IEEE CSS

Received XX September 2021; revised ; accepted XX October 2021; Date of publication XX November 2021; date of current version XX November 2021. This paper was recommended by Associate Editor XXX.

Digital Object Identifier 10.1109/OJCSYS.2021.Doi Number

Attack-Resilient Supervisory Control of Discrete-Event Systems: A Finite-State Transducer Approach

Yu Wang[1], Alper Kamil Bozkurt[2], Nathan Smith[1], Miroslav Pajic[2]

[1]Department of Mechanical & Aerospace Engineering, University of Florida, Gainesville, FL 32611, USA.
[2]Department of Electrical & Computer Engineering, Duke University, Durham, NC 27710, USA.

CORRESPONDING AUTHOR: Yu Wang (e-mail: yuwang1@ufl.edu)

This work was partly supported by the awards ONR N00014-17-1-2012 and N00014-17-1-2504, AFOSR FA9550-19-1-0169, and NSF CNS-1652544.

ABSTRACT Resilience to sensor and actuator attacks is a major concern in the supervisory control of discrete events in cyber-physical systems (CPS). In this work, we propose a new framework to design supervisors for CPS under attacks using finite-state transducers (FSTs) to model the effects of the discrete events. FSTs can capture a general class of regular-rewriting attacks in which an attacker can nondeterministically rewrite sensing/actuation events according to a given regular relation. These include common insertion, deletion, event-wise replacement, and finite-memory replay attacks. We propose new theorems and algorithms with polynomial complexity to design resilient supervisors against these attacks. We also develop an open-source tool in Python based on the results and illustrate its applicability through a case study.

INDEX TERMS cyber-physical systems, formal methods, control system security

I. Introduction

Supervisory control is a widely-used high-level control technique to deal with discrete events (e.g., turning on/off one of many switches) in cyber-physical systems (CPS) that work for various applications including transportation [1, 2], smart infrastructure [3], and healthcare [4]. The goal of the supervisor, typically implemented by cyber controllers, is to ensure that the possible discrete events happen in the correct sequences to prevent system failure. Mathematically, the discrete events are captured by a set of symbols that are implemented by sensors and actuators, their effect on the controlled physical plant is captured by finite-state machines whose transitions are driven by these symbols, and the goal is to apply feedback control to restrict the plant's execution, as shown in Fig. 1.

With increasing applications in contested scenarios, such as autonomous driving [5, 2] and distributed manufacturing [6, 7], there is a growing interest in ensuring the resilience of supervisors against sensor/actuator attacks [8, 9, 10]. In supervisory control, the possible attacks are illustrated in Fig. 1. The sensor attacks aim to corrupt the true sensing symbols from the plant to the supervisor by either hacking into the plant's sensor [11, 12, 13] or the communication network [14, 15, 16, 17]. Similarly, the actuator attacks aim to corrupt the true actuating symbols from the supervisor to the plant by either hacking into the plant's actuator or the communication network. This setup captures simultaneous (including coordinated) attacks on sensors and actuators and is more general than [18, 19] for either sensor or actuator attacks.

The block diagram for supervisory control under sensor/actuator attacks is shown in Fig. 2. In a real system, multiple sensor attacks can happen via different vectors (e.g., network or sensor). In Fig. 2, the overall effect is represented by a single attacker $\mathcal{A}_s$. The input-output relation of $\mathcal{A}_s$ captures how a sequence of true sensing symbols may be nondeterministically rewritten into a sequence of corrupted sensing symbols. Similarly, the overall effect of multiple possible actuator attacks is represented by a single attacker $\mathcal{A}_a$, whose input-output relation captures how a sequence of true actuator symbols may be nondeterministically rewritten into a sequence of corrupted actuator symbols.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

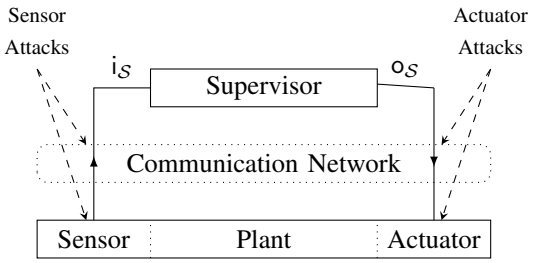


FIGURE 1: Supervisory control of cyber-physical systems. At each time, the plant sends a sensing symbol, which will be corrupted by $o_{\mathcal{P}}$ to the supervisor. Then, the supervisor replies to the plant with an actuating symbol $i_{\mathcal{P}}$. A plant transition can only happen if the pair $(i_{\mathcal{P}}, o_{\mathcal{P}})$ matches the transition's label. In practice, the symbols $o_{\mathcal{P}}$ and $i_{\mathcal{P}}$ may be revised by malicious attackers, causing the plant to make transitions not allowed by the supervisor.

The common presence of attackers in CPS has motivated recent works on developing new theories and algorithms for attack-resilient supervisory control. However, many of them only deal with simple and history-independent attack strategies, e.g., symbol insertion, deletion, or replacement attacks (e.g., [18]), while many attackers in CPS can use complex and history-dependent attack strategies. Among the works involving history dependent attacks (e.g., [20]), the attacks are typically modeled by a function: the attack is determined by the plant's transition history. Our attacker model is described by an FST giving an intuitive visual representations of the attackers.

Take the replay attack [21, 22] as an example, which is a common strategy for launching cyber-attacks. It works by first recording a fragment of symbols and then replaying it repeatedly to trap the system. Implementing this attack requires the attacker to decide when to continue recording or start replaying. This requires the attackers to possess internal states to make such decisions based on previous actions. The idea of using states to model attack behaviors has been proposed in [20]. However, in that work, the state space is shared between the attack and the system. In practice, the attacker and the system are typically implemented separately. Thus, it is more appropriate to model the attacker individually with its own state space. This will help study the impact of different attacks on the same plant and study the impact of multiple combined attacks.

This work proposes to use finite-state transducers (FSTs) [23, 24], which generalize finite-state automata, to model the complex and history-dependent strategies of the attackers. These FST models can be viewed as abstractions of cyber/network attacks implemented by embedded programs (e.g., malware). FST transitions are driven by an input symbol and also produce a different output. This feature allows FSTs to capture general complex attack strategies, including previously-studied attacks (e.g., symbol insertion, deletion, or replacement) and new history-dependent attacks (e.g., replay attacks). In addition, one can easily compose multiple attack strategies to form new attack strategies via FST model compositions [25, 26, 27, 28], which have polynomial complexity and are well supported by existing C++/Python libraries (e.g., [29]). Finally, one can implement constraints on unconstrained attack strategies and facilitate problem analysis with common security measures (e.g., intrusion detectors [30], intermittently authentication [16, 31, 32]).

Our main contribution is the development of a new theory to synthesize resilient supervisors against sensor and actuator attacks. The supervisory control diagram is shown in Fig. 2. We assume the FST models for attackers are known a priori and capture all the possible attack behaviors (e.g., nondeterministic symbol deletion or insertion). The supervisor is resilient in the sense that it can restrict the plant's execution to an allowed set under the attacks. Our theory gives a constructive algorithm with polynomial complexity to synthesize the supervisor, if feasible. It also shows that the feasible supervisor is realizable by an FST.

This work improves our previous work [33] in two aspects. First, we generalize it to plants modeled by FSTs (instead of automata) to handle CPS equipped with sensors and actuators that yield different input and output symbols. Our plant model is similar to [34, 35], although no attacks were considered in those works. Second, we develop an open-source tool in Python based on the new results and illustrate its applicability through a case study. The rest of the paper is organized as follows. We provide preliminaries on FSTs and regular relations in Section II and formalize the problem in Section III. Then, we demonstrate the advantages of using FSTs to model attacks in Section IV and provide resiliency conditions and algorithms to design resilient supervisors for FST-based sensor and actuator attacks in Section IV. Finally, case studies are presented in Section VI, before concluding in Section VII.

Related work

Supervisory control under attacks has been studied extensively in the literature, so we provide a detailed comparison with previous work as follows. This work considers simultaneous actuator and sensor attacks, while only sensor attacks are considered in [36]. Furthermore, our attack model allows symbol revision of unbounded length, while [36] only studied bounded attacks. We notice that [37, 38] consider simultaneous actuator and sensor attacks. Our work can handle regular desired languages while [37] only focuses on reachability. In addition, we provide an explicit attack model via FST, while the attack model is only given implicitly in [37] through the attacked plant model. Our attack model can be history-dependent and thus more complex than the history-independent attack model in [38]. In addition, we deal with the regular desired languages instead of liveness in [38]. The problem of synthesizing complex attackers has been studied in [39, 40, 41, 20, 42, 43], while our work focuses on the supervisor synthesis problem. Our work can synthesize supervisors to counter the attacks, while [44]

only focuses on synthesizing estimators to detect the attacks. Our work can handle attackers that tamper with the system dynamics, while [45] can only handle eavesdroppers. Compared to [10] that deals with supervisor design under attacks, we focus on restricting the plant to a desired language $\mathscr{K}$, while the supervisor in [10] prevents the plant from reaching unsafe states. Consequently, their supervisor synthesis problem is converted to an equivalent game theoretic dual problem. Lastly, our framework handles both sensor and actuator attackers simultaneously while [10] is restricted to only sensor attackers.

II. Preliminaries on FSTs

To start with, we introduce the following common notations for arithmetic over symbols. Specifically, we denote the set of natural numbers including zero by $\mathbb{N}$, set cardinality by $|S|$, power set by 2^S, and set subtraction by $S_1 \backslash S_2 = \{s \in S_1 \mid s \notin S_2\}$. We write "iff" for "if and only if". For $n \in \mathbb{N}$, let $[n] = \{1, .., n\}$. We follow the common notations for sequences. For a given finite set of symbols, a finite-length sequence of symbols is called a word. A set of words is called a language. A word can be viewed as a singleton language. The empty symbol/word is denoted by ε. The concatenation of two languages (or words) is defined by

$$L_1 L_2 = \{I_1 I_2 \mid I_1 \in L_1, I_2 \in L_2\}.$$

The Kleene star (i.e., finite repetition) of a language (or word) is defined by $L^* = \{I_1 \dots I_n \mid I_1, \dots, I_n \in L, n \in \mathbb{N}\}$; by convention, $\varepsilon \in L^*$. For singleton sets, the notation convention is $I_1^* = \{I_1\}^*$. The union of two languages (or words) is the same as the union of sets, denoted by $L_1 \cup L_2$. A language is *regular* if it can be represented only using union, concatenation, and Kleene star [46]. In addition, a word I_1 is a prefix of another word I if there exists I_2 such that $I = I_1 I_2$. The prefix closure $\overline{L}$ of a language L is derived by including all prefixes of all $I \in L$ in $\overline{L}$. A language L is prefix-closed if $L = \overline{L}$.

