






# Opacity Enforcement via Greedy Privately-and-Publicly Known Insertion Functions

Rongjian Liu , Jianquan Lu , Senior Member, IEEE, Yang Liu , Senior Member, IEEE, Xiang Yin , Member, IEEE, and Christoforos N. Hadjicostis , Fellow, IEEE

**Abstract**—In this article, we investigate the enforcement of (current-state) opacity, an important information-flow security property, via insertion functions. An insertion function is an obfuscation mechanism that inserts fictitious events to the outputs in order to confuse the outside observer (intruder) such that the secret of the system is not revealed. In some situations, the secret may be revealed when the insertion mechanism is (or becomes) publicly known. This leads to the problem of synthesizing private-and-public enforcing (PP-enforcing) insertion functions in the sense that opacity is still enforced even when the mechanism is discovered or published by the designer. Existing works that have investigated this synthesis problem are either only sound or have limited applicability as we show in this work. For this reason, and more importantly, to better solve the synthesis problem, a new approach is proposed upon an improved greedy criterion. We show that the proposed algorithm is both sound and complete, and can be used to completely solve the synthesis problem for the PP-enforcing insertion function. With slight modifications of our algorithm, infinite-step opacity and  $K$ -step opacity can also be enforced under publicly-known insertion mechanisms.

**Index Terms**—Discrete event systems (DES), insertion mechanism, opacity enforcement, publicly known obfuscation.

## I. INTRODUCTION

SECURITY is a major concern in network communication systems, and people put a lot of effort to protect confidential information. Various security notions related to information flow have been proposed in the literature: among others, anonymity, noninterference, nondeductibility, and opacity, see, for example, [1], [2], and [3]. We are interested in

Manuscript received 12 June 2023; accepted 18 August 2023. Date of publication 23 August 2023; date of current version 29 March 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 61973078, Grant 62233004, and Grant 62173308, in part by the Scientific and Technological Research Program of Chongqing Municipal Education Commission Grant KJZD-K202300501, in part by the Chongqing Normal University Foundation under Grant 22XLB012, and in part by the Natural Science Foundation of Zhejiang Province of China under Grant LR20F030001 and Grant LD19A010001. Recommended by Associate Editor J. Komenda. (Corresponding author: Jianquan Lu.)

Rongjian Liu is with the National Center for Applied Mathematics in Chongqing, Chongqing Normal University, Chongqing 401331, China, and also with the School of Cyber Science and Engineering, Southeast University, Nanjing 210096, China (e-mail: rongjliu@foxmail.com).

Jianquan Lu is with the School of Mathematics, Southeast University, Nanjing 210096, China (e-mail: jqluma@seu.edu.cn).

Yang Liu is with the Key Laboratory of Intelligent Education Technology and Application of Zhejiang Province, School of Mathematical Sciences, Zhejiang Normal University, Jinhua 321004, China (e-mail: liuyang@zjnu.edu.cn).

Xiang Yin is with the Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: yinxiang@sjtu.edu.cn).

Christoforos N. Hadjicostis is with the Department of Electrical and Computer Engineering, University of Cyprus, 1678 Nicosia, Cyprus (e-mail: chadjic@ucy.ac.cy).

Digital Object Identifier 10.1109/TAC.2023.3307791

opacity, which is a confidentiality property that aims to determine whether the secret information of the system can be inferred by an outside intruder with malicious intentions [4].

Discrete event systems (DES) are efficient and successful in providing formal treatments and analytical techniques for many man-made systems, such as computer systems, communication systems, database systems, and traffic systems [5]. Opacity in DES has been extensively studied in recent years, and various notions of opacity have been considered in the literature, such as current-state opacity [6], language-based opacity [7], initial-state opacity [8], initial-and-final-state opacity [9], and  $K$ -step opacity and infinite-step opacity [10], [11]. Different approaches to enforce opacity in DES have been developed in the literature, such as supervisory control theory [12], [13], [14], [15] or dynamic mask [16], [17]. For additional information about opacity in DES, the readers can refer to [18], [19], and [20], and survey [21], or book [4].

Another important mechanism for enforcing opacity is the use of an output obfuscation mechanism, which includes insertion mechanisms [22] and edit mechanisms [23]. An insertion function is a monitoring interface placed at the output of the system and does not interact with the system. Ji et al. [24] pointed out that, if the existence of the insertion function becomes known to the intruder (e.g., by performing reverse engineering) or becomes public (e.g., as in public-key cryptography), the secret may be revealed. In such situations, a publicly-known insertion mechanism, as illustrated in Fig. 1, needs to be investigated. Notice that the edit mechanism, which is a more general version of the insertion mechanism, has been investigated in a publicly known manner in [25]. However, due to the use of erasures in the edit mechanism, the synthesis of such obfuscation strategies rely on distinct methodologies that require exponential complexity in  $|X_{\mathcal{E}}|$ , where  $X_{\mathcal{E}}$  is the state set of the standard observer of the system. This should be compared with polynomial complexity (with respect to  $|X_{\mathcal{E}}|$ ) for the publicly-known insertion mechanism proposed in this article. In addition, the insertion mechanism is more convenient than the edit mechanism when there exists an intended or legitimate receiver at the outside for recovering purposes [22], [26]. For additional research about opacity enforcement under insertion/edit mechanisms, one can refer to [27], [28], and [29].

To characterize the requirement that the insertion function needs to maintain the secret even when the insertion function is publicly known, the notion of private-and-public enforceability (PP-enforceability) was proposed in [24] and [28]. An (only) sound algorithm was proposed in [28] for synthesizing private-and-public enforcing (PP-enforcing) insertion functions, and a “sound and complete” approach was attempted in [24] based on a greedy-maximal criterion (GM-criterion). The GM-criterion in [24], makes insertion decisions by comparing the lengths of strings that are to be inserted, but it has limitations as it is (only) valid for a class of systems (see Example 3). Therefore, the existing GM-criterion needs to be improved, and the question of whether or not opacity is PP-enforceable when it is privately enforceable needs further investigation.

In this article, the notion of PP-enforceability is first revisited for accuracy, and a sound and complete algorithm for synthesizing PP-enforcing insertion functions is proposed. In fact, we solve a partially-solved open problem (the synthesis of PP-enforcing insertion functions), in a complete manner.

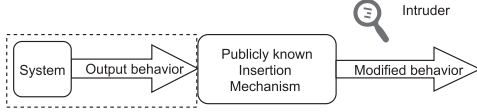


Fig. 1. Publicly known insertion mechanism.

## II. SYSTEM MODEL, OPACITY, INSERTION MECHANISM, AND ENFORCEABILITY

### A. System Model

Let  $E$  be a finite set of events and  $E^*$  be the set of all finite strings over  $E$  including the empty string  $\varepsilon$ . For a nonempty string  $s = e_1 e_2 \dots e_{|s|} \in E^*$ , where  $|s|$  is the length of string  $s$ , we use  $(s)_i$  to denote the  $i$ th event in  $s$ , i.e.,  $(s)_i = e_i, i = 1, 2, \dots, |s|$ . We also use  $|E|$  to denote the cardinality of  $E$ , i.e., the number of events in  $E$  is equal to  $|E|$ . For strings  $s$  and  $t$ , we use  $s \Subset t$  to denote that there exists an integer sequence  $1 \leq j_1 < j_2 < \dots < j_{|s|} \leq |t|$  such that  $(t)_{j_k} = (s)_k, k = 1, 2, \dots, |s|$ , i.e.,  $s$  is “embedded” in  $t$ . For two natural numbers  $k$  and  $l$  with  $k < l, [k, l] = \{k, k+1, \dots, l\}$ . We use  $t' \preceq t$  ( $t' \prec t$ ) to denote that string  $t'$  is a prefix (strict-prefix) of string  $t$ . For string  $t \preceq s$ , we use  $s/t$  to denote the post string of  $t$  in  $s$ , i.e.,  $s = t(s/t)$ . A language  $L \subseteq E^*$  is a subset of  $E^*$ . We use  $L|t$  to denote the set of strings  $s$  in  $L$  having prefix  $t$ , and define  $L|t = \{s \in L : t \preceq s\}$ . The prefix closure of language  $L$  is denoted by  $\bar{L}$ , where  $\bar{L} = \{t' \in E^* : \exists t \in L \text{ s.t. } t' \preceq t\}$ . Language  $L$  is prefix-closed if  $L = \bar{L}$ .

