Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 331 (2017) 71–85

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Software Watermarking: A Semantics-based Approach

Mila Dalla Preda[1,2,4] Michele Pasqua[1,3]

Department of Computer Science
University of Verona
Strada le Grazie 15, 37134, Verona, ITALY

Abstract

Software watermarking is a defence technique used to prevent software piracy by embedding a signature, i.e., an identifier reliably representing the owner, in the code. When an illegal copy is made, the ownership can be claimed by extracting this identifier. The signature has to be hidden inside the program and it has to be difficult for an attacker to detect, tamper or remove it. In this paper we show how the ability of the attacker to identify the signature can be modelled in the framework of abstract interpretation as a completeness property. We view attackers as abstract interpreters that can precisely observe only the properties for which they are complete. In this setting, hiding a signature in the code corresponds to inserting it in terms of a semantic property that can be retrieved only by attackers that are complete for it. Indeed, any abstract interpreter that is not complete for the property specifying the signature cannot detect, tamper or remove it. The goal of this work is to introduce a formal framework for the modelling, at a semantic level, of software watermarking techniques and their quality features.

Keywords: Software Watermarking, Abstract Interpretation, Program semantics, Program transformation

1 Introduction

Software developers are interested in protecting the intellectual property of their products against software piracy, namely to prevent the illegal reuse of their code. Software watermarking is a technique for embedding a signature, i.e., an identifier reliably representing the owner, in a cover program. This allows software developers to prove their ownership by extracting their signature from the pirated copies. In the last two decades researchers have developed a variety of software watermarking techniques (e.g., [3,4]) that can be classified in three main categories according to their extraction process: static, dynamic and abstract watermarking. *Static*

[1] We would like to thank Roberto Giacobazzi for the initial discussions on this work and Isabella Mastroeni for the discussions on high-order abstract non-interference.

[2] Email: mila.dallapreda@univr.it

[3] Email: michele.pasqua@univr.it

[4] This work has been supported by the MIUR FIRB 2013 project FACE RBFR13AJFT.

http://dx.doi.org/10.1016/j.entcs.2017.02.005
1571-0661/© 2017 The Author(s). Published by Elsevier B.V.

watermarking inserts signatures in the cover program either as data or code and then extracts them statically, namely without executing the code [4]. Conversely, *dynamic watermarking* inserts signatures in the program execution state (i.e., in its semantics) and the extraction process requires the execution of the program, often on a special enabling input [4]. *Abstract watermarking*, introduced in [10], encodes the signature in such a way that it could be extracted only by a suitable abstract execution of the program. A watermarking scheme is typically evaluated w.r.t. the following features: credibility (how strongly it proves authorship), secrecy (how difficult it is to extract the mark), transparence (how difficult it is to realize that a program is marked), accuracy (observational equivalence of the marked and original program), resilience to attacks (how difficult it is to compromise the correct extraction of the signature) and data-rate (amount of information that can be encoded). The quality of each existing watermarking technique is specified in terms of their features that are typically claimed to hold w.r.t. the peculiar embedding and extraction methods. There exists a variety of embedding and extraction algorithms that often work on different objects (control flow graph, variables, registers, etc.) and this makes it difficult to compare the efficiency of different watermarking systems. It is therefore difficult to formally prove and compare limits and potentialities of the different watermarking systems and to decide which one is better to use in a given scenario.

These problems derive also by the fact that, at the state of the art, there is a poor theoretical investigation about software watermarking. The concept was formally defined in [1] and, in the same work, the authors showed that the existence of indistinguishability obfuscators implies that software watermarking cannot exist. Furthermore the recent candidate construction of an indistinguishability obfuscator [12] lowers the hope of building meaningful watermarking scheme. Fortunately this impossibility result relies on the fact that the signed program computes the same function as the original program. Indeed, in [1] the authors suggested that if we relax this last constraint, i.e., we require that the watermarking process has only to preserve an “approximation” of original program’s functionality, then positive results may be possible. This naturally leads to reason about software watermarking at semantic level, as we do in the present work.

A first attempt to provide a formal definition, in the semantics setting, of a watermarking system has been proposed in [13]. Here the author introduced the idea that static and dynamic watermarking are instances of abstract watermarking. Intuitively, the latter can be seen like static watermarking because the extraction of the signature requires no execution. But, at the same time, it can be seen like dynamic watermarking because the signature is hidden in the semantics. So all these three types of techniques could be seen as particular instances of a common watermarking scheme based on program semantics and abstract interpretation.

In this work, we start from that intuition and we transform the scheme proposed in [13] in a formal and consistent definition of what a software watermarking system is. The idea is to model the embedding of the secret signature s as the encoding of s in a semantic property $\overline{\mathfrak{M}}(s)$ that is then inserted in the semantics of the cover program. In this setting, the extraction process requires an analysis of the marked code that has to be at least as precise as $\overline{\mathfrak{M}}(s)$. This notion of precision of the extraction corresponds to the notion of completeness of the analysis in abstract interpretation. This means that in order to extract the signature it is necessary to know how it is encoded. In this view the semantic property for which the analysis has to be complete in order to extract the signature plays the role of an extraction key. Indeed, the signature is hidden to any observer of the program's semantics which is not complete for $\overline{\mathfrak{M}}(s)$, namely which does not know the "secret key". Based on these ideas we provide a formal semantics-based definition of a watermarking system. Moreover, we provide a specification of the features of a watermarking system in the semantic framework in terms of semantic program properties (this problem was not addressed at all in [13]). For example, it turns out that a watermarking scheme is transparent w.r.t. an observer when the embedding process preserves the program properties in which the observer is interested. Moreover, the resilience of a watermarking scheme to collusive attacks, that attempt to remove the signature by comparing different marked programs, can be modelled as a property of abstract non-interference among programs.

Finally we do a more precise validation than the one done in [13] (which is just sketched). We take into account two known watermarking techniques and we define them in our framework. Our investigation and study in this direction has led to the following contributions.

- Specification of a formal framework based on program semantics and abstract interpretation for the modelling of software watermarking. The framework refines and extends the one proposed in [13].
- Formalization of the features (resilience, secrecy, transparence, accuracy) used to measure the quality of a watermarking system in the framework.
- Validation of the framework on two watermarking techniques.

2 Preliminaries

Mathematical notation

Given two sets S and T, we denote with $\wp(S)$ the powerset of S, with $S \setminus T$ the set-difference between S and T, with $S \subset T$ strict inclusion and with $S \subseteq T$ inclusion. Let $S_\bot$ be set S augmented with the undefined value $\bot$, i.e., $S_\bot = S \cup \{\bot\}$. $\langle P, \leq \rangle$ denotes a poset P with ordering relation $\leq$, while a complete lattice P, with ordering $\leq$, least upper bound (lub) $\vee$, greatest lower bound (glb) $\wedge$, greatest element (top) $\top$, and least element (bottom) $\bot$ is denoted by $\langle P, \leq, \vee, \wedge, \top, \bot \rangle$. $\sqsubseteq$ denotes the pointwise ordering between functions. If $f : S \longrightarrow T$ and $g : T \longrightarrow Q$ then $g \circ f : S \longrightarrow Q$ denotes the composition of f and g, i.e., $g \circ f = \lambda x.g(f(x))$. $f : P \xrightarrow{c} Q$ on posets is (Scott)-continuous when f preserves lub of countable chains in P. $f : C \longrightarrow D$ on complete lattices is additive [co-additive] when, for any $Y \subseteq C$, $f(\vee_C Y) = \vee_D f(Y)$ [$f(\wedge_C Y) = \wedge_D f(Y)$]. The right [left] adjoint of a function f is $f^+ \overset{\text{def}}{=} \lambda x. \bigvee\{y \mid f(y) \leq x\}$ [$f^- \overset{\text{def}}{=} \lambda x. \bigwedge\{y \mid x \leq f(y)\}$].

