Natural Sciences Publishing Cor.

Since the total size of the original document is significantly smaller than the carrier data used in [19], both, the number of partitions and the size of the embedded watermark have been shortened accordingly to offer high-grade extractability and a sufficiently large detection rate.

The experimental results of this proposal are visualized differently, due to the different characteristics of the algorithmic approaches. The graphs show the relationship between the altered tuple base after the attack has been performed, and the hamming distance of incorrectly extracted watermarks, if any inconsistencies occured.

5.2 Simulated attacks

Three types of attacks have been simulated on watermarked content:

- **Subset deletion attack:** Hereby, the attacker tries to delete or heavily distort the embedded watermark by removing information from the document, with the intent of making the watermark unextractable. In our experiment, the deletion process randomly removes sentences from the document until a minimum remnant of 5% of its original size has been reached.
- **Subset alteration attack:** The attacker tries to destroy or heavily distort the embedded watermark by modifying contents of existing text without actually deleting entire sentences. In our experiment, we randomly substituted roughly half of the characters within a watermarked sentence.
- **Insertion attack:** Hereby, the original watermarked sentences are left unaltered, but new (and therefore unwatermarked) sentences are consecutively added to the document. In our experiment, we consecutively add new ones until the size of the document increases to about 150% of its original size.

5.3 Subset deletion attack

The subset deletion attack has been simulated by deleting tuples $\tau_i \in \Phi$ from a previously watermarked document in steps of 5 % of the original tuple base. The results of attack and watermark extraction are shown in Figure 5.3.

The watermark extraction process can consistently verify the correct value of ω for every simulated parametrization, even when 80% of the original document has been deleted (or is not available at the time of extraction). This corresponds to a total number of 122 sentences.

The further trend constitutes, that lower values for l and q result in lower hamming distances of the extracted watermark, as opposed to the correct watermark (stored on the server). Note, Equation (6.1) incorporate fundamental parametrization constraints, for assuring all

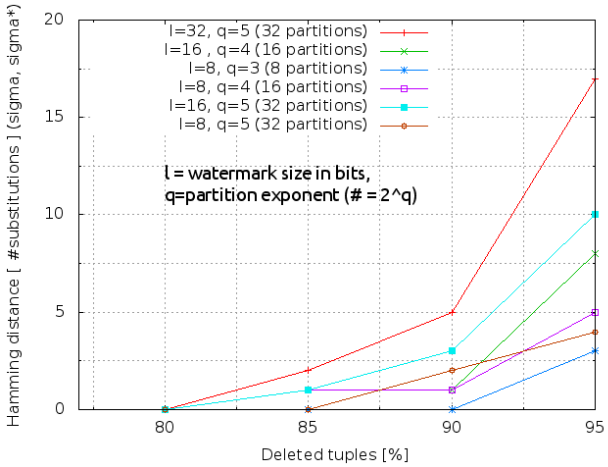


Fig. 1 Subset deletion attack

watermark bits to be fully embedded at least once across the whole document, since all tuples of a certain partition contain the very same bit.

$$|\Pi| \geq |\omega| \Leftrightarrow 2^q \geq l \tag{6}$$

Being the least favourable result, hiding a 32 bit watermark in 32 partitions shows the highest hamming distance at a deletion rate greater than 80 %, ultimately expanding to a peak of 17 when 95 % of the tuples have been deleted.

The experiment suggests, that the results tend to be significantly better when the watermark is embedded more than once across all partitions. Therefore, embedding a 16 bit watermark within 32 partitions shows a noticable flatter increase rate, providing a hamming distance of 1 with 85 %, 3 with 90 % and 10 with 95 % deleted tuples.

The best result is reached when utilizing only 8 partitions and an 8 bit watermark, which allows fullly correct watermark extraction even when 90 % of the tuples have been deleted. Choosing a short watermark also offers major drawbacks, which are discussed further in Section 6.

5.4 Subset alteration attack

For the subset alteration attack, we subsequently modify the textual content of tuples in steps of 5 % of the original document size.

