On the Rôle of Abstract Non-Interference in Language-based Security

Isabella Mastroeni

Department of Computing and Information Sciences - Kansas State University, Manhattan, Kansas, USA (isabellm@cis.ksu.edu)

Abstract. In this paper, we illustrate the rôle of the notion of Abstract Non-Interference in language based security, by explaining how it models both the weakening of attackers' observational capability, and the declassification of private information. Namely, we show that in abstract non-interference we model both attackers that can only observe properties of public data, and private properties that can or cannot flow. Moreover, we deepen the understanding of abstract non-interference by comparing it, by means of examples, with some the most interesting approaches to the weakening of non-interference, such as the PER model, robust declassification, delimited release and relaxed non-interference.

Keywords: Language-based Security, Non-Interference, Declassification.

1 Introduction

An important task of language based security is to protect confidentiality of data manipulated by computational systems. Namely, it is important to guarantee that no information, about confidential/private data, can be caught by an external viewer. The standard way used to protect private data is access control: special privileges are required in order to read confidential data. Unfortunately, these methods allow to restrict accesses to data but cannot control propagation of information. Once the information is released from its container, it can be improperly transmitted without any further control. This means that security mechanisms such as digital signature and antivirus scanning, do not provide assurance that confidentiality is maintained during the whole execution of the checked program. This implies that, if a user wishes to keep some data confidential, he might state a policy stipulating that no data visible to other users is affected, within the executed program, by modifying confidential data. This policy allows programs to manipulate and modify private data, as long as visible outputs of those programs do not reveal information about these data. A policy of this sort is called *non-interference* policy [13], since it states that confidential data may *not interfere* with public data, but it is also referred as *secrecy* [25].

The standard approach to non-interference, is based on a characterization of attackers that does not impose any observational or complexity restriction on the attackers' power. This means that the attackers are *all-powerful*, modeled without any limitation in their quest to obtain confidential information. For this reason non-interference, as defined in literature, is an extremely restrictive policy. The problem of refining this kind of security policies has been addressed by many authors as a major challenge in language-based information flow security [22], where refining security policies means weakening standard non-interference checks. In order to adapt security policies to practical cases, we need a weaker notion of non-interference where the power of the attacker (or external viewer) is bounded, and where intentional leakage of information is allowed. In [9], we introduce a weak notion of non-interference for characterizing the secrecy degree of programs by identifying the most powerful attacker that is not able to disclose confidential information by observing the execution of programs, but also in order to characterize the maximal amount of information released. Clearly, this is not the only, and not even the first attempt to weaken the notion of non-interference, from both these points of view, as we will see in Sect. 3. But this is the only one, to the best of our knowledge, that can model, in the same formalism, both weaker attackers and released information. In this paper we show, by means of examples, that different techniques proposed in literature for weakening non-interference (e.g., PER model) and for declassifying information (e.g., selective dependency, delimited release, relaxed non-interference) can be equivalently formalized in abstract non-interference, but moreover we show that in some cases abstract non-interference captures a much more precise notion and allows to derive certifications of the security degree of programs. The aim of this paper is to show how abstract non-interference can be simply adapted in order to cope with different problems concerning secure information flow, and to allow a deep insight of this notion.

2 Abstract Interpretation: A panoramic view

In the following of this paper we will use the standard framework of abstract interpretation [5, 6] for modeling the observational capability of attackers. The idea is that, instead of observing the concrete semantics of programs, namely the concrete values of public data, the attackers can only observe *properties* of public data, namely *abstract semantics* of the program. For this reason we model attackers by means of abstract domains. Abstract domains are used for denoting properties of concrete domains, since their mathematical structure guarantees, for each concrete element, the existence of the *best correct approximation* in the abstract domain. This is due to the property, of abstract domains, of being closed under the concrete greatest lower bound. So for example an abstract do-

main for the sign analysis is $Sign \stackrel{\text{def}}{=} \{\mathbb{Z}, 0+, 0-, \{0\}, \varnothing\}$[1], while if we weed out $\{0\}$, then it is no more an abstract domain. Formally, the *lattice of abstract interpretations of* C is isomorphic to the lattice $uco(C)$ of all the upper closure operators on C [6]. An *upper closure operator* $\rho : C \to C$ on a poset C is monotone, idempotent, and extensive[2]. Closure operators are uniquely determined by the set of their fix-points $\rho(C)$, which are abstract domains formalized independently of the representation of their objects. If C is a complete lattice then $\langle uco(C), \sqsubseteq, \sqcup, \sqcap, \lambda x.\ \top, \lambda x.\ x\rangle$ is the lattice of upper closures, where for every $\rho, \eta \in uco(C)$, $\{\rho_i\}_{i\in I} \subseteq uco(C)$ and $x \in C$: $\rho \sqsubseteq \eta$ iff $\eta(C) \subseteq \rho(C)$; $\sqcup_{i\in I}\rho_i = \bigcap_{i\in I} \rho_i$; and $\sqcap_{i\in I}\rho_i = \mathcal{M}(\bigcup_{i\in I} \rho_i)$, where $\mathcal{M}$ is the operation of closing a domain by concrete greatest lower bound, e.g., intersection on power domains.

3 Non-Interference and Declassification: The road map

Non-interference for programs essentially means that *"a variation of confidential (high or private) input does not cause a variation of public (low) output"* [22]. When this happens, we say that the program has only *secure information flows* [1, 4, 7, 15, 25]. This situation has been modeled by considering the denotational (input/output) semantics $[\![P]\!]$ of the program P. In the following, we consider programs where data are typed as private (H) or public (L). Program states, whose set is Σ, are functions (represented as tuples) mapping variables in the set of values $\mathbb{V}$. Any input state s, can be seen as a pair (h, l), where $s^{\mathtt{H}} = h$ is a value for private data and $s^{\mathtt{L}} = l$ is a value for public data. In this case, *(standard) non-interference* can be formulated as follows: Let P be a program

$$\boxed{P \text{ is } \textit{secure} \text{ if } \forall\ s, t \in \Sigma.\ s^{\mathtt{L}} = t^{\mathtt{L}} \ \Rightarrow\ ([\![P]\!](s))^{\mathtt{L}} = ([\![P]\!](t))^{\mathtt{L}}}$$

This notion has been formulated also as a *Partial Equivalence Relation* [14, 23].

