Language-based Security and Time-inserting Supervisor

Damas P. Gruska

Department of Applied Informatics, Comenius University,
Mlynska dolina, 842 48 Bratislava, Slovakia,
gruska@fmph.uniba.sk.

Abstract. Algebraic methods are employed in order to define language-based security properties of processes. A supervisor is introduced that can disable unwanted behavior of an insecure process by controlling some of its actions or by inserting timed actions to make an insecure process secure. We assume a situation where neither the supervisor nor the attacker has complete information about the ongoing systems behavior. We study the conditions under which such a supervisor exists, as well as its properties and limitations.

1 Introduction

Exploiting formal models and methods to define, study, or even discover system vulnerabilities regarding security threats is a hot topic nowadays. Formal methods allow us, in many cases, to show, even prove, that a given system is not secure. Then we have a couple of options as to what to do. We can either re-design its behavior, which might be costly, difficult, or even impossible, in the case that it is already part of a hardware solution, proprietary firmware, and so on, or we can use supervisory control (see [26, 22, 23, 7, 27]) or other means ([16]) to restrict the system's behavior in such a way that the system becomes secure. A supervisor can see (some) system actions and can control (disable or enable) a set of system actions. In this way, it restricts the behavior of the system to guarantee its security (see also [14]). This is a trade-off between security and functionality.

The situation is different in the case of timing attacks. They use timing information leakage, which is the ability for an attacker to deduce internal (private) information depending on timing information. They, as side-channel attacks, represent a serious threat to many systems. They allow intruders "break" "unbreakable" systems, algorithms, protocols, etc.

One way to limit time attacks is to use a function that inserts time delays between individual actions, making it impossible for an attacker to obtain sensitive information. In this paper, we will introduce a special

type of supervisor, in addition to limiting actions, will also be able to insert time delays to guarantee the safety of the process.

As for the formalism, we will work with a timed process algebra and language-based opacity. This formalism also allows us to formalize timing attacks. Process algebras, unlike finite automata or Timed Automata, allow us to use various compositional constructs to express observations as well as intruders and supervisors themselves. In [12, 13] we studied conditions under which there exists a timed insertion function for a given process and state-based security property and we have presented some decidability and undecidability results. Here we study this concept for language-based security properties and combine insertion functions with supervisors.

Supervisors that can guarantee the security of systems concerning opacity are often studied for finite automata or other models with a finite number of states. Here we work with a formalism equivalent to a Turing machine (see [21]), which introduces new challenges. We show whether the existence of a supervisor is decidable or undecidable. In addition, we assume more general observations, not only those that hide some actions. *Main contributions of this paper* We work with a supervisor who is

- can only partially observe systems,
- can enable or disable actions, and also can insert time actions,
- are modeled by processes that interact with a system to be attacked.

The paper is organized as follows. In Section 2, we describe the timed process algebra TPA, which will be used as a basic formalism. Language-based security properties are introduced in the next section. The time inserting supervisor is defined and studied in Section 4.

2 Working Formalism

In this section, we briefly recall our working formalism which will be based on Timed Process Algebra, TPA for short. TPA itself is based on Milner's Calculus of Communicating Systems (for short, CCS, see [21]), so we will start with this. To define the language CCS, we first presuppose a set of atomic action symbols A not containing symbols τ, and that for every $a \in A$ there exists an $\bar{a} \in A$ and $\bar{\bar{a}} = a$ i.e. actions which represent receiving and sending from a channel a, respectively. We define $Act = A \cup \{\tau\}$ where τ represents internal action, for example, a result of internal communication. We let $a, b, \ldots$ range over A; $x, y, \ldots$ range over Act. The set of CSS terms is defined by the following BNF expression

where $x \in Act, X \in Var$, Var is a set of process variables, f is ranging over relabelling functions, $f : Act \to Act$ is such that $\overline{f(a)} = f(\bar{a})$ for $a \in A, f(\tau) = \tau$ and finally, $L \subseteq A$:

$$P ::= Nil \mid X \mid x.P \mid P + P \mid P \mid P \mid \mu XP \mid P \setminus L \mid P[f]$$

We use the usual definitions for free and bound variables, open and closed terms and guarded recursion. The set **CCS** of *processes* consists of closed and guarded CCS terms.

Nil represents process doing nothing, X is a process variable. Process $x.P$ can perform action x and then behaves as process P (prefix operator) , $+$, $\mid$ represent nondeterministic choice and parallel operator, respectively. A recursion operator is denoted by μXP i.e. recursive definition of the process given by equation $X = P$. The unary operators $P \setminus L$ and $P[f]$ represent restriction and relabeling, respectively. Formal definition of a structural operational semantics for **CCS** is defined in terms of Labeled Transition Systems.

Definition 1. *A Labeled Transition System is a triple* $(S, T, \{\overset{x}{\to}, x \in T\})$ *where* S *is a set of states,* T *is a set of labels and* $\{\overset{x}{\to}, x \in T\}$ *is the transition relation such that each* $\overset{x}{\to}$ *is a binary transition relation on* S. *We write* $P \overset{x}{\to} P'$ *instead of* $(P, P') \in \overset{x}{\to}$ *and* $P \overset{x}{\not\to}$ *if there is no* P' *such that* $P \overset{x}{\to} P'$.

