Machado *et al. EURASIP Journal on Information Security* (2016) 2016:8
DOI 10.1186/s13635-016-0032-5

EURASIP Journal on Information Security

RESEARCH Open Access

Software control and intellectual property protection in cyber-physical systems

Raphael C. S. Machado[1*], Davidson R. Boccardo[1], Vinícius G. Pereira de Sá[2] and Jayme L. Szwarcfiter[2]

Abstract

Software control is a critical issue in cyber-physical systems (CPS); if the expected behavior of the software embedded in a single device of a CPS cannot be enforced then the behavior of the whole CPS may be in jeopardy. Thus, CPS stakeholders like having some level of control over the embedded software. Third-party demands to control the software, however, conflict with the intellectual property protection demanded by software developers, since some level of detail about the software at hand would have to be disclosed. In the present paper, we discuss the issue of controlling the software embedded in CPS devices and address the problem of how to achieve an increased level of software control without compromising the protection of intellectual property. We propose a two-party fingerprinting scheme that allows for attribution of responsibility in the case of intellectual property leaks. Our fingerprinting scheme is such that neither party may obtain an advantage over the other by misbehaving, misrepresenting or by prematurely aborting the protocol, therefore providing a fair means to resolve disputes.

1 Introduction

A *cyber-physical system* (CPS) is a system composed of computational devices and physical environments, where the computational devices can interact via communication networks, can control physical environments via *actuators*, and can receive feedback from physical environments via *sensors*. In a sense, a CPS can be understood as a generalization of the classical control systems where sensors and actuators correspond to much more sophisticated computational devices that can make local autonomous decisions and global coordinated decisions based on sophisticated logic, which is afforded by the use of embedded-software smart devices and communication networks (see Fig. 1). While the precise definition of a CPS can vary from author to author, there are several common examples of CPS in the literature, such as smart grids, autonomous vehicles, medical monitoring, process control systems, distributed robotics, and automatic pilot avionics. Because they control physical systems, failures on a CPS may have catastrophic consequences such as energy blackouts or airplane crashes. Moreover, CPSs are frequently associated to public infrastructures, therefore having a large number of *stakeholders*, including government, regulators, infrastructure operators, civil rights organizations and citizen, which turn the governance of such systems into a really complex task.

An important characteristic for proper working of a CPS in the precise characterization of its composing devices, such as sensors, actuators, and other computational devices. The stakeholders of a CPS need to be aware of each device that influences the behavior of the CPS, including the software that is embedded on such devices.

Software control can occur both in the syntactic level—i.e., by guaranteeing that the software embedded in a device corresponds to a previously validated software version—and in the semantic level—i.e., by guaranteeing that the prescribed specification is met. In any case, the need for software control gives origin to a scenario where two potentially conflicting parts need to interact and cooperate. On one side, CPS stakeholders need to have more control over the software that is embedded in the CPS devices deployed in several application fields. On the other side, CPS device manufacturers consider their software as sensitive intellectual property and are not willing to reveal details of their software, let alone run the risk of propagating them among third parties. Thus, there is a tradeoff between third-party software control and protection of intellectual property.

*Correspondence: machado.work@gmail.com
[1] Instituto Nacional de Metrologia, Qualidade e Tecnologia, Rio de Janeiro, Brazil
Full list of author information is available at the end of the article

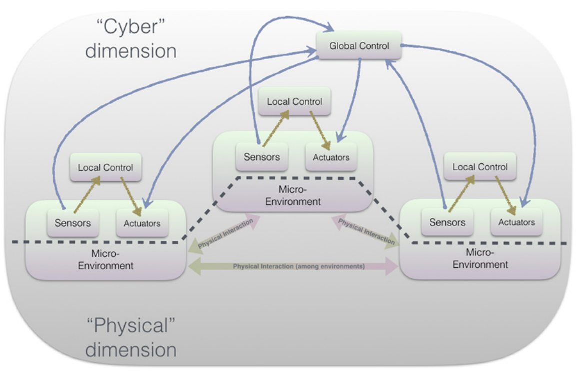

Fig. 1 Schematic diagram of a cyber-physical system

Watermarking techniques allow for the embedding of information in a digital artifact in such a way that the artifacts' functionalities are not impacted. Furthermore, the removal of a digital watermark must be a difficult task for a malicious user, 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 2) 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 responsibilities in cases of misuse of the artifact. As we will see, suspicion can usually befall upon both parties when a dispute takes place. 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*. Software fingerprints have been used as intellectual property protection tools: if a fingerprinted software is distributed and an illegal copy is found afterwards, it will be possible to retrieve its fingerprint and identify the responsible for the leakage. Software fingerprinting can therefore be used, in principle, as an ally to achieve software control with an adequate intellectual property protection level, since a third party that has access to a fingerprinted software will be discouraged to leak it. That does not eliminate, however, the risk that one of the parties that had access to it does eventually leak it—accidentally or intentionally.

