Attack-Resilient Supervisory Control of Discrete-Event Systems

Yu Wang, Alper Kamil Bozkurt and Miroslav Pajic

***Abstract*—In this work, we study the problem of supervisory control of discrete-event systems (DES) in the presence of attacks that tamper with inputs and outputs of the plant. We consider a very general system setup as we focus on both deterministic and nondeterministic plants that we model as finite state transducers (FSTs); this also covers the conventional approach to modeling DES as deterministic finite automata. Furthermore, we cover a wide class of attacks that can nondeterministically add, remove, or rewrite a sensing and/or actuation word to any word from predefined regular languages, and show how such attacks can be modeled by nondeterministic FSTs; we also present how the use of FSTs facilitates modeling realistic (and very complex) attacks, as well as provides the foundation for design of attack-resilient supervisory controllers. Specifically, we first consider the supervisory control problem for deterministic plants with attacks (i) only on their sensors, (ii) only on their actuators, and (iii) both on their sensors and actuators. For each case, we develop new conditions for controllability in the presence of attacks, as well as synthesizing algorithms to obtain FST-based description of such attack-resilient supervisors. A derived resilient controller provides a set of all safe control words that can keep the plant work desirably even in the presence of corrupted observation and/or if the control words are subjected to actuation attacks. Then, we extend the controllability theorems and the supervisor synthesizing algorithms to nondeterministic plants that satisfy a nonblocking condition. Finally, we illustrate applicability of our methodology on several examples and numerical case-studies.**

I. Introduction

Security has been an major concern in many Cyber-Physical Systems (CPS) domains, such as autonomous systems [29, 25], smart grid [5], medical devices [14], industrial automation [9, 7], and distributed control systems [36]. In contested scenarios, several components of such systems including the controllers, the plants and their physical environment, as well as communication between system components, may be subject to simultaneous cyber and/or physical attacks. To allow for the development of security-critical CPS capable of operating even in the presence of malicious activity, there has recently been an effort of developing theory and tools for analyzing and synthesizing attack-resilient CPS (e.g., see [33, 18, 24, 10, 26, 15] and references therein). The goal has been to provide strong performance guarantees for CPS using partial knowledge about the possible attacks [11, 3], as opposed to simple tolerance of random failures or robustness under small disturbances.

This work was supported in part by the National Science Foundation (NSF) Grant CNS-1652544, as well as by the Office of Naval Research (ONR) under agreements number N00014-17-1-2012 and N00014-17-1-2504. Some of the preliminary results have appeared in [37].

The authors are with the Department of Electrical & Computer Engineering, Duke University, Durham, NC 27707, USA. Email: yu.wang094@duke.edu, alper.bozkurt@duke.edu, miroslav.pajic@duke.edu.

In this work, we tackle the attack-resilient CPS problem from the prospective of the supervisory control of discrete-event systems [6], where the behavior of a discrete-time finite-state plant is controlled by a supervisor in closed-loop. Between the plant and the supervisor, we consider a wide class of regular-rewriting attacks that can compromise the sensors and actuators of the plant, as well as their communication with the supervisor. These attackers have the ability to inject, delete, and replace events/symbols according to certain prespecified rules. Such attack model generalizes the attacks studied previously in the context of discrete-event systems [35, 32, 12, 4], as well as captures new attacks which were previously not considered in such context (e.g., replay attacks [19, 18]).

Mathematically, the regularly-rewriting attacks define a regular relation between the input and output languages of the attackers [28, 8], and can be compactly realized by finite state transducers (FSTs) or nondeterministic Mealy machines. These finite-state models, especially deterministic ones, have found applications in a wide-range of application domains, such as applications in speech recognition [20, 23, 21, 22]. In this work, we exploit the nondeterminism in FSTs, which is crucial in capturing possible attacker behaviors, as well as design resilient control for all cases (including the worst-case).

Another advantage of using FSTs to model attacks is the availability of a library of mathematically rigorous and computationally feasible operations. For example, FSTs are closed under inversion and composition. Thus, complex system configurations and multiple attack-points/attackers can usually be simplified to a few basic configurations. Furthermore, as we show in this work, modeling the attackers as FSTs facilitates capturing of constraints imposed on the attacker by the system design and underlying platform, such as how frequently attacks may be enabled when security primitives (e.g., message authentication) are only intermittently employed, as in [15, 16].

Besides its focus on attack-resiliency, this work differs from the conventional supervisory control of deterministic discrete-event systems (DES) with uncontrollable and unobservable symbols in the following two aspects. First, we consider a more general supervisor model, where we model the supervisor also as an FST; note that this also covers supervisor modeling as automata, which is more a conventional approach for DES. This gives the supervisor the ability to rewrite, in addition to regulating, in order to counter the attacks and improve system resiliency; as will be discussed (in Section V), conventionally supervisors may be feeble under some regular-rewriting attacks. In addition, we consider a more general class of plants modeled by FSTs instead of deterministic automata. FSTs can generate output symbols nondeterministically from

predefined regular languages upon receiving a possibly-empty input word, instead of simply dropping unobservable symbols and autonomously triggering uncontrollable symbols, and are more versatile in modeling plants with complex actuating and sensing architectures.

As illustrated by Figure 1, we are mainly concerned with three configurations, covering attacks on sensors and/or actuators of the plant. They capture most forms of malicious activity in CPS, possibly resulting from network-based attacks that corrupt the data communicated between the plant and supervisor (e.g., as in [31, 33]), as well as non-invasive attacks that affect the environment of the plant (e.g., such as GPS spoofing attacks on autonomous vehicles [27, 38] or Anti-lock Braking Systems (ABS) in cars [30]). To simplify our presentation, even if only the communication between sensors and/or actuators and the supervisor is compromised, and not sensors (actuators) themselves, if a sensor's (actuator's) information delivered to (from) the supervisor is compromised, we will refer to the sensor (actuator) as being under attack.

By attack-resilient supervisory control we refer to a supervisor that even in the presence of attacks is able to constraint the behavior of the controlled plant to a set of desired ones; such desired behaviors are specified as a regular language. Specifically, our focus is on deriving conditions for attack-resiliency, which we capture as conditions for controllability under attack, as well as methods to design such attack-resilient supervisors. Compared to existing studies on supervisory control of DES in the presence of attacks [35, 32, 12, 4], this work is different in the following three aspects. First, we consider a more general class of attacks modeled by FSTs than the replacement-removal and the injection-deletion attacks in [35], actuator enabling/disabling and sensor erasure/injection attacks in [4], and the replacement by bounded length attack in [32]. We show how additional attacks can be modeled as FSTs, as well as how FSTs allow for capturing more complex attack scenarios. For example, consider coordinated attacks on sensors or actuators of the plant, or the communication network, capturing different 'point-of-entry' for the attack vectors – e.g., false data injection via sensor spoofing on part of plant sensors [34, 29] in addition to Denial-of-Service attacks on transmitted measurements from other sensors. In such cases, while each attack may be modeled by an individual FST, the coordinated attacks from multiple attack points shown in Figure 1 can be captured by composition of such attack FSTs.

Second, the fact that supervisor is also modeled by FSTs, is giving it the extra ability to rewrite. These supervisors generate control words that can keep the plant work desirably from potentially corrupted observations from the plant, even if these control words are subject to actuation attacks. Note that the execution of the supervisors is possibly nondeterministic for a given observation, yielding the set of all feasible control words under that observation. Consequently, controllability theorems different from [35, 32, 12, 4] are derived.

Third, both attacks on the sensors and actuators of the plant are considered simultaneously, while [35] only studies attacks on the output, and [4] considers attacks on the sensors and actuators individually. Finally, we consider a more general class of plants modeled by nondeterministic FSTs instead of deterministic automata with uncontrollable and unobservable symbols as in [35, 32, 12, 4].

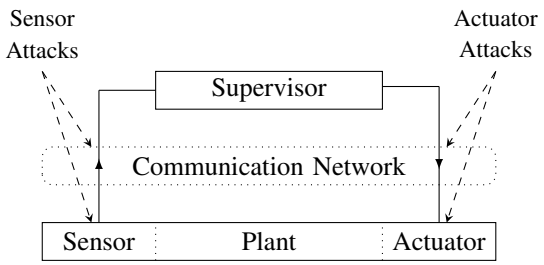


Fig. 1: Supervisory control under attacks on sensors, actuators as well as the communication between the sensors, controller and actuators. In this work, we propose the use of finite-state transducers to capture a very general class of such attacks.

In this work, we introduce a new set of controllability conditions and synthesizing algorithms for attack-resilient supervisors used for supervisory control of DES where attacks are modeled by FSTs. For actuator attacks, the controllability theorem derived here generalizes the standard controllability theorem [6]. As opposed to previous studies [6, 35, 32, 4], we show that when using FSTs as the supervisor, the design can be performed separately for actuator and sensor attacks. In addition, we show that sensor attacks have no effect on the controllability of desired languages for *deterministic* plants, even in the presence of actuator attacks. This is caused by the fact that the supervisor is model-based and can encode a copy of the plant's model into the control policy, unlike traditional supervisors that can only allow/disallow symbols. For nondeterministic plants, however, we show that the sensor attacks may influence the controllability by blocking control words of the supervisor.

To highlight impacts of different attack points, we present our results progressively by considering first the supervisory control problem for deterministic plants with (i) attacks only on their sensors, (ii) attacks only on their actuators, and (iii) attacks on both the sensors and actuators For deterministic plants, we show that sensor attacks can be countered by a supervisor derived by the serial composition of the inversion of the attack model and a model of the desired language. Actuator attacks on deterministic plants can be partly countered by the supervisor serially composing a model of the desired language and the inversion of the attacker. The exact controllability is achieved if the desired language is invariant under the attack. For simultaneous sensor and actuator attacks, an attack-resilient supervisor can be derived by serially composing the supervisors for the above two cases. Finally, we extend the results for deterministic plants, to cover scenarios with nondeterministic plants, under nonblocking conditions.

The rest of the paper is organized as follows. The preliminaries are provided in Section II. The considered system model and problem formulation are presented in Section III. Using FSTs to model attacks is demonstrated in Section IV. We study the supervisor control problem for deterministic plants in the presence of only sensor attacks in Section V, with only actuator attacks in Section VI, and in the presence of both

sensor and actuator attacks in Section VII. In Section VIII, we present extension of these results when nondeterministic plants are considered. Finally, we illustrate applicability of our attack-resilient supervisory control framework in Section IX, before providing concluding remarks in Section X.

II. Preliminaries

The empty string is denoted by ε. A finite-length sequence of symbols taken from a given finite set is called a *word*. A set of words is called a *language* of the symbols. The cardinality and the power set of a set $\mathbf{I}$ are denoted by $|\mathbf{I}|$ and $2^{\mathbf{I}}$, respectively. For two sets $\mathbf{I}$ and $\mathbf{O}$, let $\mathbf{I}\backslash\mathbf{O} = \{\mathsf{i} \in \mathbf{I} \mid \mathsf{i} \notin \mathbf{O}\}$. For $n \in \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, let $[n] = \{1, .., n\}$. For a word $I = \mathsf{i}_1\mathsf{i}_2 \dots \mathsf{i}_n$, we call $\mathsf{i}_1\mathsf{i}_2 \dots \mathsf{i}_k$, with $k \leq n$, a prefix of I. For a language L, its prefix-closure is defined by $\overline{L} = \{I \mid I \text{ is a prefix of J}, J \in L\}$. The language L is *prefix-closed* if $L = \overline{L}$. Also, we adopt the following convention on generating regular expressions: a superscript * means repeating a symbol or a set of symbols finitely many times, and a comma means "or".

