Opacity-enforcing for Process Algebras [⋆]

Damas P. Gruska[1] and M. Carmen Ruiz[2]

[1] Comenius University, Slovakia
[2] Universidad de Castilla-La Mancha, Spain

Abstract. Supervisory control as a way how to guarantee security of processes is discussed and studied. We work with a security property called processes opacity and we investigate how it can be enforced. Supervisors can restrict behaviour of the original systems by enabling or disabling some actions to guarantee its security. We study maximal supervisors as the least restricting supervisory control processes. Moreover, we study also enhanced supervisory control which can add idling between system's action to prevent timing attacks.

Keywords: security, opacity, process algebras, information flow, supervisory control

1 Introduction

The great revolution brought about by the internet of things involves the emergence of new devices, new protocols and, of course, new security needs to fulfill the new requirements. New protocols come into operation before they have been evaluated in depth. This leads to the appearance of new versions of the protocol that is not always compatible with its predecessors and that companies will not always incorporate in their devices with sufficient speed. In addition, these solutions usually require downloading a new code and this itself is open to security attacks. This lack of security has been detected even in our own works. For example in [Gar16] we present an architecture for Wireless Sensor and Actuator Networks (WSAN) using the Bluetooth Low Energy (BLE) and TCP/IP protocols in conjunction, which make necessary to include bridges that lack basic security requirements. Another example can be found in [Hor17] where we propose a new packet format and a new BLE mesh topology, with two different configurations: Individual Mesh and Collaborative Mesh. All these represent our motivation to study applicability of formal models and formal methods to define and enforce system's security. As regards formalism, we will work with timed process algebra. Then we exploit information flow based security properties (see [GM82]) which assume an absence of any information flow between private and public systems activities. This means that systems are considered to be secure if from observations of their public activities no information about private activities or states can be deduced. This approach has found many reformulations and among them opacity (see [BKR04,BKMR06]) could be considered as the

⋆ Work supported by the grant VEGA 1/0778/18.

most general one and many other security properties could be viewed as its special cases (see, for example, [Gru15,Gru12,Gru11,Gru10,Gru08,Gru07]). Opacity properties could be divided into two types: language based opacity, expressing security (privacy) of system's actions or traces of actions and state based one, concentrating on system's states (see an overview paper [JLF16]). The former one is much more studied for process algebra's formalism. But also for the later one there is some research already done. In [Gru15] we consider an intruder who wants to discover whether a process reaches a confident state. Resulting security property is called process opacity. It turned out that in this way some new security flaws could be expressed. If a process is not secure with respect to process opacity we can either re-design its behavior, what 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 [RW89]) to restrict system's behaviour in such a way that the system becomes secure. A supervisor can see (some) system's actions and can control (disable or enable) some set of system's action. In this way it restricts system's behaviour to guarantee its security. This is a trade-off between security and functionality. But in many cases it is not a fatal problem. Suppose that a communication protocol can reach (with a low probability) a state which is not secure. In that case the transmission of a packet is interrupted and it should start from the begging. Sometimes this restriction has even smaller impact on system's behavior. Suppose that the system can perform action a and b in an arbitrary order but only a sequence $b.a$ could leak some classified information about intermediate states. Restricting this sequence make system secure but could not have influence on overall system's functionality. In this paper we do not assume any relation among a set of actions visible for an intruder, a set of actions visible for a controller and a set of controllable actions, i.e. sets E_I, E_S, E_C, respectively, similarly to [TLSG18]. Note that in [DDM10] it is assumed that $E_I \subseteq E_S$ (or $E_S \subseteq E_I$) and $E_C \subseteq E_S$. In [YL10] $E_I \subseteq E_S$ is assumed and in [TLSG16] $E_C \subseteq E_S$ is assumed. As regards the related work, besides already mentioned works there is a large body of work on controller synthesis in temporal model checking. From the rest we mention just two papers. In [RS01] the idea of controller to enforce secure information flow is discussed for language based security in process algebra setting. Opacity-enforcing (called strategic noninterference) was proposed and investigated for transition systems in [JT15].

Timing attacks, as side channel attacks, represent serious threat for many systems. They allow intruders "break" "unbreakable" systems, algorithms, protocols, etc. Even relatively recently discovered possible attacks on most of currently used processors (Meltdown and Spectre) also belong to timing attacks. To protect systems against some type of timing attacks we propose to enhance capabilities of the supervisory control. Such controller can add some idling between actions to enforce process's security.