As a set of states we use the set of CCS terms, the set of labels is equal to Act, and transition relations are defined as follows.

Definition 2. *The transition relations* $T = \{\overset{x}{\to}_{CCS}, x \in Act\}$ *are defined as the least relations satisfying the following inference rules:*

$$\frac{}{x.P \overset{x}{\to} P} \qquad\qquad \frac{P \overset{x}{\to} P'}{P + Q \overset{x}{\to} P', \; Q + P \overset{x}{\to} P'}$$

$$\frac{P \overset{x}{\to} P'}{P \setminus L \overset{x}{\to} P' \setminus L}, (x, \bar{x} \notin L) \qquad\qquad \frac{P[\mu XP/X] \overset{x}{\to} P'}{\mu XP \overset{x}{\to} P'}$$

$$\frac{P \overset{x}{\to} P'}{P[f] \overset{f(x)}{\to} P'[f]} \qquad\qquad \frac{P \overset{a}{\to} P', Q \overset{\bar{a}}{\to} Q'}{P \mid Q \overset{\tau}{\to} P' \mid Q'}$$

$$\frac{P \overset{u}{\to} P'}{P \mid Q \overset{u}{\to} P' \mid Q, Q \mid P \overset{u}{\to} Q \mid P'}$$

We are now ready to define the working formalism, which is the time extension of CCS. In TPA we use the special time action t which expresses elapsing of (discrete) time is added and hence the set of actions is extended from Act to $Actt$. The presented language is a slight simplification of Timed Security Process Algebra (tSPA) introduced in [5]. We omit an explicit idling operator ι used in tSPA and instead of this we allow implicit idling of processes. Hence processes can perform either "enforced idling" by performing t actions which are explicitly expressed in their descriptions or "voluntary idling" (i.e. for example, the process $a.Nil$ can perform t action despite the fact that this action is not formally expressed in the process specification). But in both cases, internal communications have priority to action t in the parallel composition. Moreover, we do not divide actions into private and public ones as it is in tSPA. TPA differs also from the tCryptoSPA (see [9]). TPA does not use value passing and strictly preserves *time determinancy* in case of choice operator $+$ what is not the case of tCryptoSPA (see [10]). To define $At = A \cup \{t\}, Actt = Act \cup \{t\}$, moreover we suppose that $S(t) = t$ for every relabelling function S. We give a structural operational semantics of terms again by means of labeled transition systems. The set of terms represents a set of states, labels are actions from $Actt$. The transition relation $\rightarrow$ is a subset of $\text{TPA} \times Actt \times \text{TPA}$. We define the transition relation as the least relation satisfying the inference rules for CCS plus the following inference rules for t action (for more details see [10]).:

$$\frac{}{Nil \xrightarrow{t} Nil} \quad A1 \qquad \frac{}{u.P \xrightarrow{t} u.P} \quad A2$$

$$\frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q', P \mid Q \xrightarrow{\tau}}{P \mid Q \xrightarrow{t} P' \mid Q'} \; Pa \quad \frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q'}{P + Q \xrightarrow{t} P' + Q'} \; S$$

For $s = x_1.x_2.\ldots.x_n, x_i \in Act$ we write $P \xrightarrow{s}$ instead of $P \xrightarrow{x_1}\xrightarrow{x_2} \ldots \xrightarrow{x_n}$ and we say that s is a trace of P. The set of all traces of P will be denoted by $Tr(P)$. By ϵ we denote the empty sequence and by M^* we denote the set of finite sequences of elements from M. We use $\xRightarrow{x}$ as an abbreviation for transitions including τ actions i.e. $(\xrightarrow{\tau})^* \xrightarrow{x} (\xrightarrow{\tau})^*$ (see [21]). By $s|_B$ we will denote the sequence obtained from s by removing all actions not belonging to B. By $L(P)$ we will denote a set of actions that can be performed by P, i.e. $L(P) = \{x | P \xrightarrow{s.x}, s \in Actt^*\}$.

Now we define two behavior equivalences, namely weak trace equivalence and bisimulation, respectively (see [21]).

Definition 3. *The set of weak traces of process P is defined as $Tr_w(P) = \{s \in (A \cup \{t\})^\star | \exists P'.P \stackrel{s}{\Rightarrow} P'\}$. Two processes P and Q are weak trace equivalent (denoted as $P \approx_w Q$) iff $Tr_w(P) = Tr_w(Q)$.*

Definition 4. *Let $(TPA, Act, \rightarrow)$ be a labeled transition system (LTS). A relation $\Re \subseteq TPA \times TPA$ is called a* bisimulation *if it is symmetric and it satisfies the following condition: if $(P,Q) \in \Re$ and $P \stackrel{x}{\rightarrow} P', x \in Actt$ then there exists a process Q' such that $Q \stackrel{x}{\rightarrow} Q'$ and $(P',Q') \in \Re$. Two processes P, Q are bisimilar, abbreviated $P \sim Q$, if there exists a bisimulation relating P and Q.*

3 Opacity

System security based on the absence of information flow assumes that an attacker observing the system is unable to determine classified properties of the system. This concept probably first appeared in[8]. To formalize an information flow we do not divide actions into public and private ones at the system description level, as it is done for example in [9], but we use a more general concept of observation and opacity. This concept was exploited in [3] and [2] in a framework of Petri Nets and transition systems, respectively. Firstly we define the observation function on sequences from $Act^\star$. Various variants of observation functions differ according to contexts which they take into account. For example, an observation of action can depend on the previous actions.

