Fair fingerprinting protocol for attesting software misuses

Davidson R. Boccardo
Raphael C. S. Machado
Instituto Nacional de Metrologia, Qualidade e Tecnologia

Vinícius G. Pereira de Sá
Jayme L. Szwarcfiter
Universidade Federal do Rio de Janeiro

***Abstract*—Watermarking schemes allow for the embedding of information into digital artifacts, in such a way that the functionalities of the host artifact remain unchanged. A watermark aimed at uniquely identifying the digital artifact in which it is embedded is referred to as a fingerprint. Fingerprinting mechanisms are important tools in the design of methods to protect the intellectual property. In this paper, we present a formal definition of the concept of digital watermarks, a protocol — associated with a fingerprinting method — to solve disputes around intellectual property breach, and a secure verification protocol for software fingerprinting.**

I. Introduction

Consider the following scenario. Alice wishes to sell a computer program to Bob, but suspects Bob may distribute the program to others. To prevent the illegal program redistribution by Bob, Alice codifies, within the program, the following sentence: "This computer program is for the exclusive use of Bob, and cannot be redistributed to others." Alice, cautiously, utilizes cryptography and digital watermarking techniques to codify the sentence in a certain way to avoid the identification and removal of the sentence within of program by Bob.

A few months later, Alice discover Eva is using an excellent computer program she had found on Internet. Analyzing the program, Alice discovers the program is the exactly the version sold to Bob, and decides to take the case to court. During the trial, Alice presents the selling contracts of the computer program, and presents the version utilized By Eva, together with the codified sentence that exclusively identifies Bob. However, Bob reaffirms his access to the computer program, but denies the fact he redistributed it to Internet. Moreover, he adds the argument Alice could have inserted that sentence in any program, and herself, distributed the program in the Internet. The judge agrees with Bob's argument and decides to dismiss the case for lack of evidence.

The method for "codifying information within a computer program" is a use case of *digital watermarking*. Watermarking schemes are notable tools for protection of intellectual property, allowing to embed in a digital artifact information about authorship, ownership, or even, uniquely identifiers of the own artifact. Although, the current watermarking schemes does not allow the attribution guilty in cases of misuse of a artifact. As seen in the example above, the suspicion can fall at both sides involved in the dispute.

Digital watermarking techniques allow to embed in digital artifacts information that may be after recovered. The addition of a watermark should not impact in the artifact semantics; while its removal must be a difficult task to be performed by third parties, and any attempt will likely lead to an artifact deterioration or destruction. Currently, many researches are carried out in distinct application fields (see Section II), aiming at the protection of artistic, industrial and intellectual property. Although, the resilience of such watermarks against attacks still lacks from a more strict formalization, it is undeniable the existence of watermarking schemes that offer, in practice, an acceptable level of security. On the other hand, it is important to conduct studies on the use of watermarks in the chain of security protocols, assuming by hypothesis, the existence of such watermarks with the desirable properties.

In the present work, we consider the use of watermarks in the context of intellectual property protection. In such a scenario, the watermark to be embedded in a digital artifact carries an information that allows to distinguish uniquely that artifact, enabling its subsequent traceability in case of misuse conditions. In this specific context, a watermark is termed *fingerprint*, and the artifact that carries it is *traceable* (in opposition of an artifact without watermark, that is *untraceable*).

A limitation in the classical model of fingerprinting algorithms is its inherent asymmetry, or bias, whereby the side responsible for the fingerprint addition will be in advantage since he/she has access to both artifact versions: the traceable (with the embedded fingerprint) and the untraceable (without the fingerprint). Such knowledge gives the advantage, to a malicious participant, to choose which version will be used in a trial for misuse of the artifact; naturally, the malicious participant will choose the version that does not incriminate him. The problem can be solved using a Trusted Third Party (TTP), responsible for including the fingerprint that identifies the artifact. However, when only two parties are participating in the protocol, one will need to be in charge of embedding the watermark, eventually taking advantage to the other party.

In the present paper, we describe a protocol involving two participants: the seller, which we call Alice, and the buyer, which we call Bob. The protocol will be built in a way the Alice sends to Bob a traceable artifact, but whose fingerprint even she will not be able to know a posteriori. Such property is achieved through modifications in the oblivious transfer protocol [1]. With such protocol, it is possible to determine with an error probability arbitrarily low, the real responsible for the misuse of the artifact. We also describe a secure verification protocol for attesting software misuses without needing to reveal the fingerprint contents/location even to the arbitrator.

