Confidentiality for Multithreaded Programs via Bisimulation*

Andrei Sabelfeld

Department of Computer Science, Cornell University, Ithaca, NY 14853, USA
Fax: +1 607 255 4428 E-mail: andrei@cs.cornell.edu

Abstract. Bisimulation has been a popular foundation for characterizing the confidentiality properties of concurrent programs. However, because a variety of bisimulation definitions are available in the literature, it is often difficult to pin down the "right" definition for modeling a particular attacker. Focusing on timing- and probability-sensitive confidentiality for shared-memory multithreaded programs, we clarify the relation between different kinds of bisimulation by proving inclusion results. As a consequence, we derive the relationship between *scheduler-specific*, *scheduler-independent*, and *strong* confidentiality definitions. A key result justifying strong confidentiality is that it is the most accurate (largest) compositional indistinguishability-based confidentiality property that implies scheduler-independent confidentiality.

1 Introduction

Modern computing systems are increasingly vulnerable to application-level attacks. These attacks are particularly dangerous because they circumvent the standard low-level protection mechanisms (such as OS-based monitors and access control). Furthermore, application-level attacks are easier to create (or simply download and launch) exactly because of their high-level nature. Because standard security low-level enforcement mechanisms offer only limited protection against application-level attacks, there is high demand for models of *language-based security* [17, 31, 28] aimed at defending against threats at the programming-language and, hence, application level.

In this paper, we concentrate on *confidentiality* for multithreaded computation (we assume that security is synonymous with confidentiality in the rest of the paper). The objective is to prevent shared-memory multithreaded applications (programs) from leaking confidential information to the attacker. Such an information leak (or *information flow*) may take place when executing a (potentially untrusted) program on a trusted machine with legitimate access

* This research was supported by the Department of the Navy, Office of Naval Research, ONR Grant N00014-01-1-0968. Any opinions, findings, conclusions, or recommendations contained in this material are those of the author and do not necessarily reflect the views of the Office of Naval Research.

In *Proceedings of Andrei Ershov 5th International Conference on Perspectives of System Informatics*, Akademgorodok, Novosibirsk, Russia, July 9–12, 2003. LNCS.

to confidential data. Because modern computing systems incorporate much untrusted code (and/or unintended bugs), undesired leaks pose a serious threat to security-critical applications. This threat is of major concern for business, medical, and military computing systems that access and manipulate highly sensitive information.

Noninterference [12] is widely used to ensure that programs do not introduce undesired information flow. Noninterference requires that the publicly-observable behavior of a program does not depend on the program's secret inputs. The absence of dependency means that the publicly-observable behavior of the program does not vary (or is indistinguishable) as secret inputs are varied. Hence, central to the noninterference approach is the notion of *indistinguishability* of program behaviors by the attacker. *Bisimulation* has been a popular foundation for process indistinguishability and, thus, for characterizing the confidentiality properties of concurrent programs (e.g., [8, 9, 29, 34, 33, 4, 2, 27, 11]). Different kinds of bisimulation allow for modeling different kinds of the attacker's observational power. For example, the attacker's power in *timing attacks* [16, 7] includes the ability to observe the timing behavior of the system. This can be captured by timing-sensitive attacker models. The attacker's power in *probabilistic attacks* [13, 22] includes the ability to iteratively run computation and collect the stochastic properties of public data. This can be captured by probability-sensitive attacker models.

Timing- and probability-sensitive bisimulation [36, 5, 18] provides a suitable abstraction to characterize these attackers. However, because a variety of bisimulation definitions are available in the literature, it is often difficult to pin down the "right" definition for modeling a particular attacker. In this paper, building on earlier work on timing- and probability-sensitive confidentiality [29], we clarify the relation between different kinds of bisimulation by proving inclusion results. As a consequence, we derive the relationship between *scheduler-specific*, *scheduler-independent*, and *strong* security definitions. A key result justifying strong security is that it is the most accurate (largest) compositional indistinguishability-based security property that implies scheduler-independent security.

In the rest of the paper, we present a simple multithreaded language with dynamic thread creation (Section 2); give the definitions of scheduler-specific, scheduler-independent, and strong security (Section 3); argue that strong security is the largest compositional indistinguishability-based security property that implies scheduler-independent security, illustrate the security hierarchy by examples (Section 4); and, finally, conclude by a discussion on related and future work (Section 5).

2 A Language with Dynamic Thread Creation

In this section, we sketch a simple multithreaded language with dynamic thread creation. Our confidentiality conditions will be defined in terms of the semantics of this language. For brevity, we only give an informal description of the syntax and semantics.[1]

We consider an imperative language with skip, assignment, sequential composition, conditional, while, and fork operators. The semantics of the language are defined by two kinds of transitions: deterministic ($\twoheadrightarrow$) and probabilistic ($\rightarrow_p$ for some p where $0 < p \leq 1$ or simply $\rightarrow$ when the exact value of p is unimportant). Semantic rules are defined as transitions between configurations. A configuration $\langle\!| C, s |\!\rangle$ consists of a command (or a command thread pool represented by a vector of commands) C and a state s (memory). For example, the transition defining the semantics for the fork operator

$$\langle\!| \mathsf{fork}(C, \vec{D}), s |\!\rangle \twoheadrightarrow \langle\!| C\vec{D}, s |\!\rangle$$

states that $\mathsf{fork}(C, \vec{D})$ spawns a collection (vector) of threads $\vec{D}$ in parallel to the main thread C without changing the program state s.