Definition 5. (Observation) *Let Θ be a set of elements called observables. Any function $\mathcal{O} : Actt^\star \rightarrow \Theta^\star$ is an observation function. It is called static /dynamic /orwellian / m-orwellian ($m \geq 1$) if the following conditions hold respectively (below we assume $w = x_1 \ldots x_n$):*

- *static if there is a mapping $\mathcal{O}' : Actt \rightarrow \Theta \cup \{\epsilon\}$ such that for every $w \in Actt^\star$ it holds $\mathcal{O}(w) = \mathcal{O}'(x_1) \ldots \mathcal{O}'(x_n)$,*
- *dynamic if there is a mapping $\mathcal{O}' : Actt^\star \rightarrow \Theta \cup \{\epsilon\}$ such that for every $w \in Actt^\star$ it holds $\mathcal{O}(w) = \mathcal{O}'(x_1).\mathcal{O}'(x_1.x_2) \ldots \mathcal{O}'(x_1 \ldots x_n)$,*
- *orwellian if there is a mapping $\mathcal{O}' : Actt \times Actt^\star \rightarrow \Theta \cup \{\epsilon\}$ such that for every $w \in Actt^\star$ it holds $\mathcal{O}(w) = \mathcal{O}'(x_1, w).\mathcal{O}'(x_2, w) \ldots \mathcal{O}'(x_n, w)$,*
- *m-orwellian if there is a mapping $\mathcal{O}' : Actt \times Actt^\star \rightarrow \Theta \cup \{\epsilon\}$ such that for every $w \in Actt^\star$ it holds $\mathcal{O}(w) = \mathcal{O}'(x_1, w_1).\mathcal{O}'(x_2, w_2) \ldots \mathcal{O}'(x_n, w_n)$ where $w_i = x_{max\{1, i-m+1\}}.x_{max\{1, i-m+1\}+1} \cdots x_{min\{n, i+m-1\}}$.*

In the case of the static observation function, each action is observed independently from its context. In the case of the dynamic observation

function, an observation of an action depends on the previous ones, in the case of the orwellian and m-orwellian observation function an observation of an action depends on the all and on m previous actions in the sequence, respectively. The static observation function is the special case of m-orwellian one for $m = 1$. Note that from the practical point of view the m-orwellian observation functions are the most interesting ones. An observation expresses what an observer - eavesdropper can see from a system behavior and we will alternatively use both the terms (observation - observer) with the same meaning. Note that the same action can be seen differently during an observation (except static observation function) and this expresses a possibility of accumulating some knowledge by an intruder. For example, an action not visible at the beginning could become somehow observable. An observation function can be naturally extended to a set of sequences. From now on we will assume that $\Theta \subseteq Actt$.

Now suppose that we have some security property over process traces. This might be an execution of one or more classified actions, an execution of actions in a particular classified order which should be kept hidden, etc. Suppose that this property is expressed by the predicate ϕ over process traces. We would like to know whether an the observer can deduce the validity of the property ϕ just by observing sequences of actions from $Actt^\star$ performed by given process. The observer cannot deduce the validity of ϕ if there are two traces $w, w' \in Actt^\star$ such that $\phi(w), \neg\phi(w')$ and the traces cannot be distinguished by the observer i.e. $\mathcal{O}(w) = \mathcal{O}(w')$. We formalize this concept by opacity.

Definition 6. (Language Opacity) *Given process P, a predicate ϕ over $Actt^\star$ is opaque w.r.t. the observation function $\mathcal{O}$ if for every sequence w, $w \in Tr(P)$ such that $\phi(w)$ holds and $\mathcal{O}(w) \neq \epsilon$, there exists a sequence $w', w' \in Tr(P)$ such that $\neg\phi(w')$ holds and $\mathcal{O}(w) = \mathcal{O}(w')$. The set of processes for which the predicate ϕ is language opaque with respect to $\mathcal{O}$ will be denoted by $LOp_\mathcal{O}^\phi$.*

A predicate is opaque if for any trace of a system for which it holds, there exists another trace for which it does not hold and both traces are indistinguishable for an observer (which is expressed by an observation function). This means that the observer (intruder) cannot say whether a trace for which the predicate holds has been performed or not.

Suppose that all actions are divided into two groups, namely public (low level) actions L and private (high level) actions H. It is assumed that $L \cup H = A$. Strong Nondeterministic Non-Interference (SNNI, for short, see [5]) property assumes an intruder who tries to learn whether

a private action was performed by a given process while (s)he can observe only public ones. If this cannot be done, then the process has SNNI property. Note that SNNI property is the special case of opacity for static observation function $\mathcal{O}(x) = x$ iff $x \notin H$ and $\mathcal{O}(x) = \epsilon$ otherwise, and $\phi(w)$ is such that it is true iff w contains an action from H.

