Synthesis of Sensor Deception Attacks at the Supervisory Layer of Cyber-Physical Systems

Rômulo Meira-Góes [a], Eunsuk Kang [b], Raymond Kwong [c], Stéphane Lafortune [a]

[a] *Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109, USA*

[b] *School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA*

[c] *Department of Electrical Engineering and Computer Science, University of Toronto, Toronto, ON M5S 3G4, CA*

Abstract

We study the security of Cyber-Physical Systems (CPS) in the context of the supervisory control layer. Specifically, we propose a general model of a CPS attacker in the framework of discrete event systems and investigate the problem of synthesizing an attack strategy for a given feedback control system. Our model captures a class of *deception attacks*, where the attacker has the ability to hijack a subset of sensor readings and mislead the supervisor, with the goal of inducing the system into an undesirable state. We utilize a game-like discrete transition structure, called *Insertion-Deletion Attack structure* (IDA), to capture the interaction between the supervisor and the environment (which includes the system and the attacker). We show how to use IDAs to synthesize three different types of successful stealthy attacks, i.e., attacks that avoid detection from the supervisor and cause damage to the system.

Key words: Discrete Event Systems; Supervisory Control; Cyber-Physical Systems; Cyber-Security; Deception Attacks.

1 Introduction

Cyber-Physical Systems (CPS) are characterized by the interaction of computational entities with physical processes. These systems are found in a broad spectrum of safety-critical applications, such as smart grids, process control systems, autonomous vehicles, medical devices, etc. In such applications, some undesired behavior could cause damage to the physical system itself or to people relying on the system. Undesired behavior of CPS may be caused by faulty behavior or by external attacks on the system. In fact, some attacks on CPS have already been reported, see, e.g., (Kerns *et al.* 2014, Checkoway *et al.* 2011), including the well-known StuxNet attack (Farwell and Rohozinski 2011).

For this reason, cyber-security of CPS became a subject of increasing attention in the control community, see, e.g., (Weerakkody *et al.* 2019). The work in (Cardenas *et al.* 2008, Teixeira *et al.* 2012) have classified different types of cyber attacks on control systems. These cyber attacks have assumed attackers that have some prior knowledge of the CPS. Our paper focuses on security issues that arise in the feedback control of CPS for a specific class of these cyber attacks called *sensor deception attacks*. In this class of attacks, the sensor readings are hijacked by an attacker that is assumed to have some prior knowledge of the control system. More specifically, we are concerned with the problem of synthesizing a *sensor deception attack* strategy at the supervisory layer of a given CPS (Amin *et al.* 2013), an attack strategy that manipulates the sensor measurements received by the controller/supervisor[1] in order to achieve the attacker's goals. Three different types of attacks are investigated, where each attack has different capabilities regarding manipulating the sensor measurements that are sent to the supervisor.

Given that we are investigating cyber-attacks at the supervisory layer of a CPS, we use the formalism of Discrete Event Systems (DES) to model both the behavior of the attacker as well as the behavior of CPS itself. In other words, we assume that a discrete abstraction of the underlying CPS has already been performed. This allows us to leverage the concepts and techniques of the theory of super-

⋆ This work was supported in part by the US National Science Foundation under grants CNS-1421122, CNS-1446298 and CNS-1738103.

Email addresses: romulo@umich.edu (Rômulo Meira-Góes), eskang@cmu.edu (Eunsuk Kang), kwong@control.utoronto.ca (Raymond Kwong), stephane@umich.edu (Stéphane Lafortune).

[1] Since our focus is the supervisory layer of the control system, we will use the term *supervisor* in the remainder of the paper.

visory control of DES. Several recent works have adopted similar approaches to study cyber-security issues in CPS; see, e.g., (Carvalho *et al.* 2018, Paoli *et al.* 2011, Thorsley and Teneketzis 2006, Wakaiki *et al.* 2018).

Previous works such as (Carvalho *et al.* 2018, Paoli *et al.* 2011, Thorsley and Teneketzis 2006) on intrusion detection and prevention of cyber-attacks using discrete event models were focused on modeling the attacker as faulty behavior. Their corresponding methodologies relied on fault diagnosis techniques.

Recently, (Su 2018) proposed a framework similar to the one adopted in our paper, where they formulated a model of bounded sensor deception attacks. Our approach is more general than the one in (Su 2018), since we do not impose a *normality* condition to create an attack strategy; this condition is imposed to obtain the so-called *supremal controllable and normal language* under the attack model. In addition to bounded sensor deception attacks, we consider two other attack models. An additional difference between this paper and the approach in (Su 2018) is the way the dynamical interaction between the attacker and the supervisor is captured. This will be revisited in Section 3.

In (Wakaiki *et al.* 2018), the authors presented a study of supervisory control of DES under attacks. They introduced a new notion of *observability* that captures the presence of an attacker. However, their study is focused on the supervisor's viewpoint and they do not develop a methodology to design attack strategies. They assume that the attack model is given and they develop their results based on that assumption. In that sense, the work (Wakaiki *et al.* 2018) is closer to robust supervisory control and it is complementary to our work.

Several prior works considered robust supervisory control under different notions of robustness (Alves *et al.* 2014, Lin 2014, Rohloff 2012, Xu and Kumar 2009, Yin 2016), but they did not study robustness against attacks. In the cyber-security literature, some works have been carried out in the context of discrete event models, especially regarding opacity and privacy or secrecy properties (Cassez *et al.* 2012, Lin 2011, Meira-Góes *et al.* 2019, Saboori and Hadjicostis 2007, Wu *et al.* 2018). These works are concerned with studying information release properties of the system, and they do not address the impact of an intruder over the physical parts of the system.

Our approach is based on a general model of sensor deception attacks at the supervisory layer of the control system of the CPS. In that context, we investigate the problem of synthesizing *successful stealthy sensor deception attacks*. We make the following assumptions about the attacker: (i) it has knowledge of both the system and its supervisor; and (ii) it has the ability to alter the sensor information that is received by the supervisor. The goal of the attacker is to induce the supervisor into allowing the system to reach a pre-specified unsafe state, thereby causing damage to the system. Throughout the paper, we discuss the reasoning and the impact of these assumptions.

In this paper, we study three specific types of attacks that are based on the interaction between the attacker and the controlled system. The methodology developed to synthesize these attacks is inspired by the work in (Wu *et al.* 2018, Yin and Lafortune 2016*a*, Yin and Lafortune 2016*b*). As in these works, we employ a discrete structure to model the game-like interaction between the supervisor and the environment (system and attacker in this paper). We call this structure an *Insertion-Deletion Attack* structure (or IDA). By construction, an IDA embeds all desired scenarios where the attacker modifies some subset of the sensor events without being noticed by the supervisor. Once constructed according to the classes of attacks under consideration, an IDA serves as the basis for solving the synthesis problem. In fact, this game-theoretical approach provides a structure for each attack class that incorporates *all successful stealthy attacks*. Different stealthy attack strategies can be extracted from this structure. This is a distinguishing feature of our work as compared to previous works mentioned above. By providing a general synthesis framework, our goal is to allow CPS engineers to detect and address potential vulnerabilities in their control systems.

The remainder of this paper is organized as follow. Section 2 introduces necessary background and some notations used throughout the paper. The attack model as well as the problem statement are formalized in Section 3. Section 4 describes the IDA structure and its properties. Section 5 introduces the AIDA (All Insertion-Deletion Attack structure) and provides a construction algorithm for it. Sections 6 and 7 introduce for each attack type their stealthy IDA structure, together with a simple synthesis algorithm to extract an attack function. Lastly, Section 8 presents concluding remarks. Preliminary and partial versions of some of the results in this work have appeared in (Meira-Góes *et al.* 2017). Due to space limitation, proofs are omitted in this final version and they are available in (Meira-Góes *et al.* 2020).

2 Supervisory control system model

We assume that a given CPS has been abstracted as a discrete transition system that is modeled as a finite-state automaton. A finite-state automaton G is defined as a tuple $G = (X, \Sigma, \delta, x_0)$, where: X is a finite set of states; Σ is a finite set of events; $\delta : X \times \Sigma \to X$ is a partial transition function; and $x_0 \in X$ is the initial state. The function δ is extended in the usual manner to domain $X \times \Sigma^*$. The language generated by G is defined as $\mathcal{L}(G) = \{s \in \Sigma^* | \delta(x_0, s)!\}$, where ! means "is defined".

In addition, $\Gamma_G(S)$ is defined as the set of active events at the subset of states $S \subseteq X$ of automaton G, given by:

$$\Gamma_G(S) := \{e \in \Sigma | (\exists u \in S) \text{ s.t. } \delta(u, e)!\} \qquad (1)$$

By a slight abuse of notation, we write $\Gamma_G(x) = \Gamma_G(\{x\})$ for $x \in X$.

Language $\mathcal{L}(G)$ is considered as the *uncontrolled* system behavior, since it includes all possible executions of G. The limited actuation capabilities of G are modeled by a partition in the event set $\Sigma = \Sigma_c \cup \Sigma_{uc}$, where Σ_{uc} is the set of uncontrollable events and Σ_c is the set of controllable events.

It is assumed that G is controlled by a supervisor S_P that dynamically enables and disables the controllable events such that it enforces some safety property on G. In the notation of the theory of supervisory control of DES initiated in (Ramadge and Wonham 1987), the resulting controlled behavior is a new DES denoted by S_P/G with the closed-loop language $\mathcal{L}(S_P/G)$ defined in the usual manner (Cassandras and Lafortune 2008). The set of admissible control decisions is defined as $\Gamma = \{\gamma \subseteq \Sigma \mid \Sigma_{uc} \subseteq \gamma\}$, where admissibility guarantees that a control decision never disables uncontrollable events.