A *relation* $\mathcal{R}$ between two sets $\mathbf{I}$ and $\mathbf{O}$ is a set $\mathcal{R} \subseteq \mathbf{I} \times \mathbf{O}$. For $\mathsf{i} \in \mathbf{I}$, let $\mathcal{R}(\mathsf{i}) = \mathcal{R}(\mathsf{i}, \cdot) = \{\mathsf{o} \in \mathbf{O} \mid (\mathsf{i}, \mathsf{o}) \in \mathcal{R}\}$. The relation $\mathcal{R}(\mathsf{i})$ is a *partial function* for the input a if $|\mathcal{R}(\mathsf{i})| \leq 1$, for any $\mathsf{i} \in \mathbf{I}$. More generally, for $\mathbf{I}' \subseteq \mathbf{I}$, while slightly abusing the notation we define $\mathcal{R}(\mathbf{I}') = \mathcal{R}(\mathbf{I}', \cdot) = \{\mathsf{o} \in \mathbf{O} \mid (\mathsf{i}', \mathsf{o}) \in \mathcal{R}, \mathsf{i}' \in \mathbf{I}'\}$. Thus, $\mathcal{R}(\cdot)$ defines a function $2^{\mathbf{I}} \to 2^{\mathbf{O}}$. For relation $\mathcal{R} \subseteq \mathbf{I} \times \mathbf{O}$, its inversion is defined by $\mathcal{R}^{-1} = \{(\mathsf{o}, \mathsf{i}) \in \mathbf{O} \times \mathbf{I} \mid (\mathsf{i}, \mathsf{o}) \in \mathcal{R}\}$. Finally, for two relations $\mathcal{R} \subseteq \mathbf{I} \times \mathbf{O}$ and $\mathcal{R}' \subseteq \mathbf{I}' \times \mathbf{O}'$, their (serial) composition is defined by $\mathcal{R} \circ \mathcal{R}' = \{(\mathsf{i}, \mathsf{o}') \in \mathbf{I} \times \mathbf{O}' \mid \exists \mathsf{o} \in \mathbf{O} \cap \mathbf{I}' : (\mathsf{i}, \mathsf{o}) \in \mathcal{R} \wedge (\mathsf{i}', \mathsf{o}') \in \mathcal{R}'\}$.

A. Finite State Transducers and Regular Relations

Finite State Transducers extend (nondeterministic) automata by generating a sequence of outputs nondeterministically during execution, by augmenting each transition with output.

Definition 1 (Finite State Transducer)**.** *A (normalized) finite state transducer (FST) is a tuple* $\mathcal{A} = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{I}, \mathbf{O}, \mathsf{Trans}, \mathsf{S}_{\text{final}})$ *where*

- S *is a finite set of* states*;*
- $\mathsf{s}_{\text{init}} \in \mathsf{S}$ *is the* initial state*;*
- $\mathbf{I}$ *is a finite set of* inputs*;*
- $\mathbf{O}$ *is a finite set of* outputs*;*
- $\mathsf{Trans} \subseteq \mathsf{S} \times \big(\mathbf{I} \cup \{\varepsilon\}\big) \times \big(\mathbf{O} \cup \{\varepsilon\}\big) \times \mathsf{S}$ *is a transition relation;*
- $\mathsf{S}_{\text{final}} \subseteq \mathsf{S}$ *is a finite set of* final states*.*

The FST is deterministic if $\mathsf{Trans}(\mathsf{s}, \mathsf{i}, \cdot, \cdot)$ *is a partial function for the input* (s, i)*. Specially, nondeterministic automata are treated as special FSTs with identical input and output.*

A sequence $(\mathsf{s}_{\text{init}}, \mathsf{i}_1, \mathsf{o}_1, \mathsf{s}_1)(\mathsf{s}_1, \mathsf{i}_2, \mathsf{o}_2, \mathsf{s}_2)...(\mathsf{s}_{n-1}, \mathsf{i}_n, \mathsf{o}_n, \mathsf{s}_n)$ is called an *execution* of the FST $\mathcal{A}$, if $(\mathsf{s}_{i-1}, \mathsf{i}_i, \mathsf{o}_i, \mathsf{s}_i) \subseteq \mathsf{Trans}$ for $i \in [n]$, with $\mathsf{s}_0 = \mathsf{s}_{\text{init}}$. Note that this defines a regular relation $\mathcal{R}_{\mathcal{A}}$ *modeled* by the FST $\mathcal{A}$ by letting $\mathcal{R}_{\mathcal{A}}$ only contain such pairs of $(\mathsf{i}_1 \dots \mathsf{i}_n, \mathsf{o}_1 \dots \mathsf{o}_n)$. On the other hand, a relation $\mathcal{R} \subseteq \mathbf{I}^* \times \mathbf{O}^*$ is *regular*, only if it is modeled by FSTs. Finally, the *input language* and *output language* of $\mathcal{A}$ are defined by $L_{\text{in}}(\mathcal{A}) = \mathcal{R}_{\mathcal{A}}^{-1}(\mathbf{O}^*)$ and $L_{\text{out}}(\mathcal{A}) = \mathcal{R}_{\mathcal{A}}(\mathbf{I}^*)$, respectively.

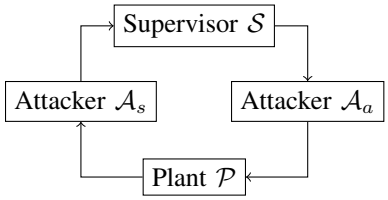


Fig. 2: Controllability under actuator and sensor attacks.

Remark 1 (Normalization)**.** *The expressiveness of normalized FSTs is the same as general FSTs, whose transitions are labeled by regular languages of the input and output symbols. For a general FST, we can always find a normalized FST modeling the same regular relation by normalization [13, 22].*

Remark 2 (Complexity)**.** *Normalized FSTs can be viewed as nondeterministic automata with labels in* $(\mathbf{I} \cup \{\varepsilon\}) \times (\mathbf{O} \cup \{\varepsilon\})$*. Accordingly, a regular relation on* $\mathbf{I}^* \times \mathbf{O}^*$ *can be viewed as a regular language of* $\mathbf{I} \times \mathbf{O}$*. Roughly, the computational complexity for FST is the same as nondeterministic automata with exceptions such as determinization of FSTs [8].*

III. System Model and Problem Formulation

In this work, we consider the problem of supervisory control of discrete event systems subject to attacks both on the plant's sensors and actuators. Specifically, we focus on the setup illustrated in Figure 2, where the supervisor $\mathcal{S}$ controls the behavior of the plant $\mathcal{P}$ by observing the symbols that the plant generates and then sending the possible control symbols back to the plant. Physically, this is usually performed by a number of sensors and actuators on the plant. In reality, these sensors and actuators, as well as their communication with the supervisor, may be simultaneously compromised by multiple coordinated malicious attacks that have the ability of inject, remove, or replace symbols in both the control (i.e., actuation commands) and the observation (i.e., sensor measurements).

The overall effects of these attacks can be represented by two attackers $\mathcal{A}_a$ and $\mathcal{A}_s$ on the actuators and sensors of the plant, respectively. The first challenge that needs to be addressed is a suitable model that can capture attacker's impact on the system. In this work, we propose to model the attack behaviors $\mathcal{A}_a$ and $\mathcal{A}_s$ as FSTs. With FTSs, $\mathcal{A}_a$ and $\mathcal{A}_s$ can *regularly rewrite* an acceptable word, i.e., replace a symbol *nondeterministically* with an arbitrary word taken from some predefined regular language, including injection, replacement and deletion; in Section IV we show how this allow us to capture all reported attacks on supervisory-control systems.

It is important to highlight that we do not assume to know what actions the attacker may perform. Rather, the FST models employ nondeterminism to capture all possible actions of the attacker for a specific set of compromised resources (e.g., sensors, actuators), as well as all potential limitations imposed on the attacker's actions by the system design (e.g., the use of cryptographic primitives on some communication messages to prevent false-data injecting attacks over the network).

Furthermore, we consider systems where the supervisor's behavior can also be captured by an FST, since the power of rewriting symbols is essential to counter the attacks and

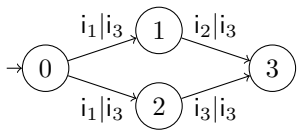


Fig. 3: Example of a nondeterministic plant.

improve system resiliency. The supervisor receives possibly corrupted observations (i.e., sensor measurements) from the plant, and generates control words (i.e., actuation commands) that can keep the plant work desirably even if the system is subject to actuation attacks. In addition, we do not restrict the supervisor's models to deterministic FST; the execution of the supervisor can be nondeterministic for a given corrupted observation, yielding the set of all such safe control words under that observation. In implementation, the nondeterminism in the supervisors can be eliminated by choosing one of the feasible controls either arbitrarily or by certain merit (see Remark 4). Note that modeling supervisors with FSTs generalizes the models used for conventional supervisors for DES that only regulate the plant [6, 35, 32, 4, 37], as FSTs can be used to model.

Finally, we also assume that the plant can be modeled as an FST where the output can be nondeterministic and different from the input. Using such FSTs to model the plants generalizes deterministic automata that are traditionally used to model DES; FSTs are versatile in modeling discrete plants with complex actuating and sensing architectures, compared to conventional plant models as deterministic automata that simply drop unobservable symbols and autonomously trigger uncontrollable symbols.

To simplify our discussion, without lost of generality, we assume that the supervisor, plant, and attackers share the same set of symbols for both their inputs and outputs. The closed-loop system shown in Figure 2 is driven and clocked by the transitions of the plant — at each time instance, the plant sends a fragment of symbols as output, which is subject to the nondeterministic revisions by the attacker and the supervisor, before returning to the input of the plant. Then, the plant makes state transitions according to the pair of input and output, which we assume are always allowed by the model of the plant and will be discussed later in details, and start the next round. The plant can stop upon entering a *final* state.

The nondeterminism in the plant brings an extra challenge for supervisory control. When the plant is deterministic, the control can be executed without blocking. However, blocking may happen in nondeterministic plants. For example, consider the nondeterministic plant shown in Figure 3 and the desired control $\mathsf{i}_1\mathsf{i}_2$. The plant may transit from the state 0 to the state 2 upon receiving i_1, thus blocking the next symbol i_2. We will discuss this issue in detail in Section VIII.

A. Problem Formulation

We consider the problem of designing a supervisor $\mathcal{S}$ that constraints the behavior of the plant $\mathcal{P}$ to certain desired ones, even in the presence of malicious activity on the plant sensors and actuators, captured by the FSTs $\mathcal{A}_s$ and $\mathcal{A}_a$. The desired behavior can be suitably described by a desired *regular* language $\mathscr{K} \subseteq L_{\rm in}(\mathcal{P})$, as formally stated in Definition 2. Specifically, our goal is to develop conditions for resiliency against attacks, which are captured as conditions for controllability in the presence of attacks, as well as methods to synthesize such supervisors. Note that these controllability conditions and supervisor synthesizing algorithms extend to nondeterministic plants with an additional nonblocking condition, and will be discussed in Section VIII.

Definition 2. *The supervisor $\mathcal{S}$ weakly controls the deterministic plant $\mathcal{P}$ to the desired language $\mathscr{K}$, in the presence of actuator and sensor attacks $\mathcal{A}_a$ and $\mathcal{A}_s$ if it holds that*

$$\mathscr{K} \subseteq_{\min} L_{\rm in}(\mathcal{P}|\mathcal{S}, \mathcal{A}_a, \mathcal{A}_s). \tag{1}$$

Here, $L_{\rm in}(\mathcal{P}|\mathcal{S}, \mathcal{A}_a, \mathcal{A}_s)$ denotes the language passed to the input of $\mathcal{P}$ in the closed-loop system with both actuator and sensor attacks, while $\subseteq_{\min}$ stands for minimal inclusion – i.e., any supervisor $\mathcal{S}'$ with $\mathscr{K} \subseteq L_{\rm in}(\mathcal{P}|\mathcal{S}', \mathcal{A}_a, \mathcal{A}_s)$ satisfies $L_{\rm in}(\mathcal{P}|\mathcal{S}, \mathcal{A}_a, \mathcal{A}_s) \subseteq L_{\rm in}(\mathcal{P}|\mathcal{S}', \mathcal{A}_a, \mathcal{A}_s)$. Furthermore, we say that the supervisor controls the plant $\mathcal{P}$ to the desired language $\mathscr{K}$ when the equality in (1) *holds.*

The languages passed to the input of $\mathcal{P}$ in the closed-loop setup with only actuator or sensor attacks are denoted by $L_{\rm in}(\mathcal{P}|\mathcal{S}, \mathcal{A}_a)$ and $L_{\rm in}(\mathcal{P}|\mathcal{S}, \mathcal{A}_s)$, respectively. Both controllability and weak controllability for these cases are similarly defined.

To simplify our presentation, we make the following assumptions (e.g., as in [6, 35, 32, 4, 37], For the plant $\mathcal{P}$, all states are final $\mathsf{S}_{\rm final} = \mathsf{S}$, i.e., both the sets of their inputs and outputs are prefix-closed. Accordingly, the desired language $\mathscr{K}$ is also prefix-closed $\mathscr{K} = \overline{\mathscr{K}}$ and regular. Furthermore, we assume that in the considered setup from Figure 2, the FSTs $\mathcal{A}_a, \mathcal{A}_s, \mathcal{S}$, and $\mathcal{P}$ always receive acceptable inputs[1] – i.e., the attackers $\mathcal{A}_a$ and $\mathcal{A}_s$ only try to affect/break the supervisory control by generating *undesired* instead of *unacceptable* words to the plant. Note that this is generally achievable with the proper use of FST models for the attacks $\mathcal{A}_a$ and $\mathcal{A}_s$, and the supervisor $\mathcal{S}$ (as we show in Section IV).