The system is modeled as a deterministic finite-state automaton<sup>1</sup>  $G = (X, E, f, X_0)$  with a finite set of states  $X$ , a finite set of events  $E$ , a partial deterministic state transition function  $f : X \times E \rightarrow X$ , and a set of initial states  $X_0 \subseteq X$ . In the context of opacity, the intruder does not have a priori knowledge on the precise initial state, and thus we include a set of initial states  $X_0$  in the definition of  $G$ . The transition function  $f$  is extended to domain  $X \times E^*$  in the standard manner [5]. For the sake of simplicity, transition  $f(x, e) = x'$ ,  $e \in E$  is written as  $x \xrightarrow{e} x'$ . The language generated by  $G$  from state  $x$  is denoted by  $\mathcal{L}(G, x) = \{s \in E^* : f(x, s)!\}$  (“!” means “is defined,” and we also use “ $f$ ” to denote “is not defined”). The language generated by  $G$  is denoted by  $\mathcal{L}(G, X_0) := \cup_{x_0 \in X_0} \mathcal{L}(G, x_0)$ . For simplicity, we will write  $\mathcal{L}(G)$  as shorthand for  $\mathcal{L}(G, X_0)$  when  $X_0$  is clearly defined. Automaton  $G = (X, E, f, \{x_0\})$  is also simply denoted as  $G = (X, E, f, x_0)$ , where  $x_0 \in X$  is the unique initial state of automaton  $G$ . The system  $G$  is generally partially observed. Thus, the event set  $E$  is partitioned as  $E = E_o \cup E_{uo}$ , where  $E_o$  is the set of observable events and  $E_{uo}$  is the set of unobservable events. The natural projection  $P : E^* \rightarrow E_o^*$  is defined as:  $P(\varepsilon) = \varepsilon; P(te) = P(t)$ , if  $e \in E_{uo}$  and  $P(te) = P(t)e$  if  $e \in E_o$ . For a language  $L$ ,  $P(L) = \{P(s) : s \in L\}$ .

### B. Opacity Notation

**Definition 1 (Current-State Opacity (CSO)):** Given system  $G = (X, E, f, X_0)$ , natural projection  $P$ , and the set of secret states  $X_S \subseteq X$ , the system is current-state opaque if  $\forall t \in L_S := \{t \in \mathcal{L}(G) : \exists x_0 \in X_0, f(x_0, t) \in X_S\}, \exists t' \in L_{NS} := \{t \in \mathcal{L}(G) : \exists x_0 \in X_0, f(x_0, t) \in (X \setminus X_S)\} \text{ s.t. } P(t) = P(t')$ .

CSO can be verified by checking the state information of the standard *observer automaton* of  $G$  built according to [5]. Hereafter, we will work on the observer automaton denoted by  $\mathcal{E} = (X_{\mathcal{E}}, E_o, f_{\mathcal{E}}, x_{\mathcal{E},0})$  directly, where  $\mathcal{E}$  is a deterministic automaton with the unique initial state  $x_{\mathcal{E},0} \in X_{\mathcal{E}}$ . In the remainder,  $\mathcal{E}$  is also called *current-state*

<sup>1</sup>Generally, the initial state is taken to be unique for a “deterministic” finite-state automaton. In our case, however the initial state might not be precisely known (due to a variety of reasons) but the automaton is still considered deterministic in the sense that its transition function is deterministic [4].

*estimator* or simply *estimator*. The safe language for CSO, denoted as  $L_{\text{safe}}$ , is defined as [22]

$$L_{\text{safe}} = P(\mathcal{L}(G)) \setminus ((P(\mathcal{L}(G)) \setminus P(L_{NS}))E_o^*).$$

The unsafe language for CSO is defined as  $L_{\text{unsafe}} = P(\mathcal{L}(G)) \setminus L_{\text{safe}}$ . Language  $L_{\text{safe}}$  is prefix-closed, and the intruder can never infer the secret by observing strings in  $L_{\text{safe}}$ . A string  $s$  is said to be safe if  $s \in L_{\text{safe}}$ , and unsafe otherwise. The continuations of a string  $t \in L_{\text{unsafe}}$  still belong to  $L_{\text{unsafe}}$ .

Moreover, we can build a deterministic automaton  $\mathcal{E}_d$ , called *desired estimator*, which is in the form  $\mathcal{E}_d = (X_{\mathcal{E}_d}, E_o, f_{\mathcal{E}_d}, x_{\mathcal{E}_d,0})$ , for the safe language  $L_{\text{safe}}$ .  $\mathcal{E}_d$  can be constructed from the estimator  $\mathcal{E}$  by deleting all the secret revealing states in  $\mathcal{E}$  and taking the accessible part from the initial state  $x_{\mathcal{E}_d,0} = x_{\mathcal{E},0}$ . For  $\mathcal{E}_d$ , we have  $\mathcal{L}(\mathcal{E}_d) = L_{\text{safe}}$  [22].

### C. Insertion Function and PP-Enforceability

An insertion function is defined as a potentially partial function  $f_I : E_o^* \times E_o \rightarrow E_o^+$ , where  $E_o^+ = E_o^* E_o$ , that outputs a string with inserted events based on the past observed events string and the current observed event. Given an observable string  $te_o \in P(\mathcal{L}(G))$ ,  $f_I(t, e_o) = t_I e_o$  when string  $t_I \in E_o^*$  is inserted before  $e_o$ . The *string-based* version of  $f_I$ , denoted by  $f_I^{\text{str}}$ , is defined as:  $f_I^{\text{str}}(\varepsilon) = \varepsilon$  and  $f_I^{\text{str}}(te_o) = f_I^{\text{str}}(t) f_I(t, e_o)$ , where  $t \in E_o^*, e_o \in E_o$ . Then, based on  $f_I$ , the modified language is  $f_I^{\text{str}}(P(\mathcal{L}(G))) = \{\tilde{t} \in E_o^* : \exists t \in P(\mathcal{L}(G)), f_I^{\text{str}}(t) = \tilde{t}\}$ . We assume that the events in the modified output string are observed one by one [24], i.e., from the intruder’s viewpoint, the modified output language of the system is prefix-closed. Note that the insertion functions  $f_I$  (and its string-based version  $f_I^{\text{str}}$ ) considered in this article are deterministic.

In this article, the synthesized insertion function is encoded as an input/output (I/O) automaton IA  $= (X_{ia}, E_o, E_o^+, f_{ia}, q_{ia}, x_{ia,0})$  [22], where  $X_{ia}$  is the set of states,  $E_o$  is the input events set, the output set is a set of strings in  $E_o^+ = E_o^* E_o$ ,  $f_{ia} : X_{ia} \times E_o \rightarrow X_{ia}$  is the transition function of IA, the output function  $q_{ia}$  is defined as:  $q_{ia}(x, e_o) = q_{ia}(f_{ia}(x_{ia,0}, t), e_o) = t_I e_o$ , where  $x = f_{ia}(x_{ia,0}, t)$ ,  $f_I(t, e_o) = t_I e_o$ , and  $x_{ia,0}$  is the unique initial state.

We formally recall and revise the definition of *PP-Enforceability* in the following.