Now, we provide a mathematical introduction to FSTs. FSTs can be seen as automata with input and output symbols on their transitions. Similarly, automata can be viewed as FSTs with identical input and output.

Definition 1. *A 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 non-empty input symbols;*
- $\mathbf{O}$ *is a finite set of non-empty output symbols;*
- $\mathsf{Trans} \subseteq \mathsf{S} \times (\mathbf{I} \cup \varepsilon) \times (\mathbf{O} \cup \varepsilon) \times \mathsf{S}$ *is a transition relation, where* ε *is the empty symbol;*
- $(s, \varepsilon, \varepsilon, s) \in \mathsf{Trans}$ *for all* $s \in S$*;*
- $\mathsf{S}_{\text{final}} \subseteq \mathsf{S}$ *is a finite set of final states.*

In Definition 1, the symbol ε stands for the empty symbol. An FST transition with ε as the input symbol can self-trigger. An FST transition with ε as the output symbol yields no output symbol. The self-loop $(s, \varepsilon, \varepsilon, s)$ means the FST can stay in the same state without receiving an input symbol and generating an output symbol.

Languages of an FST

For the FST $\mathcal{A}$, we define an *execution* by a sequence of transitions $(\mathsf{s}_0, \mathsf{i}_1, \mathsf{o}_1, \mathsf{s}_1) \dots (\mathsf{s}_{n-1}, \mathsf{i}_n, \mathsf{o}_n, \mathsf{s}_n)$ where $\mathsf{s}_0 = \mathsf{s}_{\text{init}}$ and $(\mathsf{s}_{i-1}, \mathsf{i}_i, \mathsf{o}_i, \mathsf{s}_i) \in \mathsf{Trans}$ for $i \in [n]$. The execution defines an allowed word of input/output pairs $(\mathsf{i}_1, \mathsf{o}_1) \dots (\mathsf{i}_n, \mathsf{o}_n)$. The set of such allowed words (i.e., sequences of input/output symbol pairs) is called the *language* of the FST $\mathcal{A}$, denoted by $L(\mathcal{A})$. In addition, we call $I = \mathsf{i}_1 \dots \mathsf{i}_n$ an allowed input word and $O = \mathsf{o}_1 \dots \mathsf{o}_n$ an allowed output word. The set of allowed input words is the *input language* of $\mathcal{A}$, denoted by $L_{\text{in}}(\mathcal{A})$. Similarly, the *output language* is defined and denoted by $L_{\text{out}}(\mathcal{A})$.

FSTs and Relations

The FST $\mathcal{A}$ defines a *relation* between $\mathbf{I}^*$ and $\mathbf{O}^*$ by

$$\mathcal{R}_{\mathcal{A}} = \{(\mathsf{i}_1 \dots \mathsf{i}_n, \mathsf{o}_1 \dots \mathsf{o}_n) \mid \exists \text{ an execution } (\mathsf{s}_{\text{init}}, \mathsf{i}_1, \mathsf{o}_1, \mathsf{s}_1) \dots (\mathsf{s}_{n-1}, \mathsf{i}_n, \mathsf{o}_n, \mathsf{s}_n) \text{ and } \mathsf{s}_n \in \mathsf{S}_{\text{final}}\} \subseteq \mathbf{I}^* \times \mathbf{O}^*$$

In this case, we say the FST $\mathcal{A}$ realizes the relation $\mathcal{R}_{\mathcal{A}}$. More specifically, $\mathcal{R}_{\mathcal{A}}$ can be seen as a relation between the input language $L_{\text{in}}(\mathcal{A})$ and the output language $L_{\text{out}}(\mathcal{A})$, since $L_{\text{in}}(\mathcal{A}) = \mathcal{R}_{\mathcal{A}}^{-1}(\mathbf{O}^*)$ and $L_{\text{out}}(\mathcal{A}) = \mathcal{R}_{\mathcal{A}}(\mathbf{I}^*)$. Here, $\mathcal{R}_{\mathcal{A}}$ is not necessarily a function since an input word can be mapped nondeterministically to multiple output words.

To facilitate further discussion on relations, we introduce the following common notations. For a relation $\mathcal{R} \subseteq S_1 \times S_2$, we define (with a slight abuse of notation) the image of a subset $T_1 \subseteq S_1$ (an element is viewed as a singleton set) over the relation $\mathcal{R}$ by

$$\mathcal{R}(T_1) = \{s_2 \in S_2 \mid s_1 \in T_1, (s_1, s_2) \in \mathcal{R}\}.$$

The relation $\mathcal{R}$ is a partial function if $|\mathcal{R}(s_1)| \leq 1$ for any $s_1 \in S_1$. The inverse of the relation $\mathcal{R}$ is defined by

$$\mathcal{R}^{-1} = \{(s_2, s_1) \mid (s_1, s_2) \in \mathcal{R}\},$$

whereas the composition of two relations is defined by

$$\mathcal{R}_1 \circ \mathcal{R}_2 = \{(s_1, s_3) \mid \exists s_2.\ (s_1, s_2) \in \mathcal{R}_1, (s_2, s_3) \in \mathcal{R}_2\}.$$

Here, the relation composition is read from left to right.

FSTs and Automata

Like finite automata define regular languages, FSTs define *regular relations* as described below.

Definition 2. *For two finite sets* $\mathbf{I}$ *and* $\mathbf{O}$*, the relation* $\mathcal{R} \subseteq \mathbf{I}^* \times \mathbf{O}^*$ *is a regular relation iff*

$$\{(\mathsf{i}_1, \mathsf{o}_1) \dots (\mathsf{i}_n, \mathsf{o}_n) \mid (\mathsf{i}_1 \dots \mathsf{i}_n, \mathsf{o}_1 \dots \mathsf{o}_n) \in \mathcal{R}\}$$

is a regular language over $\mathbf{I} \times \mathbf{O}$*. Here,* $\mathsf{i}_1, \mathsf{o}_1, \dots \mathsf{i}_n, \mathsf{o}_n$ *can be the empty symbol* ε*.*

Since FSTs can be seen as automata with input and output symbols on their transitions, we have the following lemma [47].

Lemma 1. *The relation defined by an FST is regular. Any regular relation is realizable by an FST.*

III. Problem Formulation

This section introduces our problem formulation for attack-resilient supervisory control. Section III-A discusses how to properly use FST models in supervisory control. Section III-B presents the fundamentals of the supervisory control of plants modeled by FSTs without attacks. Section III-C discusses the supervisory control with attackers modeled by FSTs.

A. Using FST models in supervisory control

We use the FSTs from Definition 1 to model the plant in Figure 2. Since the FSTs are placed in a control loop, the transitions are implicitly timed. The FSTs are executed iteratively, and each iteration requires a unit of time.

Definition 3. *An FST $\mathcal{A}$ is observable if 1) it has no transition labeled by $(\varepsilon, \varepsilon)$ other than self-loops and 2) only one execution exists for an allowed word in $L(\mathcal{A})$.*

The observability from Definition 3 ensures that the executions of the FSTs, and thus their current states, are uniquely determined by both the previous input/output words. It can facilitate the hardware and software implementation of the plant and attackers. In addition, the observable FSTs from Definition 3 allow empty input or output symbols. In the supervisory control setup, an empty symbol means the plant or attackers give no input or output for the current time.

The observability described in Definition 3 strict the transition in the FST Definition 1 from a relation to a (partial) function $\mathsf{Trans} : \mathsf{S} \times \mathbf{I} \times \mathbf{O} \to \mathsf{S}$. An observable FST is different from a Mealy automaton whose transition is defined as the (partial) function $\mathsf{Trans} : \mathsf{S} \times \mathbf{I} \to \mathsf{S} \times \mathbf{O}$ [48]. Thus, observable FSTs can still capture nondeterministic attack behaviors that rewrite an input symbol into different output symbols. We will discuss these issues in detail in Section IV.

Two words $(\mathsf{i}_1, \mathsf{o}_1) \dots (\mathsf{i}_n, \mathsf{o}_n)$ and $(\mathsf{i}'_1, \mathsf{o}'_1) \dots (\mathsf{i}'_m, \mathsf{o}'_m)$ of the plant or attackers are viewed as different even if $\mathsf{i}_1 \dots \mathsf{i}_n = \mathsf{i}'_1 \dots \mathsf{i}'_m$ and $\mathsf{o}_1 \dots \mathsf{o}_n = \mathsf{o}'_1 \dots \mathsf{o}'_m$ after ignoring the empty symbols. More specifically, for a plant, $(\mathsf{i}_1, \varepsilon)(\varepsilon, \mathsf{o}_1)$ and $(\mathsf{i}_1, \mathsf{o}_1)(\varepsilon, \varepsilon)$ are different words. The former means the plant inputs i_1 and outputs nothing at Time 1 and inputs nothing and outputs i_2 at Time 2; the latter means the plant inputs i_1 and outputs i_2 at Time 1 and inputs and outputs nothing at Time 2.

An FST transition labeled with input/output symbols $(\varepsilon, \varepsilon)$ is similar to the ε-transitions in automata. They can happen spontaneously without receiving and generating any input and output. In addition, observing the FST's input and output cannot determine whether they have happened or not. Such unobservability is unrealistic and undesirable in modeling plants and attackers in the control loop. Thus, we focus on using observable FSTs in this work.

B. Supervisory control of FST plants

The plant modeled by FSTs can generate output symbols that are different from the input symbols. Thus, unlike classic supervisory control [49] using automata as supervisors, we use FSTs as supervisors to control such plants, as shown in Figure 2. Accordingly, the control loop executes iteratively as follows. Both the supervisor and the plant start from their initial states.

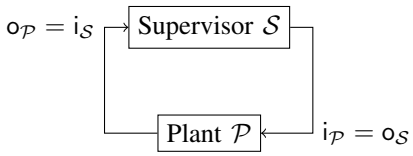


FIGURE 2: Block diagram for supervisory control of plants modeled by FSTs.

1) The supervisor (modeled by an FST) chooses an output symbol $\mathsf{o}_\mathcal{S}$, which will be sent to the plant, based on an eligible transition that has this symbol as its output.
2) The plant receives $\mathsf{i}_\mathcal{P} = \mathsf{o}_\mathcal{S}$ and makes a transition non-deterministically whose input symbol matches it. This transition will generate an output symbol $\mathsf{o}_\mathcal{P}$.
3) Upon receiving $\mathsf{i}_\mathcal{S} = \mathsf{o}_\mathcal{P}$, the supervisor completes this iteration by executing the transition labeled by $(\mathsf{i}_\mathcal{S}, \mathsf{o}_\mathcal{S}) = (\mathsf{o}_\mathcal{P}, \mathsf{i}_\mathcal{P})$. Otherwise, when there is no such transition, the supervisor will stop the control loop and raise an alarm.

