Preserving Secrecy under Refinement [⋆]

Rajeev Alur, Pavol Černý, and Steve Zdancewic

University of Pennsylvania

Abstract. We propose a general framework of *secrecy* and *preservation of secrecy* for labeled transition systems. Our definition of secrecy is parameterized by the distinguishing power of the observer, the properties to be kept secret, and the executions of interest, and captures a multitude of definitions in the literature. We define a notion of *secrecy preserving refinement* between systems by strengthening the classical trace-based refinement so that the implementation leaks a secret only when the specification also leaks it. We show that secrecy is in general not definable in μ-calculus, and thus not expressible in specification logics supported by standard model-checkers. However, we develop a simulation-based proof technique for establishing secrecy preserving refinement. This result shows how existing refinement checkers can be used to show correctness of an implementation with respect to a specification.

1 Introduction

Security and confidentiality are growing concerns in software and system development [14]. The question of how to ascertain that an attacker cannot easily get information about classified data is central in this domain. We investigate the possibilities for using automated verification techniques (such as model checking) to answer this question, and in particular, we focus on the notion of *refinements* that preserve secrecy. Stepwise refinement is considered to be the correct approach to system and software construction, since it enables developers to find design errors in earlier stages of development. Refinements are useful for synthesizing implementations from higher level specifications, for instance via compilation or other code transformations. Such refinement based approach has been advocated by, for example, Hoare [6] and Lamport [8]. Our goal is to develop a formal and general framework for refinement that also takes into account secrecy.

Our contributions are two fold. First, we introduce a general framework for reasoning about secrecy requirements in a system. We use the standard verification framework – labeled transition systems. Our notion of secrecy depends on three parameters: (1) the equivalence relation on runs of the system that models the distinctions the observer can make, (2) the properties that are to be kept secret, and (3) the set of runs that are of interest. Intuitively, a property is secret if, for every run of interest, there is an equivalent run such that only one of these

[⋆] This research was partially supported by NSF Cybertrust award CNS 0524059.

two runs satisfies the property. We show that by varying these three parameters, it is possible to capture *possibilistic* definitions of secrecy found in the literature such as noninterference and perfect security property [13, 16]. We study whether such a general notion of secrecy can be specified using temporal logics. The answer is negative: we prove that secrecy is not expressible in μ-calculus. It has been claimed (see [3]) that it is possible to specify secrecy in temporal logic on self-composition (self-composition is a composition of a program with itself). However, we demonstrate that this too is not possible for the general definition of secrecy.

It is well-known that standard notions of refinement (e.g. trace inclusion) do not preserve secrets, and the refined program may leak more secrets than the original program [10]. Our second main contribution is that we define secrecy-preserving refinement and present a simulation-based technique for proving that one system is a refinement of another. In our definition, an implementation is a refinement of the specification, if for every run r of the implementation, there exists a run r' of the specification such that the observer cannot distinguish r from r', and for every property that the observer can deduce from r in the implementation can also be deduced by observing r' in the specification. Simulation is a standard technique: in order to show that a program P refines a program Q (in the classical sense), one can show that Q simulates P. This can also be part of the simulation based proof of secrecy-preserving refinement, since we require trace inclusion in the usual way. However, in order to show that P does not leak more secrets than Q, one must also show that P simulates Q. The reason is that using this simulation relation, one can prove that if P leaks a secret, then so does Q. This implies that even though secrecy is not specifiable in μ-calculus, and thus cannot be directly checked by existing model-checkers, showing that implementation preserves secrets of the specification can be done using existing tools (such as Mocha [2], CadenceSMV [11], PVS [12]) by establishing a simulation relation.

Related Work

We know of only two notions of secrecy preserving refinements that were defined previously. Mantel [9] assumes that some fixed, strong information-flow properties of the system are enforced and his definition of refinement preserves those properties. Our approach is more flexible because it permits the specification of arbitrary secrecy requirements. This means that if the specification program does not maintain secrecy of a certain property, the implementation program does not need to either. Jürjens [7] considers a different (and weaker) definitions of secrets. In his approach, a secret is leaked if the program (possibly when interacting with an adversary) outputs the secret. This approach thus ignores information-flow leaks, i.e. cases when the adversary can infer something about the secret without explicitly seeing it.

There is a large body of literature in language-based security (see [13] for an overview). Various definitions of secrecy have been considered, but all possibilistic variations (those that ignore probabilistic information about the distribution of system behaviors) can be captured in our framework. More closely related is the work on checking for secrecy using self-composition techniques—the work by Barthe et al. mentioned above and [15, 4], where the authors consider only deterministic programs. Halpern and O'Neill [5] define a notion of secrecy in the context of multiagent systems that is similar to our definitions, but they do not consider secrecy-preserving refinements. The preservation of secrecy has been studied in the context of programming language translation by Abadi [1] using techniques based on full abstraction.