Probabilistic transitions are defined through deterministic transitions (where $\{\!| \cdot |\!\}$ denotes a multiset):

$$(\text{Pick})\frac{\langle\!| C_i, s |\!\rangle \twoheadrightarrow \langle\!| \vec{C}, s' |\!\rangle}{\langle\!| \langle C_0 \dots C_{n-1} \rangle, s |\!\rangle \rightarrow^i_{\sigma(i,n)} \langle\!| \langle C_0 \dots C_{i-1}\vec{C}C_{i+1} \dots C_{n-1} \rangle, s' |\!\rangle}$$

$$(\text{Sum})\frac{p = \sum \{\!| q | \langle\!| \vec{C}, s |\!\rangle \rightarrow^i_q \langle\!| \vec{D}, s' |\!\rangle |\!\} \quad p \neq 0}{\langle\!| \vec{C}, s |\!\rangle \rightarrow_p \langle\!| \vec{D}, s' |\!\rangle}$$

A probabilistic transition originates from a configuration if a thread from the configuration's thread pool can make a deterministic step under the configuration's memory. The probability $\sigma(i, n)$ of the transition may depend on both the number i of the picked thread and the size n of the thread pool. Because no information about previous thread choices is used for determining the probability of the transition, this corresponds to a *history-independent* scheduler.[2] We assume that $\forall n(n > 0), i(i \in \{0, \dots, n-1\}).\, \sigma(i, n) > 0$ and $\forall n(n > 0).\ \sum_{i \in \{0,\dots,n-1\}} \sigma(i, n) = 1$. The labeling in the rule Pick together with the rule Sum ensure that $\forall \vec{C}, s.\ \sum_{D,s'} \{\!| p \mid \langle\!| \vec{C}, s |\!\rangle \rightarrow_p \langle\!| \vec{D}, s' |\!\rangle |\!\} = 1$. For example, instantiating σ as $uni(i, n) = \frac{1}{n}$ corresponds to the *uniform* scheduler that picks threads with uniform probability.

For simplicity, we assume that variables are partitioned into two classes—of *high* and *low* confidentiality (with typical variables h and l of respective confidentiality levels). We define *low equality* $=_L$ on memories by $s_1 =_L s_2$ if and only if s_1 and s_2 agree on the low variables.

[1] The complete syntax as well as deterministic and probabilistic semantics are given in Figures 2, 3, and 4, respectively, in Appendix A.

[2] This simple scheduler model is sufficient for our purposes. It is an instance of a general model of history-dependent schedulers [29].

3 Bisimulation-Based Confidentiality

As foreshadowed in Section 1, we can conveniently apply the machinery of bisimulation to characterizing security. The intuition is that a program is secure when a variation of the values of the high variables does not influence the low-observable behavior of the program. We have already defined the low equality of memories to represent the variation of the values of the high variables. What helps us model whether the low-observable behavior of the program is unchanged is *low-bisimulation.* This security-sensitive bisimulation captures whether two program behaviors are different through the view of an attacker. In the rest of this section, we present (with some simplifications) the bisimulation-based definitions of scheduler-specific, scheduler-independent, and strong security [29].

3.1 Scheduler-Specific Security

The worst-case-scenario attacker is capable of observing the timing and probabilistic properties of program execution. Hence, our goal is to capture probabilistic- and timing-sensitive security properties. Therefore, a suitable low-bisimulation definition is a timing-sensitive *probabilistic low-bisimulation.* Suppose we have a fixed scheduler σ for our semantics. Under this scheduler, let us state the definition of *σ-probabilistic low-bisimulation.*

Definition 1 *A partial equivalence relation R is a* σ-probabilistic low-bisimulation *on programs if whenever $\vec{C}\ R\ \vec{D}$ then*

$$
\begin{aligned}
&\forall s_1 =_L s_2. \langle\!| \vec{C}, s_1 |\!\rangle \to \langle\!| \vec{C'}, s'_1 |\!\rangle \Longrightarrow \exists \vec{D'}, s'_2.\ \langle\!| \vec{D}, s_2 |\!\rangle \to \langle\!| \vec{D'}, s'_2 |\!\rangle \textit{ such that}\\
&\quad (i) \sum \{\!| p \,|\, \langle\!| \vec{C}, s_1 |\!\rangle \to_p \langle\!| \vec{S}, s |\!\rangle, \vec{S} \in [\vec{C'}]_R, s =_L s'_1 |\!\} =\\
&\quad\quad \sum \{\!| p \,|\, \langle\!| \vec{D}, s_2 |\!\rangle \to_p \langle\!| \vec{S}, s |\!\rangle, \vec{S} \in [\vec{D'}]_R, s =_L s'_2 |\!\}\\
&\quad (ii)\ \vec{C'}\ R\ \vec{D'}, s'_1 =_L s'_2
\end{aligned}
$$

where $[\vec{E}]_R$ stands for the R-equivalence class which contains $\vec{E}$.

(Observe that the probabilities p are determined by the scheduler σ according to the semantic rules Pick and Sum.) The intuition is that under arbitrary variations of the high-confidentiality data of a state two related commands must execute in lock-step, affect the low memory in the same way, and the summed probability of stepping to the configurations of commands and memories from the same equivalence class must be the same. Note that low-bisimulation is defined directly on programs (with the variation of high data built in). Hence, the low-bisimulation relation is not necessarily reflexive (the intention is that reflexivity breaks on insecure programs), and, thus, is a partial equivalence relation. Two commands C and D are *low-bisimilar* ($C \sim_L^\sigma D$) *under σ-probabilistic low-bisimulation* if there exists a σ-probabilistic low-bisimulation that relates the commands. One can show that $\sim_L^\sigma$ itself is a σ-probabilistic low-bisimulation. The definition of σ-security is now straightforward:

$$\vec{C} \text{ is } \sigma\text{-}secure \iff \vec{C} \sim_L^\sigma \vec{C}$$

Examples of *uni*-secure programs are $h := l + 3; l := l + 1$ (h is not accessed), $\mathsf{if}\ l > 0\ \mathsf{then}\ h := h + 1\ \mathsf{else}\ h := 0$ (l is not assigned), $\mathsf{fork}(l := l+1, h := h-1)$ (independent threads), and $\mathsf{if}\ h > 0\ \mathsf{then}\ \mathsf{fork}(l := 0, l := 1)\ \mathsf{else}\ \mathsf{fork}(l := 1, l := 0)$ (secure because $\mathsf{fork}(l := 0, l := 1) \sim_L^{uni} \mathsf{fork}(l := 1, l := 0)$). These programs are secure because for each of them we can construct a partial equivalence relation that relates the program to itself and satisfies Definition 1.