**Definition 2 (PP-Enforceability):** Consider  $G, P, L_S, L_{NS}, L_{\text{safe}}$  and  $L_{\text{unsafe}}$ . An insertion function  $f_I$  with its  $f_I^{\text{str}}$ , is PP-enforcing if

- 1)  $\forall te_o \in P(\mathcal{L}(G))$ , where  $t \in E_o^*, e_o \in E_o, \exists t_I \in E_o^* \text{ s.t. } f_I(t, e_o) = t_I e_o$ ,
- 2)  $\forall t \in P(\mathcal{L}(G)), f_I^{\text{str}}(t) \in L_{\text{safe}}$ , and
- 3)  $\forall \tilde{t} \in f_I^{\text{str}}(P(\mathcal{L}(G))), \exists t \in L_{\text{safe}} \text{ s.t. } \tilde{t} \preceq f_I^{\text{str}}(t)$ .

When only Conditions (1) and (2) in Definition 2 are satisfied, the insertion function  $f_I$  is privately-enforcing [24]. The system is called *privately enforceable* if there exists a privately-enforcing insertion function  $f_I$ . The problem of verifying whether a given system is privately enforceable<sup>2</sup> or not has been well studied in [22]. When only Conditions (1) and (3) in Definition 2 are satisfied, the insertion function  $f_I$  is publicly-enforcing [24], [28]. Notice that [24] and [28] require  $\tilde{t} = f_I^{\text{str}}(t)$  in Condition (3), unlike our definition here. The following example highlights the difference.

**Example 1:** Suppose that there is an observer/estimator of a system  $G$  (the system  $G$  is not explicitly presented here) with  $E_o = \{a, b, c, d, e\}$ , set of observable strings  $\{abcde, bcde, c, d\}$ , and set of secret revealing strings  $\{c, bcde\}$ . Then, we have  $L_{\text{safe}} = \{abcde, bcd, d\}$ . Consider the insertion function  $f_I : f_I(\varepsilon, d) = bcd, f_I(\varepsilon, c) = bc, f_I(\varepsilon, b) = ab$ , and  $f_I(s, e_o) = e_o \forall se_o \in \{abcde, bcde\} \setminus \{b\}$ . Clearly,  $f_I$  is privately-enforcing since  $f_I^{\text{str}}(abcde) = abcde \in L_{\text{safe}}, f_I^{\text{str}}(bcde) = abcde \in L_{\text{safe}}, f_I^{\text{str}}(c) = bc \in L_{\text{safe}}, f_I^{\text{str}}(d) = bcd \in L_{\text{safe}}$ , and  $f_I^{\text{str}}(P(\mathcal{L}(G))) = \{abcde, bcd\}$ ;  $f_I$  is also PP-enforcing since Condition (3) holds: for any string

<sup>2</sup>More specifically, private enforceability is termed *i-enforceability* in [22].

$\tilde{t} \in f_I^{\text{str}}(P(\mathcal{L}(G)))$ , in particular, for strings  $abcde = f_I^{\text{str}}(bcde)$  and  $bc = f_I^{\text{str}}(c)$  in  $f_I^{\text{str}}(P(\mathcal{L}(G)))$ , there exist two safe strings  $abcde$  and  $d$  in  $L_{\text{safe}}$  such that  $abcde \preceq f_I^{\text{str}}(abcde)$  and  $bc \preceq f_I^{\text{str}}(d)$ . Notice that  $f_I^{\text{str}}(d) = bcd \neq bc$ , i.e., for  $\tilde{t} = bc = f_I^{\text{str}}(c)$ , there does not exist exactly a string  $t \in L_{\text{safe}}$  such that  $bc = f_I^{\text{str}}(t)$ .

### III. ALL INSERTION STRUCTURE (AIS) AND ANALYSIS

For synthesizing purposes, a structure, named AIS [22], [24], [30], is recalled in the following.

Given  $\mathcal{E} = (X_{\mathcal{E}}, E_o, f_{\mathcal{E}}, x_{\mathcal{E},0})$ ,  $\mathcal{E}_d = (X_{\mathcal{E}_d}, E_o, f_{\mathcal{E}_d}, x_{\mathcal{E}_d,0})$ , and letting  $Q = X_{\mathcal{E}_d} \times X_{\mathcal{E}}$  be the information state set, the AIS is a game-like bipartite structure, and formally defined as

$$\text{AIS} = \text{TRIM}^*(\text{AIS}_{\text{pre}} = (Y, Z, E_o, E_o^*, f_{\text{AIS},yz}, f_{\text{AIS},zy}, y_0))$$

where

- 1)  $Y \subseteq Q$  is the set of  $Y$ -states and a state  $y = (x_{\mathcal{E}_d}, x_{\mathcal{E}}) \in Y$  is called a  $Y$  deadlock state if  $\exists e_o \in E_o$  s.t.  $[f_{\mathcal{E}}(x_{\mathcal{E}}, e_o)!] \wedge [f_{\text{AIS},yz}(y, e_o)!]$ .
- 2)  $Z \subseteq Q \times E_o$  is the set of  $Z$ -states, and a state  $z \in Z$  is called a  $Z$  deadlock state if there are no  $f_{\text{AIS},zy}$  transitions defined at  $z$ . Let  $\mathcal{Q}(z), \mathcal{E}(z)$  denote the information state component and event component of  $z \in Z$  respectively, so that  $z = (\mathcal{Q}(z), \mathcal{E}(z))$ .
- 3)  $f_{\text{AIS},yz} : Y \times E_o \rightarrow Z$  is the transition function from  $Y$ -state to  $Z$ -state. For every  $y = (x_{\mathcal{E}_d}, x_{\mathcal{E}}) \in Y$ ,  $e \in E_o$ , we have:  $f_{\text{AIS},yz}(y, e) = z$  if  $[f_{\mathcal{E}}(x_{\mathcal{E}}, e)!] \wedge [\mathcal{Q}(z) = y] \wedge [\mathcal{E}(z) = e]$ .
- 4)  $f_{\text{AIS},zy} : Z \times E_o^* \rightarrow Y$  is the transition function from  $Z$ -state to  $Y$ -state. For every  $z = ((x_{\mathcal{E}_d}, x_{\mathcal{E}}), e) \in Z$  and “cycle-free” string<sup>3</sup>  $t_I$  starting from  $x_{\mathcal{E}_d}$  in  $\mathcal{E}_d$  (i.e.,  $t_I \in \{s \in \mathcal{L}(\mathcal{E}_d, x_{\mathcal{E}_d}) : \nexists s' \prec s \text{ s.t. } f_{\mathcal{E}_d}(x_{\mathcal{E}_d}, s') = f_{\mathcal{E}_d}(x_{\mathcal{E}_d}, s)\}$ ), we have:  $f_{\text{AIS},zy}(z, t_I) = y$  if  $[f_{\mathcal{E}_d}(x_{\mathcal{E}_d}, t_I e)!] \wedge [y = (f_{\mathcal{E}_d}(x_{\mathcal{E}_d}, t_I e), f_{\mathcal{E}}(x_{\mathcal{E}}, e))]$ .
- 5)  $y_0 = (x_{\mathcal{E}_d,0}, x_{\mathcal{E},0}) \in Y$  is the unique initial  $Y$  state.
- 6) TRIM is an operation that prunes away all  $Y$  and  $Z$  deadlock states in the structure and takes the accessible part from  $y_0$ . TRIM\* means repeating TRIM operation on AIS<sub>pre</sub> until no more  $Y$  and  $Z$  deadlock states exist.

An instance of AIS is presented in Example 3.

*Proposition 1* (see [22], [24]): The AIS embeds all and only all privately-enforcing insertion functions in its transitions.