Abstract Interpretation

Abstract interpretation is based on the idea that the behaviour of a program at different levels of abstraction is an approximation of its (concrete) semantics [7,8]. The concrete program semantics is computed on the concrete domain $\langle C, \leq_C \rangle$, while approximation is encoded by an abstract domain $\langle A, \leq_A \rangle$. In abstract interpretation abstraction is specified as a Galois connection (GC) (C, α, γ, A), namely as an abstraction map $\alpha : C \longrightarrow A$ and a concretization map $\gamma : A \longrightarrow C$ that are monotone and that form an adjunction: $\forall y \in A, x \in C : \alpha(x) \leq_A y \Leftrightarrow x \leq_C \gamma(y)$ [7,8]. α [resp. γ] is the left[right]-adjoint of γ [α] and it is additive [co-additive], i.e. it preserves the lub [glb] of all the subsets of the domain (empty set included). Abstract domains can be equivalently formalized as upper closure operators on the concrete domain [8]. An upper closure operator, or closure, on a poset $\langle C, \leq \rangle$ is an operator $\varphi : C \longrightarrow C$ that is monotone, idempotent and extensive (i.e. $\forall c \in C : c \leq \varphi(c)$). Closures are uniquely determined by the set of their fixpoints $\varphi(C)$. The set of all closures on C is denoted by $uco(C)$. The lattice of abstract domains of C is therefore isomorphic to $uco(C)$ [7,8]. If C is a complete lattice, then $\langle uco(C), \sqsubseteq, \sqcup, \sqcap, \lambda x.\top, id \rangle$ is a complete lattice, where $id \overset{\text{def}}{=} \lambda x.x$ and for every $\rho, \eta \in uco(C)$, $\rho \sqsubseteq \eta$ iff $\forall y \in C : \rho(y) \leq \eta(y)$ iff $\eta(C) \subseteq \rho(C)$.

Precision of an abstract interpretation is typically defined in terms of completeness. Depending on where we compare the concrete and the abstract computations we obtain two different notions of completeness [15]. If we compare the results in the abstract domain, we obtain what is called backward completeness ($\mathcal{B}$-completeness) while, if we compare the results in the concrete domain, we obtain the so called forward completeness ($\mathcal{F}$-completeness). Formally, if $f : C \longrightarrow C$ and $\rho, \eta \in uco(C)$, then $\langle \rho, \eta \rangle$ is a pair of $\mathcal{B}[\mathcal{F}]$-complete abstractions for f if $\rho \circ f = \rho \circ f \circ \eta$ [$f \circ \eta = \rho \circ f \circ \eta$] (equivalently, we say that f is $\mathcal{B}[\mathcal{F}]$-complete for $\langle \rho, \eta \rangle$). The least fixpoint (lfp) of an operator F on a poset $\langle P, \leq \rangle$, when it exists, is denoted by $lfp^{\leq} F$, or by $lfp F$ when $\leq$ is clear. Any continuous operator $F : C \longrightarrow C$ on a complete lattice $\langle C, \leq, \vee, \wedge, \top, \bot \rangle$ admits a least fixpoint: $lfp^{\leq} F = \bigvee_{n \in \mathbb{N}} F^i(\bot)$, where for any $i \in \mathbb{N}$ and $x \in C$: $F^0(x) = x$; $F^{i+1}(x) = F(F^i(x))$.

Semantics of programs

We consider an imperative programming language Imp, similar to the one described in [9], equipped with a command `input` X that receives an input value from the user. The input stream given to the program is modelled as a sequence of values. At the beginning this sequence contains all the input values given, in order, to the program from its first element to its last. Each statement `input` "consumes" the first element of the sequence and so when the sequence is empty there are no more values that can be passed to the program. The small-step operational semantics of Imp induces a transition system $\langle \Sigma, \mathbf{S} \rangle$, where Σ is the set of possible program states. A program state is a pair $\langle C, \zeta \rangle$ where C is the command that has to be executed in the context $\zeta = \langle \rho, \iota \rangle$ that specifies both the assignment of values to variables ρ and the input stream ι that still needs to be consumed. As usual, the *transition relation*

$\mathbf{S} \in \Sigma \longrightarrow \wp(\Sigma)$ over program states specifies the set of states that are reachable from a given state. Let us denote with Σ_P the set of states of a program P, and with $\mathbf{S}_P : \Sigma_P \longrightarrow \wp(\Sigma_P)$ the transition relation over states of P. As usual Σ^+ denotes the set of all possible finite non-empty sequences of states and ϵ the empty sequence. Given a sequence of states $\sigma = \varsigma_0 \dots \varsigma_{n-1} \in \Sigma^+$, let $|\sigma| = n \in \mathbb{N}$ denote its length and σ_i its i-th element. A *trace* $\sigma \in \Sigma^+$ is a sequence of states $\varsigma_0 \dots \varsigma_{n-1}$ such that: $\forall i \in [1, n) \,.\, \varsigma_i \in \mathbf{S}(\varsigma_{i-1})$.