The paper is organized as follows. In Section 2 we describe the timed process algebra TPA which will be used as a basic formalism. In Section 3 we present supervisory control. The next section contains some basic definition on information flow security and process opacity. Sections 5 and 6 deals with supervisory and enhanced supervisory control for process opacity, respectively.

2 Timed Process Algebra

In this section we define Timed Process Algebra, TPA for short. TPA is based on Milner's CCS but the special time action t which expresses elapsing of (discrete) time is added. The presented language is a slight simplification of Timed Security Process Algebra introduced in [FGM00]. 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 since it is not contained 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 [GM04]). TPA does not use value passing and strictly preserves *time determinancy* in case of choice operator $+$ what is not the case of tCryptoSPA.

To define the language TPA, we first assume a set of atomic action symbols A not containing symbols τ and t, and such that for every $a \in A$ there exists $\overline{a} \in A$ and $\overline{\overline{a}} = a$. We define $Act = A \cup \{\tau\}, At = A \cup \{t\}, Actt = Act \cup \{t\}$. We assume that $a, b, \ldots$ range over A, $u, v, \ldots$ range over Act, and $x, y \ldots$ range over $Actt$. Assume the signature $\Sigma = \bigcup_{n \in \{0,1,2\}} \Sigma_n$, where

$$\begin{aligned}
\Sigma_0 &= \{Nil\} \\
\Sigma_1 &= \{x. \mid x \in A \cup \{t\}\} \cup \{[S] \mid S \text{ is a relabeling function}\} \\
&\quad \cup \{\backslash M \mid M \subseteq A\} \\
\Sigma_2 &= \{|, +\}
\end{aligned}$$

with the agreement to write unary action operators in prefix form, the unary operators $[S], \backslash M$ in postfix form, and the rest of operators in infix form. Relabeling functions, $S : Actt \to Actt$ are such that $\overline{S(a)} = S(\bar{a})$ for $a \in A, S(\tau) = \tau$ and $S(t) = t$.

The set of TPA terms over the signature Σ is defined by the following BNF notation:

$$P \ ::= \ X \ \mid \ op(P_1, P_2, \ldots P_n) \ \mid \ \mu X P$$

where $X \in Var$, Var is a set of process variables, $P, P_1, \ldots P_n$ are TPA terms, $\mu X-$ is the binding construct, $op \in \Sigma$.

The set of CCS terms consists of TPA terms without t action. We will use an usual definition of opened and closed terms where μX is the only binding operator. Closed terms which are t-guarded (each occurrence of X is within some subterm $t.A$ i.e. between any two t actions only finitely many non timed actions can be performed) are called TPA processes.

We give a structural operational semantics of terms 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 TPA $\times$ $Actt$ $\times$ TPA. We write $P \xrightarrow{x} P'$ instead of $(P, x, P') \in \rightarrow$ and $P \overset{x}{\not\rightarrow}$ if there is no P' such that $P \xrightarrow{x} P'$. The meaning of the expression $P \xrightarrow{x} P'$ is that the term P can evolve to P' by performing action x, by $P \xrightarrow{x}$ we will denote that there exists a term P' such that $P \xrightarrow{x} P'$. We define the transition relation as the least relation satisfying the inference rules for CCS plus the following inference rules:

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

$$\frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q', P \mid Q \overset{\tau}{\not\rightarrow}}{P \mid Q \xrightarrow{t} P' \mid Q'}\ Pa \qquad \frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q'}{P + Q \xrightarrow{t} P' + Q'}\ S$$

Here we mention the rules that are new with respect to CCS. Axioms $A1, A2$ allow arbitrary idling. Concurrent processes can idle only if there is no possibility of an internal communication (Pa). A run of time is deterministic (S) i.e. performing of t action does not lead to the choice between summands of $+$. In the definition of the labeled transition system we have used negative premises (see Pa). In general this may lead to problems, for example with consistency of the defined system. We avoid these dangers by making derivations of τ independent of derivations of t. For an explanation and details see [Gro90].