With a slight abuse of notation, we denote AIS by the form AIS =  $(Y, Z, E_o, E_o^*, f_{\text{AIS},yz}, f_{\text{AIS},zy}, y_0)$  hereafter, and recall/extend the notions related to AIS in [24] as below.

A run  $r$  generated by the AIS is a sequence of transitions starting from the initial state  $y_0$  and ending up with a  $Y$ -state:  $r = \langle y_0 \xrightarrow{e_0} z_0 \xrightarrow{s_0} y_1 \xrightarrow{e_1} \dots y_{k-1} \xrightarrow{e_{k-1}} z_{k-1} \xrightarrow{s_{k-1}} y_k \rangle$ , where  $e_i \in E_o$ ,  $s_i \in E_o^*$ ,  $z_i = f_{\text{AIS},yz}(y_i, e_i)$ , and  $y_{i+1} = f_{\text{AIS},zy}(z_i, s_i)$  for  $0 \leq i < k$ . The set of runs in the AIS is denoted by  $\mathcal{R}$ . The original events in run  $r$  are defined as:  $S_{\text{ori}}(r) = e_0 e_1 \dots e_{k-1}$ . The map  $S_{\text{ori}}$  erases all states and inserted strings in run  $r$ , and shows the original events corresponding to the run  $r$ . The string generated by run  $r$  is defined as:  $S_p(r) = s_0 e_0 s_1 e_1 \dots s_{k-1} e_{k-1}$  and the map  $S_p$  erases all states in run  $r$  and then swaps every consecutive  $e_i$  and  $s_i$  pair. In particular, for run  $r = \langle y_0 \rangle$ , we define  $S_p(\langle y_0 \rangle) = S_{\text{ori}}(\langle y_0 \rangle) = \varepsilon$ . For a run  $r \neq \langle y_0 \rangle$ , we have  $S_{\text{ori}}(r) \in S_p(r)$ , i.e., string  $S_{\text{ori}}(r)$  is “embedded” in string  $S_p(r)$ .

For  $t \in \mathcal{L}(\mathcal{E})$ , we define  $M_p(t) = \{S_p(r) \in L_{\text{safe}} : S_{\text{ori}}(r) = t, \text{ where } r \in \mathcal{R}\}$ . The  $M_p(t)$  is the set of strings that  $t$  can map to under these privately-enforcing insertion functions that can be synthesized from the AIS. For any  $s \in M_p(t)$ , we have  $t \in s$ .

*Proposition 2:* For a given finite length string  $t$ , we have that the cardinality of  $M_p(t)$  is also finite.

<sup>3</sup>We claim that the cycle-free requirement, which has also been deployed in [24], does not compromise the functionality of the AIS.

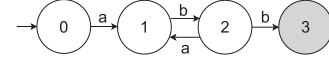


Fig. 2. System model with  $E = E_o = \{a, b\}$  and secret state set  $X_S = \{3\}$ .

Now, we are ready to discuss and derive one of our main results. In [24], the following partition is proposed for safe language  $L_{\text{safe}}$  as  $L_{\text{safe}}^1 \dot{\cup} L_{\text{safe}}^2$ , where  $L_{\text{safe}}^1 = \overline{L_{\text{safe}}}$ ,  $\overline{L_{\text{safe}}} = \{s \in L_{\text{safe}} : \nexists u \in L_{\text{unsafe}}, \text{ s.t., } s \preceq u\}$ , and  $L_{\text{safe}}^2 = L_{\text{safe}} \setminus L_{\text{safe}}^1$ .  $\overline{L_{\text{safe}}}$  is the set of safe strings that are not prefixes of any unsafe strings, and  $L_{\text{safe}}^2$  is a subset of safe prefixes of unsafe strings. Based on the abovementioned partition [24, Prop. 2] claims that  $L_{\text{safe}}^1 \neq \emptyset$  if private safety is enforceable. However, we argue that the abovementioned result is not always correct.

*Example 2:* Consider system  $G$  in Fig. 2 with  $E = E_o = \{a, b\}$  and secret state set  $X_S = \{3\}$ , where  $L_{\text{safe}} = \{a\}\{ba\}^*$ ,  $L_{\text{unsafe}} = \{a\}\{ba\}^*\{bb\}$ ,  $L_{\text{safe}}^1 = \emptyset$  and  $L_{\text{safe}}^2 = L_{\text{safe}}$ . Define  $f_I$  so that  $f_I(E^*b, b) = ab$  and  $f_I(s, e) = e, s \in \mathcal{L}(G), e \in E$  otherwise. Because  $f_I^{\text{str}}(P(\mathcal{L}(G))) = f_I^{\text{str}}(\mathcal{L}(G)) = \{a\}\{ba\}^* \subseteq L_{\text{safe}}$ ,  $f_I$  is a privately-enforcing insertion function. One can conclude that system  $G$  in Fig. 2 is privately enforceable, while  $L_{\text{safe}}^1 = \emptyset$ , which contradicts [24, Prop. 2].

The partition  $L_{\text{safe}}^1 \dot{\cup} L_{\text{safe}}^2$  is based on the observation that some safe strings are prefixes of unsafe strings while others are not. In [24], Proposition 2 is proved by contradiction: since all the continuations of unsafe strings are also unsafe if  $L_{\text{safe}}^1 = \emptyset$ , then one can never map an unsafe string to a string in  $L_{\text{safe}}^1$ . However, Example 2 shows that this is not always true. We can map unsafe strings to  $L_{\text{safe}}^2$  if  $L_{\text{safe}}^1 = \emptyset$ . The condition that continuations of unsafe strings are unsafe, may have no effect on the map as they can also be mapped into  $L_{\text{safe}}^2$ . Also important is the fact that the length/number of safe prefixes of unsafe strings could be infinite. Indeed, it does not matter whether  $L_{\text{safe}}^1$  is empty or not for the enforceability of private safety.

By reviewing the insertion mechanism, we observe that some strings must be modified while others need not. For example, unsafe strings in  $L_{\text{unsafe}}$  should be modified. Based on this observation, the observed system language can be partitioned as follows.

Language  $P(\mathcal{L}(G))$  is partitioned into two parts: (i)  $L_{\mathcal{E}}^1 \subseteq P(\mathcal{L}(G))$ : a set of strings  $s \in P(\mathcal{L}(G))$  that satisfy  $f_I^{\text{str}}(s) \neq s$  under all possible privately-enforcing insertion functions  $f_I$ ; (ii)  $L_{\mathcal{E}}^2 := P(\mathcal{L}(G)) \setminus L_{\mathcal{E}}^1$ . Or, we can define  $L_{\mathcal{E}}^2$  as the collection of strings  $s \in P(\mathcal{L}(G))$ , which satisfy  $f_I^{\text{str}}(s) = s$  under a privately-enforcing insertion function  $f_I$ .

$L_{\mathcal{E}}^1$  is the set of strings that must be modified, while  $L_{\mathcal{E}}^2$  is the set of strings that can remain unchanged. Clearly, we have  $L_{\mathcal{E}}^2 = \overline{L_{\mathcal{E}}^1}$ , and  $L_{\text{unsafe}} \subseteq L_{\mathcal{E}}^1$  as any string  $s \in L_{\text{unsafe}}$  should be modified for safety. Also  $L_{\mathcal{E}}^2 \subseteq L_{\text{safe}}$  as there may exist a safe string that must be modified. For example, consider the system in Example 1, we have  $L_{\mathcal{E}}^1 = \{c, bcde\}$ ,  $L_{\text{unsafe}} = \{c, bcde\}$ ,  $L_{\mathcal{E}}^2 = \{d, abcde\}$ , and  $L_{\text{safe}} = \{d, bcd, abcde\}$ , where  $L_{\text{unsafe}} \subseteq L_{\mathcal{E}}^1$  and  $L_{\mathcal{E}}^2 \subseteq L_{\text{safe}}$ . Since  $\{bcd\}$  is the set of safe prefixes for unsafe string  $bcde$ , any string  $s \in \{bcd\}$  should be modified. While, in Example 2, the string set that must be modified is  $L_{\mathcal{E}}^1 = L_{\text{unsafe}} = \{a\}\{ba\}^*\{bb\}$  and  $L_{\mathcal{E}}^2 = L_{\text{safe}} = \{a\}\{ba\}^*$ . The new partition here captures the feature of strings in  $P(\mathcal{L}(G))$  better.