In this setup, the supervisor has the freedom to choose the symbol $\mathsf{o}_\mathcal{S}$ in each iteration. Upon receiving $\mathsf{i}_\mathcal{P} = \mathsf{o}_\mathcal{S}$, the plant has the freedom to choose one of many transitions whose input symbol matches it. The set of all possible plant executions in the supervisory control loop can be captured by an FST derived from the loop composition defined below.

Definition 4. *The loop composition of the plant $\mathcal{P} = (\mathsf{S}_\mathcal{P}, \mathsf{s}_{i,\mathcal{P}}, \mathbf{I}_\mathcal{P}, \mathbf{O}_\mathcal{P}, \mathsf{Trans}_\mathcal{P}, \mathbf{S}_{f,\mathcal{P}})$ with the supervisor $\mathcal{S} = (\mathsf{S}_\mathcal{S}, \mathsf{s}_{i,\mathcal{S}}, \mathbf{I}_\mathcal{S}, \mathbf{O}_\mathcal{S}, \mathsf{Trans}_\mathcal{S}, \mathbf{S}_{f,\mathcal{S}})$ in Figure 2 is an FST given by $\mathcal{P}|\mathcal{S} = (\mathsf{S}_\mathcal{P} \times \mathsf{S}_\mathcal{S}, (\mathsf{s}_{i,\mathcal{P}}, \mathsf{s}_{i,\mathcal{S}}), \mathbf{I}_\mathcal{P}, \mathbf{O}_\mathcal{P}, \mathsf{Trans}_{loop}, \mathsf{S}_{f,\mathcal{P}} \times \mathsf{S}_{f,\mathcal{S}})$ where the transition $((\mathsf{s}_{1,\mathcal{P}}, \mathsf{s}_{1,\mathcal{S}}), \mathsf{i}_\mathcal{P}, \mathsf{o}_\mathcal{P}, (\mathsf{s}_{2,\mathcal{P}}, \mathsf{s}_{2,\mathcal{S}})) \in \mathsf{Trans}_{loop}$ if the following two conditions are met;*

1) $(\mathsf{s}_{1,\mathcal{P}}, \mathsf{i}_\mathcal{P}, \mathsf{o}_\mathcal{P}, \mathsf{s}_{2,\mathcal{P}}) \in \mathsf{Trans}_\mathcal{P}$
2) $(\mathsf{s}_{1,\mathcal{S}}, \mathsf{o}_\mathcal{P}, \mathsf{i}_\mathcal{P}, \mathsf{s}_{2,\mathcal{S}}) \in \mathsf{Trans}_\mathcal{S}$.

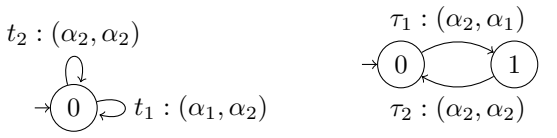
Accordingly, the plant's executions in the supervisory control loop in Figure 2 is given by $L(\mathcal{P}|\mathcal{S})$, i.e., the language of the loop composition $\mathcal{P}|\mathcal{S}$.

Definition 5. *Let $\mathcal{P}$ be an observable FST plant and $\mathscr{K} \subseteq L(\mathcal{P})$ be the prefix-closed regular language capturing its desired executions. The supervisor $\mathcal{S}$ controls the plant $\mathcal{P}$ to the language $\mathscr{K}$ if and only if the set of plant executions under the supervisor's regulation satisfies*

$$L(\mathcal{P}|\mathcal{S}) = \mathscr{K}.$$

Example 1. *Figure 3 shows an example of the supervisory control of FST plants. Suppose the plant is modeled by the FST shown by Figure 3a. It has one state 0 and two*

transitions t_1 and t_2 with input/output symbols (α_1, α_2) and (α_2, α_2), respectively. The goal is to restrict the plant execution to $\overline{(t_1 t_2)^*}$, or equivalently restrict the plant language to $\mathscr{K} = \overline{\big((\alpha_1, \alpha_2)(\alpha_2, \alpha_2)\big)^*}$. For this goal, consider a supervisor realized by an FST shown by Figure 3b, which has two states 0 and 1 and two transitions τ_1 and τ_2. At Time 1, only the transition τ_1 labeled by (α_2, α_1) is allowed by the supervisor whose current state is 0. Thus, only the input/input pair (α_1, α_2) is allowed on the plant, and only the transition t_1 can happen. The symbols are flipped since the plant's input is the supervisor's output and the plant's output is the supervisor's input. At Time 2, the supervisor state jumps to 1 following the transition τ_1. Now, only the transition τ_2 labeled by (α_2, α_2) is allowed by the supervisor. Thus, only the input/output pair (α_2, α_2) is allowed on the plant, and only the transition t_2 can happen. If the plant tries to execute the other transition t_1, the supervisor will stop the control loop and raise the alarm. Repeating the process, the supervisor restricts the plant's execution to $\overline{(t_1 t_2)^*}$ before stopping.



(a) Plant $\mathcal{P}$. (b) Supervisor $\mathcal{S}$.

FIGURE 3: Example of supervisory control of FST plants.

Based on the loop composition, we introduce the following theorem on the supervisory control of FST plants without attacks.

Theorem 1. *The observable FST $\mathcal{S}$ that has the language*

$$L(\mathcal{S}) = \big\{(\mathsf{o}_1, \mathsf{i}_1) \dots (\mathsf{o}_n, \mathsf{i}_n) \mid (\mathsf{i}_1, \mathsf{o}_1) \dots (\mathsf{i}_n, \mathsf{o}_n) \in \mathscr{K}\big\}$$

controls the $\mathcal{P}$ to the desired language $\mathscr{K}$.

Proof:
The FSTs $\mathcal{S}$ and $\mathcal{P}$ can be viewed as automata that take input/output symbol pairs. Thus, the theorem follows the classic supervisor control theory for automata [49]. Specifically, the FST realization $\mathcal{S}$ of the supervisor exists since $\mathscr{K}$ is regular. Because $\mathscr{K}$ is prefix-closed, any state of $\mathcal{S}$ is a final state, i.e., the executions of $\mathcal{S}$ can stop at any time. Finally, the symbols are flipped since the plant's input is the supervisor's output and the plant's output is the supervisor's input. ■

C. Including attacks in supervisory control

Now we consider simultaneous sensor and actuator attacks between the supervisor $\mathcal{S}$ and the plant $\mathcal{P}$ as shown in Figure 4.

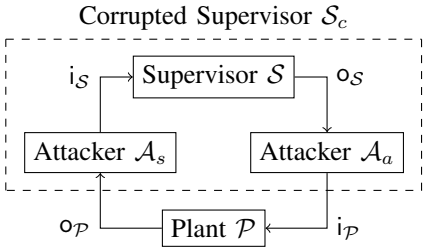


FIGURE 4: Block diagram for supervisory control under sensor and actuator attacks. The serial composition of $\mathcal{A}_a$, $\mathcal{P}$, and $\mathcal{A}_s$ in the dashed box form the corrupted supervisor. The executions of the plant $\mathcal{P}$ in this control loop is derived by its loop composition with the corrupted supervisor.

Attack Model

In practice, there can be many attacks (e.g., via hacking into sensors or communications) that corrupt the symbols sent between the plant to the supervisor. In this work, we propose to capture their overall effect by two FSTs, the actuator attacker $\mathcal{A}_a$ and the sensor attacker $\mathcal{A}_s$. The input-output relation of the FSTs $\mathcal{A}_a$ and $\mathcal{A}_s$ captures how a sequence of true sensing/actuating symbols can be revised nondeterministically into another sequence of corrupted symbols. Mathematically, these relations fall into the category of *regular relations* [47], according to Lemma 1. For simplicity, we assume $\mathcal{A}_s$ and $\mathcal{A}_a$ are observable so their internal executions can be captured by their languages.

The regular-rewriting attacks provide a general framework to model various attack behaviors. They generalize previous history-independent attacks (e.g., insert, delete, and replace) studied in [18, 50, 39, 19] and can capture new history-dependent attacks such as finite-memory replay attacks [21, 22]. Finally, we clarify that attackers modeled by observable FSTs can still rewrite input symbols in different ways to capture nondeterministic attack behaviors, according to Definition 3. We will discuss these issues in detail in Section IV.

Control Loop under Attacks

With the attackers $\mathcal{A}_s$ and $\mathcal{A}_a$, the supervisory control loop of Figure 4 works as follows. At each time, consider a state transition of the plant with the input and output symbols $\mathsf{i}_\mathcal{P}$ and $\mathsf{o}_\mathcal{P}$. Due to the attacks, the plant's true input and output symbols may be revised to corrupted symbols $\mathsf{i}_\mathcal{P}$ and $\mathsf{i}_\mathcal{S}$, respectively, before the output symbol is sent to the supervisor. The supervisor has no access to the true output symbol from the plant and can only see the corrupted symbol. The plant's transition is allowed if and only if the corrupted symbol pair $(\mathsf{i}_\mathcal{S}, \mathsf{o}_\mathcal{S})$ is allowed by the supervisor. Otherwise, the supervisor will stop the control loop and raises the alarm. This procedure is formalized with the following steps:

1) The supervisor (modeled by an FST) chooses an output symbol $\mathsf{o}_{\mathcal{S}}$, which will be sent to the plant, based on an eligible transition that has this symbol as its output.
2) The actuator attacker $\mathcal{A}_a$ receives $\mathsf{o}_{\mathcal{S}}$ and makes a transition whose input symbol matches $\mathsf{o}_{\mathcal{S}}$. There may be many such transitions and the attacker can choose anyone non-deterministically. This transition will generate an output symbol $\mathsf{i}_{\mathcal{P}}$ which will be sent to the plant.
3) The plant receives $\mathsf{i}_{\mathcal{P}}$ and non-deterministically makes a transition whose input symbol matches it. This transition will generate an output symbol $\mathsf{o}_{\mathcal{P}}$.
4) The sensor attacker $\mathcal{A}_s$ receives $\mathsf{o}_{\mathcal{P}}$ and non-deterministically makes a transition whose input symbol matches it. This transition will generate an output symbol $\mathsf{i}_{\mathcal{S}}$, which will be sent to the supervisor.
5) Upon receiving $\mathsf{i}_{\mathcal{S}}$., the supervisor completes this iteration by executing the transition labeled by $(\mathsf{i}_{\mathcal{S}}, \mathsf{o}_{\mathcal{S}}) = (\mathsf{o}_{\mathcal{P}}, \mathsf{i}_{\mathcal{P}})$. Otherwise, when there is no such transition, the supervisor will stop the control loop and raise an alarm.