For $s = x_1.x_2.\ldots.x_n, x_i \in Actt$ 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 will denote the empty sequence of actions, by $Succ(P)$ we will denote the set of all successors of P i.e. $Succ(P) = \{P' | P \xrightarrow{s} P', s \in Actt^*\}$. If the set $Succ(P)$ is finite we say that P is a finite state process. We define modified transitions $\overset{x}{\Rightarrow}_M$ which "hide" actions from M. Formally, we will write $P \overset{x}{\Rightarrow}_M P'$ for $M \subseteq Actt$ iff $P \xrightarrow{s_1}\xrightarrow{x}\xrightarrow{s_2} P'$ for $s_1, s_2 \in M^\star$ and $P \overset{s}{\Rightarrow}_M$ instead of $P \overset{x_1}{\Rightarrow}_M \overset{x_2}{\Rightarrow}_M \ldots \overset{x_n}{\Rightarrow}_M$. We will write $P \overset{x}{\Rightarrow}_M$ if there exists P' such that $P \overset{x}{\Rightarrow}_M P'$. We will write $P \overset{\widehat{x}}{\Rightarrow}_M P'$ instead of $P \overset{\epsilon}{\Rightarrow}_M P'$ if $x \in M$. Note that $\overset{x}{\Rightarrow}_M$ is defined for arbitrary action x but in definitions of security properties we will use it for actions (or sequence of actions) not belonging to M. We can extend the definition of $\Rightarrow_M$ for sequences of actions similarly to $\xrightarrow{s}$. Let $s \in Actt^\star$. By $|s|$ we will denote the length of s i.e. a number of action contained in s. By $s|_B$ we will denote the sequence obtained from s by removing all actions not belonging to B. For example, $|s|_{\{t\}}|$ denote a number of occurrences of t in s, i.e. time length of s. By $Sort(P)$ we will denote the set of actions from A which can be performed by P. The set of traces of process P is defined as $L(P) = \{s \in Actt^\star | \exists P'.P \overset{s}{\Rightarrow} P'\}$. The set of weak timed traces of process P is defined as $L_w(P) = \{s \in (A \cup \{t\})^\star | \exists P'.P \overset{s}{\Rightarrow}_{\{\tau\}} P'\}$. Two processes P and Q are weakly timed trace equivalent ($P \simeq_w Q$) iff $L_w(P) = L_w(Q)$. We conclude this section with definitions of M-bisimulation and weak timed trace equivalence.

Definition 1. *Let* $(TPA, Actt, \rightarrow)$ *be a labelled transition system (LTS). A relation* $\Re \subseteq TPA \times TPA$ *is called a* M-bisimulation *if it is symmetric and it satisfies the following condition: if* $(P, Q) \in \Re$ *and* $P \xrightarrow{x} P', x \in Actt$ *then there exists a process* Q' *such that* $Q \overset{\widehat{x}}{\Rightarrow}_M Q'$ *and* $(P', Q') \in \Re$*. Two processes* P, Q *are M-bisimilar, abbreviated* $P \approx_M Q$*, if there exists a M-bisimulation relating* P *and* Q*.*

3 Supervisory control

In this section we introduce some basic concepts of supervisory control theory. For more details see [RW89]. Let us assume deterministic finite automaton (DFA) $G = (X, E, \delta, x_0)$, where X is the finite set of states, E is the set of events, $\delta : X \times E \rightarrow X$ is the (partial) transition function, $x_0 \in X$ is the initial state. The transition function can be naturally extended to strings of events. The generated language of $G = (X, E, \delta, x_0)$ is defined as $L(G) = \{s, s \in E^*$ such that $\delta(x_0, s)$ is defined$\}$.

The goal of supervisory control is to design a control agent (called supervisor) that restricts the behavior of the system within a specification language $K \subseteq L(G)$. The supervisor observes a set of observable events $E_S \subseteq E$ and is able to control a set of controllable events $E_C \subseteq E$. The supervisor enables or disables controllable events. When an event is enabled (resp., disabled) by the supervisor, all transitions labeled by the event are allowed to occur (resp., prevented from occurring). After the supervisor observes a string generated by the system it tells the system the set of events that are enabled next to ensure that the system will not violate the specification.

A supervisor can be represented by $Sup = (Y, E_S, \delta_s, y_0, \Psi)$, where (Y, E_S, δ_s, y_0) is an automaton and $\Psi : Y \rightarrow \{E' \subseteq E | E_{UC} \subseteq E'\}$ where $E_{UC} = E \setminus E_C$ specifies the set of events enabled by the supervisor in each state. System G under the control of a suitable supervisor Sup is denoted as Sup/G, and it satisfies $L(Sup/G) \subseteq K$.