In addition, due to the limited sensing capabilities of G, the event set is also partitioned into $\Sigma = \Sigma_o \cup \Sigma_{uo}$, where Σ_o is the set of observable events and Σ_{uo} is the set of unobservable events. Based on this second partition, the *projection* function $P_o : \Sigma^* \to \Sigma_o^*$ is defined as:

$$P_o(\epsilon) = \epsilon \text{ and } P_o(se) = \begin{cases} P_o(s)e & \text{if } e \in \Sigma_o \\ P_o(s) & \text{if } e \in \Sigma_{uo} \end{cases} \quad (2)$$

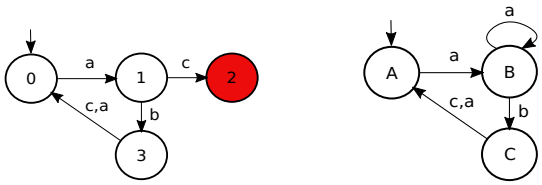
The inverse projection $P_o^{-1} : \Sigma_o^* \to 2^{\Sigma^*}$ is defined as $P_o^{-1}(t) = \{s \in \Sigma^* | P(s) = t\}$.

Formally, a partial observation supervisor is a function $S_P : P_o(\mathcal{L}(G)) \to \Gamma$. Without loss of generality, we assume that S_P is realized (i.e., encoded) as a deterministic automaton $R = (Q, \Sigma, \mu, q_0)$, such that, $\forall q \in Q$, if $e \in \Sigma_{uo}$ is an enabled unobservable event at state q by S_P, then we define $\mu(q, e) = q$ as is customary in a supervisor realization (cf. (Cassandras and Lafortune 2008)). In this manner, given a string $s \in \mathcal{L}(G)$ the control decision is defined as $S_P(s) = \Gamma_R(\mu(q_0, s))$.

Example 1 *Consider the system G represented in Fig. 1(a). Let $\Sigma = \Sigma_o = \{a, b, c\}$ and $\Sigma_c = \{b, c\}$. Figure 1(b) shows the realization R of a supervisor S_P that was designed for G. In this case, the language generated by $\mathcal{L}(S_P/G)$ guarantees that state 2 is unreachable in the controlled behavior.*

For convenience, we define two operators that are used in this paper together with some useful notation. The *unobservable reach* of the subset of states $S \subseteq X$ under the subset of events $\gamma \subseteq \Sigma$ is given by:

$$UR_\gamma(S) := \{x \in X | (\exists u \in S)(\exists s \in (\Sigma_{uo} \cap \gamma)^* \text{ s.t. } x = \delta(u, s)\} \quad (3)$$



(a) G (b) Supervisor R

Fig. 1. A system automaton along with its supervisor (Example 1)

The *observable reach* (or next states) of the subset of states $S \subseteq X$ given the execution of the observable event $e \in \Sigma_o$ is defined as:

$$NX_e(S) := \{x \in X | \exists u \in S \text{ s.t. } x = \delta(u, e)\} \quad (4)$$

For any string $s \in \Sigma^*$, let s^i denote the prefix of s with the first i events, and let e_s^i be the i^{th} event of s, so that $s^i = e_s^1 \dots e_s^i$; by convention, $s^0 = \epsilon$. We define $\bar{s}$ as the set of prefixes of string $s \in \Sigma^*$. Lastly, we denote by $\mathbb{N}$, $\mathbb{N}^+$, and $\mathbb{N}^n = \{0, \dots, n\}$ the sets of natural numbers, positive natural numbers, and natural numbers bounded by n, respectively.

3 The sensor deception attack problem

3.1 The general attack model

We start by defining the model for sensor deception attacks, as illustrated in Fig. 2. The attacker intervenes in the communication channels between the system's sensors and the supervisor. We assume that the attacker observes all events in Σ_o that are executed by the system. In addition, it has the ability to edit some of the sensor readings in these communication channels, by inserting fictitious events or deleting the legitimate events. The subset of sensor readings that can be edited is defined as the *compromised event set* and denoted by Σ_a. For generality purposes, we assume that $\Sigma_a \subseteq \Sigma_o$. To formally introduce the *attack function* shown

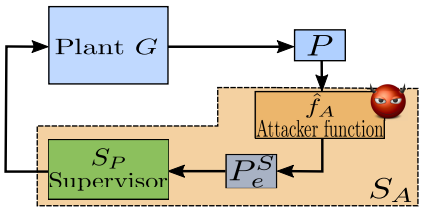


Fig. 2. Model for Sensor Deception Attacks

in Fig. 2, we first define two new sets of events. The sets $\Sigma_a^i = \{e_i | e \in \Sigma_a\}$ and $\Sigma_a^d = \{e_d | e \in \Sigma_a\}$ are defined as the sets of inserted and deleted events, respectively. These sets represent the actions of the attacker, and we use the subscripts to clearly distinguish them from events generated by G. Note that, Σ_a^i, Σ_a^d, and Σ_a are disjoint. For convenience, we define $\Sigma_a^e = \Sigma_a^i \cup \Sigma_a^d$ as the set of editable events. These events are used to identify the insertion or deletion of an event.

We also define the projections P_e^S, P_e^G, and the mask $\mathcal{M}_e$ to analyze strings consisting of events in $\Sigma_a^e \cup \Sigma$. P_e^S is a natural projection that treats editable events in the following manner: $P_e^S(e_i) = e$, $e_i \in \Sigma_a^i$; $P_e^S(e_d) = \epsilon$, $e_d \in \Sigma_a^d$, and $P_e^S(e) = e$, $e \in \Sigma$. We use the superscript S because P_e^S describes how the *supervisor* observes the modified events. Given an event, P_e^S outputs the legitimate event if the event was inserted by the attacker, it outputs the empty string if the event was deleted by the attacker, and it outputs the legitimate event otherwise. In addition, we define P_e^G to describe how the modified events interact with the system G: $P_e^G(e_i) = \epsilon$, $e_i \in \Sigma_a^i$; $P_e^G(e_d) = e$, $e_d \in \Sigma_a^d$, and $P_e^G(e) = e$, $e \in \Sigma$. Finally, $\mathcal{M}_e$ is a mask that removes the subscript $\{i, d\}$ if it exists: $\mathcal{M}_e(e_i) = \mathcal{M}_e(e_d) = e$, if $e_i, e_d \in \Sigma_a^e$, and $\mathcal{M}_e(e) = e$, $e \in \Sigma$.

Formally, we model an attacker as a nondeterministic string edit function. The nondeterminism model provides different class of attack strategies when compared to the deterministic model in (Meira-Góes *et al.* 2017).

Definition 2 *Given a system G and a subset $\Sigma_a \subseteq \Sigma_o$, an attacker is defined as a (potentially partial) function $f_A : (\Sigma_o \cup \Sigma_a^e)^* \times (\Sigma_o \cup \{\epsilon\}) \to 2^{(\Sigma_o \cup \Sigma_a^e)^*}$ s.t. f_A satisfies the following constraints:*

(1) $f_A(\epsilon, \epsilon) \subseteq {\Sigma_a^i}^*$ *and* $\forall s \in ((\Sigma_o \cup \Sigma_a^e)^* \backslash \{\epsilon\}) : f_A(s, \epsilon) = \emptyset$;

(2) $\forall s \in (\Sigma_o \cup \Sigma_a^e)^*$, $e \in \Sigma_o \backslash \Sigma_a : f_A(s, e) \subseteq \{e\}{\Sigma_a^i}^*$;

(3) $\forall s \in (\Sigma_o \cup \Sigma_a^e)^*$, $e \in \Sigma_a$: $f_A(s, e) \subseteq \{e,\ e_d\}{\Sigma_a^i}^*$.

The function f_A captures a general model of sensor deception attack. Given the past *edited* string s and observing a new event e executed by G, the attacker may choose to edit e based on Σ_a and replace e by selecting an edited suffix from the set $f_A(s, e)$. The first case in the above definition gives an initial condition for an attack. The second case constrains the attacker from erasing e when e is outside of Σ_a. However, the attacker may insert an arbitrary string $t \in {\Sigma_a^i}^*$ after the occurrence of e. Lastly, the third case in Definition 2 is for $e \in \Sigma_a$; the attacker can edit the event to any string in the set $\{e,\ e_d\}{\Sigma_a^i}^*$.

As mentioned before, f_A only defines the possible edited suffixes based on the last executed event and the edit history. It is interesting to define a function that defines the possible edited strings based on the executed string. Formally, the string-based edit (potentially partial) function $\hat{f}_A : \Sigma_o^* \to 2^{(\Sigma_o \cup \Sigma_a^e)^*}$ is defined as $\hat{f}_A(se) = \{ut \in (\Sigma_o \cup \Sigma_a^e)^* | u \in \hat{f}_A(s) \wedge t \in f_A(u, e)\}$ for any $s \in \Sigma_o^*$ and $e \in \Sigma_o$, and $\hat{f}_A(\epsilon) = f_A(\epsilon, \epsilon)$. The function $\hat{f}_A(s)$ returns the set of possible edited strings for a given string $s \in \Sigma_o^*$. Note that, in general, $\hat{f}_A$ is a partial function, and $\hat{f}_A(s)$ may only be defined for selected $s \in \Sigma_o^*$.

Remark: We have assumed that the attacker has the same observable capabilities as the supervisor. This assumption is accepted in papers, like ours, where the worst case analysis is performed. The problem where the attacker and the supervisor has incomparable observation is known to be hard. Nevertheless, in some cases the attacker might only have interception capabilities but full not transmission capabilities since it has not taken control of all the hardware. For example, this assumption holds in wireless sensor networks.