IV. Modeling Attacks with Finite State Transducers

In this section, we discuss several issues related to modeling attacks with FSTs and show how they generalize all existing attack models. Specifically, we show how FSTs can be used to capture all previously reported attacks on control of DES, as well as additional attacks and attack features. For example, FSTs can be used to capture constraints imposed on the attacks by the system design, as well as model finite-memory replay attacks, where the attacker records a finite-length of symbols and replays it repeatedly [19]; note however, that capturing replay attacks without a memory constraint (i.e., with infinite memory) is beyond the capability of the FST formalism.

[1]This is formally captured as Assumptions 1 and 2 in Sections V and VI, respectively.

A. Examples of Attack Modeling using FSTs

One of the contribution of this work is to propose the use of FSTs to model attacks in the supervisor control of DES. Thus, we start by showing that attack models proposed in previous works [35, 4] can be also represented by FSTs as shown in the following examples.

Example 1 (Projection/Deletion/Injection Attack)**.** *Let* $\mathbf{I}' \subseteq \mathbf{I}$*. The projection attack defined as*

$$\mathrm{Project}_{\mathbf{I}'}(\mathsf{i}) = \begin{cases} \mathsf{i}, & \textit{if} \quad \mathsf{i} \in \mathbf{I}' \\ \varepsilon, & \textit{otherwise,} \end{cases} \tag{2}$$

captures attacker's actions that result in removing all symbols that belong to $\mathbf{I} \setminus \mathbf{I}'$*. On the other hand, the (nondeterministic) deletion attack defined as*

$$\mathrm{Delete}_{\mathbf{I}'}(\mathsf{i}) = \begin{cases} \mathsf{i}, & \textit{if} \quad \mathsf{i} \in \mathbf{I}' \\ \varepsilon \textit{ or } i, & \textit{otherwise,} \end{cases} \tag{3}$$

extends the $\mathrm{Project}_{\mathbf{I}'}$ *attack as it captures that the attacker may (or may not) remove symbols from* $\mathbf{I} \setminus \mathbf{I}'$*; e.g., if* $\mathbf{I}' = \mathbf{I}$ *this model can be used to capture Denial-of-Service attacks [39] over the communication network. Finally, the (nondeterministic) injection attack defined as*

$$\mathrm{Inject}_{\mathbf{I}'}(\mathsf{i}) = (\mathbf{I}')^* \mathsf{i} (\mathbf{I}')^* \tag{4}$$

captures that a finite number of symbols from $\mathbf{I}'$ *can be added before and/or after the symbols. These attacks can be represented by FSTs as shown in Figure 4a, Figure 4b and Figure 4c, respectively.*

Example 2 (Replacement-removal Attack)**.** *A replacement-removal attack defined by the replacing-removing rule* $\phi : \mathbf{I} \to 2^{\mathbf{I} \cup \{\varepsilon\}}$ *is represented by an FST as shown in Figure 4d.*

Example 3 (Injection-removal Attack)**.** *Let* $\mathbf{I}' \subseteq \mathbf{I}$*. An injection-removal attack nondeterministically injects or removes symbols in* $\mathbf{I}'$ *from a word. This is modeled by the FST in Figure 4e.*

Example 4 (Finite-Memory Replay Attack)**.** *For systems with continuous-state dynamics, replay attacks have been modeled and studied in e.g., [17, 18]. On the other hand, for DES no such models have been introduced. In DES, a replay attack records a prefix of a word and replaces the rest with the repetitions of the recorded prefix, with the prefix size being bounded by the finite-memory capacity (i.e., size)* N*. For example, a replay attack recording a prefix of length up to* $N = 2$ *for any word of symbols* $\mathbf{I} = \{\mathsf{i}_1, \mathsf{i}_2\}$ *can be modeled by an FST as shown in Figure 4f. Note that the FST can be viewed as the parallel composition of two replay attacks recording prefixes of length* 1 *and* 2*, respectively.*

B. Composition of Finite State Transducers

One of the main advantages of using FSTs to model attacks on DES is their natural support for composition of multiple attacks that are captured with the corresponding FST models. With the general architecture from Figure 1, the system may be under a coordinated attack from multiple deployed attackers,

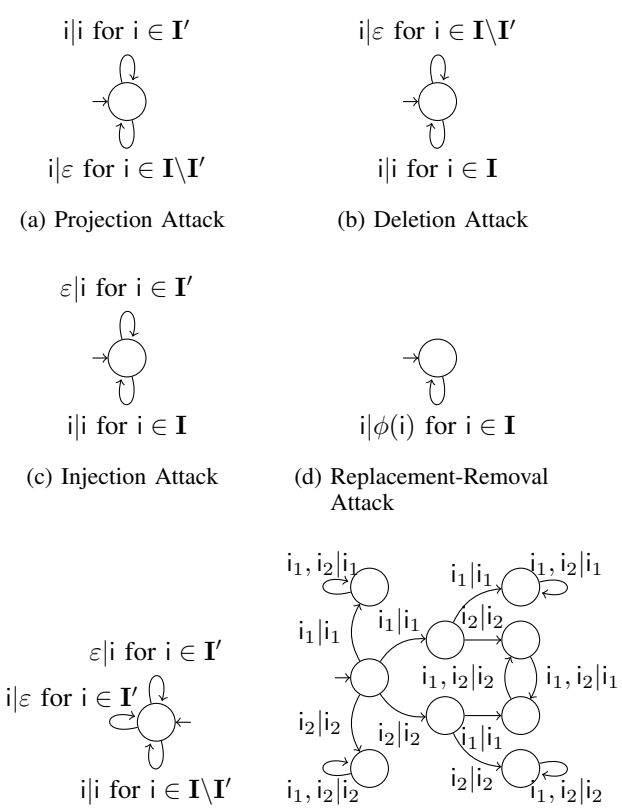
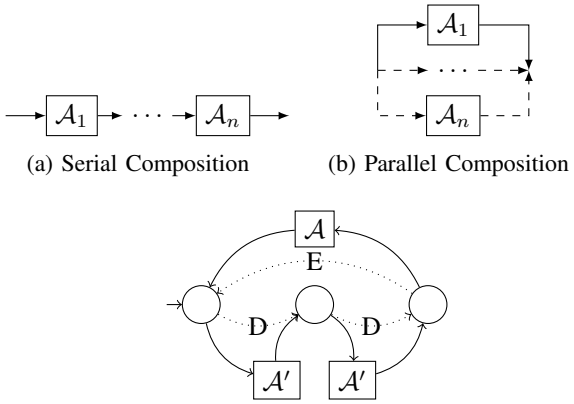


Fig. 4: FST realizations of different attack models.

capturing different 'point-of-entries' for the attack vectors on sensors/actuators and communication network – e.g., false data injection via sensor spoofing on part of plant sensors [34, 29] in coordination with Denial-of-Service attacks on transmitted measurements from other sensors. This attack configurations are illustrated in Figure 5a, and the overall effect is equivalent to the serial composition $\mathcal{A}_1 \circ \ldots \circ \mathcal{A}_n$. In addition, the attacker may decide on using a specific attack vector from a set of available attacks $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ as studied in [35], e.g., when the attacker's constraints limit the number simultaneously active malicious components. Such attack configuration is illustrated in Figure 5b, and the overall effect can be captured using the parallel composition $\mathcal{A}_1 \| \ldots \| \mathcal{A}_n$.

1) Serial Composition: Two normalized FSTs $\mathcal{A} = (\mathsf{S}, \mathsf{s}_{\mathrm{init}}, \mathbf{I}, \mathbf{O}, \mathsf{Trans}, \mathsf{S}_{\mathrm{final}})$ and $\mathcal{A}' = (\mathsf{S}', \mathsf{s}'_{\mathrm{init}}, \mathbf{I}', \mathbf{O}', \mathsf{Trans}', \mathsf{S}'_{\mathrm{final}})$ can be serially composed to $\mathcal{A}'' = \mathcal{A} \circ \mathcal{A}'$ if $\mathbf{I}' = \mathbf{O}$ by taking the output of $\mathcal{A}$ as the input of $\mathcal{A}'$.[2] Generally, the composed FST is derived by (i) taking the Cartesian product of the states and transitions, (ii) contracting the transitions on which the output of the first component is identical to the input of the second component, as well as (iii) keeping transitions with ε output in $\mathcal{A}$ and ε input in $\mathcal{A}'$, while discarding the others; this is captured in Algorithm 1. Specially, FSTs can be composed with DESs by treating each DES as an FST with identical inputs and outputs.

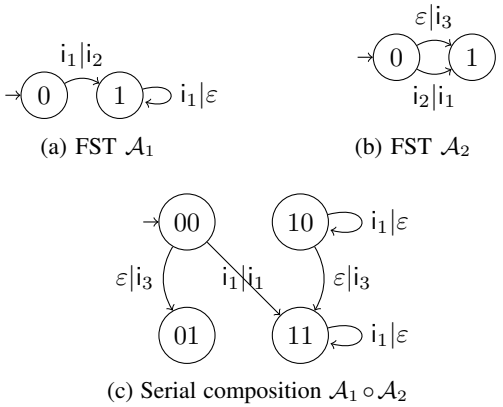
[2]The serial composition is also done for $\mathbf{I}' \neq \mathbf{O}$ by neglecting the symbols not in $\mathbf{I}' \cap \mathbf{O}$. Still, to simplify our presentation we assume that $\mathbf{I}' = \mathbf{O}$.



(a) Serial Composition (b) Parallel Composition

(c) Modeling a frequency constraint imposed on the attacker

Fig. 5: Composition of FSTs and attack constraints modeling.



(a) FST $\mathcal{A}_1$ (b) FST $\mathcal{A}_2$

(c) Serial composition $\mathcal{A}_1 \circ \mathcal{A}_2$

Fig. 6: Example of composition of FSTs.

Algorithm 1 Composition of Normalized FSTs

Require: Normalized FSTs $\mathcal{A} = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{I}, \mathbf{O}, \mathsf{Trans}, \mathsf{S}_{\text{final}})$
1: and $\mathcal{A}' = (\mathsf{S}', \mathsf{s}'_{\text{init}}, \mathbf{O}, \mathbf{O}', \mathsf{Trans}', \mathsf{S}'_{\text{final}})$
2: Let $\mathcal{A}'' = \mathcal{A} \circ \mathcal{A}' = (\mathsf{S} \times \mathsf{S}', (\mathsf{s}_{\text{init}}, \mathsf{s}'_{\text{init}}), \mathbf{I}, \mathbf{O}', \emptyset, \mathsf{S}_{\text{final}} \times \mathsf{S}'_{\text{final}})$.
3: Add transition $((\mathsf{s}_1, \mathsf{s}'_1), \mathsf{i}, \mathsf{o}', (\mathsf{s}_2, \mathsf{s}'_2))$ to $\mathcal{A}''$ if there exists $\mathsf{o} \in \mathbf{O}$ such that $(\mathsf{s}_1, \mathsf{i}, \mathsf{o}, \mathsf{s}_2) \in \mathsf{Trans}$ and $(\mathsf{s}'_1, \mathsf{o}, \mathsf{o}', \mathsf{s}'_2) \in \mathsf{Trans}'$.
4: Add transition $((\mathsf{s}_1, \mathsf{s}'), \mathsf{i}, \varepsilon, (\mathsf{s}_2, \mathsf{s}'))$ to $\mathcal{A}''$ for any $(\mathsf{s}_1, \mathsf{i}, \varepsilon, \mathsf{s}_2) \in \mathsf{Trans}$ and any $\mathsf{s}' \in \mathsf{S}'$.
5: Add transition $((\mathsf{s}, \mathsf{s}'_1), \varepsilon, \mathsf{o}, (\mathsf{s}, \mathsf{s}'_2))$ to $\mathcal{A}''$ for any $(\mathsf{s}'_1, \varepsilon, \mathsf{o}, \mathsf{s}'_2) \in \mathsf{Trans}'$ and any $\mathsf{s} \in \mathsf{S}$.
6: **return** $\mathcal{A}''$

Finally, it is worth noting also that Algorithm 1 provides an FST realization for composition of regular relations $\mathcal{R}_{\mathcal{A}_1} \circ \mathcal{R}_{\mathcal{A}_2} = \mathcal{R}_{\mathcal{A}_1 \circ \mathcal{A}_2}$, and shows the closeness of regular relations under serial composition.