2 Secrecy requirements

In this section, we introduce a framework in which we can reason about properties of a system being secret, i.e. not inferable by an observer who sees the behavior of the system. The framework we present is general enough to capture all possibilistic definitions of secrecy defined in both programming language and verification literature, to the best of our knowledge.

A *labeled transition system* (LTS) T is a tuple (Q, L, δ, I), where Q is a set of states, L is a set of labels, $\delta \subseteq Q \times L \times Q$ is a transition relation, and $I \subseteq Q$ a set of initial states.

A sequence $r = q_0 l_0 q_1 \ldots$ of alternating states and labels is a *run* of the labeled transition system T iff $q_0 \in I$ and $\forall i : 0 \leq i < |r| \Rightarrow (q_i, l_i, q_{i+1}) \in \delta$. Let $R(T)$ be the set of all runs of the LTS T.

A *property* α is a subset of the set of runs, i.e. $\alpha \subseteq R(T)$. A *state-property* is a property that depends only on the last state of a run. Formally, α is a state-property iff there is a set of states $Q_s \subseteq Q$ such that $r \in \alpha$ iff $r = q_0 l_0 q_1 l_1 \ldots l_{n-1} q_n$ and q_n is in Q_s.

Given this model of systems, we want to define what an observer can see and what he or she can infer based on those observations. The observer cannot see everything about the current run of the system, that is to say, in general, several runs can correspond to the same observation. We model this using an equivalence relation on runs, $\equiv \subseteq R(T) \times R(T)$. For a property α, the observer is able to conclude that α holds, if α holds for all the runs that correspond to his or her observations. He or she is able to conclude that α does not hold, if it does not hold for all the runs that correspond to the observations. The third possibility is that the observer is not able to conclude whether α holds or not. We will thus need to use a three-valued domain, $\{\top, \bot, m\}$ (true, false, maybe), and a partial order that models the knowledge the observer has. $\sqsubseteq$ is the following partial order on $\{\top, \bot, m\}$: $m \sqsubseteq m, m \sqsubseteq \top, m \sqsubseteq \bot, \bot \sqsubseteq \bot, \top \sqsubseteq \top$.

Function *IP – inferable properties*, is a function that, given a run r, a property α and an equivalence relation $\equiv$, represents the knowledge of the observer about the property α after the run r. $IP(r, \alpha, \equiv) = \top$ if $\forall r' : r' \equiv r \Rightarrow r' \in \alpha$, $IP(r, \alpha, \equiv) = \bot$ if $\forall r' : r' \equiv r \Rightarrow r' \notin \alpha$ and $IP(r, \alpha, \equiv) = m$ otherwise. Our notion of secrecy depends on one additional parameter: instead of requiring a property α to be secret in every run of the system, we may want to focus only on a subset β of runs that are of interest, e.g. the set of all terminating runs. This leads to the following formalization of secrecy:

> Let T be a labeled transition system and α and β be two properties. The property α is a *secret* in β for T w.r.t. $\equiv$ if for all $r \in \beta$, $IP(r, \alpha, \equiv) = m$.

We present the following examples in order to show that our definition is general enough to capture several standard information-flow properties such as noninterference or Perfect Security Property. We can capture these definitions by varying the parameters $\equiv$ and β.

Linear-time Secrecy

Consider an observer who can see the actions of the system, i.e. the labels in L. These labels, for example, might be the messages sent or received by the system. Assume that L contains a symbol τ, which models internal actions of the system.

We define the strong (time-sensitive) equivalence relation ($\approx$) as follows. Let Tr be an erasing homomorphism defined on runs: $Tr(q) = \epsilon$, $Tr(l) = l$, i.e. Tr erases all states. Two runs r and r' are strongly equivalent ($r \approx r'$) iff $Tr(r) = Tr(r')$. The equivalence class to which a run r belongs can be represented by $Tr(r)$, which corresponds to what the observer sees when r is the current execution of the system. $Tr(r)$ is a sequence of labels, and such sequences are called *traces*. $Tr(T)$ is the set of all traces of the LTS T.