The limitation of this notion of non-interference is that it is an extremely restrictive policy. Indeed, non-interference requires that *any* change upon confidential data has not to be revealed through the observation of public data. There are at least two problems with this approach. On one side, many real systems are intended to leak some kind of information. On the other side, even if a system satisfies non-interference, too restrictive tests could reject it as insecure. These observations address the problem of *weakening* the notion of non-interference both characterizing the information that is *allowed* to flow, and considering *weaker* attacker models, that can observe only properties of public data. The first problem is treated by means of *declassification* of private information, while the second problem characterizes the observational capability of the attacker.

[1] $0+ \stackrel{\text{def}}{=} \{n \mid n \geq 0\}$, $0- \stackrel{\text{def}}{=} \{n \mid n \leq 0\}$.
[2] $\forall x \in C.\ x \leq_C \rho(x)$.

Different notions of non-interference dealing with declassification have been introduced. The first formalization is *selective dependency* [4], where a property on private input is declassified, by letting the private input range only upon sets of confidential values with the same declassified property. In this case non-interference is checked only whenever a fixed input property holds. So, if we consider the property $\varphi(h) : h \ mod \ 3 = 1$, then non-interference is checked only when the input h satisfies φ. Declassification is considered in presence of active attackers, i.e., attackers that can modify the code of the program in *robust declassification* [26]. Declassification is robust whenever an active attacker cannot disclose more information than what can be observed by a passive attacker. This notion is then better formalized in [19], where also a type system is provided for enforcing robust declassification. More recently two other weakenings of non-interference are provided for dealing with declassification: *delimited release* [21] and *relaxed non-interference* [17]. Both these notions enforce the respective non-interference with a type system, the first on a simple imperative language with explicit declassification of expressions, and the second on λ-calculus. The end-to-end policy in both of them is the same, and it corresponds to selective dependency [4]. In [17] a method is also provided for deriving the declassification policy that makes the program satisfying relaxed non-interference. All these approaches are *qualitative*. There exist also two approaches for *quantifying* the information released [3, 18].
As far as the model of the attacker is concerned, a possible limitation of the attackers consists in restricting the possible computational complexity of attackers [8, 16], but we can also think of defining non-interference parametric on what the attacker can observe of public data. This model can be given in terms of equivalence relations [14, 23, 26] or in terms of abstract domains [9–12].

4 Abstract Non-Interference for modeling attackers

Consider the program $P \stackrel{\text{def}}{=} l := |l| * \mathit{Sign}(h)$, suppose that $|\cdot|$ is the absolute value function and suppose $\mathit{Sign}(h)$ returns the sign of h, then "*only a portion of l is affected [by h], in this case l's sign. Imagine if an observer could only observe l's absolute value and not l's sign*" [4] then we could say that the program is secure. This is the basic idea in the notion of abstract non-interference [9], used for modeling both weaker attack models and declassification. The idea is that an attacker can observe only some properties, modeled as abstract interpretations of the concrete program semantics. In the following, if $\mathtt{T} \in \{\mathtt{H}, \mathtt{L}\}$, $n = |\{x \in \mathit{Var}(P) | x \text{ of type } \mathtt{T}\}|$, and $v \in \mathbb{V}^n$, we abuse notation by denoting $v \in \mathbb{V}^{\mathtt{T}}$ the fact that v is a possible value for the variables with security type $\mathtt{T}$. The *model of an attacker*, also called *attacker*, is therefore a pair of abstractions $\langle \eta, \rho \rangle$, with $\eta, \rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$, representing what it can observe about, respectively, the public input and output of a program. We obtain so far the notion of

$$[\eta]P(\rho) \text{ if } \forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}} \,.\, \eta(l_1) = \eta(l_2) \;\Rightarrow\; \rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![P]\!](h_2, l_2)^{\mathtt{L}})$$

$$(\eta)P(\rho) \text{ if } \forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l \in \mathbb{V}^{\mathtt{L}} \,.\rho([\![P]\!](h_1, \eta(l))^{\mathtt{L}}) = \rho([\![P]\!](h_2, \eta(l))^{\mathtt{L}})$$

Table 1. Narrow and Abstract Non-Interference (without declassification).

narrow (abstract) non-interference (NANI) denoted $[\eta]P(\rho)$, provided in Table 1. It says that if the attacker is able to observe the property η of public input, and the property ρ of public output, then no information flow concerning the private input is observable from the public output. The problem with this notion is that it introduces *deceptive flows* [9], generated by different *public* output due to different *public* inputs, with the same input property η. Consider, for instance, $[Par]$**if** l **then** $l := 2h + l$ **else** $l := 2h + 1(Par)$, we can observe a variation of the output's parity due to the fact that both 0 and 2, for example, are even numbers, revealing a flow not due to different private inputs. Most known models for weakening non-interference (e.g., PER model [23]) and for declassifying information (e.g., robust declassification [26]) corresponds to instances of NANI [9, 14]. In order to avoid deceptive interference we introduce a weaker notion of non-interference which considers, as public input, the *set* of all the elements sharing the same property η. Hence, in the previous example, the observable output for l is the set of the possible values obtained by considering all the inputs with the same parity, e.g., if $Par(l) = even$ then we check the parity of $\left\{ \; 2h + l \,\middle|\, l \neq 0 \text{ is even} \; \right\} \cup \{2h + 1\}$ which is always unknown, since $2h + 1$ is always odd, and h *does not* interfere with the final parity. This notion, denoted $(\eta)P(\rho)$, is called *abstract non-interference* (ANI) without declassification, and it is formally defined in Table 1. So, if $P \stackrel{\text{def}}{=} l := |l| * Sign(h)$, and we consider the closure Abs that forgets the sign of an integer number, i.e., $Abs(n) = \{n, -n\}$, then $[Abs]P(Abs)$. While, for example if $P \stackrel{\text{def}}{=}$ **if** $h > 0$ **then** $l = m$ **else** $l = -m$, then $[id]P(Abs)$. Hence abstract non-interference allows the weakening of attackers models without introducing deceptive interference. Moreover, this abstract model can be used for characterizing the security degree of programs. In particular, the most concrete output observation for a program, given the input one, for both narrow and abstract non-interference can be systematically derived [9]. The idea is that of abstracting in the same object all the elements that, if distinguished, would generate a visible flow. These most concrete harmless attackers, with a fixed η in input, are, respectively, denoted $[\eta][\![P]\!](id)$ and $(\eta)[\![P]\!](id)$, both in $uco(\wp(\mathbb{V}^{\mathtt{L}}))$. Hence, in both the programs above, we note that each value n has to be abstracted in $\{n, -n\}$, in order not to generate visible flows, hence the most concrete harmless attacker can at most observe Abs, i.e., $[Abs][\![P]\!](id) = Abs$.

