# Fair fingerprinting protocol for attesting software misuses

Raphael C. S. Machado Davidson R. Boccardo
Instituto Nacional de Metrologia, Qualidade e Tecnologia
Duque de Caxias, RJ – Brazil
rcmachado@inmetro.gov.br, drboccardo@inmetro.gov.br

Vinícius G. Pereira de Sá Jayme L. Szwarcfiter
Universidade Federal do Rio de Janeiro
Rio de Janeiro, RJ – Brazil
vigusmao@dcc.ufrj.br, jayme@nce.ufrj.br

***Abstract*—Digital watermarks embed information into a host artifact in such a way that the functionalities of the artifact remain unchanged. Allowing for the timely retrieval of authorship/ownership information, and ideally hard to be removed, watermarks discourage piracy and have thus been regarded as important tools to protect the intellectual property. A watermark aimed at uniquely identifying an artifact is referred to as a fingerprint. After presenting a formal definition of digital watermarks, we introduce an unbiased fingerprinting protocol—based on oblivious transfer—that lends no advantage to the prosecuting party in a dispute around intellectual property breach.**

***Keywords*—software fingerprinting; oblivious transfer**

## I. Introduction

Consider the following scenario. Alice wishes to sell a computer program to Bob, but suspects that Bob may distribute the program to others. Alice therefore endows the program with the following sentence: "This software is meant to be used exclusively by Bob and cannot be redistributed." Alice cautiously employs digital watermarking techniques to embed the sentence in a way that avoids it being found and removed by a malicious Bob.

A few months later, Alice hears that Eva is using an excellent computer program she downloaded from the Internet. The program looks suspiciously familiar to Alice, who finds out after due analysis that it happens to be a copy of that exact version she had sold to Bob. Alice goes to court. During the trial, Alice provides the judge with the selling contracts of her software and the version downloaded by Eva, in which she is able to pinpoint the encoded sentence that explicitly points to Bob. When interrogated, the defendant Bob denies having criminally shared it on the Internet. Moreover, he argues that Alice, being the author of the program, could have inserted the incriminating sentence into any copy of the program she wished and might as well have distributed the program herself. The judge grants Bob the benefit of the doubt and dismisses the case for lack of evidence.

Watermarking techniques allow for the embedding of information in a digital artifact in such a way that the artifact's functionalities are not impacted. Furthermore, the removal of a digital watermark must be a difficult task for an attacker, so that an attempt to do so will likely lead to deterioration or destruction of the host artifact. Much research is under way in distinct application fields (see Section II) aiming at the protection of artistic, industrial and intellectual property. However, the resilience of watermarks against attacks still lacks formalization. Moreover, the applicability of watermarks in the chain of security protocols still demands further study, since the existing watermarking schemes do not allow for the attribution of guilt in cases of misuse of the artifact. As we have seen, suspicion may fall upon both sides of the dispute.

In the context of intellectual property protection, a watermark that provides unique identification of a digital artifact is termed *fingerprint*, and the artifact that carries it is said to be *traceable*. A limitation of the existing fingerprinting algorithms is therefore their inherent asymmetry (bias), whereby the side responsible for embedding the fingerprint (who may perfectly well be the ill-intentioned side) shall be in advantage by having access to different artifact versions, both traceable (with an embedded fingerprint) and untraceable (with no fingerprint). In a trial for misuse, the embedder would obviously pick the version that would not incriminate her. This problem could be solved by having a Trusted Third Party (TTP) responsible for embedding the fingerprint. However, when only two parties take part in the protocol, the one in charge of the embedding process will eventually have an unfair advantage over the other party in a dispute.

In the present paper, we describe a protocol involving two participants: the seller, henceforth called Alice, and the buyer, henceforth called Bob. The protocol will be built in a way that Alice sends to Bob a traceable artifact whose fingerprint Alice herself will not be able to know a posteriori. Such property is achieved by making use of the oblivious transfer protocol [1] with some appropriate modifications. Under such a protocol, it will be possible to determine—with an arbitrarily low probability of error—the real responsible for the misuse of the artifact. We also describe a verification mechanism that does not require the disclosure of the fingerprint contents or location even to the arbitrator.

The paper is structured as follows. In Section II, we describe known approaches to support the protection of digital intellectual property. In Section III, we present formal concepts associated to watermarks. Based on those concepts, we introduce our fingerprinting protocol in Section IV. In Section V, we describe a verification scheme, based on the partial transfer of knowledge, which keeps the fingerprint contents and location safe in the case of a trial. Section VI contains our concluding remarks.

## II. Related Works

In the literature, there are plenty of works tackling the embedding of digital watermarks in artifacts such as images [2], audio and video [2], [3], text documents [4], [5], and computer program [6], [7]. However, it is in the field of software protection that the development of fruitful approaches has been most markedly noted.

Methods for protection of intellectual property of software range from legal protection to technological protection. From the standpoint of legal protection, government and industry seek regulatory mechanisms related to industry and trade, such as patent laws, copyrights and trade secret [8]. From the standpoint of technological protection, several techniques have been developed to avoid reverse engineering, tampering and illegal distribution of software [9].

Obfuscation techniques aim at making reverse engineering a difficult task. They are based on semantics-preserving transformations which manage to considerably worsen the results provided by standard software analysis tools [9]. Tamper-proofing techniques aim at detecting unauthorised software modifications and responding to them. Detection methods are based on code introspection, state inspection and/or environment verification. The responding methods vary, but the most common ones are program termination, performance degradation or program restore [9].

Software watermarking techniques aim at discouraging piracy by detecting and tracing illegal distributions. They can be classified based on their embedder and extractor algorithms, and on their static or dynamic nature. Static watermarks lie within the code or data segments of a program, whereas dynamic watermarks are built in the program states during its execution. Embedding algorithms are typically based on code substitution, code reordering, register allocation, control flow graphs, abstraction interpretation and opaque predicates. Surveys on state-of-the-art watermarking techniques can be found in [6], [7].

The focus of this work is to support dispute resolving by means of an intelligent watermark *usage*. The text is therefore agnostic to embedding and extracting algorithms' particularities. In a typical dispute resolving scenario, two participants claim authorship (or legitimate ownership) of a digital artifact. The goal of the proposed protocol is to allow that a TTP (a judge, an arbitrator) solves the dispute by comparing the proofs presented by the participants. There are numerous works that deal with this problem for audio, video and image artifacts [10], [11]. We deal with the dispute resolving problem concerning software. To our knowledge, our study is the first, in the context of software protection, which deals with the fairness between parties, that is, which is able to dismiss arguments about the seller's (ill-)intentions. Moreover, it is to our knowledge the first to provide a verification scheme in which the arbitrator does not need to have access to the fingerprint itself. This allows for, say, a partially trustworthy arbitrator.