A state ς is a blocking state, for the program P, when $\mathbf{S}_P(\varsigma) = \varnothing$. Let Σ_P^T be the set of blocking states of P. A *maximal finite trace* of P, is a finite trace of P where the last state is blocking. The *maximal finite traces semantics* $(\!|P|\!)_+$ of the program P is given by the union of all maximal finite traces of length $n > 0$ and can be expressed as the least fixpoint of the transfer function $F_+ \in \wp(\Sigma^+) \xrightarrow{m} \wp(\Sigma^+)$ defined as: $F_+ \overset{\text{def}}{=} \lambda S \,.\, \Sigma_P^T \cup \{\varsigma\varsigma'\sigma \mid \varsigma' \in \mathbf{S}_P(\varsigma) \wedge \varsigma'\sigma \in S\}$. We can define the *maximal input semantics* function $(\!\{P\}\!)_+ \overset{\text{def}}{=} \lambda X.\{\sigma \in (\!|P|\!)_+ \mid \sigma_0 \in X\}$ that returns the set of maximal traces with initial state in X. It is possible to compute $(\!\{P\}\!)_+(X)$ as $\lambda X.\mathit{lfp}_{\varnothing}^{\leqslant} F_+^X$, where the fixpoint of function $F_+^X : \wp(\Sigma^+) \xrightarrow{c} \wp(\Sigma^+)$, defined on the DCPO $\langle \wp(\Sigma^+), \leqslant, \uplus, \varnothing \rangle$, is the maximal finite traces semantics starting from X. The partial order $\leqslant \subseteq \wp(\Sigma^+) \times \wp(\Sigma^+)$ is defined as: $X \leqslant Y \Leftrightarrow (\forall \sigma \in X \,\exists \sigma' \in Y \,.\, \sigma \in \mathtt{pref}(\sigma')) \wedge ((\forall \sigma' \in Y \,\exists \sigma \in X \,.\, \sigma \in \mathtt{pref}(\sigma')) \Rightarrow Y \subseteq X)$. Here $\mathtt{pref} : \Sigma^+ \longrightarrow \wp(\Sigma^+)$ is a function that returns the set of prefix of a given trace, so $\mathtt{pref}(\sigma) \overset{\text{def}}{=} \{\sigma' \in \Sigma^+ \cup \{\epsilon\} \mid \exists \sigma'' \in \Sigma^+ \,.\, \sigma = \sigma'\sigma''\}$. The least upper bound $\biguplus$ is defined as: $\biguplus \mathcal{X} \overset{\text{def}}{=} \{\sigma \in \bigcup_{X \in \mathcal{X}} X \mid \forall \sigma' \in \bigcup_{X \in \mathcal{X}} X \,.\, \sigma \in \mathtt{pref}(\sigma') \Rightarrow \sigma = \sigma'\}$. The bottom element is $\varnothing \in \wp(\Sigma^+)$. Finally $F_+^X \overset{\text{def}}{=} \lambda S \,.\, \{\varsigma \in \Sigma_P \mid \varsigma \in X\} \uplus \{\sigma\varsigma'\varsigma \mid \varsigma \in \mathbf{S}_P(\varsigma') \wedge \sigma\varsigma' \in S\}$. We have: $\mathit{lfp}_{\varnothing}^{\leqslant} F_+^X = \biguplus_{n \in \mathbb{N}} {F_+^X}^n(\varnothing) = \{\sigma \in (\!|P|\!)_+ \mid \sigma_0 \in X\}$.

Semantics can be abstracted by computing the least fixpoint of the best correct approximation (bca) of the corresponding transfer function on the desired abstract domain. Given the concrete domain $\langle \wp(\Sigma^+), \subseteq, \cup, \cap, \Sigma^+, \varnothing \rangle$, the bca of $(\!|P|\!)_+$ in $\rho \in \mathit{uco}(\wp(\Sigma^+))$ is $(\!|P|\!)_+^\rho \overset{\text{def}}{=} \mathit{lfp}_{\varnothing}^{\subseteq} \rho \circ F_+ \circ \rho$. Let $\langle \wp(\Sigma^+), \leqslant, \uplus, \varnothing \rangle$ be the concrete domain, the bca of $(\!\{P\}\!)_+$ in ρ is: $(\!\{P\}\!)_+^\rho \overset{\text{def}}{=} \lambda S \,.\, \mathit{lfp}_{\varnothing}^{\leqslant} \rho \circ F_+^S \circ \rho$.

Abstract non-interference

Abstract non-interference (ANI) [14] is a natural weakening of non-interference by abstract interpretation. In order to model non-interference in code transformations, such as software watermarking, we consider an higher-order version of ANI (HOANI), where the objects of observations are programs instead of values. Hence, we have a part of a program (semantics) that can change and that is secret, and the environment which remains the same up to an observable property. Let P be the set of cover programs, Q the set of secret programs and $\mathfrak{I} : \textsc{Imp} \times \textsc{Imp} \longrightarrow \textsc{Imp}$ an integration function. As usual, the attacker is modelled as a couple $\langle \eta, \rho \rangle$, with $\eta, \rho \in \mathit{uco}(\wp(\Sigma^+))$, that represents the input and output public observation power. In contrast, $\phi \in \mathit{uco}(\wp(\Sigma^+))$ is the property of the secret input. We say that the integration $\mathfrak{I}$, given $\eta, \phi, \rho \in \mathit{uco}(\wp(\Sigma^+))$, satisfies HOANI w.r.t. $\langle \eta, \phi, \rho \rangle$ and $\langle \mathsf{P}, \mathsf{Q} \rangle$, denoted as ${}^{\mathrm{H}}_{+}[\eta]\mathfrak{I}(\phi \Rightarrow \rho)_{\mathtt{bca}}$, if $\forall P_1, P_2 \in \mathsf{P} \,\forall Q_1, Q_2 \in \mathsf{Q}$: $(\!|P_1|\!)_+^\eta = (\!|P_2|\!)_+^\eta \wedge (\!|Q_1|\!)_+^\phi =$

$(\!|Q_2|\!)^{\phi}_{+} \Rightarrow (\!|\mathfrak{I}(P_1, Q_1)|\!)^{\rho}_{+} = (\!|\mathfrak{I}(P_2, Q_2)|\!)^{\rho}_{+}$. This means that the integration function permits to the attacker to distinguish nothing more than the property ϕ of the secret programs. As done in [14] for ANI, we provide a characterization of the most concrete attacker for which a program is safe. Consider $\eta, \phi, \rho \in uco(\wp(\Sigma^+))$ and an integration function $\mathfrak{I}$, such that ${}^{\mathrm{H}}_{+}[\eta]\mathfrak{I}(\phi \Rightarrow \rho)_{\mathtt{bca}}$ does not hold. We define the *higher-order abstract secret kernel* as the most concrete $\hat{\rho}$ more abstract than ρ such that ${}^{\mathrm{H}}_{+}[\eta]\mathfrak{I}(\phi \Rightarrow \hat{\rho})_{\mathtt{bca}}$ holds, namely $\mathcal{K}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)} \overset{\mathsf{def}}{=} \lambda\rho \,.\, \bigsqcap\{\beta \mid \rho \sqsubseteq \beta \wedge {}^{\mathrm{H}}_{+}[\eta]\mathfrak{I}(\phi \Rightarrow \beta)_{bca}\}$.

3 Semantics-based Software Watermarking

We follow the nomenclature introduced in [10] for describing the basic components of a watermarking technique for programs written in Imp and signatures $s \in \mathcal{S}$.

Stegomarker $\mathfrak{M} : \mathcal{S} \longrightarrow$ Imp, a function that generates a program which is the encoding of a given signature $s \in \mathcal{S}$, i.e., it generates the *stegomark* $\mathfrak{M}(s) \in$ Imp

Stegoembedder $\mathfrak{L}$: Imp $\times$ Imp $\longrightarrow$ Imp, a function that generates a program which is the composition of a stegomark and a cover program, the *stegoprogram* $\mathfrak{L}(P, \mathfrak{M}(s)) \in$ Imp.

Stegoextractor $\mathfrak{F}$: Imp $\longrightarrow \mathcal{S}$, a function that extracts the signature from a stegoprogram; for all $s \in \mathcal{S}$ it must be $s = \mathfrak{F}(\mathfrak{L}(P, \mathfrak{M}(s)))$.