Definition 2 (Controllability). *Given a DFAG, a set of controllable events* E_C*, and a language* $K \subseteq L(G)$*,* K *is said to be controllable (wrt* $L(G)$ *and* E_C*) if*

$$\bar{K}E_{UC} \cap L(G) \subset \bar{K}$$

where $\bar{K}$ *is the prefix closer of* K*.*

The controllability of K requires that for any prefix $s, s \in K$, if s followed by an uncontrollable event $e \in E_{UC}$ is in $L(G)$, then it must also be a prefix of a string in K.

Definition 3 (Observability). *Given a DFAG, a set of controllable events* E_C *, a set of observable events* E_S *, and a language* $K \subseteq L(G)$*,* K *is said to be observable (wrt* $L(G)$*,* E_S *and* E_C*) if for all* $s, s' \in \bar{K}$ *and all* $e \in E_C$ *such that* $se \in L(G)$*,* $s =_S s'$ *(*$s =_S s'$ *is means that strings are equal with respect to the set* E_S*),* $s.e \in \bar{K}$*.*

Observability requires that supervisors observation of the system (i.e., the projection of s on E_S) provides sufficient information to decide after the occurrence of a controllable event whether the resultant string is still in $\bar{K}$.

Proposition 1. *Let $K \subseteq L(G)$ be a prefix-closed nonempty language, E_C the set of controllable events and E_S the set of observable events. There exists a supervisor Sup such that $L(Sup/G) = K$ if and only if K is controllable and observable.*

4 Information flow

In this section we will present our working security concept. First we define the absence-of-information-flow property - Strong Nondeterministic Non-Interference (SNNI, for short, see [FGM00]). 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 $L \cup H = A$. SNNI 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. Namely, process P has SNNI property (we will write $P \in SNNI$) if $P \setminus H$ behaves like P for which all high level actions are hidden (namely, replaced by action τ) for an observer. To express this hiding we introduce the hiding operator $P/M, M \subseteq A$, for which it holds that if $P \xrightarrow{a} P'$ then $P/M \xrightarrow{a} P'/M$ whenever $a \notin M \cup \overline{M}$ and $P/M \xrightarrow{\tau} P'/M$ whenever $a \in M \cup \overline{M}$. Formally, we say that P has SNNI property, and we write $P \in SNNI$ iff $P \setminus H \simeq_w P/H$. A generalization of this concept is given by opacity (this concept was exploited in [BKR04], [BKMR06] and [Gru07] in a framework of Petri Nets, transition systems and process algebras, respectively). Actions are not divided into public and private ones at the system description level but a more general concept of observations and predicates are exploited. 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 the 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. Now let us assume a different scenario, namely that an intruder is not interested in traces and their properties but he or she tries to discover whether a given process always reaches a state with some given property which is expressed by a (total) predicate. This property might be process deadlock, capability to execute only traces with time length less then n time unites, capability to perform at the same time actions form a given set, incapacity to idle (to perform t action) etc. We do not put any restriction on such predicates but we only assume that they are consistent with some suitable behavioral equivalence. The formal definition follows.

Definition 4. *We say that the predicate ϕ over processes is consistent with respect to relation $\cong$ if whenever $P \cong P'$ then $\phi(P) \Leftrightarrow \phi(P')$.*

As the consistency relation $\cong$ we could take bisimulation ($\approx_\emptyset$), weak bisimulation ($\approx_{\{\tau\}}$) or any other suitable equivalence. A special class of such predicates are such ones (denoted as $\phi^Q_\cong$) which are defined by a given process Q and equivalence relation $\cong$ i.e. $\phi^Q_\cong(P)$ holds iff $P \cong Q$.

We suppose that the intruder can observe only some activities performed by the process. Hence we suppose that there is a set of public actions which can be observed and a set of hidden (not necessarily private) actions. To model such observations we exploit the relation $\overset{s}{\Rightarrow}_M$ where actions from M are those ones which could not be seen by the observer. The formal definition of process opacity (see [Gru15]) is the following.