The paper is structured in the following way. In Section II,

we describe known approaches to support the intellectual property protection. In Section III, we present formal concepts associated to watermarks and based on them we describe the protocol in Section IV. In Section V, we describe a way to secure the fingerprint contents/location even in a trial through a partial transfer of knowledge scheme. The Section VI contains our concluding remarks.

II. Related Works

In the literature, there are a plenty of researches with respect to methods aiming at embedding watermarks in distinct types of digital 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 is observed the the development of most fruitful approaches dealing with watermarking schemes, as we described in the following.

Methodologies for protection of software intellectual property, in general, range from legal protection to technological protection. From the view point of legal protection, government and industry seek regulatory mechanisms related to industry and trade — such as patent laws, copyrights and trade secret — to reduce the incidence of piracy and software counterfeiting [8]. From the view point of technological protection, several techniques have been developed to avoid reverse engineering, tampering and illegal distribution of software [9].

Obfuscation techniques aim at making the software reverse engineering a difficult task. They are based on semantics-preserving transformations, deteriorating results provided by software analysis tools [9]. Tamper-proofing techniques aim at detecting an unauthorised software modification and responding against the modification. The detection methods are based on code introspection, state inspection or environment verification. The responding methods vary, but the common ones are program termination, performance degradation of the program or program restore [9].

Software watermarking techniques aim at inhibiting the piracy, detecting and tracing ilegal distributions. They can be classified based on their embedder and extractor algorithms, and in their static or dynamic nature. Static watermarks are within the code or data segment of a program code; and dynamic watermarks are built in some state of the program code execution. Embedding algorithms are typically based on code substitution, code reordering, register allocation, control flow graphs, abstraction interpretation and opaque predicates. A survey of watermarking techniques can be found on [6], [7].

Although the focus of this work is based on the watermark use, it is independent of the embedding and extracting algorithms. The concern of this work is to support the dispute resolving. In a dispute resolving, two or more participants claim authorship or ownership of a digital artifact. The goal of such scheme is to allow a third trustable party solves the dispute by comparing the proofs presented by the participants. There are distinct works that deal with this problem for audio, video and image artifacts [10], [11]. Similarly, our proposal deals with the dispute resolving problem, providing specific inputs to encounter the responsible for breaching an intellectual property, by using the evidences provided by the concerned parties (on the one hand, the defendant, on the other, the owner or author of the program). To our knowledge, our study is the first dealing with this problem in the software context; the first dealing with the fairness between the parties, due to the fact that the party responsible to embed the watermark has both versions of the artifact, traceable and untraceable; and the first providing a scheme that allows dispute resolution by an arbitrator that do not need to have access to the watermark location (say, a partially trustable arbitrator).

III. Background

A. Watermarks

Watermarking schemes refer to techniques which allow to embed an information in an artifact, such that once embedded, its removal becomes "unfeasible" by third parties. These embedded information are utilized, generally, for authorship/ownership identification, or even, to attest a unique identity to an artifact. This latter case is the one considered in the present work, and it is called *fingerprinting*.

The basic properties of a watermark are stealthy, resilience, and verifiability, described briefly below:

- Stealthy. Stealthy refers to the property of a watermarked artifact be indistinguishable of an artifact without the watermark.
- Resilience. Resilience refers to the property of a watermark be hard to removal or tamper with, as otherwise will compromise the artifact functionality.
- Verifiability. Verifiability refers to the property of a watermark be exhibit to third parties, attesting the authorship/owner of an artifact.

Watermarks are based on the embedding of a *content-information* in a digital artifact — the *host* – in such a way that the final artifact — which we call *product* — be semantically equivalent to the host, i.e., it does not affect the program functionality.

Essentially, a watermarking scheme is defined by two algorithms. The first, called *embedder*, takes as input an digital artifact u and a content-information m to be embedded, and outputs a product $\tilde{u}$, semantically equivalent to host, but which "contains", the information m embedded. The second, called *extractor* takes as input the watermarked artifact $\tilde{u}$, which contains the embedded information m, and outputs m. We will see it is convenient to assume that both algorithms: embedder and extractor receive an additional secret information that determines not only one transformation for embedding and one for extracting, but also distinct transformations.

