A Functional Taxonomy for Software Watermarking

Jasvir Nagra* **Clark Thomborson*** **Christian Collberg†**

*Department of Computer Science
The University of Auckland,
Private Bag 92019
Auckland, New Zealand,
{jas,cthombor}@cs.auckland.ac.nz

†Department of Computer Science
University of Arizona
Tucson, AZ 85721 USA
collberg@cs.arizona.edu

Abstract

Despite the recent surge of interest in digital watermarking technology from the research community, we lack a comprehensive and precise terminology for software watermarking. In this paper, we attempt to fill that gap by giving distinctive names for the various protective functions served by software watermarks: Validation Mark, Licensing Mark, Authorship Mark and Fingerprinting Mark. We identify the desirable properties and specific vulnerabilities of each type of watermark, and we illustrate the utility of our terminology in a discussion of recent results in software watermarking.

Keywords: Watermark, fingerprint, software licensing, authentication, steganography, software authorship.

1 Introduction

Digital watermarking has recently received a flurry of attention from the research community. Various aspects and applications of watermarking have been identified, but we lack an unambiguous and comprehensive terminology. This leads to some confusion and even apparent contradiction in the published literature.

We define *watermarking* as the process of embedding a small amount of identifying information in media and we define such embedded information as a *watermark*. In this paper, we are especially interested in watermarks that are embedded in software, however, the terms defined are equally applicable to other types of audio visual media.

The type of information identified by a watermark depends on the function that a watermark is designed to serve. We, hence further introduce four novel terms to unambiguously and conveniently denote the various functionally-distinct watermarks for software protection: Authorship Mark, Fingerprinting Mark, Validation Mark, and Licensing Mark.

In this paper we attempt to clarify some of the confusion that has arisen in the discussion of watermarks in academic writing. In Section 3, we seek to show examples were the lack of unified terminology may cause ambiguity among authors.

We develop and support our taxonomic definitions in Sections 4 and 5 below. To aid further investigation, in Sections 6 and 7 we offer a careful definition of the desirable properties of software watermarks, such as fragility and robustness. Our taxonomy and definitions enabled us to identify several novel classes of transformations on watermarked objects, which we use to further refine our taxonomy. In Sections 8, 9 and 10 we conclude this paper by giving simple examples of software watermarks, considering attacks on the various categories of watermarks, and finally developing what we believe to be a novel categorization of the attacks on fragile watermarks.

2 Motivation

An excellent and authoritative survey of digital watermarking commences with the following definition: "A digital watermark embeds an *imperceptible* [emphasis added] signal into data such as audio, video and images, for a variety of purposes, including captioning and copyright control" (Miller, Cox, Linnartz & Kalker 1999). In another article, two of the authors of this survey assert (we believe rightly) to the contrary: that some watermarks should be readily perceptible. "Copy protection applications require that a watermark can be read by anyone, even by potential copyright pirates, but nonetheless only the sender should be able to embed and erase the watermark" (Cox & Linnartz 1998). Other authors concur with this requirement of visibility: "By *watermarks* we mean 'marks' that are readily detectable, even by a casual user, such as a 'logo' or 'banner' that appears to be 'lightly' printed over each page of a document" (Kaplan 1996).

The unadorned term "watermark" is thus deeply ambiguous. Depending on the context, a watermark may be required to be invisible, "(usually) indistinguishable from the original" (Lacy, Quackenbush, Reibman & Snyder 1998), barely noticeable, or readily detectable. Others have noted this difficulty, and have made admirable (but incomplete) steps toward addressing it: "Several names have been coined for such techniques, and therefore it is necessary to clarify the differences. *Visible watermarks* ... are visual patterns ... very similar to visible paper watermarks. *Watermarking* ... has the additional notion of robustness against attacks" (Kutter & Hartung 2000).

Robustness is another contested property of digital watermarks. According to the passage just cited, all watermarks are robust. Elsewhere, we read that digital watermarks are generally designed to be "robust ... [so that they will] survive common distortions... [but] In some applications, we want exactly the opposite of robustness. Consider, for example, the use of physical watermarks in bank notes. The point of these watermarks is that they do not survive any kind of copying, and therefore can be used to indicate the bill's authenticity" (Miller et al. 1999).

In view of this definitional confusion about the (in)visibility and (non)robustness that may be required of watermarks, we have come to believe that the ambiguous term "watermark" should be used only in its generic sense if academic, technical or legal precision is required. To fill the resulting linguistic gap, we introduce four novel terms to unambiguously and conveniently denote the various functionally-distinct digital constructs for software protection, that have been called "watermarks" elsewhere: Authorship Mark, Fingerprinting Mark, Validation Mark, and Licensing Mark.

Other authors have constructed a similar taxonomy to ours, but for watermarks on media rather than software, by considering functionality. However either they didn't name their categories (Miller et al. 1999) or they didn't identify the full range of generality (Kutter & Hartung 2000).

In our survey of the literature, we have identified a possible fifth category that we would call Secret Marks. Such marks carry subliminal information for military or espionage purposes (Miller et al. 1999); and therefore they are outside the scope of this paper. We are concerned only with protective marks on software. In our view, Secret Marks are steganographic ("invisible writing") but they should probably not be considered watermarks. They do have a long and fascinating history, starting from a mention in Herodotus (Petitcolas 2000) to recent academic inquiry (Kurak & McHugh 1992).

3 Background

The Oxford English Dictionary defines watermarks as being marginally perceptible: "a distinguishing mark or device impressed in the substance of a sheet of paper during manufacture, usually barely noticeable except when the sheet is held against strong light" (Simpson & Weiner 2000).

The art of media watermarking may have first been practiced in the Western world in the late thirteenth century, when discerning Italian customers and merchants in the town of Fabriano could examine paper for embossed watermarks. This was apparently a reliable means of identifying the paper mill at which it was produced, and perhaps even the artisan who smoothed the paper with a "calendar" stone. (Kutter & Hartung 2000).

Over the next seven hundred years, watermarking found many further applications. In 1990, the first digital watermarks were invented as a protective measure for digital imagery (Tanaka, Nakamura & Matsui 1990), cited in (Kutter & Hartung 2000). Image watermarking has received an ever-increasing level of attention since then; a recent survey contains references to fifty-four articles on this topic (Dugelay & Roche 2000).

Popular image-processing software such as Adobe's PhotoShop, Corel's Photo-Paint, and Paint Shop Pro offer the technically-literate public a straightforward way to embed easily-visible watermarks in their graphic creations (Chastain 2001).