## III. Background

### A. Watermarks

The basic properties of a digital watermark (or fingerprint, when the goal is to uniquely identify a software copy) are:

- *stealth*, which refers to how undistinguishable a traceable artifact becomes from an untraceable one;
- *resilience*, which refers to how difficult it is for an attacker to remove or tamper with the watermark without compromising the artifact's functionalities; and
- *verifiability*, which refers to the ability to unquestionably identify the artifact (or to attest its authoship/ownership) when exhibited to a TTP.

We say watermarks embed *content information* into a digital artifact—the *host*—in such a way that the final artifact—the *product*—is semantically equivalent to the host, i.e., the watermark does not alter the functionalities of the program.

Essentially, a watermarking scheme is defined by two algorithms. The first one, called the *embedder* algorithm, takes as input a digital artifact $u$ and an information $m$ to be embedded, and outputs a product $\tilde{u}$, semantically equivalent to $u$, but containing $m$. The second one, called the *extractor*, takes as input the watermarked artifact $\tilde{u}$ and outputs the embedded information $m$.

Formally, given a universe of digital artifacts $U$ (all possible hosts) endowed with a definition of equivalence classes, and a set $M$ of content information, a *watermarking scheme* is a tuple of functions $(f, g, r)$ related in the following way:

The *embedder function* $f : U \times M \times A \to U$ takes as input a digital artifact $u \in U$, an information $m \in M$ to be embedded and an auxiliar input $a \in A$ (which can be regarded as an *embedding key*), and outputs an artifact $\tilde{u} \in U$ semantically equivalent to $u$.

The *extractor function* $g : U \times B \to M$, takes as input a digital artifact $\tilde{u} \in U$ and an auxiliar input $b \in B$ (which can be regarded as an *extraction key*), outputting content information $m \in M$.

The bijective function $r : A \to B$ maps embedding to extraction keys, that is, it takes as input an element $a \in A$ and outputs an element $r(a) \in B$, in such way that, for any $u \in U$, $m \in M$, $a \in A$, if $\tilde{u} = f(u, m, a)$ so $m = g(\tilde{u}, r(a))$.

The size of the sets $A$ and $B$ will determine the effort necessary for a brute-force attack to succeed in recovering the embedded information. The addition of such auxiliary inputs—typically associated to the watermark location—allows for more than one watermark in the same artifact. This enables, for instance, the use of temporal protocols [12] in the watermark. This may increase its resilience against *addition attacks*, in which an adversary embeds additional watermarks to confound the extractor algorithms.

*Semantic equivalence and immersibility:* A key concept for the development of embedding information techniques within digital artifacts is the concept of "semantic equivalence". Watermarking algorithms must be able to modify the host

artifact, generating a product artifact with the same meaning or utility, but containing the intended information. In order to say "same meaning or utility", it is necessary to define the "semantics" related to the host artifact. Such semantics should be preserved after the embedding of content information.

Consider the set $U$ of artifacts of a certain kind in an application field. Consider a partition $(U_1, .., U_i, ...)$ of $U$ such that the artifacts of each $U_i$ have the same "meaning" or "utility" for that application field. Thus, we say the artifacts of each $U_i$ are *semantically equivalent*.

Examples of semantically equivalent artifacts may be given for distinct application fields. In the field of the "linguistics", for instance, we can say two texts are semantically equivalent if they are identical except for some swaps between the words "though" and "although". In the application field of computer programs it is possible to generate two semantically equivalent programs including, for example, dummy codes[1]. More sophisticated methods include diversifying computer programs by obfuscations means [13]. The definition of semantically equivalent classes depends, mainly, of the application field.

*Resilience:* A watermarking scheme should allow content information to be embedded in a way that is resilient to attacks. Simply put, a watermark would be optimally resilient if any efficient algorithm which removes or damages the watermark ends up producing, with high probability, an artifact which belongs to a distinct class of semantic equivalence.

Formally, let $X$ be a polynomial-time algorithm that somehow removes watermarks, i.e., a procedure that takes as input a watermarked artifact $\tilde{u}$ and outputs an artifact $u'$ without the watermark (or with a corrupted watermark). A watermarking scheme $(f, g, r)$ should ideally guarantee that the probability that $u'$ and $u$ belong to the same class of semantically equivalent artifacts is negligible.

Although no watermarking schemes are known which are totally resilient according to the above definitions, we henceforth assume their hypothetical existence.

*Verifiability:* In the classical watermarking examples in real artifacts, the watermark is visible and verifiable for anyone having access to the watermarked artifact. The difficulty to remove the watermark is based on the physical process of the watermark embedding. For instance, consider the difficulty to remove the low relief watermark in a printed document.

Watermarks in digital artifacts brought the need of an additional property, called *verifiability*, i.e., a way of "exhibiting" a watermark while still preserving its contents/location secret. Owing to its digital nature, a naively designed watermark may be easily removed if an adversary knows its contents/location.

Verifiability is one of the challenges for state-of-the-art watermarking schemes. In the most recent watermarking schemes, the simple exhibition of the watermark in a digital artifact makes its removal a trivial task for an attacker. However, for the fingerprinting application considered in the present work, we implement a scheme that avoids the disclosure of the watermark contents/location.

[1] A *dummy code* consists entirely of instructions that do not alter the program semantics.

### B. Oblivious transfer

Suppose Alice sells digital books and Bob wants to buy a book from Alice. However, Bob would like to keep secret about the item he is interested in, either for privacy reasons, or to avoid receiving unwanted advertising in the future, or for any other reason. The cryptography protocol called oblivious transfer makes it possible to transfer the electronic content from Alice to Bob in such way that Alice will have no idea about which content has interested Bob.

More formally, assume Alice defines $m_1, ..., m_k$ messages and Bob wants access to the $i$-th message from Alice. Essentially, Alice transfers all the messages $m_1, ..., m_k$, each one encrypted with a distinct symmetric key derived from a key chosen by Bob so that only the $i$-th encrypted message can be decrypted by Bob. Further details of this protocol are described in Section IV.

Several oblivious transfer models can be found in the literature, but they are in a sense equivalent [14], [15], [16], [17]. The oblivious transfer concept plays an important role in building other more complex protocols [18]. In Section IV, we modify it slightly to build a fingerprinting protocol for digital artifacts.

## IV. Proposed fingerprinting protocol

As discussed along the paper, the biggest difficulty in conceiving a fair fingerprinting protocol without recurring to a TTP is due to the fact that the party responsible for inserting the watermark will necessarily have access to both versions of the digital artifact: the traceable and the untraceable one. If this party is ill-intentioned—for example, intending to distribute copies of the digital artifact—she may choose to distribute the version that does not incriminate her; or, equivalently, which does incriminate someone else.