When $\mathfrak{L}$ and $\mathfrak{M}$ are clear from the context we denote the stegoprogram $\mathfrak{L}(P, \mathfrak{M}(s))$ as P_s. The stegoextractor takes a stegoprogram, analyses it either statically or dynamically or by abstract interpretation and then it returns the signature encoded in the stegomark. It is well known [8] that static analysis can be modelled in the context of abstract interpretation, where a property is extensionally represented as a closure operator representing the abstract domain of data satisfying it. In particular, static analysis is performed as an abstract execution of programs, namely as the (fixpoint) semantics computation on the abstract domain. Instead, dynamic analysis can be modelled as an approximated observation of a potentially abstract execution since it describes partial knowledge of the execution (only on certain inputs). This means that, in all cases, the encoded signature can be seen as a property of the stegomark's semantics and therefore of the stegoprogram's semantics. In this view a stegoextractor is an abstract interpreter that executes the stegoprogram in the abstract domain $\beta \in uco(\wp(\Sigma^+))$ that allows it to observe the hidden signature. In order to deal with dynamic watermarking we need to model the enabling input that allows to extract the signature. Since in our model the residual input stream is part of the program state, the enabling input can be modelled as a state property $\eta \in uco(\wp(\Sigma))$. We consider a set $\mathcal{P} \subseteq$ Imp of cover programs and we specify a watermarking system as a tuple $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$.

Definition 3.1 [Software Watermarking System] Given $\mathfrak{L}$: Imp $\times$ Imp $\longrightarrow$ Imp, $\mathfrak{M} : \mathcal{S} \longrightarrow$ Imp and $\beta \in uco(\wp(\Sigma^+))$, the tuple $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$ is a software watermarking system for programs in $\mathcal{P}$ and signatures in $\mathcal{S}$ if $\mathfrak{M}$ is injective and there exists

$\eta \in uco(\wp(\Sigma))$ such that $\forall P \in \mathcal{P}\, \forall s \in \mathcal{S}$:

$$\{\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!\}^{\beta}_{+} = \lambda X \, . \begin{cases} \{\!|\mathfrak{M}(s)|\!\}^{\beta}_{+}(X) & \text{if } X \in \eta(\wp(\Sigma)) \\ \{\!|P|\!\}^{\beta}_{+}(X) & \text{otherwise} \end{cases}$$
$$X \in \eta(\wp(\Sigma)) \Rightarrow \{\!|\mathfrak{M}(s)|\!\}^{\beta}_{+}(X) = (\!|\mathfrak{M}(s)|\!)^{\beta}_{+}$$

This means that when computing the semantics in the abstract domain β, the stegoprogram $\mathfrak{L}(P, \mathfrak{M}(s))$ behaves like the stegomark $\mathfrak{M}(s)$ on the enabling inputs, and like the cover program P otherwise. Here $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+}$ is precisely the information representing the watermark at semantic level, namely the property of the stegomark where the signature is hidden. It is clear that, in this setting, it is possible to reduce the precise extraction of the signature to a completeness problem. To this end we associate the stegomarker $\mathfrak{M}$ with its semantic counterpart $\overline{\mathfrak{M}} : \mathcal{S} \longrightarrow uco(\wp(\Sigma^{+}))$, which encodes a signature in a semantic program property. In particular, given the watermarking system $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$ we define $\overline{\mathfrak{M}} \overset{\text{def}}{=} \lambda s. \{\varnothing, (\!|\mathfrak{M}(s)|\!)^{\beta}_{+}, \Sigma^{+}\}$ [5]. Indeed, $\overline{\mathfrak{M}}(s)$ provides a semantic representation of the signature s. Observe that, by construction, we have that $\forall s \in \mathcal{S} \, . \, \beta \sqsubseteq \overline{\mathfrak{M}}(s)$ and this ensures that β is precise enough for extracting the signature.

Moreover, the abstract semantics computed on β of the stegoprogram reveals the watermark information $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+} \in \overline{\mathfrak{M}}(s)$ under the enabling input $X \in \eta$ only if it is $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)$. This means that the stegoembedder makes programs in a way that the stegoextractor has a full comprehension of their semantics and so it is able to extract the property which represents the signature.

If $\{\!|P_s|\!\}^{\beta}_{+}$ is $\mathcal{F}$-complete then $\{\!|P_s|\!\}^{\beta}_{+} \circ \eta = \overline{\mathfrak{M}}(s) \circ \{\!|P_s|\!\}^{\beta}_{+} \circ \eta$ holds. When the input X belongs to η, we have that $\{\!|P_s|\!\}^{\beta}_{+}(X) = \overline{\mathfrak{M}}(s) \circ \{\!|P_s|\!\}^{\beta}_{+}(X)$ and consequently we have that $\{\!|P_s|\!\}^{\beta}_{+}(X) \in \overline{\mathfrak{M}}(s)$. This means that $\{\!|P_s|\!\}^{\beta}_{+}(X)$ is an element of $\overline{\mathfrak{M}}(s)$ and, excluding the non interesting case where $X = \varnothing$ or $X = \Sigma$, it is precisely $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+}$, so it represents the signature s. If X does not belong to η, the system should guarantee that the abstraction of the stegoprogram doesn't reveal the signature, so we have to chose β in a way that $\overline{\mathfrak{M}}(s)(\{\!|P|\!\}^{\beta}_{+}(X)) = \Sigma^{+}$ minimizes false positive. Note that if the abstract semantics of the stegoprogram is complete, it may well happen that the concrete semantics of the stegoprogram is not complete, i.e., $\{\!|P_s|\!\}_{+}$ is not $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)$. This means that the knowledge of the stegomark may not be sufficient in order to extract the signature without knowing the semantic property used to embed it.

The different kinds of software watermarking techniques can be seen as instances of Definition 3.1.

- Static and abstract watermarking correspond to a system where $\eta = id$ and β is decidable (i.e., implementable with static analysis). This captures the fact that the interpretation of the stegoprogram always reveals the stegomark, independently from the input.

[5] This is the atomic closure of $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+}$.

- Dynamic watermarking corresponds to a system where $\eta \neq id$ and β is a generic (concrete) interpreter. In this case the concrete semantics of the stegoprogram reveals the stegomark only when a particular input sequence is given.

Now we provide a semantic formalization of the features typically used to measure the quality of a watermarking system. Of course there are features strictly related to implementation, like data-rate [5] or credibility, for which we do not provide a characterization.

3.1 Resilience

Resilience concerns the capacity of a software watermarking system to be immune to attacks. There exist four types of attacks [4]: *distortive attacks*, that change the stegoprogram in order to compromise the extraction of the stegomark; *collusive attacks*, that compare different stegoprograms of the same cover program in order to obtain information on the stegomark; *subtractive attacks*, that try to eliminate the stegomark from the stegoprogram; *additive attacks*, that add another stegomark into the stegoprogram. Observe that subtractive attacks and collusive attacks are related to the localization of the stegomark and the resilience to these attacks reduces to problems of secrecy (explained below). In fact, following [4], we denote as subtractive only the attacks which locate, in some way, the stegomark. Those which perform a subtractive attack without knowing anything about the embedded watermark by creating a functionally equivalent program without the signature, in our work are considered as distortive attacks (they can be seen as distortive attacks which preserve the denotational semantics). Resilience to additive attacks is very difficult to obtain; in fact, if an attacker adds another signature (with another technique) it is practically impossible to prove which stegomark was inserted first. For this reason in the following we focus on the resilience to distortive attacks.