Watermarking of software is an even more recent development, with a correspondingly smaller level of publication activity (Grover 1992), (Holmes 1994), (Samson 1994), (Davidson & Myhrvold 1996), (Moskowitz & Cooperman 1998), (Monden, Iida et al. 1998), (Collberg & Thomborson 1999),(Stern, Hachez, Fran & Quisquater 2000), (Pieprzyk 1999), (Palsberg, Krishnaswamy, Minseok, Ma, Shao & Zhang 2000), (Venkatesan, Vazirani & Sinha 2001).

Watermarking is only one of many approaches for dealing with the protection of intellectual property in software. Other technical means include the use of a registration database (Shivakumar & García-Molina 1996), advanced cryptography with hardware support (*e.g.* (Ostrovsky & Goldreich 1992), (Nardone et al. 2001)), obfuscation (Collberg, Thomborson & Low 1998), and tamperproofing (Aucsmith 1996). Non-technical means of protection include prosecution under copyright and patent law (Burk 2001), enforcement of licenses under contract law, appropriate business models (Davis 2001) and ethical controls (Pfleeger 1997).

An approximate determination of the author of a software product may be obtained by analysis of what we might call Inadvertent Authorship Marks. These have been studied elsewhere (Krsul 1994) and will not be considered further in this paper.

4 Setting

We develop a precise terminology for watermarking from the point of view of the person embedding the watermark. Where possible, we have kept the terminology consistent with terminology generally accepted in the information hiding research community.

To illustrate our terminology, we adopt stereotypical names and rules from cryptographic research: Alice, author of software O, wishes to distribute O via her distributor Douglas to her customer, Catherine. According to custom, the adversary is named Bob, who in our scenario wishes to gain financially by stealing Alice's intellectual property. Furthermore, to more accurately reflect the software distribution model, we extend this simple scenario to include the possibility that each of these characters represents multiple authors, distributors or customers, respectively.

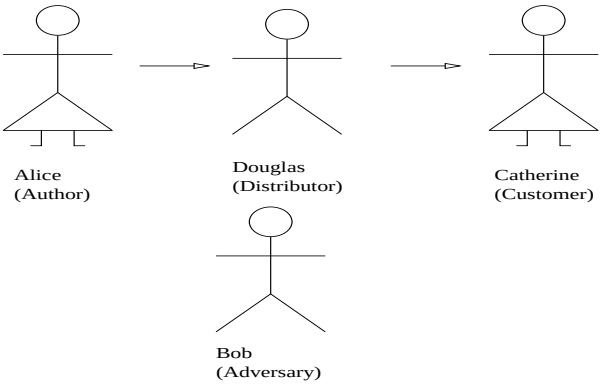

Figure 1: Actors in our analysis of software distribution. The arrows denote the desired flow of software, which Bob is trying to disrupt.

5 Applications of Digital Watermarking

Several scenarios in watermarking can be discovered by considering the interests of each of the players shown in Figure 1. In particular, there are three "good guys": our authors, Alice; the distributors, Douglas; and consumers, Catherine. Each one has a different interest to protect. Our adversary, Bob is involved not to protect his own interest but to infringe on the interests of the remaining participants. We will discuss Bob's activities in Section 9, when we define the effectiveness of watermarks.

5.1 Author(s)

The most commonly perceived application for watermarking is to identify the author of the software and to protect their intellectual property. In this application, the objective for our author, Alice is to embed

in her source code a mark that prevents Bob from claiming to have authored O. A common variation for such a scheme would be to allow multiple authors to alter the original and for each contributor to leave a mark so that a recognizer could identify the contributors to the final product, O. We define such marks as Authorship Marks.

> An **Authorship Mark(AM)** is a watermark that embeds in the software, information identifying its author.

Authorship Marks may identify a single author, in which case they are called Single Authorship Marks. Alternatively, they may allow a finite or arbitrarily large number of authors, in which case they are called Multiple Authorship Marks. These latter marks apply when the original author would like additional contributors to the source to be able to embed their own authorship marks.

We expect Authorship Marks to be visible and robust. Generally authors want their name to be visible to the end-user, as an assertion of quality and/or copyright. Robustness is important as a defense against copyright infringement.

5.2 Distributor(s)

The ability to identify the channel of distribution of O from Alice to Catherine may be valuable in order to identify one or more distributors of a particular illegal copy of O. The identification of distribution channels will also be useful when gathering statistics, for instance about the effectiveness of particular distributors or distribution channels. We define a mark designed to identify the distribution channel as a Fingerprinting Mark or a Fingerprint.

> A **Fingerprinting Mark(FM)** is a watermark that embeds information in the software identifying the serial number or purchaser of that software.

While Authorship Marks embed the same watermark in all copies of the same content, Fingerprinting Marks allow each distributed copy to be customized for each recipient. Such a scheme makes it possible for a watermark recognizer to track the distribution history of a particular copy of the source. The history may identify only a single distributor; alternatively a Fingerprinting Mark may record a succession of agents in a distribution chain, in which case we would call it a Multiple Fingerprinting Mark.

We expect Fingerprinting Marks to be invisible and robust. Generally the end-user is not interested in seeing information about the distribution of the object they are using, and invisibility tends to increase the robustness of the watermark. Robustness is a very important property of Fingerprinting Marks, in situations where license infringements are suspected.

5.3 Consumer(s)

At the end-user level, two actors have interests to protect. Firstly the end-user Catherine needs to be able to ascertain that the version of the software she is using has not been altered in any way. Secondly, the author Alice may want to ensure that Catherine is not violating her license agreement, *i.e.* that Catherine has paid for the software that she is in possession of, and that she is not using an illegal number of copies. Accordingly, we define two marks: Validation Marks to assure non-alteration, and Licensing Marks to control payment.

5.3.1 Validation Mark

> A **Validation Mark(VM)** is a watermark that embeds in the software, information verifying that the software is still essentially the same as when it was authored.

(We will give a precise definition of "essentially the same" in the next section.)

Digitally signed Java Applets (Sun Microsystems 2001) are Validation Marks by our definition because they allow the applet sandbox to validate incoming applets. In fact, like signed applets, Validation Marks often are a cryptographic digest of the document to be protected (Nat'l Inst. of Standards and Technology 1999). A general scheme for implementing this class of Validation Marks is illustrated in Figure 2.