Definition 5 (Process Opacity). *Given process P, a predicate ϕ over processes is process opaque w.r.t. the set M if whenever $P \overset{s}{\Rightarrow}_M P'$ for $s \in (Actt \setminus M)^*$ and $\phi(P')$ holds then there exists P'' such that $P \overset{s}{\Rightarrow}_M P''$ and $\neg\phi(P'')$ holds. The set of processes for which the predicate ϕ is process opaque w.r.t. to the M will be denoted by POp^ϕ_M.*

Note that if $P \cong P'$ then $P \in POp^\phi_M \Leftrightarrow P' \in POp^\phi_M$ whenever ϕ is consistent with respect to $\cong$ and $\cong$ is such that it is a subset of the trace equivalence (defined as $\simeq_w$ but instead of $\overset{s}{\Rightarrow}_{\{\tau\}}$ we use $\overset{s}{\Rightarrow}_\emptyset$).

$$\begin{array}{lcl} P & \overset{s}{\Longrightarrow}_M & \phi(P') \\ \\ P & \overset{s}{\Longrightarrow}_M & \neg\phi(P'') \end{array}$$

Fig. 1. Process opacity

5 Supervisory Control of Process Opacity

In this section we will concentrate on enforcing process opacity, namely, how to guarantee that there is no leakage of information on validity of ϕ in a current state, i.e. security with respect to process opacity. Let $M \subseteq Actt$ by $\bar{M}$ we will denote the complement of M i.e. $\bar{M} = Actt \setminus M$. Let $s \in Actt^*$, by $s_{\bar{M}}$ we denote the string obtained form s by removing all elements belonging to M. Formally, $\epsilon_{\bar{M}} = \epsilon$, $s.x_{\bar{M}} = s.x$ iff $x \notin M$ and $s.x_{\bar{M}} = s$ iff $x \in M$. We can extend this definition to a set of strings. Let $T \subseteq Actt^*$ then $T_{\bar{M}} = \{s_{\bar{M}} | s \in T\}$.

Now let as suppose that process P is not secure with respect to process opacity POp^ϕ_M i.e. $P \notin POp^\phi_M$. That means that there exists $s \in L(P)_{\bar{M}}$ such that $P \overset{s}{\Rightarrow}_M P'$ and $\phi(P')$ holds then there does not exist P'' such that $P \overset{s}{\Rightarrow}_M P''$ and $\neg\phi(P'')$ holds. Hence, by observing s, an intruder knows that a state satisfying ϕ has been reached. For security reasons we want to prohibit such computations what will be the role for the supervisory control. Formally, let $K, K \subseteq L(P)_{\bar{M}}$ is a set of safe observations, i.e. for every $s \in K$, $P \overset{s}{\Rightarrow}_M P'$ and $\phi(P')$ does not hold or if it holds then there exists P'' such that $P \overset{s}{\Rightarrow}_M P''$ and

$\neg\phi(P'')$ holds. Clearly, if $P \in POp_M^\phi$ then $K = L(P)_{\bar{M}}$, otherwise $K \subset L(P)_{\bar{M}}$ but $K \neq L(P)_{\bar{M}}$. The aim of the control is to design a supervisor *Sup* which will restrict behaviour of the original process P in such a way that for the resulting process Sup/P we have $L(Sup/P) \subseteq K$. Note that we do not assume any relations among set of actions visible for an intruder, a set of actions visible for a controller and a set of controllable actions, i.e. sets $E_I (E_I = \bar{M}), E_S, E_C$, respectively, similarly to [TLSG18]. Note that in [DDM10] it is assumed that $E_I \subseteq E_S$ (or $E_S \subseteq E_I$) and $E_C \subseteq E_S$. In [YL10] $E_I \subseteq E_S$ is assumed and in [TLSG16] $E_C \subseteq E_S$ is assumed.

Example 1. Let $P = c.(a.b.Nil + b.(a.Nil + d.Nil))$, $M = \{c, d\}$ and predicate ϕ is defined as follows: $\phi(Q)$ holds iff $Q \xrightarrow{d}$. Then it is easy to check that $P \notin POp_M^\phi$. The execution of $c.b$ (visible as b) at the beginning leads to the state satisfying ϕ but no execution visible as b can lead to a state not satisfying ϕ.

Now we will model supervisory control by means a special process *Sup*. Process *Sup* runs in parallel with P, communicates with environment via actions $Sort(P)$ and internally with P by actions renamed by function f which maps every action a from $Sort(P)$ to a new "ghost" action a' (see Fig. 2). The formal definition of process supervisor is the following.