Examples of *uni*-insecure programs are $l := h$ (explicit leak), $\mathsf{if}\ h > 0\ \mathsf{then}\ l := 1\ \mathsf{else}\ l := 0$ (implicit leak [6]), $\mathsf{while}\ h > 0\ \mathsf{do}\ h := h - 1$ (timing leak), $\mathsf{if}\ h > 0\ \mathsf{then}\ \mathsf{fork}(l := 0, l := 1)\ \mathsf{else}\ (\mathsf{skip}; l := 0; l := 1)$ (which fails because $\mathsf{fork}(l := 0, l := 1) \not\sim_L^{uni} (\mathsf{skip}; l := 0; l := 1)$). We will return to some of these examples later in this section and in Section 4.

3.2 Scheduler-Independent Security

Scheduler algorithms are typically not specified by implementations. Thus, a higher security assurance than scheduler-specific security is needed—a security specification that is robust with respect to all schedulers. A stronger *scheduler-independent* security gives such assurance by quantifying over all schedulers. We now define both scheduler-independent low-bisimulation $\approx_L$ and scheduler-independent security (SI-bisimulation and SI-security for short):

$$\vec{C} \approx_L \vec{D} \iff \forall\sigma.\, \vec{C} \sim_L^\sigma \vec{D} \qquad\qquad \vec{C} \text{ is } SI\text{-}secure \iff \vec{C} \approx_L \vec{C}$$

Clearly, $\approx_L \subseteq \sim_L^\sigma$ for all σ. An example from the previous section shows that the converse inclusion does not hold. More precisely, the example illustrates that the uniform scheduler assumption does not imply security for other schedulers. Recall the program:

$$\mathsf{if}\ h > 0\ \mathsf{then}\ \mathsf{fork}(l := 0, l := 1)\ \mathsf{else}\ \mathsf{fork}(l := 1, l := 0)$$

While this is a secure program for the uniform scheduler, it is insecure for any other scheduler. Although the attacker may not know the scheduling algorithm in advance, he or she may infer it with sufficient certainty (using a program which does not touch high data) and apply the knowledge to deduce the value of h.

3.3 Strong Security

SI-security guarantees robustness with respect to all schedulers. However, the universal quantification used in its definition complicates direct proofs for SI-security. A compositional property that implies SI-security is an attractive alternative to verifying SI-security directly. Such a compositional property is of particular benefit if it is the *largest* compositional indistinguishability-based property that implies SI-security. We will prove that *strong security* (defined below) is indeed such a property.

Definition 2 *Define the* strong *low-bisimulation* $\cong_L$ *to be the union of all symmetric relations* R *on thread pools of equal size, so that whenever* $\langle C_0 \dots C_{n-1}\rangle$ R $\langle D_0 \dots D_{n-1}\rangle$ *then*

$$\forall s_1 =_L s_2 \forall i. \langle\!| C_i, s_1 |\!\rangle \rightharpoonup \langle\!| \vec{C}', s_1' |\!\rangle \implies \\ \exists \vec{D}', s_2'. \langle\!| D_i, s_2 |\!\rangle \rightharpoonup \langle\!| \vec{D}', s_2' |\!\rangle, \vec{C}' \; R \; \vec{D}', s_1' =_L s_2'$$

Two strongly low-bisimilar thread pools must have the same number of threads, and each thread must execute in lock-step and affect the low memory in the same way as the respective thread in the related thread pool. The strong security specification is given by the following definition:

$$\vec{C} \text{ is } \textit{strongly secure} \iff \vec{C} \cong_L \vec{C}$$

Recalling *uni*-secure examples, programs $h := l + 3; l := l + 1$, $\mathsf{if}\ l > 0\ \mathsf{then}\ h := h + 1\ \mathsf{else}\ h := 0$, and $\mathsf{fork}(l := l + 1, h := h - 1)$ are still secure under strong security. However, the *uni*-secure program $\mathsf{if}\ h > 0\ \mathsf{then}\ \mathsf{fork}(l := 0, l := 1)\ \mathsf{else}\ \mathsf{fork}(l := 1, l := 0)$ is not strongly secure because the component-wise low-bisimilarity for the commands under the fork is broken: $l := 0 \not\cong_L l := 1$.

In general, strong low-bisimulation is indeed stronger than the scheduler-independent one: as desired, strong security implies SI-security (and, thus, all scheduler-specific security properties).

Proposition 1 ([29]) $\cong_L \subseteq \approx_L$.

Corollary 1 $\vec{C}$ *is strongly secure* $\implies \vec{C}$ *is SI-secure.*

Clearly, the converse does not hold. For example, the program

$$\mathsf{if}\ h > 0\ \mathsf{then}\ \mathsf{fork}(\mathsf{skip}, \mathsf{skip})\ \mathsf{else}\ \mathsf{skip}; \mathsf{skip}; \mathsf{skip}$$

is SI-secure (because $\mathsf{fork}(\mathsf{skip}, \mathsf{skip}) \approx_L \mathsf{skip}; \mathsf{skip}; \mathsf{skip}$) but not strongly secure because the number of threads depends on high data (we have $\mathsf{fork}(\mathsf{skip}, \mathsf{skip}) \not\cong_L \mathsf{skip}; \mathsf{skip}; \mathsf{skip}$).