The "Original" document shown in the figure is the document or software to be protected. Suppose Alice publishes a paper on watermarking and wishes to embed a Validation Mark to assure readers of it authenticity. However, she realises that the layout, spacing and font characteristics may need to be altered by her publisher (who plays the role of distributors in our scheme). The *essence extraction* process shown is a function whose input is the original document, and whose output is invariant for all versions of the original document that are essentially the same. In our current example, the essence function would strip presentation information from Alice's document. A cryptographic digest of this "essence" can then be generated and added to the document by the aggregation function shown in the figure. The aggregation may be as simple as concatenation, where Alice simply attaches the digest to the document or it may be more complex.

The Marked document can now be distributed. If a reader would like to verify the authenticity of the document, the publicly available Verifier detaches the body of the document from the digest, runs the essence extractor on the body and computes a digest. If the digest is missing or different from the one computed, then the reader knows the document has been altered in a way that Alice had not intended.

A number of essence-extraction algorithms for placing Validation Marks on digital imagery have been investigated. For example, see (Lin & Chang 2000) who distinguish between "robust authentication" in which no perceptible change is allowed in the protected image, and "content authentication" by a "semi-fragile watermark" which verifies the semantic content. We make this distinction in a lower level of our taxonomy, by suitable definition of what is "essentially the same" for a given application.

We expect Validation Marks to be fragile and visible. The end-user must be able to detect a Validation Mark to be assured of the integrity of the underlying object, and an appropriate level of fragility of a Validation Mark provides assurance that the object hasn't been modified to the point that it has become invalid.

5.3.2 Licensing Mark

> A **Licensing Mark(LM)** is a watermark that embeds in the software, information controlling how the software can be used.

Licensing Marks for software generally contain a decryption key as a fundamental part of their license control, and the software under license control is generally held in encrypted form. The decryption key will become ineffective if the License Mark is damaged. We thus expect a License Mark to be fragile; and we also expect it to be invisible, so that it is somewhat more difficult for an adversary (Bob, in our scenario) to forge.

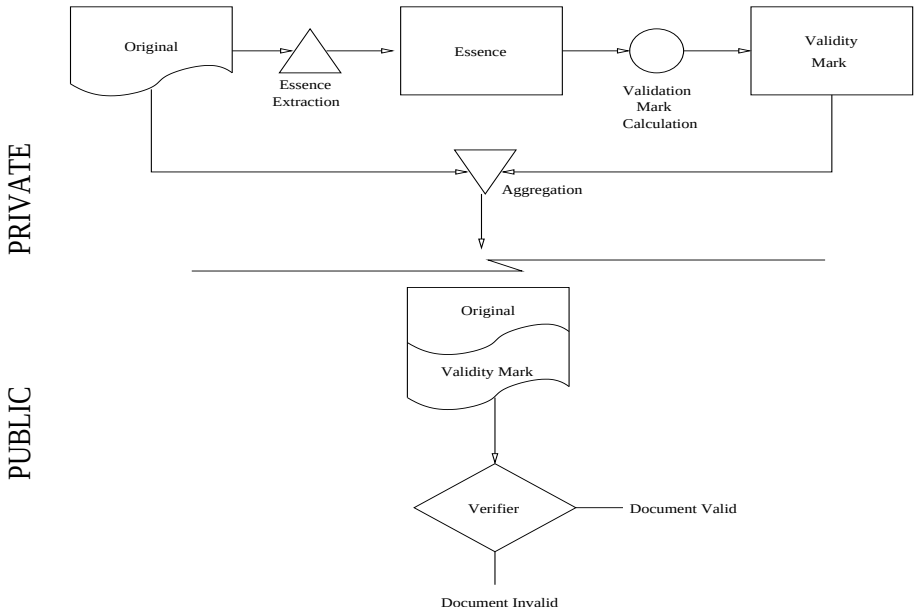

Figure 2: A general scheme for a class of Validation Marks

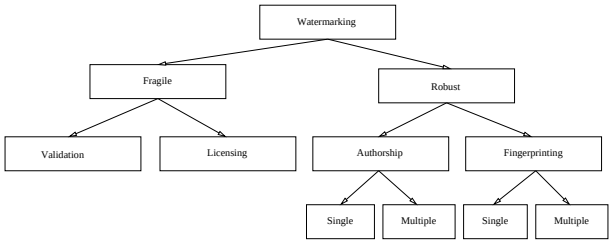

Figure 3: A taxonomy tree for digital watermarking

After we develop some mathematical notation in the next section, we will be able to describe how a License Mark can be used to limit the number of copies that can be made by a licensed user.

Figure 3 is a graphical presentation of our taxonomy for software watermarking.

6 Fundamental Properties

We adapt the customary mathematical notation for image watermarking, to give formal expressions and definitions for the most important concepts in software watermark embedding, recognition and other transformations on software.

Let O be a computer *program* that is available for manipulation in the current *state* S of a computer system. We use the notation $S = [O, ...]$ to denote a state S in which there is exactly one copy of O; we write $S' = [O, O, ...]$ to denote a state S' containing two identical copies of O. (We are not particularly concerned, in this paper, with the location or address of O in the state S.)

Let ω be a watermark, and $\mathcal{E}$ be a *watermark embedding function*, then

$$\mathcal{E}(S, \omega) = S_\omega$$

where $S = [O, ...]$ is a computer state containing an object to be watermarked and a desired watermark, and $S_\omega = [O_\omega, ...]$ is a computer state containing the desired watermarked object. The corresponding watermark recognition function, $\mathcal{R}$, has the property

$$\forall S_\omega : \mathcal{R}(S_\omega) = \omega$$

False-recognition of watermarks is not desirable, so we require

$$\forall S_\omega, \omega' \neq \omega : \mathcal{R}(S_\omega) \neq \omega'$$

The assertions above must be understood as *desiderata*. In any practical setting, to obtain a feasible solution, we would be forced to admit a small probability of error in watermark recognition.

A watermarking algorithm, $A = (\mathcal{E}, \mathcal{R})$, is a combination of an embedding function $\mathcal{E}$ with its corresponding recognition function $\mathcal{R}$.

6.1 Visible and Invisible Watermarks

In keeping with the spirit in which *visible watermark* is used in academic literature, we use this phrase to describe a watermarking algorithm $A = (\mathcal{E}, \mathcal{R})$ in which the recognition function $\mathcal{R}$ is public knowledge. In terms of our scenario of Figure 1, any visible watermark will be legible to our customer Catherine, and to our adversary Bob, because they know the recognition function.