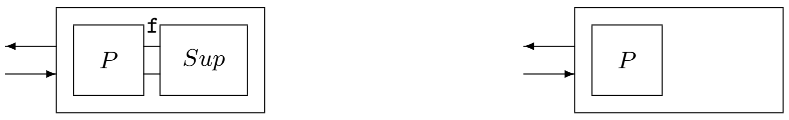

Fig. 2. Supervisory Control

Definition 6 (Process Supervisor). *Given process P, a process Sup is called supervisor if $L_w(Sup/P) \subseteq L_w(P)$ where $Sup/P = (P[f]|Sup) \setminus f(Sort(P))$ where $f : Sort(P) \to Sort(P)'$ where $Sort(P)' = \{x' | x \in Sort(P), x \neq \tau\}$.*

We will use a process supervisor to restrict behaviour of the original process in such a way that the resulting process becomes secure with respect to process opacity.

Definition 7 (Process Supervisor for Process Opacity). *Given process P, process Sup is called supervisor for opacity property POp_M^ϕ iff $P \notin POp_M^\phi$ but $Sup/P \in POp_M^\phi$. By $Sup(P, POp_M^\phi)$ we will denote the set of all supervisors for opacity property POp_M^ϕ for a given process P.*

Example 2. Let us continue with the Example 1. Let $Sup_1 = c'.c.Nil$ $Sup_2 = c'.c.a'.aNil$ and $Sup_3 = c'.c.a'.a.b'.b.Nil$ then it is easy to check that all processes Sup_i are process supervisors for P and opacity property POp_M^ϕ. Actually these process supervisors restrict the execution of action b immediately after c.

Clearly, $Sup(P, POp_M^\phi) \neq \emptyset$ since $Nil \in Sup(P, POp_M^\phi)$. Note that supervisor Nil restricts all behaviour of P which consequently becomes trivially secure. We can formulate some properties of the set $Sup(P, POp_M^\phi)$.

Proposition 2. *Let $Sup_1, Sup_2 \in Sup(P, POp^{\phi}_M)$. Then $Sup_1 + Sup_2 \in Sup(P, POp^{\phi}_M)$.*

Proof. The main idea. The first actions which is performed by $(Sup_1 + Sup_2)/P$ is performed either by Sup_1 or by Sup_2.

Proposition 3. *Let $Sup_1 \in Sup(P, POp^{\phi}_M)$ and $Sup_1 \approx_{\emptyset} Sup_2$. Then $Sup_2 \in Sup(P, POp^{\phi}_M)$.*

Proof. Sketch. The proof follows from the the fact that trace equivalence is congruence i.e. for $Sup_1 \approx_{\emptyset} Sup_2$ we have $(P[f]|Sup_1)\setminus f(Sort(P)) \approx_{\emptyset} (P[f]|Sup_1)\setminus f(Sort(P))$ and so $L((P[f]|Sup_1) \setminus f(Sort(P))) = L((P[f]|Sup_2) \setminus f(Sort(P)))$ i.e.$L(Sup_1/P) = L(Sup_2/P)$.

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

Definition 8 (Maximal Process Supervisor for Process Opacity). *Process $Sup \in Sup(P, POp^{\phi}_M)$ is called maximal process supervisor for process opacity POp^{ϕ}_M iff for every $Sup' \in Sup(P, POp^{\phi}_M)$ $L(Sup'/P) \subseteq L(Sup/P)$.*

Example 3. Let us continue with the Examples 1 and 2. It is easy to check that process Sup_3 is a maximal process supervisor for P and opacity property POp^{ϕ}_M. Processes Sup_1 and Sup_2 are not maximal process supervisors for P and opacity property POp^{ϕ}_M.

Unfortunately it is undecidable to verify whether a process Sup is a process supervisors for P and opacity property POp^{ϕ}_M as it is stated by the following proposition.

Proposition 4. *The property that Sup is a process supervisor for process opacity for process P is undecidable in general.*

Proof. The proof is based on an idea that already process opacity is undecidable (see Proposition 2. in [Gru15]). 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 process supervisor for process opacity for process P iff $P \in POp^{\phi}_M$. Hence we would be able to decide process opacity what contradicts its undecidability.