Formally, given a universe of digital artifacts U (denoting the role of hosts) provided with a definition of equivalence class, 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 $a \in A$, and outputs an artifact $\tilde{u} \in U$ semantically equivalent to u.

The *extractor function* $g : U \times B \rightarrow M$, takes as input a digital artifact $\tilde{u} \in U$ and an auxiliar input $b \in B$, outputting a content-information $m \in M$.

The bijective function $r : A \rightarrow B$ 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 the brute-force attack in recovering the embedded information in a digital artifact. The addition of auxiliary inputs — tipically associated to the content-information location — allows more than one watermark in the same artifact. This enables the use of temporal protocols [12], for instance, in the watermark, increasing its robustness against *addition attacks*. In such attach an adversary embeds an additional watermark to confuse the original watermark during a trial.

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 which brings itself, the content-information. In order to say "same meaning or utility", it is necessary to define the "semantic" related with the host artifact, and such should be preserved in the "immersion" process of content-information. In practice, we will work with "equivalence classes" between semantically equivalent artifacts.

Consider the set U of artifacts of certain kind involving 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. So, we say the artifacts of each U_i are *semantically equivalent*. Observe that the semantic equivalence must be defined regarding its application.

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 by replacements between the words "however" and "although", or by separations of sequence periods, such as "period" and "semicolon". 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 program by obfuscations means [13]. The definition of semantically equivalent classes depends, mainly, of the application field.

Resilience: A watermarking scheme should allow to embedded an information in a "resilient" manner in a digital artifact. Intuitively, only with the product (digital artifact with embedded content-information), a watermarking scheme should guarantee the watermarking removal, preserving the "utility" or "meaning" of the digital artifact, is impracticable. In other words, any "efficient" algorithm that removes the watermarking will generate, with high probability, an artifact belonging to a distinct semantically equivalent class.

Formally, consider X a removal algorithm of watermarks, i.e., a procedure that takes as input a watermarked artifact $\tilde{u}$ and an auxiliary input b such that $g(\tilde{u}, b) = m$, and outputs an artifact u' without the watermark (or with a corrupted watermark). For any probabilistic algorithm of polynomial time X, a watermarking scheme (f, g, r) should guarantee the probability of X to return an artifact semantically equivalent to u without the watermark m, is neglected.

Although there are no watermarking schemes provably secure according with the above definitions, we assume the existence, hypothetically, of such schemes with the goal of developing a fair fingerprinting protocol for intellectual property applications.

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., the way of "exhibiting" a watermark preserving its contents/location secret. Digital artifacts are easily modified and counterfeited, so an ingenious designed watermark may be easily removed if an adversary knows its contents/location.

The verifiability property is one of the challenges for the 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.

B. Fingerprinting

Fingerprinting refers to the act of assigning "identity" to an artifact. This may be done attaching a simple identifier — for instance, a sequence number or a hash — to the artifact. In practice, methods more sophisticated — aiming at the resistance against removal attacks — are desirable in real applications of intellectual property protection.

Frequently the identification of an artifact by means of fingerprinting aims at the identification of artifact misuses by third parties. In this context, it is reasonable to conceive a attack model in which the buyer of a fingerprinted artifact will try to remove the identifier of it, with the goal of redistribution in an untraceable manner. For such reason, it is expected a fingerprinting method has exactly the same properties of a watermarking method. In practice, the robust fingerprinting methods are exactly the ones based on watermarking algorithms, with the difference the embedded information is not related with the authorship — but with information related to specific artifact or with the buyer.

Formally, we can define a *fingerprinting scheme* in the same way of a watermarking scheme, with the same embedded and extractor algorithms, and satisfying the same requirements of stealthy, resilience and verifiability, defined in Section III-A.

Fingerprinting are interesting tools in scenarios of intellectual property protection, in the way they embed on a digital artifact information that allows the traceability of the responsible for "misusing" that artifact. Typically, the fingerprinting use

[1]Dummy codes are code instructions that do not interfere in the program semantics.

is exactly the one described in the beginning of the paper: the artifact seller embeds an information that identifies the artifact buyer. In this way, in a scenario of illegal redistribution, it is impossible for a third party to identify which party (seller or buyer) is responsible for the artifact misuse. A natural approach to solve the "unfairness" between the parties consists in using a TTP: a third trust party is responsible to embed a watermark that unique identifies each "transaction" of a artifact distribution, and the fingerprinted artifact is delivered only to buyer (the seller does not have access to the fingerprinted artifact). In the case of a misuse of the fingerprinted artifact, the responsibility certainly will be assigned to buyer.