4.1 Abstract Non-Interference vs PER model

In [14], the authors analyze the relationship between abstract non-interference and the PER model of secure information flow. Given the equivalence relations All, Id such that $\forall x, y. x \; All \; y$ and $\forall x, y. \; Id \; y$ iff $x = y$, then the PER model of secure information flow [23] says that P satisfies non-interference if:

$$x \; All \times Id \; y \;\Rightarrow\; [\![P]\!](x) \; All \times Id \; f[\![P]\!](y)$$

where each state x is partitioned in its confidential and public component, i.e., $x = \langle x^{\mathtt{H}}, x^{\mathtt{L}} \rangle$. In [14] it is shown that this is an instance of NANI, since it considers particular equivalence relations, and since equivalence relations can be modeled by a subset of upper closure operators, called *partitioning* [20]. This also implies that abstract non-interference, without declassification, is strictly more general, as it is proved in [14].

4.2 Abstract Non-Interference vs Security for Robust Declassification

In [26], in order to accommodate programs that do leak confidential information, the authors allow the information flow controls to include a notion of *declassified* information, where declassifying means *downgrading* the sensitivity labels on data. Declassification is defined by saying that a *passive attacker*, i.e., attackers that can make only observations of the system and draw inference from those observations, may be able to learn some confidential information by observing the system, but by assumption, that information leakage is allowed by the security policy. The problem is that once a channel is added to the system along which sensitivity labels are downgraded, there is the potential for the channel to be abused to release sensitive information other than that intended. This is clearly possible whenever we are in presence of *active attackers*, i.e., programs running concurrently with the system. When such an attacker cannot obtain more confidential information than what a passive attacker can by simply observing the system, then we say that the system is *robust*. In other words, robust declassification guarantees that if a *passive* attackers may not distinguish between two memories where the secret part is altered, then no *active* attackers may distinguish between these two memories [24].
We compare this paper with ANI without declassification, since robust declassification considers a programming language without explicit declassification and does not characterize the information declassified. It simply derives robustness of declassification by proving a security property that is equivalent to NANI defined on trace semantics [9].

Let $S = \langle \Sigma, \rightarrow \rangle$ be a transition system, and $\langle\!|S|\!\rangle$ the induced trace semantics. Suppose the observational capability of the attacker is characterized by the

equivalence relation $\approx$, then the observation of a trace τ of S, through $\approx$ is the trace $\tau/\approx$, such that for each i, $(\tau/\approx)_i \stackrel{\text{def}}{=} [\tau_i]_\approx$[3], and the observation of S is $\langle\!|S|\!\rangle_\approx \stackrel{\text{def}}{=} \{\ \tau/\approx \mid \tau \in \langle\!|S|\!\rangle\ \}$. The set of all the traces in S starting from a state σ is denoted $\langle\!|S|\!\rangle(\sigma)$ and two traces are equivalent if they are equal up to stuttering, while two sets of traces are equivalent, i.e., $\equiv$, if all their traces are equivalent: $X \equiv Y \ \Leftrightarrow\ \forall \tau \in X\ \exists \tau' \in Y.\, \tau \equiv \tau'$ and vice versa. The security property for a system S, is $S \models \mathcal{SP}(\approx)$, which holds iff $S[\approx] \supseteq \approx$ (note that $\approx \supseteq S[\approx]$ always holds), where

$$\boxed{\forall \sigma, \sigma' \in \Sigma.\ \sigma\ S[\approx]\ \sigma' \ \Leftrightarrow \langle\!|S|\!\rangle(\sigma)_\approx \equiv \langle\!|S|\!\rangle(\sigma')_\approx}$$

Hence, $S \models \mathcal{SP}(\approx)$ holds iff $\forall \sigma, \sigma' \in \Sigma.\ \sigma \approx \sigma' \ \Rightarrow \langle\!|S|\!\rangle(\sigma)_\approx \equiv \langle\!|S|\!\rangle(\sigma')_\approx$, namely the set of the traces starting form σ and observed through $\approx$ is equivalent to the set of the $\approx$-observable traces starting from σ'. A $\approx$-attack is a system $A = \langle \Sigma, \rightarrow_A \rangle$ such that $A \models \mathcal{SP}(\approx)$, which means that the attacker does not know any secret before running with the program. At this point, given an attack A and a system S, both specified in terms of the same set of states Σ, the attack on S by A is the union of the systems: $A \cup S$, and S is *robust* w.r.t. $\approx$, if for all the $\approx$-attacks A, the system $A \cup S$, observed through $\approx$, does not release more information than what is released by S, always observed through $\approx$, i.e., $S[\approx] \subseteq (S \cup A)[\approx]$. Moreover, Theorem 4.1 in [26] says that $S \models \mathcal{SP}(\approx)$ implies the *system is robust* for all the possible $\approx$-attacks.

Consider narrow non interference defined in terms of trace semantics [12], where the abstraction ρ of a trace τ is the trace $\rho(\tau)$ such that $\forall i.\rho(\tau)_i \stackrel{\text{def}}{=} \rho(\tau_i)$. Consider the attacker modeled by using partitioning closures, i.e., equivalence relations [14], then it is straightforward to prove that narrow non-interference $\models_{\langle\!|S|\!\rangle}\ \ [\rho_\approx]S(\rho_\approx)$[4], where $\rho_\approx \stackrel{\text{def}}{=} \{\ [x]_\approx \mid x \in \mathbb{V}^{\mathtt{L}}\ \} \cup \{\top, \bot\}$, is equivalent to the property $S \models \mathcal{SP}(\approx)$ [9]. This means that, by Theorem 4.1 [26], also NANI can be used for characterizing *robust declassification*. Note that, in both these works there is not explicit declassification in the language, the idea is simply that what is naturally released from the semantics of the system has not to be exploited for obtaining much more information.