Mathematically, the attacker FSTs are joined into the control loop with serial composition. Each attacker is serially composed with the supervisor to create the corrupted supervisor $\mathcal{S}_c$. We now describe how serial composition can be done.

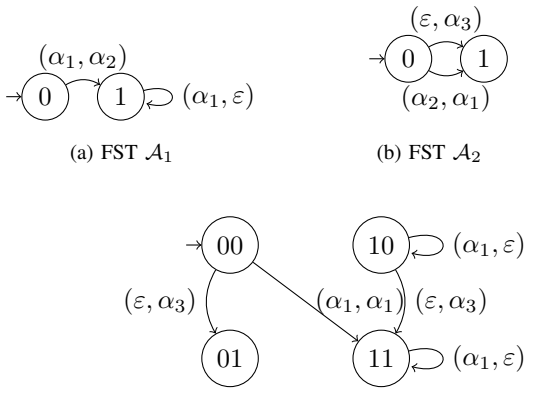
Definition 6. *Let* $\mathcal{A} = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{I}, \mathbf{O}, \mathsf{Trans}, \mathsf{S}_{\text{final}})$ *and* $\mathcal{A}' = (\mathsf{S}', \mathsf{s}'_{\text{init}}, \mathbf{I}', \mathbf{O}', \mathsf{Trans}', \mathsf{S}'_{\text{final}})$ *be two FSTs. The serial composition* $\mathcal{A}'' = \mathcal{A} \circ \mathcal{A}'$ *is derived by connecting the FSTs* $\mathcal{A}$ *and* $\mathcal{A}'$ *in series as shown in Fig. 7a, where the input word sequentially passes* $\mathcal{A}$ *and* $\mathcal{A}'$*. The composition is defined as the FST given by* $\mathcal{A}'' = (\mathsf{S} \times \mathsf{S}', (\mathsf{s}_{\text{init}}, \mathsf{s}'_{\text{init}}), \mathbf{I}, \mathbf{O}', \mathsf{Trans}'', \mathsf{S}_{\text{final}} \times \mathsf{S}'_{\text{final}})$*. A transition* $((s_1, s'_1), i, o', (s_2, s'_2))$ *if there exists an* $\alpha \in \mathbf{O} \cap \mathbf{I}'$ *with the following properties:*

1) (s_1, i, α, s_2) *is a valid transition in* Trans.
2) (s'_1, α, o', s'_2) *is a valid transition in* Trans'.

Lemma 2. *For any two FSTs* $\mathcal{A}_1$ *and* $\mathcal{A}_2$*, it holds that* $\mathcal{R}_{\mathcal{A}_1} \circ \mathcal{R}_{\mathcal{A}_2} = \mathcal{R}_{\mathcal{A}_1 \circ \mathcal{A}_2}$ *where* $\mathcal{R}$ *denotes the regular relation defined by an FST.*

The model derived from the series composition is still an observable FST. Since FSTs define regular relations, Definition 6 also provides an FST realization for the composition of regular relations in Lemma 2 [47]. We can now explicitly define the corrupted supervisor as $\mathcal{S}_c = \mathcal{A}_s \circ \mathcal{P} \circ \mathcal{A}_a$.

Example 2. *Consider two FSTs* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *shown in Fig. 5a and 5b, where one replaces* α_1 *with* α_2 *and the other injects* α_3*. The overall FST attack model, captured with the serial composition of the two FSTs is derived using Definition 6 and shown in Fig 5c. Also, it holds that* $\mathcal{R}_{\mathcal{A}_1 \circ \mathcal{A}_2} = \mathcal{R}_{\mathcal{A}_1} \circ \mathcal{R}_{\mathcal{A}_2}$.



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

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

FIGURE 5: Example for serial composition of FSTs.

Control Goal

Let $\mathscr{K}$ be a desired prefix-closed regular sub-language $\mathscr{K} \subseteq L(\mathcal{P})$ for the plant. The attackers aim to "stealthily" incur an undesired plant execution that yields a word outside $\mathscr{K}$ without triggering the supervisor's alarm.

We assume that the attackers have some knowledge about the plant. We will now describe why this assumption is made with several simple scenarios where the supervisor can detect an attack, stop the control loop, and raise the alarm. If the actuator attacker sends $\mathsf{i}_{\mathcal{P}}$ to the plant while the plant currently has no possible transition defined by $\mathsf{i}_{\mathcal{P}}$ as an input symbol, then the plant will be at a standstill. The supervisor will recognize this abnormality and raise the alarm. This would occur when $L_{\text{out}}(\mathcal{A}_a) \not\subseteq L_{\text{in}}(\mathcal{P})$.

Similarly, if the sensor attacker intercepts a message $\mathsf{o}_{\mathcal{P}}$ and has no valid transitions with $\mathsf{o}_{\mathcal{P}}$ as an input symbol in its current state then the supervisor will notice the delay and raise the alarm. In other words, if the supervisor is expecting a response from the plant and it does not receive one then it will raise the alarm and stop the control loop. This failed attack occurs if $L_{\text{out}}(\mathcal{P}) \not\subseteq L_{\text{in}}(\mathcal{A}_s)$. For the rest of the analysis, we will assume that the attackers have enough knowledge of the plant so that $L_{\text{out}}(\mathcal{A}_a) \subseteq L_{\text{in}}(\mathcal{P})$ and $L_{\text{out}}(\mathcal{P}) \subseteq L_{\text{in}}(\mathcal{A}_s)$.

For the sensor attacker $\mathcal{A}_s$, at some time, its input symbol $\mathsf{o}_{\mathcal{P}}$ may be disallowed by the supervisor in its current state, or its output symbol $\mathsf{o}_{\mathcal{S}}$ (as the supervisor's input symbol) may be disallowed in the supervisor's current state. Similarly, for the actuator attacker $\mathcal{A}_a$, at some time, its input symbol $\mathsf{o}_{\mathcal{S}}$ may be disallowed in its current state, or its output symbol $\mathsf{i}_{\mathcal{P}}$ (as the plant's input symbol) may be disallowed in the plant's current state. In all these cases, the attackers are viewed as failed since the supervisor or plant can detect the abnormality, immediately stop the supervisory control loop, and raise the alarm.

Given the discussion above, the supervisor $\mathcal{S}$ aims to restrict the executions of the plant $\mathcal{P}$ to ones that only yield

words in the desired language $\mathscr{K} \subseteq L(\mathcal{P})$. We denote by $L(\mathcal{P}|\mathcal{S}_c)$ the language of the plant and corrupted supervisor's loop composition. This is the set of possible executions of the plant in the supervisory control loop with the attackers. Formally, the control goal is given below.

Definition 7. *Let $\mathcal{P}$, $\mathcal{A}_s$, and $\mathcal{A}_a$ be observable FSTs modeling the plant, sensor attacker, and actuator attacker, respectively, and $\mathscr{K} \subseteq L(\mathcal{P})$ be the prefix-closed regular language capturing its desired executions. The supervisor $\mathcal{S}$ controls the plant $\mathcal{P}$ to the language $\mathscr{K}$ iff the set of plant executions under the supervisor's regulation satisfies*

$$L(\mathcal{P}|\mathcal{S}_c) = \mathscr{K}.$$

This ensures that 1) the plant execution is restricted by the desired set $\mathscr{K}$ under the attacks and 2) each word in $\mathscr{K}$ corresponds to a possible plant execution under certain behavior of the supervisor and attackers.

IV. Advantages of FSTs to Model Attacks

This section introduces the advantages of using FSTs to model attack behaviors in supervisory control, following the discussions in Section III-C. Since the attack behaviors are mathematical by regular relations, we refer to the class of attacks modeled by FSTs as *regular-rewriting attacks*. The regular relations can mathematically capture the attackers' nondeterministic behaviors. These include previously studied history-independent attacks (e.g., insert, delete, and replace) [18, 50, 39, 19] and more sophisticated history-dependent attacks such as finite-memory replay attacks [21, 22], which can potentially bypass existing intrusion detectors by recording and replaying previous system executions. The work in [20] also implements history-dependent attacks. However, they use sensor attackers modeled by a function: the attack is determined by the plant's transition history. Comparatively, our attacker model is described by an FST allowing relatively straightforward analysis and intuitive visual representations of the attackers.

A. Modeling common attacks by FSTs

FSTs can model a diverse range of nondeterministic attacks in supervisory control. FST models can be derived from the regular relations that the attacks obey, e.g., by analyzing the security services and (possible) attack surfaces from the system's architecture and deployment. In return, the resilience analysis given later in this paper can guide improving the security services in the deployed system. Below, we give a few examples of modeling common attacks with FSTs.

Example 3. *Let $\mathbf{I}' \subseteq \mathbf{I}$. The projection attack*

$$\mathrm{Project}_{\mathbf{I}'}(\mathsf{i}) = \begin{cases} \mathsf{i}, & \text{if} \quad \mathsf{i} \in \mathbf{I}' \\ \varepsilon, & \text{otherwise,} \end{cases} \tag{1}$$

captures the attack that results in removing all symbols that belong to $\mathbf{I} \setminus \mathbf{I}'$. On the other hand, the nondeterministic deletion attack is defined as

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

It 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 [51] over the communication network. Finally, the nondeterministic injection attack is defined as

$$\mathrm{Inject}_{\mathbf{I}'}(\varepsilon) = \mathsf{i} \text{ where } \mathsf{i} \in \mathbf{I}'. \tag{3}$$

In it, 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 Fig. 6a, Fig. 6b and Fig. 6c, respectively.