The main problem with such fingerprinting methods, as we explain next, is that they are not *fair*, since they do not prevent, for one thing, non-repudiation attacks, which therefore compromises their applicability in dispute resolution scenarios. A protocol is fair if no party can gain an advantage over other parties by misbehaving, misrepresenting or by prematurely aborting the protocol. Fairness is an important requirement in exchange protocols. The fairness requirement arises when two parties are willing to exchange digital items, but do not trust each other. In order to avoid that some of the parties interrupt the protocol right after receiving the desired digital item, it is important that the exchange process is atomic, i.e., that all the items are exchanged at once. Some classical examples where fairness is a clear requirement are those of contract signing [1, 2], certified messages [3], and selling secrets [3].

It is usually difficult to achieve fairness in two-party protocols. The easiest way of assuring that an exchange protocol is fair is by recurring to a trusted third party (TTP); in the first stage, both parties deliver their items to the TTP who will exchange the items in a second stage. The scheme is secure, as long as the TTP is honest and unquestionable. However, such a protocol has the disadvantages of any protocol that requires the involvement of a third party in each and every exchange. Thus, a lot of effort has been put up lately towards the development of practical, fair exchange protocols.

The fingerprinting problem considered in the present work does not involve the exchange of items, since only one item—the watermarked software—is delivered. We can consider fairness as a highly important requirement all the same. Indeed, as discussed before, there are two potentially conflicting parties, i.e., the software *manufacturer*, who developed the software and wants to protect the intellectual property, and the CPS *stakeholder*, who wants to have some level of control over the software that is embedded in a CPS device[1]. It can be argued that, if one party is the responsible for fingerprinting the software, then both parties could have acted maliciously in the event of a leakage, since both parties had access to the fingerprinted software. In other words, how could one guarantee that it was not the manufacturer the ill-intendioned party, willing to impose guilt on a certain user/stakeholder?

To be more concrete, consider the following scenario. Alice wishes to sell a smart device—i.e., a device with embedded software—to Bob, but fears that Bob could distribute the software 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 learns that Eva is using an embedded software device for the precise same application as the device Alice delivered to Bob. The device's behavior looks suspiciously familiar to Alice, who finds out after due analysis that it happens to have a copy of the software that was embedded in the device she had sold to Bob. Alice goes to court. During the trial, Alice provides the judge with the selling contracts of her device (and software) and the version obtained 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. Moreover, he argues that Alice, being the author of the program, could herself 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.

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 [4] 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 2, we describe known approaches to support the protection of digital intellectual property. We introduce our fingerprinting protocol in Section 3. In Section 4, 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 5 contains our concluding remarks.

2 Related works

2.1 Software control in cyber-physical systems

CPSs are frequently related to critical infrastructures, therefore having a large number of stakeholders interested in their correct operation. In particular, such stakeholders now perceive that an appropriate control over a CPS depends on the control of the software that is embedded in the CPS devices. There are basically three key points the stakeholder must be confident in:

- that the embedded version of the software indeed functions as it is supposed to;
- that, under normal operation, the software can not be arbitrarily substituted by an operator;
- that, during a periodic control and particularly in case of suspicion of misuse, the CPS stakeholder will have tools to check *which* precise version of the software is embedded in a CPS device.

Formally, *software control* refers to the general problem of establishing an adequate level of control over the software that is used in a given application. Software control is not a requirement restricted to cyber-physical systems, and comprises a series of activities that are classical in Computer Science, namely:

Validation. The ability to verify that a software behaves as specified.
Authorization. The ability to prevent arbitrary software substitution.
Verification. The ability to identify the embedded software version.

The origins of software *validation* trace back to the very origins of computer programs and their attempted formal definitions. Indeed, classical theoretical computer science results refer to the general impossibility of determining/enforcing properties of programs, such as the well-known theorems by Turing [5] and Rice [6]. Classical programming analysis tools and their representations such as flowcharts, control flow graphs, and call graphs [7] help both the development and the characterization of computer programs, and some formal methods [8] aim at developing software whose behaviors (possible states and their transitions) can be completely characterized. A myriad of software engineering methods [9, 10] has been developed with the goal of providing some level of control over complex enterprise information systems. Despite the large number of methods and tools, the general goal is one and the same: to provide confidence that the software behaves as it should behave, performing the tasks it was programmed to perform—and nothing else. In the field of cyber-physical systems and critical systems in general, it has become increasingly common that government, regulators and users require more mature validation processes before a new version of software is deployed. Examples include regulations in Canada and Brazil [11, 12], which specify requirements to be adopted before the approval of metering devices. Both regulations are based on the International Organization of Legal Metrology's Guidance document D-31: General Requirements for Software Controlled Measuring Instruments [13]. In the case of Canada [11], the source code per se does not constitute an input for the validation, as the assessment process is based on functional tests and documentation review. The regulation in Brazil [12], on the other hand, relies on source code disclosure to perform deeper validation tests during the assessment process. It also includes functional tests and documentation review.

Authorization refers to the process whereby a stakeholder registers their accordance to a particular software version to be used in a given application. In the field of cyber-physical systems, it aims at guaranteeing that no software will be embedded in a device without proper permission. In practice, it means that the device will have the ability of verifying the credentials of users that want to update the software, and to check that the new version was authorized for that application, in the first place—for instance, by means of a digital signature protocol. Example usages of such a protocol include mobile computing applications, which assure users that the signed applications are from a known source and they have not been altered, at least from the last signature [14]. Apple manufacturer also uses code signing, not only on applications but also to perform operating system updates on personal computers and mobile devices. As an example of smart devices that use digital signature protocol, we can cite the payroll recorders adopted in some countries, in which the software update is only possible after the authority agency has digitally signed the software.