2) Parallel Composition: Two normalized FSTs $\mathcal{A} = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{I}, \mathbf{O}, \mathsf{Trans}, \mathsf{S}_{\text{final}})$ and $\mathcal{A}' = (\mathsf{S}', \mathsf{s}'_{\text{init}}, \mathbf{I}', \mathbf{O}', \mathsf{Trans}', \mathsf{S}'_{\text{final}})$ can be composed in parallel to $\mathcal{A}'' = \mathcal{A} \| \mathcal{A}'$ if $\mathbf{I}' = \mathbf{I}$ and $\mathbf{O}' = \mathbf{O}$.[3] The composed FST is derived by (i) taking the union of the states, final states and transitions, and (ii) adding a new starting state with transitions ε/ε to the initial states s_{init} and $\mathsf{s}'_{\text{init}}$. The detailed algorithm can be found in [13, 22] and is omitted here.

Example 5 (Serial Composition)**.** *Consider two FSTs $\mathcal{A}_1$ and $\mathcal{A}_2$, illustrated in Figures 6a and 6b, where one replaces i_1 with i_2, and the other injects i_3. The composition of the FSTs derived using Algorithm 1 is presented in Figure 6c. It is easy to check that $\mathcal{R}_{\mathcal{A}_1 \circ \mathcal{A}_2} = \mathcal{R}_{\mathcal{A}_1} \circ \mathcal{R}_{\mathcal{A}_2}$.*

C. Modeling Constraints on Attackers

Another advantage of using FSTs for attack modeling is that they facilitate capturing of attack constraints that are imposed by the underlying platform. For instance, in some networked control systems, cryptographic primitives (e.g., Message Authentication Codes – MACs) can only be intermittently used due to resource constraints, and thus only intermittently preventing injection (i.e., injection) attacks from occurring [15, 16]. FSTs can be used to model such restrictions specified as counting or frequency constraints imposed on the attacker – e.g., allowing an attack to occur at most a certain number of times f within every window of size l [15, 16] or as studied in [32].

For an FST $\mathcal{A} = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{I}, \mathbf{O}, \mathsf{Trans}, \mathsf{S}_{\text{final}})$, let $\mathcal{A}' = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{I}, \mathsf{Trans}, \mathsf{S}_{\text{final}})$ be the automaton derived from $\mathcal{A}$ by removing the output symbols of $\mathcal{A}$. Namely, $\mathcal{A}'$ accepts the same language as $\mathcal{A}$, but does not revise. It accepts the same input language as $\mathcal{A}$ without rewriting. Let $\mathcal{F}$ be a frequency counter modeled by a automaton. For example, the automaton in Figure 5c with transitions drawn in dotted lines constraints the attack frequency to once every three steps. The symbols D/E in Figure 5c stands for attack disabling and enabling.

An attack with frequency constraint is derived by combining the frequency counter $\mathcal{F}$ with $\mathcal{A}$ and $\mathcal{A}'$ into a new FST as shown in Figure 5c. Each transition with label D/E is replaced by a connection to $\mathcal{A}'/\mathcal{A}$, respectively. Note that for the resulting FST from Figure 5c, the transitions in solid lines are labeled by $\varepsilon|\varepsilon$, namely, they are automatically triggered without generating any output symbol.

Remark 3 (Relationship with existing work)**.** *The problem considered in this work generalizes previously investigated problems. To show this, let us assume the plant to be a deterministic finite state automaton. By specifying suitable models of actuator and sensor attacks, several previous problem formulations can be derived as follows:*

(i) Let the actuator attack model $\mathcal{A}_a^{(s)} = \text{Inject}_{\mathbf{I}_{uc}} \circ \mathcal{P}$ be the serial composition of the injection attack from (4) *and the plant. These injected symbols are acceptable symbols of the plant, but not uncontrollable by the supervisor. In addition, let the sensor attack model satisfy $\mathcal{A}_s^{(s)} = \text{Project}_{\mathbf{I}_o}$ from* (2)*. The removed symbols can be viewed as the unobservable symbols generated by the plant. Such scenario results in the standard (i.e., without taking security/attacks into account) supervisory control formulation [6] with uncontrollable events $\mathbf{I}_{uc}$,*

[3]The parallel composition is also possible for $\mathbf{I}, \mathbf{O} \neq \mathbf{I}', \mathbf{O}'$.

and unobservable events $\mathbf{I}_{uo} = \mathbf{I} \backslash \mathbf{I}_o$.

(ii) Let the actuator attack be modeled as the serial composition $\mathcal{A}_a^{(ed)} \circ \mathcal{A}_a^{(s)}$*, where* $\mathcal{A}_a^{(s)}$ *is defined as in (i) and* $\mathcal{A}_a^{(ed)}$ *is the injection-removal attack on a set of vulnerable control symbols from Example 3. Such scenario results in the problem of supervisory control under the actuator enablement/disablement attacks which was studied in [4]. Similarly, if the sensor attack is* $\mathcal{A}_s^{(ed)} \circ \mathcal{A}_s^{(s)}$*, with* $\mathcal{A}_s^{(s)}$ *defined as in (i) and* $\mathcal{A}_s^{(ed)}$ *is the injection-removal attack on a set of vulnerable plant output symbols, then the problem considered in this work results in the problem of supervisory control under the sensor enablement/disablement attack problem from [4].*

(iii) Let the sensor attack be $\mathcal{A}_s^{(rr)} \circ \mathcal{A}_s^{(s)}$ *where* $\mathcal{A}_s^{(s)}$ *is defined as in (i) and* $\mathcal{A}_s^{(rr)}$ *is the replacement-removal attack from Example 2. In this case, our problem formulation results in the problem of supervisory control under replacement-removal attack from [35]. Similarly, adding a sensor attack module of injection-deletion attack from Example 3 yields the problem of supervisory control under injection-deletion sensor attacks studied in [35].*

V. Controllability Under Sensor Attacks

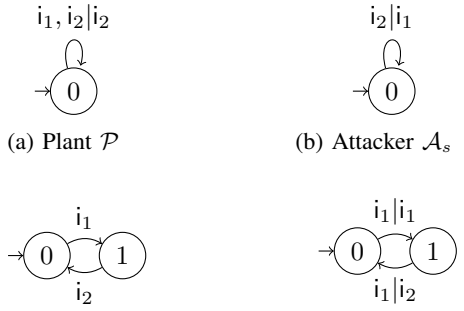
In Sections V to VII, we study the supervisory control of FST-modeled deterministic plants in the presence of attacks. Specifically, we start by considering resiliency (i.e., controllability) under sensor attacks in this section. We consider the setup from Figure 2 without $\mathcal{A}_a$, where the sensors symbols are under the attack modeled as $\mathcal{A}_s$ before they are received by the supervisor $\mathcal{S}$. At first glance, this looks like a trivial question — the supervisor can simply be the automata generating the desired language $\mathscr{K}$. As the words generated by the supervisor are directly sent to the plant, the plant is guaranteed to execute exactly words in $\mathscr{K}$.

However, when the attacker $\mathcal{A}_s$ may compromise sensor measurements, and assuming that the supervisor $\mathcal{S}$ can only be automata, then it is generally impossible to close the loop between the sensors of the plant to the input to the supervisor as illustrated in the following example.

Example 6. *Following the architecture from Figure 2 without* $\mathcal{A}_a$*, consider a set of symbols* $\mathbf{I} = \{\mathsf{i}_1, \mathsf{i}_2\}$ *and a plant* $\mathcal{P}$ *accepting the language* $(\mathsf{i}_1, \mathsf{i}_2)^*$*, as shown in Figure 7a. The (prefix-closed) desired language is* $\mathscr{K} = \overline{(\mathsf{i}_1\mathsf{i}_2)^*}$*, represented by a discrete event system* $\mathcal{M}_{\mathscr{K}}$ *as shown in Figure 7c. Finally, let us assume that the attack on sensors* $\mathcal{A}_s$ *is represented by the FST from Figure 7b.*

In this case, supervising the plant without the ability to revise symbols is impossible, as the output of the supervisor should be the desired the language $(\mathsf{i}_1, \mathsf{i}_2)^*$*, but the input is always* i_1^**, as the plant only generates* i_2^* *and the attacker rewrites it to* i_1^**. However, for such attack model there exists an attack-resilient supervisor for the plant, modeled as an FST* $\mathcal{S}$ *presented in Figure 7d. It counters the attacks by revising* i_1 *back to* i_2 *every other step.*

As illustrated in the above example, the problem is that if the sensors are corrupted by the attacker $\mathcal{A}_s$, the supervisor $\mathcal{S}$ will need the ability to change it back. But, when modeled by an automaton, the supervisor can only enable or disable events. This is insufficient to counter the attack and provide resiliency against such attacks. Therefore, in this work, we consider the use FSTs to model supervisors. Modeling supervisors as FSTs instead of automata provides them the extra ability to revise symbols that are required to counter attacks on the plant's sensors, as shown in Example 6.



(a) Plant $\mathcal{P}$ (b) Attacker $\mathcal{A}_s$

(c) Model of the desired language $\mathscr{K}$ (d) Supervisor $\mathcal{S}$

Fig. 7: Example of sensor attacks that can only be countered by FST-based supervisors.

Now, when we consider the problem of controllability under attack, in the rest of the section, we make the following assumption on the sensor attack model $\mathcal{A}_s$.

Assumption 1. *The attacker* $\mathcal{A}_s$ *can (i) accept and (ii) only accept words that are generated by the plant* $\mathcal{P}$*, i.e.,* $L_{\text{in}}(\mathcal{A}_s) = L_{\text{out}}(\mathcal{P})$.

The first part of Assumption 1 means that the attacker $\mathcal{A}_s$ is well-defined for any acceptable word of the plant $\mathcal{P}$. Otherwise, an error will occur when $\mathcal{A}_s$ receives an unacceptable input. The second part of Assumption 1 is always achievable by trimming the attacker $\mathcal{A}_s$.

Intuitively, to supervise the plant under sensor attacks while ensuring attack-resiliency, the supervisor needs to (i) recover the possible output of the plant, (ii) recover the possible input of the plant, and (iii) constrain the input to the plant. The task (i) is performed by taking the inversion of the attack model described as an FST; note that the attack model may capture a wide range of actions as it may be overly conservative to assume a specific (e.g., deterministic) attack mapping. Similarly, the task (ii) is achievable by taking the inversion of the plant.

Inversion: A normalized FST $\mathcal{A} = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{I}, \mathbf{O}, \mathsf{Trans}, \mathsf{S}_{\text{final}})$ is inversed to $\mathcal{A}^{-1} = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{O}, \mathbf{I}, \mathsf{Trans}, \mathsf{S}_{\text{final}})$ by flipping the input and output symbol on each transition, as presented in Algorithm 2.

Note that Algorithm 2 provides an FST realization for inversion of regular relations $\mathcal{R}_{\mathcal{A}^{-1}} = \mathcal{R}_{\mathcal{A}}^{-1}$, and shows the closeness of regular relations under inversion. We now introduce the following lemma that follows immediately from Algorithm 2.

Lemma 1. *For any FST* $\mathcal{A}$*, the composed regular relations* $\mathcal{R}_{\mathcal{A}^{-1} \circ \mathcal{A}}$ *and* $\mathcal{R}_{\mathcal{A} \circ \mathcal{A}^{-1}}$ *contains the identity relation, i.e.,*

$$\mathcal{R}_{\mathcal{M}_{L_{\text{in}}(\mathcal{A})}} \subseteq \mathcal{R}_{\mathcal{A} \circ \mathcal{A}^{-1}}, \quad \mathcal{R}_{\mathcal{M}_{L_{\text{out}}(\mathcal{A})}} \subseteq \mathcal{R}_{\mathcal{A}^{-1} \circ \mathcal{A}}.$$

Algorithm 2 Normalized FST Inversion

Require: Normalized FST $\mathcal{A} = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{I}, \mathbf{O}, \mathsf{Trans}, \mathsf{S}_{\text{final}})$
1: Let $\mathcal{A}^{-1} = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{O}, \mathbf{I}, \emptyset, \mathsf{S}_{\text{final}})$.
2: **for** $(\mathsf{s}, I, O, \mathsf{s}') \in \mathsf{Trans}$ and $|I| > 1$ **do**
3: Add transition $(\mathsf{s}, O, I, \mathsf{s}')$ to $\mathcal{A}^{-1}$.
4: **end for**
5: **return** $\mathcal{A}^{-1}$

Algorithm 3 Design of a supervisor resilient to sensor attacks

Require: Desire language $\mathscr{K}$, plant $\mathcal{P}$, sensor attacker $\mathcal{A}_s$.
1: Find $\mathcal{M}_{\mathscr{K}}$ realizing $\mathscr{K} \subseteq L_{\text{out}}(\mathcal{P})$.
2: Compute composition $\mathcal{P} \circ \mathcal{A}_s$.
3: Compute inversion $(\mathcal{P} \circ \mathcal{A}_s)^{-1}$.
4: Compute composition $\mathcal{S} = (\mathcal{P} \circ \mathcal{A}_s)^{-1} \circ \mathcal{M}_{\mathscr{K}}$.
5: **return** Supervisor $\mathcal{S}$.

where $\mathcal{M}_{L_{\text{in}}(\mathcal{A})}$ and $\mathcal{M}_{L_{\text{out}}(\mathcal{A})}$ are FSAs (and thus also FSTs by Remark 2) realizing the input and output languages $L_{\text{in}}(\mathcal{A})$ and $L_{\text{out}}(\mathcal{A})$ of the FST $\mathcal{A}$, respectively.