The modification of one tuple alters roughly 50 % of the words therein, which are arbitrarily chosen and substituted by random character sequences. The results of the subset alteration attack are visualized in Figure 5.4.

Similar to the preceeding experiment, large values for l and q tend to heighten the hamming distances of the

© 2013 NSP
Natural Sciences Publishing Cor.

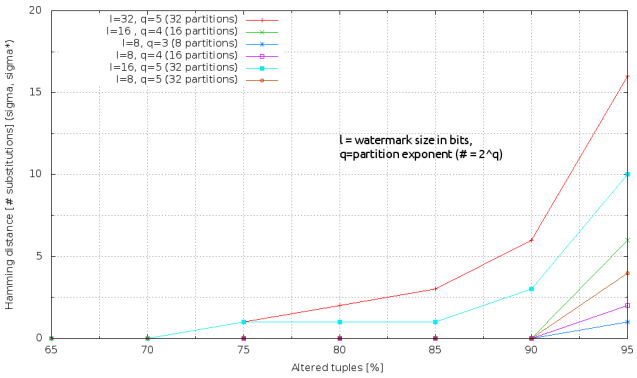


Fig. 2 Subset alteration attack

extracted watermark compared to the originally embedded one, starting to increase at 70 % tuple alteration.

Utilizing 32 partitions and a 16 bit watermark noticably flattens the similarity measurement curve, providing a hamming distance of 1 with 85 % tuples removed. All other parametrizations result in unadulterated watermark extraction with only 10 % tuples left, and show varying robustness when additionally halving the remnant contents to a total of 5 %.

5.5 Insertion attack

For the insertion attack, we subsequently added new sentences to a watermarked document in steps of 5% of its original size. The newly added tuples do not contain any watermark, and have been randomly choosen from an external, contentually independent documental resource.

Since the watermark extraction process solely relies on the extraction of the characters defined in Table 2 at pre-calculated places, the watermark extraction process is able to fully extract the embedded watermark even if the document grows to 200% of its original size.

Therefore, if no watermarking bits can be found in a specific tuple, it does not influence the watermark extraction vector, and therefore does not introduce distortion due to false positives or false negatives. In theory, there may, however, be certain exceptions to this assumption, if the modification process inserts any characters given in Table 2 at places calculated by the ζ function. The proability of such a scenario without an attacker specifically trying to exploit this very circumstance is sufficiently reasonable (with the given set of substitition characters), and therefore not covered by the attack simulation.

A discussion of more serious intimidations regarding complete removal of the watermark is discussed in Section 4.8.

6 Discussion

6.1 Carrier-dependent parametrization

The parametrization of the watermarking algorithm(s) is, as the preceeding experiments clearly point out, an utterly crucial process regarding subsequent extractability and watermark verification.

The parametrization in general encompasses three primary variables (among some secondary ones, mostly related to the structure of the document and the watermark):

1. The number of partitions $|\Pi|$ the original document is divided into,
2. the size of the watermark $|\omega|$ itself (the length of the bit vector), and
3. the delimiter δ for document decomposition purposes.

While the only hard constraint concerning the parametrization of the proposed watermarking algorithm is given by , the corresponding values should be chosen carefully according to the size and structural properties of the original carrier document.

If the size of the carrier document is very limited, by implication, the number of suitable positions for steganographic bit embedding is similarly low. This conclusively also limits the number of partition the carrier is dividable into, and hence, also the size of the watermark.

As a general rule of thumb, the more often a single particular bit of the watermark can be embedded into the carrier, the more likely the chance of a successful and error-free extraction process. We can establish a basic metric for this factor as shown in Equation (7) and (8).

$$\xi_\pi = \frac{|T|}{|\Pi|} = \frac{|T|}{2^q} \tag{7}$$

$$\xi_\omega = \frac{|\Pi|}{|\omega|} \tag{8}$$

In the presented experiments, the average partition size ξ_π ranges from 19.156 to 76.625. Note, that certain partitions may contain tuples of significantly diverging size which, in turn, affects the possible steganographic bit density.