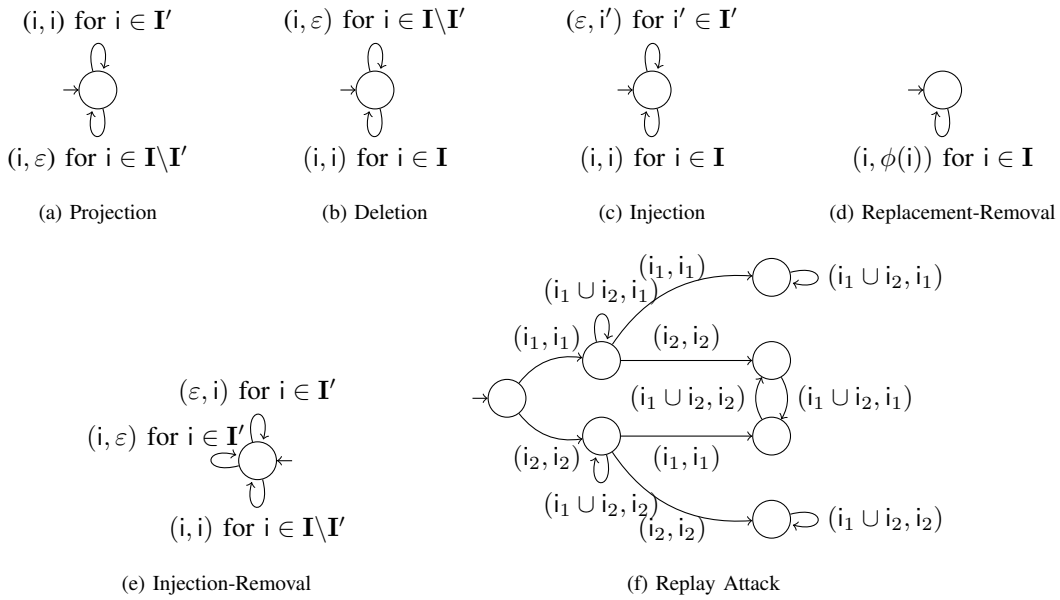
Example 4. *A replacement-removal attack defined by the rule $\phi : \mathbf{I} \to 2^{\mathbf{I} \cup \varepsilon}$ is represented by an FST as shown in Fig. 6d.*

Example 5. *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 Fig. 6e.*

Beyond the simple attackers in Example 3-5, FSTs can also model complex history-dependent attacks, motivated by cyber/network attacks implemented by embedded programs (e.g., malware). Since the attackers modeled by FSTs have (internal) states, they can perform different attack actions depending on their current state, which is affected by their previous attack actions. Take the replay attack as an example, which is a common strategy for launching cyber attacks [52, 53]. It works by first recording a fragment of symbols and then replaying it repeatedly to trap the system. Implementing this attack requires the attacker to know when to continue recording or start replaying.

Example 6. *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. This relation is regular, so can be modeled by FSTs. For example, a replay attack recording a prefix of length up to $N = 2$ for any word $\mathbf{I} = \{\mathsf{i}_1, \mathsf{i}_2\}$ can be modeled by an FST as shown in Fig. 6f. Note that the FST can be viewed as the parallel composition of two replay attacks recording prefixes of length 1 and 2, respectively.*

In [20], the attackers are also modeled with states. However, these states are shared with the plant (referred to as the environment). For different plant models, this approach requires building a new combined model to capture the same replay attack. On the other hand, our approach allows modeling a replay attack separately and then composing it with the plant model to study its impact. Our compositional approach can help the users to 1) model each component separately, 2) study the impact of different attacks on the same plant, 3) study the impact of multiple combined attacks.



(a) Projection (b) Deletion (c) Injection (d) Replacement-Removal

(e) Injection-Removal (f) Replay Attack

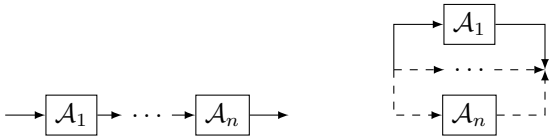
FIGURE 6: FST models for common attacks.

B. Composition of multiple attacks

Multiple attacks modeled by FSTs can be combined into an overall FST model. This justifies our modeling from the general system architecture in Fig. 1 to the control diagram in Fig. 2. Specifically, in Fig. 1, there may be coordinated attacks from multiple deployed attack vectors with different "point-of-entries" from the sensors, actuators, and communication networks. For example, on the sensor side, there may be false data injection via sensor spoofing [54, 1] together with denial-of-service (DoS) attacks on network transmissions. The overall effect of those attacks is equivalent to their serial composition as illustrated in Fig. 7a, which is represented by a single FST $\mathcal{A}_s$ in Fig. 2.

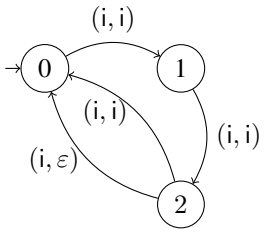
Similarly, if there is one of several possible attacks that may be deployed at the same "point-of-entry" as studied in [18], the overall effect, and the corresponding FST model, is equivalent to the parallel composition of the two FSTs capturing the basic attacks, as shown in Fig. 7b. Algorithmically, the parallel composition is computed as follows.

1) Compute the union of the states, final states, and transitions.
2) Add a new starting state with transitions ε/ε to the initial states s_{init} and $\mathsf{s}'_{\text{init}}$.



(a) When n attacks happen simultaneously (e.g., false data injection attack on sensors $\mathcal{A}_1$, denial-of-service attack on network $\mathcal{A}_2$, ...), the overall effect can be captured by the serial composition.

(b) When (only) one of n possible attacks may occur at a "point-of-entry" (i.e., attack point), the overall effect can be captured by the parallel composition of the attack models.



(c) Modeling frequency constraints; e.g., the operational constraint that an FST-based projection attacker $\mathcal{A}$ happens once every three steps can be captured by augmenting the FST model

FIGURE 7: Modeling attack composition and attack constraints.

C. Imposing constraints on attacks

FSTs also facilitate imposing (operational) constraints and transform history-independent attacks (e.g., injection attacks from (3)) to history-dependent attacks. For example, in some networked control systems, cryptographic primitives (e.g., Message Authentication Codes) can only be intermittently used due to resource constraints, and thus can only intermittently prevent injection attacks [16, 17, 55]. FSTs can facilitate modeling such constraints as frequency constraints on the attacker as studied in [50].

To illustrate this, we take the simple FST in Figure 6a and show how to impose a frequency constraint.

Example 7. *The FST in Figure 6a can remove the input symbol* i *non-deterministically. The removal can happen for all symbols in the worst case. Suppose we want to constrain the frequency to at most once every three symbols. Then, we can model it by the FST model in Figure 7c which is modified from the former FST.*

V. Attack-Resilient Supervisor Design

This section presents theories and algorithms to synthesize resilient supervisors for the formulation in Definition 7. Since the plants are modeled by FSTs and can generate different input and output symbols, we consider supervisors that can monitor a pair of different input and output symbols each time, as discussed in Section III-A. Synthesizing the resilient supervisor that satisfies Definition 7 is not always feasible. Thus, we consider a related problem of designing a candidate resilient supervisor that can restrict $L(\mathcal{P}|\mathcal{S}_c)$ where $\mathcal{S}_c = \mathcal{A}_a \circ \mathcal{S} \circ \mathcal{A}_s$ to the minimal feasible super-language of $\mathscr{K}$, i.e., finding a $\mathcal{S}$ such that $L(\mathcal{P}|\mathcal{S}_c)$ is the smallest possible language that contains $\mathscr{K}$.

Definition 8. *The FST* $\mathcal{S}$ *is a candidate resilient supervisor for a desired language* $\mathscr{K}$ *iff*

1) $\mathscr{K} \subseteq L(\mathcal{P}|\mathcal{A}_a \circ \mathcal{S} \circ \mathcal{A}_s)$*, and*
2) *for any* $\mathcal{S}'$ *that satisfies* $\mathscr{K} \subseteq L(\mathcal{P}|\mathcal{A}_a \circ \mathcal{S}' \circ \mathcal{A}_s)$*, it holds that* $L(\mathcal{P}|\mathcal{A}_a \circ \mathcal{S} \circ \mathcal{A}_s) \subseteq L(\mathcal{P}|\mathcal{A}_a \circ \mathcal{S}' \circ \mathcal{A}_s)$.

If $L(\mathcal{P}|\mathcal{A}_a \circ \mathcal{S} \circ \mathcal{A}_s)$ is equal to $\mathscr{K}$, then the candidate resilient supervisor is the resilient supervisor that satisfies Definition 7. In the following, we study attacks on sensors in Section V-A, on actuators in Section V-B, and on both sensors and actuators in Section V-C.

A. Design resilient supervisors to sensor attacks

To synthesize the candidate resilient supervisor in Definition 8 with only the sensor attacker $\mathcal{A}_s$, we use the FST inverse $\mathcal{A}_s^{-1}$ as the countermeasure.

Definition 9. *Given an FST,* $\mathcal{A} = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{I}, \mathbf{O}, \mathsf{Trans}, \mathsf{S}_{\text{final}})$*, the FST inverse is also an FST denoted by* $\mathcal{A}^{-1} = (\mathsf{S}, \mathsf{s}_{\text{init}}, \mathbf{O}, \mathbf{I}, \mathsf{Trans}^{-1}, \mathsf{S}_{\text{final}})$*, where the transition* $(\mathsf{s}, \mathsf{o}, \mathsf{i}, \mathsf{s}') \in \mathsf{Trans}^{-1}$ *if and only if* $(\mathsf{s}, \mathsf{i}, \mathsf{o}, \mathsf{s}') \in \mathsf{Trans}$.

The inverse of an FST is derived by flipping the input/output symbols on each transition, as given in definition 9. By inverting an FST, the regular relation it defines is also reverted, as stated by the following lemma [47], and definition 9 provides an FST realization for the inverse of regular relations.

Lemma 3. *For an FST* $\mathcal{A}$*, it holds that* $\mathcal{R}_{\mathcal{A}^{-1}} = \mathcal{R}_{\mathcal{A}}^{-1}$*, where* $\mathcal{R}_{\mathcal{A}}$ *denotes the regular relation defined by* $\mathcal{A}$.

The composition of a relation and its inverse includes the identity map. Thus, we have the following lemma on the composition of an FST and its inverse.

Lemma 4. *For any FST* $\mathcal{A}$*, it holds that*

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

Equivalently, it holds that

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

Here, $L_{\text{in}}(\mathcal{A})$ *and* $L_{\text{out}}(\mathcal{A})$ *are the input and output languages of* $\mathcal{A}$*, and* $\mathcal{M}_L$ *is the automaton realization (thus also an FST) of the regular language* L.

Recall Theorem 1. For the desired language $\mathscr{K} \subseteq L(\mathcal{P})$ of the plant $\mathcal{P}$, the FST $\mathcal{S}$ can supervise the plant's language in the closed loop without attacks, i.e., let $L(\mathcal{P}|\mathcal{S}) = \mathscr{K}$, if $\mathcal{S} = \mathcal{M}_{\mathscr{K}}^{-1}$ where $\mathcal{M}_{\mathscr{K}}$ is a deterministic FST realizing $\mathscr{K}$. The supervisor should be the inverse of $\mathcal{M}_{\mathscr{K}}$ since the plant and supervisor have flipped input and output. Now, after introducing the sensor attacker $\mathcal{A}_s$, we show that the composition of the inverse of $\mathcal{A}_s$ and $\mathcal{M}_{\mathscr{K}}^{-1}$ can potentially serve as a resilient supervisor, as stated below. Intuitively, in the supervisor $\mathcal{S} = \mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}}^{-1}$, the part $\mathcal{A}_s^{-1}$ partially reverts the attacks of $\mathcal{A}_s$ by finding out the possible true words before the attacks, and then the part $\mathcal{M}_{\mathscr{K}}^{-1}$ filters out undesirable words. Thus, the natural choice of the candidate supervisor is $(\mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_s)^{-1} = \mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}}^{-1}$. The theorem below formally explains this approach.