We are especially interested in how the recognition function will operate on watermarked objects that undergo various transformations $T : S \to S$, such as those occurring in data compression, decompilation, etc. We thus formalise our definition of "legible" as follows:

We say that a watermarking algorithm, $A = (\mathcal{E}, \mathcal{R})$ is *legible* after a transformation T if

$$\forall S_\omega : \mathcal{R}(T(\mathcal{E}(S, \omega))) = \mathcal{R}(\mathcal{E}(\mathcal{S}, \omega))$$

By contrast, in an *invisible watermarking* algorithm, the recognition function (or some critical component thereof, such as an encryption key or "where to look for the mark") is not public knowledge. Such marks are intended to remain illegible to everyone except the watermarker (which may be either the Author or the Distributor, in our scenario).

6.2 Robust and Fragile Watermarks

In the case of a *robust watermark*, we want to be assured that the watermark is *legible* (can be recognized correctly by someone who knows $\mathcal{R}$), after a sufficiently-mild transformation has been applied. In the case of a *fragile watermark*, we want to be

assured that the watermark will become illegible if the transformation is sufficiently severe. We now formalise these notions.

A watermarking algorithm, $A = (\mathcal{E}, \mathcal{R})$ is *robust over* a set of transforms $\mathcal{T}$, if A is legible after all $T \in \mathcal{T}$.

Fragility is more subtle than robustness. A useful definition (as will be seen below) is that a watermarking algorithm, $A = (\mathcal{E}, \mathcal{R})$ is *fragile except for* a set of transforms $\mathcal{T}$, if A is not legible under any $T \notin \mathcal{T}$. The mathematically-inclined reader may note that this is a reasonable way to define fragility as "exactly the opposite" of robustness – in conformance with a prior informal English description elsewhere (Miller et al. 1999).

The simplest interesting class of transforms is $\mathcal{T}_{identity}$, which contains only the transform $T_{identity}$ making no change to state S.

Figure 4 is a Venn diagram showing $\mathcal{T}_{identity}$, as well as other classes of interest in software watermarking.

Transforms in the "move" set, $\mathcal{T}_{moves}$, allow the program O to be moved to different regions of memory but not to be duplicated. For convenience we include $T_{identity}$ as a "null-move" in $\mathcal{T}_{moves}$.

We are now able to describe the operation of the copy-restrictions embodied in the Content Protection for Recordable Media (CPRM) scheme proposed by a consortium called the 4C entity (4C Entity 2001*a*), (4C Entity 2001*b*), (Orlowski 2000). The 4C Entity consists of Intel, IBM, Toshiba and Matsushita. The CPRM proposal for copy-protection involves the encryption of copy-protected data on a portable disk or other removable storage device. The data can be decrypted only by a CPRM-compliant device or application (unless there is a breach of security, for example by public disclosure of the cryptographic keys and methods involved). The clever part of the scheme is that some information about the decryption key is stored in a restricted area of the removable storage device, inaccessible to devices and applications that are not CPRM-compliant. Thus if the protected data is copied (without the "permission" of the CPRM device or application) to some other device, it will be copied in encrypted form without its accompanying decryption key, and thus it will be unusable. Generally, CPRM licensing will permit the data to be moved from one CPRM-compliant device to another, but the old copy must be invalidated in the process. This can easily be accomplished by invalidating the decryption key on the first device while the movement is in progress. (We note that there is an unavoidable "race condition" in this invalidation: if the key is invalidated before the movement is complete, then an aborted move will destroy all usable copies of the protected object. In this case, the licensee must apply to the licensor for another copy. If the key is invalidated only after the movement is complete, then a carefully-timed abortion of the movement has a finite, but perhaps inconsequentially-small, chance of producing two usable copies.)

In terms of our taxonomy, the CPRM system described above is a License Mark that is fragile under $\mathcal{T}_{move}$.

The Content Protection System Architecture (of which the CPRM is a part) specifies another form of watermark for use when the protected data is sent in cleartext form (4C Entity 2001*a*). It would be tempting to call this a License Mark, however it must be highly robust rather than fragile – so that it will survive the many transformations and signal degradations that occur in broadcast media. In our taxonomy, this second CPSA mark is a Fingerprinting Mark. In this case the mark identifies the Distributor as someone who wishes to participate in the copy-protection system of the CPSA. The highly-robust mark will be recognized by CPSA-compliant storage devices, so they will refuse accept the protected data for writing.

Returning now to Figure 4, outside of $\mathcal{T}_{moves}$ we find $\mathcal{T}_{limited\text{-}copy(n)}$. These are useful for License Marks that allow up to n copies. Each of the transforms in $\mathcal{T}_{limited\text{-}copy(n)}$ have a curious behavior: they destroy one licensed copy and create two.

For example, if we wished to allow a single backup copy to be created, we would design a License Mark that is fragile under the set of transforms $\mathcal{T}_{moves} \cup T_1$, where

$$T_1(S) = \begin{cases} [O_2, O_3, ...] & \text{if } S = [O_1, ...] \\ S & \text{otherwise} \end{cases}$$

The original software installation must give rise to the state $S = [O_1, ...]$.

We could allow a second backup copy if we included T_2 in the fragility set of our License Mark, where

$$T_2(S) = \begin{cases} [O_4, O_5, ...] & \text{if } S = [O_2, ...] \\ S & \text{otherwise} \end{cases}$$

The scheme can be generalized to allow an arbitrary number of copies, if we have a sufficient range of watermark values $\omega = 1, 2, ..., n$ available in our License Mark.

Referring again to Figure 4, transformations in $T_{unlimited\ copy}$ allow an unlimited number of unchanged copies to be made. This set is simply defined by adding $T[O, ...] = [O, O, ...]$ to $\mathcal{T}_{limited\text{-}copy(n)}$ for arbitrary n.

Similar-text transforms $\mathcal{T}_{similar}$ apply to software which is written in a high level language and then compiled to produce a binary. Specifically: transformations in $\mathcal{T}_{similar}$ make alterations to the source code that results in no changes to the actual final binary after compilation. These include add, deletion or modification of comments, and in some cases variable and method names. We give a name to this set of transforms so that we can succinctly describe the robustness of simple watermarks that are carried in program text. For example, an Authorship Mark robust to $\mathcal{T}_{similar}$ might be a comment bearing the Author's name. Such marks are common in open-source code; we would say their security is based on the ethical and social constraints in the open-source community, rather than on the trivial effort required to strip comments from source code.