A distortive attack can be seen as a program transformer $\mathfrak{t} : \textsc{Imp} \longrightarrow \textsc{Imp}$ that modifies programs preserving their functionality. So there will be program properties that the attacker preserves and others that it does not preserve. According to [11] we denote with $\delta_{\mathfrak{t}} \in uco(\wp(\Sigma^+))$ the most concrete property preserved by transformation $\mathfrak{t}$ on program semantics, namely such that $\forall P \in \textsc{Imp} \,.\, \delta_{\mathfrak{t}}((\!|P|\!)_+) = \delta_{\mathfrak{t}}((\!|\mathfrak{t}(P)|\!)_+)$. Observe that when $\delta_{\mathfrak{t}} \sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) \mid s \in \mathcal{S}\}$ it means that the attacker preserves the semantic encoding of all the signatures and therefore the watermarking system is resilient against $\mathfrak{t}$. Otherwise, it could be that $\mathfrak{t}$ preserves $\overline{\mathfrak{M}}(s)$ for only certain signatures, in particular for those which $\delta_{\mathfrak{t}} \sqsubseteq \overline{\mathfrak{M}}(s)$. So we can characterize which stegoprograms are immune to $\mathfrak{t}$ and which are not. In the worst case, when $\forall s \in \mathcal{S} \,.\, \delta_{\mathfrak{t}} \not\sqsubseteq \overline{\mathfrak{M}}(s)$, the software watermarking system is not able to fend off the attacker $\mathfrak{t}$. This leads to the definition of the following levels of resilience.

Definition 3.2 [$\mathfrak{t}$-resilience] A software watermarking system $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$ is:

- $\mathfrak{t}$-resilient, when $\delta_{\mathfrak{t}} \sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) \mid s \in \mathcal{S}\}$
- $\mathfrak{t}$-vulnerable, when $\exists s \in \mathcal{S} \,.\, \delta_{\mathfrak{t}} \not\sqsubseteq \overline{\mathfrak{M}}(s)$
- $\mathfrak{t}$-ineffective, when $\forall s \in \mathcal{S} \,.\, \delta_{\mathfrak{t}} \not\sqsubseteq \overline{\mathfrak{M}}(s)$

Often distortive attacks use code obfuscation for modifying programs while preserving their functionality, and obfuscating transformations typically preserve the denotational semantics of programs, $\texttt{DenSem} \in uco(\wp(\Sigma^+))$[6]. For this reason we say that a watermarking system is *resilient* when it is $\mathfrak{t}$-resilient to all those distortive attacks $\mathfrak{t}$ that preserve $\texttt{DenSem}$, i.e., when $\texttt{DenSem} \sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) \mid s \in \mathcal{S}\}$. A software watermarking system which exhibits such behaviour has not yet been found and it is an open research topic to demonstrate its existence or not[7].

This formalization of resilience allows us to compare two watermarking systems w.r.t. resilience. Given two software watermarking systems $\mathfrak{A}_1 = \langle \mathfrak{L}_1, \mathfrak{M}_1, \beta_1 \rangle$ and $\mathfrak{A}_2 = \langle \mathfrak{L}_2, \mathfrak{M}_2, \beta_2 \rangle$, if it holds that $\bigsqcap\{\overline{\mathfrak{M}}_1(s) \mid s \in \mathcal{S}\} \sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}_2(s) \mid s \in \mathcal{S}\}$ then we have that $\{\mathfrak{t} \mid \delta_{\mathfrak{t}} \sqsubseteq \{\overline{\mathfrak{M}}_1(s) \mid s \in \mathcal{S}\}\}$ is contained in $\{\mathfrak{t} \mid \delta_{\mathfrak{t}} \sqsubseteq \{\overline{\mathfrak{M}}_2(s) \mid s \in \mathcal{S}\}\}$. Therefore $\mathfrak{A}_2$ is, in general, more resilient than $\mathfrak{A}_1$.

3.2 Secrecy

Secrecy concerns the difficulty of recovering the stegomark embedded in a stegoprogram. A watermarking system is secret when it is impossible to extract the signature from a stegoprogram without knowing the stegoextractor. In practice, secrecy can be seen as the ability of the watermarking system to make indistinguishable to the attacker a set of signatures embedded in a program. This clearly relates to the resilience to collusive attacks, which requires that an attacker is not able to distinguish between stegoprograms that embed different signatures in the same cover program. This notion can be formalized in terms of HOANI where the private input is the set of possible stegomarks $\mathsf{Q} = \{\mathfrak{M}(s) \mid s \in \mathcal{S}\}$, while the public input is the set of cover programs $\mathsf{P} = \mathcal{P}$. Let $\phi \in uco(\wp(\Sigma^+))$ be a property that represents some stegomarks, and indeed some signatures. We assume that the attacker doesn't have access to cover programs, so the abstraction of the public input is *id*.

Definition 3.3 [ϕ-secrecy] A software watermarking system $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$ is ϕ-secret w.r.t. an attacker ρ if $\overset{\mathrm{H}}{+}[id]\mathfrak{L}(\phi \Rightarrow \rho)_{\texttt{bca}}$ holds, i.e., if $\forall P \in \mathsf{P}\, \forall Q_1, Q_2 \in \mathsf{Q}$ we have that: $(\!|Q_1|\!)^{\phi}_{+} = (\!|Q_2|\!)^{\phi}_{+} \Rightarrow (\!|\mathfrak{L}(P, Q_1)|\!)^{\rho}_{+} = (\!|\mathfrak{L}(P, Q_2)|\!)^{\rho}_{+}$.

This means that if we mark a cover program with two different signatures that are equivalent in ϕ, then the attacker ρ does not distinguish between the two generated stegoprograms. Thus, any signature with the same property ϕ can be used for generating stegoprograms resilient to collusive attacks from the attacker ρ. We say that a system is *secret* when it is $\top$-secret, meaning that the set of indistinguishable signatures is $\mathcal{S}$. Given a property ϕ, specifying a set of signatures, we can characterize the most concrete observer $\hat{\rho}$ for which $\overset{\mathrm{H}}{+}[id]\mathfrak{L}(\phi \Rightarrow \hat{\rho})_{\texttt{bca}}$ holds, called *most powerful ϕ-secret attacker*. It can be characterized in terms of the secret kernel of higher-order abstract non-interference. Indeed it corresponds to the most concrete domain $\hat{\rho}$ more abstract than *id* such that $\overset{\mathrm{H}}{+}[id]\mathfrak{L}(\top \Rightarrow \hat{\rho})_{\texttt{bca}}$

[6] This domain can be obtained from maximal finite traces semantics with the abstraction $\texttt{DenSem}(X) = \{\sigma \in \Sigma^+ \mid \exists \sigma' \in X \,.\, \sigma_0 = \sigma'_0 \wedge \sigma_{|\sigma|-1} = \sigma'_{|\sigma|-1}\}$.