We can define partial ordering on the set of observation functions which reflects strength of observations.

Definition 7. (Ordering on Observation Functions) *Given two observation functions $\mathcal{O}_1, \mathcal{O}_2$ we say that observation function $\mathcal{O}_2$ is stronger than $\mathcal{O}_1$, denoted as $\mathcal{O}_1 \preceq \mathcal{O}_2$ iff whenever $w, w' \in Actt^*$ such that $\mathcal{O}_2(w) = \mathcal{O}_2(w)$ then also $\mathcal{O}_1(w) = \mathcal{O}_1(w)$. If $\mathcal{O}_1 \preceq \mathcal{O}_2$ and $\mathcal{O}_2 \preceq \mathcal{O}_1$ we say that the observation functions are comparable (denoted as $\mathcal{O}_1 \simeq \mathcal{O}_2$)*

Proposition 1. *Let $\mathcal{O}_1, \mathcal{O}_2$ be two observation functions and such that $\mathcal{O}_2 \preceq \mathcal{O}_1$. Let ϕ_1, ϕ_2 be two predicates over sequences such that $\phi_2 => \phi_1$. Then $LOp_{\mathcal{O}_1}^{\phi_1} \subseteq LOp_{\mathcal{O}_2}^{\phi_2}$.*

Proof. Let $P \in LOp_{\mathcal{O}_1}^{\phi_1}$ and let $w \in Tr(P)$ such that $\phi_2(w)$ holds $\mathcal{O}_2(w) \neq \epsilon$. Since $\phi_2 => \phi_1$ we have $\phi_1(w)$ holds. Since $P \in LOp_{\mathcal{O}_1}^{\phi_1}$ we have that there exists a sequence $w', w' \in Tr(P)$ such that $\neg\phi_1(w')$ holds and $\mathcal{O}_1(w) = \mathcal{O}_1(w')$. Since $\neg\phi_1 => \neg\phi_2$ we have that also $\neg\phi_2(w')$ holds and since $\mathcal{O}_2 \preceq \mathcal{O}_1$ we have that $\mathcal{O}_2(w) = \mathcal{O}_2(w')$ and hence $P \in LOp_{\mathcal{O}_2}^{\phi_2}$.

4 Inserting Supervisor

In this section, we will deal with the problem of what to do if a process is not secure in the sense of language opacity. Changing its design is often not a solution because of the high price, already existing hardware implementation, etc.

We will work with the concept of a supervisor who monitors the process and in case there is a threat of leakage of sensitive information, its activity will either be completely terminated or modified so that leakage is not possible despite its continuation. In this case, it tries to insert time actions and thus precedes especially, for example, time attacks that uses time information. In most cases, our formalism allows the insertion of time actions (see rules A1, A2, Pa, S in Section 2), so that the resulting sequence of actions is still possible and therefore not suspect. Similar to the attacker, whose observation skills are expressed by his observational function, we will also assume that the supervisor has only partial information about the system's activity, expressed by its own observational function.

4.1 Safe traces and supervisors

Now we define a set of L-safe traces of a process, which are those traces which cannot leak validity of ϕ under a given observation. The task of the supervisor is to ensure that the traces of process P always belong to this set. Later, with the help of TPA processes we will model not only the system whose security we want to ensure but also the observational functions, the predicate ϕ as well as supervisors. This allow us to study the very existence of the supervisor for a given process.

Definition 8. (L-Safe Traces) *Given process P, a predicate ϕ over $Actt^\star$ and the observation function $\mathcal{O}$. We define $K^{P,\phi,\mathcal{O}} \subseteq Tr(P)$ as $w \in K^{P,\phi,\mathcal{O}}$ iff $\neg\phi(w)$ holds or there exists a sequence $w', w' \in Tr(P)$ such that $\neg\phi(w')$ holds and $\mathcal{O}(w) = \mathcal{O}(w')$.*

Clearly, we have $K^{P,\phi,\mathcal{O}} = Tr(P)$ iff $P \in LOp_{\mathcal{O}}^{\phi}$, hence if $P \notin LOp_{\mathcal{O}}^{\phi}$ we have $K^{P,\phi,\mathcal{O}} \subset Tr(P)$.

So, to keep process P secure with respect to language opacity, we need to prohibit it from performing any trace from $Tr(P) \setminus K^{P,\phi,\mathcal{O}}$. To do so we exploit a special concept of supervisor Sup, which can prohibit some actions and it can also insert a sequence of actions t so that the resulting trace performed by supervised process always belong to $K^{P,\phi,\mathcal{O}}$.

We assume that similarly to an attacker also the observer cannot see all process actions, which can be expressed by its own, possibly different, observational function $\mathcal{O}_S$ (see Definition 5) i.e. in general $\mathcal{O}_S$ and $\mathcal{O}_S$ might be two different functions. After observing the sequence of the process actions by $\mathcal{O}_S$, the supervisor decides whether to intervene by prohibiting the action from C or inserting time delays. If it prohibits the action, the process execution stops.