Example 1. Consider the following program and its semantics:

$$P \stackrel{\text{def}}{=}\ l := 2h + l \text{ and } \langle\!|P|\!\rangle = \{\ \langle h, l\rangle \rightarrow \langle h, 2h + l\rangle \mid h \in \mathbb{V}^{\mathtt{H}}, l \in \mathbb{V}^{\mathtt{L}}\ \}$$

This program is neither robust nor satisfies non-interference, being $S[=_L] \subsetneq =_L$ since $\langle h, l\rangle \rightarrow \langle h, 2h + l\rangle \not\approx \langle 0, l\rangle \rightarrow \langle h, l\rangle$ when $h \neq 0$ and $\approx$ requires the equality of the projection on the low values. On the other hand, if we consider

[3] τ_i denotes the i-th element of the trace τ.

[4] $\models_{\langle\!|P|\!\rangle}\ [\eta]P(\rho)$ denotes when NANI is checked by using the trace semantics $\langle\!|P|\!\rangle$.

the equivalence relation $\approx_{Par}$[5], then $\forall h_1, h_2.\ \langle h_1, l\rangle \rightarrow \langle h_1, 2h_1 + l\rangle \approx_{Par} \langle h_2, l\rangle \rightarrow \langle h_2, 2h_2 + l\rangle$, since adding $2h_i$ to l does not change l's parity. Indeed, both $S[\approx_{Par}] = \approx_{Par}$ and narrow abstract non-interference hold. Hence, the program is robust for all the attacks $A \models \mathcal{SP}(\approx_{Par})$. For instance, the attack consisting in the program $A \stackrel{\text{def}}{=} l := 2l$ cannot observe more than what a passive attacker can, from the execution of P. Indeed,

$$\langle\!|S \cup A|\!\rangle = \{\langle h, l\rangle \rightarrow \langle h, 2h + l\rangle \rightarrow \langle h, 4h + 2l\rangle, \langle h, l\rangle \rightarrow \langle h, 2l\rangle \rightarrow \langle h, 2h + 2l\rangle\}$$

and the parity of the public output in all the possible executions does not depend on the input h.

This example shows that, even if declassification in ANI may appear as "vacuously robust" [24] since we do not consider explicitly active attackers, also narrow abstract non-interference can be used for proving *robust* declassification of confidential data, without changing any definition.

5 Abstract Non-Interference for Declassification

In abstract non-interference we can also model more *selective* security properties. For example, "*we may not care if output variable b reflects whether input variable a is odd or even. However, we might like to show that b depends upon a in no other way*" [4]. We can also consider another point of view. Suppose that we do want the output variable b does not reflect whether input variable a is odd or even, but we do not care that b depends upon a in other ways. These are two aspects of declassification: the first specifies what is admitted to flow, the other specifies what cannot flow, admitting that something else may be released.

Abstract non-interference considers a confidential data property, modeled by an upper closure operator ϕ, which represents the confidential property we are interested in keeping secret. This notion is provided in the first definition in Table 2, where $(\eta)P(\phi \rightsquigarrow\!\!\!\!\!| \rho)$ means that the program P keeps secret ϕ when the attacker is modeled by the I/O pair of observable properties $\langle \eta, \rho\rangle$ and is called *declassified ANI via blocking*. So for example the property $(id)l := l{*}h^2(Sign \rightsquigarrow\!\!\!\!\!| Sign)$ is satisfied, since the public result's sign does not depend on the private input sign, which is kept secret. In this case, whenever a flow of information is revealed this can only be due to the change of the property ϕ. In this way we model a notion of non-interference which is parametric both on what the attacker can observe of public data and on the confidential property that we want to keep secret.
On the other hand, in ANI [9], we also provide a construction for characterizing

[5] $\langle h_1, l_1\rangle \approx_{Par} \langle h_2, l_2\rangle$ iff $Par(l_1) = Par(l_2)$

$$(\eta)P(\phi \rightsquigarrow \rho) \text{ if } \forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l \in \mathbb{V}^{\mathtt{L}} \,.\; \rho([\![P]\!](\phi(h_1), \eta(l))^{\mathtt{L}}) = \rho([\![P]\!](\phi(h_2), \eta(l))^{\mathtt{L}})$$

$$(\eta)P(\phi \Rightarrow \rho) \text{ if } \forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l \in \mathbb{V}^{\mathtt{L}} \,.\; \phi(h_1) = \phi(h_2) \;\Rightarrow \rho([\![P]\!](h_1, \eta(l))^{\mathtt{L}}) = \rho([\![P]\!](h_2, \eta(l))^{\mathtt{L}})$$

Table 2. Abstract Non-Interference

the maximal amount of confidential information that *flows*, for a fixed attacker model. This corresponds to characterizing the most concrete property that has to be declassified in order to make the program secure [11]. Declassification can be made explicit also in abstract non-interference, following the idea of selective dependency [4]. Namely, given the property ϕ that we want to declassify, we check abstract non-interference *only* when the private input has the same property ϕ. This notion is provided in the second definition in Table 2, where $(\eta)P(\phi \Rightarrow \rho)$ means that the program P is secure when we let ϕ to flow in presence of an attacker modeled by $\langle \eta, \rho \rangle$, and is called *declassified ANI via allowing*. So for example, consider $P \stackrel{\text{def}}{=} l := l + (h \; mod \; 3)$, if we want to understand which property flows, we have to characterize which values of the private input generate different public outputs, and we can note that all the elements in $\left\{ \; h \;\middle|\; h \; mod \; 3 = 0 \; \right\}$ generate the same output l, such as the elements in $\left\{ \; h \;\middle|\; h \; mod \; 3 = 1 \; \right\}$ and in $\left\{ \; h \;\middle|\; h \; mod \; 3 = 2 \; \right\}$. In particular, this partition is such that each pair of values in the same set generates the same public output, while each pair of values from different sets generates different public values. This is exactly the *maximal amount of information disclosed*.