Verification refers to the ability of proving that the software embedded in a device corresponds to a version of software previously authorized for that application. In some countries, the verification of smart meters such as energy meters and payroll recorders is a requirement [12, 15]. The subject has been extensively studied in the last two decades, coinciding with the widespread dissemination of smart devices and embedded software in general. Some works propose the formalization of the basic steps of the verification process [16, 17], while other works are concerned in establishing the *root of trust* (RoT) [18]. Recently, some works [19, 20] propose the use of physically unclonable functions (PUFs) to integrate the verification process, making it more tamper-resistant. Kovah et al. [21] present a verification scheme based on time for personal and corporate computers, claiming that even with the network delay, it achieves good results. A similar approach was conducted by Preschern et al. [22] for safety-critical systems. The work by Francillon et al. [17] maps the minimal collection of hardware and software components that are needed for a secure verification, based on a precise definition of the desired verification service. Armknecht2013 et al. [16] present a security framework that formally captures security goals, attacker models and various system and design parameters aiming at increasing the confidence on the verification process.

2.2 Watermarking for intellectual property protection

In the literature, there are plenty of works tackling the embedding of digital watermarks in artifacts such as images [23], audio and video [23, 24], text documents [25, 26], and computer programs [27, 28]. 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 secrets [29]. From the standpoint of technological protection, several techniques have been developed to avoid reverse engineering, tampering, and illegal distribution of software.

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. 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.

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 [27, 28].

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 algorithm's 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 [30, 31]. 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.

3 The proposed fingerprinting protocol

As discussed along the paper, the main difficulty in conceiving a fair fingerprinting protocol without recurring to a TTP is 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 ilegal 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, where n is a sufficient number of functionally equivalent versions of the software, but each with a syntactically distinct watermark—even though they all identify the author. 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 3.2) until we get to a more robust version (Section 3.4). Note that although our motivating problem is the one of fingerprinting software, we describe the protocols generically for "digital artifacts". As long as a digital content—e.g., text files, images, music etc.—can be watermarked, they can be involved in such protocols.

Through this section, we assume the existence of *watermarking* tools and *diversity* tools, i.e.,

Watermarking assumption. There exist practical tools that receive as input a digital item together with an information to be embedded in that item and output an "equivalent" digital item that encodes that input information (i.e., the information appear as a watermark of the output item).
Diversity assumption. There exist practical tools that receive as input a digital item and output an "equivalent" digital item whose content is distinct form the input.

The theoretical existence of the above described tools is a current research topic, but in practice, there are several tools in for software watermarking and software diversity—a good reference for these tools is the book of Collberg.

3.1 Basics of oblivious transfer

In the present section, we describe the classical concept of oblivious transfer [4], which is the key tool to achieve fairness in our proposed fingerprinting protocol. Oblivious transfer allows the transference of an item among a set of items in such a way that the sender is not aware of which item was transferred. We illustrate with a concrete example.

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 a way that Alice will have no idea about which content has interested Bob.

More formally, assume Alice defines $m_1, \ldots, m_k$ messages and Bob wants access to the ith 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 ith encrypted message can be decrypted by Bob. Further details of this protocol are described in Section 3.

Several oblivious transfer models can be found in the literature, but they are in a sense equivalent [32–35]. The oblivious transfer concept plays an important role in building other more complex protocols [3].

3.2 Initial fingerprinting protocol based on oblivious transfer

We introduce a naive version of a fair fingerprinting protocol that 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.

Concretely, Alice, a software developer, generates a large number of equivalent versions of the same software—which can be achieved via the use of software diversity tools—and each version is distinctly watermaked by Alice—using an watermarking scheme at choice, such as the ones in [36] and references thereby.

Basic fingerprinting protocol

1. Alice creates α semantically equivalent variations $n_1, \ldots, n_\alpha$ of a digital artifact.
2. Alice watermarks each of the variations of the of a digital artifact.
3. Alice creates α pairs of public/private keys $Pr_1, Pu_1, \ldots, Pr_\alpha, Pu_\alpha$, and sends the public keys $Pu_1, \ldots, Pu_\alpha$ to Bob.
4. 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.
5. 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 $j = 1, \ldots, \alpha$.

6. 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 $E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j)$.
7. 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 Step 5, Alice will obtain α "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 the software, 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 $n_1, \ldots, n_\alpha$.

Observe that is still necessary to guarantee that Alice, in fact, generated α software versions with distinct fingerprints, and to have mechanisms to ensure that Bob, in fact, participated of the protocol execution.

3.3 Fingerprinting protocol with guarantee of distinct fingerprints

The verification protocol above allows a simple attack by Alice. Alice can generate α 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 α 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 [36].