Strong security features a number of compositionality properties. In particular, if a collection of commands are strongly secure then both the sequential (via ";") and parallel (via thread juxtaposition or fork) compositions of the commands are also strongly secure [29]. These are essential properties for modular security program analysis.

4 The Accuracy of Compositional Security

Let us show that strong security relation is the largest compositional relation contained in the SI-security relation. Observe that the combination of scheduler-independence and compositionality results in the condition that two low-bisimilar programs must have the same number of threads. The following example clarifies why this is the case. Consider these programs: the double-threaded $\langle l := 1, l := 1\rangle$ and the single-threaded $l := 1; l := 1$. These programs are SI-bisimilar, i.e., $\langle l := 1, l := 1\rangle \approx_L (l := 1; l := 1)$. Under any scheduler, no attacker is able to distinguish between the executions of the programs. However, the parallel composition of these programs with the thread $l := 2$ breaks low-bisimilarity: $\langle l := 1, l := 1, l := 2\rangle \not\approx_L \langle (l := 1; l := 1), l := 2\rangle$. To see this, it is sufficient to observe that for the uniform scheduler, $\langle l := 1, l := 1, l := 2\rangle \not\sim_L^{uni} \langle (l := 1; l := 1), l := 2\rangle$ as the probability of the value of l being 1 after the first step is $\frac{2}{3}$ for the first program and $\frac{1}{2}$ for the second. (The probabilities of the terminal values of l also differ for the two programs.)

In order to prove a result formalizing the observation that compositionality requires the same number of threads, let us first rigorously define what a compositional indistinguishability relation on thread pools is. For our purposes, it is sufficient to consider a weak definition of compositionality: the compositionality of the parallel composition, requiring $l := 0$ and $l := 1$ to be secure ground contexts (for stronger notions of compositionality our results hold as well).

Definition 3 *An indistinguishability relation R on programs is* compositional *whenever $(l := 0)\ R\ (l := 0), (l := 1)\ R\ (l := 1)$ for some low variable l and if $\vec{C}\ R\ \vec{D}, \vec{C'}\ R\ \vec{D'}$ then $\langle\vec{C}\vec{C'}\rangle\ R\ \langle\vec{D}\vec{D'}\rangle$.*

Clearly, programs $l := 0$ and $l := 1$ are strongly secure. Because strong low-bisimulation is preserved under parallel composition, the strong low-bisimulation security definition is an example of a compositional indistinguishability relation.

4.1 The Accuracy Result via a Special-Purpose Scheduler

A *superincreasing* sequence of integers is a sequence where each element is greater than the sum of all previous elements. Let us define a scheduler *sup* (based on superincreasing sequences). This scheduler will be used for proving the main result of this section. The key property we desire for *sup* is the following. Suppose $X \subseteq \{0 \dots n\}$, $X \neq \{0 \dots n\}$, and $Y \subseteq \{0 \dots m\}$ for some natural n and m. Then

$$\sum_{i\in X} sup(i,n) = \sum_{i \in Y} sup(i,m) \Longleftrightarrow X = Y \ \ \&\ \ n = m \qquad (*)$$

For the scheduler to distinguish between n and m we will use different prime numbers in the denominators of the fractions of the probability values, so that the sum of the probability values for X never equals the sum of the scheduler values for Y (because X is a proper subset of $\{0 \dots n\}$ the values never sum up to 1). If $n = m$ then we use the numerators in the fractions as decimal encodings of X and Y based on a superincreasing sequence $1, 2, 4, 8, \dots$. These encodings are equal if and only if the subsets are equal (no number in a superincreasing sequence can be written as the sum of other elements).

Formally, the sequence $d_1, d_2, \dots$ of denominators is formed by the rules:

$$d_1 = 1 \qquad d_n = \min\{p \mid p \text{ is prime} \ \ \&\ \ p > d_{n-1} \ \ \&\ \ p \geq 2^n - 1\}$$

Note that $\forall n, m (n \neq m).\, gcd(d_n, d_m) = 1$. The scheduler *sup* is defined by:

$$sup(i,n) = \begin{cases} \frac{2^i}{d_n}, & \text{if } i \neq n-1 \\ \frac{d_n - (2^{n-1}-1)}{d_n}, & \text{if } i = n-1 \end{cases}$$

Clearly, *sup* is a valid scheduler that satisfies the property $(*)$.

Proposition 2 $\langle \vec{C}, l := 0, l := 1\rangle \sim_L^{sup} \langle \vec{D}, l := 0, l := 1\rangle \Longrightarrow |\vec{C}| = |\vec{D}|$.

Proof. Consider states s_1 and s_2 such that $\forall Id.\, s_1.Id = s_2.Id = 0$. We have $\langle\!| \langle \vec{C}, l := 0, l := 1\rangle, s_1 |\!\rangle \rightarrow \langle\!| \langle \vec{C}, l := 1\rangle, s_1 |\!\rangle$. By the definition of *sup*-probabilistic low-bisimulation $\exists \vec{D}', s_2'.\, \langle\!| \langle \vec{D}, l := 0, l := 1\rangle, s_2 |\!\rangle \rightarrow \langle\!| \vec{D}', s_2' |\!\rangle$ so that $s_1 =_L s_2'$ and

$$\begin{aligned} &\sum \{\!| p | \langle\!| \langle \vec{C}, l := 0, l := 1\rangle, s_1 |\!\rangle \rightarrow_p \langle\!| \vec{S}, s |\!\rangle, \vec{S} \in [\langle \vec{C}, l := 1\rangle]_{\sim_L^{sup}}, s =_L s_1 |\!\} = \\ &\sum \{\!| p | \langle\!| \langle \vec{D}, l := 0, l := 1\rangle, s_2 |\!\rangle \rightarrow_p \langle\!| \vec{S}, s |\!\rangle, \vec{S} \in [\vec{D}']_{\sim_L^{sup}}, s =_L s_2' |\!\} \end{aligned}$$