The protocol we propose consists in a modification of the oblivious transfer protocol. Essentially, we employ the oblivious transfer protocol 1-of-$n$ with sufficiently large $n$. In this protocol, one of $n$ possible messages is transmitted, but Alice (the seller) is unaware of which one was transmitted. Thus, although the seller has been responsible for inserting the watermark, she does not know which version was actually received by the buyer. If, later on, Alice finds any reason to distribute (maliciously) any version of the software, then with high probability she will distribute a version that was not the same one sent to Bob (the buyer). And if she distributes all versions, then it will become quite clear for a judge that the seller herself is to blame. It takes several adjustments to the protocol to prevent non-repudiation attacks. We describe next such adjustments, starting from a basic fingerprinting protocol based on oblivious transfer (Section IV-A) until we get to a more robust version (Section IV-C).

### A. Initial fingerprinting protocol based on oblivious transfer

A naive version of the fair fingerprinting protocol we intend to introduce makes use of the classic version of the oblivious transfer protocol, which allows the transference of an element from a set without the transmitter knowing which one was transmitted. We start with this basic version to better understand each proposed modification, and we evolve to the

final protocol step by step after considering several possible attack models.

**Basic fingerprinting protocol**

1) Alice creates $\alpha$ semantically equivalent variations $n_1, ..., n_\alpha$ of a digital artifact.
2) Alice creates $\alpha$ pairs of public/private keys $Pr_1, Pu_1, ..., Pr_\alpha, Pu_\alpha$, and sends the public keys $Pu_1, ..., Pu_\alpha$ to Bob.
3) Bob creates a random symmetric key $k$ and encrypts it with one public key $Pu_i$ among its $\alpha$ public keys of Alice, resulting in $E_{Pu_i}(k)$. Bob sends $E_{Pu_i}(k)$ to Alice.
4) Alice decrypts $E_{Pu_i}(k)$ with each private key $Pr_j$ among its $\alpha$ private keys, obtaining $D_{Pr_j}(E_{Pu_i}(k))$, with $j = 1, ..., \alpha$.
5) Alice encrypts each artifact $n_j$ among its $\alpha$ variations with the key $D_{Pr_j}(E_{Pu_i}(k))$, obtaining $E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j)$, and sends to Bob each $E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j)$.
6) Bob decrypts each $E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j)$, obtaining $D_k(E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j))$ and only when $i = j$ will he have a consistent digital artifact.

The key for understanding the above protocol is to note that, in step 4, Alice will obtain $\alpha$ "possible keys", only one of which will be Bob's actual key $k$. It is however impossible for Alice to know which key that is.

To identify responsibilities for improper use of digital artifacts, the arbitrator will need to have access to all the information generated during the protocol steps: the versions of the original digital artifact, the public and private keys. The verification protocol, to be executed by the arbitrator, for identifying a fingerprinted version $f(\tilde{n})$ is the following:

**Basic verification protocol**

1) Verify if the fingerprint $f(\tilde{n})$ is the same as in any of the traceable artifacts $n_1, ..., n_\alpha$.

Observe that is still necessary to guarantee that Alice, in fact, generated $\alpha$ artifacts with distinct fingerprints, and to have mechanisms to ensure that Bob, in fact, participated of the protocol execution.

### *B. Fingerprinting protocol with guarantee of distinct fingerprints*

The verification protocol above allows a simple attack by Alice. Alice can generate $\alpha$ variants of the digital artifact containing all the same fingerprint. This allows her to distribute any of the artifacts and blaming Bob. To avoid this attack, the arbitrator must verify that Alice, in fact, generated $\alpha$ artifacts with distinct fingerprints. A recent work about the generation and verification of distinct fingerprints based on a randomized graph-based scheme can be found in [19].

**Verification protocol with a naive test of distinct fingerprints**

1) Verify if the fingerprints $f(n_1), ..., f(f_\alpha)$ are mutually distinct.
2) Verify if the fingerprint $f(\tilde{n})$ is the same as in any of the artifacts $n_1, ..., n_\alpha$.

The modification above seems to be enough to guarantee that Alice can not distribute one of the artifacts $n_1, ..., n_\alpha$ because, with high probability, it will be a distinct artifact from the one obtained by Bob. However, there is no guarantee that the artifacts shown to the arbitrator are, in fact, the artifacts involved in the protocol. At this point, the distinction between the aim of the proposed protocol and that of the classic oblivious transfer protocol should be clear. In the basic protocol, there is only an interest in transferring an element from a certain set from Alice to Bob, without Alice knowing which the transferred element was. In this modified version of the fingerprinting protocol, it is fundamental that it can be demonstrated later on that all the elements were involved during the execution of the protocol, that is, the arbitrator should know which elements *could have been transferred* to Bob. This necessity requires some more modifications.

Our protocol will now encompass actions to make sure that Alice indeed generated $\alpha$ variants of the artifact with different fingerprints. To guarantee that, the modified protocol includes sending cryptographic hashes of the artifacts by Alice to Bob.

**Fingerprinting protocol with guarantee of distinct fingerprints**

1) Alice creates $\alpha$ semantically equivalent variations $n_1, ..., n_\alpha$ of a digital artifact.
2) Alice generates cryptographic hashes $h(n_1), ..., h(n_\alpha)$, signs them and sends to Bob $(h(n_1), ..., h(n_\alpha), s_A(h(n_1)|...|h(n_\alpha))$.
3) Bob verifies the signature of the hashes signed by Alice and returns the cryptographic hashes signed by him: $(h(n_1), ..., h(n_\alpha), s_B(h(n_1)|...|h(n_\alpha))$.
4) Alice verifies the signature of the hashes sent by Bob, creates $\alpha$ pairs of public/private keys $Pr_1, Pu_1, ..., Pr_\alpha, Pu_\alpha$, and sends the public keys $Pu_1, ..., Pu_\alpha$ to Bob.
5) Bob creates a random symmetric key $k$ and encrypts it with one public key $Pu_i$ among its $\alpha$ public keys of Alice, resulting in $E_{Pu_i}(k)$. Bob sends $E_{Pu_i}(k)$ to Alice.
6) Alice decrypts $E_{Pu_i}(k)$ with each private key $Pr_j$ among its $\alpha$ private keys, obtaining $D_{Pr_j}(E_{Pu_i}(k))$, with $j = 1, ..., \alpha$.
7) Alice encrypts each artifact $n_j$ among its $\alpha$ variations with the key $D_{Pr_j}(E_{Pu_i}(k))$, obtaining $E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j)$, and sends to Bob each $E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j)$.
8) Bob decrypts each $E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j)$, obtaining $D_k(E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j))$ and only when $i = j$ he will have a consistent digital artifact.