3.2 The controlled behavior under sensor deception attack

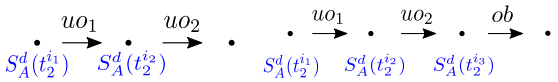
The presence of the attacker induces a new controlled language that needs to be investigated. More specifically, S_P, $\hat{f}_A$, and P_e^S together effectively generate a new supervisor S_A for system G, as depicted in Fig. 2. Formally, we define $S_A : \Sigma_o^* \to 2^\Gamma$ as $S_A(s) = [S_P \circ P_e^S \circ \hat{f}_A(s)]$. Note that $\hat{f}_A(s)$ returns the set of modified strings and S_P assigns a control decision to each projected modified string; S_A returns the set containing all these control decisions. An equivalent definition is $S_A(s) = \{\gamma |\ \exists s_A \in \hat{f}_A(s)$ s.t. $\gamma = S_P(P_e^S(s_A))\}$. Defining the supervised language based on nondeterministic control decisions is cumbersome and complicated (Lin 2014). Nonetheless, we can avoid this difficulty by analyzing the language generated by the events in $\Sigma \cup \Sigma_e^a$. For this reason, we define the function $S_A^d = S_P \circ P_e^S$ to be the deterministic part of S_A. The function S_A^d is used when the attacker has decided which modified string to send to the supervisor. Based on f_A and S_A^d the language generated by S_A/G is defined recursively as follows:

(1) $\epsilon \in \mathcal{L}(S_A/G)$

(2) $\big(t_1 \in \mathcal{L}(G) \cap \Sigma_{uo}^*(\Sigma_o \cup \{\epsilon\})\big) \wedge \big(\exists t_2 \in f_A(\epsilon, \epsilon)$ and $i_1 \leq i_2 \leq \ldots \leq i_{|t_1|} \in \mathbb{N}^{|t_2|}$ s.t. $\forall j \in \mathbb{N}^{|t_1|} : e_{t_1}^j \in S_A^d(t_2^{i_j})\big) \wedge \big(P_o(t_1) \neq \epsilon \Rightarrow i_{|t_1|} = |t_2|\big) \Leftrightarrow t_1 \in \mathcal{L}(S_A/G)$

(3) $\big(s \in \mathcal{L}(S_A/G)\big) \wedge \big(e_s^{|s|} \in \Sigma_o\big) \wedge \big(st_1 \in \mathcal{L}(G)$ where $t_1 \in \Sigma_{uo}^*(\Sigma_o \cup \{\epsilon\})\big) \wedge \big(\exists t_3 \in \hat{f}_A(P_o(s^{|s|-1})),\ \exists t_2 \in f_A(t_3, e_s^{|s|})$ and $i_1 \leq i_2 \leq \ldots \leq i_{|t_1|} \in \mathbb{N}^{|t_2|}$ s.t. $\forall j \in \mathbb{N}^{|t_1|} : e_{t_1}^j \in S_A^d(t_3 t_2^{i_j})\big) \wedge \big(P_o(t_1) \neq \epsilon \Rightarrow i_{|t_1|} = |t_2|\big) \Leftrightarrow st_1 \in \mathcal{L}(S_A/G)$

The above definition captures the intricate interaction between plant, supervisor and attacker. Two important concepts are applied in this definition. First, the supervisor issues a control decision whenever it receives an observable event. An observable event can be either a legitimate event or a fictitious event inserted by an attacker. Second, the plant can execute any unobservable event enabled by the current control decision. To demonstrate how to compute $\mathcal{L}(S_A/G)$, we illustrate condition (2) in Figure 3.



(a) Only unobservable events (b) Last event is observable

Fig. 3. Demonstration of Condition (2)

Assume that $uo_1, uo_2 \in \Sigma_{uo}$, $ob \in \Sigma_o$ and uo_1uo_2, $uo_1uo_2ob \in \mathcal{L}(G)$. Figure 3(a) describes how $t_1 = uo_1uo_2 \in \mathcal{L}(S_A/G)$. The string uo_1uo_2 belongs to $\mathcal{L}(S_A/G)$ whenever there exists a string $t_2 \in f_A(\epsilon, \epsilon)$ and indices $i_1 \leq i_2 \in \mathbb{N}$ such that $uo_1 \in S_A^d(t_2^{i_1})$ and $uo_2 \in S_A^d(t_2^{i_2})$. Similarly, Fig. 3(b) describes how $uo_1uo_2ob \in \mathcal{L}(S_A/G)$. If we use the same indices i_1, i_2 as before, then we just need to test if $ob \in S_A^d(t_2^{i_3}) = S_A^d(t_2)$. In the case of the last event being observable, we assume that the attacker has finished its entire modification t_2. On the other hand, we assume that unobservable events can be executed based on control decisions of string prefixes of t_2. Condition (3) applies the same mechanism of Condition (2), however it has nuances related to previous modifications made by the attacker.

Similarly to the language $\mathcal{L}(S_A/G)$, we recursively define the set of reachable states of G under the supervision of S_A^d driven by the edited string s, for any $s \in (\Sigma_o \cup \Sigma_a^e)^*$, as follows:

$$RE_G^i(s, S_A^d) := \begin{cases} \{x \in X | \exists u \in NX_{P_e^G(e_s^i)}\big(RE_G^{i-1}(s, S_A^d)\big), \\ \exists t \in (S_A^d(s^i) \cap \Sigma_{uo})^* : x = \delta(u, t)\}, \\ \qquad\qquad \text{if } e_s^i \in \Sigma_o \cup \Sigma_a^d \\ \{x \in X | \exists u \in \big(RE_G^{i-1}(s, S_A^d)\big), \\ \exists t \in (S_A^d(s^i) \cap \Sigma_{uo})^* : x = \delta(u, t)\}, \\ \qquad\qquad \text{if } e_s^i \in \Sigma_a^i \end{cases} \tag{5}$$

for $1 \leq i \leq |s|$ and:

$$RE_G^0(s, S_A^d) := \{x \in X | \exists t \in (S_A^d(\epsilon) \cap \Sigma_{uo})^* : x = \delta(x_0, t)\} \tag{6}$$

We extended the definition under S_A^d for edited strings since S_A is nondeterministic. In the above definition, we want to define the reachable set of states *after a particular edited string was selected*. Note that, the definitions of $RE_G^i(s_A, S_A^d)$ and of $\mathcal{L}(S_A/G)$ share many similarities since they are related.

The above-described dynamical interaction between the attacker and the supervisor is different than the one in (Su 2018), where the supervisor reacts to *strings* of inserted events produced by the attacker.

3.3 Attacker objectives

In order to formally pose the problem, we must specify the objective of the attacker. We assume that G contains a set of *critical* unsafe states defined as $X_{crit} \subset X$ such that $\forall x \in X_{crit}$, x is never reached when S_P controls G and no attacker is present. In general, not all states reached by strings of G that are disabled by S_P (when no attacker is present) are critically unsafe. In practice, there will be certain states among those that correspond to physical damage to the system, such as "overflow" states or "collision" states, for instance. Similar notions of critical unsafe states have been used in other works, e.g., (Paoli and Lafortune 2005, Paoli *et al.* 2011). Therefore, the objective of the attacker is to force the controlled behavior under attack $\mathcal{L}(S_A/G)$ to reach any state in X_{crit}.

The second objective of the attacker is to remain stealthy, i.e., the attacker should feed the supervisor with *normal* behavior. By normal behavior, we mean that the supervisor should receive strings in the language $\mathcal{L}(S_P/G)$. We assume that if the supervisor receives a string that is not included in $\mathcal{L}(S_P/G)$, then an intrusion detection module detects the attacker.

To capture the behavior that detects the attacker, we define an automaton that captures both the control decisions of the supervisor and the normal/abnormal behavior. We start by defining the automaton $H = obs(R||G)$, where obs is the standard observer automaton of G (cf. (Cassandras and Lafortune 2008)) and $H = (X_H, \Sigma_o, \delta_H, x_{0,H})$, where $X_H \subseteq 2^{X_R \times X_G}$. The automaton H captures only the normal projected behavior of the controlled system. However, H cannot be used as an supervisor since it only could contain inadmissible control decisions and it does not have all decisions of R.

Based on H, we define $\tilde{R} = (\tilde{Q}, \Sigma, \tilde{\mu}, \bar{q}_0)$, where $\tilde{Q} = X_H \cup \{\text{dead}\}$, and $\tilde{\mu}$ is defined to include all the transitions in δ_H plus the additional transitions: $(\forall q \in X_H)$ $(\forall e \in [(\Sigma_{uc} \cap \Sigma_o) \backslash \Gamma_H(q)])$ $\tilde{\mu}(q, e) =$ dead, $(\forall q \in X_H)$ $(\forall e \in (\Sigma_{uc} \cap \Sigma_{uo}))$ $\tilde{\mu}(q, e) = q$, $(\forall q \in X_H)$ $(\forall e \in (\Sigma_c \cap \Sigma_{uo})$ s.t. $\exists x \in q,\ e \in \Gamma_{R||G}(x))$ $\tilde{\mu}(q, e) = q$ and $(\forall e \in \Sigma_{uc})$ $\tilde{\mu}(\text{dead}, e) = \text{dead}$. In this manner, automaton $\tilde{R}$ embeds the same admissible control decisions as automaton R and it differentiates normal/abnormal behavior. The state *dead* in $\tilde{R}$ captures the abnormal behavior of the controlled system. The attacker remains stealthy as long as $\tilde{R}$ does not reach the state *dead*. Figure 4 illustrates the supervisor $\tilde{R}$ computed for Example 1.

Remark: No new transition to the dead state with controllable events are included in the definition of $\tilde{R}$. For simplicity, we assume that the supervisor does not enable controllable events unnecessarily. In this manner, only uncontrollable events can reach the dead state.

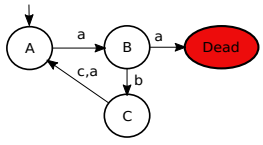


Fig. 4. Supervisor $\tilde{R}$ for Example 1

3.4 Problem formulation

Finally, we are able to formally state the problem formulation of the synthesis of stealthy sensor deception attack problem.