As  $L_{\mathcal{E}}^1$  is the set of strings that must be modified, the string in  $L_{\mathcal{E}}^1$  cannot map to itself according to the definition of  $L_{\mathcal{E}}^1$ . We have the following proposition for  $L_{\mathcal{E}}^1$ .

*Proposition 3:* For any string  $s \in P(\mathcal{L}(G))$ ,  $s \in L_{\mathcal{E}}^1$  if and only if  $s \notin M_p(s)$ .

If  $s \in L_{\mathcal{E}}^2$ , then  $\forall \alpha \in M_p(s)$ , we have  $s \in \alpha$  and  $|s| < |\alpha|$ .

For  $L_{\mathcal{E}}^2$ , we have Proposition 4.



**Proposition 4:** For any string  $s \in P(\mathcal{L}(G))$ ,  $s \in L_{\mathcal{E}}^2$  if and only if  $s \in M_p(s)$ . Or, For any string  $st \in P(\mathcal{L}(G))$ ,  $s \in L_{\mathcal{E}}^2$  if and only if there exists a string  $\alpha \in M_p(st)$  such that  $s \preceq \alpha$ .

**Proof:** From the definitions of  $L_{\mathcal{E}}^2$  and  $M_p$ , we have that  $s \in L_{\mathcal{E}}^2$  if and only if  $s \in M_p(s)$ .

The second statement can be proven as follows. First, we have that for a string  $\alpha \in M_p(st)$ , there must exist a string  $\alpha' \preceq \alpha$  such that  $\alpha' \in M_p(s)$ . This is because, in AIS, any run  $r' \in \mathcal{R}$  s.t.  $S_{\text{ori}}(r') = s$  can be extended to a run  $r \in \mathcal{R}$  s.t.  $S_{\text{ori}}(r) = st$  and any run  $r \in \mathcal{R}$  s.t.  $S_{\text{ori}}(r) = st$  can only be extended from a run  $r'$  s.t.  $S_{\text{ori}}(r') = s$ . Then,  $(\Rightarrow)$  for a string  $st \in P(\mathcal{L}(G))$ , we have that  $\forall \alpha' \in M_p(s), \exists \alpha \in M_p(st)$  such that  $\alpha' \preceq \alpha$ . In particular, if  $s \in M_p(s)$ , i.e.,  $s \in L_{\mathcal{E}}^2$ , then  $\exists \alpha \in M_p(st)$  such that  $s \preceq \alpha$ .  $(\Leftarrow)$  If  $\nexists \alpha \in M_p(st)$  s.t.  $s \preceq \alpha$ , then this implies that  $s$  must be modified, i.e.,  $s \in L_{\mathcal{E}}^1$ . ■

**Proposition 5:**  $\forall s \in L_{\mathcal{E}}^2 \forall st \in P(\mathcal{L}(G))$ , we have  $M_p(st)|s \neq \emptyset$ .

**Proof:** If  $M_p(st)|s = \emptyset$ , then  $\forall \alpha \in M_p(st)$ ,  $s$  is not a prefix of  $\alpha$ , and it implies that  $s \in L_{\mathcal{E}}^1$  must be modified, which is a contradiction. ■

According to [24],  $L_{\text{safe}} \neq \emptyset$  if private safety is enforceable. Furthermore, the following proposition shows the nonemptiness of  $L_{\mathcal{E}}^2 \subseteq L_{\text{safe}}$  when private safety is enforceable.

**Proposition 6:**  $L_{\mathcal{E}}^2 \neq \emptyset$  if private safety is enforceable.

**Proof:** If private safety is enforceable, then the AIS is not empty. We prove that  $L_{\mathcal{E}}^2 \neq \emptyset$  by contradiction. If  $L_{\mathcal{E}}^2 = \emptyset$ , then  $L_{\mathcal{E}}^1 = P(\mathcal{L}(G))$ . For a finite length  $t_0 \in P(\mathcal{L}(G)) = L_{\mathcal{E}}^1$ , the cardinality of  $M_p(t_0)$  is finite (Proposition 2). Let  $n = |E_o|$  be the number of events in  $E_o$ , then we can find a sequence of finite length strings  $t_i$  such that  $M_p(t_i) \neq \emptyset$ ,  $t_i \in M_p(t_{i-1})$  (i.e.,  $t_{i-1} \in t_i$ ) and  $|t_i| - |t_{i-1}| \geq 1$ ,  $i = 1, 2, \dots, n$ . Since there are only  $n$  different events in  $E_o$ , there must exist two indices  $i_1$  and  $i_2$  with  $0 \leq i_1 < i_2 \leq n$  such that  $t_{i_1}$  and  $t_{i_2}$  have a same first event  $e_o \in E_o$ , i.e.,  $e_o = (t_{i_1})_1 = (t_{i_2})_1$ . Notice that  $t_{i_1} \in t_{i_2}$  means that when one maps string  $t_{i_1}$  to  $t_{i_2}$ , for the prefix  $e_o \preceq t_{i_1}$ , one can also keep  $e_o$  unchanged, which implies that  $e_o \in L_{\mathcal{E}}^2$ . However,  $e_o$  is assumed to be  $e_o \in L_{\mathcal{E}}^1 = P(\mathcal{L}(G))$ , which is a contradiction. The proof is completed. ■

**Proposition 7:** If AIS is not empty, i.e., private safety is enforceable, then  $\forall s \in P(\mathcal{L}(G)), \exists r \in \mathcal{R}, t \in L_{\mathcal{E}}^2$  s.t.  $S_{\text{ori}}(r) = s \wedge S_p(r) = t$ .

**Proof:** For every string  $s \in P(\mathcal{L}(G))$ , we can always find a run  $r \in \mathcal{R}$  s.t.  $S_{\text{ori}}(r) = s$  due to the fact that the AIS is built based on  $\mathcal{E}$ , and all observed system behaviors have been embedded in the AIS.

For every string  $s \in L_{\mathcal{E}}^2$ , we can always find a run  $r \in \mathcal{R}$  s.t.  $S_p(r) = s$  as there exists a privately-enforcing insertion function  $f_I$  s.t.  $f_I^{\text{str}}(s) = s$ .

For strings  $s \in L_{\mathcal{E}}^1$ , we prove by contradiction. Assume that there exists a string set  $A \subseteq L_{\mathcal{E}}^1$ , such that  $\forall s \in A \forall r \in \mathcal{R}$  satisfying  $S_{\text{ori}}(r) = s$ ,  $\nexists t \in L_{\mathcal{E}}^2$  s.t.  $S_p(r) = t$ . Then,  $\forall s_0 \in A$ , we have  $[M_p(s_0) \cap L_{\mathcal{E}}^2 = \emptyset] \wedge [M_p(s_0) \cap (L_{\mathcal{E}}^1 \setminus A) = \emptyset]$ , as  $L_{\mathcal{E}}^1 \setminus A$  is the string subset of  $L_{\mathcal{E}}^1$  that can map to  $L_{\mathcal{E}}^2$ . If  $M_p(s_0) \cap (L_{\mathcal{E}}^1 \setminus A) \neq \emptyset$ , then there exist strings  $l_1 \in (L_{\mathcal{E}}^1 \setminus A)$ ,  $l_2 \in L_{\mathcal{E}}^2$  such that  $[s_0 \in l_1 \in l_2] \Rightarrow [s_0 \in l_2]$ . Therefore,  $\forall s \in A$ , we have  $M_p(s) \subseteq A$ . However, such subset  $A$  does not exist, see Appendix. ■

Roughly speaking, in the proof of Proposition 7, as  $L_{\mathcal{E}}^1$  is the collection of strings that must be modified, one cannot map  $L_{\mathcal{E}}^1$  to a subset of itself. Proposition 7 points out that there always exist safe strings  $t \in L_{\mathcal{E}}^2$  for strings  $s \in P(\mathcal{L}(G))$  to map to whenever the system is privately enforceable and in later sections, this observation will be helpful for synthesizing PP-enforcing insertion functions.