Returning to the resilient supervisory control problem, to counter the attacker $\mathcal{A}_s$, which may be nondeterministic, and recover the possible input of the plant $\mathcal{P}$, we construct the inversion $(\mathcal{P} \circ \mathcal{A}_s)^{-1}$. For any input I, of the plant, the corresponding output is $\mathcal{P}(I)$, and then the attacker $\mathcal{A}_s$ rewrites it to a word in $\mathcal{R}_{\mathcal{P}\circ\mathcal{A}_s}(I)$. Thus, the inversion can reverse it back to $\mathcal{R}_{(\mathcal{P}\circ\mathcal{A}_s)\circ(\mathcal{P}\circ\mathcal{A}_s)^{-1}}(I)$, which is the set of possible words passing through the plant and yielding the same observation after attack as I. Restricting this set of words to the desired language $\mathscr{K}$ guarantees the supervisory control goal. Therefore, the supervisor $\mathcal{S}$ should be designed to be $(\mathcal{P} \circ \mathcal{A}_s)^{-1} \circ \mathcal{M}_{\mathscr{K}}$, where $\mathcal{M}_{\mathscr{K}}$ is the automaton realizing the desired language $\mathscr{K}$. This is summarized by Theorem 1 and Algorithm 3.

Theorem 1 (Controllability under sensor attack)**.** *In Figure 2 without $\mathcal{A}_a$, the plant $\mathcal{P}$ is controllable to the desired regular language $\mathscr{K} \subseteq L_{\text{out}}(\mathcal{P})$ under the attacker $\mathcal{A}_s$ on the plant's sensors. This can be achieved by the supervisor $\mathcal{S} = \mathcal{A}_s^{-1} \circ \mathcal{P}^{-1} \circ \mathcal{M}_{\mathscr{K}}$.*

Proof. It suffices to show the language passing the plant is exactly $\mathscr{K}$ under the supervisor $\mathcal{S} = \mathcal{A}_s^{-1} \circ \mathcal{P}^{-1} \circ \mathcal{M}_{\mathscr{K}}$. By construction, the supervisor only generates words contained in $\mathscr{K}$. On the other hand, for any $I \in \mathscr{K}$, noting that by Lemma 1, $I \in \mathcal{R}_{(\mathcal{P}\circ\mathcal{A}_s)\circ(\mathcal{P}\circ\mathcal{A}_s)^{-1}}(I)$, we know $I \in \mathcal{R}_{\mathcal{P}\circ\mathcal{A}_s\circ\mathcal{S}}(I)$, i.e., the word I is allowed to transmit to the plant. □

Theorem 1 provides a computational method to design supervisor, which we introduce in Algorithm 3.

Example 7 (Supervisor design under sensor attacks)**.** *Let us revisit Example 6 and the system from Figure 2 with only $\mathcal{A}_s$ (and no $\mathcal{A}_a$). Attack-resilient supervisor $\mathcal{S}$ in Figure 7d can be derived by the serial composition of the inversion of the attacker $\mathcal{A}_s$ in Figure 7b and the model of desired language $\mathscr{K}$ in Figure 7c – i.e., $\mathcal{S} = (\mathcal{P} \circ \mathcal{A}_s)^{-1} \circ \mathcal{M}_{\mathscr{K}}$.*

Algorithm 4 Construction of an FST Filter for the Desired Language $\mathscr{K}$

Require: Desire language $\mathscr{K}$, plant $\mathcal{P}$.
1: Find automata $\mathcal{M}$ and $\mathcal{M}'$ with $L(\mathcal{M}) = L_{\text{out}}(\mathcal{P})$ and $L(\mathcal{M}') = \mathscr{K}$.
2: Convert $\mathcal{M}$ and $\mathcal{M}'$ to FSTs by adding ε as output and input symbol for each transition, respectively.
3: $\mathcal{A} = \mathcal{M} \circ \mathcal{M}'$.
4: Trim off transitions with output symbol ε in $\mathcal{A}$.
5: **return** Supervisor $\mathcal{A}$.

VI. Controllability Under Actuator Attacks

In this section, we study the supervisory control problem under only actuator attacks for deterministic plants. We consider the setup from Figure 2 without $\mathcal{A}_s$ – the actuators of the plant $\mathcal{P}$ are under the attack modeled with $\mathcal{A}_a$, but the supervisor $\mathcal{S}$ has direct access to the sensing symbols (words) coming from the plant. Note that this problem is more complex then the resilient supervisory control problem with sensor attacks studied in Section V, as the control words of the supervisor are not directly sent to the plant. Consequently, the desired language may not be controllable to the plant for different attack models. For example, if the attacks $\mathcal{A}_a$ generates the empty word ε upon all inputs, capturing Denial-of-Service (DoS) attacks, then the only controllable desired language for the plant is $\{\varepsilon\}$.

In the rest of the section, we make the following assumption on the actuator attacks $\mathcal{A}_a$.

Assumption 2. *The attacker $\mathcal{A}_a$ can (i) generate and (ii) only generate words that are acceptable to the plant $\mathcal{P}$, i.e., $L_{\text{out}}(\mathcal{A}_a) = L_{\text{in}}(\mathcal{P})$.*

Similarly to the case of attacks on sensors, the first part of Assumption 2 means that the actuator attacks $\mathcal{A}_a$ will not cause an error by sending an unacceptable input of the plant $\mathcal{P}$, as such attacks are easy to detect. The way the attacker tries to interfere with the control goal is to inject an undesired word in $L_{\text{in}}(\mathcal{P})\backslash\mathscr{K}$ to the plant. The second part of Assumption 2 ensures that every word in $L_I(\mathcal{P})$ may possibly be sent to the plant $\mathcal{P}$.

For attack-resilient supervision of the plant under sensor attacks, the supervisor $\mathcal{S}$ needs to (i) recover the possible input word of the plant $\mathcal{P}$ from its output, (ii) constrain the possible word within the desired language, and (iii) rewrite these words to counter the effects of the actuator attacks $\mathcal{A}_a$. The task (i) is achievable by the inversion $\mathcal{P}^{-1}$ of the plant.

Filter: Unlike the approach introduced in Section V, in this case task (ii) is achieved by an FST filter $\mathcal{M}_{\mathscr{K}}$ of the desired language $\mathscr{K} \in L_{\text{in}}(\mathcal{P})$. This is because an automaton of $\mathscr{K}$ cannot handle input words in $L_{\text{in}}(\mathcal{P})\backslash\mathscr{K}$. The filter takes a word $I \in L_{\text{in}}(\mathcal{P})$ and sends the same word if $I \in \mathscr{K}$, and ε otherwise. To achieve this we introduce Algorithm 4.

Finally, for the task (iii), we construct the inversion $\mathcal{A}_a^{-1}$ of the actuator attack model. Consequently, the supervisor is

$$\mathcal{S} = \mathcal{P}^{-1} \circ \mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1}. \tag{5}$$

Specifically, for any word $I \in L_{\text{out}}(\mathcal{P})$ from the output of the plant, the inversion $\mathcal{R}_{\mathcal{P}^{-1}}(I)$ recovers the corresponding possible inputs of the plant. Then the filter $\mathcal{M}_{\mathscr{K}}$ constraints them to $\mathcal{R}_{\mathcal{P}^{-1}\circ\mathcal{M}_{\mathscr{K}}}(I) \subseteq \mathscr{K}$. Finally, the inversion $\mathcal{A}_a^{-1}$ rewrites them to $\mathcal{R}_{\mathcal{P}^{-1}\circ\mathcal{M}_{\mathscr{K}}\circ\mathcal{A}_a^{-1}}(I)$ to counter the actuator attacks modeled by the FST $\mathcal{A}_a^{-1}$.

By construction, only the words $I' \in \mathscr{K}$ can pass the filter $\mathcal{M}_{\mathscr{K}}$. But, noting that $\mathcal{R}_{\mathcal{A}_a^{-1}\circ\mathcal{A}_a}(\mathscr{K})$ is not necessarily contained in $\mathscr{K}$, the input language of the plant may not be exactly $\mathscr{K}$; instead, the supervisor $\mathcal{S}$ from (5) may only restrict the language passing the plant to a minimal superset of $\mathscr{K}$. The desired language $\mathscr{K}$ is controllable, when the containment holds. This is equivalent to checking if $\mathscr{K}$ is contained in the output language of $\mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1} \circ \mathcal{A}_a$.

This is summarized by the following theorem.

Theorem 2 (Controllability under actuator attacks)**.** *Consider the setup from Figure 2 without $\mathcal{A}_s$. The plant $\mathcal{P}$ is weakly controllable to the desired regular language $\mathscr{K} \subseteq L_{\text{in}}(\mathcal{P})$ under the actuator attack $\mathcal{A}_a$ by the supervisor $\mathcal{S} = \mathcal{P}^{-1} \circ \mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1}$. Furthermore, the minimal controllable language containing $\mathscr{K}$ is*

$$\tilde{\mathscr{K}} = \mathcal{R}_{\mathcal{A}_a^{-1}\circ\mathcal{A}_a}(\mathscr{K}). \tag{6}$$

The desired language is controllable if and only if $\tilde{\mathscr{K}} = \mathscr{K}$, or equivalently the output language of $\mathcal{M}_{\mathscr{K}}\circ\mathcal{A}_a^{-1}\circ\mathcal{A}_a$ satisfies

$$L_{\text{out}}(\mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1} \circ \mathcal{A}_a) \subseteq \mathscr{K}. \tag{7}$$

Proof. Sufficiency: It suffices to show that the language passing to the plant is exactly $\mathscr{K}$ under the supervisor $\mathcal{S} = \mathcal{P}^{-1} \circ \mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1}$. By (7), we have $L_{\text{out}}(\mathcal{S} \circ \mathcal{A}_a) \subseteq \mathscr{K}$, thus the plant only receives words in $\mathscr{K}$. On the other hand, for any $I \in \mathscr{K}$, noting that $I \in \mathcal{R}_{\mathcal{A}_a^{-1}\circ\mathcal{A}_a}(I)$ and $I \in \mathcal{R}_{\mathcal{P}\circ\mathcal{P}^{-1}}(I)$ by Lemma 1, we have

$$I \in \mathcal{R}_{\mathcal{P}\circ\mathcal{S}\circ\mathcal{A}_a}(I) = \mathcal{R}_{\mathcal{P}\circ\mathcal{P}^{-1}\circ\mathcal{M}_{\mathscr{K}}\circ\mathcal{A}_a^{-1}\circ\mathcal{A}_a}(I),$$

namely, the word I is allowed to be transmitted to the plant.

Necessity: It suffices to show the minimality of $\tilde{K}$ in (6). Suppose there exists a supervisor $\mathcal{S}$ that can weakly control the plant to $\mathscr{K}'$ such that

$$\mathscr{K} \subseteq \mathscr{K}' \subseteq \tilde{\mathscr{K}}. \tag{8}$$

Then for any $I' \in \mathscr{K}'$, there exists an output word of the supervisor I such that $I \in \mathcal{R}_{\mathcal{A}_a}^{-1}(I')$. Consequently, any word in $\mathcal{A}_a(I)$ can be transmitted to the plant, i.e., $\mathcal{A}_a(I) \subseteq \mathscr{K}'$. Namely,

$$\mathcal{A}_a^{-1} \circ \mathcal{A}_a(\mathscr{K}') \subseteq \mathscr{K}' \tag{9}$$

Combining (6),(8), and (9), it follows that

$$\tilde{\mathscr{K}} = \mathcal{A}_a^{-1} \circ \mathcal{A}_a(\mathscr{K}) \subseteq \mathcal{A}_a^{-1} \circ \mathcal{A}_a(\mathscr{K}') \subseteq \mathscr{K}'.$$

This implies that $\mathscr{K}' = \tilde{\mathscr{K}}$. □

Theorem 2 provides a computational method to design such supervisor, as provided in Algorithm 5.