We define the weak (time-insensitive) equivalence relation ($\approx_w$) as follows. Let Tr_w be an erasing homomorphism defined on runs: $Tr_w(q) = \epsilon$, $Tr_w(l) = l$ for $l \neq \tau$ and $Tr_w(\tau) = \epsilon$, i.e. Tr_w erases all states and all internal actions. Two runs r and r' are weakly equivalent ($r \approx_w r'$) iff $Tr_w(r) = Tr_w(r')$. The equivalence class can be represented by $Tr_w(r)$ and is called a *weak trace*. Let $Tr_w(T)$ be the set of all weak traces of the LTS T.

Consider the following two programs.

```
A: x=?; y=0; z=x; send z;
B: x=?; y=0; z=y; send z;
```

It is easy to see how they can be modeled as transition systems in our framework. The states are valuations of variables. The set L contains three labels s_0, s_1, τ. s_0 denotes the fact that 0 was sent, s_1 that 1 was sent and τ denotes all the internal (silent) actions. The input ($x = ?$) is intended not to be seen by an observer and thus is modeled by a silent action. We want to analyze what an observer might infer about whether or not $x = 0$ during the execution of the program if he or she can observe what the program sends. We model this using the strong observational equivalence $\approx$ and the state property $x = 0$. Suppose that the input is 0. Note that the observer sees the same trace for both programs, namely $t = \tau\tau\tau 0$. For the program A, the observer, after having seen the trace t was sent, can conclude that $x = 0$ holds. For the program B, the observer does not know whether $x = 0$ holds or not after having seen the trace t. We can conclude that for program A, the state property $x = 0$ is not a secret in the set of all runs w.r.t. $\approx$ and it is a secret for program B.

Noninterference

Consider the standard formulation of termination insensitive noninterference [13]. It is defined using low and high variables, where low variables are visible to the observer and high variables are not. Noninterference can be then formulated informally as follows: "if two input states share the same values of low variables then the behaviors of the program executed from these states are indistinguishable by the observer".

We define functional equivalence $\approx_f$ as follows. Let $\approx_f^i$ and $\approx_f^o$ be two equivalence relations on states. For all terminating runs r and r' we have $r \approx_f r'$ iff their initial states are related by $\approx_f^i$ and their final states by $\approx_f^o$. We model noninterference by functional equivalence defined above. Two states q and q' are related by $\approx_f^i$ (and $\approx_f^o$) exactly when the valuation of the low variables is the same in q and q'.

The purpose of using noninterference is to determine whether some property α of high variables is inferable by an observer who sees only low variables. We can capture this in our framework as follows. Let $\mathcal{P}$ be the set of all expressible properties of high variables. For example, if every property of high variables is considered to be expressible, $\mathcal{P}$ corresponds to the powerset of the set of valuations of high variables. Consider a classic requirement such as "a secret key should stay secret." In our framework, this can be expressed as "a secret key stays secret with respect to a set of predicates $\mathcal{P}$", i.e. none of the properties of the secret key that are in $\mathcal{P}$ will be revealed.

Let β_t be the set of all terminating runs. We can conclude that the system satisfies the noninterference property w.r.t. $\mathcal{P}$ iff for all $\alpha \in \mathcal{P}$, α is secret in β_t w.r.t. $\approx_f$.

Perfect Security Property

Let us consider the Perfect Security Property (PSP) [16]. It is an information-flow property defined in a trace-based setting. In order to define it, we divide the labels into low-security and high-security categories. The observer knows the specification of the system - i.e. the set of all possible traces (sequences of labels) and he or she can observe low-security labels. PSP ensures that the observer cannot deduce any information about occurrences of high-security events.

We can model the PSP in our framework by choosing an appropriate equivalence relation on runs and a property on runs. Let $\mathit{Low} \subseteq L$ be a set of low-security labels and let $\mathit{High} \subseteq L$ be a set of high-security labels such that Low and High partition L. We use the following equivalence relation. Let Tr_{psp} be an erasing homomorphism defined on runs as follows: $\mathit{Tr}_{psp}(q) = \epsilon$, $\mathit{Tr}_{psp}(l) = l$ for $l \in \mathit{Low}$ and $\mathit{Tr}_w(l) = \epsilon$ for $l \in \mathit{High}$, i.e. Tr_{psp} erases all states and all high-security actions. Two runs r and r' are *psp*-equivalent ($r \approx_{psp} r'$) iff $\mathit{Tr}_{psp}(r) = \mathit{Tr}_{psp}(r')$. For each label $h \in \mathit{High}$, we define the property α_h: a run r is in α_h if h occurs in r. Now we can conclude that PSP holds iff α_h is secret in β_{all} w.r.t. $\approx_{psp}$ for all $h \in \mathit{High}$.