Outside of trusted communities, robust marks must survive at least trivial attacks, so we turn again to Figure 4 to consider the $\mathcal{T}_{state\text{-}preserving}$ transforms. Such transforms are relevant only to executable software. They may adjust the software code arbitrarily, but they can not change the data structures built by the software during its execution. Dynamic software watermarks are robust to such transforms, however static software watermarks are not (Collberg & Thomborson 1999).

In our exploration of Figure 4 we have now passed the outer limit of current technology. An ideal robustness for Authorship and Fingerprinting Marks, in many applications for software protection, would be over the set $\mathcal{T}_{semantic\text{-}preserving}$. Transforms in this set must preserve the observable behavior of the program, but they may change all other aspects.

Theoretical computer scientists have recently made some intriguing arguments about the impossibility of making a watermark that will withstand an attack by someone who is able to search through a

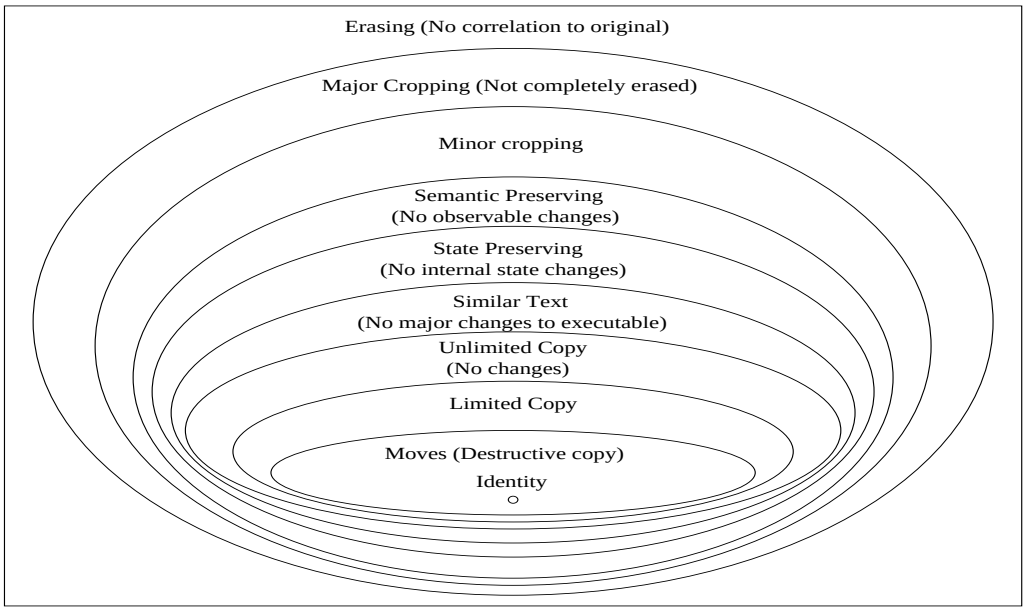

Figure 4: Classes of transformations

sufficiently wide variety of semantic-preserving transforms (Barak et al. 2001). We are still evaluating these arguments.

Conceivably, software watermarks could be designed to survive the $\mathcal{T}_{minor\text{-}cropping}$ transforms, which preserve most of the semantics of the protected object. The ultimate level of robustness in software watermarking would be an algorithm that is robust to all $\mathcal{T}_{major\text{-}cropping}$ transforms. Such transforms must preserve at least some of the original semantics, and would be very useful when protecting the intellectual property in each module of a large software system.

It is obviously impossible for any watermark to be robust to $\mathcal{T}_{erasing}$ transformations, which eradicate all information about the protected object (perhaps by over-writing it with information about some other object).

7 Other Important Properties

It is a common practice to compare the effectiveness of different watermarking schemes using metrics for visibility, robustness, efficiency, and fidelity (Miller et al. 1999, Voyatzis, Nikolaidis & Pitas 1998). However, as noted in the Introduction, metrics of visibility and robustness would have different meaning and desirability depending on the type of mark being evaluated. In the previous section, we indicated how visibility and robustness could be evaluated qualitatively, by considering the class of transforms after which the watermarking algorithm is guaranteed to remain legible. Similarly, invisibility and fragility could be evaluated by considering the class of transforms such that any transform outside this class is guaranteed to render the watermarking algorithm illegible.

In this section we briefly discuss the other important properties of watermarks.

7.1 Efficiency

There are two aspects of watermarking efficiency that need to be evaluated. These are, firstly, the computational cost involved in developing, embedding and recognizing marks; and secondly, the impact on the running time and memory consumption of embedding a watermark into a program.

7.1.1 Developer Time Costs

Often, before a watermark can be embedded into software, the software needs to be "prepared" in some way. This generally involves annotating the source code to indicate the locations where the watermark should reside, the parts of the source code that it should avoid (for example, highly optimized or extremely fragile sections of code) and the information that should be embedded. Depending on the watermarking scheme, the cost of this preparation phase may vary.

In one static watermarking scheme, "a dummy method (of a class), which will never be executed, is appended to a target Java source program" (Monden, Iida & ichi Matsumoto 2000). Presumably, since the methods is never executed, any arbitrary method inserted would suffice, however, unless some care is taken in ensuring the inserted method appears similar to other methods in the source, an adversary may be able to quickly identify these methods that carry the watermark. The ability to generate plausible looking methods that fool a human adversary may be difficult to automate fully and would require the intervention of a developer to be robust in practice. Also, a well-designed "opaque predicate" (Collberg et al. 1998) must be added to guard a plausible (but never-executed) call to the method, otherwise it will be easily eliminated as dead-code.

An even greater amount of developer time is required to embed watermarks using the current version of the SandMark system (Collberg & Townsend 2001). This system requires annotations throughout the source, to define points where the code to generate dynamic watermarks may be inserted.

We are not aware of any quantitative research on the developer time costs of various techniques for software watermarking. However we would expect to see a tradeoff between robustness and developer time, because writing an Authorship Mark into a comment field takes only a trivial amount of a developer's time.

7.1.2 Embedding and Recognition Time Costs

Usually watermarking consists of two operations that occur at different times. These are, the operation embedding a watermark into a source program and

the operation to recognize a watermark in an existing application.

In most applications, a time consuming embedding method would be acceptable in exchange for other beneficial properties. This is because most software is produced slowly. However, for other applications, such as livestream video or audio, a fast embedding method would be critical.

Similarly, the need for fast recognition of watermarks vary from application to application.