Example 8 (Supervisor design for actuator attacker)**.** *Following the configuration from Figure 2 without $\mathcal{A}_s$, consider a set of symbols $\mathbf{I} = \{\mathsf{i}_1, \mathsf{i}_2\}$ and a plant $\mathcal{P}$ accepting the language $(\mathsf{i}_1, \mathsf{i}_2)^*$, as shown in Figure 8a. The (prefix-closed) desired language is $\mathscr{K} = \overline{(\mathsf{i}_1, \mathsf{i}_2)\mathsf{i}_2}$, associated with the FST filter $\mathcal{M}_{\mathscr{K}}$ (Figure 8b). Now consider the following two cases.*

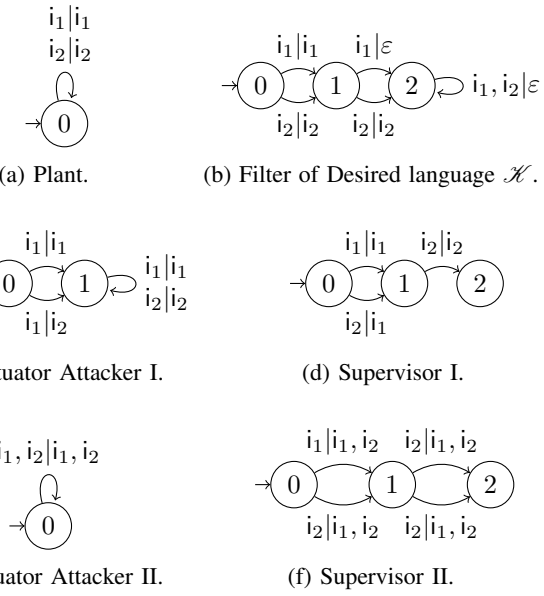
Controllable: The attacker $\mathcal{A}_I$ on the input of the plant is represented by an FST shown in Figure 8c. It rewrites the first i_1 nondeterministically to i_1 or i_2. The output language $L_{\text{out}}(\mathcal{S} \circ \mathcal{A}_I) = \overline{(\mathsf{i}_1, \mathsf{i}_2)\mathsf{i}_2}$ is exactly $\mathscr{K}$. Thus, the plant is controllable to $\mathscr{K}$ by the supervisor $\mathcal{S}$ even under the attack.

Weakly Controllable: The attacker $\mathcal{A}_I$ on the input of the plant is represented by an FST from Figure 8e. It rewrites any i_1 nondeterministically to i_1 or i_2. It sends first i_1 and then i_1 or i_2 upon receiving i_2. The output language $L_{\text{out}}(\mathcal{S} \circ \mathcal{A}_I) = \overline{(\mathsf{i}_1, \mathsf{i}_2)(\mathsf{i}_1, \mathsf{i}_2)}$ is larger than $\mathscr{K}$. Therefore, the plant is uncontrollable to $\mathscr{K}$ by the supervisor $\mathcal{S}$. It is easy to see that $\overline{(\mathsf{i}_1, \mathsf{i}_2)(\mathsf{i}_1, \mathsf{i}_2)}$ is a minimal superset of $\mathscr{K}$.

Algorithm 5 Design of a supervisor resilient to actuator attacks

Require: Desired language $\mathscr{K}$, plant $\mathcal{P}$, model of actuator attacks $\mathcal{A}_a$.

```
Find FST filter M_K for K .
Compute inversions A_a^{-1}, P^{-1} .
Compute serial composition S = P^{-1} ∘ M_K ∘ A_a^{-1} .
if the output language L_out(M_K ∘ A_a^{-1} ∘ A_a) ⊆ K then
    return K is controllable.
else
    return K is not controllable.
end if
return Supervisor S.
```



(a) Plant. (b) Filter of Desired language $\mathscr{K}$.

(c) Actuator Attacker I. (d) Supervisor I.

(e) Actuator Attacker II. (f) Supervisor II.

Fig. 8: Example supervisors for actuator attacks.

Remark 4. *Note that the supervisor in Figure 8f, derived in Example 8, is nondeterministic, corresponding to multiple allowable controls. For example, the supervisor at the state* 0 *can either send i_1 or i_2 upon receiving i_1. The controllability*

theorem guarantees that in the presence of attacks, the union of all possible words received by the plant under all these allowable controls is exactly the desired language $\mathscr{K}$. In implementation, the nondeterminism can be resolved by choosing one of the allowable controls. Accordingly, the possible words received by the plant is contained in $\mathscr{K}$. An avenue for future work is to resolve the nondeterminism optimally when there are different costs for the supervisor to revise the control symbols.

VII. Controllability Under Both Sensor and Actuator Attacks

In this section, we study the supervisory control problem under both sensor and actuator attacks for deterministic plants, as shown in Figure 2. This is a combination of the supervisory control problems studied in Sections V and VI. In the rest of the section, we assume that the actuator attacker $\mathcal{A}_a$ and the sensor attacker $\mathcal{A}_s$ are well-defined as captured in Assumptions 1 and 2.

From Section VI, it follows that the FST $\mathcal{P}^{-1} \circ \mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1}$ can constrain the input of the plant $\mathcal{P}$, and revise these words to counter the actuator attacks modeled by $\mathcal{A}_a$. Here, $\mathcal{M}_{\mathscr{K}}$ is a filter. This, in combination with the analysis in Section V, implies that the supervisor $\mathcal{A}_s^{-1} \circ \mathcal{P}^{-1} \circ \mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1}$ can additionally counter the sensor attacks $\mathcal{A}_s$. It is easy to check that $\mathscr{K} \subseteq \mathcal{R}_{\mathcal{P}\circ\mathcal{A}_s\circ\mathcal{S}\circ\mathcal{A}_a}(\mathscr{K})$. However, $\mathcal{R}_{\mathcal{P}\circ\mathcal{A}_s\circ\mathcal{S}\circ\mathcal{A}_a}(\mathscr{K})$ is not necessarily contained in $\mathscr{K}$. The supervisor $\mathcal{S} = \mathcal{A}_s^{-1}\circ\mathcal{P}^{-1}\circ\mathcal{M}_{\mathscr{K}}\circ\mathcal{A}_a^{-1}$ only restricts the language passing the plant to a minimal superset of $\mathscr{K}$. The desired language $\mathscr{K}$ is controllable, when the containment holds. This is summarized by the following theorem.

Theorem 3 (Controllability under actuator and sensor attacks)**.** *In Figure 2, the plant $\mathcal{P}$ is weakly controllable to the desired regular language $\mathscr{K} \subseteq L_{\rm in}(\mathcal{P})$ under the attacks $\mathcal{A}_a$ and $\mathcal{A}_s$ on its input and output, respectively, by the supervisor $\mathcal{S} = \mathcal{A}_s^{-1} \circ \mathcal{P}^{-1} \circ \mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1}$. Furthermore, the minimal controllable language containing $\mathscr{K}$ is*

$$\tilde{\mathscr{K}} = \mathcal{R}_{\mathcal{A}_a^{-1}\circ\mathcal{A}_a}(\mathscr{K}). \tag{10}$$

The desired language is controllable if and only if $\tilde{\mathscr{K}} = \mathscr{K}$, or equivalently the output language of $\mathcal{M}_{\mathscr{K}}\circ\mathcal{A}_a^{-1}\circ\mathcal{A}_a$ satisfies

$$L_{\rm out}(\mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1} \circ \mathcal{A}_a) \subseteq \mathscr{K}. \tag{11}$$

Proof. Necessity: Same as the proof for necessity for Theorem 2.

Sufficiency: It suffices to show that the language passing the plant is exactly $\mathscr{K}$ under the supervisor $\mathcal{S} = \mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1}$. By (11), we have that

$$L_{\rm out}(\mathcal{S} \circ \mathcal{A}_a) \subseteq L_{\rm out}(\mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1} \circ \mathcal{A}_a) \subseteq \mathscr{K},$$

namely, the plant only receives words in $\mathscr{K}$. On the other hand, for any $I \in \mathscr{K}$, noting that $I \in \mathcal{R}_{\mathcal{A}_a^{-1}\circ\mathcal{A}_a}(I)$ and $I \in \mathcal{R}_{(\mathcal{P}\circ\mathcal{A}_s)\circ(\mathcal{P}\circ\mathcal{A}_s)^{-1}}(I)$, it follows that

$$I \in \mathcal{R}_{\mathcal{A}_s\circ\mathcal{S}\circ\mathcal{A}_a}(I) = \mathcal{R}_{\mathcal{P}\circ\mathcal{A}_s\circ(\mathcal{P}\circ\mathcal{A}_s)^{-1}\circ\mathcal{M}_{\mathscr{K}}\circ\mathcal{A}_a^{-1}\circ\mathcal{A}_a}(I),$$

namely, the word I is allowed to transmit to the plant. □

Algorithm 6 Design supervisor under both actuator and sensor attacks

Require: Plant $\mathcal{P}$, actuator attacker $\mathcal{A}_a$, sensor attacker $\mathcal{A}_s$, desired language $\mathscr{K}$.

```
Find a model M_K of K .
Compute inversion A_a^{-1}, P^{-1} and A_s^{-1} .
Compute composition S = A_s^{-1} ∘ P^{-1} ∘ M_K ∘ A_a^{-1} .
if the output language L_out(M_K ∘ A_a^{-1} ∘ A_a) ⊆ K then
    return K is controllable
else
    return K is not controllable
end if
return Supervisor S.
```

As illustrated by the statement and proof of Theorem 3, the design of supervisor can be performed by separately taking into accounts the actuator and sensor attacks, and the sensor attacks, modeled by $\mathcal{A}_s$, have no influence on the controllability. This result is different from most previous works [4, 6, 35]. This is because the supervisor when modeled by an FST, and not an automaton, has more power in generating control words by itself, and depends much less on the input word it receives. By adding a component $\mathcal{A}_s^{-1}$, the effect of the sensor attacker $\mathcal{A}_s$ is totally countered, as is summarized by the following corollary.

Corollary 1. *For deterministic plants, the sensor attacker $\mathcal{A}_s$ has no influence on the controllability of a desired language in the supervisor control problem shown in Figure 2.*

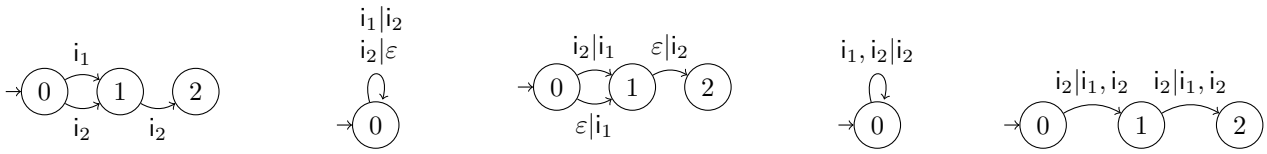
Theorem 3 also provides a computational method to design such supervisor, as given in Algorithm 6.

Example 9 (Supervisor design for resiliency under both sensor and actuator attacks)**.** *Following Example 8 and the configuration in Figure 2, consider a set of symbols $\mathbf{I} = \{\mathsf{i}_1, \mathsf{i}_2\}$ and a plant $\mathcal{P}$ accepting the language $(\mathsf{i}_1, \mathsf{i}_2)^*$, as shown in Figure 8a. The (prefix-closed) desired language is $\mathscr{K} = \overline{(\mathsf{i}_1, \mathsf{i}_2)\mathsf{i}_2}$, represented by an automaton $\mathcal{M}_{\mathscr{K}}$ as shown in Figure 9a. Now, let us consider the following two cases.*

Controllable: The attackers $\mathcal{A}_a$ and $\mathcal{A}_s$ on the sensors and actuators of the plant are modeled by FSTs from Figures 8c and 9b, respectively. The actuator attacker rewrites the first i_1 nondeterministically to i_1 or i_2. The sensor attacker removes i_2 and replaces i_1 with i_2. The supervisor shown in Figure 9c does not contain any transition with input label i_1, as it will never appear due to the attack. The output language $L_{\rm out}(\mathcal{A}_s \circ \mathcal{S} \circ \mathcal{A}_a) = \overline{(\mathsf{i}_1, \mathsf{i}_2)\mathsf{i}_2}$ is equal to $\mathscr{K}$. Thus, the plant is controllable to $\mathscr{K}$ by the supervisor $\mathcal{S}$.

Weakly Controllable: The attackers $\mathcal{A}_a$ and $\mathcal{A}_s$ are modeled by FSTs from Figures 8e and 9d, respectively. The actuator attacker rewrites i_1 nondeterministically to i_1 or i_2. The sensor attacker replaces i_1 with i_2. The supervisor does not contain any transition with input label i_1, as it will never appear. Obviously, the output language $\overline{(\mathsf{i}_1, \mathsf{i}_2)(\mathsf{i}_1, \mathsf{i}_2)}$ minimally contains $\mathscr{K}$.