We describe a protocol based on the oblivious transfer protocol, dismissing the need of a TTP and still making the seller not aware — at least with a high probability — of the artifact in possession of the buyer.

C. Oblivious transfer

Suppose Alice is a distributor of artistic content — say, a book and Bob is a client wanting access to a certain content. However, Bob would like to keep confidential the item of his interest, either for privacy reasons or to avoid receiving future unwanted advertising. The cryptographic protocol called oblivious transfer allows to transfer the content from Alice to Bob in such way that Alice has no idea about which content was transferred.

Formally, assume Alice defines $M_1, ..., M_k$ messages and Bob wants the i-th message of Alice. Essentially, Alice transfer 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 the protocol operation are described in Section IV.

Several oblivious transfer models are encountered in literature. Some of them are non-interactive [14]: Alice can simply publish her messages and Bob will understand only one of them. Although there are several kinds of oblivious transfers described in literature — I have two secrets and you obtain one of them with probability of $1/2$, I have n secrets and you obtain one of them with probability of $1/n$ — all of them are equivalent [15], [16], [17]. Oblivious transfer is a important block for building several protocols [18]. It will also be modified, in Section IV, to build a traceability (fingerprinting) protocol of digital artifacts.

IV. Proposed fingerprinting protocol

As discussed along the paper, the great difficulty in conceiving a balanced (fair) fingerprinting protocol without recurring to TTP is due the fact of the person responsible for inserting the watermark will have necessarily knowledge of both versions of the digital artifact: "traceable" and "untraceable". If this person is malicious — for example, he is intended to distribute copies of the digital artifact — he may choose to distribute the version that does not incriminate him. If the watermark is inserted at the "origin", then the seller can insert the watermark, transmit the traceable artifact to the buyer, and distribute copies of the "traceable" artifact. If the watermark is inserted at the "target", then even if the buyer insert the watermark in the artifact, he will certainly distribute copies of the "untraceable" artifact.

The idea to circumvent the apparent paradox described above consists in a modification of the oblivious transfer protocol. Essentially, we take the oblivious transfer protocol 1 of n, with a sufficiently large n, where one of n possible messages is transmitted, but the seller is unaware of which one was transmitted. So, although the seller has been responsible for inserting the watermark, he does not know which version was actually received by the buyer. Thus, if he chooses to distribute (maliciously) any version, then with a high probability he will distribute a version that was not the one sent to the buyer. And if he distributed all the versions it will be possible during the trial to impute the blame on the seller since more than one version may be found on Internet. It takes several adjustments to the protocol to prevent non-repudiation attacks. We describe in the following such adjustments, starting from a basic fingerprinting protocol based on the oblivious transfer protocol in Section IV-A, to a resistant version against attacks in Section IV-C.

A. Initial fingerprinting protocol based on oblivious transfer

The most naive version of the fingerprinting protocol makes use of the classic version of the oblivious transfer protocol, which allows the transference of an element of a set without the knowledge of what element was transmitted. We start with this basic version to better understanding of the proposed modifications, and we evolve to the final protocol step by step according to possible attacks.

Basic fingerprinting protocol

1) Alice creates α semantically equivalent variations $n_1, ..., n_\alpha$ of a digital artifact.
2) Alice creates α 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 α 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 α private keys, obtaining $D_{Pr_j}(E_{Pu_i}(k))$, with $j = 1, ..., \alpha$.
5) Alice encrypts each artifact n_j among its α 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 $i = j$ he will have a consistent digital artifact.

The key for understanding the above protocol is to note that, in the Step 4, Alice will obtain α "possible keys", but only one of which will be the key k of Bob, however, it is not possible to Alice to know what is the key.

To identify responsibilities for improper use of digital artifacts, the arbitrator will need access all information generated during the protocol steps: variants of the digital artifacts; and public, private and private keys. The verification protocol, to be

executed by a 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 any of the artifacts $n_1, ..., n_\alpha$.