Specifying Secrecy in Temporal Logics

It is well-known that secrecy cannot be expressed as a predicate on a single trace and hence cannot be specified in linear-time specification languages such as linear temporal logic (see, for example, [10], for a proof). We prove that secrecy is not a branching-time property either.

Let us consider finite trees over alphabet Σ. The vertices are labeled by elements of Σ (edges are not labeled). A tree T can be seen as an LTS T', where states correspond to vertices of the tree, edges are the parent-child edges, and all the edges are labeled by the same symbol. For each label $\alpha \in \Sigma$, let α' be a state-property corresponding to the set of all vertices labeled by α.

Theorem 1. *The set S of trees T over $\{\alpha, \beta\}$ such that α' is secret in β' w.r.t. $\approx$ for T' is not a regular tree-language.*

Proof. For a proof by contradiction, suppose that S is regular. Then the following special case, defined by a regular condition that β' is false only for the root of the tree, would also be regular. The fact that α' is secret in β' w.r.t. $\approx$ corresponds to the fact that at each depth d ($d > 0$) of the tree, there is a node in α' and a node not in α'. It is well-known that this is not a regular property. $\square$

Corollary 1. *The set of trees T over $\{\alpha, \beta\}$ such that α' is secret in β' w.r.t. $\approx$ for T' is not definable in μ-calculus.*

Note that it is possible to devise algorithms based on standard model-checking for special cases of our definition of secrecy. For example, Barthe et al. [3] claim that it is possible to use CTL model-checking to check for noninterference in finite-state systems. However, upon examination, this holds only for a specific definition of noninterference, the one based on functional equivalence relation (as opposed to, e.g., strong equivalence relation). Barthe et al. reduce checking for noninterference to model-checking a CTL formula on self-composition. Self-composition can be viewed as a (sequential or parallel) composition of a program with itself (variables are renamed in the other copy of the program). It can be shown, by a proof similar to the one above, that there is no μ-calculus formula that characterizes the general definition of noninterference on self-composition.

3 Secrecy-preserving Refinements

Let us suppose that we have two labeled transition systems T_{spec} and T_{imp}. We want to establish that T_{imp} does not leak more secrets than T_{spec}.

First, consider the classical notion of refinement, where T_{imp} refines T_{spec} iff all behaviors of T_{imp} are allowed by T_{spec}. This notion of refinement preserves all properties expressible in linear temporal logic, but does not in general preserve the secrecy of properties. Consider two of the systems in Figure 1, (a) as T_{spec} and (b) as T_{imp}. Using the classical notion, T_{imp} is a refinement of T_{spec}, since the behaviors of T_{imp} are included in behaviors of T_{spec}. This holds for both the functional (input-output) and observational (trace-based) view of behaviors.

However, T_{imp} leaks more secrets than T_{spec} does. If the observer of T_{imp} sees a trace s_0, he or she can conclude that α does not hold. On the other hand, for T_{spec}, the observer cannot determine whether α holds or not.

We proceed to introduce a new notion of refinement, one that preserves secrecy of properties. Intuitively, we want to show that for each run r of T_{imp}, there is an equivalent run r' of T_{spec}, such that the observer can deduce less about the properties of interest when observing T_{imp} executing r than when observing T_{spec} executing r'. Hence, let us extend the equivalence relation $\equiv$ to the runs of the two systems, i.e. $\equiv \subseteq (R(T_{spec}) \cup R(T_{imp})) \times (R(T_{spec}) \cup R(T_{imp}))$. Furthermore, we need to relate properties of interest for the two systems. Analogously, a property α now will be a subset of $R(T_{spec}) \cup R(T_{imp})$.

Now we are ready to state when a refined transition system preserves at least as many secrets as the original one:

Secrecy-preserving refinement:
Let T_{spec}, T_{imp} be two labeled transition system, let $\mathcal{P}$ be a set of properties and let $\equiv$ be an equivalence relation on $R(T_{spec}) \cup R(T_{imp})$. T_{imp} *$\mathcal{P}$-refines* T_{spec} w.r.t. $\equiv$ iff for all runs $r \in R(T_{imp})$, there exists a run $r' \in R(T_{spec})$ such that $r \equiv r'$ and for all properties $\alpha \in \mathcal{P}$, $IP(r, \alpha, \equiv) \sqsubseteq IP(r', \alpha, \equiv)$.

We present the following observations and an example to illustrate the definition. First, note that secrecy-preserving refinement extends the classical notion: consider the case when the set of properties $\mathcal{P}$ is empty. For strong (weak) equivalence $\mathcal{P}$-refinement corresponds to (weak) trace inclusion. For functional equivalence, $\mathcal{P}$-refinement corresponds to the requirement that the input-output relation of T_{imp} is included in the input-output relation of T_{spec}.