Moreover, the watermark size to partitions ratio ξ_ω ranges from 1 to 4, whereas higher values tendend to provide better results in terms of correct extractability.

6.2 Implications of limited watermark sizes

A watermark being limited in size implicates a limited co-domain of possible values for the watermark. Therefore, an 8 bit watermark allows the generation of $2^8 = 256$ unique and distinct watermarks.

© 2013 NSP
Natural Sciences Publishing Cor.

For practical use, it should be considered expedient not to be at risk of fully exhausting the available co-domain of watermarks.

Firstly, this heightens the chance of watermark collisions on the server-side list. This concludes, that even if the watermark extraction process is able to fully extract the embedded watermark without failures, it may not be directly mapped to the corresponding tuple on the server-side list and may match more than one watermark.

Secondly, regarding its use on a web-server, the co-domain of the watermark size should be chosen according to the estimated number of client requests for a particular document per timeframe. On heavily frequented web pages serving thousands of requests per minute, the generation of an 8 bit watermark would result in an inacceptable rate of collisions, which will make the lookup procedure of the secret parameter set Σ unusable.

Correctly chosing the size of the watermark relates to the temporal period the document in question is attractive for third parties to duplicate. For example, the articles published by an online news agency will probably only be of high interest within the first few hours or days after publication.

On the other hand, chosing a watermark of 64 bit in size will recude the chance for collisions to an improbable small value. However, the document carrier has to offer enough useable bit positions for bit embedding, strongly correlating to the total document size. Approaches for enlarging the informational steganographic bit density are discussed in 6.5.

6.3 Selection of the similarity measurement algorithm

In the presented attacks simulations, the similarity measurement algorithm in use has always been the well-known Hamming Distance. The Hamming Distance, contrary to derivate forms like the Levenshtein Distance, only supports similarity measurement for strings of equal length. This constraint is met in our experiments, since the extracted watermark is always the same size as the original.

This restriction may cause difficulties if the size of the extracted bit vector differs from the original, which may be the case if either

1. no partitioning is used, and the watermark is repeatedly embedded throughout the document carrier with no pre-defined tuple sets which are known to contain the same bit, or
2. the attack eliminates all tuples from a certain partition, so that not even one tuple of the corresponding partition is left for extracting the originally stored bit value.

In the first case, the majority voting approach is inapplicable, since there are no pre-defined tuple sets (partitions) whose extracted bits could represent the basis for majority voting.

In the second case, while being highly unlikely without knowing the private σ parameter used during the partioning process, we could alter the extraction algorithm to insert a blank bit value in the extracted bit vector, since we can assume that the corresponding partition must have been created during the embedding process, but isn't contained in the watermarked data portion in question. This automatically increases the distance calculation by one, but – assuming a sufficiently large watermark co-domain – may still be unique in the server-side list of private parameters, and therefore allows direct mapping of the watermark.

In both cases, it is possible to utilize a similarity measurement algorithm which does not have the same-size restrictions described above, like the Levenshtein Distance or the Levenshtein-Damerau Distance. In addition to character substitution, these approaches also support character insertion and are hence well suited for this task.

6.4 Avoiding similarity measurement and boosting detection rate

The more traditional approach towards watermark insertion, was to simply repeat the entire vector ω throughout the carrier, as often as its size allows embedding bits, as shown in Listing 4 (a). This technique shows several disadvantages with respect to extractability, since majority voting is not applicable. Moreover, it's very complex to keep track of the exact position of a bit within the watermark, which just got extracted.

This is why we chose to embed the very same bit throughout an entire partition, shown in Listing 4 (b).

Listing 4 Bit sequence ordering

(a) $\forall \tau_j \in \pi_i$:

n:	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
$b_n^\omega \in \tau_0^{\pi_i}$:	x	y	y	x	x	y	x	y	x	y	y	x	x	y	x	y ...
$b_n^\omega \in \tau_1^{\pi_i}$:	x	y	y	x	x	y	x	y	x	y	y	x	x	y	x	y ...
$b_n^\omega \in \tau_2^{\pi_i}$:	x	y	y	x	x	y	x	y	x	y	y	x	x	y	x	y ...