Verification protocol with a naive test of distinct fingerprints

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

The modification above seems to be enough to guarantee that Alice cannot distribute one of the artifacts $n_1, \ldots, n_\alpha$ 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 α 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 α semantically equivalent variations $n_1, \ldots, n_\alpha$ of a digital artifact.
2. Alice generates cryptographic hashes $h(n_1), \ldots, h(n_\alpha)$, signs them and sends to Bob $(h(n_1), \ldots, h(n_\alpha), s_A(h(n_1)| \ldots |h(n_\alpha))$.
3. Bob verifies the signature of the hashes signed by Alice and returns the cryptographic hashes signed by him: $(h(n_1), \ldots, h(n_\alpha), s_B(h(n_1)| \ldots |h(n_\alpha))$.
4. Alice verifies the signature of the hashes sent by Bob, creates α pairs of public/private keys $Pr_1, Pu_1, \ldots, Pr_\alpha, Pu_\alpha$, and sends the public keys $Pu_1, \ldots, Pu_\alpha$ 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 $j = 1, \ldots, \alpha$.
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 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 $(h(n_1), \ldots, h(n_\alpha), s_A(h(n_1)| \ldots |h(n_\alpha))$ 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 $s_A(h(n_1)|\ldots|h(n_\alpha)$ is valid and if each artifact n_i has the correct cryptographic hash $h(n_i)$.
2. Verify if the signature $s_B(h(n_1)|\ldots|h(n_\alpha))$ is valid.
3. Verify if the fingerprints $f(n_1),\ldots,f(f_\alpha)$ are mutually distinct.
4. Verify if the fingerprint $f(\tilde{n})$ is the same as in any of the artifacts $n_1,\ldots,n_\alpha$.

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

3.4 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 α semantically equivalent variations $n_1,\ldots,n_\alpha$ of a digital artifact.
2. Alice generates cryptographic hashes $h(n_1),\ldots,h(n_\alpha)$, signs and sends them to Bob $(h(n_1),\ldots,h(n_\alpha),s_A(h(n_1)|\ldots|h(n_\alpha))$.
3. Bob verifies the signature of the hashes signed by Alice and returns the cryptographic hashes signed by him: $(h(n_1),\ldots,h(n_\alpha),s_B(h(n_1)|\ldots|h(n_\alpha))$.
4. Alice verifies the signature of the hashes signed by Bob, creates α pairs of public/private keys $Pr_1,Pu_1,\ldots,Pr_\alpha,Pu_\alpha$, and sends the public keys $Pu_1,\ldots,Pu_\alpha$ 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 $j=1,\ldots,\alpha$.
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 $s_B(E_{Pu_i}(k))$. Access to all α public keys and α private keys generated by Alice is also necessary.

Verification protocol with identification of Bob's key

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

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 2 wraps up the proposed fingerprinting protocol.

4 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

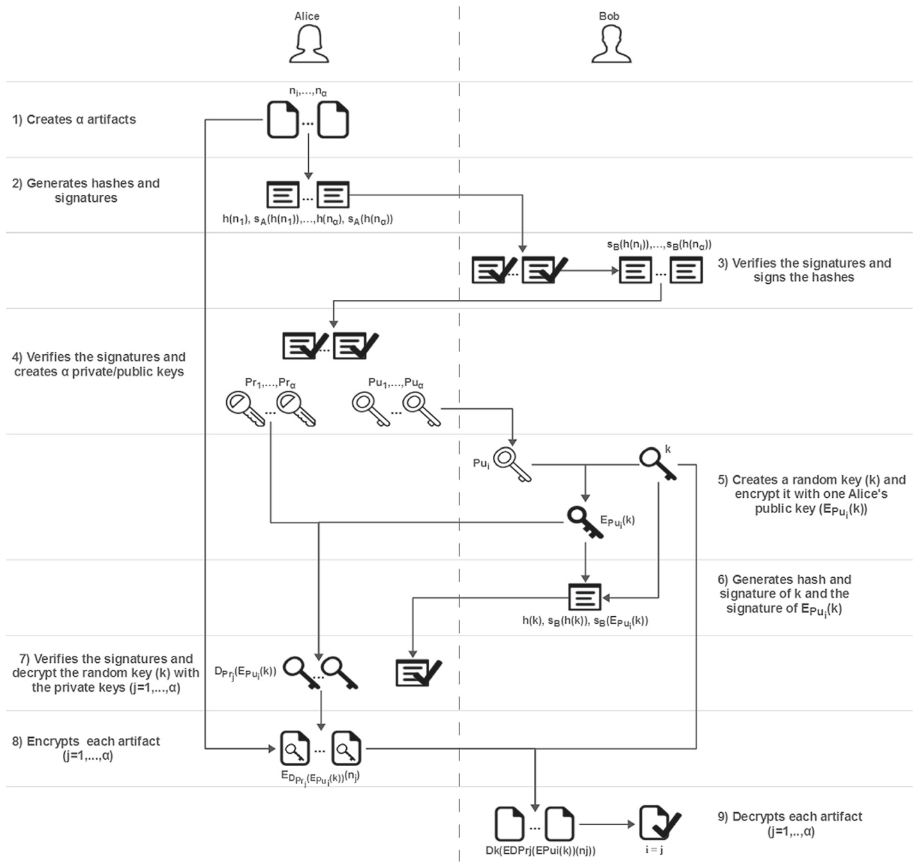
Alice
Bob
$n_i,\ldots,n_\alpha$
1) Creates α artifacts
2) Generates hashes and signatures
$h(n_1), s_A(h(n_1)),\ldots,h(n_\alpha), s_A(h(n_\alpha))$
$s_B(h(n_i)),\ldots,s_B(h(n_\alpha))$
3) Verifies the signatures and signs the hashes
4) Verifies the signatures and creates α private/public keys
$Pr_1,\ldots,Pr_\alpha$
$Pu_1,\ldots,Pu_\alpha$
Pu_i
k
5) Creates a random key (k) and encrypt it with one Alice's public key ($E_{Pu_i}(k)$)
$E_{Pu_i}(k)$
6) Generates hash and signature of k and the signature of $E_{Pu_i}(k)$
$h(k), s_B(h(k)), s_B(E_{Pu_i}(k))$
7) Verifies the signatures and decrypt the random key (k) with the private keys (j=1,...,α)
$D_{Pr_j}(E_{Pu_i}(k))$
8) Encrypts each artifact (j=1,...,α)
$E_{D_{Pr_j}(E_{Pu_i}(k))}(n_j)$
9) Decrypts each artifact (j=1,..,α)
Dk(EDPrj(EPui(k))(nj))
i = j