Observe that is still necessary to guarantee Alice, in fact, generated α artifacts with distinct fingerprints, and 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 enables a simple attack by Alice. Alice can generate α variants of the digital artifact containing all the same fingerprint. This allows her distributing any of the artifacts and imputing blame on Bob. To avoid this attack, the arbitrator must verify that Alice, in fact, generated α artifacts with distinct fingerprints. A recent work about the generation and verification of distinct fingerprints based on a randomised graph-based scheme can be found on [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 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 very high probability, it will be a distinct artifact that one obtained by Bob. However, there is no guarantee that the artifacts shown to arbitrator are, in fact, the artifacts involved in the protocol. At this point, it is clear the distinction between the objectives of the proposed protocol and the objectives of the classic oblivious transfer protocol. In the basic protocol, there is only an interest of transferring an element of a set from Alice to Bob, without Alice knowing the transferred element. In the resistant version of the fingerprinting protocol, it is fundamental that it can be demonstrated later all the elements involved during the execution of the protocol, being important an arbitrator to know *what elements might have been transferred to Bob*. This need will demand some modifications in the classical protocol.

In order to avoid the attack of identical fingerprints, the protocol should encompass actions to verify if Alice, in fact, generated α variants of the artifact with different fingerprints. To guarantee that, the modified protocol includes the sending of cryptographic hashes of the artifacts by Alice to Bob.

Fingerprinting protocol with guarantee of distinct fingerprints

1) Alice creates α 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 α 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 α 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 α private keys, obtaining $D_{Pr_j}(E_{Pu_i}(k))$, with $j = 1, ..., \alpha$.
7) Alice encrypts each artifact n_j among its α 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 allows to guarantee 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 his verification algorithm. The Step 3 indicates Bob had knowledge of the artifact 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 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 version 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 to Bob for a possible artifact misuse. Obviously, this identification must occur latter, i.e., with the arbitrator. Although, the inputs for this identification must be provided by Alice. In practice, the proposed solution consists in Bob to sent to Alice a cryptographic hash of his secret key, together with the digital signature. This modification can be seen in Step 6.

Resistant protocol version against non-repudiation attacks

1) Alice creates α 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 α 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 α 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 α private keys, obtaining $D_{Pr_j}(E_{Pu_i}(k))$, with $j = 1, ..., \alpha$.
8) Alice encrypts each artifact n_j among its α 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 $s_B(E_{Pu_i}(k))$. It also need access to all α public keys and α private keys generated by Alice.

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 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 Bob given, in fact, the same private key encrypted with one of the Alice's public keys during the execution of the protocol. With the informations above, the arbitrator is capable to identify the version n_i obtained by Bob — testing each Alice's private key — and, finally, to verify if the fingerprint $f(n_i))$ is the same as the fingerprint $f(\tilde{n})$. Figure 1 sums up the proposed fingerprinting protocol.

V. Secure verification protocol for software fingerprinting

In this section, we develop a protocol that allows the fingerprint verification during a trial without revealing its contents/location, even to the arbitrator. The simple exhibition of its contents/location makes easy to an adversary/attacker to tamper with it. The protocol development starts from a scheme of partial transfer of knowledge and after we describe its application in the scenario of secure verification of fingerprints. The main advantage of the use of partial transfer of knowledge scheme is the possibility to reveal information about authorship/ownership embed exactly in the bits to be shown, without making easier for an attacker to extract the content/location of the fingerprint. As the transfer is partial, an attacker willing to remove the fingerprint will not have enough information to locate within of the artifact, in a way its removal will remain so hard after verification as 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 of 500 bits which is know 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 bits to Bob. Lilian proposes a method that allows Bob to be sure that Alice indeed knows one of the factors of n, and it still allows that assurance happens upon individual bits of q. 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 also ensures that will not be disclosure without the consent of Alice. Finally, it allows the use of *oblivious transfer* [21] in a way Alice is unaware of bit sets actually disclosed.

In the following, we present a scheme for a simplified scenario of disclosure of some bits of q, allowing to observe essential aspects which we denoted "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. In the other side, Alice only agrees to reveal a certain subset of bits of q — by convention, we assume Alice always reveal the most significant bits of q, although her choice is arbitrary. The fact the set is predetermined precludes the use of *oblivious transfer*. In such a scenario, Alice is able to show the most significant bits of q, proving 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 factoring hypothesis. It is easy to verify the proposed methods can be adapted to classical problems notedly hard, such as discrete logarithm.