Theorem 2. *With only the sensor attacker* $\mathcal{A}_s$*, for any FST plant* $\mathcal{P}$ *and desired prefix-closed regular language* $\mathscr{K} \subseteq L(\mathcal{P})$*, the deterministic FST satisfying*

$$\mathcal{R}_{\mathcal{S}_{\min}} = \mathcal{R}_{\mathcal{A}_s^{-1}\circ\mathcal{M}_{\mathscr{K}}^{-1}}$$

is a candidate resilient supervisor. It is a resilient supervisor iff

$$L(\mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_s \circ \mathcal{A}_s^{-1}) \cap L(\mathcal{P}) = \mathscr{K}.$$

Proof:
We defer the proof until Theorem 4. ■

B. Design resilient supervisors to actuator attacks

Similar to V-A, when there is only the actuator attacker $\mathcal{A}_a$, we use the FST inverse $\mathcal{A}_a^{-1}$ as the countermeasure. The theorem below formally explains this approach.

Theorem 3. *With only the actuator attacker* $\mathcal{A}_a$*, for any plant* $\mathcal{P}$ *and desired prefix-closed regular language* $\mathscr{K} \subseteq L(\mathcal{P})$*, the deterministic FST satisfying*

$$\mathcal{R}_{\mathcal{S}_{\min}} = \mathcal{R}_{\mathcal{M}_{\mathscr{K}}^{-1}\circ\mathcal{A}_a^{-1}}$$

is a candidate resilient supervisor. It is a resilient supervisor iff

$$L(\mathcal{A}_a^{-1} \circ \mathcal{A}_a \circ \mathcal{M}_{\mathscr{K}}) \cap L(\mathcal{P}) = \mathscr{K}.$$

Proof:
We defer the proof until Theorem 4. ■

Intuitively, the supervisor is corrupted by the composition with $\mathcal{A}_a$, thus the natural choice of the candidate supervisor is $\mathcal{M}_{\mathscr{K}}^{-1} \circ \mathcal{A}_a^{-1}$, according to Theorem 1.

C. Design resilient supervisors to both sensor and actuator attacks

When both the sensor and actuator attackers $\mathcal{A}_s$ and $\mathcal{A}_a$ exist, we combine the approaches from Section V-A and V-B and use the FST inverse $\mathcal{A}_s^{-1}$ and $\mathcal{A}_a^{-1}$ as the countermeasure. Again, we can think of the composition $\mathcal{A}_a \circ \mathcal{P} \circ \mathcal{A}_s$ as the corrupted plant, and $L(\mathcal{A}_a \circ \mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_s)$, thus the natural choice of the candidate supervisor is $(\mathcal{A}_a \circ \mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_s)^{-1} = \mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}}^{-1} \circ \mathcal{A}_a^{-1}$, according to Theorem 1. The theorem below formally explains this approach.

Theorem 4. *For any plant $\mathcal{P}$ and desired prefix-closed regular language $\mathscr{K} \subseteq L(\mathcal{P})$, the deterministic FST satisfying*

$$\mathcal{R}_{\mathcal{S}} = \mathcal{R}_{\mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}}^{-1} \circ \mathcal{A}_a^{-1}} \tag{4}$$

is a candidate resilient supervisor. It is a resilient supervisor iff

$$L(\mathcal{A}_a^{-1} \circ \mathcal{A}_a \circ \mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_s \circ \mathcal{A}_s^{-1}) \cap L(\mathcal{P}) = \mathscr{K}. \tag{5}$$

Proof:
We start with proving the deterministic FST $\mathcal{S}_{\min}$ satisfying (4) is a candidate resilient supervisor. The existence of $\mathcal{S}_{\min}$ results from Lemma 1 and that $\mathcal{R}_{\mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}}^{-1} \circ \mathcal{A}_s^{-1}}$ is a regular relation.

First, we prove the supervisor $\mathcal{S}_{\min}$ satisfies Condition 2) of Definition 8. Consider another supervisor $\mathcal{S}$ that satisfies Condition 1) of Definition 8. Thus, it should allow any word in $\mathscr{K}$ for the plant under the attackers. Namely, any pair of input and output words $I_{\mathcal{P}} = \mathsf{i}_1 \dots \mathsf{i}_n$ and $O_{\mathcal{P}} = \mathsf{o}_1 \dots \mathsf{o}_n$ satisfying $(\mathsf{i}_1, \mathsf{o}_1) \dots (\mathsf{i}_n, \mathsf{o}_n) \in \mathscr{K}$, i.e.,

$$(I_{\mathcal{P}}, O_{\mathcal{P}}) \in \mathcal{R}_{\mathcal{M}_{\mathscr{K}}}, \tag{6}$$

should be allowed for the plant. Here, $\mathsf{i}_1, \mathsf{o}_1, \dots \mathsf{i}_n, \mathsf{o}_n$ may be the empty symbol ε. Similarly, the pair of input and output words $O_{\mathcal{S}}$ and $I_{\mathcal{P}}$, should be allowed by the attacker $\mathcal{A}_a$ if

$$(O_{\mathcal{S}}, I_{\mathcal{P}}) \in \mathcal{R}_{\mathcal{A}_a}. \tag{7}$$

And the pair of input and output words $O_{\mathcal{P}}$ and $I_{\mathcal{S}}$, should be allowed by the attacker $\mathcal{A}_a$ if

$$(O_{\mathcal{P}}, I_{\mathcal{S}}) \in \mathcal{R}_{\mathcal{A}_s}. \tag{8}$$

Since the plant $\mathcal{P}$ and attackers $\mathcal{A}_s$ and $\mathcal{A}_a$ are deterministic, their executions corresponding determined by $I_{\mathcal{P}}, O_{\mathcal{P}}, I_{\mathcal{S}}, O_{\mathcal{S}}$. Combining (6)(7)(8) gives

$$(O_{\mathcal{S}}, I_{\mathcal{S}}) \in \mathcal{R}_{\mathcal{A}_a} \circ \mathcal{R}_{\mathcal{M}_{\mathscr{K}}} \circ \mathcal{R}_{\mathcal{A}_s}.$$

That is, by Lemma 2 and 3,

$$(I_{\mathcal{S}}, O_{\mathcal{S}}) \in \mathcal{R}_{\mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}}^{-1} \circ \mathcal{A}_s^{-1}}. \tag{9}$$

By construction, any such pair of $I_{\mathcal{S}}$ and $O_{\mathcal{S}}$ should be accepted by $\mathcal{S}$, thus

$$\mathcal{R}_{\mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}}^{-1} \circ \mathcal{A}_s^{-1}} = \mathcal{R}_{\mathcal{S}_{\min}} \subseteq \mathcal{R}_{\mathcal{S}}.$$

Therefore, $\mathcal{S}_{\min}$ is the "minimal" supervisor.

Second, we prove the supervisor $\mathcal{S}_{\min}$ satisfy Condition 1) of Definition 8. Again, consider a pair of words $I_{\mathcal{P}}$ and $O_{\mathcal{P}}$ satisfying (6) generated from the plant $\mathcal{P}$. Let $I_{\mathcal{S}}$ and $O_{\mathcal{S}}$ be the corresponding corrupted words sent to the supervisor. Then, the pair $(O_{\mathcal{S}}, I_{\mathcal{P}})$ should satisfies (7) and the pair $(O_{\mathcal{P}}, I_{\mathcal{S}})$ should satisfies (8). Since the supervisor $\mathcal{S}_{\min}$ and attackers $\mathcal{A}_s$ and $\mathcal{A}_a$ are deterministic, their executions corresponding determined by $I_{\mathcal{P}}, O_{\mathcal{P}}, I_{\mathcal{S}}, O_{\mathcal{S}}$. Combining the above conditions, we have that any such pair of $I_{\mathcal{S}}$ and $O_{\mathcal{S}}$ should satisfy (9), and thus allowed by the supervisor $\mathcal{S}_{\min}$.

Finally, we show that the left-hand side of (5) gives the language $L(\mathcal{P} \mid \mathcal{A}_a \circ \mathcal{S} \circ \mathcal{A}_s)$ of the plant in the closed loop with supervisor and attackers. Following the formulation in Section III-C, the set of plant words (i.e., sequences of input/output symbol pairs of the plant) allowed by the supervisor $\mathcal{S}_{\min}$ and attackers $\mathcal{A}_s$ and $\mathcal{A}_a$ is

$$L((\mathcal{A}_s \circ \mathcal{S}_{\min} \circ \mathcal{A}_a)^{-1}) = L(\mathcal{A}_a^{-1} \circ \mathcal{A}_a \circ \mathcal{M}_{\mathscr{K}} \circ \mathcal{A}_s \circ \mathcal{A}_s^{-1}). \tag{10}$$

This result can be derived by viewing $\mathcal{A}_s \circ \mathcal{S}_{\min} \circ \mathcal{A}_a$ as a corrupted supervisor and applying Theorem 1. The inverse is because the plant's input is the supervisor's output and the plant's output is the supervisor's input. Therefore, the plant language $L(\mathcal{P} \mid \mathcal{A}_a \circ \mathcal{S} \circ \mathcal{A}_s)$ is the intersection of (10) with $L(\mathcal{S})$. When (5) holds, we have $L(\mathcal{P} \mid \mathcal{A}_a \circ \mathcal{S} \circ \mathcal{A}_s) = \mathscr{K}$, thus $\mathcal{S}_{\min}$ is a resilient supervisor. ■

Theorem 4 gives Algorithm 1 for resilient supervisors to the actuator and sensor attacks. This algorithm computes a minimal superlanguage of $\mathscr{K}$. If the aforementioned super language equates to $\mathscr{K}$, then the designed supervisor is resilient. The algorithm only involves FST inverse and composition and thus has polynomial complexity in time. Our previous work [32] develops a method to compute the largest subset of $\mathscr{K}$ that the plant can follow with supervisory control under the influence of attackers. The latter is known as a maximal sublanguage. However, this approach is much more computationally expensive than the former since it relies on fixed point operations. Below is an illustrative example. A more complex example is given in Section VI.