Fig. 2 Proposed fingerprinting protocol

will remain as hard after the verification as it used to be before it.

4.1 Partial transfer of knowledge scheme

In Kilian's doctoral thesis [37], 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.

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.

4.1.1 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 [38]. 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.

4.1.2 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 (1) the bit S resulting from the sum of the three input bits and (2) a new "carry" bit C_o (Fig. 3).

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

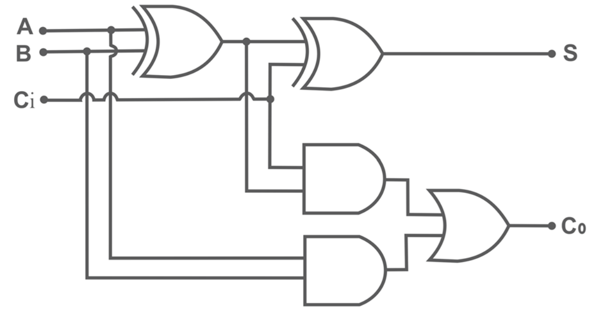

Fig. 3 Representation of a 1-bit full adder

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

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 (Fig. 4).

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 shift, 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 $B << i$.

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 $C = B \wedge A_0 + (B << 1) \wedge A_1 + (B << 2) \wedge A_2 + (B << 3) \wedge A_3$ (Fig. 5).

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 Fig. 6, 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 if and only if the binary number n may be written as a product of two binary numbers of up to d bits.*

4.1.3 The PREFACTOR problem

Now, we consider the problem of determining if a number may be written as a product of two other

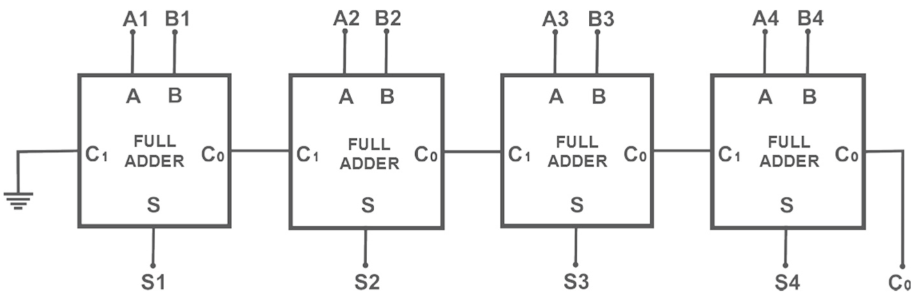

Fig. 4 Representation of a 4-bit adder

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.
Sa?da: 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 Fig. 6, but with some of the bits shortcircuited directly into the last stage of the circuit, which receives an additional AND port as illustrated in Fig. 7.

Formally, a circuit PRE-MULT(d, n, k), where d is integer and n and k are binary numbers, is built as shown in