Problem 3 (Synt. of Stealthy Sensor Deception Attacks)

Given an attacker that has full knowledge of the models G and $\tilde{R}$, and is capable of compromising events $\Sigma_a \subseteq \Sigma_o$, synthesize an attack function f_A such that it generates a controlled language $\mathcal{L}(S_A/G)$ that satisfies:

1. $\forall s \in P_o(\mathcal{L}(S_A/G))$, $\hat{f}_A(s)$ *is defined* ***(Admissibility)****;*
2. $\forall s \in \mathcal{L}(S_A/G)$, $P_e^S\big(\hat{f}_A(P_o(s))\big) \subseteq P_o(\mathcal{L}(S_P/G))$ ***(Stealthiness)****;*

3(a). $\exists s \in \mathcal{L}(S_A/G)$ *s.t.* $\big(\forall t \in [P_o^{-1}(P_o(s)) \cap \mathcal{L}(S_A/G)]\big)$ $\delta(x_0, t) \in X_{crit}$.

In this case, we say that f_A is a ***strong attack****. We additionally define the notion of a* ***weak attack*** *as follows:*

3(b). $\exists s \in \mathcal{L}(S_A/G)$ *s.t.* $\delta(x_0, s) \in X_{crit}$.

The *Admissibility* condition guarantees that f_A is well defined for all projected strings in the modified controlled language $P_o(\mathcal{L}(S_A/G))$. The *Stealthiness* condition guarantees that the attacker stays undetected by the supervisor, meaning that any string in $P_o(\mathcal{L}(S_A/G))$ should be modified to a string within the original controlled behavior. In this manner, $\tilde{R}$ never reaches state "dead". Lastly, the reachability of critical states is stated in condition 3, where condition 3(a) is a strong version of the problem. In the strong case, the attacker is sure that the system has reached a critical state if string s occurs in the system. Condition 3(b) is a relaxed version, where the attacker might not be sure if a critical state was reached, although it could have been reached. Both variations of condition 3 guarantee the existence of at least one successful attack, namely, when string s occurs in the new controlled behavior.

Remark: Problem 3 assumes that the attacker has full knowledge of the plant and the supervisor models. Although it might be difficult to achieve this assumption in practice, our paper studies the worst attack scenario case. Moreover, this assumption is common practice within the cyber-security domain.

3.5 Attack scenarios

Problem 3 is defined for a general function f_A, which is defined as in Definition 2. However, there exists an interaction between the attacker, the system, and the supervisor that may limit the power of the attacker. For this reason, we propose three different types of attack functions. Each of them has different assumptions that are application-dependent.

The first scenario assumes that the attacker has "time" to perform any *unbounded* modification. In other words, the system does not execute any event until the attacker finishes its modification. Such an attack function is defined as an *unbounded deterministic* attack function.

Definition 4 *An* ***unbounded*** *deterministic attack function is an attack function f_A s.t.* $\big(\forall s \in (\Sigma_o \cup \Sigma_a^e)^*\big)(\forall e \in \Sigma_o)[f_A(s,e)! \Rightarrow |f_A(s,e)| = 1]$.

The previous scenario considers a powerful attacker, and it might be unrealistic in many real applications. In this case, the second scenario limits the first one by considering only *bounded deterministic* attack functions. In other words, we assume that the system does not react up to a bounded number of modifications made by the attacker. Bounded deterministic attack functions are defined next.

Definition 5 *Given $N_A \in \mathbb{N}^+$, a* ***bounded*** *deterministic attack function is an attack function f_A s.t.* $\big(\forall s \in (\Sigma_o \cup \Sigma_a^e)^*\big)(\forall e \in \Sigma_o)$ $[(f_A(s,e)! \Rightarrow |f_A(s,e)| = 1) \wedge (s_A \in f_A(s,e) \Rightarrow |s_A| \leq N_A)]$.

Finally, the last scenario is the least powerful attacker we consider. We assume that the system can interrupt the attacker's modification at any point. We call this function an *interruptible* attack function. Formally, we define it as:

Definition 6 *An attack function f_A is an* ***interruptible*** *attack function if* $(\forall s \in (\Sigma_o \cup \Sigma_a^e)^*)(\forall e \in \Sigma_o)$ $[s_A \in f_A(s,e) \Rightarrow \bar{s_A}\backslash\{\epsilon\} \subseteq f_A(s,e)]$.

We use a simple example to provide more intuition about the above-described scenarios.

Example 7 *Let $\mathcal{L}(G) = \overline{\{a\}\{b\}^*}$, where $\Sigma_a = \Sigma_o = \{a, b\}$. Assume that S_P never disables an event. We partially define three attack functions: f_A^1, f_A^2, and f_A^3.*

$$f_A^1(\epsilon, a) = \{a\} \tag{7}$$
$$f_A^2(\epsilon, a) = \{ab_i^n\},\ \textit{where } n \in \mathbb{N} \tag{8}$$
$$f_A^3(\epsilon, a) = \{a, ab_i, ab_ib_i\} \tag{9}$$

f_A^1 is a bounded deterministic attack function with $N_A = 1$, f_A^2 is an unbounded deterministic attack function, and lastly, f_A^3 is an interruptible attack function.

4 Insertion-Deletion Attack Structure

4.1 Definition

An Insertion-Deletion Attack structure (IDA) is an extension of the notion of *bipartite transition structure* presented in (Yin and Lafortune 2016*b*). An IDA captures the game between the environment and the supervisor considering the possibility that a subset of the sensor network channels may be compromised by a malicious attacker, whose moves will be constrained according to various rules. In this game we fix the supervisor's decisions as those defined in the given $\tilde{R}$. The environment's decisions are those of the attacker and the system. Therefore, in this game, environment states have both attacker's and system's decisions. In this section, we define the generic notion of an IDA. In the next sections, we will construct specific instances of it, according to the permitted moves of the attacker.

In order to build the game, we define an information state as a *pair* $IS \in 2^X \times \tilde{Q}$, and the set of all information states as $I = 2^X \times \tilde{Q}$. The first element in an IS represents the *correct state estimate* of the system, as seen by the attacker for the actual system outputs. The second element represents the supervisor's state, which is the current state of its realization based on the edited string of events that it receives. As defined, an IS embeds the necessary information for either player to make a decision.

Definition 8 *An Insertion-Deletion Attack structure (IDA) A w.r.t. G, Σ_a, and $\tilde{R}$, is a 7-tuple*

$$A = (Q_S, Q_E, h_{SE}, h_{ES}, \Sigma, \Sigma_a^e, y_0) \tag{10}$$

where:

- $Q_S \subseteq I$ *is the set of S-states, where S stands for Supervisor and where each S-state is of the form* $y = (I_G(y), I_S(y))$*, where* $I_G(y)$ *and* $I_S(y)$ *denote the correct system state estimate and the supervisor's state, respectively;*
- $Q_E \subseteq I$ *is the set of E-states, where E stands for Environment; each E-state is of the form* $z = (I_G(z), I_S(z))$ *defined in the same way as in the S-states case;*
- $h_{SE} : Q_S \times \Gamma \to Q_E$ *is the partial transition function from S-states to E-states, defined only for* $\gamma = \Gamma_{\tilde{R}}(I_S(y))$*:*

$$h_{SE}(y, \gamma) := (UR_\gamma(I_G(y)),\ I_S(y)) \tag{11}$$

- $h_{ES} : Q_E \times (\Sigma_o \cup \Sigma_a^e) \to Q_S$ *is the partial transition function from E-states to S-states, satisfying the following constraints: for any* $y \in Q_S$*,* $z \in Q_E$ *and* $e \in \Sigma_o \cup \Sigma_a^e$*, if* $h_{ES}(z, e)$ *is defined, then* $h_{ES}(z, e) := y$ *where:*

$$y = \begin{cases} \big(NX_e(I_G(z)), \tilde{\mu}(I_S(z), e)\big), \textbf{ if} \\ \qquad e \in \Gamma_{\tilde{R}}\big(I_S(z)\big) \cap \Gamma_G\big(I_G(z)\big) & \text{(12a)} \\ \big(I_G(z), \tilde{\mu}(I_S(z), P_e^S(e))\big), \textbf{ if } e \in \Sigma_a^i \text{ and} \\ \qquad \mathcal{M}_e(e) \in \Gamma_{\tilde{R}}\big(I_S(z)\big) & \text{(12b)} \\ \big(NX_{P_e^G(e)}(I_G(z)), I_S(z)\big), \textbf{ if } e \in \Sigma_a^d \text{ and} \\ \qquad \mathcal{M}_e(e) \in \Gamma_{\tilde{R}}\big(I_S(z)\big) \cap \Gamma_G\big(I_G(z)\big) & \text{(12c)} \end{cases}$$

- Σ *is the set of events of* G*;*
- Σ_a^e *is the set of editable events;*
- $y_0 \in Q_S$ *is the initial S-state:* $y_0 := (\{x_0\}, q_0)$*.*

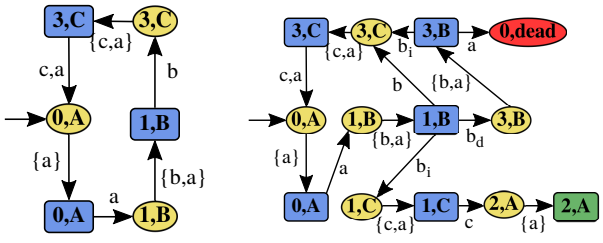
Since the purpose of an IDA is to capture the game between the supervisor and the environment, we use a bipartite structure to represent each entity. An S-state is an IS containing the state estimate of the system G and the supervisor's state; it is where the supervisor issues its control decision. An E-state is an IS at which the environment (system or attacker) selects one among the observable events to occur.