Example 1. Let $\phi(w)$ holds iff w contains action h and $\mathcal{O} = \mathcal{O}_s$ are static observation function such that $\mathcal{O}(h) = \epsilon$ and $\mathcal{O}(x) = x$ for $x \neq h$. Then for $P_1 = h.l.Nil$ we have $P_1 \notin LOp_{\mathcal{O}}^{\phi}$ and a supervisor has to prohibit action l to keep process P_1 secure. For $P_2 = h.l.Nil + l.Nil$ we have $P_2 \in LOp_{\mathcal{O}}^{\phi}$ and a supervisor need to do nothing. For $P_3 = h.l.Nil + t.l.Nil$ the supervisor has to insert action t before performing l.

We will model both the supervisor and its observational function as separate processes that communicate with process P (see Fig. 1). A trace w of process P is first translated to a sequence seen by the supervisor by process $\mathcal{O}_S$ and then the supervisor decides whether to forbid the next action or inserts t action.

We restrict the ability of the supervisor to disable only actions from a set of controllable actions $C \subseteq A$. We can see the supervisor Sup_C (we will write just Sup for a given C) as mapping which takes a trace w of a process P as it is seen by its observation function $\mathcal{O}_S$ and decides whether a next action can be performed but only in the case that it is controllable, moreover it can insert a sequence of t actions all to guarantie that that resulting sequence belongs to $K^{P,\phi,\mathcal{O}}$. Formally, $Sup \circ \mathcal{O}_S : Tr(P) \rightarrow K^{P,\phi,\mathcal{O}}$ such that $Sup(\mathcal{O}_S(w.x)) = Sup(\mathcal{O}_S(w)).x.t^i$ iff $x \in Actt \setminus C$ and $Sup(\mathcal{O}_S(w.x)) = Sup(\mathcal{O}_S(w)).y.t^i$ iff $x \in C$ where $y \in \{x, \epsilon\}$.

Now we define a set of all supervisor which gurantee language based security for a given process P and corresponding observation functions and the predicate.

Definition 9. (Set of Supervisors) *Given process P, a predicate ϕ over $Actt^\star$, $C \subset A$ and the observation functions $\mathcal{O}, \mathcal{O}_S$. We define $Sup(P, LOp_{\mathcal{O}}^{\phi}, C, \mathcal{O}_S)$ as a set of all supervisors such that $Tr(\mathcal{C}(P, \mathcal{O}_S, Sup)) \subseteq K^{P,\phi,\mathcal{O}}Tr(P)$.*

Clearly, $Sup(P, LOp_{\mathcal{O}}^{\phi}, C, \mathcal{O}_S) = Sup(P', LOp_{\mathcal{O}}^{\phi}, C, \mathcal{O}_S)$ if P and P' are bisimilar, i.e. $P \sim P'$.

Definition 10. (Controllability) *Given process P, a predicate ϕ over processes, $C \subset A$ and the observation functions $\mathcal{O}, \mathcal{O}_S$. We say that set $T^{P,\phi,\mathcal{O}}$ is controllable iff $w \in T^{P,\phi,\mathcal{O}}$ and $x \in A \setminus C$ then $w.x \in T^{P,\phi,\mathcal{O}}$.*

Proposition 2. *Given process P, a predicate ϕ over processes, $C \subset A$ and the observation functions $\mathcal{O}, \mathcal{O}_S$. Let set $T^{P,\phi,\mathcal{O}}$ is not controllable. Then $Sup(P, LOp_{\mathcal{O}}^{\phi}, C, \mathcal{O}_S) = \emptyset$.*

Proof. Let set $T^{P,\phi,\mathcal{O}}$ is not controllable. Hence there exists $w \in T^{P,\phi,\mathcal{O}}$ and $x \in A \setminus C$ such that $w.x \in\in T^{P,\phi,\mathcal{O}}$ i.e. no supervisor can prohibit process P to reach an unsefa state.

Corollary Given process P, a predicate ϕ over processes, $C = A$ and the observation functions $\mathcal{O}, \mathcal{O}_S$. Then $Sup(P, LPOp_{\mathcal{O}}^{\phi}, C, \mathcal{O}_S) \neq \emptyset$.

To guarantee a minimal restriction of process behavior our aim is to find a maximal process supervisor in the sense that it minimally restricts the behavior of the original process. The formal definition is the following.

Definition 11. (Maximal Supervisor for Process Opacity) *Process $Sup \in Sup(P, LOp_{\mathcal{O}}^{\phi}, C, \mathcal{O}_S)$ is called maximal process supervisor for language opacity $LOp_{\mathcal{O}}^{\phi}$ iff for every $Sup' \in Sup(P, LOp_{\mathcal{O}}^{\phi}, C, \mathcal{O}_S)$ we have $Tr(\mathcal{C}(P, \mathcal{O}_S, Sup')) \subseteq Tr(\mathcal{C}(P, \mathcal{O}_S, Sup))$.*

Unfortunately it is undecidable to verify whether the process Sup is a process supervisors for P and language opacity as it is stated by the following proposition.