Under the scheduler *sup*, the probability of getting from a configuration with the program $\langle \vec{C}, l := 0, l := 1\rangle$ to a configuration a state s such that $s.l = 0$ is less than 1 (because the last thread is possible to execute). Hence, we can rewrite the above sum equality as

$$\sum_{i \in X} sup(i,n) = \sum_{i \in Y} sup(i,m)$$

for some $X \subseteq \{0 \ldots (n+2)\}$, $X \neq \{0 \ldots (n+2)\}$, and $Y \subseteq \{0 \ldots (m+2)\}$ where $n = |\vec{C}|$ and $m = |\vec{D}|$. We conclude that $n = m$ by the property $(*)$. □

Proposition 3 *Strong security is the largest compositional indistinguishability relation contained in the SI-security relation. I.e., if R is a compositional indistinguishability relation and $R \subseteq \approx_L$ then $R \subseteq \cong_L$.*

Proof. Define the projection $\vec{C}\lfloor_X$ for a thread pool $\vec{C}$ and $X \subseteq \{0, \ldots, |\vec{C}|-1\}$ to be the thread pool formed only by those threads from $\vec{C}$ that are indexed in X. Given a relation R satisfying the premises of the proposition, we construct a relation Q such that $R \subseteq Q$ and $Q \subseteq \cong_L$. Define Q by

$$\{(\vec{C}\lfloor_X, \vec{D}\lfloor_X) \mid X \subseteq \{0, \ldots, |\vec{C}|-1\}, \langle \vec{C}, l := 0, l := 1\rangle \sim_L^{sup} \langle \vec{D}, l := 0, l := 1\rangle\}$$

The projection $\vec{D}\lfloor_X$ is well-defined in this definition because the size of $\vec{C}$ and $\vec{D}$ must be the same by Proposition 2. To see that $R \subseteq Q$, suppose $\vec{C}\ R\ \vec{D}$ for some programs $\vec{C}, \vec{D}$. By the compositionality of R, we have $\langle \vec{C}, l := 0, l := 1\rangle\ R\ \langle \vec{D}, l := 0, l := 1\rangle$. Because $R \subseteq \approx_L$ and $\approx_L \subseteq \sim_L^{sup}$, we receive $\langle \vec{C}, l := 0, l := 1\rangle \sim_L^{sup} \langle \vec{D}, l := 0, l := 1\rangle$. Hence, $\vec{C}\ Q\ \vec{D}$ (setting $X = \{0, \ldots, |\vec{C}|-1\}$).

It remains to show that $Q \subseteq \cong_L$. In order to prove this inclusion, we argue that the relation Q is one of the relations from the union that defines strong low-bisimulation (cf. Definition 2). In other words, we need to show that

Q is a symmetric relation on thread pools of equal size such that whenever $\langle C_0 \dots C_{n-1}\rangle \; Q \; \langle D_0 \dots D_{n-1}\rangle$ then

$$\begin{aligned}&\forall s_1 =_L s_2 \forall i. \langle\!| C_i, s_1 |\!\rangle \rightharpoondown \langle\!| \vec{C'}, s_1' |\!\rangle \Longrightarrow \\ &\qquad \exists \vec{D'}, s_2'. \langle\!| D_i, s_2 |\!\rangle \rightharpoondown \langle\!| \vec{D'}, s_2' |\!\rangle, \vec{C'} \; Q \; \vec{D'}, s_1' =_L s_2'\end{aligned}$$

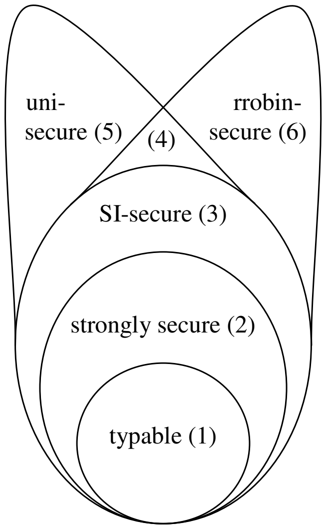
The symmetry of the relation Q follows from that of $\sim_L^{sup}$. Recall that only thread pools of equal size may be related by Q by Proposition 2. Suppose $\vec{C} \downharpoonleft_X Q \; \vec{D} \downharpoonleft_X$ for some $\vec{C} = \langle C_0 \dots C_{n-1}\rangle, \vec{D} = \langle D_0 \dots D_{n-1}\rangle$ where $X \subseteq \{0, \dots, n-1\}$ and $\langle \vec{C}, l := 0, l := 1\rangle \sim_L^{sup} \langle \vec{D}, l := 0, l := 1\rangle$. Assume $\langle\!| C_i, s_1 |\!\rangle \rightharpoondown \langle\!| \vec{C'}, s_1' |\!\rangle$ for some $i (i \in X)$ and s_1. Hence, we have $\langle\!| \langle \vec{C}, l := 0, l := 1\rangle, s_1 |\!\rangle \rightarrow \langle\!| \langle C_0 \dots C_{i-1} \vec{C'} C_{i+1} \dots C_{n-1}, l := 0, l := 1\rangle, s_1' |\!\rangle$. Because $\langle \vec{C}, l := 0, l := 1\rangle \sim_L^{sup} \langle \vec{D}, l := 0, l := 1\rangle$ for all s_2 such that $s_1 =_L s_2$ we have