## IV. PRIVATELY-AND-PUBLICLY KNOWN INSERTION FUNCTIONS

In the following, we first recall the existing greedy synthesis method, and then improve it by proposing a new criterion. A formal proof is presented for the existence of PP-enforcing insertion functions.

### A. Existing Greedy Synthesizing Method

Here, we first recall some necessary knowledge for the existing greedy synthesis method. Recall the sufficient condition for the existence of PP-enforcing insertion functions.

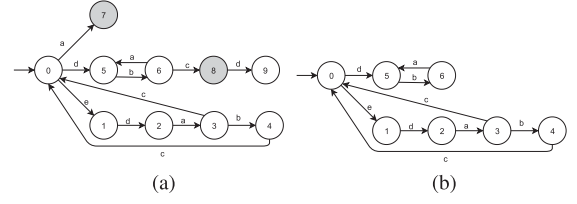


Fig. 3. Estimator  $\mathcal{E}$  with secret revealing states 7 and 8, and the desired estimator  $\mathcal{E}_d$ .

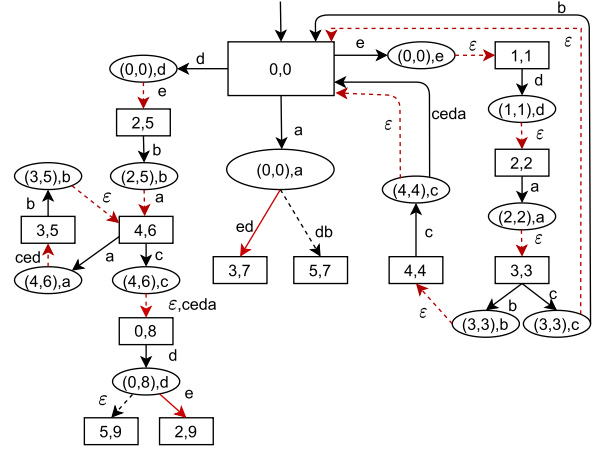


Fig. 4. AIS. The rectangular/elliptical states are Y/Z states. The dashed arrows are chosen under the GM-criterion in Definition 3 and the red arrows are chosen under the improved GM-criterion in Definition 4. These red-dashed arrows coincide (by chance) with both GM-criteria in Definitions 3 and 4 for this model in Fig. 3(a).

**Lemma 1** (see [24]): A privately-enforcing insertion function  $f_I$  is also publicly-enforcing if there exists a language  $L \subseteq L_{\text{safe}}$  such that  $f_I^{\text{str}}(P(\mathcal{L}(G))) = L$  and  $f_I^{\text{str}}(L) = L$ .

Depending on whether the  $\varepsilon$  transition is defined at Z-states, the set of Z-states can be partitioned into two subsets: 1)  $Z_1$ , defined as the Z-states in AIS where  $\varepsilon$  transitions exist and 2)  $Z_2$ , defined as the remaining Z-states, where only non  $\varepsilon$  transitions are defined.

**Definition 3 (GM-Criterion [24]):**

- 1) At every  $z \in Z_1$  in the AIS, choose  $\varepsilon$  insertion.
- 2) At every  $z \in Z_2$  in the AIS, choose for insertion choice any string  $s_{\text{max}} \in \arg \max\{|s|, s \in \Gamma(z)\}$ , where  $\Gamma(z) = \bigcup\{s \in E_o^* : f_{\text{AIS}, z, y}(z, s) \text{ is defined}\}$ .

Based on [24, Def. 3, Algorithm 2], one can synthesize the so-called “greedy-maximal insertion function  $f_{\text{greedy}}$ ” [24] from the AIS. We would like to point out that the above synthesized “greedy-maximal insertion functions  $f_{\text{greedy}}$ ” are not always PP-enforcing as claimed in [24]. We illustrate this in the following counter-example.

**Example 3:** Consider a new system  $G$  whose estimator  $\mathcal{E}$  is obtained in Fig. 3(a) with secret-revealing states 7 and 8. Then, for  $\mathcal{E}$  in Fig. 3(a), the corresponding desired estimator is shown in Fig. 3(b). The model here is a customized estimator derived in [24, Example 6]. The AIS is illustrated in Fig. 4 [transition  $((4, 4), c) \xrightarrow{\text{ceda}} (0, 0)$  here should be added to Fig. 9 in [24, Example 6] since string  $\text{ceda} \in \mathcal{L}(\mathcal{E}_d, 4)$  is a cycle-free string]. For Z state  $((0, 0), a) \in Z_2$  in Fig. 4, according to Definition 3, string  $db$  can be chosen as the lengths of insertion choices  $db$  and  $ed$  are equivalent. Moreover,  $e$  transition at state  $((0, 0), d)$  is chosen. The transitions represented by the dashed arrows at Z-states are the choices obtained under the GM-criterion given in Definition 3, and the synthesized “greedy-maximal insertion function  $f_{\text{greedy}}$ ” [24] is encoded in Fig. 5 with  $0 \xrightarrow{a/dba} 7$ .

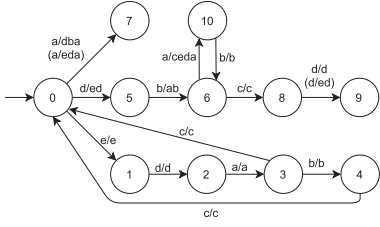


Fig. 5. I/O automata. The transitions  $0 \xrightarrow{a/dba} 7$  and  $8 \xrightarrow{d/d} 9$  are obtained under the GM-criterion in Definition 3 for “greedy-maximal insertion function  $f_{\text{greedy}}$ ” [24] and transitions  $0 \xrightarrow{a/eda} 7$  and  $8 \xrightarrow{d/ed} 9$  are obtained under the improved GM-criterion in Definition 4 for  $f_{I,\text{greedy}}$  in Algorithms 2 or 3 in this article. Note that the states in the I/O automata synthesized from the AIS have been relabelled with numbers  $0, 1, \dots, 10$ .

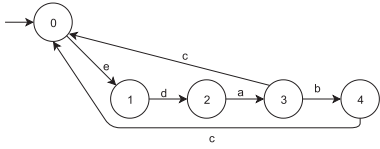


Fig. 6. Subestimator/subautomaton  $\mathcal{E}_d^{\text{pp}}$ .

For the above obtained  $f_{\text{greedy}}$ , we show that it is not PP-enforcing. Given unsafe string  $a \in \mathcal{L}(\mathcal{E})$ , we have  $f_{\text{greedy}}^{\text{str}}(a) = dba$ , and the original system’s safe behavior  $dba$  has been modified to  $f_{\text{greedy}}^{\text{str}}(dba) = edabcda$ . When the intruder observes string  $dba$ , the intruder can assert that secret behavior  $a$  in  $0 \xrightarrow{a} 7$  has occurred and secret-revealing state 7 is exposed, since there is no safe string  $t \in L_{\text{safe}}$  that can map to a safe behavior string  $\tilde{t}_1 \in L_{\text{safe}}$  such that  $\tilde{t}_2 = dba$  is a prefix of  $\tilde{t}_1$ , i.e., Condition (3) in Definition 2 has been violated. As a result, in [24, Th. 2], i.e., the statement that “a greedy-maximal insertion function is PP-enforcing,” is not correct according to this example. Whether opacity is PP-enforceable when it is privately enforceable (in [24, Corollary 1]), is still unclear and we further investigate it later in this article. ■

Furthermore, the requirement that choosing  $\varepsilon$  insertion at  $Z_1$  states in Definition 3 may also not be permissible.