Proposition 3. *The property that Sup is a process supervisor for language opacity for process P is undecidable in general.*

Proof. The proof is based on the idea that already language opacity is undecidable (see Proposition 2. in [11]). Suppose that the property is decidable. Let $Sup = \mu X. \sum_{x \in Actt} x'.x.X$ i.e Sup does not restrict anything. We have that Sup is a supervisor for language opacity for process P iff $P \in LOp_{\mathcal{O}}^{\phi}$). Hence we would be able to decide language opacity what contradicts its undecidability.

By a similar argument, we can prove the following statement, which claims that we cannot even decide whether there is at least one supervision that guarantees the security of systems.

Proposition 4. *It is undecidable whether $Sup(P, LOp_{\mathcal{O}}^{\phi}, C, \mathcal{O}_S) = \emptyset$ in general.*

In the proposition that follows, we articulate the underlying assumptions that rigorously guarantee the existence of a nontrivial supervisor.

Proposition 5. *Given process P, a predicate ϕ over processes and the observation functions $\mathcal{O}, \mathcal{O}_S$ such that $\mathcal{O} \preceq \mathcal{O}_S$ and $C = A$. Then $Sup(P, LOp_{\mathcal{O}}^{\phi}, C, \mathcal{O}_S)$ contains at least one nontrivial supervisor, i.e. such which does not prohibit all actions.*

Proof. Let $w \in Tr(P) \setminus T^{P,\phi,\mathcal{O}}$. If such w does not exist than a supervisor which does not prohibit any action as well as does not insert actions t, guarantees security of P. The supervisor's job is to prohibit P from performing w. Since $\mathcal{O} \preceq \mathcal{O}_S$ thee supervisor does not see less than an attacker and can prohibit only those actions that would help an intruder learn the validity of ϕ.

4.2 Timing Attacks

Our formal model can also account for the time-dependent aspects of system behavior, process opacity can also be used to express vulnerabilities to timing attacks. These attacks use timing information leakage, which is the ability of an attacker to deduce internal (private) information depending on timing information. They, as side-channel attacks, represent a serious

threat to many systems. They allow intruders "break" "unbreakable" systems, algorithms, protocols, etc. For example, by carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems (see [19]). This idea was developed in [4] where a timing attack against smart card implementation of RSA was conducted. In [15], a timing attack on the RC5 block encryption algorithm is described. The analysis is motivated by the possibility that some implementations of RC5 could result in data-dependent rotations taking a time that is a function of the data. In [17], the vulnerability of two implementations of the Data Encryption Standard (DES) cryptosystem under a timing attack is studied. It is shown that a timing attack yields the Hamming weight of the key used by both DES implementations. Moreover, the attack is computationally inexpensive. A timing attack against an implementation of AES candidate Rijndael is described in [20], and the one against the popular SSH protocol in [24]. In [1] several novel timing attacks against the common table-driven software implementation of the AES cipher are described. Also possible attacks on most of the currently used processors (Meltdown and Spectre) belong to timing attacks. Timing attacks on web privacy and some corresponding formal models can be found in [6]. Let $\mathcal{O}$, $\mathcal{O}'$ be observation functions such that $\mathcal{O}(w|_{Act}) = \mathcal{O}'(w)$ where $w|_{Act}$ represents the sequence w without t actions. Then P is prone to timing attacks iff $P \notin LOp_{\mathcal{O}}^{\phi}$ but $P \in LOp_{\mathcal{O}'}^{\phi}$. Consequently, the elapsed time becomes a side channel, where the duration of the computation itself reveals information about the validity of ϕ. For instance, a shorter execution time for a particular input might indicate that a certain condition related to was met (or not met), allowing the attacker to deduce properties of the secret."

Let $w, w' \in Actt^*$ such that we can obtain w from w' by removing som t actions from w'. We say that w is time subsequence of W, denoted $w \prec_t w'$. For example, we have $atbc \prec_t tattbtct$. We say that predicate ϕ over sequences from $Actt^*$ is time dependable if for every $w \in Actt^*$ such that $\phi(w)$ holds then there exists $w', w \prec_t w'$ such that $\neg\phi(w')$ holds.

Proposition 6. *Let P is prone to timming attack with respect to $\mathcal{O}$ and time dependable predicate ϕ. Let $\mathcal{O} \preceq \mathcal{O}_S$. Then there exist a supervisor for P with empty controlable set C.*

Proof. Let for every $w \in Tr(P) \setminus K^{P,\phi,\mathcal{O}}$ there exists $w', w \prec_t w'$ such that $\neg\phi(w')$ holds. Clearly, $w' \in Tr(P)$ (see transition rules A1, A2, Pa, S) and hence it is enough to insert corresponding t actions.

4.3 Observations, Predicate and Supervisor as Processes

In this subsection, we will model observation functions and predicates by process and then we show some decidability results for language opacity and supervisor existence.