Given a positive integer n product of two prime number 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 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, problem well-known as SAT [22]. More precisely, we consider the variant EQUICOMPOSITE of the factorization problem, where it is possible to determine if a integer n can be written as a product of two factors, each one with the most $\lceil log_2(n)/2 \rceil$ bits.

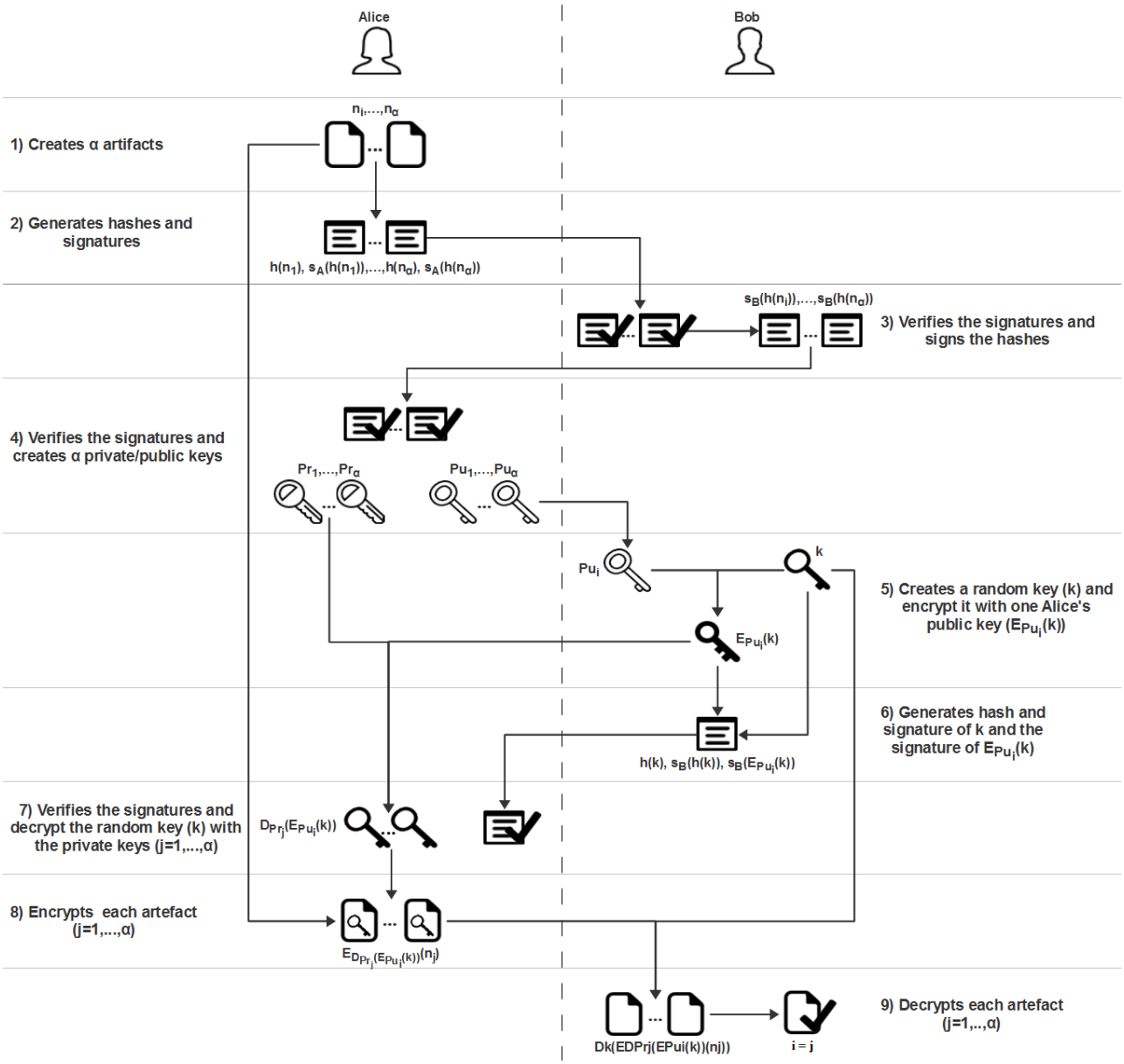

Fig. 1. Proposed fingerprinting protocol.

EQUICOMPOSITE
Input: binary number n, with $\lceil log_2(n) \rceil$ bits.
Output: YES, if n is the product of two number 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, equally, using logical expressions involving the bits of the operands.