**Example 4:** Reconsider Example 3. If one chooses the  $\varepsilon$  insertion at  $Z_1$  state  $((0, 8), d)$  in Fig. 4 (which corresponds to transition  $8 \xrightarrow{d/d} 9$  in Fig. 5), then the unsafe string  $dbcd$  would be modified as  $edabcd$ . However, the original safe behavior  $edabcd$  has been modified as  $edabcde$  and no safe behavior  $t \in L_{\text{safe}}$  could be modified as a string  $\tilde{t} \in L_{\text{safe}}$  that has prefix  $edabcde$ , i.e., Condition (3) in Definition 2 is not satisfied. Thus, the insertion function in Fig. 5 with transition  $8 \xrightarrow{d/d} 9$  would not be PP-enforcing. ■

In other words, one cannot synthesize PP-enforcing insertion functions from the AIS by purely examining the lengths of the insertion choices at  $Z_2$  states or simply choosing  $\varepsilon$  insertion at  $Z_1$  states in the AIS.

## B. Improved Greedy Synthesizing Method

Following the ideas in Lemma 1 and [28] that a language  $L \subseteq L_{\text{safe}}$  s.t.  $f_I^{\text{str}}(L) = L$  should be first identified. For this purpose, we first build/identify a subestimator/subautomaton of  $\mathcal{E}_d$  through Algorithm 1, which is denoted as  $\mathcal{E}_d^{\text{pp}} = (X_{\mathcal{E}_d}^{\text{pp}}, E_o, f_{\mathcal{E}_d}^{\text{pp}}, x_{\mathcal{E}_d,0}^{\text{pp}})$ . When building  $\mathcal{E}_d^{\text{pp}}$ , we only consider those  $\varepsilon$  transitions at  $Z$ -states. For example, the built  $\mathcal{E}_d^{\text{pp}}$  w.r.t. AIS in Fig. 4 and  $\mathcal{E}_d$  in Fig. 3(b) is shown in Fig. 6. To obtain  $\mathcal{E}_d^{\text{pp}}$ , Algorithm 1 starts from  $y_0 = (0, 0)$  and  $Q$  is initialized as  $Q = \{y_0\}$ . For events  $a, d$ , and  $e$  at  $y = y_0 = (0, 0)$ ,  $Z$  states  $z_1 = ((0, 0), a)$ ,  $z_2 = ((0, 0), d)$ , and  $z_3 = ((0, 0), e)$  are reachable from

## Algorithm 1: Build $\mathcal{E}_d^{\text{pp}}$ .

**Input:** AIS =  $(Y, Z, E_o, E_o^*, f_{\text{AIS},yz}, f_{\text{AIS},zy}, y_0)$ ,  
 $\mathcal{E}_d = (X_{\mathcal{E}_d}, E_o, f_{\mathcal{E}_d}, x_{\mathcal{E}_d,0})$   
**Output:**  $\mathcal{E}_d^{\text{pp}} = (X_{\mathcal{E}_d}^{\text{pp}}, E_o, f_{\mathcal{E}_d}^{\text{pp}}, x_{\mathcal{E}_d,0}^{\text{pp}})$   
1:  $f_{\mathcal{E}_d}^{\text{pp}} := \emptyset, Q := \{y_0\}$   
2: **for all**  $y \in Q$  that have not been examined **do**  
3: **for**  $e \in E_o$  s.t.  $f_{\text{AIS},yz}(y, e)$  is defined **do**  
4:  $z = ((x_{\mathcal{E}_d}, x_{\mathcal{E}}), e) = f_{\text{AIS},yz}(y, e)$   
5: **if**  $f_{\text{AIS},zy}(z, \varepsilon)$  is defined **then**  
6:  $f_{\mathcal{E}_d}^{\text{pp}}(x_{\mathcal{E}_d}, e) := f_{\mathcal{E}_d}(x_{\mathcal{E}_d}, e)$   
7:  $Q := Q \cup \{f_{\text{AIS},zy}(z, \varepsilon)\}$   
8:  $\mathcal{E}_d^{\text{pp}} := (X_{\mathcal{E}_d}, E_o, f_{\mathcal{E}_d}^{\text{pp}}, x_{\mathcal{E}_d,0}^{\text{pp}})$   
9: **Return** the Trimmed  $\mathcal{E}_d^{\text{pp}}$

$y = (0, 0)$ , while, only at state  $z_3$ , there exists an  $\varepsilon$  transition; thus, transition  $0 \xrightarrow{e} 1$  is added to  $\mathcal{E}_d^{\text{pp}}$  and  $Q = Q \cup \{(1, 1)\} = \{(0, 0), (1, 1)\}$ . For  $y = (1, 1) \in Q$ , we need to check  $\varepsilon$  transitions for all  $Z$  states that are reachable from  $(1, 1)$  with a single-step transition, and so forth. Now, we are in the position to find a PP-enforcing insertion function  $f_I$  ( $f_I^{\text{str}}$ ) such that  $f_I^{\text{str}}(\mathcal{L}(\mathcal{E}_d^{\text{pp}})) = \mathcal{L}(\mathcal{E}_d^{\text{pp}})$  and  $f_I^{\text{str}}(P(\mathcal{L}(G))) = \mathcal{L}(\mathcal{E}_d^{\text{pp}})$ .

**Definition 4 (Improved GM-Criterion):** At every  $z = ((x_1, x_2), e) \in Z$  in the AIS

- 1) if  $z \in Z_1$ , choose  $\varepsilon$  transition if  $z \xrightarrow{\varepsilon} y = (x'_1, x'_2)$  s.t.  $x'_1 \in X_{\mathcal{E}_d}^{\text{pp}}$  in  $\mathcal{E}_d^{\text{pp}}$ ;
- 2) if the above condition is not satisfied or  $z \in Z_2$ , then randomly choose one  $s_I$ , where  $z \xrightarrow{s_I} y = (x'_1, x'_2)$  s.t.  $x'_1 \in X_{\mathcal{E}_d}^{\text{pp}}$  and  $f_{\mathcal{E}_d}^{\text{pp}}(x_1, s_I)$  is defined in  $\mathcal{E}_d^{\text{pp}}$ ;
- 3) otherwise, undefined.

Condition (1) in Definition 4 is used to guarantee that every string  $s \in \mathcal{L}(\mathcal{E}_d^{\text{pp}})$  is mapped to itself. Condition (2) in Definition 4 is used to guarantee that the modified output strings belong to  $\mathcal{L}(\mathcal{E}_d^{\text{pp}})$ .

By leveraging Lemma 1 with the improved GM-criterion in Definition 4, Algorithm 2 is shown here for synthesizing a (greedy) insertion function  $f_I$ . Algorithm 2 starts from initial state  $x_{ia,0} = y_0$ . For a state  $x_{ia}$ , all transitions  $f_{\text{AIS},yz}(x_{ia}, e)$  defined are considered. For a reached  $Z$ -state, i.e.,  $z = f_{\text{AIS},yz}(x_{ia}, e)$ , we first check whether  $z \in Z_1$ . If  $z \in Z_1$ , then we use Condition (1) in Definition 4 (i.e., lines 5–9 in Algorithm 2) to choose next the  $x_{ia}$  state; otherwise, we use Condition (2) in Definition 4 (i.e., lines 11–14 in Algorithm 2) to choose the next  $x_{ia}$  state. The insertion function  $f_I$  synthesized from Algorithm 2 is called a *greedy insertion function* hereafter, denoted as  $f_{I,\text{greedy}}$ .