A transition from a S-state to an E-state represents the updated unobservable reach in G's state estimate together with the current supervisor state. Note that h_{SE} is only defined for y and γ such that $\gamma = \Gamma_{\tilde{R}}(I_S(y))$. On the other hand, a transition from an E-state to a S-state represents the "observable reach" immediately following the execution of the observable event by the environment. In this case, both the system's state estimate and the supervisor's state are updated. However, these updates depend on the type of event generated by the environment: (i) true system event unaltered by the attacker; (ii) (fictitious) event insertion by the attacker; or (iii) deletion by the attacker of an event just executed by the system. Thus, the transition rules are split into three cases, described below.

The partial transition function h_{ES} is characterized by three cases: Equations (12a),(12b), and (12c). Equation (12a) is related to *system's* actions, while Equations (12b) and (12c) are related to *attacker's* actions. In the case of Equation (12a), the system generates a feasible (enabled) event and the attacker lets the event reach the supervisor intact, either because it cannot compromise that event, or because it chooses not to make a move. In the case of Equation (12b), the attacker only inserts events consistent with the control decision of the current supervisor state. In the case of Equation (12c), it only deletes actual observable events generated by the system. Equation (12c) differs from Equation (12a) since it adds the condition that the event executed is compromised and that the attacker deleted it. *Remark:* a given IDA will contain some attack moves (since it is a generic structure), all of which have to satisfy the constraints in the definition of h_{ES}.

In the remainder of this paper, we assume that all states included in an IDA are reachable from its initial state.

Example 9 *Let us consider system G and supervisor $\tilde{R}$ from Example 1. Considering the compromised event set $\Sigma_a = \{b\}$, Fig. 5 gives two IDA examples. In the figure, oval states represent S-states and rectangular states represent E-states. Moreover, the red state indicates where the supervisor reaches the "dead" state and the green state (2,A) represents the successful reaching of a critical state.*



(a) An IDA with no attacks (b) A second IDA example

Fig. 5. IDA Structures

Given two IDAs A_1 and A_2, we say that A_1 is a subsystem of A_2, denoted by $A_1 \sqsubseteq A_2$, if $Q_E^{A_1} \subseteq Q_E^{A_2}$, $Q_S^{A_1} \subseteq Q_S^{A_2}$,

and for any $y \in Q_S^{A_1}$, $z \in Q_E^{A_1}$, $\gamma \in \Gamma$, and $e \in \Sigma$, we have that:

(1) $h_{SE}^{A_1}(y,\gamma) = z \Rightarrow h_{SE}^{A_2}(y,\gamma) = z$ and
(2) $h_{ES}^{A_1}(z,e) = y \Rightarrow h_{ES}^{A_2}(z,e) = y$.

Before, we discuss relevant properties of the IDA structure, we define a property about E-states.

(P1) An E-state $z \in Q_E$ is called a *race-free state*, if the following condition holds:

$$\forall e \in \Gamma_{\tilde{R}}(I_S(z)) \cap \Sigma_o : (\exists x \in I_G(z) : \delta(x,e)!) \Rightarrow [h_{ES}(z,e)! \vee h_{ES}(z,e_d)!]$$

Property (P1) ensures the non-existence of a race condition between the attacker and the system in a given E-state. Specifically, when (P1) holds in any E-state, the attacker has the option of either letting the event reach the supervisor intact or preventing the event from reaching the supervisor (or allowing both). It means that the attacker can wait for the system's response to the most recent control action and react accordingly. Note that if the event in question is a compromised event, then the attacker has the option of allowing both actions since we defined nondeterministic attack functions. In the case of deterministic attack functions, we force the attacker to select one of the actions. If (P1) does not hold at a particular E-state, then the attacker *must insert an event*, and this insertion must take place before the system reacts to the most recent control action sent by the supervisor. (In some sense, the attacker is "racing" with the system.) An IDA that satisfies (P1) for all E-states is called a *race-free* IDA.

Definition 10 (Induced E-state) *Given an IDA A, $IE(z,s)$ is defined to be the E-state induced by string $s \in (\Sigma_o \cup \Sigma_a^e)^*$, when starting in the E-state z. $IE(z,s)$ is computed recursively as:*

$$IE(z,\epsilon) \coloneqq z$$

$$IE(z,se) \coloneqq \begin{cases} h_{SE}\big(y, \Gamma_{\tilde{R}}\big(I_S(y)\big)\big) & \text{if } h_{ES}(IE(z,s),e)! \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $y = h_{ES}(IE(z,s),e)$

We also define $IE(s) \coloneqq IE(z_0,s)$, where $z_0 = h_{SE}(y_0, \Gamma_{\tilde{R}}(I_S(y_0)))$.

We conclude this section by defining the notion of *embedded* attack function in an IDA and presenting two lemmas about reachability in IDAs. The first lemma shows that after observing an edited string s_A, the attacker correctly keeps track of the supervisor state $\tilde{R}$. Then we show that an IDA correctly estimates the set of possible states of the system after the occurrence of edited string s_A.

Definition 11 (Embedded f_A) *An attack function f_A is said to be embedded in an IDA A if $\big(\forall s \in P_o(\mathcal{L}(S_A/G))\big)\big(\forall t \in \hat{f}_A(s)\big)\big(\forall i \in \mathbb{N}^{|t|-1}\big)$, then $IE(t^i)$ is defined.*

Intuitively, an attack function is embedded in an IDA if for every modified string that is consistent with the behavior of G, we can find a path in the IDA for that string. Note that we limit this constraint on f_A to strings in $P_o(\mathcal{L}(S_A/G))$, since these are the ones consistent with G under a given f_A. We are not interested in the definition of any f_A for strings outside of the controlled behavior.

Lemma 12 *Given a system G, supervisor $\tilde{R}$, and an IDA structure A with an embedded attack function f_A, for any string $s \in \mathcal{L}(S_A/G)$ and any string $s_A \in \hat{f}_A(P_o(s))$, we have*

$$I_S\big(IE(s_A)\big) = \tilde{\mu}\big(q_0, P_e^S(s_A)\big) \tag{13}$$

$$I_G\big(IE(s_A)\big) = RE_G^{|s_A|}\big(s_A, S_A^d\big) \tag{14}$$

5 All Insertion-Deletion Attack Structure

In this section, we define the *All* Insertion-Deletion Attack Structure, a specific type of IDA abbreviated as AIDA hereafter. Given S_P and Σ_a, the AIDA embeds *all* insertion-deletion actions the attacker is able to execute. We discuss its construction and properties.

5.1 Definition

As consequence of Lemma 12, if we construct an IDA structure based on S_P and Σ_a that is "as large as possible", then it will include all valid insertion and deletion actions for the attacker. We formally define such a structure as the All Insertion-Deletion Attack structure.

Definition 13 ($\mathsf{AIDA}(G, S_P, \Sigma_a)$) *Given a system G, a supervisor S_P, and a set of compromised events Σ_a, the **All Insertion-Deletion Attack structure (AIDA)**, denoted by $\mathsf{AIDA}(G, S_P, \Sigma_a) = (Q_S, Q_E, h_{SE}, h_{ES}, \Sigma, \Sigma_a^e, y_0)$, is defined as the **largest** IDA w.r.t to G, S_P and Σ_a s.t.*

(1) For any $y \in Q_S$, we have $|\Gamma_{AIDA}(y)| = 0 \Leftrightarrow I_S(y) =$ dead
(2) For any $z \in Q_E$, we have
  (a) $\forall e \in \Gamma_{\tilde{R}}(I_S(z)) \cap \Gamma_G(I_G(z)) \cap \Sigma_o : (h_{ES}(z,e)! \vee h_{ES}(z,e_d)!)$ or
  (b) $I_G(z) \subseteq X_{crit} \Rightarrow |\Gamma_{AIDA}(z)| = 0$

Condition 2(a) alone satisfies the non-existence of a race condition in the AIDA. Conditions 1 guarantees that the AIDA stops its search once the attack is detected. Similarly, Condition 2(b) stops the search given that the attacker knows it has reached its goal.

By "largest" structure, we mean that for any IDA A satisfying the above conditions: $A \sqsubseteq \mathsf{AIDA}(G, S_P, \Sigma_a)$. This notion

of "largest" IDA is well defined. If A_1 and A_2 are two IDA structures satisfying the above conditions, then their union still satisfies the conditions, where the union $A_1 \cup A_2$ is defined as: $Q_E^{A_1 \cup A_2} = Q_E^{A_1} \cup Q_E^{A_2}$, $Q_S^{A_1 \cup A_2} = Q_S^{A_1} \cup Q_S^{A_2}$, and for any $y \in Q_E^{A_1 \cup A_2}$, $z \in Q_S^{A_1 \cup A_2}$, $\gamma \in \Gamma$ and $e \in \Sigma \cup \Sigma_a^e$, we have that $h_{SE}^{A_1 \cup A_2}(y, \gamma) = z \Leftrightarrow \exists i \in \{1, 2\} : h_{SE}^{A_i}(y, \gamma) = z$ and $h_{ES}^{A_1 \cup A_2}(z, e) = y \Leftrightarrow \exists i \in \{1, 2\} : h_{ES}^{A_i}(z, e) = y$.

5.2 Construction

The construction of the AIDA follows directly from its definition. It enumerates all possibles transitions for each state by a breath-first search. Each S-state has at most one control decision, which is related to the supervisor state $\tilde{R}$. On the other hand, each E-state enumerates both system's and attacker's actions, according to its system and supervisor estimate, respectively. In practice, we do not need to search the entire state space; we stop the branch search when an S-state reaches a state with the supervisor at the "dead" state or when the system estimate of an E-state is a subset of X_{crit}.