[7] The results in [1] and recently in [12] about impossibility of watermarking seem to lead to a negative answer.

holds, i.e., $\hat{\rho} = \mathcal{K}^{\mathrm{H}+}_{\mathfrak{L},id,(\phi)}(id)$. For example, the most powerful $\top$-secret attacker is $\mathcal{K}^{\mathrm{H}+}_{\mathfrak{L},id,(\top)}(id) = \{X \in \wp(\Sigma^+) \mid P \in \mathsf{P}, X = \bigcup_{Q \in \mathsf{Q}} (\!|\mathfrak{L}(P,Q)|\!)_+\} \cup \{\Sigma^+\}$ and it abstracts in the same object the traces of all possible stegoprograms related to the same cover program. Of course, any attacker with at least the same precision of the extractor β violates the secrecy property. Thus, the secrecy level of a watermarking system is given by the most abstract property ϕ and by the most concrete observer $\hat{\rho}$ for which non-interference $\overset{\mathrm{H}}{+}[id]\mathfrak{L}(\phi \Rightarrow \hat{\rho})_{\mathtt{bca}}$ holds. The more ϕ is abstract, the more the system is secret. Vice versa, more $\hat{\rho}$ is concrete and more the system is secret. Observe that ϕ can range from id (all the signatures are distinguishable) to $\top$ (no signature is distinguishable). When the most powerful ϕ-secret attacker $\hat{\rho}$ is equal to $\top$ then every attacker is able to distinguish the signatures. Otherwise, the more $\hat{\rho}$ is concrete, the more secret the system is.

This formalization of secrecy allows us to compare two watermarking systems w.r.t. secrecy. Given two watermarking systems $\mathfrak{A}_1 = \langle \mathfrak{L}_1, \mathfrak{M}_1, \beta_1 \rangle$ and $\mathfrak{A}_2 = \langle \mathfrak{L}_2, \mathfrak{M}_2, \beta_2 \rangle$ we consider their most powerful ϕ-secret attackers $\hat{\rho}_1$ and $\hat{\rho}_2$. If $\hat{\rho}_1 \sqsubseteq \hat{\rho}_2$ we have that $\mathfrak{A}_1$ is more secret than $\mathfrak{A}_2$ w.r.t. ϕ. Indeed a stronger attacker is necessary in order to violate ϕ-secrecy in $\mathfrak{A}_1$ than in $\mathfrak{A}_2$.

3.3 Transparence

Transparence concerns the ability to make hard to discover if a generic program is a stegoprogram. A watermarking system is invisible w.r.t. an observer if the latter is not able to distinguish a generic cover program from every stegoprogram generated starting from it.

Definition 3.4 [Transparence] A software watermarking system $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$ is transparent w.r.t. an attacker $\rho \in uco(\wp(\Sigma^+))$ if $\forall P \in \mathcal{P}\, \forall s \in \mathcal{S}\,.\, (\!|P|\!)^{\rho}_{+} = (\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!)^{\rho}_{+}$.

The greatest is the set of observers for which the system is transparent, the greatest is the level of transparence. So the characterization of the most concrete observer $\tilde{\rho}$ for which the system is invisible is a good measure of the transparence of the software watermarking system. This observer $\tilde{\rho}$ is called *most powerful transparent attacker*. This attacker can be characterized with a slightly differentiation of the most powerful $\top$-secret attacker. In fact a system, in order to be invisible w.r.t. an attacker has clearly to be also $\top$-secret w.r.t. that attacker. Clearly the system is not invisible for the extractor β.

Similarly to what we have done for secrecy, given two software watermarking systems $\mathfrak{A}_1$ and $\mathfrak{A}_2$, if $\tilde{\rho}_1 \sqsubseteq \tilde{\rho}_2$ we have that $\mathfrak{A}_1$ is more transparent than $\mathfrak{A}_2$.

3.4 Accuracy

A watermarking system is accurate if it preserves the functionality of the cover program, i.e., the cover program and the stegoprogram have to exhibit the same *observable behaviour*. This concept can be defined as "behaviour as experienced by the user" [5]. Precisely, the stegoprogram can do something that the cover program doesn't do, but this *side-effects* must be not visible to the user. Clearly this definition is very loose and it depends on what the user is able to observe of program execution. We formalize this by requiring that the stegoprogram and the original program have the same *observable* denotational semantics. This means that, fixed what the user wants to (or is able to) observe, the stegoprogram and the cover program must exhibit the same input/output behaviour, w.r.t. the fixed observation level.

Definition 3.5 [Accuracy] Given a poset $\langle D_{\mathcal{O}}, \leq_{\mathcal{O}}\rangle$ and an observational abstraction $\alpha_{\mathcal{O}} : \wp(\Sigma) \longrightarrow D_{\mathcal{O}}$ such that $(\langle \wp(\Sigma), \subseteq\rangle, \alpha_{\mathcal{O}}, \alpha_{\mathcal{O}}^{+}, \langle D_{\mathcal{O}}, \leq_{\mathcal{O}}\rangle)$ is a GC, we have that a watermarking system $\langle \mathfrak{L}, \mathfrak{M}, \beta\rangle$ is accurate, w.r.t. $\alpha_{\mathcal{O}}$, if for each program $P \in \mathcal{P}$ and for each signature $s \in \mathcal{S}$ it holds that $\alpha_{\mathcal{O}}(\llparenthesis \mathfrak{L}(P, \mathfrak{M}(s))\rrparenthesis_{\texttt{DenSem}}) = \alpha_{\mathcal{O}}(\llparenthesis P\rrparenthesis_{\texttt{DenSem}})$.[8]

So the accuracy says that, for a fixed observational abstraction $\alpha_{\mathcal{O}}$, every cover program P is $\alpha_{\mathcal{O}}$-*observationally equivalent* (in the sense of [9]) to any stegoprogram P_s embedding a generic signature s.

As regarding accuracy, this is a property that is not directly comparable among different watermarking techniques since it is defined w.r.t. the observational abstraction of interest. Namely, we can say that a system is accurate and another system is not accurate, w.r.t. an observational abstraction, but we cannot say that a system is more accurate than another. However, the proposed formal framework provides the right setting for formally proving the accuracy of a watermarking system w.r.t. a specific observational property.

4 Model Validation

In order to validate our model we have formalized two known watermarking techniques, one dynamic and one static, in our framework (the case of abstract watermarking is immediate). Doing so we want emphasise our main claim, i.e., that static and dynamic watermarking are instances of abstract watermarking.

4.1 Path-based watermarking

One dynamic technique, conceived by *Collberg et al.*[2], that encodes the signature (a natural number) in the sequence of choices (true/false) made at conditional statements during a particular execution of the program. This execution is generated by a particular sequence of enabling input values. The embedder takes the program code and it adds bogus branches in order to generate the desired false/true sequence when executed on the enabling input.