Fig. 1. Supervisory Control

We use contexts to model communications between a process, observation function and supervisor. By context $\mathcal{C}$ we mean a process term with placeholders $\mathcal{H}$. Formally, the set of TPA simple contexts is defined by the following BNF notation:

$$\mathcal{C} \quad ::= \quad \mathcal{H}_i \mid op(\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_n)$$

where $\mathcal{C}, \mathcal{C}_1, \ldots \mathcal{C}_n$ are TPA contexts, $op \in \{[S], \backslash, |\}$ (i.e. operations relabelling, restriction and parallel composition) and $\mathcal{H}$ is the place holder. By $\mathcal{C}(P)$ we denote process obtained from process simple context $\mathcal{C}$ and process P by substituting P by place holders $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$, i.e. $\mathcal{C}(P, O, Sup) = \mathcal{C}[P/\mathcal{H}_1, O/\mathcal{H}_2, Sup/\mathcal{H}_3]$. We require that $Tr(\mathcal{C}(P, \mathcal{O}_S, Sup)) \subseteq K^{P,\phi,\mathcal{O}}Tr(P)$.

Proposition 7. *Let Sup, Sup' are processes coressponding to supervisors and let $Sup \sim Sup'$. Then $Tr(\mathcal{C}(P, \mathcal{O}_S, Sup)) = Tr(\mathcal{C}(P, \mathcal{O}_S, Sup'))$.*

Proof. The proof follows from the fact that bisimulation $\sim$ is the congruence relation and it is stronger than the trace equivalence.

Corollary For any supervisor Sup, if $Sup \sim Q$ then also Q is a supervisor.
  To obtain a decidable variant of the previous Proposition 4, we insert some restrictions on trace predicates. First, we model predicates by special processes called tests. The tests communicate with processes's trace and produce $\sqrt{}$ action if corresponding predicates hold for the trace. In the subsequent proposition, we show how to exploit this idea for process opacity.

Definition 12. *We say that the process T_ϕ is the test representing the predicate ϕ if $\phi(w)$ holds iff $(w.Nil|T_\phi) \setminus At \approx_t \sqrt{}.Nil$ where $\sqrt{}$ is a new*

action indicating a passing of the test. If T_ϕ is the finite state process we say that ϕ is the finitely definable predicate.

Suppose that both ϕ and $\neg\phi$ are the finitely definable predicates. Then we can reduce checking whether Sup is a process supervisor for language opacity to checking bisimulation. Since we can reduce the problem of decidability to finite automata (see [25]) we obtain the following result.

Proposition 8. *Let ϕ and $\neg\phi$ are finitely definable predicates and $\mathcal{O}$, $\mathcal{O}_S$ are static. The property that Sup is a process supervisor for language opacity for finite state process P and is decidable. Moreover, we can always find a maximal supervisor for language opacity.*

Moreover, for static observation functions $\mathcal{O}$, $\mathcal{O}_S$ and ϕ and $\neg\phi$ finitely definable predicates there exists finite state maximal process supervisor for language opacity for any finite state process P. This follows from the fact that such observation function can be emulated by finite-state processes since only finite memory is required.

Proposition 9. *Let ϕ and $\neg\phi$ are finitely definable predicates and $\mathcal{O}$, $\mathcal{O}_S$ are static. Then for any finite-state process P there exists finite-state process Sup which is the maximal supervisor for corresponding language opacity.*

Note that the above-mentioned properties can be directly extended to m-orwellian observation functions. As regards dynamic and orwellian observation functions it is more complex and we will leave it to future research. It is well known that both trace equivalence and bisimulation are congruences (see [21]) and we can replace in process $\mathcal{C}(P, \mathcal{O}_S, Sup))$ any of them by equivalent one without changing its functionality, i.e. $Tr(\mathcal{C}(P, \mathcal{O}'_S, Sup')) \subseteq K^{P,\phi,\mathcal{O}}Tr(P)$ will still hold.

5   Conclusions

We have investigated time-inserting supervisors for timed process algebra, which enforces the security concerning L-timing attacks. Time-inserting supervisors can disable some actions as well as add some delays to the system's behavior, which is particularly useful to prevent timing attacks. We study the existence of time-inserting supervisors for a given process, given observation functions, and a predicate over process's traces.

In future work, we plan to investigate minimal time inserting supervisors, i.e. such functions that add as little as possible time delays to

guarantee process security with respect to language opacity. The presented approach allows us to exploit also process algebras enriched by operators expressing other "parameters" (space, distribution, networking architecture, power consumption, and so on). Hence, we could obtain security properties that have not only theoretical but also practical value. Moreover, we can use similar techniques as in [18] to minimize time, as well as other resources, added to the process's behavior.