The procedure mentioned above is described in Algorithm 1 (Construct-AIDA). It has the following parameters: $AIDA$ is the graph structure of the AIDA we want to construct; $AIDA.E$ and $AIDA.S$ are the E- and the S-state sets of the structure, respectively; $AIDA.h$ is its transition function; Q is a queue. We begin the procedure by initializing $AIDA.S$ with a single element $y_0 = (\{x_0\}, q_0)$. The breath-first search is then performed by the procedure DoBFS. The transitions between S-states and E-states are dealt within lines 7 to 11. The transitions between E-states and S-states are defined in lines 12 to 25, where each attack possibility is analyzed. These transitions are defined exactly as in Definition 8, equations (12a), (12b) and (12c). (For the sake of readability, we employ the usual triple notation (origin, event, destination) for the transition function.) The procedure converges when all uncovered states (states in the queue) are covered, meaning we have traversed the whole reachable space of E- and S-states. Note that lines 29 and 33 impose the stop conditions of Definition 13.

Theorem 14 *Algorithm Construct-AIDA correctly constructs the AIDA.*

Example 15 *We return to system G and supervisor $\tilde{R}$ in Fig. 1(b) with $\Sigma_a = \{b\}$. The IDA shown in Fig. 5(b) is its AIDA. Note that there exists an S-state where $\tilde{R}$ reaches state* dead*. Moreover, there exists an E-state where G reaches a critical state, $(2, A)$.*

Remark: The AIDA has at most $2^{|X|+1}|\tilde{Q}|$ states since $Q_1, Q_2 \subseteq I$. If $\Sigma_{uo} = \emptyset$, then it has at most $2|X||\tilde{Q}|$ states.

Algorithm 1 Construct-AIDA

Require: G, $\tilde{R}$ and Σ_a
Ensure: $AIDA$
1: $AIDA \leftarrow$ DoBFS$\big(G, \tilde{R}, (\{x_0\}, q_0)\big)$
2: **procedure** DoBFS(G, R, y)
3: $AIDA.S \leftarrow \{y\}$, $AIDA.E \leftarrow \emptyset$, $AIDA.h \leftarrow \emptyset$
4: Queue $Q \leftarrow \{y\}$
5: **while** Q is not empty **do**
6: $c \leftarrow Q.dequeue(\,)$
7: **if** $c \in AIDA.S$ **then**
8: $\gamma \leftarrow \Gamma_{\tilde{R}}\big(I_S(c)\big)$
9: $z \leftarrow \big(UR_\gamma\big(I_G(c)\big), I_S(c)\big)$
10: $AIDA.h \leftarrow AIDA.h \cup \{(c, \gamma, z)\}$
11: Add-State-to-AIDA$(z, AIDA, Q)$
12: **else if** $c \in AIDA.E$ **then**
13: **for all** $e \in \Sigma_o \cap \Gamma_{\tilde{R}}\big(I_S(c)\big)$ **do**
14: **if** $e \in \Gamma_G\big(I_G(c)\big)$ **then**
15: $y \leftarrow \Big(NX_e\big(I_G(c)\big), \tilde{\mu}\big(I_S(c), e\big)\Big)$
16: $AIDA.h \leftarrow AIDA.h \cup \{(c, e, y)\}$
17: Add-State-to-AIDA$(y, AIDA, Q)$
18: **if** $e \in \Gamma_G\big(I_G(c)\big) \wedge e \in \Sigma_a$ **then**
19: $y \leftarrow \big(NX_e\big(I_G(c)\big), I_S(c)\big)$
20: $AIDA.h \leftarrow AIDA.h \cup \{(c, e_d, y)\}$
21: Add-State-to-AIDA$(y, AIDA, Q)$
22: **if** $e \in \Sigma_a$ **then**
23: $y \leftarrow \big(I_G(c), \tilde{\mu}\big(I_S(c), e\big)\big)$
24: $AIDA.h \leftarrow AIDA.h \cup \{(c, e_i, y)\}$
25: Add-State-to-AIDA$(y, AIDA, Q)$
26: **procedure** ADD-STATE-TO-AIDA$(c, AIDA, Q)$
27: **if** $c \notin AIDA.E \wedge c$ is an E-state **then**
28: $AIDA.E \leftarrow AIDA.E \cup \{c\}$
29: **if** $I_G(c) \nsubseteq X_{crit}$ **then**
30: $Q.enqueue(c)$
31: **else if** $c \notin AIDA.S \wedge\ c$ is a S-state **then**
32: $AIDA.S \leftarrow AIDA.S \cup \{c\}$
33: **if** $I_S(c) \neq$ dead **then**
34: $Q.enqueue(c)$

6 Synthesis of Stealthy IDA: Interruptible Case

The AIDA embeds all attack functions, including non-stealthy strategies, i.e., those that lead to state *dead* of the supervisor. In this section, we show how to synthesize *interruptible* and *stealthy* attack functions that solve Problem 3. First, we present a pruning process that removes non-stealthy interruptible strategies from the AIDA. The resulting pruned IDA is then used in the synthesis algorithm. Recall that three attack types were presented in Section 3 (Definitions 4, 5, and 6). We focus our attention on the interruptible case in this section; next, in Section 7, we present our results for the two remaining cases.

6.1 Pruning Process

The AIDA could reach state *dead* of supervisor $\tilde{R}$. Each time this occurs, it means that the last step of the attack is no longer stealthy. Hence, we must prune the AIDA in order to embed only stealthy attacks. We pose this pruning process as a *meta-supervisory-control* problem, where the "plant" is the entire AIDA, the specification for that plant is to prevent reaching state *dead* of the supervisor, and the *controllable* events are the actions of the attacker. We assume that the reader is familiar with the standard "Basic Supervisory Control Problem" of (Ramadge and Wonham 1987); we adopt the presentation of that problem as BSCP in (Cassandras and Lafortune 2008). First, we show necessary modifications to the standard BSCP algorithm in order to construct the Stealthy-AIDA.

Algorithm 2 Modified BSCP for Interruptible Attacker

Require: $A = (Q_E \cup Q_S, E, f_{se} \oplus f_{es}, a_0 = y_0)$, where $E \subseteq (\Sigma_o \cup \Sigma_a^e \cup \Gamma)$ and $A_{trim} = (A^t, E, f^t, a_0^t)$, where $A^t \subseteq Q_E \cup Q_S$

1: **Step 1** Set $H_0 = (A_0, E, g_0, a_0) = A_{trim}$, and $i = 0$
2: **Step 2** Calculate
3: **Step 2.1** $A_i' = \{a \in A_i | \Gamma_A(a) \cap E_{uc} \subseteq \Gamma_{H_i}(a)\}$
4: **Step 2.2** $A_i^* = \{a \in A_i' | e \in \Gamma_A(a) \Rightarrow (e \in \Gamma_{H_i}(a) \vee e_d \in \Gamma_{H_i}(a)\}$
$g_i' = g_i | A_i^*$ [transition function update]
5: **Step 2.3** $H_{i+1} = Trim(A_i^*, E, g_i', a_0)$
6: **Step 3** If $H_{i+1} = H_i$, Stop; otherwise $i \leftarrow i + 1$, back to Step 2

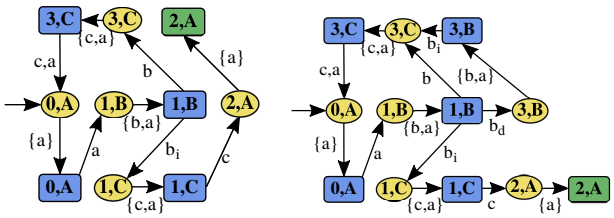
The difference between the original BSCP algorithm (see, e.g., (Cassandras and Lafortune 2008)) and its modified version in Algorithm 2 is the addition of Step 2.2. Since the BSCP algorithm has quadratic worst-time complexity in the number of states of the automaton A_{trim}, it follows that Algorithm 2 also has quadratic worst-time complexity. In order to ensure the desired interruptibility condition of an attack function, we need that each state that it visits in the AIDA be race free. Therefore, at Step 2.2 we enforce the race-free condition at every E-state of the IDA. In order to enforce such condition, the algorithm deletes E-states where both the "let through" transition *and* the "erasure" transition are absent for a feasible system event. Therefore, the resulting IDA from Algorithm 2 is a race-free IDA.

To compute the stealthy AIDA structure, we define as system the AIDA constructed according to Algorithm 1. Moreover, any event $e \in \Sigma_a \cup \Sigma_a^e$ is treated as *controllable* while any control decision $\gamma \in \Gamma$ and any event $e \in \Sigma_o \backslash \Sigma_a$ is treated as *uncontrollable*. The specification language, realized by A_{trim}, is obtained by deleting the states where the supervisor reaches the dead state, i.e., by deleting in the AIDA all states of the form $y = (S, \text{dead})$ for any $S \subseteq X$.

We formalize the pruning process for obtaining all stealthy insertion-deletion attacks as follows.

Definition 16 *Given the AIDA constructed according to Algorithm 1, define the system automaton $A_G = AIDA$ with $\Sigma_c^A = \Sigma_a \cup \Sigma_a^e$ as the set of controllable events and $\Sigma_{uc}^A = (\Sigma_o \backslash \Sigma_a) \cup \Gamma$ as the set of uncontrollable events. The specification automaton is defined by A_{trim}, which is obtained by trimming from A_G all its states of the form $(S, dead)$, for any $S \subseteq X$. The* Stealthy AIDA structure*, called the ISDA (Interruptible Stealthy Deceptive Attack), is defined to be the automaton obtained after running Algorithm 2 on A_{trim} with respect to A_G and Σ_c^A.*

Example 17 *We return to the AIDA in Figure 5, where we would like to obtain the ISDA as in Definition 16. Figure 6(a) shows the resulting ISDA, where the attacker cannot play event b_d at E-state $(1, B)$. For example, if the attacker takes such decision, then the uncontrollable event a could be executed, revealing the attack to the supervisor.*



(a) ISDA (b) USDA

Fig. 6. Stealthy Deceptive Structures

Lemma 18 *If an interruptible attack function f_A satisfies the admissibility and the stealthy conditions from Problem 3, then it is embedded in the ISDA.*

Lemma 19 *If an interruptible attack function f_A is constructed from the ISDA, then the stealthy condition of Problem 3 is satisfied.*

Constructing an f_A from the ISDA means selecting decisions at E-states and properly defining f_A from the selected decisions. *Remark*: Lemma 19 does not guarantee the admissibility condition. An inadmissible interruptible f_A could be synthesized from the ISDA. However an admissible interruptible f_A can always be synthesized from the ISDA. Since the ISDA is race-free, an inadmissible f_A synthesized from the ISDA can always be extended to make it admissible; see Example 6 below.