Corollary. The property that Sup is a maximal process supervisor for process opacity for process P is undecidable in general.

To obtain a decidable variant of the previous property we put some restriction on process predicates. First we model predicates by special processes called tests. For now we assume that action τ is not visible for an intruder, i.e. $\tau \in M$. The tests communicate with processes and produce $\surd$ action if corresponding predicates hold for the processes. In the subsequent proposition we show how to exploit this idea to express process opacity by means of appropriate M-bisimulation.

Definition 9. *We say that the process T_ϕ is the test representing predicate ϕ if $\phi(P)$ holds iff $(P|T_\phi)\setminus At \approx_t \surd.Nil$ where $\surd$ 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 process opacity to checking bisimulation (see Proposition 4. in [Gru15]). Since we can reduce the problem of decidability to finite automata (see [TLSG18]) we obtain the following result.

Proposition 5. *Let ϕ and $\neg\phi$ are finitely definable predicates. The property that Sup is a process supervisor for process opacity for finite state process P is decidable. Moreover, we can always find a maximal supervisor for process opacity.*

6 Enhanced Supervisory Control

Time attacks belong to powerful tools for attackers who can observe or interfere with systems in real time. By the presented formalism we can distinguish timing attacks. Suppose that $P \notin POp_M^\phi$ but $P \in POp_{M\cup\{t\}}^\phi$. This means that an attack is possible only for an observer who can see elapsing of time, i.e. there is a possibility of timing attacks. To prevent them, we can use process supervisor which restricts process behaviour with respect to actions from A or we introduce a new type of process supervisor, called *enhanced process supervisor* which can add some idling between actions to ensure that the resulting process becomes secure with respect to timing attacks. In this case the restriction with respect to atomic actions from A could be smaller as in the case of original supervisory control.

Definition 10 (Enhanced Process Supervisor). *Given process P, a process ESup is called enhanced supervisor if whenever $s \in L_w(P)$ then $ESup/P \overset{s}{\Rightarrow}_{t,\tau}$ where $ESup/P = (P[f]|ESup) \setminus f(Sort(P))$ where $f : Sort(P) \rightarrow Sort(P)'$ where $Sort(P)' = \{x'|x \in Sort(P), x \neq \tau\}$.*

Definition 11 (Enhanced Process Supervisor for Process Opacity). *Given process P, $P \in POp_{M\cup\{t\}}^\phi$, a process ESup is called enhanced supervisor for opacity property POp_M^ϕ iff $P \notin POp_M^\phi$, but $ESup/P \in POp_M^\phi$. By $ESup(P, POp_M^\phi)$ we will denote the set of all supervisors for opacity property $EPOp_M^\phi$ for a given process P.*

Now we can formulate a condition which guarantees existence of an enhanced supervisory control.

Proposition 6. *Let $P \notin POp_M^\phi$ and there exists P', $P \approx_{\tau,t} P'$ such that $P' \in POp_M^\phi$. Then there exists nontrivial (not equivalent to Nil with respect to $\approx_\tau$) enhanced supervisory control for P.*

Proof. The main idea. According to the assumption processes P and P' behave essentially in the same way but the later performs more idling between actions from Act. The enhanced supervisory control adds this idling to behaviour of P in such a way that the resulting process is process opaque.

Moreover, the previous proposition has the following consequence which guaranties that the maximal enhanced supervisory control does not restrict action from A.

Corollary. Let $P \in POp^{\phi}_{M\cup\{t\}}$ and $P \notin POp^{\phi}_{M}$. For the maximal enhanced supervisory control for P we have $L(ESup/P)_{\bar{A}} = K_{\bar{A}}$.

7 Conclusions

We have presented the new concepts called supervisory and enhanced supervisory controller for process algebra which enforce the security property called process opacity. Particularly, we have investigated finite state systems and time sensitive observations. A supervisor can see (some) system's actions and can control (disable or enable) some set of system's action. In this way it restricts system's behaviour to guarantee its security. Sometimes either we simply cannot redesign original insecure system which could have, for example, hardware implementation or some small restriction of system's behaviour is not essential for overall system functionality. In the case of enhanced supervisory controller it can only add some idling between actions which would have no influence on system non-timing properties. The presented approach allows us to exploit also process algebras enriched by operators expressing other "parameters" (space, distribution, networking architecture, processor or power consumption and so on). In this way also other types of attacks, which exploit information flow through various covert channels, can be described and enforced. Hence we could obtain security properties which have not only theoretical but also practical value, since many protocols, particularly low level protocols for IoT, could be described by means of some process algebra formalism.