The inclusion of step 2 makes it possible to check the set of artifacts that Alice generated during the execution of the protocol. Naturally, the arbitrator will need to have access to the message $(h(n_1), ..., h(n_\alpha), s_A(h(n_1)|...|h(n_\alpha))$ to execute the verification algorithm. Step 3 indicates that

Bob had knowledge of the cryptographic hashes involved in the protocol. The verification protocol to be executed by the arbitrator is the following.

**Verification protocol to guarantee distinct fingerprints**

1) Verify if the signature $s_A(h(n_1)|...|h(n_\alpha)$ is valid and if each artifact $n_i$ has the correct cryptographic hash $h(n_i)$.
2) Verify if the signature $s_B(h(n_1)|...|h(n_\alpha))$ is valid.
3) Verify if the fingerprints $f(n_1), ..., f(f_\alpha)$ are mutually distinct.
4) Verify if the fingerprint $f(\tilde{n})$ is the same as in any of the artifacts $n_1, ..., n_\alpha$.

The step 1 allows the arbitrator to certify the set of artifacts involved during the execution of the protocol, resisting against the identical fingerprints attack. The step 2 ensures that Bob was involved during the execution of the protocol.

### C. Resistant protocol against non-repudiation attacks

Another challenge for the construction of the proposed fingerprinting protocol consists in identifying the version sent to Bob. Without it, it is impossible to the arbitrator imputing blame onto Bob for a possible artifact misuse. Obviously, this identification must occur a posteriori, i.e., upon the arbitrator's request. However, the inputs for this identification must still be provided by Alice. In practice, the proposed solution consists in Bob sending to Alice a cryptographic hash of his secret key, together with the digital signature. This modification can be seen in step 6.

**Resistant protocol against non-repudiation attacks**

1) Alice creates $\alpha$ semantically equivalent variations $n_1, ..., n_\alpha$ of a digital artifact.
2) Alice generates cryptographic hashes $h(n_1), ..., h(n_\alpha)$, signs and sends them to Bob $(h(n_1), ..., h(n_\alpha), s_A(h(n_1)|...|h(n_\alpha))$.
3) Bob verifies the signature of the hashes signed by Alice and returns the cryptographic hashes signed by him: $(h(n_1), ..., h(n_\alpha), s_B(h(n_1)|...|h(n_\alpha))$.
4) Alice verifies the signature of the hashes signed by Bob, creates $\alpha$ pairs of public/private keys $Pr_1, Pu_1, ..., Pr_\alpha, Pu_\alpha$, and sends the public keys $Pu_1, ..., Pu_\alpha$ to Bob.
5) Bob creates a random symmetric key $k$ and encrypts it with one public key $Pu_i$ among its $\alpha$ public keys of Alice, resulting in $E_{Pu_i}(k)$. Bob sends $E_{Pu_i}(k)$ to Alice.
6) Bob sends to Alice a cryptographic hash $h(k)$ of $k$, together with the digital signatures of $h(k)$ and $E_{Pu_i}(k)$: $(h(k), s_B(h(k)), s_B(E_{Pu_i}(k)))$
7) Alice verifies the signature of the objects signed by Bob and decrypts $E_{Pu_i}(k)$ with each private key $Pr_j$ among its $\alpha$ private keys, obtaining $D_{Pr_j}(E_{Pu_i}(k))$, with $j = 1, ..., \alpha$.
8) Alice encrypts each artifact $n_j$ among its $\alpha$ variations with the key $D_{Pr_j}(E_{Pu_i}(k))$, obtaining $E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j)$, and sends to Bob each $E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j)$.
9) Bob decrypts each $E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j)$, obtaining $D_k(E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j))$ and only when $i = j$ he will have a consistent digital artifact.

To execute the new verification algorithm, the arbitrator will need to have access to $k$, given by Bob, as well as to $h(k)$, $s_B(h(k)))$, $E_{Pu_i}(k)$ e $s_B(E_{Pu_i}(k))$. Access to all $\alpha$ public keys and $\alpha$ private keys generated by Alice is also necessary.

**Verification protocol with identification of Bob's key**

1) Verify if the signature $s_A(h(n_1)|...|h(n_\alpha)$ is valid and if each artifact $n_i$ has the correct cryptographic hash $h(n_i)$.
2) Verify if the signature $s_B(h(n_1)|...|h(n_\alpha))$ is valid.
3) Verify if the fingerprints $f(n_1), ..., f(f_\alpha)$ are mutually distinct.
4) Verify if the fingerprint $f(\tilde{n})$ is the same as in any of the artifacts $n_1, ..., n_\alpha$
5) Verify if the signatures $s_B(h(k))$ over $h(k)$ and $s_B(E_{Pu_i}(k))$ over $E_{Pu_i}(k)$ are valid and if the key $k$ given by Bob has, in fact, the cryptographic hash $h(k)$.

The Step 5 ensures that Bob was indeed given the same private key encrypted with one of Alice's public keys during the execution of the protocol. With the informations above, the arbitrator is able to identify the version $n_i$ obtained by Bob—by testing each of Alice's private keys—and, finally, to verify whether the fingerprint $f(n_i))$ is the same as the fingerprint $f(\tilde{n})$. Figure 1 wraps up the proposed fingerprinting protocol.

## V. Secure verification protocol for software fingerprinting

In this section, we develop a protocol that allows for fingerprint verification during a trial without revealing its contents or location, not even to the arbitrator. The simple exhibition of its contents or location makes it easier for an adversary/attacker to tamper with it. The protocol development starts from a scheme of partial transfer of knowledge, and afterwards we describe its application in the scenario of secure verification of fingerprints. The main advantage of the use of the partial transfer of knowledge scheme is the possibility to reveal information about authorship/ownership embed exactly in the bits to be shown, without making it any easier for an attacker to infer the content or location of the fingerprint. Since the transfer is partial, an attacker willing to remove the fingerprint will not have enough information to locate it within the artifact, so its removal will remain as hard after the verification as it used to be before it.

### A. Partial transfer of knowledge scheme

In Kilian's doctoral thesis [20], the following problem is described. Bob wants to factor a number $n$ with 500 bits which is known to be the product of five prime numbers of 100 bits. Alice knows one of the factors, denoted $q$, and is willing to sell 25 of its bits to Bob. Kilian proposes a method that allows Bob to make sure that Alice indeed knows one of the factors of $n$, and moreover allows Bob to make sure that each individual bit of $q$ has been correctly informed by Alice. The proposed scheme not only allows the disclosure of only some bits of $q$ but also uses schemes of commitment of individual bits of $q$ to ensure that those bits will not be disclosed without the consent of Alice. Finally, it allows for the use of oblivious transfer in such a way that Alice is unaware of the bits that get actually disclosed to Bob.