Comparing to Example 8, adding a sensor attacker has no influence on the controllability. This agrees with Corollary 1.



(a) Desired language $\mathscr{K}$ (b) Sensor Attacker I (c) Supervisor I (d) Sensor Attacker II (e) Supervisor II

Fig. 9: Example supervisors for sensor and actuator attacks.

Remark 5. *Recalling Remark 3, more controllable languages can be achieved here than previous works [35, 32, 4, 6], as more powerful supervisors that can revise symbols are used. The only exception is that Theorem 2 reduces to the standard controllability theorem $K\mathbf{I}_c \cap L_{\rm in}(\mathcal{P}) = \mathscr{K}$ in [6], since in this case, the supervisor does not revise.*

Finally, we note that if the second part of Assumption 2 is violated, i.e., $L_{\rm out}(\mathcal{A}_a) \subseteq L_{\rm in}(\mathcal{P})$, Theorems 2 and 3 still hold on the trimmed plant accepting $L_{\rm out}(\mathcal{A}_a)$.

Remark 6. *For Theorems 2 and 3, in the case of $L_{\rm out}(\mathcal{A}_a) \subseteq L_{\rm in}(\mathcal{P})$, it is easy to show that $\tilde{\mathscr{K}} = \mathcal{A}_a^{-1} \circ \mathcal{A}_a(\mathscr{K})$ is the minimal controllable language containing $K \cap L_{\rm out}(\mathcal{A}_a)$, and $\mathscr{K}$ is controllable if $L_{\rm out}(\mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_a^{-1} \circ \mathcal{A}_a) \subseteq \mathscr{K}$ and $\mathscr{K} \subseteq L_{\rm out}(\mathcal{A}_a)$.*

VIII. Controllability for Systems with Nondeterministic Plants

In this section, we extend the attack-resilient controllability theorems and supervisor synthesizing algorithms derived in Sections V to VII to systems with nondeterministic plants that are nonblocking, as formally stated in Definition 3.

Definition 3 (Nonblocking)**.** *The plant $\mathcal{P} = (\mathsf{S}, \mathsf{s}_{\rm init}, \mathbf{I}, \mathbf{O}, \mathsf{Trans}, \mathsf{S}_{\rm final})$ is nonblocking for the regular relation $\mathcal{R} \subseteq \mathcal{R}_{\mathcal{P}}$ if for any $(I, O) \in \mathcal{R}$ and $\mathscr{K} \in L_{\rm in}(\mathcal{P})$,*

$$\exists \mathsf{s} \in \mathit{Reach}_{(I,O)}(\mathsf{s}_{\rm init}), \mathsf{Trans}(\mathsf{s}, \mathsf{i}, \cdot, \cdot) \neq \emptyset \\ \Longrightarrow \forall \mathsf{s} \in \mathit{Reach}_{(I,O)}(\mathsf{s}_{\rm init}), \mathsf{Trans}(\mathsf{s}, \mathsf{i}, \cdot, \cdot) \neq \emptyset;$$

namely, if a symbol i is accepted by some s in the reachable set $\mathit{Reach}_{(I,O)}(\mathsf{s}_{\rm init})$ of an acceptable pair of input and output words (I, O) to the plant, then this symbol should be accepted by any state in the reachable set.

This property ensures that regardless of the nondeterministic past executions taken by the plant that corresponds to observed inputs and outputs, the next accepting symbol can always be executed. It rules out the situation in Figure 3 where the execution of the second input symbol i_2 or i_3 depends on the nondeterministic execution for the first input/output symbols $\mathsf{i}_1|\mathsf{i}_3$. Specially, a deterministic plant is nonblocking as there is only one execution corresponding to any accepting input.

The nonblocking property of a nondeterministic FST can be checked via the well-known powerset construction. Specifically, we treat $\mathcal{P}$ as a nondeterministic automaton as discussed in Remark 2 and determinize it by the powerset construction to

Algorithm 7 Check nonblocking for nondeterministic plants

Require: Plant $\mathcal{P}$, regular relation $\mathcal{R}$
1: Compute determinization $\mathcal{P}^D$ by (12).
2: Find automaton model $\mathcal{M}_{\mathcal{R}}$ and compute composition $\mathcal{M}_{\mathcal{R}} \circ \mathcal{P}^D$.
3: **if** (13) holds for all transitions in Trans^D **then**
4: **return** $\mathcal{P}$ is nonblocking.
5: **else**
6: **return** $\mathcal{P}$ is blocking.
7: **end if**

$\mathcal{P}^D = (\mathsf{S}^D, \{\mathsf{s}_{\rm init}\}, \mathbf{I}^D, \mathsf{Trans}^D, \mathsf{S}^D_{\rm final})$ where $\mathsf{S}^D, \mathsf{S}^D_{\rm final} \subseteq 2^{\mathsf{S}}$, $\mathbf{I}^D = (\mathbf{I} \cup \{\varepsilon\}) \times (\mathbf{O} \cup \{\varepsilon\}) \backslash \{(\varepsilon, \varepsilon)\}$ and

$$\mathsf{Trans}^D = \big\{ \big(\mathsf{S}_1, (\mathsf{i}, \mathsf{o}), \mathsf{S}_2\big) | \mathsf{S}_1 \subseteq \mathsf{S}, (\mathsf{i}, \mathsf{o}) \in \mathbf{I}^D, \\ \mathsf{S}_2 = \cup_{\mathsf{s} \in \mathsf{S}_1} \big(\mathsf{Trans}(\mathsf{s}, \mathsf{i}, \mathsf{o}, \cdot) \cup \mathsf{Trans}(\mathsf{s}, \varepsilon, \varepsilon, \cdot)\big) \big\}. \tag{12}$$

Now, the nonblocking property of $\mathcal{P}$ can be checked on $\mathcal{P}^D$ as captured by Lemma 2.

Lemma 2 (Checking Nonblocking)**.** *The plant $\mathcal{P}$ is nonblocking for the regular relation $\mathcal{R} \subseteq \mathcal{R}_{\mathcal{P}}$ if and only if for any execution $\big(\{\mathsf{s}_{\rm init}\}, (\mathsf{i}_1, \mathsf{o}_1), \mathsf{S}_1\big) \dots \big(\mathsf{S}_{n-1}, (\mathsf{i}_n, \mathsf{o}_n), \mathsf{S}_n\big)$ of $\mathcal{P}^D$ with $(\mathsf{i}_1 \dots \mathsf{i}_n, \mathsf{o}_1 \dots \mathsf{o}_n) \in \mathcal{R}$, we have for $i \in [n]$ that*

$$\mathsf{S}_n = \cap_{\mathsf{s} \in \mathsf{S}_{n-1}} \mathsf{Trans}(\mathsf{s}, \mathsf{i}_n, \mathsf{o}_n, \cdot). \tag{13}$$

Proof. First, we note that the determinized automaton $\mathcal{P}^D$ is free from ε-moves, because all its states are closed under the ε-moves of $\mathcal{P}$. By construction, $\mathcal{P}^D$ and $\mathcal{P}$ accept the same regular language/relation, and the states of $\mathcal{P}^D$ correspond to the reachable sets of $\mathcal{P}$. Thus, combining (13) and (12) gives Definition 3, and vice versa. □

Let $\mathcal{M}_{\mathcal{R}}$ be an automaton model for the regular relation $\mathcal{R}$ treated as a regular language, then the composition $\mathcal{M}_{\mathcal{R}} \circ \mathcal{P}^D$ restricts the determinized automaton $\mathcal{P}^D$ to its sublanguage $\mathcal{R}$. This fact and Lemma 2 lead to Algorithm 7.

The controllability theorems and supervisor synthesizing algorithms extend from deterministic plants to nondeterministic plants, if and only if the control words of the supervisor can be executed without blocking for the composition of the rest of the closed-loop system. This immediately leads to Theorem 4.

Theorem 4. *Theorems 1 to 3 and Algorithms 3, 5 and 6 are valid for plants modeled by nondeterministic FSTs, if and only if the composed FSTs $\mathcal{P} \circ \mathcal{A}_s$, $\mathcal{A}_a \circ \mathcal{P}$ and $\mathcal{A}_a \circ \mathcal{P} \circ \mathcal{A}_s$ are nonblocking for the regular relation induced by the supervisor $\mathcal{S}$, respectively.*

m	n	$\lvert \mathsf{S}_{\mathcal{M}_{\mathscr{K}}} \rvert$	Time (ms)	Memory (MB)
9	2	10^2	15.566	0.20944
9	3	10^3	16.309	2.1642
99	2	10^4	20.852	47.563
9	5	10^5	99.535	340.26
99	3	10^6	721.68	7106.7

TABLE I: Execution time and memory usage of the supervisory synthesizing algorithm for different values of n and m.

Finally, note that when plant $\mathcal{P}$ is deterministic, the composed FSTs $\mathcal{P} \circ \mathcal{A}_s$, $\mathcal{A}_a \circ \mathcal{P}$ and $\mathcal{A}_a \circ \mathcal{P} \circ \mathcal{A}_s$ are deterministic and satisfy (13). Unlike in case with deterministic plants, sensor attacks can influence the controllability of nondeterministic plants by blocking the control words of the supervisor.

IX. ARSC Tool and Synthesis Scalability

Based on the proposed algorithms for synthesis of Attack-Resilient Supervisory Controllers we developed an open-source tool *ARSC*, available at [1]; the tool exploits OpenFst libraries [2]. In this section, we illustrate its efficiency on problems on different scales. For all evaluations, the tool was executed on an Intel Core i7-7700K CPU, and the execution time and memory usage were measured.

To illustrate effectiveness of our approach, we consider a scheduling problem from [6], where n players independently requiring service for m sequential tasks $t_{ij},\ i \in [n], j \in [m]$ on a central server. The tasks of each player have to be served in the index order. The sensors are corrupted by an attacker that removes the tasks performed by the first player; and the actuator attacks nondeterministically rotate the input sequence $t_{1j}t_{2j}\dots t_{(n-1)j}t_{nj}$ to $t_{2j}t_{3j}\dots t_{nj}t_{1j}$ for any task index $j \in [m]$. For $n = 2, m = 2$, the desired language $\mathscr{K}$ for the system, as well as the attacks are modeled as shown in Figure 10. From Theorem 3, the language $\mathscr{K}$ is controllable and the attacks can be countered by the supervisor constructed by Algorithm 6. The supervisor for the case $n = 2, m = 2$ is displayed in Figure 10e.

The complexity of the supervisor synthesis algorithms is determined by the complexity of the composition operation. The composition $\mathcal{A}_1 \circ \mathcal{A}_2$ requires $O(|\mathsf{S}_{\mathcal{A}_1}||\mathsf{S}_{\mathcal{A}_2}|D_{\mathcal{A}_1}(\log(D_{\mathcal{A}_2}) + M_{\mathcal{A}_2}))$ time and $O(|\mathsf{S}_{\mathcal{A}_1}||\mathsf{S}_{\mathcal{A}_2}|D_{\mathcal{A}_1}M_{\mathcal{A}_2})$ space where $|\mathsf{S}_{\cdot}|$, $D_{\cdot}$ and $M_{\cdot}$ denote the number of states, the maximum out-degree and the maximum multiplicity for the FST, respectively [2]. The order in which the composition operations are performed can also change the overall complexity. For simplicity, the term $\mathcal{P}^{-1}$ is dropped and the supervisor is computed as $(\mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}}) \circ \mathcal{A}_a^{-1}$ in our implementation. Therefore, the overall time complexity is reduced to $O(|\mathsf{S}_{\mathcal{M}_{\mathscr{K}}}||\mathsf{S}_{\mathcal{A}_a}|D_{\mathcal{A}_s}\log(D_{\mathcal{A}_a}))$ where $\mathsf{S}_{\mathcal{M}_{\mathscr{K}}} \sim O((m+1)^n)$ and $\mathsf{S}_{\mathcal{A}_a}{=}D_{\mathcal{A}_s}{=}D_{\mathcal{A}_a} \sim O(mn)$ for this problem.

Table I shows the running times of the algorithm averaged over 100 synthesis and the maximum amount of memory used during the tool execution for different values of n and m. We can observe that a tenfold increase in the number of states in $\mathcal{M}_{\mathscr{K}}$ increases the execution time and the memory usage by at most 100 times. This sub-quadratic increase is a consequence of the composition operations performed by the algorithm.