Theorem 20 *The ISDA embeds all possible interruptible stealthy insertion-deletion attack strategies with respect to Σ_a, $\tilde{R}$ and G.*

6.2 Synthesis of Stealthy Interruptible Functions

Based on the ISDA, we can synthesize interruptible attack functions that satisfy the admissibility and the stealthy conditions. In order to fully satisfy Problem 3, we need to address its last condition about the existence of strong or weak attacks.

Theorem 21 *Given the ISDA, there exists an interruptible f_A that* strongly *satisfies Problem 3 if and only if there exists an E-state z in the ISDA s.t. $I_G(z) \subseteq X_{crit}$. On the other hand, it* weakly *satisfies Problem 3 if and only if there exists an E-state z in the ISDA s.t. $\exists x \in I_G(z)$, $x \in X_{crit}$.*

Theorem 21 gives necessary and sufficient conditions for synthesis of interruptible attack functions. The following algorithm (Algorithm 3) synthesizes a simple interruptible attack function that satisfies Problem 3. Specifically, Algorithm 3 encodes an interruptible attack function in an automaton F. The encoded function simply includes *one* attempt to reach a given critical state. First, it computes the shortest path from the initial state to a critical state via the function *Shortest-Path*$(ISDA, z \in Q_E^{ISDA})$ such that $I_G(z) \subseteq X_{crit}$ (strong attack) or $I_G(z) \cap X_{crit} \neq \emptyset$ (weak attack). The second step is to expand the function, based on the shortest path, in order to satisfy the admissibility condition.

Example 22 *Let us extract an interruptible f_A from the ISDA shown in Fig. 6(a). The shortest path starting from the initial state $(0, A)$ to E-state $(2, A)$ goes through E-states $(0, A), (1, B)$ and $(1, C)$. Based on this path f_A is defined, for example $f_A(\epsilon, a) = \{a, ab_i\}$. However, defining f_A solely based on this path makes it inadmissible. The string ab is defined in $\mathcal{L}(S_A/G)$ but $f_A(a, b)$ is undefined. For this reason, we have to expand the function f_A after defining it for the shortest path. In this example, we first expand it for $f_A(a, b) = \{b\}$, and later on for $f_A(ab, a) = \{a\}$ and $f_A(ab, c) = \{c\}$. Such extensions suffice to make f_A admissible.*

Remark: We presented a simple synthesis algorithm. Clearly, one could choose another strategy to extract an interruptible function from the ISDA. The important point here is that the ISDA provides a representation of the desired "solution space" for the synthesis problem. We are not focused on the synthesis of minimally invasive attack strategies as studied in (Su 2018), where minimally invasive means an attack strategy with the least number of edits to reach a critical state.

7 Other Attack Scenarios

In this section we present modifications to Algorithm 2 in order to compute similar structures as the ISDA for the remaining two attack functions investigated in this paper. We omit the respective synthesis algorithms since they are similar to Algorithm 3. Although the "expand path" function of Algorithm 3 changes for each attack function, such changes follows the assumption of each attack function.

7.1 Deterministic Unbounded Attacks

Different from the interruptible attack, in the deterministic unbounded (det-unb) attack case, we do not need to consider

Algorithm 3 Synthesis-f_A

Require: $ISDA, z \in Q_E^{ISDA}$ s.t. $I_G(z) \subseteq X_{crit}$ or $I_G(z) \cap X_{crit} \neq \emptyset$
Ensure: Encoded f_A in automaton F
1: $path \leftarrow$ Shortest-Path$(ISDA, z)$
2: $F \leftarrow$Expand-Path$(path, ISDA)$
3: **procedure** EXPAND-PATH$(path, ISDA)$
4: Queue $Q \leftarrow \emptyset$; $F \leftarrow \emptyset$
5: $F.X \leftarrow \{q_0\}$; $F.x_0 = q_0$
6: $F.\delta \leftarrow \emptyset$; $F.\Sigma = \Sigma \cup \Sigma_a^e$
7: **for all** $(q, e, f) \in path \wedge q$ is an E-State **do**
8: Add-to-F(q, e, f, Q, A)
9: **while** Q is not empty **do**
10: $q \leftarrow Q.dequeue()$
11: **for all** $(q, e, f) \in ISDA.h \wedge (q, e, f) \notin A.h$ **do**
12: **if** $e \in \Sigma_o \wedge \nexists f^*$ s.t. $(q, e_d, f^*) \in A.h$ **then**
13: Add-to-F(q, e, f, Q, A)
14: **else if** $e \in \Sigma_a^d \wedge (q, \mathcal{M}(e), f^*) \notin ISDA.h$ **then**
15: Add-to-F(q, e, f, Q, A)
16: **procedure** ADD-TO-F(q, e, f, Q, A)
17: $F.\delta \leftarrow F.\delta \cup \{(q, e, f)\}$
18: **if** $f \notin Q$ **then**
19: $Q.enqueue(f)$
20: $F.X \leftarrow F.X \cup \{f\}$

that the system may interrupt during an attack insertion. The attacker can insert events, possibly an arbitrarily long string in fact, before the system reacts. As consequence, the pruned IDA for the det-unb attack is not necessarily race free.

Algorithm 2 prunes the AIDA enforcing it to be race free (Step 2.2); however this condition needs to be relaxed for the det-unb case, resulting in Algorithm 4. Step 2.1 is also modified since we also need to relax the controllability condition. Specifically, Step 2.1 relaxes the controllability condition because the attacker "races" with the system at states that violate this condition.[2] Algorithm 4 flags states that violate the controllability condition to later analyze if they need to be pruned or not. Note that once a state is flagged, it remains flagged throughout the algorithm. Step 2.2 is divided into three steps. First, deadlocks created by the pruning process are deleted. Second, we flag all states violating the race-free condition. Then, the transition function is updated based on the flagged states. This update is such that only insertions transitions are possible from flagged states, since the attack will not wait for a reaction of the system. We can adapt Definition 16 to prune the AIDA for the case of det-unb attacks by considering the modification of Step 2.1 and Step 2.2, as presented in Algorithm 4. We name the resulting stealthy IDA as the USDA, for *Unbounded Stealthy Deceptive Attack* structure. Versions of Lemmas 18, 19, and Theorem 20 are created for this specific attacker.

[2] It is interesting to mention the similarity of "racing" in our work with the work on supervisory control with forced events (Golaszewski and Ramadge 1987).

Algorithm 4 Det-Unb Modification

1: **Step 2.1** Flag all $a \in A_i$ s.t. $\Gamma_A(a) \cap E_{uc} \not\subseteq \Gamma_{H_i}(a)$
2: **Step 2.2**
3: **Step 2.2.1** $A_i^* = \{a \in A_i | \, \Gamma_{H_i}\big(a\big) = \emptyset \Rightarrow \Gamma_A(a) = \emptyset\}$
4: **Step 2.2.2** Flag all $a \in A_i^*$ s.t. $\big(e \in \Sigma_o \wedge e \in \Gamma_A(a)\big) \Rightarrow \big(\{e, e_d\} \cap \Gamma_{H_i}(a) = \emptyset\big)$
5: **Step 2.2.3** For $a \in A_i^*$ and $e \in E$

$$g_i'(a,e) = \begin{cases} g_i(a,e) & \text{if } e \in (\Sigma_a^i \cup \Gamma) \wedge \\ & g_i(a,e)! \\ g_i(a,e) & \text{if } e \in (\Sigma_a^d \cup \Sigma_o) \wedge g_i(a,e)! \wedge \\ & a \text{ not flagged} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Lemma 23 *A deterministic unbounded attack function f_A is embedded in the USDA if it satisfies conditions (1) and (2) from Problem 3.*

Lemma 24 *If a det-unb attack function f_A is synthesized from USDA, then the stealthy condition of Problem 3 is satisfied.*

Theorem 25 *The USDA embeds all possible det-unb stealthy insertion-deletion attack strategies with respect to Σ_a, R and G.*

Example 26 *As we did for the ISDA, we also show the USDA structure for the AIDA in Figure 5. Figure 6(b) shows the USDA, where the only deleted state is $(0, dead)$. Note that decision b_d was deleted in the ISDA at state $(1, B)$, however it is maintained in the USDA. It is only possible since we know that decision b_i can be played before the execution of event a at state $(3, B)$.*

7.2 Deterministic Bounded Attacks

The AIDA structure is general enough for the interruptible and the unbounded attack scenarios; however, as constructed, it does not capture the bound in the case of deterministic bounded attacks (or det-bounded case). We now present a simple mechanism in order to computed a bounded version of the AIDA, that we term BAIDA. (The $||$ operation is the standard parallel composition of automata.)

Definition 27 *Given the AIDA constructed by Algorithm 1 and the automaton shown in Figure 7, the BAIDA is defined as $BAIDA = AIDA||G_{bound}$. (For the purpose of $||$, the AIDA is treated as an automaton.)*

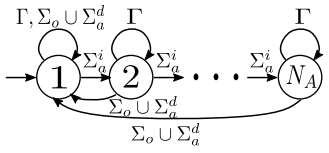


Fig. 7. G_{bound}

The det-bounded attack case has similar conditions as the previously-discussed det-unb case, however, it can only perform bounded modifications. We need to take into account how many modifications the attacker has already performed at a given E-state. Therefore, synthesis algorithms for det-bounded attacks must use the BAIDA, as in Definition 27.