Consider the programs A and B from Section 2 again. As before, suppose that the observer does not see the input, but this time, we fix the input to be 0 in order to simplify the example. We consider the strong observational equivalence and we are interested in the state-property α that is true iff $x = 0$. There is only one run in each of the programs. Those runs are equivalent, since the trace is simply $\tau\tau\tau s_0$ in both cases. Let r_A denote the run of A and let r_B denote the run of B. As we have seen, $IP(r_A, \alpha, \approx) = \top$ and $IP(r_B, \alpha, \approx) = m$. Thus we can conclude that A does not $\mathcal{P}$-refine B w.r.t. $\approx$, but B $\mathcal{P}$-refines A w.r.t. $\approx$.

The following theorem states that the $\mathcal{P}$-refinement preserves the secrets from $\mathcal{P}$, i.e. that if T_{spec} does not leak a secret $\alpha \in \mathcal{P}$ and T_{imp} is a $\mathcal{P}$-refinement of T_{spec}, then also T_{imp} does not leak the secret α. Before stating the theorem, we need to define one more condition on the set of runs that are of interest, β. A property β is $\equiv$-preserving iff for all r and for all r', if $r \in \beta$ and $r \equiv r'$, then $r' \in \beta$.

Theorem 2. *Let T_{spec} and T_{imp} be two transition systems such that T_{imp} $\mathcal{P}$-refines T_{spec} w.r.t. $\equiv$ and let β be a an $\equiv$-preserving property. If $\alpha \in \mathcal{P}$ is a secret in β for T_{spec} w.r.t. $\equiv$, then α is a secret in β for T_{imp} w.r.t. $\equiv$.*

4 Proving Secrecy Using a Simulation Relation

In this section we restrict our attention to the strong (time-sensitive) and weak (time-insensitive) equivalence relations on runs and we consider only state-properties. With appropriate modifications, simulation-based proof techniques can be developed for other equivalences such as the ones used for noninterference and perfect security.

Let $T_{spec} = (Q_{spec}, L, \delta_{spec}, I_{spec})$ and $T_{imp} = (Q_{imp}, L, T_{imp}, I_{imp})$ be labeled transition systems. As above, let $\mathcal{P}$ denote both a set of properties about T_{spec} and a corresponding set of properties about T_{imp}. Note that the two transition systems have the same set of labels, thus the relation $\approx$ (the strong observational equivalence) can be seen as a relation on $R(T_{spec}) \cup R(T_{imp})$.

A binary relation $\lesssim \subseteq Q_{spec} \times Q_{imp}$ is a simulation relation iff for all $q_1 \lesssim q'_1$ for all state properties $\alpha \in \mathcal{P}$, $q_1 \in \alpha$ iff $q'_1 \in \alpha$ and for every $q_2 \in Q_{spec}$ and $l \in L$ such that $q_1 \xrightarrow{l} q_2$ there exists $q'_2 \in Q_{imp}$ such that $q'_1 \xrightarrow{l} q'_2$ and $q_2 \lesssim q'_2$.

We say that T_{imp} simulates T_{spec} ($T_{spec} \lesssim T_{imp}$), if there exists a simulation relation $\lesssim$ such that for every $q_1 \in I_{spec}$ there exists $q'_1 \in I_{imp}$ such that $q_1 \lesssim q'_1$.

A binary relation $\lesssim_w \subseteq Q_{spec} \times Q_{imp}$ is a weak simulation relation iff for all $q_1 \lesssim_w q'_1$, for all state properties $\alpha \in \mathcal{P}$, $q_1 \in \alpha$ iff $q'_1 \in \alpha$ and we have:

- $q'_1 \xrightarrow{\tau} q'_2$ implies that there exists a q_2 such that $q_1 \xrightarrow{\tau}{}^{*} q_2$ and $q_2 \lesssim_w q'_2$
- $q'_1 \xrightarrow{l} q'_2$ implies that there exists a q_2 such that $q_1 \xrightarrow{\tau}{}^{*} \xrightarrow{l} \xrightarrow{\tau}{}^{*} q_2$ and $q_2 \lesssim_w q'_2$.

Weak simulation between transition system is defined similarly to simulation between transition systems.

Let us consider the case of strong (time-sensitive) equivalence relation on runs. Firstly, we note that it follows from the definition of $\mathcal{P}$-refinement that the standard refinement condition ($Tr(T_{imp}) \subseteq Tr(T_{spec})$) is a necessary condition for the $\mathcal{P}$-refinement.