(b) $\forall \tau_j \in \pi_i$:

$b_n^\omega \in \tau_0^{\pi_i}$:	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x ...
$b_n^\omega \in \tau_1^{\pi_i}$:	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y ...
$b_n^\omega \in \tau_2^{\pi_i}$:	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x ...

(c) $\forall \tau_j \in \pi_i$:

$b_n^\omega \in \tau_0^{\pi_i}$:	x	y	y	x	x	y	x	y	x	x	x	x	x	x	x	x ...
$b_n^\omega \in \tau_1^{\pi_i}$:	x	y	y	x	x	y	x	y	y	y	y	y	y	y	y	y ...
$b_n^\omega \in \tau_2^{\pi_i}$:	x	y	y	x	x	y	x	y	x	x	x	x	x	x	x	x ...

© 2013 NSP
Natural Sciences Publishing Cor.

However, using a hybrid technique, as shown in Listing 4 (c), may allow us to combine the positive aspects of both techniques. Hereby, we embed the whole watermark (or a *magic number* used solely for internal referencing) at the very beginning of every partition at first. Once this is done, the insertion procedure continues to embed the very same bit of the watermark until no tuples are left in the current partition.

Further research is currently going on within the frame of bit sequence ordering during the insertion process.

6.5 Steganographic information density

Packing more bits into a fixed-size carrier document increases its steganographic density. Various factors can influence the density, most notably the number of potential words we can hide bits in.

In the given algorithm, we focus on two-syllable words (which means, string tokens whose ζ value, returning the number of possible breaks, equals 1). We can dramatically enlarge potential bit positions by not limiting the insertion procedure to word breaks only – a virtual restriction which has been made in order to minimize suspicion.

Moreover, it is possible to enlarge the number of partitions by carefully chosing the δ delimiter character.

6.6 Watermark durability regarding typical duplication operations

Various experiments have shown, that an arbitrary portion of watermarked data using the characters given in Table 2, easily withstands opening, reading, writing and conversion in and between multiple different UTF-8 capable text processing applications. The results are shown in Table 3.

Table 3 Text editors used for processing watermarked data

Text editor	Character survival	Visibility
Microsoft Word 2010	Yes	No
Microsoft WordPad	Yes	Yes
Microsoft Notepad	Yes	No
Gedit	Yes	No
Vim	Yes	Yes
Open Office Writer	Yes	No

Only if the given portion of data is converted to a non-Unicode text encoding scheme like ASCII, the steganographic characters are either completely omitted during the conversion process, or interpreted according to the code table of the encoding scheme in question, which mostly leads to unreadable and/or non-printable characters (commonly referred to as *garbage characters*).

However, the conversion to another encoding scheme is, on most platforms and in most editors, a process which has to be willingly done and is not enforced automatically.

This implicates that both, byte-wise duplicates of the watermarked information, as well as direct copy-and-paste operations which may introduces changes in file type, rendering engine (editor), operating system and (partly) formatting are not supposed to interfere with the watermark

6.7 Alternative carriers for web-pages

Especially within the frame of dynamic web-pages, making use of generative HTML documents as carrier is standing to reason. Approaches towards steganographic watermark embedding techniques include:

- –Hiding information within HTML tags, attributes or attribute values covered in the corresponding W3C standard,
- –introducing new tags or attributes to hide information in,
- –rearranging the structure of the original HTML document (e. g. by resorting tags in such a way, that the rearrangement does not interfere with the visual page rendering,
- –changing the stylistics and occurrences of whitespaces (e. g. concerning the document's indentation or intra-tag spacing).

The application areas for such techniques are wide-spread, since HTML typically allows a wide variety of structuring a document on the one hand, and due to the fact that HTML rendering engines (typically represented by and included with the browser) are highly fault-tolerant to non-standard compliant pages.