For some applications, it may in fact be desirable to have a recognizer that works slowly in order to stall oracle attacks. An *oracle attack* is where an adversary has access to the recognizer and makes small changes to the software until the watermark recognizer fails. Such a system would be particularly beneficial for watermarks that would need to be recognized only occasionally.

7.1.3 Runtime Costs

The runtime cost of a watermark is the increase in memory consumption and slow down in the running of a watermarked software compared to the same software without watermarks. Inserting software watermarks can result in software that runs more slowly or is considerably larger than the unwatermarked version. Whilst it is clear that some static marks such as those that use dummy methods or instruction ordering to encode marks will have minimal impact on the runtime efficiency, experimentation is required to establish the efficiency of complex dynamic watermarks.

Experimental results (Palsberg et al. 2000) indicate that "planted plane cubic tree" dynamic watermarks (Collberg & Thomborson 1999) increase the running time of a typical programs by no more than 7%.

7.2 Fidelity

A concept that is closely related to visibility is fidelity. We define Fidelity as "the extent to which embedding the watermark deteriorates the original content". In a software context, this measures how much a watermark introduces errors into a piece of software or how much it changes its stability.

Ideally a watermark preserves the entire semantics of a program, however, this may not be possible and occasionally not even desired (Moskowitz & Cooperman 1998).

Some watermarks may depend on highly unusual input to activate a copyright display. These watermarks are known as *Easter Eggs* (Wolfsites LLC 2001), and they suffer from some specific problems. For example, if the input that displays the Easter Egg should instead activate some other component in the system, errors could be introduced into an otherwise correct program.

The distinction between visibility and fidelity may require some clarification. A mark is visible or invisible by intent, depending upon the purpose: a copyright notice should be visible, while other marks may be invisible to avoid attacks based on an adversary knowing the location of the mark. Fidelity is concerned with how usable (semantically accurate) a program remains, in spite of the insertion of the watermark. The Netscape copyright message of Figure 10 has high fidelity, despite its high visibility, because it introduces little if any deterioration in the correctness of the program as perceived by its users.

8 Examples

In this section we give some simple examples of software watermarks, to illustrate how visible and invisible marks might be embedded. We begin by pointing out a fundamental technological distinction between "static" and "dynamic" software watermarks. This distinction does not arise in non-executable media.

A *static watermark* is defined in (Collberg & Thomborson 1999) as one which is stored in the application executable itself. For a simple example, consider the program in Figure 5. This program can be watermarked by inserting a simple visible, static watermark, namely a print statement that displays an authorship notice as shown in Figure 6.

```
main() {
      int a = 10;
      int b = 20;
      print(a+b);
}
```

Figure 5: Original unwatermarked program

Alternatively, an invisible static watermark could be used where the watermark is encoded as the ordering of the assignment statements as shown in Figure 7. In this case, the encoding is susceptible to attacks where the attacker, hoping to destroy the mark, randomly reorders statements in a way that maintains the semantics of the original.

```
main() {
      int a = 10;
      int b = 20;
      print("Authored by Alice");
      print(a+b);
}
```

Figure 6: Visible Static watermarked program

On the other hand, *dynamic watermarks* are stored in a program's execution state, rather than in the program code itself (Collberg & Thomborson 1999). In Figure 8, our original program is altered to introduce a variable W, our dynamic watermark.

```
main() {
      int b = 20;
      int a = 10;
      print(a+b);
}
```

Figure 7: Invisible Static watermarked program

8.1 Visible Authorship Marks

It is common for software to display a copyright notice as it starts up, or as an easily accessible menu option. See Figure 9.

8.2 Invisible Marks

The simplest kind of "invisible marks" are those that embed strings in the software itself, as in the example of Figure 6. In this case, the watermark recognizer is extremely simple, and involves merely looking through the binary for these strings. We illustrate this simple recognition process for a commercial program in Figure 10.

The static-string watermarking algorithm of Figures 6 and 10 suffers from an obvious shortcoming: an adversary may easily locate, and then modify, the strings that are serving as watermarks. In order to make it more difficult to find such watermarks, the

```
CM version 4e1, Copyright (C) 1990, 1991, 1992, 1993, 1994 Aubrey Jaffer.
SCM comes with ABSOLUTELY NO WARRANTY; for details type `(terms)'.
```

Figure 9: Copyright notice displayed by SCM, a scheme interpreter

```
jas@firebird jas\$ strings `which netscape` | grep Copyright
\# Copyright Netscape Communications Corp (C) 1996, 1997
bnlib 1.1 Copyright (c) 1995 Colin Plumb.
Copyright (C) 1995, Thomas G. Lane
 * Copyright (C) 1998 Netscape Communications Corporation.  All Rights
"The CPS and this certificate are copyrighted: Copyright (c) 1997 VeriSign, "
"Copyright (c)1996, 1997 VeriSign, Inc.  All Rights Reserved. CERTAIN "
Copyright
Copyright [c] 1995  INSO Corporation
 inflate 1.0.4 Copyright 1995-1996 Mark Adler
 deflate 1.0.4 Copyright 1995-1996 Jean-loup Gailly
jas@firebird jas\$
```

Figure 10: Copyright notices embedded in Netscape. Note that not all of these messages are in fact displayed while running the software.

```
main() {
      int a = 10;
      int b = 20;
      int W = b*a*10;
      print(a+b);
}
```

Figure 8: Dynamic watermarked program

contents of the string could be encrypted or encoded to appear part of the program data. Also, various forms of tamperproofing have been proposed, so that it becomes more difficult for the adversary to modify the watermarked program without damaging or destroying its functionality.

Alternatively, in an invisible watermarking scheme proposed in (Monden et al. 1998), the authorship information is encoded in the choice of opcodes and operand arguments. This encoding is then inserted as dummy methods which are known not to be executed. The drawbacks of this method were discussed in Section 7.1.1.

All the above methods suffer from the problem that the location of the watermarks can be straightforwardly deduced and removed, because the watermarks and software are not highly dependent on each other.

In (Stern et al. 2000), the authors propose encoding watermarks by hiding bits of information in the free choice of synonymous instruction sequences for live code, for example by using a MOV instruction instead of a PUSH instruction in x86 machine code. Each MOV could represent a "0" (in contexts where a PUSH could be substituted), and each PUSH could represent a "1". Regrettably, the authors do not discuss what seems to be an obvious threat – our adversary Bob could easily "flatten" the watermark by rewriting the code to use only MOV instructions. Some complex sets of synonyms may be resistant to this attack.