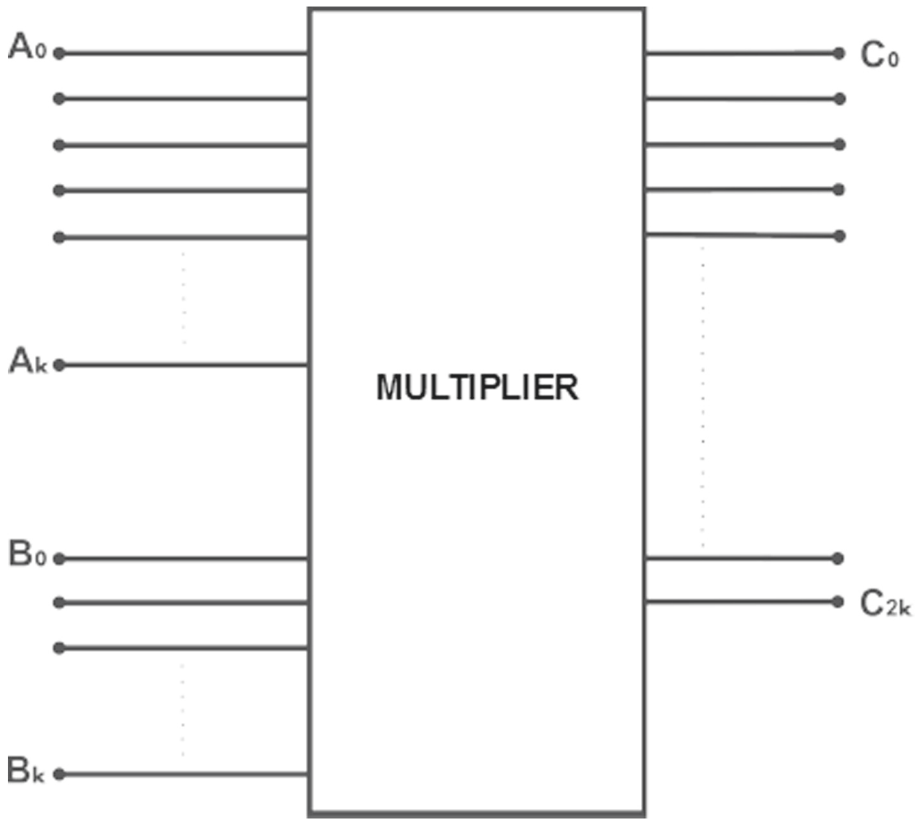

Fig. 5 Representation of a k-bit multiplier

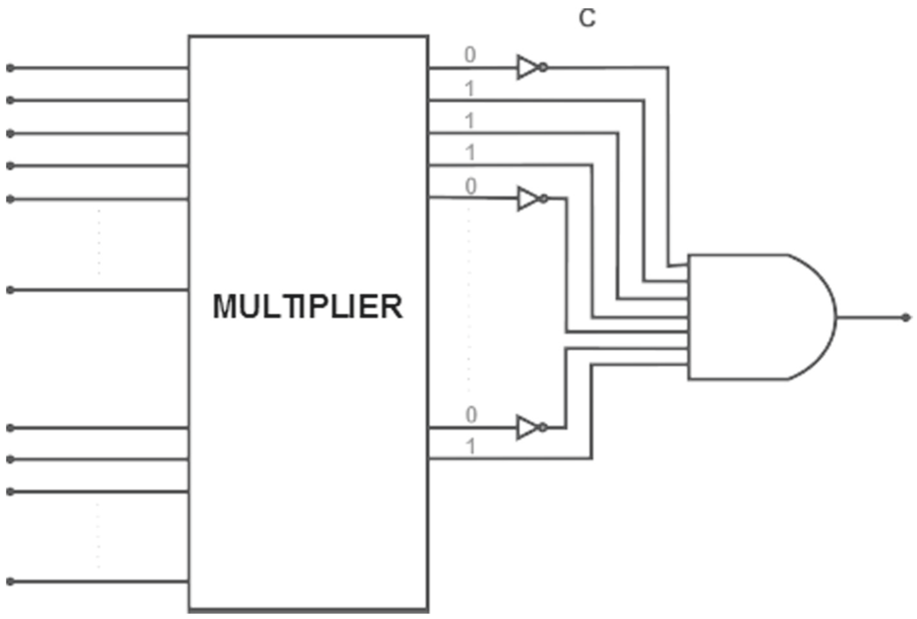

Fig. 6 UNI-MULT: fixing the output bits with a final AND port

Fig. 7. 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.*

4.1.4 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 [39] allows us

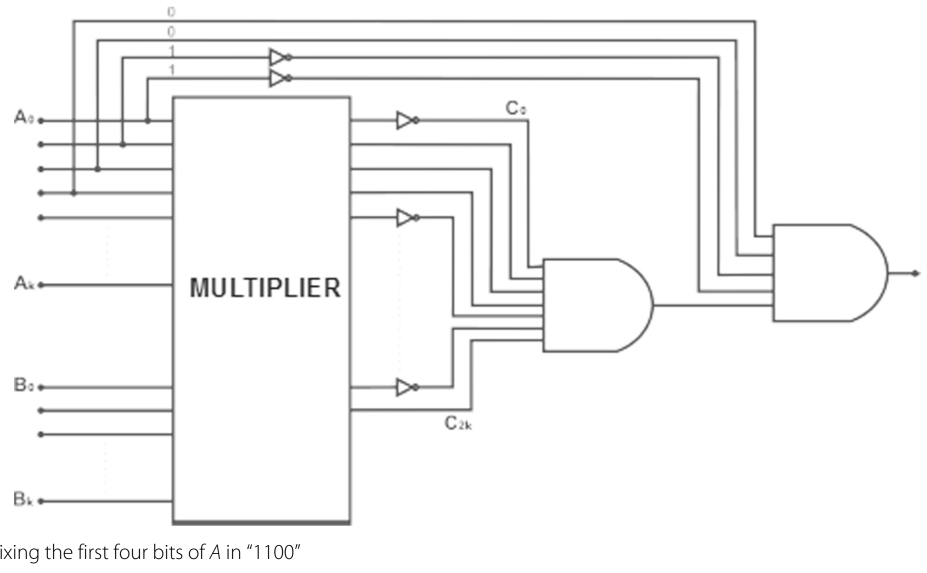

Fig. 7 PRE-MULT: fixing the first four bits of *A* in "1100"

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.