Product of integers as a logical function: It is known the product operation of two binary numbers can be described as a combinational circuit, being each digit of the result a logical expression upon the digits of the operands. For the sake of completeness, lets briefly reviews the theory behind it.

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 (generated by a adder in the previous stage), and returns as output the bit S resulting of the sum of the three inputted bits and a new bit of carry C_o (Figure 2).

Observe that both bits S and C_o can be described as a logical expression 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

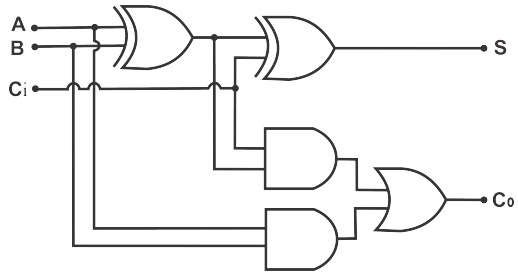

Fig. 2. Full adder.

with more than one bit, we need to chain full adders, and sends the output carry bit from one stage to the input of the next stage (Figure 3). One more time, each one of the output

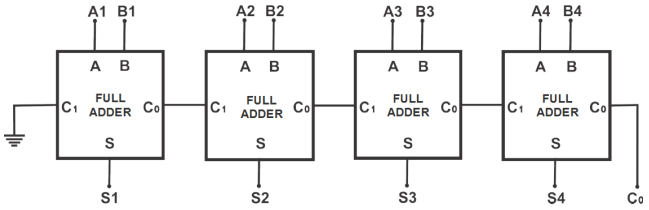

Fig. 3. 4-bits Adder.

bits can be described as a logical expression applied over the input bits.

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

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 most right bit of one of the operands, we say, A. If the bit A_0 is 1, so 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, if and only if $A_i = 1$, we do a shift left of size i on B (i.e., we multiply B by 2^i) and we add it to the result. The result, written in 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).

Building a single output. Knowing how to describe the product of two binary numbers in the form of a combinational circuit, 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. So, 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 tone bit 0 of n, and with a AND port connecting all the $2d$ outputs (inverted or not). The Theorem 1 whose proof is simple has the following meaning.

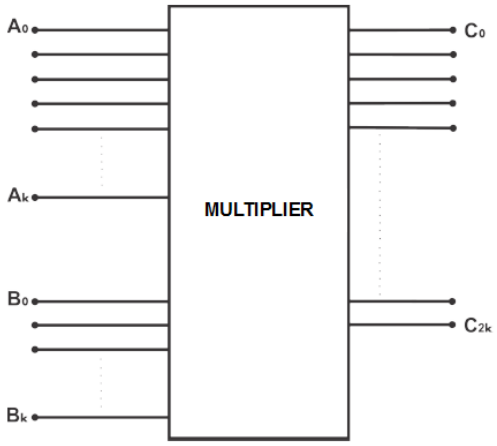

Fig. 4. Multiplication.

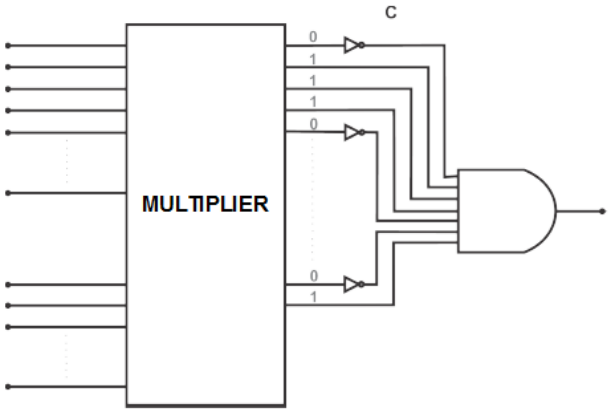

Fig. 5. UNI-MULT: fixing the output bits with a final AND port

Theorem 1: The circuit UNI-MULT(d, n) returns the bit 1 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 are previously fixed. More precisely, consider the following decision problem, which we call PREFACTOR.

PREFACTOR
Input: binary numbers n and k.
Sada: YES, if n is equi-composite and has one factor being k its prefix ;
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 if a number is the product of two other numbers, and one of them starts with a prefixed set of bits. Our strategy is to build a similar circuit with the Figure 5 but with the "transfer" of some input bits directly to the last stage of the circuit, which receives an additional AND port as illustrated in Figure 6.