The basic idea of the construction is to collect, for each possible public input, all the private inputs that give the same result as public output. In this way we obtain a partition $\Pi(\eta, \rho)_{|_L}$ for each possible property L in input:

$$\begin{array}{ll} \Pi(\eta, \rho) & \stackrel{\text{def}}{=} \left\{ \; \langle \left\{ \; h \in \mathbb{V}^{\mathtt{H}} \;\middle|\; \rho([\![P]\!](h, \eta(l))^{\mathtt{L}}) = A \; \right\}, \eta(l) \rangle \;\middle|\; l \in \mathbb{V}^{\mathtt{L}}, A \in \rho \; \right\} \\ \Pi(\eta, \rho)_{|_L} & \stackrel{\text{def}}{=} \left\{ \; H \;\middle|\; \langle H, L \rangle \in \Pi(\eta, \rho) \; \right\} \end{array}$$

So, like delimited release [21], in presence of passive attacker, it is sufficient to declassify the property depending on the public input we are observing. Hence $\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$, where $\forall L \in \eta.\pi_L \stackrel{\text{def}}{=} \Pi(\eta, \rho)_{|_L}$, we have

$$[h_1]_{\pi_L} = [h_2]_{\pi_L} \;\Rightarrow\; \rho([\![P]\!](h_1, L)^{\mathtt{L}}) = \rho([\![P]\!](h_2, L)^{\mathtt{L}}) \tag{1}$$

Let us denote this policy as $\forall L \in \eta.\, (\eta)P(\pi_L \Rightarrow \rho)$. It is clear that checking Eq. 1, could be considered sufficient in presence of only passive attackers, since the standard idea of non-interference compares only the outputs obtained from computations with a fixed public input. But is it a realistic notion of non-interference? Namely, if we have an attacker that observes a system, then we cannot avoid him to observe computations due also to different public inputs, but if this happens then the definition in Eq. 1 is no more a non-interference property. So, if

$P \stackrel{\text{def}}{=} \textbf{if } h = l \textbf{ then } l := 0 \textbf{ else } l := 1$, then the declassified information should be $\Pi(id, id)_l = \{\{l\}, \{\ h \mid h \neq l\ \}, \mathbb{Z}, \varnothing\}$. But if the attacker can control the public input, or can observe computations of P with different public inputs for l, then it could collect the results, deducing more than what is declassified about h. For this reason we are interested in a unique property to declassify, which is independent of the fixed public input, namely we derive the most abstract property containing all the properties that should be declassified for each public input and such that $(\eta)P(\phi \Rightarrow \rho)$ [9]:

$$\phi \stackrel{\text{def}}{=} \prod_{L \in \eta} \mathcal{M}(\Pi(\eta, \rho)_{|_L}) \tag{2}$$

5.1 Abstract Non-Interference vs Enforcing Robust Declassification

In [19] the notion of robust declassification [26] is expressed in a language-based setting and is generalized in order to make untrusted code and data explicitly part of the system rather than appearing only when the attacker is active. In this work the skills of the attacker are made explicit distinguishing between what it can *observe* and what it can *modify*. Hence, each variable on the program has two security classifications, the first says who can observe and the second says who can modify it, e.g., $x_{\texttt{LH}}$ means that the variable can be observed but cannot be modified by a low level user.
More precisely, in [19], the authors consider a syntax, for defining programs, which includes explicit declassification of expressions to the low security level, and which allows to leave *holes* in the code:

$$e ::= v \mid x \mid e_1 op\, e_2 \mid \mathit{declassify}(e)$$
$$c[\overrightarrow{\bullet}] ::= \textbf{nil} \mid x := e \mid \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \mid \textbf{while } e \textbf{ do } c \mid [\bullet]$$

In these holes the attacker can insert his own code, possibly modifying the computation, i.e., $c[a]$ is the program c under the attack a which means that each hole $[\bullet]$ in c is substituted by the code a. The system $c[\overrightarrow{\bullet}]$ is robust if $\forall \sigma, \sigma' \in \Sigma$ (possible initial memories) and $\forall a, a'$, when both $\langle\!|c[a]|\!\rangle(\sigma)$ and $\langle\!|c[a]|\!\rangle(\sigma')$ terminate, and $\langle\!|c[a]|\!\rangle(\sigma) \approx \langle\!|c[a]|\!\rangle(\sigma')$ then $\langle\!|c[a']|\!\rangle(\sigma) \approx \langle\!|c[a']|\!\rangle(\sigma')$, where $\approx$ requires the equality of low-level data. This definition says that a system is robust if any attack cannot change the observable computations of the system.

By considering this view of the problem, we may conclude that it is implicit in non-interference, and therefore also in abstract non-interference, that all the secure systems, where only the input public data can be modified by an untrusted user, are robust. Namely, if we consider the assumption that the attacker may be able to change inputs, i.e., *controls* the inputs, but cannot change the code itself [2], then declassification in ANI is *precisely* robust, as shown in the following examples. Clearly it is a limitation on the possible attacks, and indeed the

extension of ANI to all the possible (active) attacks described in [19] deserves further investigation. On the other hand, declassified ANI via allowing provides the construction for deriving *what* is released by a program.

Example 2. 1. Consider $P[\overrightarrow{\bullet}] \stackrel{\text{def}}{=} [\bullet]; l := l + \mathit{declassify}(h \ \mathit{mod}\ 3)$, where $l : \mathtt{LL}$ and $h : \mathtt{HH}$. This program satisfies robust declassification. On the other hand, we can derive the maximal amount of information disclosed from the program $P' \stackrel{\text{def}}{=} l := l + h \ \mathit{mod}\ 3$: $\phi = \{\top, 3\mathbb{Z}, 3\mathbb{Z}+1, 3\mathbb{Z}+2, \bot\}$, i.e., it is the most concrete property that has to be declassified in order to guarantee secrecy. We can prove that $(id)P(\phi \Rightarrow id)$, namely for each possible observation, when we declassify ϕ, the program is secure. This also prove that the program $P[\overrightarrow{\bullet}]$ is robust, since ϕ describes exactly the property $h \ \mathit{mod}\ 3$ declassified, and non-interference cannot be violated even if the attacker controls the input l.