4.1.5 *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 [40], and then use a classical scheme of zero-knowledge proof for this last problem [41].

4.2 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 4.1, 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.

4.2.1 *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.

4.2.2 *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.

4.2.3 *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—depends on the digital artifact type. In the case of a software, one could use a watermarking scheme based on the modification of the control flow graph [36].

4.2.4 *Verifying the fingerprint*

The fingerprint verification comprises the following steps:

1. Extraction of the sequence of bits embedded in the digital artifact—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 4.1, while the polynomial reduction from SAT to 3-COLORING is well known [40]. An interactive proof scheme of zero-knowledge for 3-COLORING is described on [41] and can be used in the last stage to exhibit the fingerprint without revealing n or any of its factors.

5 Conclusions

In the present work, we describe a protocol which increases the use of fingerprinting tools in dispute scenarios related to intellectual property. The proposed protocol assumes the existence of watermarking schemes that meet the 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, which is an application of Zero-Knowledge Proofs.

Endnote

[1]Note that there can be several stakeholders and interested parties, such as users, regulators etc. Their relations, though, will be modelled as two-party relations in this paper.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
A preliminary version of this paper has been published in the Proceedings of the 10th International Conference on Availability, Reliability and Security (ARES'2015), pp. 110–119. Research partially supported by FAPERJ, Capes and CNPq.

Author details
[1]Instituto Nacional de Metrologia, Qualidade e Tecnologia, Rio de Janeiro, Brazil. [2]Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil.

Received: 12 February 2016 Accepted: 29 March 2016
Published online: 12 April 2016