X. Conclusions

In this work, we have studied the problem of supervisory control of discrete-event plants in the presence of attacks on the plant's sensors and actuators. We have considered a very general class of attacks that have the ability to nondeterministically rewrite a word to any word of a regular language, and proposed to model them by FSTs that possess a library of mathematically rigorous and computationally feasible operations, such as inversion and composition. Furthermore, we have onsidered a general supervisor model where the supervisors are also captured by FSTs.

We have first focused on the attack-resilient supervisory control problem for deterministic plants in three setups where attacks occur on the plant's: (i) sensors, (ii) actuators, and (iii) both actuators and sensors; we have introduced new sets of controllability theorems and synthesis algorithms for attack-resilient supervisors. We have shown that for (i), the attacks on sensors can be countered by a supervisor derived by the serial composition of the inversion of the attacker and a model of the desired language; for (ii), the attacks on actuators can be partly countered by a supervisor derived by the serial composition of a model of the desired language and the inversion of the attacker; and for (iii), a supervisor can be derived by serially composing the supervisors in the cases (i) and (ii). The above results have been also extended to nondeterministic plants with the nonblocking conditions. Finally, we have introduced a tool for synthesis of such attack-resilient supervisors and demonstrated its scalability. An avenue for future work is to resolve the nondeterminism optimally when there are different costs for the supervisor to revise the control symbols.

References

[1] ARSC. https://github.com/alperkamil/arsc. Accessed: 2019-03-08.

[2] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst: A General and Efficient Weighted Finite-State Transducer Library. In J. Holub and J. Žďárek, editors, *Implementation and Application of Automata*, Lecture Notes in Computer Science, pages 11–23. Springer Berlin Heidelberg, 2007.

[3] A. A. Cardenas, S. Amin, and S. Sastry. Secure Control: Towards Survivable Cyber-Physical Systems. In *2008 The 28th International Conference on Distributed Computing Systems Workshops*, pages 495–500, 2008.

[4] L. K. Carvalho, Y.-C. Wu, R. Kwong, and S. Lafortune. Detection and mitigation of classes of attacks in supervisory control systems. *Automatica*, 97:121–133, 2018.

[5] C. G. Cassandras. Smart Cities as Cyber-Physical Social Systems. *Engineering*, 2(2):156–158, 2016.

[6] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, New York, NY, 2. ed edition, 2008.

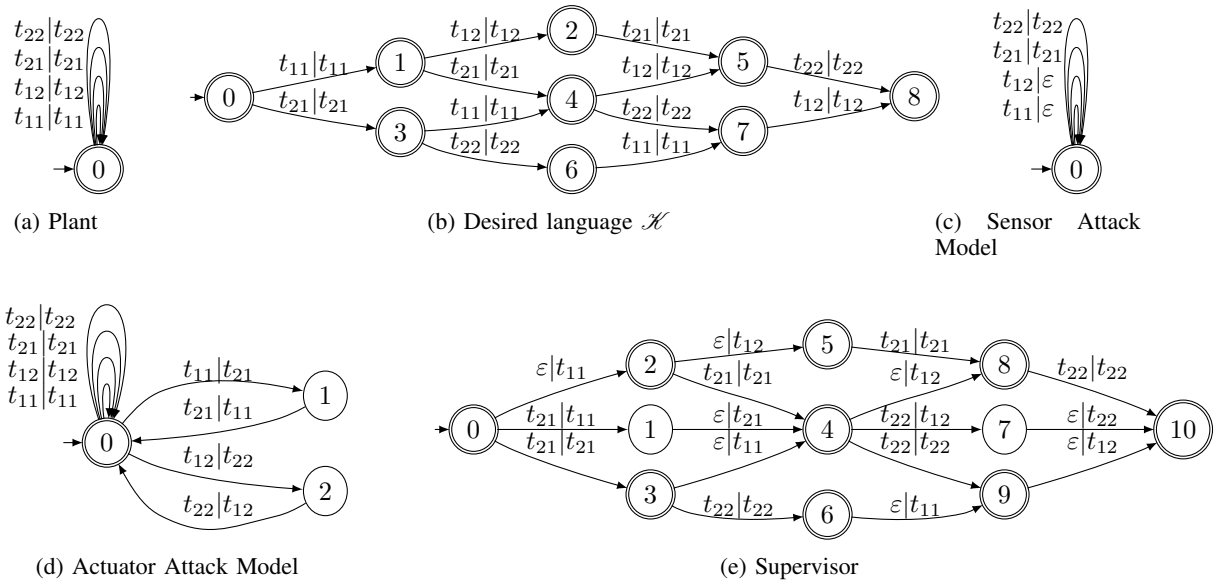
[7] T. M. Chen and S. Abu-Nimeh. Lessons from stuxnet. *Computer*, 44(4):91–93, April 2011.

[8] M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, 2009.

[9] J. P. Farwell and R. Rohozinski. Stuxnet and the future of cyber war. *Survival*, 53(1):23–40, 2011.

[10] H. Fawzi, P. Tabuada, and S. Diggavi. Secure estimation and control for cyber-physical systems under adversarial attacks. *IEEE Trans. Autom. Control*, 59:1454–1467, 2014.

[11] H. Fawzi, P. Tabuada, and S. Diggavi. Secure Estimation and Control for Cyber-Physical Systems Under Adversarial Attacks.



(a) Plant (b) Desired language $\mathscr{K}$ (c) Sensor Attack Model

(d) Actuator Attack Model (e) Supervisor

Fig. 10: Example supervisors resilient to sensor and actuator attacks.

IEEE Transactions on Automatic Control, 59(6):1454–1467, 2014.

[12] R. M. Goes, E. Kang, R. Kwong, and S. Lafortune. Stealthy deception attacks for cyber-physical systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 4224–4230, Melbourne, Australia, 2017. IEEE.

[13] M. Holcombe. *Algebraic Automata Theory*. Cambridge University Press, 1982.

[14] Z. Jiang, M. Pajic, and R. Mangharam. Cyber-Physical Modeling of Implantable Cardiac Medical Devices. *Proceedings of the IEEE*, 100(1):122–137, Jan 2012.

[15] I. Jovanov and M. Pajic. Relaxing integrity requirements for attack-resilient cyber-physical systems. *IEEE Transactions on Automatic Control*, pages 1–1, 2019. to appear.

[16] V. Lesi, I. Jovanov, and M. Pajic. Security-aware scheduling of embedded control tasks. *ACM Trans. Embed. Comput. Syst.*, 16(5s):188:1–188:21, Sept. 2017.

[17] F. Miao, M. Pajic, and G. Pappas. Stochastic game approach for replay attack detection. In *IEEE 52nd Annual Conference on Decision and Control (CDC)*, pages 1854–1859, Dec 2013.

[18] Y. Mo, T.-H. Kim, K. Brancik, D. Dickinson, H. Lee, A. Perrig, and B. Sinopoli. Cyber–physical security of a smart grid infrastructure. *Proceedings of the IEEE*, 100(1):195–209, 2012.

[19] Y. Mo and B. Sinopoli. Secure control against replay attacks. In *47th Annual Conference on Communication, Control, and Computing (Allerton)*, pages 911–918, Sept 2009.

[20] M. Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23:42, 1997.

[21] M. Mohri. Weighted Finite-State Transducer Algorithms. An Overview. In J. Kacprzyk, C. Martín-Vide, V. Mitrana, and G. Păun, editors, *Formal Languages and Applications*, volume 148, pages 551–563. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[22] M. Mohri. Weighted Automata Algorithms. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, pages 213–254. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[23] M. Mohri, F. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[24] M. Pajic, I. Lee, and G. J. Pappas. Attack-Resilient State Estimation for Noisy Dynamical Systems. *IEEE Transactions on Control of Network Systems*, 4(1):82–92, 2017.

[25] M. Pajic, J. Weimer, N. Bezzo, O. Sokolsky, G. J. Pappas, and I. Lee. Design and implementation of attack-resilient cyberphysical systems: With a focus on attack-resilient state estimators. *IEEE Control Systems*, 37(2):66–81, April 2017.

[26] F. Pasqualetti, F. Dorfler, and F. Bullo. Control-theoretic methods for cyberphysical security: Geometric principles for optimal cross-layer resilient control systems. *IEEE Control Systems*, 35(1):110–127, Feb 2015.

[27] A. H. Rutkin. Spoofers Use Fake GPS Signals to Knock a Yacht Off Course, www.technologyreview.com/news/517686/spoofers-use-fake-gps-signals-to-knock-a-yacht-off-course, 2013.

[28] J. Sakarovitch and R. Thomas. Elements of Automata Theory. page 784, 2003.

[29] D. Shepard, J. Bhatti, and T. Humphreys. Drone hack. *GPS World*, 23(8):30–33, 2012.

[30] Y. Shoukry, P. Martin, P. Tabuada, and M. Srivastava. Non-invasive spoofing attacks for anti-lock braking systems. In *Cryptographic Hardware and Embedded Systems-CHES 2013*, pages 55–72. Springer, 2013.

[31] R. Smith. A decoupled feedback structure for covertly appropriating networked control systems. *Proc. IFAC World Congress*, pages 90–95, 2011.

[32] R. Su. Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations. *Automatica*, 94:35–44, 2018.

[33] A. Teixeira, D. Pérez, H. Sandberg, and K. H. Johansson. Attack models and scenarios for networked control systems. In *Conf. on High Confid. Net. Sys. (HiCoNS)*, pages 55–64, 2012.

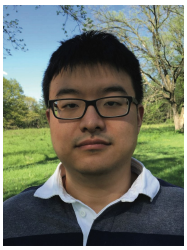
[34] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun. On the requirements for successful gps spoofing attacks. In *18th ACM Conf. on Computer and Com. Security*, CCS, pages 75–86, 2011.

[35] M. Wakaiki, P. Tabuada, and J. P. Hespanha. Supervisory Control of Discrete-event Systems under Attacks. *arXiv:1701.00881 [cs, math]*, 2017.

[36] Y. Wang, Z. Huang, S. Mitra, and G. E. Dullerud. Differential privacy in linear distributed control systems: Entropy minimizing mechanisms and performance tradeoffs. *IEEE Transactions on Control of Network Systems*, 4(1):118–130, 2017.

[37] Y. Wang and M. Pajic. Supervisory control of discrete event systems in the presence of sensor and actuator attacks. In *Decision and Control (CDC), 2019 IEEE 58rd Annual Conference On*, page Under Review, 2019.

[38] J. S. Warner and R. G. Johnston. A simple demonstration that the global positioning system (gps) is vulnerable to spoofing. *Journal of Security Administration*, 25(2):19–27, 2002.
[39] A. D. Wood and J. A. Stankovic. Denial of service in sensor networks. *computer*, 35(10):54–62, 2002.



Yu Wang is currently a Postdoctoral Associate in the Department of Electrical and Computer Engineering at Duke University. He received his Ph.D. degree in Mechanical Engineering and his M.S. degrees in Statistics, Mathematics, and Mechanical Engineering in 2018, 2017, 2016 and 2014, respectively from the University of Illinois at Urbana-Champaign. Before that, he received his B.S. degree in Engineering Mechanics from the School of Aerospace of Tsinghua University in 2012.

Alper Kamil Bozkurt received the B.S. and M.S. degrees in computer engineering from Bogazici University, Turkey, in 2015 and 2018, respectively. He is currently a Ph.D. student in the Department of Computer Science at Duke University. His research interests lie at the intersection of machine learning, control theory, and formal methods. In particular, he focuses on developing learning based algorithms that synthesize provably safe and reliable controllers for cyber-physical systems.

Miroslav Pajic (S'06-M'13-SM'19) received the Dipl. Ing. and M.S. degrees in electrical engineering from the University of Belgrade, Serbia, in 2003 and 2007, respectively, and the M.S. and Ph.D. degrees in electrical engineering from the University of Pennsylvania, Philadelphia, in 2010 and 2012, respectively.

He is currently the Nortel Networks Assistant Professor in the Department of Electrical and Computer Engineering at Duke University. He also holds a secondary appointment in the Computer Science Department. His research interests focus on the design and analysis of cyber-physical systems, and in particular security-aware distributed/networked control systems, embedded systems, and high-confidence control systems.

Dr. Pajic received various awards including the ACM SIGBED Early-Career Award, NSF CAREER Award, ONR Young Investigator Program Award, ACM SIGBED Frank Anger Memorial Award, Joseph and Rosaline Wolf Best Dissertation Award from Penn Engineering, IBM Faculty Award, as well as six Best Paper and Runner-up Awards, such as the Best Paper Awards at the 2017 ACM SIGBED International Conference on Embedded Software (EMSOFT) and 2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), and the Best Student Paper award at the 2012 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). He is an associate editor in the ACM Transactions on Computing for Healthcare (ACM HEALTH) and a co-chair of the 2019 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS'19).