2. Let $P[\overrightarrow{\bullet}] \stackrel{\text{def}}{=} [\bullet]; \textbf{if } \mathit{declassify}(h = l) \textbf{ then } h := h - l; \textbf{ else } l := l + h$, where $l : \mathtt{LL}$ and $h : \mathtt{HH}$. In this case the program is not robust since the attacker can modify the input of l and therefore can obtain the value of h. Indeed, if we consider ANI for the program $P' \stackrel{\text{def}}{=} \textbf{if } h = l \textbf{ then } h := h - l; \textbf{ else } l := l + h$, and we derive the most concrete property that has to be declassified in order to guarantee secrecy, we obtain the identity:

$$\Pi(id, id)_{|_l} = \left\{ \mathbb{Z}, \{l\}, \left\{ \ h \ \middle| \ h \neq l \ \right\}, \varnothing \right\} \text{ and } \phi = \bigsqcap_{l \in \mathbb{Z}} \Pi(id, id)_{|_l} = id$$

Namely, we proved that the program cannot be made secure unless we declassify the value of private data.

5.2 Abstract Non-Interference vs Delimited Release

The aim of delimited release [21] is to find a definition of non-interference which takes into account explicit declassifications in the code of programs. However, explicit declassification does not concern with what is actually leaked during the execution of the program, but simply determines the downgrading policy to check. Delimited release is a generalization of Cohen's selective dependency [4]. In standard non interference we check the security property for all the possible private inputs, this means that the private input can range over the whole domain of possible values. In delimited release the idea is to select, by using the declassification explicitly given (consider the simplified *declassify* operation used in the previous section), for which private inputs we have to check non-interference. Let $\approx_e$ the equivalence relation induced by the evaluation of the expression e, i.e., $\sigma \approx_e \sigma'$ iff $[\![e]\!](\sigma) = [\![e]\!](\sigma')$, we can define $\sigma \approx_E \sigma'$ iff $\forall e \in E. \sigma \approx_e \sigma'$. Delimited release is defined as follows: Let E be the set of all the declassified expressions in the program P

$$\boxed{\forall \sigma, \sigma' \in \Sigma. (\sigma^{\mathtt{L}} = \sigma'^{\mathtt{L}}) \ \wedge \ \sigma \approx_E \sigma' \ \Rightarrow \ [\![P]\!](\sigma)^{\mathtt{L}} = [\![P]\!](\sigma')^{\mathtt{L}}}$$

This generalizes selective dependency because $\approx_E$ is applied to the whole state and not only to the high component. But, whenever the expressions in E depend only on the high input, then the two notions collapse, corresponding also to declassified ANI via allowing. However, whenever the expression E depends also on public inputs, then we obtain a notion of declassified non-interference which depends on the fixed public input, as in Eq. 1, inheriting all the problems expressed for Eq. 1.
At this point, let us see which analogies can be found between declassified ANI and delimited release. In the following, we say that a closure ϕ models the set of declassifications E if $\approx_E \subseteq \approx_\phi$, where $\sigma \approx_\phi \sigma_1$ iff $\phi(\sigma^{\mathtt{H}}) = \phi(\sigma_1^{\mathtt{H}})$.

Example 3. 1. Consider $P \stackrel{\text{def}}{=} avg := \mathit{declassify}((h_1 + h_2 + \ldots + h_n)/n)$. This program satisfies delimited release [21], and if we derive the maximal amount of information disclosed in the program $P' \stackrel{\text{def}}{=} avg = (h_1 + h_2 + \ldots + h_n)/n$, we obtain the following closure:

$$\phi = \left\{ \left\{ \langle h_1, \ldots, h_n \rangle \,\middle|\, h_1 + \ldots + h_n = x \right\} \,\middle|\, x \in \mathbb{Z} \right\} \cup \{\mathbb{Z}, \varnothing\}$$

which models exactly the declassified property (being n a known constant).

2. Consider the program

$$P \stackrel{\text{def}}{=} \begin{bmatrix} h_1 := h_1;\ h_2 := h_1;\ \ldots h_n := h_1; \\ avg := \mathit{declassify}((h_1 + h_2 + \ldots + h_n)/n) \end{bmatrix}$$

This program does not satisfy delimited release [21], and if we consider ANI and we derive the maximal amount of information disclosed in the program without the *declassify* operation, we obtain:

$$\phi = \left\{ \left\{ \langle h, k_2, \ldots, k_n \rangle \,\middle|\, \forall i.k_i \in \mathbb{V}^{\mathtt{H}} \right\} \,\middle|\, h \in \mathbb{Z} \right\} \cup \{\mathbb{Z}, \varnothing\}$$

namely the maximal amount of information disclosed is the identity on the first private input. Hence the program is not secure, since ϕ does not model the information explicitly declassified, e.g., $\langle 1, 0, \ldots, 0 \rangle \approx_e \langle 0, 1, \ldots, 0 \rangle$ while $\phi(\langle 1, 0, \ldots, 0 \rangle) \neq \phi(\langle 0, 1, \ldots, 0 \rangle)$.

3. Let $P \stackrel{\text{def}}{=}$ **if** $\mathit{declassify}(h \geq k)$ **then** $(h := h - k; l := l + k)$ **else nil**. This program does satisfy delimited release [21]. Consider now abstract non-interference, the maximal information disclosed, for each public input is:

$$\Pi(id, id)_{|_k} = \{\left\{ h \,\middle|\, h \geq k \right\}, \left\{ h \,\middle|\, h < k \right\}, \mathbb{Z}, \varnothing\}$$

Therefore, if we consider the notion of abstract non-interference in Eq. 1, we have that the program is secure, but if we suppose that the attacker can control the public input, or can observe computations corresponding to different public inputs, then we have to compute the maximal amount

of information disclosed, that in this case is $\phi = id$. Namely, the program cannot be made secure, unless we declassify the value of private data. For these reasons, when we consider the program under an attack that can change the value of k during the computation, as happens in the following program [21], then it does no more satisfy delimited release.

$$P \stackrel{\text{def}}{=} \left[\begin{array}{l} l := 0; \\ \textbf{while } n \geq 0 \textbf{ do} \\ \quad k := 2^{n+1}; \\ \quad \textbf{if } \mathit{declassify}(h \geq k) \textbf{ then } (h := h - k; l := l + k) \textbf{ else nil}; \\ \quad n := n - 1; \end{array}\right.$$