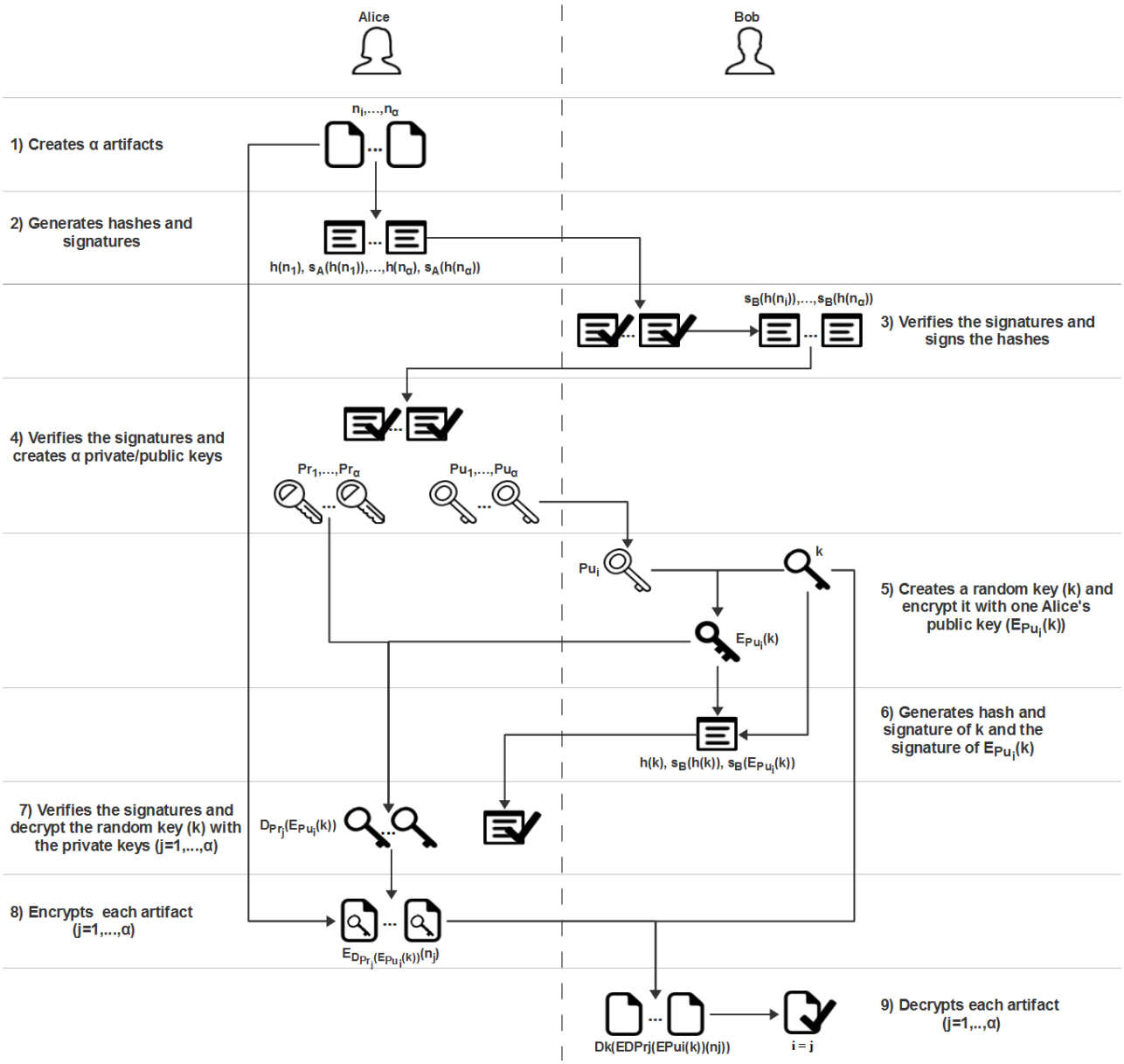


Fig. 1. The proposed fingerprinting protocol.

Next, we present a scheme for a simplified scenario of disclosure of some bits of $q$, allowing us to observe essential aspects of the protocol, which we have referred to as "partial transfer of knowledge". In the proposed scenario, Alice is not financially interested in the bits to be transferred: Alice is willing to reveal some bits of $q$ to anyone wanting to know them. On the other hand, Alice only agrees to reveal a certain subset of bits of $q$—by convention, we assume that Alice always reveal the most significant bits of $q$, although her choice is arbitrary. The fact that such set is predetermined makes it unnecessary to use *oblivious transfer* here. In such a scenario, Alice is able to show the most significant bits of $q$, proving to whom it may concern that, in fact, they are part of the bits of one of the factors of $n$. The scheme is simple and intuitive, and makes use of polynomial reductions and zero-knowledge proofs, based on the difficulty of factorization hypothesis. It is easy to verify that the proposed methods can be adapted to employ other classical problems that are notedly hard, such as the discrete logarithm.

Given a positive integer $n$ which is the product of two prime numbers $p$ and $q$, we want to show that a given sequence of bits $k$ corresponds to the most significant bits (or prefix) of $p$, without revealing any of the factors. The protocol is based, essentially, in the application of zero-knowledge schemes and polynomial transformations between variants of the integer factorization problem and variants of the boolean satisfiability problem.

*1) Reducing EQUICOMPOSITE to SAT:* Initially, we show the problem to determine if a number is composite can be easily reduced to the problem of determining if a logical expression is satisfiable, a problem well known as SAT [21]. More precisely, we consider the variant EQUICOMPOSITE of the factorization problem, where it is possible to determine

if an integer $n$ can be written as a product of two factors, each one with at most $\lceil log_2(n)/2 \rceil$ bits.

**EQUICOMPOSITE**
**Input**: binary number $n$, with $\lceil log_2(n) \rceil$ bits.
**Output**: YES, if $n$ is the product of two numbers with bit size up to $\lceil log_2(n)/2 \rceil$;
NO, otherwise.

To deal with the EQUICOMPOSITE problem using zero-knowledge proofs, we will study the implementation of variants of the multiplication operation using combinational circuits, or, equivalently, using logical expressions involving the bits of the operands.

*Product of integers as a logical function:* It is known that the product operation of two binary numbers can be described as a combinational circuit, being each digit of the result a logical expression on the digits of the operands. For the sake of completeness, a brief review about the theory behind it is given.

**Adding bits.** It is easy to implement a combinational circuit that receives as input two bits $A$ and $B$ (the operands) and a third bit $C_i$ (the "carry", defined in the previous stage), returning as output (i) the bit $S$ resulting from the sum of the three input bits, and (ii) a new "carry" bit $C_o$ (Figure 2).