$$\begin{aligned}&\exists \vec{E}, s_3. \langle\!| \langle \vec{D}, l := 0, l := 1\rangle, s_2 |\!\rangle \rightarrow \langle\!| \vec{E}, s_3 |\!\rangle \text{ such that} \\ (i) &\sum \{\!| p | \langle\!| \langle \vec{C}, l := 0, l := 1\rangle, s_1 |\!\rangle \rightarrow_p \langle\!| \vec{S}, s |\!\rangle, \\ &\qquad \vec{S} \in [\langle C_0 \dots \vec{C'} \dots C_{n-1}, l := 0, l := 1\rangle]_{\sim_L^{sup}}, s =_L s_1' |\!\} = \\ &\sum \{\!| p | \langle\!| \langle \vec{D}, l := 0, l := 1\rangle, s_2 |\!\rangle \rightarrow_p \langle\!| \vec{S}, s |\!\rangle, \vec{S} \in [\vec{E}]_{\sim_L^{sup}}, s =_L s_3 |\!\} \\ (ii) &\; \langle C_0 \dots \vec{C'} \dots C_{n-1}, l := 0, l := 1\rangle \sim_L^{sup} \vec{E}, s_1' =_L s_3\end{aligned}$$

The probability equality (i) has the form $\sum_{k \in Y} sup(k, n) = \sum_{k \in Z} sup(k, n)$ for some $Y, Z \subsetneq \{0, \dots, n-1\}$. The sets Y and Z correspond to different choices of the scheduler that takes the two configurations to the same equivalence classes. As we have observed earlier, the probability equality holds if and only if $Y = Z$. Thus, $\exists \vec{D'}. \langle\!| D_i, s_2 |\!\rangle \rightharpoondown \langle\!| \vec{D'}, s_2' |\!\rangle$ so that $\vec{E} \sim_L^{sup} \langle D_0 \dots D_{i-1} \vec{D'} D_{i+1} \dots D_{n-1}, l := 0, l := 1\rangle$ and $s_3 =_L s_2'$. Therefore, due to $\langle C_0 \dots \vec{C'} \dots C_{n-1}, l := 0, l := 1\rangle \sim_L^{sup} \langle D_0 \dots \vec{D'} \dots D_{n-1}, l := 0, l := 1\rangle$, we have $\vec{C'} \; Q \; \vec{D'}$ which can be seen as $\langle C_0 \dots \vec{C'} \dots C_{n-1}, l := 0, l := 1\rangle \downharpoonleft_{\{i, \dots, (i+m)\}} Q \; \langle D_0 \dots \vec{D'} \dots D_{n-1}, l := 0, l := 1\rangle \downharpoonleft_{\{i, \dots, (i+m)\}}$ where $m = |\vec{C'}| = |\vec{D'}|$. □

4.2 Security Hierarchy

The formal hierarchy of low-bisimulation allows us to derive the relationship between the corresponding security properties. The diagram in Figure 1 illustrates the inclusion relation between security properties. On the right side of the figure we present examples that demonstrate the depicted inclusions. The smallest set on the diagram is the set of *typable* programs. A program is typable if it can be given a type. In the context of security-type systems, programs are given types in order to enforce security. Considering type systems enforcing strong security (e.g., [29]), Program 1 is an example of a typable and, thus, strongly secure program. Because type systems are (necessarily) conservative, there are programs (such as Program 2) that are strongly secure but rejected by most type systems (e.g., [35, 1, 29]).



1. $h := l$
2. $l := h - h$
3. $\mathsf{if}\ h > 0\ \mathsf{then}\ \mathsf{fork}(\mathsf{skip}, \mathsf{skip})$
   $\mathsf{else}\ \mathsf{skip}; \mathsf{skip}; \mathsf{skip}$
4. $\mathsf{if}\ h > 0\ \mathsf{then}\ \mathsf{fork}((l := 0; l := 1), l := 0)$
   $\mathsf{else}\ \mathsf{fork}(l := 0, (l := 0; l := 1))$
5. $\mathsf{if}\ h > 0\ \mathsf{then}\ \mathsf{fork}(l := 0, l := 1)$
   $\mathsf{else}\ \mathsf{fork}(l := 1, l := 0)$
6. $\mathsf{if}\ h > 0\ \mathsf{then}\ \mathsf{fork}(l := 0, l := 1)$
   $\mathsf{else}\ \mathsf{skip}; l := 0; l := 1$

Fig. 1. Security hierarchy.

By Corollary 1, strong security implies scheduler-independent security. However, this inclusion is not strict. Recall that while Program 3 is secure under an arbitrary scheduler (because $\mathsf{fork}(\mathsf{skip}, \mathsf{skip}) \approx_L \mathsf{skip}; \mathsf{skip}; \mathsf{skip}$), it is not strongly secure because the number of threads depends on high data. Program 4 is secure under both the uniform and round-robin[3] schedulers. However, there are schedulers that can be used by the attacker to leak high data through this program. One such scheduler is *sup*. Under this scheduler, the first and second threads are executed with the probabilities $\frac{1}{3}$ and $\frac{2}{3}$. If $h > 0$ then the probability that the final value of l is 1 is $\frac{8}{9}$ whereas if $h \leq 0$ then this probability is only $\frac{5}{9}$. Because of this undesired flow, Program 4 is not SI-secure.

As we pointed out in Section 3, Program 5 is secure under the uniform scheduler, but insecure under any other scheduler (including round-robin). Conversely, program 6 is secure under the round-robin scheduler but not under the uniform scheduler. The security under the uniform scheduler fails because $\mathsf{fork}(l := 0, l := 1) \not\approx_L^{uni} (\mathsf{skip}; l := 0; l := 1)$.

[3] A round-robin scheduler is a deterministic scheduler that executes a thread for one step before switching to the next thread in the configuration (returning to the first thread after executing the last thread). Note that this is a history-dependent scheduler, and, thus, cannot be directly expressed as σ in the semantics. However, we use this scheduler as an example because our results apply to the more general history-dependent scheduler model of [29].