References

1. S Even, O Goldreich, A Lempel, A randomized protocol for signing contracts. Communications of the ACM. **28**(6), 637–647 (1985)
2. M Ben-Or, O Goldreich, S Micali, RL Rivest, A fair protocol for signing contracts. IEEE Trans. Inf. Theory. **36**(1), 40–46 (1990)
3. B Schneier, *Applied Cryptography*. (Wiley, New York, 1996)
4. MO Rabin, How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University (1981)
5. AM Turing, On computable numbers with an application to the Entscheidungsproblem. Proc. London Math. Soc. **s2-42**(1), 230–265 (1937)
6. HG Rice, Classes of recursively enumerable sets and their decision problems. Trans. Am. Math. Soc. **74**(2), 358–366 (1953)
7. CC Collberg, J Nagra, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. (Addison Wesley, 2010)
8. J Woodcock, PG Larsen, J Bicarregui, J Fitzgerald, Formal methods: practice and experience. ACM Comput. Surv. **41**(4), 19–11936 (2009). doi:10.1145/1592434.1592436
9. LL Beck, TE Perkins, A survey of software engineering practice: tools, methods, and results. Softw. Eng. IEEE Trans. **SE-9**(5), 541–561 (1983). doi:10.1109/TSE.1983.235114
10. A Meneely, B Smith, L Williams, Validating software metrics: a spectrum of philosophies. ACM Trans. Softw. Eng. Methodol. **21**(4), 24–12428 (2013). doi:10.1145/2377656.2377661
11. M Canada, Specifications for the approval of software controlled electricity and gas metering devices, specifications relating to event loggers for electricity and gas metering devices Technical report, An Agency of Industry, Canada (2011)
12. Inmetro, Software requirements of smart energy meters. Technical report, 999, National Institute of Metrology, Quality and Technology (2012)
13. OIML, D31 general requirements of software controlled measuring 1001 instruments, international organization of legal metrology. Technical 1002 report, Organisation Internationale de Métrologie Légale (2009)
14. Apple, Code Signing
15. Inmetro, Software requirements of payroll recorders. Technical report, 1007, National Institute of Metrology, Quality and Technology (2013)
16. F Armknecht, A-R Sadeghi, S Schulz, C Wachsmann, in *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*. CCS '13. A security framework for the analysis and design of software attestation (ACM, New York, NY, USA, 2013), pp. 1–12. doi:10.1145/2508859.2516650
17. A Francillon, Q Nguyen, KB Rasmussen, G Tsudik, in *Proceedings of the Conference on Design, Automation & Test in Europe*. DATE '14. A minimalist approach to remote attestation (European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 2014), pp. 244–12446
18. J Horsch, S Wessel, F Stumpf, C Eckert, in *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*. CODASPY '14. Sobtra: a software-based trust anchor for arm cortex application processors (ACM, New York, NY, USA, 2014), pp. 273–280. doi:10.1145/2557547.2557569
19. U Kocabas, AR Sadeghi, C Wachsmann, S Schulz, in *Proceedings of the 18th ACM Conference on Computer and Communications Security*. CCS '11. Poster: Practical embedded remote attestation using physically unclonable functions (ACM, New York, NY, USA, 2011), pp. 797–800. doi:10.1145/2046707.2093496
20. J Kong, F Koushanfar, PK Pendyala, A-R Sadeghi, C Wachsmann, in *Design Automation Conference (DAC) 2014, Best Paper Candidate*. Pufatt: Embedded platform attestation based on novel processor-based pufs (ACM, San Francisco, 2014)
21. X Kovah, C Kallenberg, C Weathers, A Herzog, M Albin, J Butterworth, in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. SP '12. New results for timing-based attestation (IEEE Computer Society, Washington, DC, USA, 2012), pp. 239–253. doi:10.1109/SP.2012.45
22. C Preschern, AJ Hormer, N Kajtazovic, C Kreiner, in *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference On*. Software-based remote attestation for safety-critical systems, (2013), pp. 8–12. doi:10.1109/ICSTW.2013.7
23. W Bender, D Gruhl, N Morimoto, in *Storage and Retrieval for Image and Video Databases (SPIE)*. Techniques for data hiding, (1995), pp. 164–173
24. I Cox, ML Miller, JA Bloom, *Digital Watermarking*. (Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 2002)
25. MJ Atallah, V Raskin, M Crogan, C Hempelmann, F Kerschbaum, D Mohamed, S Naik, in *Information Hiding*. Lecture Notes in Computer Science, ed. by IS Moskowitz. Natural language watermarking: Design, analysis, and a proof-of-concept implementation, vol. 2137 (Springer, 2001), pp. 185–199
26. MJ Atallah, V Raskin, C Hempelmann, M Karahan, R Sion, U Topkara, KE Triezenberg, in *Information Hiding*. Lecture Notes in Computer Science, ed. by FAP Petitcolas. Natural language watermarking and tamperproofing, vol. 2578 (Springer, 2002), pp. 196–212
27. W Zhu, CD Thomborson, F-Y Wang, ed. by PB Kantor, G Muresan, FS Roberts, DD Zeng, F-Y Wang, H Chen, and RC Merkle. Proc. IEEE Int'l Conference on Intelligence and Security Informatics. *ISI'05*, vol. 3495 (Springer, 2005), pp. 454–458
28. J Hamilton, S Danicic, "A survey of static software watermarking", Internet Security (WorldCIS), 2011 World Congress on, London, 100–107 (2011)
29. RL Ostergard, The measurement of intellectual property rights protection. J. Int. Business Stud. **31**(2), 349–360 (2000)
30. S Craver, ND Memon, B-L Yeo, MM Yeung, Resolving rightful ownerships with invisible watermarking techniques: limitations, attacks, and implications. IEEE J. Selected Areas Commun. **16**(4), 573–586 (1998)
31. A Sadeghi, A Adelsbach, Advanced techniques for dispute resolving and authorship proofs on digital works. Proc. SPIE 5020, Security and Watermarking of Multimedia Contents V (2003)
32. G Brassard, C Crépeau, J-M Robert, in *FOCS*. Information theoretic reductions among disclosure problems (IEEE Computer Society, 1986), pp. 168–173

33. C Crépeau, in *CRYPTO*. Lecture Notes in Computer Science, ed. by C Pomerance. Equivalence between two flavours of oblivious transfers, vol. 293 (Springer, 1987), pp. 350–354
34. C Crépeau, J Kilian, in *CRYPTO*. Lecture Notes in Computer Science, ed. by S Goldwasser. Weakening security assumptions and oblivious transfer (abstract), vol. 403 (Springer, 1988), pp. 2–7
35. M Bellare, S Micali, in *Advances in Cryptology–CRYPTO' 89 Proceedings*. Lecture Notes in Computer Science, ed. by G Brassard. Non-interactive oblivious transfer and applications, vol. 435 (Springer, 1990), pp. 547–557. doi:10.1007/0-387-34805-0_48
36. L Bento, D Boccardo, R Machado, V de Sů, J Szwarcfiter, in *Proceedings of Brazilian Symposium in Information and Computater Systems Security*. SBSEG. A randomized graph-based scheme for software watermarking (SBC, 2014), pp. 30–41
37. J Kilian, Uses of Randomness in Algorithms and Protocols. Ph. D. Thesis. Massachusetts Institute of Technology, Dept. of Mathematics (1989)
38. TJ Schaefer. The complexity of satisfiability problems. Proc. of the 10th Annual ACM Symposium on Theory of Computing, (1978), pp. 216–226
39. GS Tseitin, in *Automation of Reasoning*. Symbolic Computation, ed. by J Siekmann, G Wrightson. On the complexity of derivation in propositional calculus (Springer, 1983), pp. 466–483. doi:10.1007/978-3-642-81955-1_28
40. RM Karp, in *Complexity of Computer Computations*, ed. by RE Miller, JW Thatcher. Reducibility among combinatorial problems (Plenum Press, New York, 1972)
41. S Goldwasser, S Micali, C Rackoff, in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC '85. The knowledge complexity of interactive proof-systems (ACM, New York, NY, USA, 1985), pp. 291–304. doi:10.1145/22145.22178