Algorithm 1 Resilient supervisor for both sensor and actuator attacks

Require: Desired language $\mathscr{K} \subseteq L(\mathcal{P})$, and attackers $\mathcal{A}_s$ and $\mathcal{A}_a$.
1: Build an FST $\mathcal{M}_{\mathscr{K}}$ realizing $\mathscr{K}$.
2: Compute inverse $\mathcal{A}_a^{-1}$, $\mathcal{M}_{\mathscr{K}}^{-1}$ and $\mathcal{A}_s^{-1}$.
3: Compute composition $\mathcal{S} = \mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}}^{-1} \circ \mathcal{A}_a^{-1}$.
4: **if** $L((\mathcal{A}_s \circ \mathcal{S} \circ \mathcal{A}_a)^{-1}) \cap L(\mathcal{P}) = \mathscr{K}$ **then**
5: **return** $\mathcal{S}$
6: **else**
7: **return** Infeasible
8: **end if**

Example 8. *As shown in Fig. 8, consider*

- *set of input and output symbols* $\mathbf{I} = \{\mathsf{i}_1, \mathsf{i}_2, \mathsf{i}_3, \mathsf{i}_4, \mathsf{i}_5\}$ *and* $\mathbf{O} = \{\mathsf{o}_1, \mathsf{o}_2, \mathsf{o}_3, \mathsf{o}_4, \mathsf{o}_5\}$,

- *a plant $\mathcal{P}$ with language $\overline{\{t_1t_2, t_3t_4, t_5\}^*}$ as shown and defined in Fig. 8a,*
- *a desired language $\mathscr{K} = \overline{\{t_1t_2\}^*}$ realized by FST $\mathcal{M}_{\mathscr{K}}$ in Fig. 8b,*
- *two actuator attackers: $\mathcal{A}_{a1}$ from Fig. 8c rewrites o_1 to o_3 and vice versa while $\mathcal{A}_{a2}$ from Fig. 8d rewrites o_3 to o_1 only,*
- *two sensor attackers: $\mathcal{A}_{s1}$ from Fig. 8e rewrites i_5 to i_1 and vice versa while $\mathcal{A}_{s2}$ from Fig. 8f rewrites i_1 to i_5 only.*

In this example, we have shown two different types of attacks and supervisors for the same plant FST and desired language, $\mathscr{K}$. The supervisor shown in Fig. 8g was designed to counter $\mathcal{A}_{a1}$ and $\mathcal{A}_{s1}$. Similarly, the supervisor in Fig. 8h was designed to counter $\mathcal{A}_{a2}$ and $\mathcal{A}_{s2}$.

Both supervisors were designed using Algorithm 3. The supervisor language $L(\mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}}^{-1} \circ \mathcal{A}_s^{-1})$ was computed to determine if the plant can be restricted to $\mathscr{K}$. The choice of $\mathcal{A}_{a2}$ and $\mathcal{A}_{s2}$ does not allow for a feasible resilient supervisor design because the $\mathcal{A}_{s2}$ forces i_5 to be the only symbol sent to the plant. The supervisor will then receive o_5 and raise the alarm. Thus, no transitions will ever occur.

In practice, it is unrealistic to assume a design engineer has complete knowledge of the attackers on a CPS. However, this algorithm is still very useful because it can quickly test for resilient supervisor existence based on a wide variety of nondeterministic attacks. In practice, our proposed FST attack model would act as a conservative overestimate of the true attacker.

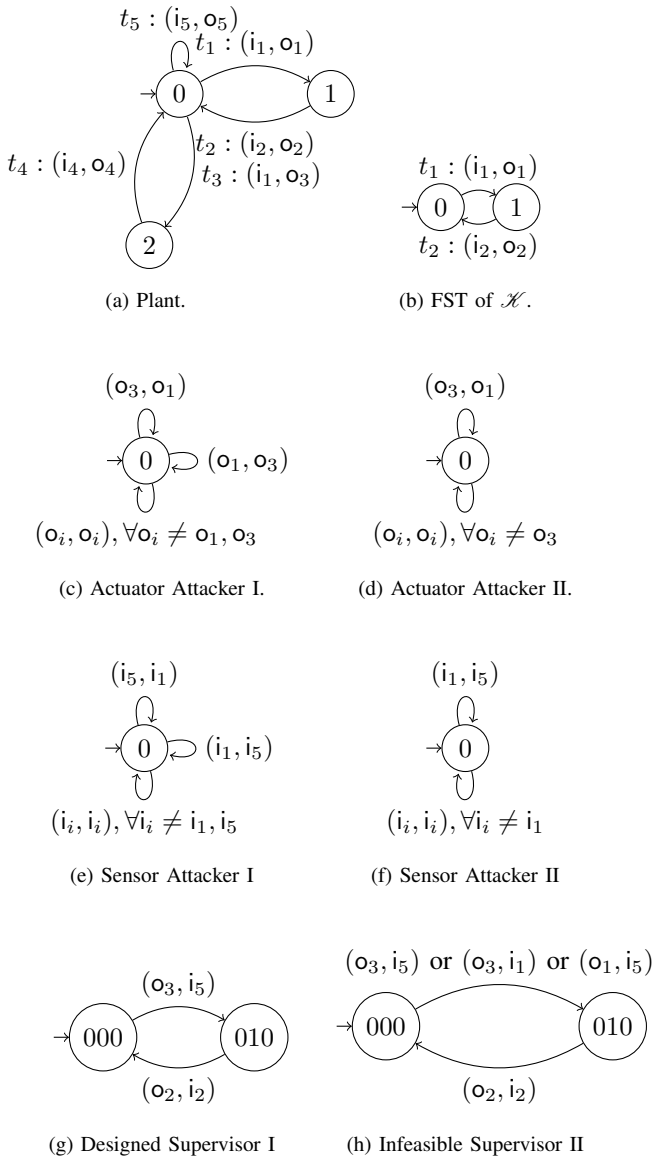
D. Relationship to Previous Work

Due to its generality, the presented FST-based framework described in Section III and IV can emulate several previously-studied setups of supervisory control. For example, we can emulate the classical supervisory control [49] with uncontrollable symbols $\mathbf{I}_{uc}$ in our framework by adopting the following setups in Fig. 2 by letting $\mathcal{P}$ be the plant (an automaton) of interest and the actuator attacker be $\mathcal{A}_a^{(s)} = \text{Inject}_{\mathbf{I}_{uc}}$ and $\text{Inject}_{\mathbf{I}_{uc}} \circ \mathcal{P}$ be the serial composition of the injection attack from (3) and the plant. This attacker injects uncontrollable symbols in $\mathbf{I}_{uc}$ whenever they are acceptable by the plant. The supervisory control under sensor/actuator enablement and disablement attacks [19] can be emulated in our framework by letting the sensor/actuator attacks be $\mathcal{A}_a^{(ed)} \circ \mathcal{A}_a^{(s)}$, where $\mathcal{A}_a^{(s)}$ is defined above and $\mathcal{A}_a^{(ed)}$ is the injection-removal attack on a set of vulnerable control symbols from Example 5. Finally, the supervisory control under replacement-removal sensor attacks [18] can be emulated in our framework by letting the sensor attacks be $\mathcal{A}_s^{(rr)} \circ \mathcal{A}_s^{(s)}$, where $\mathcal{A}_s^{(s)}$ is defined above and $\mathcal{A}_s^{(rr)}$ is the replacement-removal attack from Example 4.

VI. Case Studies: ARSC Tool and Synthesis Scalability

Based on the proposed algorithms, we developed an open-source tool *ARSC* [56] based on the OpenFst library [29] and illustrated 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 the effectiveness of our approach, we consider a scheduling problem from [49], where n players independently require 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 rotating 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



(a) Plant. (b) FST of $\mathscr{K}$.

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

(e) Sensor Attacker I (f) Sensor Attacker II

(g) Designed Supervisor I (h) Infeasible Supervisor II

FIGURE 8: Example supervisors for both sensor and actuator attacks.

TABLE 1: Execution time and memory usage of the supervisory synthesizing algorithm for different values of n and m.

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

modeled as shown in Fig. 9. From Theorem 4, the language $\mathscr{K}$ is controllable and the attacks can be countered by the supervisor constructed by Algorithm 1. The supervisor for the case $n = 2, m = 2$ is displayed in Fig. 9e.

The complexity of the supervisor synthesis algorithms is determined by 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 [29]. 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 $(\mathcal{A}_s^{-1} \circ \mathcal{M}_{\mathscr{K}}^{-1}) \circ \mathcal{A}_a^{-1}$ 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 1 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.

VII. Conclusions

We studied the supervisory control of discrete-event systems under sensor and actuator attacks. Both the system and the attacks are mathematically modeled by finite-state transducers (FST). We discussed the advantage of using FSTs to model attacks in capturing history dependency, composing multiple attacks, and imposing constraints on attacks. Then we proposed theorems and algorithms to design resilient supervisors for given attacks in three cases: only sensor attacks, only actuator attacks, and both sensor and actuator attacks. Finally, we implemented the algorithms with computer programs and demonstrated their applicability in case studies. Our work extended previous studies on attack-resilient supervisory control and provided a new framework based on FSTs.

REFERENCES

[1] D. Shepard, J. Bhatti, and T. Humphreys, "Drone hack," *GPS World*, vol. 23, no. 8, pp. 30–33, 2012.

[2] 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*, vol. 37, no. 2, pp. 66–81, April 2017.

[3] C. G. Cassandras, "Smart Cities as Cyber-Physical Social Systems," *Engineering*, vol. 2, no. 2, pp. 156–158, 2016.

[4] Z. Jiang, M. Pajic, and R. Mangharam, "Cyber-Physical Modeling of Implantable Cardiac Medical Devices," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 122–137, Jan 2012.

[5] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM, 2015, pp. 239–248.

[6] Z. Jakovljevic, V. Lesi, S. Mitrovic, and M. Pajic, "Distributing sequential control for manufacturing automation systems," *IEEE Transactions on Control Systems Technology*, 2019.

[7] V. Lesi, Z. Jakovljevic, and M. Pajic, "Security analysis for distributed IoT-based industrial automation," *arXiv preprint arXiv:2006.00044*, 2020.

[8] L. Lin, Y. Zhu, and R. Su, "Towards bounded synthesis of resilient supervisors," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 7659–7664.

[9] P. M. Lima, M. V. S. Alves, L. K. Carvalho, and M. V. Moreira, "Security against communication network attacks of cyber-physical systems," *Journal of Control, Automation and Electrical Systems*, vol. 30, no. 1, pp. 125–135, 2019.

[10] R. Meira-Góes, S. Lafortune, and H. Marchand, "Synthesis of supervisors robust against sensor deception attacks," *IEEE Transactions on Automatic Control*, 2021.

[11] 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.

[12] J. S. Warner and R. G. Johnston, "A simple demonstration that the global positioning system (GPS) is vulnerable to spoofing," *Journal of Security Administration*, vol. 25, no. 2, pp. 19–27, 2002.

[13] 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*. Springer, 2013, pp. 55–72.

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

[15] 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)*, 2012, pp. 55–64.