Secondly, note that unlike classical refinement, the condition that T_{spec} simulates T_{imp} is not sufficient for $\mathcal{P}$-refinement. To see this consider again two of the systems in Figure 1, (a) as T_{spec} and (b) as T_{imp}. Note that T_{spec} simulates T_{imp}, but T_{imp} leaks information on α on the trace s_0, whereas T_{spec} does not.

The property we are looking for is in fact the simulation in the other direction, i.e. that T_{imp} simulates T_{spec}. The reason is that using this simulation relation one can prove that if T_{imp} leaks a secret, then so does T_{spec}. Note also the condition that T_{imp} simulates T_{spec} is not a sufficient condition. Consider now the system on Figure 1(a) as T_{spec} and the system on Figure 1(c) as T_{imp}. Now T_{imp} refines T_{spec}, but for the trace s_1, T_{imp} leaks more secrets than T_{spec}.

The combination of the two conditions, $Tr(T_{imp}) \subseteq Tr(T_{spec})$ and T_{imp} simulates T_{spec} is sufficient to guarantee that $\mathcal{P}$-refinement holds.

Theorem 3. *If $Tr(T_{imp}) \subseteq Tr(T_{spec})$ and $T_{spec} \lesssim T_{imp}$, then T_{imp} $\mathcal{P}$-refines T_{spec} w.r.t. $\approx$.*

Proof. Let r be a run in $R(T_{imp})$. We have to prove that there exists a run r' in T_{spec} such that $r \approx r'$ and $IP(r, \alpha, \approx) \sqsubseteq IP(r', \alpha, \approx)$. We have that $Tr(T_{imp}) \subseteq$

$Tr(T_{spec})$, therefore there exists a run r' in T_{spec} such that $r \approx r'$. It remains to prove that $IP(r, \alpha, \approx) \sqsubseteq IP(r', \alpha, \approx)$. Let us suppose that $IP(r, \alpha, \approx) = \top$. We have to show that $IP(r', \alpha, \approx) = \top$. Let us suppose that $IP(r', \alpha, \approx) = \bot$ or $IP(r', \alpha, \approx) = m$. In any of the two cases, we know that there exists $r'' \in R(T_{spec})$ such that $r' \approx r''$ and the last state of r'' is not in α. Using the condition $T_{spec} \lesssim T_{imp}$ we prove (by induction on the length of the $Tr(r'')$) that there exists an $r''' \in R(T_{imp})$ such that $r'' \approx r'''$ and the last state of r''' is not in α. By transitivity of $\approx$, we have that $r''' \approx r$. This is a contradiction with the assumption $IP(r, \alpha, \approx) = \top$. Thus we can conclude that if $IP(r, \alpha, \approx) = \top$, then $IP(r', \alpha, \approx) = \top$. The case of $IP(r, \alpha, \approx) = \bot$ is similar. If $IP(r, \alpha, \approx) = m$, then there is nothing to prove, since $m \sqsubseteq IP(r', \alpha, \approx)$. □

For weak equivalence relation on runs, a similar theorem holds.

Theorem 4. *If $Tr_w(T_{imp}) \subseteq Tr_w(T_{spec})$ and $T_{spec} \lesssim_w T_{imp}$, then T_{imp} $\mathcal{P}$-refines T_{spec} w.r.t $\approx_w$.*

Note that Theorem 3 implies that secrecy is preserved by bisimulation, since for bisimilar systems, both conditions are met. Note also that we have shown in Section 2 that secrecy is not a branching time property.

The conjunction of the two conditions of Theorem 3 is not a necessary condition for $\mathcal{P}$ refinement. Consider the two systems in Figure 2 and suppose the set $\mathcal{P}$ of properties is the singleton $\{\alpha\}$. Note that $Tr(T_{imp}) \subseteq Tr(T_{spec})$ and T_{imp} does not simulate T_{spec}. However, T_{imp} does not leak any more secrets T_{spec}.

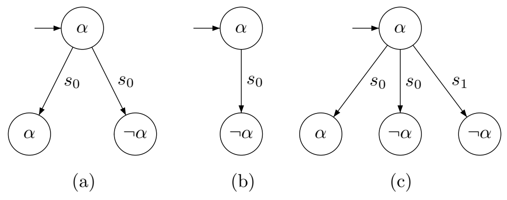


Fig. 1. Refinement by simulation

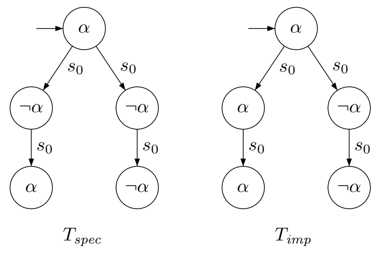


Fig. 2. Simulation is not a necessary condition.