References

[BKR04] Bryans J., M. Koutny and P. Ryan: Modelling non-deducibility using Petri Nets. Proc. of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models, 2004.

[BKMR06] Bryans J., M. Koutny, L. Mazare and P. Ryan: Opacity Generalised to Transition Systems. In Proceedings of the Formal Aspects in Security and Trust, LNCS 3866, Springer, Berlin, 2006.

[DDM10] Dubreil J., P. Darondeau and H. Marchand: Supervisory control for opacity. IEEE Trans Autom Control 55(5), 2010.

[FGM00] Focardi, R., R. Gorrieri, and F. Martinelli: Information flow analysis in a discrete-time process algebra. Proc. 13^{th} Computer Security Foundation Workshop, IEEE Computer Society Press, 2000.

[Gar16] Garrido C., V. Lopez, T. Olivares and M. C. Ruiz: Architecture Proposal for Heterogeneous, BLE-Based Sensor and Actuator Networks for Easy Management of Smart Homes. 15th ACMIEEE International Conference on Information Processing in Sensor Networks (IPSN), 2016.

[GM04] Gorrieri R. and F. Martinelli: A simple framework for real-time cryptographic protocol analysis with compositional proof rules. Science of Computer Programming, Volume 50, Issues 13, 2004.

[GM82] Goguen J.A. and J. Meseguer: Security Policies and Security Models. Proc. of IEEE Symposium on Security and Privacy, 1982.

[Gro90] Groote, J. F.: Transition Systems Specification with Negative Premises. Proc. of CONCUR'90, Springer Verlag, Berlin, LNCS 458, 1990.

[Gru15] Gruska D.P.: Process Opacity for Timed Process Algebra. In Perspectives of System Informatics, LNCS 8974, 2015.

[Gru12] Gruska D.P.: Informational analysis of security and integrity. Fundamenta Informaticae, vol. 120, Numbers 3-4, 2012.

[Gru11] Gruska D.P.: Gained and Excluded Private Actions by Process Observations. Fundamenta Informaticae, Vol. 109, No. 3, 2011.

[Gru10] Gruska D.P.: Process Algebra Contexts and Security Properties. Fundamenta Informaticae, vol. 102, Number 1, 2010.

[Gru08] Gruska D.P.: Probabilistic Information Flow Security. Fundamenta Informaticae, vol. 85, Numbers 1-4, 2008.

[Gru07] Gruska D.P.: Observation Based System Security. Fundamenta Informaticae, vol. 79, Numbers 3-4, 2007.

[Hor17] Hortelano, D., T Olivares, M.C. Ruiz, M. Carmen, C. Garrido-Hidalgo and V. López: From Sensor Networks to Internet of Things. Bluetooth Low Energy, a Standard for This Evolution. Sensors, vol 17, 2017.

[JLF16] Jacob, R., J.-J.Lesage and J.-M. Faure: Overview of discrete event systems opacity: Models, validation, and quantification, Annual Reviews in Control Volume 41, 2016.

[JT15] Jamroga W. and M. Tabatabaei: Strategic Noninterference. ICT Systems Security and Privacy Protection, SEC 2015.

[RW89] Ramadge P.J.G, Wonham W.M.: The control of discrete event systems. Proc IEEE 77(1):8198, 1989.

[RS01] Ryan P. Y. A. and S. A. Schneider: Process Algebra and Non-Interference. Journal of Computer Security 9(1/2), 2001.

[TLSG18] Tong, Y, Z. Li, Zhiwu, C. Seatzu and A. Giua: Current-state opacity enforcement in discrete event systems under incomparable observations. Discrete Event Dynamic Systems, vol. 28, 2018.

[TLSG16] Tong, Y, Z. Li, Zhiwu, C. Seatzu and A. Giua: Supervisory enforcement of current-state opacity with uncomparable observations. In: Proceedings of the 13th International workshop on discrete event systems, 2016.

[YL10] Yin X. and S. Lafortune: A new approach for synthesizing opacity-enforcing supervisors for partially-observed discrete-event systems. In: Proceedings of the 2015 American control conference. IEEE, Chicago, 2015.