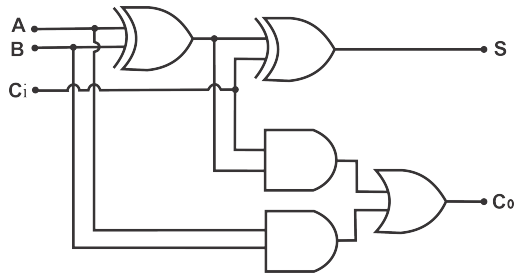


Fig. 2. A full adder.

Observe that both bits $S$ and $C_o$ can be described by logical expressions applied over the bits $A$, $B$ and $C_i$:

- $S = (A \oplus B) \oplus C_i$
- $C_o = (A \cdot B) + (C_i \cdot (A \oplus B))$

Naturally, the XOR ("exclusive or", denoted by $\oplus$) may be replaced by operations OR ($+$) and AND ($\cdot$), according to the formula $A \oplus B = \bar{A}B + A\bar{B}$.

**Chained full adders.** To do the sum of binary numbers with more than one bit, we need to chain full adders to one another, sending the output carry bit from one stage to the input of the next stage (Figure 3). One more time, each one of the output bits can be described as a logical expression applied over the input bits.

**Multiplying by a power of two.** The multiplication by two, in binary, can be performed as a simple left shit, adding a 0 bit as the least significant output digit. We write the left shift of $i$ bits (multiplication by $2^i$) applied to a binary number $B$ as $B << i$.

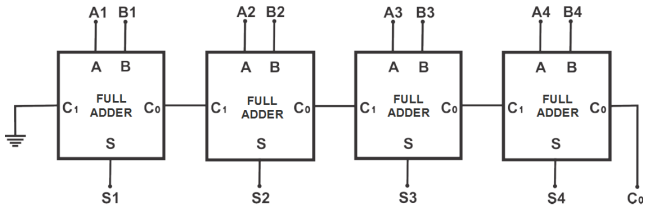


Fig. 3. A 4-bits adder.

**Obtaining the product of two binary numbers.** In a simplified way, the multiplication operation over binaries can be understood as a sequence of additions and multiplications by two. For instance, to multiply $A = A_3A_2A_1A_0$ by $B = B_3B_2B_1B_0$, we start with the rightmost bit of one of the operands, say, $A$. If the bit $A_0$ is 1, we add the value of the other operand, $B$, to the result $C$ (initially zero); if the bit $A_0$ is 0, no value is added. For each one of the consecutive bits $A_i$ of $A$, we perform a left shift of size $i$ on $B$ (i.e., we multiply $B$ by $2^i$) and we add it to the result if and only if $A_i = 1$. The result, written as a logical expression, is equivalent to $C = B \wedge A_0 + (B << 1) \wedge A_1 + (B << 2) \wedge A_2 + (B << 3) \wedge A_3$ (Figure 4).

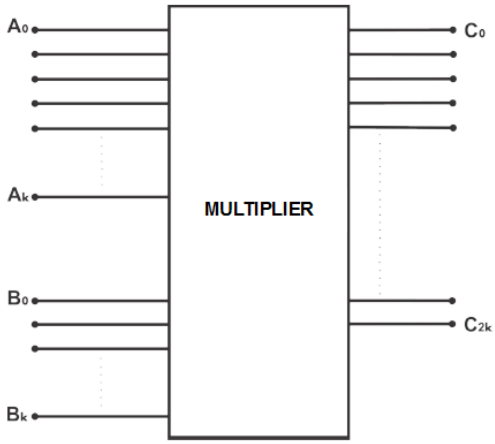


Fig. 4. A multiplier.

**Building a single output.** Knowing how to describe the product of two binary numbers in the form of a combinational circuit makes it is easy to adapt it to a modified circuit that has a single output bit whose value is 1 if and only if a certain number $n$ is equicomposite. To accomplish that, we need to add NOT ports to each output of the multiplier circuit related to one bit of $n$ that must be 0, and connect all the outputs to a single AND port.

Formally, a circuit UNI-MULT$(d, n)$, where $d$ is an integer and $n$ is a binary number, is built as shown in Figure 5, with a multiplier circuit of two binary numbers of $d$ bits, with a NOT port in each output of the multiplier, related to one bit 0 of $n$, and with an AND port connecting all the $2d$ outputs (inverted or not). Theorem 1, whose proof is quite simple, reads as follows.

*Theorem 1:* The circuit UNI-MULT$(d, n)$ returns the bit 1

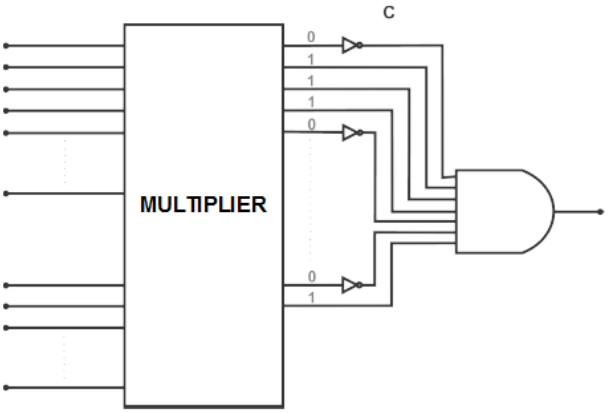


Fig. 5. UNI-MULT: fixing the output bits with a final AND port.

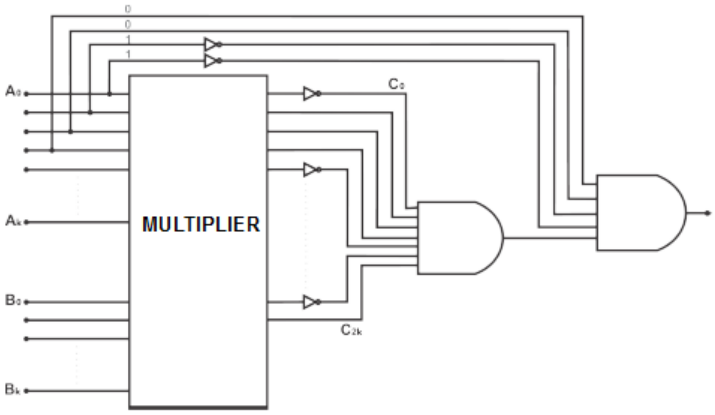


Fig. 6. PRE-MULT: fixing the first four bits of $A$ in "1100".

if and only if the binary number $n$ may be written as a product of two binary numbers of up to $d$ bits.

*2) The PREFACTOR problem:* Now, we consider the problem of determining if a number may be written as a product of two other numbers, and one of them is a set of bits whose values have been previously fixed. More precisely, consider the following decision problem, which we call PREFACTOR.

**PREFACTOR**
**Input**: binary numbers $n$ and $k$.
**Output**: YES, if $n$ is equicomposite and has a factor whose prefix is $k$;
NO, otherwise.