Let $I_0, I_1, \ldots I_k$ be the enabling input, i.e., the sequence of input values which "activates" the watermark. The embedder takes the program and it adds bogus branches in a way that the sequence of choices at conditional statements during the execution on the enabling input is equal to the binary encoding of the signature.

[8] Here $\llparenthesis P\rrparenthesis_{\texttt{DenSem}}$ is the angelic denotational semantics of [6]. Note that $\llparenthesis P\rrparenthesis_{\texttt{DenSem}}$ is isomorphic to $\texttt{DenSem}(\llparenthesis P\rrparenthesis_{+})$, so both formulations indicate the denotational semantics of P.

Let $\mathtt{Bin} : \mathbb{N} \longrightarrow \{0,1\}^\star$ be a function that returns the binary encoding of a number and $\mathtt{Branch} : \Sigma^+ \longrightarrow \{0,1\}^\star$ be a function that extracts the sequence of choices at conditional statements in a trace. For example, for a tt guard it can be assigned the value 1 and it can be assigned the value 0 for a ff guard. Let $\mathcal{E} : \mathbb{N} \longrightarrow \wp(\Sigma^+)$ be the function $\mathcal{E} \overset{\mathsf{def}}{=} \lambda k \,.\, \{\sigma \in \Sigma^+ \mid |\sigma| = n+1 \wedge \mathtt{Branch}(\sigma) = \mathtt{Bin}(k) \wedge \sigma_n = \langle C, \langle \rho, \iota \rangle\rangle \wedge \mathtt{top}(\iota) = \epsilon\}$[9]. The semantics $(\!|P|\!)^{\beta}_{+}$ has to extract the sequence of choices at conditional statements for the program P, so the domain β is $\beta \overset{\mathsf{def}}{=} \{X \in \wp(\Sigma^+) \mid k \in \mathbb{N}, X = \mathcal{E}(k)\} \cup \{\varnothing, \Sigma^+\}$ and it contains all the sets of traces which have done the same choices, when all the input values are consumed. With $\mathcal{W}_s = \mathcal{E}(s)$ we indicate the set of traces for which, when all the input values are consumed, the sequence of choices at conditional statements codify the signature s.

This is a dynamic technique, so $\eta = \wp(\mathcal{I}) \cup \{\Sigma\}$, where $\mathcal{I}$ represents the set of states enabling the watermark, i.e., $\mathcal{I} \overset{\mathsf{def}}{=} \{\varsigma \in \Sigma \mid \varsigma = \langle C, \langle \rho, \iota \rangle\rangle \wedge |\iota| = |I| \wedge \forall j \in [0, |I|) \,.\, \mathtt{top}(\mathtt{next}(\iota)^j) = I_j\}$. Clearly $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+} = \mathcal{W}_s$ and so $\overline{\mathfrak{M}}(s) = \{\varnothing, \mathcal{W}_s, \Sigma^+\}$. Let $\overline{\mathbb{N}} \overset{\mathsf{def}}{=} \{\bot, \mathbb{N}\} \cup \mathbb{N}$. The domain β can be defined as $\beta \overset{\mathsf{def}}{=} \beta_\gamma \circ \beta_\alpha$ where $\beta_\alpha : \wp(\Sigma^+) \longrightarrow \overline{\mathbb{N}}$ and $\beta_\gamma : \overline{\mathbb{N}} \longrightarrow \wp(\Sigma^+)$ are

$$\beta_\alpha \overset{\mathsf{def}}{=} \lambda X \,.\, \begin{cases} \bot & \text{if } X = \varnothing \\ k \in \mathbb{N} & \text{if } \forall \sigma \in X : \begin{array}{l} |\sigma| = n+1, \sigma_n = \langle C, \langle \rho, \iota \rangle\rangle, \\ \mathtt{top}(\iota) = \epsilon, \mathtt{Branch}(\sigma) = \mathtt{Bin}(k) \end{array} \\ \mathbb{N} & \text{otherwise} \end{cases} \qquad \beta_\gamma \overset{\mathsf{def}}{=} \lambda k \,.\, \begin{cases} \varnothing & \text{if } k = \bot \\ \mathcal{E}(k) & \text{if } k \in \mathbb{N} \\ \Sigma^+ & \text{otherwise} \end{cases}$$

If $X \in \eta(\wp(\Sigma))$ then $X \subseteq \mathcal{I}$, therefore the choices at conditional statements made by $\mathfrak{L}(P, \mathfrak{M}(s))$ starting from states in X are equal to $\mathtt{Bin}(s)$, i.e., $(\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!)^{\beta}_{+}(X) = \mathcal{W}_s$. The same reasoning can be done for $\mathfrak{M}(s)$, because it codifies the signature by design (starting from the sets of input states which encode the enabling input) and therefore $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+}(X) = \mathcal{W}_s$ for every $X \in \eta(\wp(\Sigma))$.

If $X \notin \eta(\wp(\Sigma))$ then $X \nsubseteq \mathcal{I}$ and therefore the choices at conditional statements made by $\mathfrak{L}(P, \mathfrak{M}(s))$ starting from states in X are not equal to $\mathtt{Bin}(s)$. So, when the set of initial states X encodes the enabling input, we have that both $(\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!)^{\beta}_{+}(X)$ and $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+}(X)$ are equal to $\mathcal{W}_s$, which represents the signature s. We can also note that, as expected, for every signature s, we have that $(\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!)^{\beta}_{+}$ is $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)$. The system is not resilient since it is not immune to distortive attacks that preserve the denotational semantics, i.e., $\mathtt{DenSem} \not\sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) \mid s \in \mathcal{S}\}$. Indeed the system is vulnerable to control flow obfuscation techniques (like edge-flipping and opaque predicate insertion attacks).

4.2 Static graph-based watermarking

One static technique, conceived by *Venkatesan et al.*[16], that codifies the signature (a natural number) as a graph which is added to the CFG (*Control Flow Graph*) of the cover program while preserving its semantics. In particular, a program whose CFG is equal to the graph generated starting from the signature is derived and then

[9] $\mathtt{top}(\iota)$ returns the current input value to be passed to the program and $\mathtt{next}(\iota)$ returns the tail of the input sequence ι.

added to the cover program's CFG, in a way that its semantics remains unmodified. The nodes of the added graph are marked before the embedding, in order to be identifiable at extraction time.

The embedder takes the program and it adds bogus code in a way that the CFG of the transformed program contains a graph which is the encoding of the signature. The basic blocks that form this graph are marked, in order to be distinguishable from the basic blocks of the original program.