The BAIDA, as the AIDA previously, has to be pruned. We only show the modification of Step 2 for Algorithm 2, resulting in Algorithm 5. Bounded attacks include features

Algorithm 5 Det-Bounded Modification

1: **Step 2.1**
2: **Step 2.1.1** Flag all $(a, n) \in A_i$ s.t. $\Gamma_A(a) \cap E_{uc} \not\subseteq \Gamma_{H_i}((a, n)) \wedge n < N_A$
3: **Step 2.1.2** $A_i' = \{(a, n) \in A_i | \; n = N_A \Rightarrow \Gamma_A(a) \cap E_{uc} \subseteq \Gamma_{H_i}((a, n))\}$
4: **Step 2.2**
5: **Step 2.2.1** $A_i'' = \{(a, n) \in A_i' | \; \Gamma_{H_i}\big((a, n)\big) = \emptyset \Rightarrow \Gamma_A(a) = \emptyset\}$
6: **Step 2.2.2** $A_i^* = \{(a, n) \in A_i'' | \, n = N_a \wedge \, e \in \Sigma_a \wedge e \in \Gamma_A(a) \Rightarrow \big(e \in \Gamma_{H_i}((a, n)) \vee e_d \in \Gamma_{H_i}((a, n))\big)\}$
7: **Step 2.2.3** Flag all $(a, n) \in A_i^*$ s.t. $n < N_a$, $(e \in \Sigma_o \wedge e \in \Gamma_A(a)) \Rightarrow \big(\{e, e_d\} \cap \Gamma_{H_i}((a, n)) = \emptyset\big)$
8: **Step 2.2.4** For $(a, n) \in A_i^*$ and $e \in E$

$$g_i'((a,n),e) = \begin{cases} g_i((a,n),e) & \text{if } e \in (\Sigma_a^i \cup \Gamma) \wedge \\ & g_i((a,n),e)! \\ g_i((a,n),e) & \text{if } e \in (\Sigma_a^d \cup \Sigma_o) \wedge \\ & g_i((a,n),e)! \wedge \\ & (a,n) \text{ is not flagged} \\ \text{undefined} & \text{otherwise} \end{cases}$$

from both unbounded and interruptible attacks. E-states that have reached the maximum allowed modification behave as E-states in the interruptible case, in other words, they must satisfy the race-free condition. On the contrary, E-states that have not reached the maximum allowed editions are similar to E-states in the unbounded scenario. Consequently, Step 2.1 and Step 2.2 are a combination of the corresponding steps in the previous cases. A similar structure as the ISDA and the USDA can be introduced, however we omit such definition given that it would follow the same steps as in the previous cases. The same comment holds for the lemmas and the theorems introduced in the previous cases. The details are left for the readers to work out.

8 Conclusion

We have considered the supervisory layer of feedback control systems, where sensor readings may be compromised by an attacker in the form of insertions and deletions. In this context, we have formulated the problem of synthesizing stealthy sensor deception attacks, that can cause damage to the system without detection by an existing supervisor. We

defined the attacker as a nondeterministic edit function that reacts to the plant's output and its previous editions in a way that guarantees stealthiness of its attack. We introduced three different types of attacks, based on the interaction between the system and the attacker.

Our solution procedure is game-based and relies on the construction of a discrete structure called the AIDA, which is used to solve the synthesis problem for each attack scenario. The AIDA captures the game between the environment (i.e., system and attacker) and the given supervisor. It embeds all valid actions of the attacker. Based on the AIDA, we specified a pruning procedure for each attack type, thereby constructing stealthy structures denoted as the ISDA and the USDA. Based on each type of stealthy structure, we can synthesize, if it exists, an attack function that leads the system to unsafe critical states without detection, for the corresponding attack scenario.

In the future, we plan to investigate how to modify a supervisor that is susceptible to stealthy deception attacks. We also plan to study the problem of directly designing supervisors that enforce safety and liveness specifications and at the same time are robust to deception attacks.

References

Alves, M. V. S., Joao Carlos Basilio, Antonio Eduardo C. da Cunha, Lilian Kawakami Carvalho and Marcos Vicente Moreira (2014). Robust supervisory control against intermittent loss of observations. In '12th IFAC International Workshop on Discrete Event Systems'. pp. 294 – 299.

Amin, S., X. Litrico, S. Sastry and A. M. Bayen (2013). 'Cyber security of water scada systems - Part I: Analysis and experimentation of stealthy deception attacks'. *IEEE Transactions on Control Systems Technology* **21**(5), 1963–1970.

Cardenas, A. A., S. Amin and S. Sastry (2008). Secure control: Towards survivable cyber-physical systems. In '2008 The 28th International Conference on Distributed Computing Systems Workshops'. pp. 495–500.

Carvalho, L. K., Yi-Chin Wu, Raymond Kwong and Stéphane Lafortune (2018). 'Detection and mitigation of classes of attacks in supervisory control systems'. *Automatica* **97**, 121 – 133.

Cassandras, C. G. and Stéphane Lafortune (2008). *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc.. Secaucus, NJ, USA.

Cassez, F., Jérémy Dubreil and Hervé Marchand (2012). 'Synthesis of opaque systems with static and dynamic masks'. *Formal Methods in System Design* **40**(1), 88–115.

Checkoway, S., Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner and Tadayoshi Kohno (2011). Comprehensive experimental analyses of automotive attack surfaces. In 'Proceedings of the 20th USENIX Conference on Security'. SEC'11. USENIX Association. Berkeley, CA, USA. pp. 6–6.

Farwell, J. P. and Rafal Rohozinski (2011). 'Stuxnet and the future of cyber war'. *Survival* **53**(1), 23–40.

Golaszewski, C. H. and P. J. Ramadge (1987). Control of discrete event processes with forced events. In '26th IEEE Conference on Decision and Control'. Vol. 26. pp. 247–251.

Kerns, A. J., Daniel P. Shepard, Jahshan A. Bhatti and Todd E. Humphreys (2014). 'Unmanned aircraft capture and control via GPS spoofing'. *J. Field Robot.* **31**(4), 617–636.

Lin, F. (2011). 'Opacity of discrete event systems and its applications'. *Automatica* **47**(3), 496–503.

Lin, F. (2014). 'Control of networked discrete event systems: Dealing with communication delays and losses'. *SIAM Journal on Control and Optimization* **52**(2), 1276–1298.

Meira-Góes, R., E. Kang, R. Kwong and S. Lafortune (2017). Stealthy deception attacks for cyber-physical systems. In '2017 IEEE 56th Annual Conference on Decision and Control (CDC)'. pp. 4224–4230.

Meira-Góes, R., Eunsuk Kang, Raymond Kwong and S. Lafortune (2020). 'Synthesis of sensor deception attacks at the supervisory layer of cyber-physical systems'. *ArXiv*.

Meira-Góes, R., S. Lafortune and H. Marchand (2019). 'Synthesis of supervisors robust against sensor deception attacks'. *submitted to IEEE Transactions on Automatic Control*.

Paoli, A. and StéPhane Lafortune (2005). 'Safe diagnosability for fault-tolerant supervision of discrete-event systems'. *Automatica* **41**(8), 1335–1347.

Paoli, A., Matteo Sartini and Stéphane Lafortune (2011). 'Active fault tolerant control of discrete event systems using online diagnostics'. *Automatica* **47**(4), 639–649.

Ramadge, P. J. and W. M. Wonham (1987). 'Supervisory control of a class of discrete event processes'. *SIAM Journal on Control and Optimization* **25**(1), 206–230.

Rohloff, K. (2012). Bounded sensor failure tolerant supervisory control. In '11th IFAC International Workshop on Discrete Event Systems'. pp. 272 – 277.

Saboori, A. and C. N. Hadjicostis (2007). Notions of security and opacity in discrete event systems. In '46th IEEE Conference on Decision and Control'. pp. 5056–5061.

Su, R. (2018). 'Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations'. *Automatica* **94**, 35–44.

Teixeira, A., Daniel Pérez, Henrik Sandberg and Karl Henrik Johansson (2012). Attack models and scenarios for networked control systems. In 'Proceedings of the 1st International Conference on High Confidence Networked Systems'. HiCoNS '12. ACM. New York, NY, USA. pp. 55–64.

Thorsley, D. and D. Teneketzis (2006). Intrusion detection in controlled discrete event systems. In 'Proceedings of the 45th IEEE Conference on Decision and Control'. pp. 6047–6054.

Wakaiki, M., Paulo Tabuada and João P. Hespanha (2018). 'Supervisory control of discrete-event systems under attacks'. *Dynamic Games and Applications*.

Weerakkody, S., Omur Ozel, Yilin Mo and Bruno Sinopoli (2019). 'Resilient control in cyber-physical systems: Countering uncertainty, constraints, and adversarial behavior'. *Foundations and Trends in Systems and Control* **7**(1-2), 1–252.

Wu, Y.-C., Vasumathi Raman, Blake C. Rawlings, Stéphane Lafortune and Sanjit A. Seshia (2018). 'Synthesis of obfuscation policies to ensure privacy and utility'. *Journal of Automated Reasoning* **60**(1), 107–131.

Xu, S. and R. Kumar (2009). Discrete event control under nondeterministic partial observation. In '2009 IEEE International Conference on Automation Science and Engineering'. pp. 127–132.

Yin, X. (2016). 'Supervisor synthesis for mealy automata with output functions: A model transformation approach'. *IEEE Transactions on Automatic Control*.

Yin, X. and S. Lafortune (2016*a*). 'Synthesis of maximally-permissive supervisors for the range control problem'. *IEEE Transactions on Automatic Control* **PP**(99), 1–1.

Yin, X. and S. Lafortune (2016*b*). 'A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems'. *IEEE Transactions on Automatic Control* **61**(8), 2140–2154.