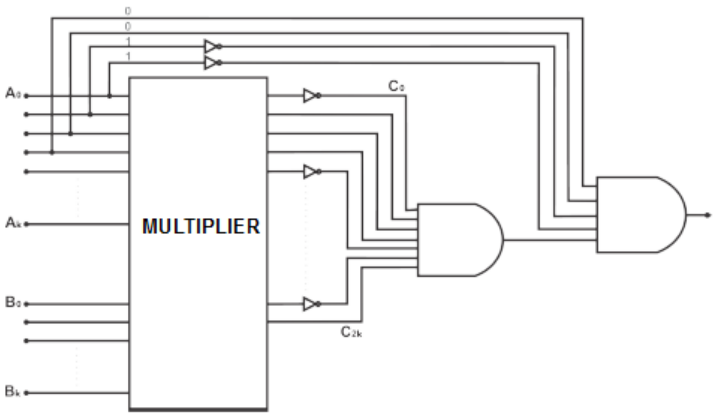

Fig. 6. PRE-MULT: fixing the first four bits of A in "1100"

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 a AND port, as well the output bit of the circuit UNI-MULT(d, n). The Theorem 2 sums up what the circuit PRE-MULT allows to do.

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

3) Converting to the conjunctive normal form: The reader will observe, again, the output of the circuit PRE-MULT(d, n, k) is a logical function upon 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 [23] allows to build, from any logical expression σ, a new logical expression σ' whose size is linear in the size of σ. Moreover, the transformation is executed in linear time in the size of σ.

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, to reduce a SAT instance to a 3-COLORING instance in polynomial time [24], so then to use a classical scheme of zero-knowledge proof for this last problem [25].

B. Fingerprint verification

Consider a fingerprint in a computer program codified as a subgraph of its control-flow graph. For an attacker who are not familiar with the subgraph location within control-flow graph of the program, the task to remove the fingerprint is quite difficult, even if the attack knows the subgraph codec. This happens because the difficult (and classic) problem in Graph Theory, the isomorphism of subgraphs. 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 to 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 definition of the extractor algorithm, which we call *pre-extractor*. This algorithm, instead of returning exactly the fingerprint (previously embedded by the embedder algorithm), it returns a sequence of bits — possibly long — that contains the fingerprint as its substring. More precisely, the substring will be 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 exists the substring n of d, with N bits, such that n is equicomposite and one of the factors has k as prefix; NO, otherwise.

It is easy to see the problem SUBSTRING-PREFACTOR also can be reduced to SAT. In fact, we need to build a circuit PRE-MULT (similar to the circuit built in the PREFACTOR problem) to each substring of size N, and to apply a OR in 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 follows the traditional methods for raffling random numbers followed by primality test (for instance, Miller-Rabin) until you get a prime number. The generation of p follows a slightly modified approach: a random number is raffled and concatenated to the right of m to, in the following, to test its primality — and the process is repeated until to get a prime number.

3) Embedding the fingerprint: The embedding fingerprint process 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 — will depend of the digital artifact type. In case of a text file, for instance, the process will only depend on the codifying scheme used to embedded the information in the text. In Section III-A, we describe a scheme in which the sequence of "however" and "although" words will 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 compromises the following steps

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

The extraction of the sequence of bits which we refer in the initial step can be done by specific algorithms, which must be defined for each filed of application and their corresponding digital artifacts. The transformation to an instance of SAT is exactly the algorithm described in Section V-A, while the polynomial reduction from SAT to 3-COLORING is a classical result of the literature [24]. An interactive proof scheme of zero-knowledge for 3-COLORING is described on [25] and can be used in the last stage to exhibit the fingerprint without revealing n or any of its factors.

VI. Conclusions

In the present work, we describe a protocol aiming at increasing the use of fingerprinting tools in dispute scenarios related with intellectual property protection. The proposed protocol assumes watermarking schemes that meet the requirements of stealthy, resilience and verifiability, although it is questionable if existing schemes, in fact, meet such requirements. We also describe a secure way for verifying the fingerprint without exposing its contents/location through a partial transfer of knowledge scheme.

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] 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

[15] 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

[16] 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

[17] 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

[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] M. O. Rabin, "How to exchange secrets with oblivious transfer," Harvard Aiken Computation Laboratory, Tech. Rep. TR-81, 1981.

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

[23] 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

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

[25] 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