Some subtle attacks on invisible watermarks have been identified, leading to a requirement that the embedding process for such marks be "nonquasi-invertible" (Craver, Memon, Yeo & Yeung 1997).

9 Attacks

We now focus our attention on our attacker, Bob. His goal is to disrupt any watermarking system and his methods will vary greatly depending on whether the watermark he is attacking is robust or fragile.

9.1 Attacking Robust Marks

The principal kinds of attacks defined in (Collberg & Thomborson 1999) against robust marks are subtractive attacks, distortive attacks and additive attacks.

A subtractive attack is where Bob estimates the approximate location of the watermark and attempts to crop it out sufficiently that while what remains remains useful, the watermark is no longer recognizable.

On the other hand, in a distortive attack, an adversary makes uniform distortive changes throughout a program, and hence to any watermarks it may contain with the intention that the watermarker can no longer recognize her mark.

Finally, in an additive attack, the attacker adds his own watermark either overriding the original watermark completely or making it impossible to detect which watermark was applied first and hence is the authentic one.

In addition to these attacks, it is also possible to prevent a conclusive determination about the ownership of a watermark if an adversary is able to introduce confusion about the identity of the "real" watermark recognizer (Craver et al. 1997).

9.2 Attacking Fragile Marks

Attacks against fragile watermarks have not been discussed extensively in the literature. These require our adversary Bob to take a different approach than when he attacks robust marks. So far we have identified three broad classes of attacks on fragile watermarks.

9.2.1 Sneaking around Watermarks

An adversary may try to find and apply a set of essence preserving transforms that nevertheless fail to maintain the authors intent. Such an attack becomes more likely if the size of the set of essence preserving transforms is large or these transforms interact with each other in ways that are not easily apparent.

For example, earlier in this paper we introduced an example of Alice authoring a paper and adding a Validation Mark to it. This Mark was designed to be fragile to all transforms except those that altered layout, font and spacing so that a user could verify the document meant what Alice had intended it to mean. However, an adversary may be able to

introduce fontsets that display some letters as other letters. By selectively applying these fonts he alters the apparent meaning of the document. However, because the transformation is merely one of changing fonts, it is allowed by Alice, and will not disturb her fragile Validation Mark.

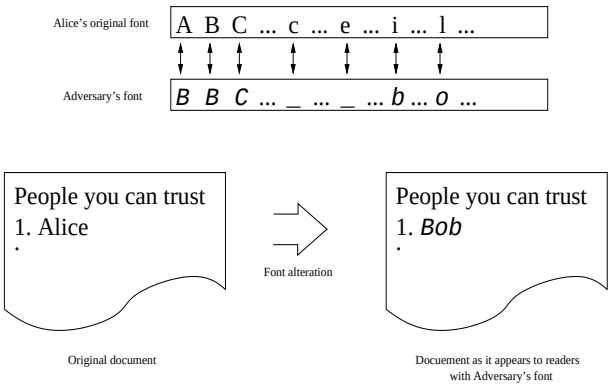

Figure 11: Substituting-fonts attack against a Validation Marked document

9.2.2 Reinserting Fragile Watermarks

If Bob has access to Alice's watermark embedder, then it is simple for him to make arbitrary changes to her software. He merely needs to embed a new fragile watermark, to validate the altered version.

In our example, this would be akin to Bob acquiring Alice's private key and digest algorithm, then creating a new digest for his altered version of the document.

9.2.3 Spoofing the Recognizer

The discussion of the watermark recognizer assumes that a customer wishing to verify the authenticity of a document is able to acquire the verifying securely. However, this may not necessarily be the case. An adversary may create a fake recognizer that ignores the absence of the author's fragile Validation Mark and validates the document anyway. This attack is also potent against robust watermarks (Craver et al. 1997).

10 Conclusion

Software watermarks can be classified based on the purpose of the mark. We have identified just four protective purposes in our survey of the literature to date. Depending on the specific purpose, differing combinations of properties such as robustness or fragility, perceptibility or invisibility, fidelity, and efficiency will be required. We introduced some new terminology to support our brief survey of the current state of the art in software watermarking.

References

4C Entity (2001*a*), 'Content protection system architecture, revision 0.81', Available `http://www.4centity.com/data/tech/cpsa/cpsa081.pdf`, August 2001.

4C Entity (2001*b*), 'Policy statement on use of content protection for recordable media, (CPRM) in certain applications', Available `http://www.4centity.com/data/tech/cprmfactsheet.pdf`, August 2001.

Aucsmith, D. (1996), Tamper resistant software: An implementation, *in* R. J. Anderson, ed., 'Information Hiding (Proceedings of the First International Workshop, IH'96), LNCS 1174', Springer, pp. 317–333.

Barak, B. et al. (2001), 'On the (im)possibility of obfuscating programs (extended abstract)'. Available `http://www.wisdom.weizmann.ac.il/~oded/obfuscate.html`, August 2001.

Burk, D. L. (2001), 'Copyrightable functions and patentable speech', *Communications of the ACM* **44**(2), 69–75.

Chastain, S. (2001), 'Protecting graphics on the web'. Available `http://graphicssoft.about.com/cs/protection1`, August 2001.

Collberg, C. & Thomborson, C. (1999), Software watermarking: Models and dynamic embeddings, *in* 'Symposium on Principles of Programming Languages', pp. 311–324.
URL: *citeseer.nj.nec.com/collberg99software.html*

Collberg, C., Thomborson, C. & Low, D. (1998), Manufacturing cheap, resilient, and stealthy opaque constructs, *in* 'Proc. 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages 1998, POPL'98', San Diego, CA (USA), pp. 184–196.

Collberg, C. & Townsend, G. (2001), 'Sandmark: Software watermarking for java'. Available `http://www.cs.arizona.edu/sandmark/`, August 2001.

Cox, I. & Linnartz, J.-P. (1998), 'Some general methods for tampering with watermarks', *IEEE Journal on Selected Areas in Communications* **16**(4), 587–593.

Craver, S., Memon, N., Yeo, B.-L. & Yeung, M. M. (1997), On the invertibility of invisible watermarking techniques, *in* 'IEEE Signal Processing Society 1997 International Conference on Image Processing (ICIP'97)', Santa Barbara, California.

Davidson, R. L. & Myhrvold, N. (1996), 'Method and system for generating and auditing a signature for a computer program', US Patent number 5,559,884.