5 Conclusion

We have investigated the relation between different kinds of timing- and probability-sensitive bisimulation for multithreaded programs. Due to the direct connection between bisimulation (as indistinguishability) and confidentiality, our results have been insightful for relating different confidentiality definitions. In particular, we have proved that strong security is the largest compositional security relation contained in the scheduler-independent security relation. This provides a forceful argument in favor of the strong security definition. The main benefit of the strong security definition is compositionality, which is critical for modular security analysis. Clearly, compositionality comes at a price of losing accuracy with respect to scheduler-independent security. However, our results show that we pay the least possible price—there exists no other compositional security relation that subsumes strong security and implies scheduler-independent security.

Related Work There is a large body of work on language-based information security (see [28] for a recent survey). Because compositionality is absolutely crucial for scalable security analysis, it has been an important topic in the context of programming languages [23, 29, 30, 26, 27, 20] as well as process algebra [8, 24, 25, 32, 9, 10], and other frameworks [21, 15, 19].

Focardi et al. [9, 10] and Aldini et al. [2, 3] investigate timing- and probability-sensitive informaion-flow security as independent extensions of Focardi and Gorrieri's security process algebra (SPA) [8]. Both studies explore the relation between different bisimulation-based definition establishing similar inclusion results as in the original security hierarchy.

Focardi et al. [9, 10] follow a similar methodology to ours. Their bisimulation-based timed nondeducability on compositions (*tBNDC*) characterizes a noninterference-like property. It is difficult to verify *tBNDC* directly because it involves a quantification over all contexts. They identify a compositional (in the parallel composition) security property *tSBSNNI* that is amenable to modular analysis and contained in *tBNDC*. However, there are no compositionality results for the probability-sensitive definitions by Aldini et al. [2, 3]. Furthermore, because of an intricate interplay between timing and probabilistic attacks, it appears essential to us to be able to model combined attacks (which is left for future work in [10]). Because our bisimulation-based characterizations are sensitive to both timing and probabilities, combined attacks are a natural part of the attacker model in this paper.

Future Work An important problem for further explorations is whether the argument in favor of strong security can be extended to a multithreaded language with synchronization [26]. As a prerequisite, scheduler-specific and scheduler-independent security properties have to be defined for the extended language.

In a more general perspective, an intriguing direction for future work is integrating bisimulation-based security for programming languages into general bisimulation-based security frameworks (see, e.g., [14] for a step in this direction).

A rigorous connection could illuminate the relation between security properties and aid in designing hybrid mechanisms for their validation. This is a noteworthy goal for future work.

Acknowledgments

Thanks are due to Roberto Gorrieri, David Sands, and Björn Victor for stimulating discussions that have motivated this paper, and to Stephen Chong for valuable comments.

References

1. J. Agat. Transforming out timing leaks. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 40–53, January 2000.
2. A. Aldini. Probabilistic information flow in a process algebra. In *Proc. CONCUR'01*, volume 2154 of *LNCS*, pages 152–168. Springer-Verlag, August 2001.
3. A. Aldini, M. Bravetti, and R. Gorrieri. A process algebraic approach for the analysis of probabilistic non-interference. Technical Report UBLCS-2002-02, University of Bologna, Bologna, Italy, 2002.
4. G. Boudol and I. Castellani. Noninterference for concurrent programs. In *Proc. ICALP*, volume 2076 of *LNCS*, pages 382–395. Springer-Verlag, July 2001.
5. K. Cerans. Decidability of bisimulation equivalences for parallel timer processes. In *Proc. CAV 1992*, volume 663 of *LNCS*, pages 302–315. Springer-Verlag, June 1992.
6. D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Comm. of the ACM*, 20(7):504–513, July 1977.
7. E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *ACM Conference on Computer and Communications Security*, pages 25–32, 2000.
8. R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *J. Computer Security*, 3(1):5–33, 1995.
9. R. Focardi, R. Gorrieri, and F. Martinelli. Information flow analysis in a discrete-time process algebra. In *Proc. IEEE Computer Security Foundations Workshop*, pages 170–184, July 2000.
10. R. Focardi, R. Gorrieri, and F. Martinelli. Real-time information flow analysis. *IEEE J. Selected Areas in Communications*, 21(1):20–35, January 2003.
11. P. Giambiagi and M.Dam. On the secure implementation of security protocols. In *Proc. European Symposium on Programming*, volume 2618 of *LNCS*, pages 144–158. Springer-Verlag, April 2003.
12. J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symp. on Security and Privacy*, pages 11–20, April 1982.
13. J.W. Gray III. Probabilistic interference. In *Proc. IEEE Symp. on Security and Privacy*, pages 170–179, May 1990.
14. K. Honda and N. Yoshida. A uniform type structure for secure information flow. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 81–92, January 2002.
15. J. Jürjens. Secure information flow for concurrent processes. In *Proc. CONCUR 2000*, volume 1877 of *LNCS*, pages 395–409. Springer-Verlag, August 2000.

16. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Proc. CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer-Verlag, 1996.
17. D. Kozen. Language-based security. In *Proc. Mathematical Foundations of Computer Science*, volume 1672 of *LNCS*, pages 284–298. Springer-Verlag, September 1999.
18. K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.
19. H. Mantel. On the composition of secure systems. In *Proc. IEEE Symp. on Security and Privacy*, pages 81–94, May 2002.
20. H. Mantel and A. Sabelfeld. A unifying approach to the security of distributed and multi-threaded programs. *J. Computer Security*, 2003. To appear.
21. D. McCullough. Specifications for multi-level security and hook-up property. In *Proc. IEEE Symp. on Security and Privacy*, pages 161–166, April 1987.
22. J. McLean. Security models and information flow. In *Proc. IEEE Symp. on Security and Privacy*, pages 180–187, May 1990.
23. J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. IEEE Symp. on Security and Privacy*, pages 79–93, May 1994.
24. P. Ryan and S. Schneider. Composing and decomposing systems under security properties. In *Proc. IEEE Computer Security Foundations Workshop*, pages 9–15, March 1995.
25. P. Ryan and S. Schneider. Process algebra and non-interference. In *Proc. IEEE Computer Security Foundations Workshop*, pages 214–227, June 1999.
26. A. Sabelfeld. The impact of synchronisation on secure information flow in concurrent programs. In *Proc. Andrei Ershov International Conference on Perspectives of System Informatics*, volume 2244 of *LNCS*, pages 227–241. Springer-Verlag, July 2001.
27. A. Sabelfeld and H. Mantel. Static confidentiality enforcement for distributed programs. In *Proc. Symposium on Static Analysis*, volume 2477 of *LNCS*, pages 376–394. Springer-Verlag, September 2002.
28. A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, January 2003.
29. A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *Proc. IEEE Computer Security Foundations Workshop*, pages 200–214, July 2000.
30. A. Sabelfeld and D. Sands. A per model of secure information flow in sequential programs. *Higher Order and Symbolic Computation*, 14(1):59–91, March 2001.
31. F. B. Schneider, G. Morrisett, and R. Harper. A language-based approach to security. In *Informatics—10 Years Back, 10 Years Ahead*, volume 2000 of *LNCS*, pages 86–101. Springer-Verlag, 2000.
32. S. Schneider. May testing, non-interference, and compositionality. Technical Report CSD-TR-00-02, Royal Holloway, University of London, January 2001.
33. G. Smith. A new type system for secure information flow. In *Proc. IEEE Computer Security Foundations Workshop*, pages 115–125, June 2001.
34. D. Volpano and G. Smith. Verifying secrets and relative secrecy. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 268–276, January 2000.
35. D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *J. Computer Security*, 4(3):167–187, 1996.
36. Y. Wang. Real-time behaviour of asynchronous agents. In *Proc. CONCUR 1990*, volume 458 of *LNCS*, pages 502–520. Springer-Verlag, August 1990.

Appendix A

$$
\begin{aligned}
C ::= &\ \mathsf{skip} \mid Id := Exp \mid C_1; C_2 \\
&\mid \mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \mid \mathsf{while}\ B\ \mathsf{do}\ C \mid \mathsf{fork}(C, \vec{D})
\end{aligned}
$$

Fig. 2. Command syntax.

$$\langle\!|\mathsf{skip}, s|\!\rangle \rightharpoonup \langle\!|\langle\rangle, s|\!\rangle$$

$$\frac{\langle\!|Exp, s|\!\rangle \downarrow n}{\langle\!|Id := Exp, s|\!\rangle \rightharpoonup \langle\!|\langle\rangle, [Id = n]s|\!\rangle}$$

$$\frac{\langle\!|C_1, s|\!\rangle \rightharpoonup \langle\!|\langle\rangle, s'|\!\rangle}{\langle\!|C_1; C_2, s|\!\rangle \rightharpoonup \langle\!|C_2, s'|\!\rangle} \qquad \frac{\langle\!|C_1, s|\!\rangle \rightharpoonup \langle\!|C_1'\vec{D}, s'|\!\rangle}{\langle\!|C_1; C_2, s|\!\rangle \rightharpoonup \langle\!|(C_1'; C_2)\vec{D}, s'|\!\rangle}$$

$$\frac{\langle\!|B, s|\!\rangle \downarrow \mathsf{True}}{\langle\!|\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, s|\!\rangle \rightharpoonup \langle\!|C_1, s|\!\rangle}$$

$$\frac{\langle\!|B, s|\!\rangle \downarrow \mathsf{False}}{\langle\!|\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, s|\!\rangle \rightharpoonup \langle\!|C_2, s|\!\rangle}$$

$$\frac{\langle\!|B, s|\!\rangle \downarrow \mathsf{True}}{\langle\!|\mathsf{while}\ B\ \mathsf{do}\ C, s|\!\rangle \rightharpoonup \langle\!|C; \mathsf{while}\ B\ \mathsf{do}\ C, s|\!\rangle}$$

$$\frac{\langle\!|B, s|\!\rangle \downarrow \mathsf{False}}{\langle\!|\mathsf{while}\ B\ \mathsf{do}\ C, s|\!\rangle \rightharpoonup \langle\!|\langle\rangle, s|\!\rangle}$$

$$\langle\!|\mathsf{fork}(C, \vec{D}), s|\!\rangle \rightharpoonup \langle\!|C\vec{D}, s|\!\rangle$$

Fig. 3. Small-step deterministic semantics of commands.

$$\text{(Pick)}\frac{\langle\!|C_i, s|\!\rangle \rightharpoonup \langle\!|\vec{C}, s'|\!\rangle}{\langle\!|\langle C_0 \dots C_{n-1}\rangle, s|\!\rangle \rightarrow^{i}_{\sigma(i,n)} \langle\!|\langle C_0 \dots C_{i-1}\vec{C}C_{i+1} \dots C_{n-1}\rangle, s'|\!\rangle}$$

$$\text{(Sum)}\frac{p = \sum \{\!|q \mid \langle\!|\vec{C}, s|\!\rangle \rightarrow^{i}_{q} \langle\!|\vec{D}, s'|\!\rangle|\!\} \quad p \neq 0}{\langle\!|\vec{C}, s|\!\rangle \rightarrow_{p} \langle\!|\vec{D}, s'|\!\rangle}$$

Fig. 4. Probabilistic semantics of thread pools.