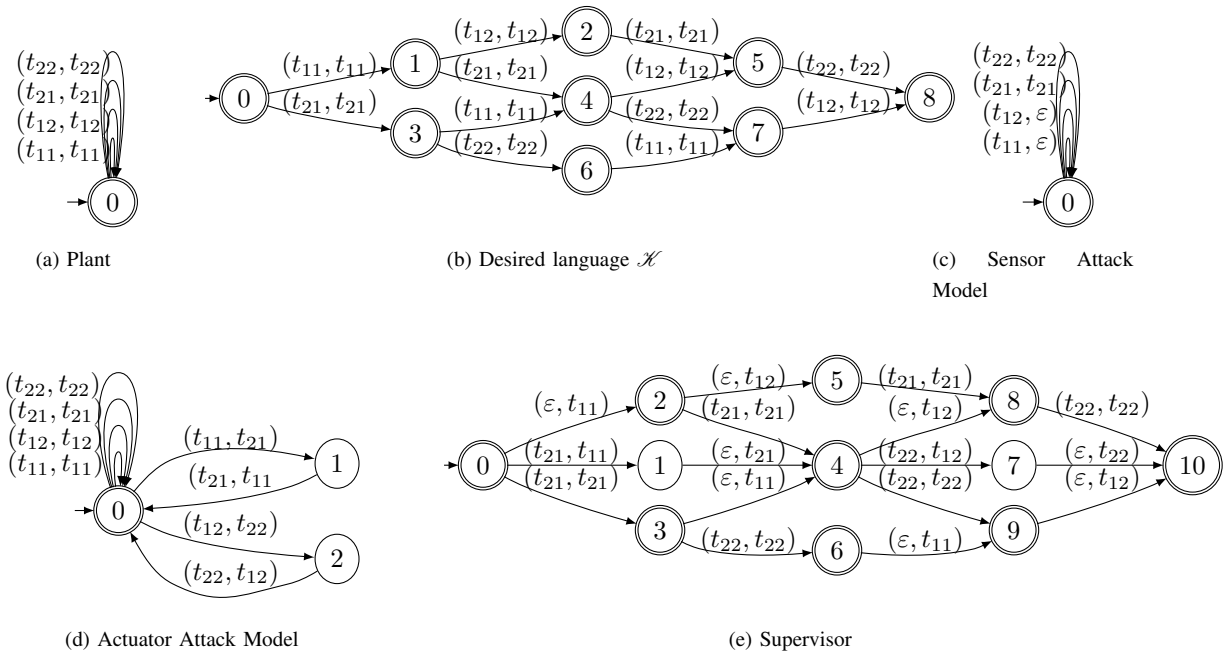


FIGURE 9: Example supervisors resilient to sensor and actuator attacks

[16] I. Jovanov and M. Pajic, "Relaxing integrity requirements for attack-resilient cyber-physical systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 12, pp. 4843–4858, 2019.

[17] V. Lesi, I. Jovanov, and M. Pajic, "Security-aware scheduling of embedded control tasks," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 188:1–188:21, Sep. 2017.

[18] M. Wakaiki, P. Tabuada, and J. P. Hespanha, "Supervisory control of discrete-event systems under attacks," *Dynamic Games and Applications*, pp. 1–19, 2017.

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

[20] R. Meira Góes, E. Kang, R. H. Kwong, and S. Lafortune, "Synthesis of sensor deception attacks at the supervisory layer of cyber-physical systems," *Automatica*, vol. 121, p. 109172.

[21] J. P. Farwell and R. Rohozinski, "Stuxnet and the future of cyber war," *Survival*, vol. 53, no. 1, pp. 23–40, 2011.

[22] T. M. Chen and S. Abu-Nimeh, "Lessons from stuxnet," *Computer*, vol. 44, no. 4, pp. 91–93, April 2011.

[23] J. Sakarovitch and R. Thomas, "Elements of Automata Theory," p. 784, 2003.

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

[25] M. Mohri, "Finite-State Transducers in Language and Speech Processing," *Computational Linguistics*, vol. 23, p. 42, 1997.

[26] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[27] M. Mohri, "Weighted Finite-State Transducer Algorithms. An Overview," in *Formal Languages and Applications*, J. Kacprzyk, C. Martín-Vide, V. Mitrana, and G. Păun, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, vol. 148, pp. 551–563.

[28] ——, "Weighted Automata Algorithms," in *Handbook of Weighted Automata*, M. Droste, W. Kuich, and H. Vogler, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 213–254.

[29] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFST: A General and Efficient Weighted Finite-State Transducer Library," in *Implementation and Application of Automata*, ser. Lecture Notes in Computer Science, J. Holub and J. Žďárek, Eds. Springer Berlin Heidelberg, 2007, pp. 11–23.

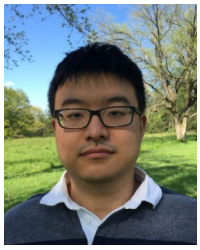
[30] A. K. Bozkurt, Y. Wang, and M. Pajic, "Secure planning against stealthy attacks via model-free reinforcement learning," *arXiv preprint arXiv:2011.01882*, 2020.

[31] V. Lesi, I. Jovanov, and M. Pajic, "Network scheduling for secure cyber-physical systems," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2017, pp. 45–55.

[32] Y. Wang and M. Pajic, "Attack-resilient supervisory control with intermittently secure communication," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 2015–2020.

[33] ——, "Supervisory control of discrete event systems in the presence of sensor and actuator attacks," in *2019 IEEE 58th Conference on Decision and Control (CDC)*,

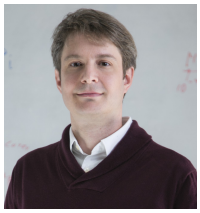
2019, pp. 5350–5355.
[34] S. Xu and R. Kumar, "Discrete event control under nondeterministic partial observation," in *2009 IEEE International Conference on Automation Science and Engineering*. IEEE, 2009, pp. 127–132.
[35] T. Ushio and S. Takai, "Nonblocking supervisory control of discrete event systems modeled by Mealy automata with nondeterministic output functions," *IEEE Transactions on Automatic Control*, vol. 61, no. 3, pp. 799–804, 2015.
[36] R. Su, "Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations," vol. 94, pp. 35–44.
[37] P. M. Lima, M. V. S. Alves, L. K. Carvalho, and M. V. Moreira, "Security of cyber-physical systems: Design of a security supervisor to thwart attacks," vol. 19, no. 3, pp. 2030–2041.
[38] D. You, S. Wang, and C. Seatzu, "A liveness-enforcing supervisor tolerant to sensor-reading modification attacks," vol. 52, no. 4, pp. 2398–2411.
[39] 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)*. Melbourne, Australia: IEEE, 2017, pp. 4224–4230.
[40] L. Lin, Y. Zhu, and R. Su, "Synthesis of covert actuator attackers for free," *Discrete Event Dynamic Systems*, vol. 30, no. 4, pp. 561–577.
[41] L. Lin and R. Su, "Synthesis of covert actuator and sensor attackers," *Automatica*, vol. 130, p. 109714.
[42] R. Tai, L. Lin, and R. Su, "Synthesis of optimal covert sensor–actuator attackers for discrete-event systems," *Automatica*, vol. 151, p. 110910.
[43] J. Yao, X. Yin, and S. Li, "Sensor deception attacks against initial-state privacy in supervisory control systems," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 4839–4845.
[44] Q. Zhang, C. Seatzu, Z. Li, and A. Giua, "Joint state estimation under attack of discrete event systems," *IEEE Access*, vol. 9, pp. 168 068–168 079.
[45] R. Tai, L. Lin, Y. Zhu, and R. Su, "Privacy-preserving co-synthesis against sensor–actuator eavesdropping intruder," *Automatica*, vol. 150, p. 110860.
[46] M. Sipser, "Introduction to the Theory of Computation," *ACM SIGACT News*, vol. 27, no. 1, pp. 27–29, 1996.
[47] M. Holcombe, *Algebraic Automata Theory*. Cambridge University Press, 1982.
[48] G. H. Mealy, "A method for synthesizing sequential circuits," *The Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, 1955.
[49] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York, NY: Springer, 2008.
[50] R. Su, "Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations," *Automatica*, vol. 94, pp. 35–44, 2018.
[51] A. D. Wood and J. A. Stankovic, "Denial of service in sensor networks," *computer*, vol. 35, no. 10, pp. 54–62, 2002.
[52] F. Miao, M. Pajic, and G. Pappas, "Stochastic game approach for replay attack detection," in *IEEE 52nd Annual Conference on Decision and Control (CDC)*, Dec 2013, pp. 1854–1859.
[53] 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*, vol. 100, no. 1, pp. 195–209, 2012.
[54] 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*, ser. CCS, 2011, pp. 75–86.
[55] V. Lesi, I. Jovanov, and M. Pajic, "Integrating security in resource-constrained cyber-physical systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 4, no. 3, May 2020.
[56] "ARSC – Tool for Synthesis of Attack-Resilient Supervisory Controllers," https://github.com/alperkamil/arsc, accessed: 2022-01-01.



Yu Wang is an Assistant Professor at the Department of Mechanical and Aerospace Engineering at the University of Florida (UF) and the Group Lead of Autonomous and Connected Vehicles of the UF Transportation Institute. He was a Postdoctoral Researcher at the Department of Electrical and Computer Engineering at Duke University. He received his Ph.D. in Mechanical Engineering and M.S. in Statistics and Mathematics from the University of Illinois at Urbana-Champaign (UIUC). His research focuses on assured autonomy, cyber-physical systems, machine learning, and formal methods. This work on statistical verification of hyperproperties for cyber-physical systems was selected as one of the Best Paper Finalists of the ACM SIGBED International Conference on Embedded Software (EMSOFT) in 2019.

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. candidate 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.

Nathan Smith received his Bachelor of Science in Aerospace Engineering and Mathematics from the University of Florida in 2021. He is currently pursuing a Ph.D. in Aerospace Engineering under Dr. Yu Wang. He is from Jupiter, Florida. His research interests include applications of Game Theory and Reinforcement Learning to model attacks on Cyber-Physical systems.



Miroslav Pajic 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 Dickinson Family Associate Professor in Department of Electrical and Computer Engineering at Duke University. He also holds secondary appointments in the Computer Science, and Mechanical Engineering and Material Science Departments. His research interests focus on the design and analysis of high-assurance cyber-physical systems with varying levels of autonomy and human interaction, at the intersection of (more traditional) areas of learning and controls, AI, embedded systems, formal methods, and robotics.

Dr. Pajic received various awards including the ACM SIGBED Early-Career Award, IEEE TCCPS 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 seven Best Paper and Runner-up Awards at top cyber-physical systems venues.