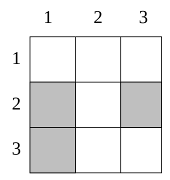
5 Example

We present an example in order to illustrate the definition of secrecy-preserving refinement and to demonstrate the simulation-based proof method defined above. We will present two implementations, T_{spec} and T_{imp}, of a protocol (more precisely, of one round of a protocol). We will show that while functionally they are equivalent (their input-output relation is the same) and T_{imp} refines T_{spec} in the classical sense (trace inclusion), the implementation T_{imp} leaks some properties that should be secret, whereas T_{spec} does not.

Consider the game Battleship. We will analyze an implementation of one of the players in one round of the protocol, so the following description is sufficient for our purposes[1]. The input (for each round) consists of a grid, where each square is either marked (meaning a ship is there) or unmarked, and of two integers i and j. The output should be *yes* if the square with coordinates i and j is marked.

Let us consider two implementations of this protocol, T_{spec} and T_{imp}. T_{spec} uses the straightforward array representation, T_{imp} uses a list representation. In T_{imp}, the board is represented by a list of rows and each row contains a list of the marked cells. (A possible motivation for the (re-)implementation T_{imp} is that it might be more efficient in case of sparse boards.)

Board:



T_{spec}:

```
array board;
int i,j;

rcv i; rcv j;
r = A[i,j];
send r;
```

T_{imp}:

```
list board, row;
int i,j;

rcv i;rcv j;
row=Board.getRow(i)
if (row.IsEmpty())
  r=0; send r;
 else
  r=row.getElem(j); send r;
```

Fig. 3. The Battleship game

We briefly explain how can the programs such as T_{spec} and T_{imp} be modeled in our framework. We use the standard operational semantics approach. The states are valuations of all variables (such as board, row and program counter pc). The label s_0 denotes the fact that 0 was sent, s_1 that 1 was sent and τ denotes all the internal (silent) actions. We model the fact that the board can in general be in any state at the beginning of a round of the battleship protocol by having multiple initial states. For the purposes of this example, we also model *receives* in this way. Thus any valuation of the variables where program counter is equal to 0 is an initial state. An assignment is modeled as an internal action τ. We model the methods (such as getRow()) by one internal action (thus a statement that contains an assignment and a method call is modeled by two internal actions). As an example consider the case shown in Figure 3 and the inputs $i = 1$ (column numbered 1) and $j = 2$ (row numbered 2). The initial state is now determined. The trace produced by T_{spec} is τs_1. The trace produced by T_{imp} is $\tau\tau\tau\tau\tau s_1$. For each cell with coordinates (i, j) of the board, we define

[1] For a full description of the game, google "battleship".

a property α_{ij} that is true iff the cell is marked. Let $\mathcal{P}$ be the set of these properties.

We will now show that T_{spec} and T_{imp} are equivalent w.r.t. $\approx_w$, that is, T_{spec} $\mathcal{P}$-refines T_{imp} and T_{imp} $\mathcal{P}$-refines T_{spec} w.r.t. $\approx_w$ and that it can be proven by simulation. We will prove also that T_{imp} does not $\mathcal{P}$-refine T_{spec} w.r.t. $\approx$.

Let us start by proving that for weak refinement, T_{spec} and T_{imp} are equivalent. We show that $Tr(T_{imp}) \subseteq Tr(T_{spec})$ and that $T_{spec} \lesssim_w T_{imp}$. To see that $Tr(T_{imp}) \subseteq Tr(T_{spec})$, note that both T_{spec} and T_{imp} have the same set of weak traces, namely $\{0, 1\}$ (since all the other actions are internal). Now we will show that T_{imp} simulates T_{spec}. We will present a function f from the states of T_{spec} to the states of T_{imp} and show that it defines a simulation relation. Recall that the states of T_{spec} and T_{imp} characterize the current board position, and contain valuations of variables for i, j and the program counter *pc*. In order to be able to define f, we divide the states of T_{spec} into Q^1_{spec} and Q^2_{spec}, where Q^1_{spec} contains those states where the value of *pc* (program counter) indicates that the send instruction has not been executed yet and Q^2_{spec} all the other states. We divide the states of T_{imp} analogously. The function f will relate each state q of T_{spec} to a state q' of T_{imp} that has the same board position, the same valuations of i and j, and $q \in Q^i_{spec}$ iff $q' \in Q^i_{imp}$. It is easy to check that f defines a weak simulation such that T_{spec} simulates T_{imp}. By Theorem 4, we can conclude that T_{imp} is a $\mathcal{P}$-refinement of T_{spec} w.r.t. $\approx$. Note also that $Tr(T_{imp}) = Tr(T_{spec})$ and the simulation we just defined is a bisimulation. Thus we can similarly conclude that T_{spec} is a $\mathcal{P}$-refinement of T_{imp} w.r.t. $\approx$.