Davis, R. (2001), 'The digital dilemma', *Communications of the ACM* **44**(2), 77–83.

Dugelay, J.-L. & Roche, S. (2000), A survey of current watermarking techniques, *in* S. Katzenbeisser & F. Petitcolas, eds, 'Information Hiding: Techniques for Steganography and Digital Watermarking', Artech House, pp. 121–148.

Grover, D. (1992), *The Protection of Computer Software: Its Technology and Applications*, The British Computer Society Monographs in Informatics, second edn, Cambridge University Press, chapter Program Identification.

Holmes, K. (1994), 'Computer software protection', US Patent number 5,287,407.

Kaplan, M. A. (1996), 'Ibm cryptolopes tm, superdistribution and digital rights management', Available `http://www.research.ibm.com/people/k/kaplan`, August 2001.

Krsul, I. (1994), Authorship analysis: Identifying the author of a program, Technical Report CSD-TR-94-030, Computer Science Deparment, Purdue University. Available: ftp://ftp.cerias.purdue.edu/pub/papers/ivan-krsul/krsul-spaf-authorship-analysis.ps, November 2000.

Kurak, C. & McHugh, J. (1992), A cautionary note on image downgrading, *in* 'Proceedings of the Eighth Annual Computer Security Applications Conference', San Antonio, TX, USA, pp. 153–159.

Kutter, M. & Hartung, F. (2000), Introduction to watermarking techniques, *in* S. Katzenbeisser & F. Petitcolas, eds, 'Information Hiding: Techniques for Steganography and Digital Watermarking', Artech House, pp. 97–120.

Lacy, J., Quackenbush, S. R., Reibman, A. R. & Snyder, J. H. (1998), Intellectual property protection systems and digital watermarking, *in* D. Aucsmith, ed., 'Information Hiding (Proceedings of the Second International Workshop, IH'98), LNCS 1525', Springer, pp. 158–168.

Lin, C.-Y. & Chang, S.-F. (2000), Semi-fragile watermarking for authenticating JPEG visual content, *in* 'SPIE International Conf. on Security and Watermarking of Multimedia Contents II, vol. 3971(13), EI '00', San Jose, USA.

Miller, M., Cox, I., Linnartz, J.-P. & Kalker, T. (1999), A review of watermarking principles and practices, *in* K. Parhi & T. Nishitani, eds, 'Digital Signal Processing in Multimedia Systems', Marcell Dekker Inc., pp. 461–485.

Monden, A., Iida, H. & ichi Matsumoto, K. (2000), A practical method for watermarking java programs, *in* 'The 24th Computer Software and Applications Conference'.

Monden, A., Iida, H. et al. (1998), A watermarking method for computer programs (in Japanese), *in* 'Proceedings of the 1998 Symposium on Cryptography and Information Security, SCIS'98', Institute of Electronics, Information and Communication Engineers. Available `http://tori.aist-nara.ac.jp/jmark`, August 2001.

Moskowitz, S. A. & Cooperman, M. (1998), 'Method for stega-cipher protection of computer code', US Patent number 5,745,569.

Nardone, J. et al. (2001), 'Tamper resistant methods and apparatus', US Patent 6,178,509 B1.

Nat'l Inst. of Standards and Technology (1999), 'Digital signature standards', FIPS Publication 186. Available `http://www.itl.nist.gov/fipspubs/fip186.htm`.

Orlowski, A. (2000), 'Everything you ever wanted to know about CPRM, but ZDNet wouldn't tell you...', *The Register* . Available `http://www.theregister.co.uk/content/2/15718.html`, August 2001.

Ostrovsky, R. & Goldreich, O. (1992), 'Comprehensive software system protection', US Patent number 5,123,045.

Palsberg, J., Krishnaswamy, S., Minseok, K., Ma, D., Shao, Q. & Zhang, Y. (2000), Experience with software watermarking, *in* 'Proceedings of the 16th Annual Computer Security Applications Conference, ACSAC '00', IEEE, pp. 308–316.

Petitcolas, F. (2000), Introduction to information hiding, *in* S. Katzenbeisser & F. Petitcolas, eds, 'Information Hiding: Techniques for Steganography and Digital Watermarking', Artech House, pp. 1–14.

Pfleeger, C. P. (1997), *Security in Computing*, 2nd edn, Prentice Hall.

Pieprzyk, J. (1999), Fingerprints for copyright software protection, *in* M. Mambo & Y. Zheng, eds, 'Proceedings of the Second International Workshop on Information Security, ISW'99 (LNCS 1729)', Springer, Germany, pp. 178–.

Samson, P. R. (1994), 'Apparatus and method for serializing and validating copies of computer software', US Patent number 5,287,408.

Shivakumar, N. & García-Molina, H. (1996), Building a scalable and accurate copy detection mechanism, *in* 'Proceedings of the First ACM International Conference on Digital Libraries DL'96', Bethesda, MD (USA). Available `http://www.acm.org/pubs/contents/proceedings/dl/226931/`, August 2001.

Simpson, J. A. & Weiner, E. S. C., eds (2000), *Oxford English Dictionary*, second edition edn, Oxford University Press, p. 176. Entry [5]"watermark".

Stern, J. P., Hachez, G., Fran c. K. & Quisquater, J.-J. (2000), Robust object watermarking: Application to code, *in* A. Pfitzmann, ed., 'Information Hiding (Proceedings of the Third International Workshop, IH'99), LNCS 1768', Springer, Germany.

Sun Microsystems (2001), 'Security and signed applet', Available `http://jsp2.java.sun.com/products/plugin/1.3/docs/netscape.html`.

Tanaka, K., Nakamura, Y. & Matsui, K. (1990), Embedding secret information into a dithered multi-level image, *in* 'Conference Record of the Military Communications Conference, MILCOM '90: A New Era, Vol. 1', IEEE, pp. 216–220.

Venkatesan, R., Vazirani, V. & Sinha, S. (2001), A graph theoretic approach to software watermarking, *in* 'Information Hiding (Proceedings of the Fourth International Workshop, IH'01)', Springer.

Voyatzis, G., Nikolaidis, N. & Pitas, I. (1998), Digital watermarking: an overview, *in* '9th European Signal Processing Conference (EUSIPCO'98)', Island of Rhodes, Greece, pp. 9–12.

Wolfsites LLC (2001), 'The Easter egg archive: Hidden secrets in software, movies, music and more!'. Available `http://www.eeggs.com`, August 2001.