Such approaches do offer the advantage, that the scope of changing a document's structure for the sake of watermarking purposes is utterly diverse with respect to the fact, that the user will not spot any differences in layout, composition and content of the page. Moreover, the same methodologies may also be applicable to related markup languages like XHTML, XML, RSS, Atom, and countless other less known XML-derived languages like MathML, GraphML, XAML or SVG [22].

The downside of HTML-based techniques is, that the previously discussed – and highly probable – case of simple copy-and-paste duplications of watermarked pages will probably not cover most changes made to the document related to the watermark.

6.8 Limitations & Further Research

Due to the nature of the watermarking approach, the algorithm does not support the calculation of a percentual similarity between the extracted and the original

© 2013 NSP
Natural Sciences Publishing Cor.

watermark, since the watermark extraction process must be 100% consistent in order to lookup the parameters of Σ.

If the watermarked document has been severely distorted, it may lead to an extracted value of σ^* which contains one or more bit flips. In our simulated attacks, this happens when about 85% of the original document has been deleted, for example.

In such a case, the $\sigma \rightarrow \Sigma$ mapping can not be done anymore. However, depending on how large the server-side mapping list has grown, and if the circumstances when and from where the original watermarked document has originally been served may be isolated to a certain degree, there are possibly reasonable chances to search for the most similar values of σ stored on the server. Since σ is generated from a cryptographic hash function whose results are quite uniformly distributed, chances are given to seriously narrow down the possible Σ parameters which could have led to a slightly altered value of the extracted σ.

In our experiments, we calculated the Hamming Distance between the original σ and the extracted σ^*, which are shown on the graphs below.

Since all substitution characters are well-defined UTF-8 entities, they readily survive copy-and-paste processes in virtually any text processing software, including Microsoft Word, Libre Office and Microsoft Wordpad. However, converting the text to plain ASCII will entirely destroy the watermark.

Moreover, there is a possibility of watermark collisions. The collision resistance is dependent only on the cryptographic hash function h, and the q least significant bits used. In our examples, we use 32 bits, featuring a co-domain of around four billion values. Whether this value is sufficient for preventing collisions or not is highly dependent on the visitor statistics of the particular web-page.

7 Conclusion

We have proposed a watermarking algorithm for dynamic web-page content, which uses newly and indepentenly generated watermark values for each HTTP request served. It is capable of tracing back specific parameters of a document, based in the watermark it contains. It proves to be quite robust against various types of attacks, although a 100% successfully extracted watermark is needed in order to provide back-tracability.

References

[1] Franklin Mark Liang, *Word Hy-phen-a-tion by Com-put-er*, Stanford University, (1983).

[2] Unicode Inc., *The Unicode Standard, Version 6.0, Core Specification*, via www.unicode.org, (2011).

[3] National Institute of Standards and Technology (NIST), *The keyed-hash message authentication code (HMAC)*, via csrc.nist.gov/publications/fips/fips198, (2002).

[4] B. Schneier, *Applied Cryptography*, John Wiley and Sons, (1995).

[5] N. Ferguson and B. Schneier and T. Kohno, *Cryptography Engineering*, John Wiley & Sons, (2010).

[6] F. Yaghmaee and M. Jamzad, *Estimating watermarking capacity in gray scale images based on image complexity*, EURASIP Journal on Advances in Signal Processing, (2010).

[7] G. C. Langelaar and I. Setyawan and R. L. Lagendijk, *Watermarking digital image and video data. A state-of-the-art overview*, IEEE Signal Processing Magazine, **17**, (2000).

[8] W. Bender and D. Gruhl and N. Morimoto and A. Lu, *Techniques in data hiding*, IBM Systems Journal, **35**, (1996).

[9] M. D. Swanson and B. Zhu and A. H. Tewfik, *Transparent robust image watermarking*, Proceedings of the International Conference on Image Processing, **3**, (1996).

[10] M. D. Swanson and B. Zhu and A. H. Tewfik and L. Boney, *Robust audio watermarking using perceptual masking*, Signal Processing, **66**, 337 – 355 (1997).