We also show that T_{imp} does not $\mathcal{P}$-refine T_{spec} w.r.t. $\approx$ according to our definition, because T_{imp} leaks more secrets in certain situations. Again, consider the case depicted in Figure 3, but this time we fix the inputs to be $i = 1$ (column numbered 1) and $j = 1$ (row numbered 1). We assume the observer knows these inputs (but note that he or she does not see the board.) We analyze what he or she can infer from the execution of T_{spec} (and T_{imp}) on these inputs. As noted above, once the initial state is fixed (by the input values) there is only one possible run of T_{imp} (we denote it by r). The corresponding trace is $t_1 = \tau\tau\tau\tau s_0$. However, after the observer observes the trace t_1, he or she can infer that the j-th row is empty. For example, he or she knows that $IP(r, \alpha_{21}, \approx) = \bot$. This can be inferred because the number of internal actions is 4 (whereas if the j-th row was not empty, 5 internal actions would be observed before the final s_0). The execution of T_{spec} is similar in that there is only one possible run r given the inputs. The corresponding trace is τs_0. Given the program T_{spec}, it is clear that it is not possible to infer information about a property $\alpha_{ij}, i \neq 1, j \neq 1$, i.e. $IP(r', \alpha_{i'j'}, \approx) = m$ for $i \neq 1$ and $j \neq 1$. In particular, $IP(r', \alpha_{21}, \approx) = m$. We can thus conclude that T_{imp} is not a $\mathcal{P}$-refinement of T_{spec} w.r.t. $\approx$.

6 Conclusion

This paper presents a general framework for formal reasoning about secrecy properties. The framework is based on labeled transition systems and is thus suitable for presentation of algorithms for automated verification of secrecy. We presented how different definitions of secrecy can be captured in our framework. We showed also that secrecy is not definable by a μ-calculus formula. The main focus of this work was on defining a notion of refinement that preserves secrecy of properties and providing a method for proving that such a refinement holds. This method is based on simulation and thus can be used for automatic verification using existing tools.

There are several directions for future research. One possibility is to extend this work with static analysis for secrecy-preserving refinements of programs. Second, it would be useful to define program transformations to help designers to transform designs in a way that guarantees the preservation of secrecy. Third, we plan to investigate a logic for secrecy of properties. Fourth, it would be interesting to apply the framework presented here to resource-driven protocol transformation for embedded systems, such as Java cards or smart cards.

References

1. M. Abadi. Protection in programming-language translations. In *Proc. of ICALP'98*, pages 868–883, 1998.
2. R. Alur, T.A. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proc. of CAV'98*, pages 521–525, 1998.
3. G. Barthe, P. D'Argenio, and T. Rezk. Secure Information Flow by Self-Composition. In *Proc. of CSFW'04*, pages 100–114, 2004.
4. Á. Darvas, R. Hähnle, and D. Sands. A Theorem Proving Approach to Analysis of Secure Information Flow. In *Proc. of SPC'05*, pages 193–208, 2005.
5. J. Halpern and K. O'Neill. Secrecy in multiagent systems. In *Proc. of CSFW'02*, pages 32–46, 2002.
6. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
7. J. Jürjens. Secrecy-preserving refinement. In *Proc. of FME'01*, pages 135–152, 2001.
8. L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.
9. H. Mantel. Preserving information flow properties under refinement. In *Proc. of SP'01*, pages 78–91, 2001.
10. J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. of SP'94*, pages 79–93, 1994.
11. K.L. McMillan. A compositional rule for hardware design refinement. In *Proc. of CAV'97*, pages 24–35, 1997.
12. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In *CADE'92*, pages 748–752, 1992.
13. A. Sabelfeld and A. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.
14. F.B. Schneider, editor. *Trust in Cyberspace.* National Academy Press, 1999.
15. T. Terauchi and A. Aiken. Secure Information Flow as a Safety Problem. In *Proc. of SAS'05*, pages 352–367, 2005.
16. A. Zakinthinos and E. S. Lee. A general theory of security properties. In *Proc. of SP'97*, pages 94–102, 1997.