Knowing how to reduce EQUICOMPOSITE to SAT, it becomes simple to understand how to reduce the PREFACTOR problem to SAT. Hence, our goal is to determine whether a number is the product of two other numbers, and one of them starts with a predetermined set of bits. Our strategy is to build a circuit that is similar to the one in Figure 5, but with some of the bits shortcircuited directly into the last stage of the circuit, which receives an additional AND port as illustrated in Figure 6.

Formally, a circuit PRE-MULT$(d, n, k)$, where $d$ is integer and $n$ and $k$ are binary numbers, is built as shown in Figure 6. Initially, we have a circuit UNI-MULT$(d, n)$. For each input bit of the circuit UNI-MULT$(d, n)$ related with a bit of $k$, we derive it and connect it to a NOT port if such bit is 0 in $k$. The derivations are all connected to an AND port, as well as to the output bit of the circuit UNI-MULT$(d, n)$. The Theorem 2 wraps up the idea of the PRE-MULT circuit.

*Theorem 2:* The circuit PRE-MULT$(d, n, k)$ returns a bit 1 if and only if the binary number $n$ may be written as a product of two binary numbers up to $d$ bits, one of them having $k$ as its prefix.

*3) Converting to the conjunctive normal form:* The reader will observe, again, that the output of the circuit PRE-MULT$(d, n, k)$ is a logical function on the input bits. However, in order to use the framework of complexity theory and its polynomial reductions, it is necessary to have a logical expression in the conjunctive normal form. Fortunately, the transformations of Tseitin [22] allows us to build, from any logical expression $\sigma$, a new logical expression $\sigma'$ whose size is linear in the size of $\sigma$. Moreover, the transformation is executed in linear time in the size of $\sigma$.

*4) Using zero-knowledge proofs:* Knowing how to reduce the problem PREFACTOR to SAT, we can simply recur to zero-knowledge proofs with polynomial reductions. We can, for example, reduce a SAT instance to a 3-COLORING instance in polynomial time [23], and then use a classical scheme of zero-knowledge proof for this last problem [24].

### B. Fingerprint verification

Consider a fingerprint in a computer program codified as a subgraph of its control-flow graph. For an attacker who is not familiar with the subgraph location within the control-flow graph of the program, the task of removing the fingerprint is quite difficult, even if the attacker knows how the subgraph is generated. This happens because of the hardness of the isomorphism of subgraphs Graph Theory classic problem. However, once the seller exhibits it, revealing its location, the fingerprint removal becomes easy.

To demonstrate the authorship/ownership of the digital artifact based on the algorithm described in Section V-A, we will use the following strategy. First, we will codify an information regarding authorship/ownership in a binary number $k$. We select two prime numbers $p$ and $q$, and one of them has exactly $k$ as the most significant bits (prefix), and we compute the product $n$ of the numbers $p$ and $q$. The resulting product will be embedded within the digital artifact, appearing in the form of a *substring*, i.e., a subsequence of bits (more precisely, appearing as a substring of the bit sequence obtained from the digital artifact after the execution of the extractor algorithm). The motivation for this strategy is the fact that we can show $k$ without being necessary to reveal the location of $n$.

Now, we will consider a slight variation of the extractor algorithm, which we call *pre-extractor*. This algorithm, instead of returning exactly the fingerprint (previously embedded by the embedder algorithm), returns a sequence of bits—possibly

a long one—that contains the fingerprint as a substring. More precisely, the substring will be, as we have seen, the product of two prime numbers, one of them having the fingerprint as a prefix.

*1) The problem SUBSTRING-PREFACTOR:* Consider the following decision problem.

**SUBSTRING-PREFACTOR**
**Input**: binary numbers $d$ and $k$, and an integer $N$.
**Output**: YES, if there is a substring $n$ of $d$, with $N$ bits, such that $n$ is equicomposite and one of its factors has $k$ as prefix; NO, otherwise.

It is easy to see the problem SUBSTRING-PREFATOR can also be reduced to SAT. In fact, we need to build a circuit PRE-MULT (similar to the circuit built in the PREFACTOR problem) for each substring of size $N$, and to apply an OR port to the outputs of each one of the $bitsize(d) - N + 1$ circuits.

*2) Generating the fingerprint:* The generation of the fingerprint follows. Given an information $m$ to be embedded, we need to generate two random prime number $p$ and $q$ of the same bit size, such that $m$ is the prefix of $p$, and compute the product $n = p \cdot q$, the sequence of bits that will be embedded in the digital artifact. The generation of $q$ may follow the traditional methods for picking random numbers and testing primality until one gets a prime number. The generation of $p$ follows a slightly modified approach: a random number is generated and concatenated to the right of $m$, only then to test primality, repeating the process a prime number ensues.

*3) Embedding the fingerprint:* The process of embedding the fingerprint aims at modifying the digital artifact, making the sequence of bits $n = p \cdot q$ appears as a substring of the string retrieved by the extractor algorithm. In practice, the exact embedding process—and the extraction process—depend of the digital artifact type. In case of a text file, for instance, the process only depends on the coding scheme used to embed the information in the text. In Section III-A, we describe a scheme in which the sequence of "though" and "although" words determine the codified information. In this case, it would suffice to select a sequence of these words and replace them appropriately to include the bits constructed by the protocol. Similar examples could be built for other application fields.

*4) Verifying the fingerprint:* The fingerprint verification comprises the following steps:

1) Extraction of the sequence of bits embedded in the digital artefact—such sequence may be very long, but it contains $n = p \cdot q$ as a substring.
2) Transformation of the generated sequence into an instance of SAT, and then to an instance of 3-COLORING.
3) Use of the zero-knowledge scheme to demonstrate that the graph obtained in the previous step is 3-colorable.

The extraction of the sequence of bits which we refer to in the first step can be done by specific algorithms, which must be defined for each field of application and their corresponding digital artifacts. The transformation to an instance of SAT is attained precisely by the algorithm described in Section V-A, while the polynomial reduction from SAT to 3-COLORING is well known [23]. An interactive proof scheme of zero-knowledge for 3-COLORING is described on [24] and can be used in the last stage to exhibit the fingerprint without revealing $n$ or any of its factors.

## VI. Conclusions

We described a fingerprinting protocol which can be appropriate in dispute scenarios related to intellectual property. It is balanced—or *fair*—in the sense that it assigns no advantage to either party (the "buyer" or the "seller"). The proposed protocol assumes the existence of watermarking schemes that meet the usual requirements of stealth, resilience and verifiability.We also describe a secure way for verifying the fingerprint without exposing its contents or location. This is achieved via partial transfer of knowledge.