[11] M. Arnold, *Audio watermarking: Features, applications, algorithms*, Proceedings of the IEEE International Conference on Multimedia and Expo, (2000).

[12] G. Dorr and J. L. Dugelay, *A guide tour of video watermarking*, Signal Processing: Image Communication, **18**, (2003).

[13] L. Zhang and W. Gao and N. Jiang and L. Zhang and Y. Zhang, *Rational databases watermarking for extual and numerical data*, Proceedings of the International Conference on Mechatronic Science, Electric Engineering and Computer, (2011).

[14] D. Hanyurwimfura and Y. Liu and Z. Liu, *Text format based relational database watermarking for non-numeric data*, Proceedings of the International Conference on Computer Design and Applications, (2010).

[15] A. Al-Haj and A. Odeh, *Robust and blind watermarking of relational database systems*, Journal of Computer Science, **4**, 1024 – 1029 (2008).

[16] R. Halder and P. Shantanu and A. Cortesi, *Watermarking techniques for relational databases: Survey, classification and comparison*, Journal of Universal Computer Science, **16**, (2010).

[17] I. Kamel, *A schema for protecting the integrity of databases*, Computers and Security, **28**, 698 – 709 (2009).

[18] A. Deshpande and J. Gadge, *New watermarking technique for relational databases*, Proceedings of the Second International Conference on Emerging Trends in Engineering and Technology, (2009).

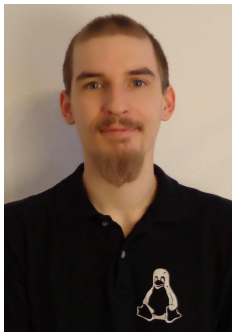
[19] E. Sonnleitner, *A robust watermarking approach for large databases*, Proceedings of the International IEEE Conference for Space and Satellite Communications (ESTEL) 2012, Security & Privacy Special Track, (2012).

[20] S. Singh, *The Code Book*, Random House, Delacorte Press, (2001).

[21] M. Farfoura and S. Horng and J. Lai and R. Run and R. Chen and M. K. Khan, *A blind reversible method for watermarking relational databases based on a time-stamping protocol*, Expert Systems with Applications, **39**, 3185 – 3196 (2012).

© 2013 NSP
Natural Sciences Publishing Cor.

[22] D. Gross-Amblard, *Query-preserving watermarking of relational databases and XML documents*, Proceedings to the ACM SIGMOD/PODS conference, (2003).
[23] Hassan I. Abdalla, *Improving data management for medical image watermarking*, Journal for Digital Information Management, **9**, 122 – 125 (2011).
[24] R. Mavudila Kongo and L. Masmoudi and M. Cherkaoui and A. Roukhe, *Dual-tree wavelet transform for medical image watermarking*, Journal of E-Technology, **3**, 144 – 160 (2012).
[25] Lizhong Zhang and Wei Gao and Nan Jiang and Liqiu Zhang and Yan Zhang, *Relational databases watermarking for textual and numerical data*, Proceedings of the 2011 International Conference on Mechatronic Science, Electric Engineering and Computer, (2011).



Erik Sonnleitner is Senior Scientist at the Institute for Application-Oriented Knowledge Processing (FAW). He studied Networks & Security and is currently focusing on digital distortive watermarking strategies in information systems, wherein he published various articles on and writes his doctoral thesis about. His interests and research topics cover sytem-, network- and application-security in general, and steganographic approaches on volatile data carriers in particular.

Josef Küng is associate professor at the Institute of Application-Oriented Knowledge Processing (FAW) at Johannes Kepler University Linz (JKU). His core competencies cover Information Systems, Knowledge Based Systems, Decision Support Systems, Sematic Technologies and Similarity Queries where he published a fair number of papers. Among other scientific service activities, Josef Küng is co-editor-in-chief of the journal on Transactions on Large-Scale Data- and Knowledge Centered Systems published by Springer.

© 2013 NSP
Natural Sciences Publishing Cor.