References

1. Joseph Bonneau and Ilya Mironov. Cache-collision timing attacks against aes. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 201–215, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
2. Jeremy Bryans, Maciej Koutny, Laurent Mazare, and Peter Ryan. Opacity generalised to transition systems. volume 7, pages 421–435, 11 2008.
3. Jeremy W. Bryans, Maciej Koutny, and Peter Y.A. Ryan. Modelling opacity using petri nets. *Electronic Notes in Theoretical Computer Science*, 121:101–115, 2005. Proceedings of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004).
4. Jean-Franois Dhem, Francois Koeune, Philippe-Alexandre Leroux, Patrick Mestr, Jean-Jacques Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. volume 1820, pages 167–182, 09 1998.
5. R. Focardi, R. Gorrieri, and F. Martinelli. Information flow analysis in a discrete-time process algebra. In *Proceedings 13th IEEE Computer Security Foundations Workshop. CSFW-13*, pages 170–184, 2000.
6. Riccardo Focardi, Roberto Gorrieri, Ruggero Lanotte, Andrea Maggiolo-Schettini, Fabio Martinelli, Simone Tini, and Enrico Tronci. Formal models of timing attacks on web privacy. *Electron. Notes Theor. Comput. Sci.*, 62:229–243, 2001.
7. Kuma Fuchiwaki and Kai Cai. Marking data-informativity and data-driven supervisory control of discrete-event systems. *Proceedings of the Japan Joint Automatic Control Conference*, 67:102–107, 2024.
8. J. A. Goguen and J. Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy*, pages 11–11, 1982.
9. Roberto Gorrieri and Fabio Martinelli. A simple framework for real-time cryptographic protocol analysis with compositional proof rules. *Science of Computer Programming*, 50:23–49, 03 2004.
10. Damas P. Gruska. Process opacity for timed process algebra. In Andrei Voronkov and Irina B. Virbitskaite, editors, *Perspectives of System Informatics - 9th International Ershov Informatics Conference, PSI 2014, St. Petersburg, Russia, June 24-27, 2014. Revised Selected Papers*, volume 8974 of *Lecture Notes in Computer Science*, pages 151–160. Springer, 2014.
11. Damas P. Gruska. Dynamics security policies and process opacity for timed process algebras. In Manuel Mazzara and Andrei Voronkov, editors, *Perspectives of System Informatics - 10th International Andrei Ershov Informatics Conference, PSI 2015, in Memory of Helmut Veith, Kazan and Innopolis, Russia, August 24-27, 2015, Revised Selected Papers*, volume 9609 of *Lecture Notes in Computer Science*, pages 149–157. Springer, 2015.

12. Damas P. Gruska. Security and time insertion. In Yannis Manolopoulos, George Angelos Papadopoulos, Athena Stassopoulou, Ioanna Dionysiou, Ioannis Kyriakides, and Nicolas Tsapatsoulis, editors, *Proceedings of the 23rd Pan-Hellenic Conference on Informatics, PCI 2019, Nicosia, Cyprus, November 28-30, 2019*, pages 154–157. ACM, 2019.

13. Damas P. Gruska. Time insertion functions. In Ladjel Bellatreche, George A. Chernishev, Antonio Corral, Samir Ouchani, and Jüri Vain, editors, *Advances in Model and Data Engineering in the Digitalization Era - MEDI 2021 International Workshops: DETECT, SIAS, CSMML, BIOC, HEDA, Tallinn, Estonia, June 21-23, 2021, Proceedings*, volume 1481 of *Communications in Computer and Information Science*, pages 181–188. Springer, 2021.

14. Damas P. Gruska and M. Carmen Ruiz. Opacity-enforcing for process algebras. In Bernd-Holger Schlingloff and Samira Akili, editors, *Proceedings of the 27th International Workshop on Concurrency, Specification and Programming, Berlin, Germany, September 24-26, 2018*, volume 2240 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.

15. Helena Handschuh and Howard M. Heys. A timing attack on rc5. In *Proceedings of the Selected Areas in Cryptography*, SAC '98, page 306318, Berlin, Heidelberg, 1998. Springer-Verlag.

16. K. Hernndez-Rueda, M.E. Meda-Campaa, and J. Armburo-Lizrraga. Enforcing diagnosability in interpreted petri nets. *IFAC-PapersOnLine*, 48(7):58–63, 2015. 5th IFAC International Workshop on Dependable Control of Discrete Systems.

17. Alejandro Hevia and Marcos Kiwi. Strength of two data encryption standard implementations under timing attacks. *ACM Trans. Inf. Syst. Secur.*, 2(4):416–437, November 1999.

18. Yiding Ji, Xiang Yin, and Stphane Lafortune. Enforcing opacity by insertion functions under multiple energy constraints. *Automatica*, 108:108476, 2019.

19. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

20. Francois Koeune, Francois Koeune, Jean-Jacques Quisquater, and Jean jacques Quisquater. A timing attack against rijndael. Technical report, 1999.

21. R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., USA, 1989.

22. P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

23. Aida Rashidinejad, Michel Reniers, and Martin Fabian. Supervisory control synthesis of timed automata using forcible events. 2021.

24. Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM'01, USA, 2001. USENIX Association.

25. Yin Tong, Zhiwu Li, Carla Seatzu, and Alessandro Giua. Current-state opacity enforcement in discrete event systems under incomparable observations. *Discret. Event Dyn. Syst.*, 28(2):161–182, 2018.

26. Walter Wonham. *On the control of discrete-event systems*, volume 135, pages 542–562. 01 1970.

27. Gang Xie, Yin Tong, Xiaomin Wang, and Carla Seatzu. Resilient supervisor synthesis for labeled petri nets against sensor attacks. *IEEE Transactions on Automatic Control*, pages 1–8, 2025.