Let $\mathcal{E} : \mathbb{N} \longrightarrow \mathbb{G}$ be a function that codify a signature in a graph. Let $\texttt{Mark} : \Sigma^+ \longrightarrow \mathbb{G}$ be a function that, given a trace σ, outputs the marked subgraph of the CFG of σ for a certain marking criterion [10] . The semantics $(\!|P|\!)^{\beta}_{+}$ extracts the marked subgraph of the CFG of P, so the extraction domain β is $\beta \overset{\text{def}}{=} \{X \in \wp(\Sigma^+) \mid \exists g \in \mathbb{G} \,.\, X = \{\sigma \in \Sigma^+ \mid \texttt{Mark}(\sigma) = g\}\} \cup \{\varnothing, \Sigma^+\}$. In β there are all the sets of traces whose CFG contains the same marked graph. With $\mathcal{W}_s \overset{\text{def}}{=} \{\sigma \in \Sigma^+ \mid \mathcal{E}(s) = \texttt{Mark}(\sigma)\}$ we indicate the set of traces whose CFG contains the marked graph which codify the signature s. This is a static technique so $\eta = id$. Clearly $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+} = \mathcal{W}_s$ and so $\overline{\mathfrak{M}}(s) = \{\varnothing, \mathcal{W}_s, \Sigma^+\}$. Let $\overline{\mathbb{G}} \overset{\text{def}}{=} \{\bot, \mathbb{G}\} \cup \mathbb{G}$. The domain β can be defined as $\beta \overset{\text{def}}{=} \beta_\gamma \circ \beta_\alpha$ where $\beta_\alpha : \wp(\Sigma^+) \longrightarrow \overline{\mathbb{G}}$ and $\beta_\gamma : \overline{\mathbb{G}} \longrightarrow \wp(\Sigma^+)$ are

$$\beta_\alpha \overset{\text{def}}{=} \lambda X \,.\, \begin{cases} \bot & \text{if } X = \varnothing \\ g & \text{if } \forall \sigma \in X \,.\, g = \texttt{Mark}(\sigma) \\ \mathbb{G} & \text{otherwise} \end{cases} \qquad \beta_\gamma \overset{\text{def}}{=} \lambda g \,.\, \begin{cases} \varnothing & \text{if } g = \bot \\ \{\sigma \in \Sigma^+ \mid g = \texttt{Mark}(\sigma)\} & \text{if } g \in \mathbb{G} \\ \Sigma^+ & \text{otherwise} \end{cases}$$

The input domain is *id* so there is not an enabling input, or equivalently, all the inputs reveal the watermark. Thus, for every possible set of initial states, the CFG of $\mathfrak{L}(P, \mathfrak{M}(s))$ is the same, i.e., it exists $g \in \mathbb{G}$ such that $\forall \sigma \in (\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!)_{+}$ we have $g = \texttt{CFG}(\sigma)$. For how the technique is designed, into g there is a marked subgraph equal to $\mathcal{E}(s)$. So we have that $(\!\{\mathfrak{L}(P, \mathfrak{M}(s))\}\!)^{\beta}_{+}(X) = \mathcal{W}_s$ for every possible set of initial states. Now, the CFG of $\mathfrak{M}(s)$ is exactly $\mathcal{E}(s)$ and it is marked by design, so $(\!\{\mathfrak{M}(s)\}\!)^{\beta}_{+}(X) = \mathcal{W}_s$ for every possible set of initial states. So, for every set of initial states X, we have that both $(\!\{\mathfrak{L}(P, \mathfrak{M}(s))\}\!)^{\beta}_{+}(X)$ and $(\!\{\mathfrak{M}(s)\}\!)^{\beta}_{+}(X)$ are equal to $\mathcal{W}_s$, which represents the signature s. We can also note that, as expected, for every signature s, $(\!\{\mathfrak{L}(P, \mathfrak{M}(s))\}\!)^{\beta}_{+}$ is $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)$.

The system is not resilient, since it is not immune to distortive attacks that preserve the denotational semantics, i.e., $\texttt{DenSem} \not\sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) \mid s \in \mathcal{S}\}$. Indeed the system is vulnerable to control flow obfuscation techniques (like a CFG flattening attack).

5 Conclusion

In this paper we introduce a semantics-based definition of software watermarking and of its qualifying features that is general enough to allow the specification of the static, abstract and dynamic watermarking techniques. Indeed, all these techniques can be seen as the exploitation of a completeness hole for the insertion of the signature in

[10] Building the CFG and locating its marked nodes are both tasks easily implementable by analysing program traces.

an efficient way. Only attacks that are complete w.r.t. the semantic encoding of the signature are able to observe the signature and potentially tamper with it. This means that the abstract domain used for the semantic encoding of the signature $\overline{\mathfrak{M}}(s)$ acts like a secret key that allows to disclose the signature to attackers that are complete w.r.t. $\overline{\mathfrak{M}}(s)$.

Regarding the quality of a watermarking scheme our general framework provides a formal setting in which to prove the efficiency of a watermarking scheme w.r.t. resilience, secrecy, transparence and accuracy. To validate our theory we have proved the efficiency of two known watermarking systems. Thus, we provide a general theory where researchers can reach a formal evidence of the quality of the watermarking system that they propose. We believe that this is an important contribution that can be considered as the first step towards a formal theory for software watermarking where new and existing techniques can be certified w.r.t. their efficiency.

References

[1] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 1–18. Springer-Verlag, 2001.

[2] C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn, and M. Stepp. Dynamic path-based software watermarking. *SIGPLAN Not.*, 39(6):107–118, 2004.

[3] C. Collberg and C. Thomborson. Watermarking, tamper-proofing, and obduscation-tools for software protection. *IEEE Trans. Software Eng.*, pages 735–746, 2002.

[4] C. Collberg and C. D. Thomborson. Software watermarking: models and dynamic embeddings. In *POPL '99: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 311–324. ACM, 1999.

[5] C. Collberg, C. D. Thomborson, and D. Low. A taxionomy of obfuscating transformations. Technical Report 148, Department of Computer Science, The University of Auckland, 1997.

[6] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theor. Comput. Sci.*, 277(1-2):47–103, 2002.

[7] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages (POPL '77)*, pages 238–252. ACM Press, 1977.

[8] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the 6th ACM Symposium on Principles of Programming Languages (POPL '79)*, pages 269–282. ACM Press, 1979.

[9] P. Cousot and R. Cousot. Systematic design of program transformation frameworks by abstract interpretation. In *Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 178–190. ACM Press, 2002.

[10] P. Cousot and R. Cousot. An abstract interpretation-based framework for software watermarking. In *Conference Record of the Thirtyfirst Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 173–185. ACM Press, New York, NY, 2004.

[11] M. Dalla Preda and R. Giacobazzi. Semantic-based code obfuscation by abstract interpretation. *Journal of Computer Security*, 17(6):855–908, 2009.

[12] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *IACR Cryptology ePrint Archive*, 2013.

[13] R. Giacobazzi. Hiding information in completeness holes - new perspectives in code obfuscation and watermarking. In *Proc. of The 6th IEEE International Conferences on Software Engineering and Formal Methods (SEFM'08)*, pages 7–20. IEEE Press., 2008.

[14] R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *Proc. of the 31st Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL '04)*, pages 186–197. ACM-Press, 2004.

[15] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretation complete. *Journal of the ACM*, March 2000.

[16] Ramarathnam Venkatesan, Vijay Vazirani, and Saurabh Sinha. A graph theoretic approach to software watermarking. In Ira Moskowitz, editor, *Information Hiding*, volume 2137 of *Lecture Notes in Computer Science*, pages 157–168. Springer Berlin / Heidelberg, 2001.