4. Consider the program

$$P \stackrel{\text{def}}{=} \left[\begin{array}{l} h := h \ \mathit{mod}\ 2; \\ \textbf{if } \mathit{declassify}(h = 0) \textbf{ then } (l := 0; h := 0) \textbf{ else } (l := 1; h := 1); \end{array}\right.$$

This program does not satisfy delimited release [21]. The maximal amount of information disclosed in ANI is

$$\phi = \{\{\ h \,|\, h \ \mathit{mod}\ 2 = 0\ \}, \{\ h \,|\, h \ \mathit{mod}\ 2 = 1\ \}, \mathbb{Z}, \varnothing\} \equiv \mathit{Par}$$

The program is not secure, since the amount of information declassified is less than the maximal amount of information released, e.g., $\langle 2, l\rangle \approx_e \langle 3, l\rangle$ (for each l) while $\phi(2) \neq \phi(3)$, i.e, $\langle 2, l\rangle \not\approx_\phi \langle 3, l\rangle$. The secure program is [21]: **if** *declassify*$(h \ \mathit{mod}\ 2)$ **then** $(l := 0; h := 0)$ **else** $(l := 1; h := 1)$.

These examples show that delimited release captures a notion of non-interference which is weaker than declassified ANI, in the sense that it corresponds to a notion of non-interference (Eq. 1) that holds only in presence of passive attackers unable to collect results due to different public inputs.

5.3 Abstract Non-Interference vs Relaxed Non-interference

The notion of delimited release for λ-calculus is *relaxed non-interference* [17]. In this notion a λ-term t, cleaned from the security labels, is secure when it is equivalent to, i.e., can be rewritten as $f(n_1\sigma_1)\ldots(n_k\sigma_k)$, where f is a λ-term without any secret variable, σ_i are all secret variables and n_i are downgrading policies (λ-terms) for σ_i such that, for each i, $n_i\sigma_i$ is a public information on σ_i. The term $f(n_1\sigma_1)\ldots(n_k\sigma_k)$ satisfies non-interference since, when we fix the public input, included all the $n_i\sigma_i$, we cannot observe any difference in the public output, allowing the confidential information to be leaked in a controlled way, i.e., controlled by the policies n_i. Unfortunately, this definition has some limitations, in particular, even if the authors do not prove that this is the only

failure situation, they note that this notion fails in presence of, for example, an exponentially long time attack. However, this failure case is due to the same problems arisen for Eq. 1. The interesting aspect of this paper is that they can derive, from the program, the right downgrading policy that makes the program secure. We show in the following example that this corresponds precisely to the maximal amount of information disclosed for a fixed input [9]. Note that, even if at a glance relaxed non-interference could seem to be a declassification more via blocking than via allowing, this is not true since declassification via blocking is characterized by checking non-interference for the semantics *collecting* the results for all the private inputs that have the same property. Instead, declassification via allowing is characterized by checking non-interference on the concrete semantics but *only* for those private inputs with a fixed property. This is exactly what happens in relaxed non-interference since the inputs $n_i\sigma_i$ are considered public (even if the σ_i's are private), and therefore fixed, hence only the σ_i's that give the same result $n_i\sigma_i$ are considered.

Example 4. Consider the program P [17] with $sec, x, y : \mathtt{H}$, and $in, out : \mathtt{L}$:

$$P \stackrel{\text{def}}{=} \begin{bmatrix} x := \mathit{hash}(sec); y := x \ \mathit{mod}\ 2^{64}; \\ \textbf{if } y = in \textbf{ then } out := 1 \textbf{ else } out := 0; \end{bmatrix}$$

where *hash* is a function. The downgrading policy for sec in the corresponding λ-program, is $\lambda sec.\lambda in.(\mathit{hash}(sec)\ \mathit{mod}\ 2^{64}) = in$ [17]. Consider now the maximal amount of information disclosed for each public input value in:

$$\begin{aligned} \Pi(id, id)_{in} &= \left\{ \ \left\{ \ sec \,\middle|\, [\![P]\!](sec, in)^{out} = k \ \right\} \,\middle|\, k \in \{0, 1\} \ \right\} \\ &= \left\{ \begin{array}{l} \left\{ \ sec \,\middle|\, (\mathit{hash}(sec)\ \mathit{mod}\ 2^{64}) = in \ \right\}, \\ \left\{ \ sec \,\middle|\, (\mathit{hash}(sec)\ \mathit{mod}\ 2^{64}) \neq in \ \right\} \end{array} \right\} \end{aligned}$$

where $[\![P]\!](sec, in)^{out}$ is the value resulting in the variable out, when the inputs are sec and in. Note that the closure $\phi_{in} \stackrel{\text{def}}{=} \Pi(id, id)_{in} \cup \{\top, \bot\}$ corresponds, to the downgrading policy above. Moreover, in ANI we can derive the maximal amount of information disclosed, independently of the public input $\phi = \bigsqcap_{in \in \mathbb{V}^{\mathtt{L}}} \Pi(id, id)_{in}$, making the program also robust in presence of active attackers.

Note that also relaxed non-interference inherits the problems expressed for Eq. 1, since it depends on the particular fixed public input. This means that relaxed non-interference requires the attacker not to be able to observe computations due to different public inputs and to control the public input.

6 Discussion

In [24] the authors provide a road map for classifying all the different approaches to declassification, where declassification means weakening non-interference. They

distinguish among four categories, depending on *what* is released, *who* controls the information released, *where* the information is released and *when* the information is released. They introduce abstract non-interference as a particular case of the PER model of information, while in [14] it is proved that, vice versa, the PER model can be seen as a particular case of abstract non-interference. Moreover, abstract non-interference is only in the *what* classification [24], since only its *partial release* aspect is considered, but as we show in this paper, there is also a *who* function that can be treated in terms of abstract non-interference, which is made even more evident by the relation existing between abstract non-interference analysis and robust declassification.