An important observation on the proposed protocol is the fact that, in a dispute scenario, the arbitrator will have access to all data transmitted during the execution of the protocol. That is not necessarily a limitation. First, all cryptographic keys (secret, private and public) disclosed to the arbitrator are temporary, used only in one execution of the protocol. Furthermore, since the artifact has already given rise to a judicial dispute, it means it has already been improperly distributed, anyway. Thus, it should not be a problem to disclose its fingerprint details to another individual, namely the arbitrator.

In future works, it should be interesting to investigate the possibility of using zero knowledge proofs to convince the arbitrator that there had been generated versions of the artifact with different fingerprints, with no need to present each of the actual traceable artifact versions.

## References


[1] M. O. Rabin, "How to exchange secrets by oblivious transfer," Harvard Aiken Computation Laboratory, Tech. Rep., 1981.

[2] W. Bender, D. Gruhl, and N. Morimoto, "Techniques for data hiding." in *Storage and Retrieval for Image and Video Databases (SPIE)*, 1995, pp. 164–173. [Online]. Available: http://dblp.uni-trier.de/db/conf/spieSR/spieSR95.html#BenderGM95

[3] I. Cox, M. L. Miller, and J. A. Bloom, *Digital Watermarking*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[4] M. J. Atallah, V. Raskin, M. Crogan, C. Hempelmann, F. Kerschbaum, D. Mohamed, and S. Naik, "Natural language watermarking: Design, analysis, and a proof-of-concept implementation." in *Information Hiding*, ser. Lecture Notes in Computer Science, I. S. Moskowitz, Ed., vol. 2137. Springer, 2001, pp. 185–199. [Online]. Available: http://dblp.uni-trier.de/db/conf/ih/ihw2001.html#AtallahRCHKMN01

[5] M. J. Atallah, V. Raskin, C. Hempelmann, M. Karahan, R. Sion, U. Topkara, and K. E. Triezenberg, "Natural language watermarking and tamperproofing." in *Information Hiding*, ser. Lecture Notes in Computer Science, F. A. P. Petitcolas, Ed., vol. 2578. Springer, 2002, pp. 196–212. [Online]. Available: http://dblp.uni-trier.de/db/conf/ih/ih2002.html#AtallahRHKSTT02

[6] W. Zhu, C. D. Thomborson, and F.-Y. Wang, "A survey of software watermarking." in *Proc. IEEE Int'l Conference on Intelligence and Security Informatics*, ser. ISI'05, P. B. Kantor, G. Muresan, F. S. Roberts, D. D. Zeng, F.-Y. Wang, H. Chen, and R. C. Merkle, Eds., vol. 3495. Springer, 2005, pp. 454–458. [Online]. Available: http://dblp.uni-trier.de/db/conf/isi/isi2005.html#ZhuTW05

[7] J. Hamilton and S. Danicic, "A survey of static software watermarking," *Proc. World Congress on Internet Security*, pp. 100–107, 2011.

[8] R. L. Ostergard, "The measurement of intellectual property rights protection," *Journal of International Business Studies*, vol. 31, no. 2, pp. 349–360, 2000. [Online]. Available: http://EconPapers.repec.org/RePEc:pal:jintbs:v:31:y:2000:i:2:p:349-360

[9] C. Collberg and J. Nagra, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*, 1st ed. Addison-Wesley Professional, 2009.

[10] S. Craver, N. D. Memon, B.-L. Yeo, and M. M. Yeung, "Resolving rightful ownerships with invisible watermarking techniques: limitations, attacks, and implications." *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 4, pp. 573–586, 1998. [Online]. Available: http://dblp.uni-trier.de/db/journals/jsac/jsac16.html#CraverMYY98

[11] A.-R. Sadeghi and A. Adelsbach, "Advanced techniques for dispute resolving and authorship proofs on digital works," in *Security and Watermarking of Multimedia Contents V*, 2003.

[12] S. Haber and W. S. Stornetta, "How to time-stamp a digital document." *J. Cryptology*, vol. 3, no. 2, pp. 99–111, 1991. [Online]. Available: http://dblp.uni-trier.de/db/journals/joc/joc3.html#HaberS91

[13] C. Liem, Y. X. Gu, and H. Johnson, "A compiler-based infrastructure for software-protection," in *Proceedings of the Third ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, ser. PLAS '08. New York, NY, USA: ACM, 2008, pp. 33–44. [Online]. Available: http://doi.acm.org/10.1145/1375696.1375702

[14] G. Brassard, C. Crpeau, and J.-M. Robert, "Information theoretic reductions among disclosure problems," in *FOCS*. IEEE Computer Society, 1986, pp. 168–173. [Online]. Available: http://dblp.uni-trier.de/db/conf/focs/focs86.html#BrassardCR86

[15] C. Crpeau, "Equivalence between two flavours of oblivious transfers." in *CRYPTO*, ser. Lecture Notes in Computer Science, C. Pomerance, Ed., vol. 293. Springer, 1987, pp. 350–354. [Online]. Available: http://dblp.uni-trier.de/db/conf/crypto/crypto87.html#Crepeau87

[16] C. Crpeau and J. Kilian, "Weakening security assumptions and oblivious transfer (abstract)." in *CRYPTO*, ser. Lecture Notes in Computer Science, S. Goldwasser, Ed., vol. 403. Springer, 1988, pp. 2–7. [Online]. Available: http://dblp.uni-trier.de/db/conf/crypto/crypto88.html#CrepeauK88

[17] M. Bellare and S. Micali, "Non-interactive oblivious transfer and applications," in *Advances in Cryptology CRYPTO 89 Proceedings*, ser. Lecture Notes in Computer Science, G. Brassard, Ed. Springer New York, 1990, vol. 435, pp. 547–557. [Online]. Available: http://dx.doi.org/10.1007/0-387-34805-0_48

[18] B. Schneier, *Applied cryptography*. Wiley New York, 1996.

[19] L. Bento, D. Boccardo, R. Machado, V. de S, and J. Szwarcfiter, "A randomized graph-based scheme for software watermarking," in *Proceedings of Brazilian Symposium in Information and Computater Systems Security*, ser. SBSEG. SBC, 2014, pp. 30–41.

[20] J. Kilian, *Uses of randomness in algorithms and protocols*. MIT Press, 1990.

[21] T. J. Schaefer, "The complexity of satisfiability problems," in *10th annual ACM symposium on Theory of Computing*. ACM, 1978, pp. 216–226.

[22] G. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of Reasoning*, ser. Symbolic Computation, J. Siekmann and G. Wrightson, Eds. Springer Berlin Heidelberg, 1983, pp. 466–483. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-81955-1_28

[23] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. New York: Plenum Press, 1972.

[24] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, ser. STOC '85. New York, NY, USA: ACM, 1985, pp. 291–304. [Online]. Available: http://doi.acm.org/10.1145/22145.22178