In this paper, we show that our point of view is quite different, in particular, as regards the *who* and *what* distinction. Hence, for us the *who* class captures all the weakenings of non-interference where the notion is weakened by modeling the power of *who* is observing. For this reason, in this class we put the PER model [23], the security policy for robust declassification [26], abstract non-interference [9], and the complexity-based approaches [8, 16]. While the *what* class contains all the weakenings consisting in modeling *what* can or cannot flow, independently of who controls information release, and therefore it contains selective dependency [4], enforced robust declassification [19], abstract non-interference [9], delimited release [21], relaxed non-interference [17], and information theory-based approaches [3, 18]. Moreover, we explained the two different aspects of the *what* class (here called declassification), one corresponds to fixing what can flow, and the other to fixing what should not flow, and we show that both can be modeled in abstract non-interference, while delimited release and relaxed non-interference can only fix what is *allowed* to flow. Finally, since abstract non-interference is in both the classes, we can study the relation between these two different ways of weakening non-interference, and indeed in [11] the authors proved that the derivation of *what* flows and the derivation of *who* can observe, is formalized in terms of a Galois connection between the two constructors introduced in [9]. As future work, in the same spirit of [11], we would like to understand the relation existing between declassification via allowing and declassification via blocking, in order, for example to derive the information to block from the maximal amount of information disclosed, and vice versa.

In Table 3 we summarize the comparisons made in this paper. In particular we use the notation $\equiv$ for saying that two notions characterize the same end-to-end security policy, and *Id* corresponds to the identity relation/closure.

Acknowledgments

I would like to thank Anindya Banerjee and Roberto Giacobazzi for their insightful comments, and the anonymous referees for their helpful suggestions for improvement.

WHO	PER model $\equiv [Id]P(Id)$ $S \models \mathcal{SP}(\approx) \equiv \models_{\langle\!\langle P \rangle\!\rangle} [\rho_{\approx}]S(\rho_{\approx})$
WHAT (VIA ALLOWING)	Selective Dependency $\equiv (Id)P(\phi \Rightarrow Id)$ Delimited Release $\equiv \forall l \in \mathbb{V}^{\mathbf{L}}.\ (Id)P(\pi_l \Rightarrow Id)$ Relaxed Non-interference $\equiv \forall l \in \mathbb{V}^{\mathbf{L}}.\ (Id)P(\pi_l \Rightarrow Id)$
WHAT (VIA BLOCKING)	$(\eta)P(\phi \rightsquigarrow\!\!\!\!\!\!\rceil\rho)$

Table 3. Outline.

References

1. D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, MITRE Corp. Badford, MA, 1973.
2. S. Chong and A. C. Myers. Security policies for downgrading. In *ACM Conf. on Computer and Communications Security*, pages 198–209. ACM-Press, NY, October 2004.
3. D. Clark, S. Hunt, and P. Malacaria. Quantified interference: Information theory and information flow (extended abstract). In *Workshop on Issues in the Theory of Security, WITS*, 2004.
4. E. S. Cohen. Information transmission in sequential programs. *Foundations of Secure Computation*, pages 297–335, 1978.
5. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of Conf. Record of the 4th ACM Symp. on Principles of Programming Languages* (*POPL '77*), pages 238–252. ACM Press, New York, 1977.
6. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of Conf. Record of the 6th ACM Symp. on Principles of Programming Languages* (*POPL '79*), pages 269–282. ACM Press, New York, 1979.
7. D. E. Denning and P. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.
8. A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 1–17. IEEE Computer Society Press, 2002.
9. R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *Proc. of the 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '04)*, pages 186–197. ACM-Press, NY, 2004.
10. R. Giacobazzi and I. Mastroeni. Proving abstract non-interference. In *Annual Conference of the European Association for Computer Science Logic (CSL'04)*, volume 3210, pages 280–294. Springer-Verlag, 2004.
11. R. Giacobazzi and I. Mastroeni. Adjoining declassification and attack models by abstract interpretation. In *Proc. of the European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 295–310. Springer-Verlag, 2005.

12. R. Giacobazzi and I. Mastroeni. Timed abstract non-interference. In *Proc. of The International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05)*, Lecture Notes in Computer Science. Springer-Verlag, 2005. To appear.
13. J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symp. on Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.
14. S. Hunt and I. Mastroeni. The PER model of abstract non-interference. In *Proc. of The 12th Internat. Static Analysis Symp. (SAS'05)*, volume 3672 of *Lecture Notes in Computer Science*, pages 171–185. Springer-Verlag, 2005.
15. R. Joshi and K. R. M. Leino. A semantic approach to secure information flow. *Science of Computer Programming*, 37:113–138, 2000.
16. P. Laud. Semantics and program analysis of computationally secure information flow. In *In Programming Languages and Systems, 10th European Symp. On Programming, ESOP*, volume 2028 of *Lecture Notes in Computer Science*, pages 77–91. Springer-Verlag, 2001.
17. P. Li and S. Zdancewic. Downgrading policies and relaxed noninterference. In *Proc. of the 32st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '05)*, pages 158–170. ACM-Press, NY, 2005.
18. G. Lowe. Quantifying information flow. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 18–31. IEEE Computer Society Press, 2002.
19. A.C. Myers, A. Sabelfeld, and S. Zdancewic. Enforcing robust declassification. In *Proc. IEEE Symp. on Security and Privacy*. IEEE Computer Society Press, 2004.
20. F. Ranzato and F. Tapparo. Strong preservation as completeness in abstract interpretation. In D. Schmidt, editor, *Proc. of the 13th European Symposium on Programming (ESOP'04)*, volume 2986 of *Lecture Notes in Computer Science*, pages 18–32. Springer-Verlag, 2004.
21. A. Sabelfeld and A. C. Myers. A model for delimited information release. In *Proc. of the International Symp. on Software Security (ISSS'03)*, volume 3233 of *Lecture Notes in Computer Science*, pages 174–191. Springer-Verlag, October 2004.
22. A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE J. on selected ares in communications*, 21(1):5–19, 2003.
23. A. Sabelfeld and D. Sands. A PER model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation*, 14(1):59–91, 2001.
24. A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *Proc. of* 18^{th} *IEEE Computer Security Foundations Workshop (CSFW-18)*. IEEE Comp. Soc. Press, 2005.
25. D. Volpano. Safety versus secrecy. In *Proc. of the 6th Static Analysis Symp. (SAS'99)*, volume 1694 of *Lecture Notes in Computer Science*, pages 303–311. Springer-Verlag, 1999.
26. S. Zdancewic and A. C. Myers. Robust declassification. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 15–23. IEEE Computer Society Press, 2001.