**Example 5:** Reconsider Example 3. The  $\mathcal{E}_d^{\text{pp}}$  built from the AIS is shown in Fig. 6.

The AIS with the improved GM-criterion in Definition 4 is shown in Fig. 4. The red arrows represent the corresponding choices when using the improved GM-criterion. The corresponding greedy insertion function  $f_{I,\text{greedy}}$  is encoded in Fig. 5 with  $0 \xrightarrow{a/eda} 7$  and  $8 \xrightarrow{d/ed} 9$ . ■

## C. Verification for the Greedy Insertion Functions $f_{I,\text{greedy}}$

**Lemma 2:** If  $f_{I,\text{greedy}}$  is synthesized from Algorithm 2, then  $f_{I,\text{greedy}}^{\text{str}}(P(\mathcal{L}(G))) = L_{\mathcal{E}}^2$ .

**Proof:** First, we prove that  $\mathcal{L}(\mathcal{E}_d^{\text{pp}}) = L_{\mathcal{E}}^2$ . Recall that  $L_{\mathcal{E}}^2$  is the collection of strings  $s$  satisfying  $f_I^{\text{str}}(s) = s$  under a privately-enforcing insertion function  $f_I$ . AIS enumerates all and only all privately-enforcing insertion functions  $f_I$ . For every string  $s \in L_{\mathcal{E}}^2$ , there exists a run  $r \in \mathcal{R}$  s.t.  $S_{\text{on}}(r) = s$  and  $S_p(r) = s$ , i.e., for all  $z$  in this  $r$ ,  $f_{\text{AIS},zy}(z, \varepsilon)$  is defined. According to the construction (Algorithm 1) of  $\mathcal{E}_d^{\text{pp}}$ , we have  $\mathcal{L}(\mathcal{E}_d^{\text{pp}}) = L_{\mathcal{E}}^2$ . Indeed,  $\mathcal{L}(\mathcal{E}_d^{\text{pp}}) = L_{\mathcal{E}}^2$  is established from the fact that every state in  $\mathcal{E}_d^{\text{pp}}$  can be derived from a run  $r$ , where the run  $r$  only has  $\varepsilon$  transitions for all  $Z$  states in it.

**Algorithm 2:** Synthesize a (Greedy) Insertion Function  $f_I$ .

---

**Input:** AIS =  $(Y, Z, E_o, E_o^*, f_{\text{AIS},yz}, f_{\text{AIS},zy}, y_0)$ ,  
 $\mathcal{E}_d^{\text{PP}} = (X_{\mathcal{E}_d}^{\text{PP}}, E_o, f_{\mathcal{E}_d}^{\text{PP}}, x_{\mathcal{E}_d,0}^{\text{PP}})$

**Output:** IA =  $(X_{ia}, E_o, E_o^+, f_{ia}, q_{ia}, x_{ia,0})$

- 1:  $x_{ia,0} := y_0, X_{ia} := \{x_{ia,0}\}$
- 2: **for all**  $x_{ia} = (x_1, x_2) \in X_{ia}$  that have not been examined **do**
- 3:   **for**  $e \in E_o$  s.t.  $f_{\text{AIS},yz}(x_{ia}, e)$  is defined **do**
- 4:      $z = ((x_1, x_2), e) = f_{\text{AIS},yz}(x_{ia}, e)$
- 5:     **if**  $[f_{\text{AIS},zy}(z, \varepsilon)$  is defined]  $\wedge [x'_1 \in X_{\mathcal{E}_d}^{\text{PP}}$ , where  
 $(x'_1, x'_2) = f_{\text{AIS},zy}(z, \varepsilon)]$  **then**
- 6:        $y = f_{\text{AIS},zy}(z, \varepsilon)$
- 7:        $x'_{ia} := y$
- 8:        $f_{ia}(x_{ia}, e) := x'_{ia}$
- 9:        $q_{ia}(x_{ia}, e) := e$
- 10:    **else**
- 11:     find a transition  $z \xrightarrow{S_I} y = (x'_1, x'_2)$  in  $f_{\text{AIS},zy}$  where  
 $x'_1 \in X_{\mathcal{E}_d}^{\text{PP}}$  and  $f_{\mathcal{E}_d}^{\text{PP}}(x_1, s_I)$  is defined
- 12:      $x'_{ia} := y$
- 13:      $f_{ia}(x_{ia}, e) := x'_{ia}$
- 14:      $q_{ia}(x_{ia}, e) := s_I e$
- 15:      $X_{ia} := X_{ia} \cup \{x'_{ia}\}$
- 16: **Return** IA

---

**Algorithm 3:** Synthesize Greedy Insertion Functions  $f_{I,\text{greedy}}$  for System  $G$ .

---

**Input:**  $G = (X, E, f, X_0), P, X_S$

**Output:** A PP-enforcing IA

- 1: Build  $\mathcal{E}, \mathcal{E}_d$
- 2: Construct All Insertion Structure (AIS)
- 3: Build  $\mathcal{E}_d^{\text{PP}}$  by Algorithm 1
- 4: Synthesize a greedy insertion function from AIS by Algorithm 2

---

Next, we prove that  $f_{I,\text{greedy}}^{\text{str}}(P(\mathcal{L}(G))) = L_{\mathcal{E}}^2$ . We have proved that  $\mathcal{L}(\mathcal{E}_d^{\text{PP}}) = L_{\mathcal{E}}^2$ , and Proposition 7 points out that all strings in  $P(\mathcal{L}(G))$  can be modified into  $L_{\mathcal{E}}^2$ . The improved GM-criterion in Definition 4 is used to guarantee that all strings are modified into  $\mathcal{L}(\mathcal{E}_d^{\text{PP}})$  and all strings in  $\mathcal{L}(\mathcal{E}_d^{\text{PP}})$  remain unchanged. Algorithm 2 is designed under the guidance of the criterion in Definition 4, thus, we have  $f_{I,\text{greedy}}^{\text{str}}(P(\mathcal{L}(G))) = L_{\mathcal{E}}^2$ . ■

Proposition 7 guarantees that Condition (3) in Definition 4 will never occur when executing Algorithm 2, i.e., we will never reach a  $Z$ -state where only non  $\varepsilon$  transitions are defined and the requirements for Condition (2) in Definition 4 are not satisfied. Furthermore, we have the following result.

*Theorem 1:* The insertion function  $f_{I,\text{greedy}}$  is PP-enforcing.

*Proof:* Notice that  $f_{I,\text{greedy}}$  is a privately-enforcing insertion function as it is derived from the AIS. Based on Lemma 2, we have  $f_{I,\text{greedy}}^{\text{str}}(P(\mathcal{L}(G))) = L_{\mathcal{E}}^2$ . Based on the criterion given in Definition 4, we have  $f_{I,\text{greedy}}^{\text{str}}(L_{\mathcal{E}}^2) = f_{I,\text{greedy}}^{\text{str}}(\mathcal{L}(\mathcal{E}_d^{\text{PP}})) = \mathcal{L}(\mathcal{E}_d^{\text{PP}}) = L_{\mathcal{E}}^2$ . Thus, according to Lemma 1, the privately-enforcing insertion function  $f_{I,\text{greedy}}$  is also PP-enforcing. ■

*Corollary 1:* There always exist PP-enforcing insertion functions when the AIS is not empty; or, the system is PP-enforceable if and only if it is privately enforceable.

Full steps for synthesizing greedy PP-enforcing insertion functions are summarized in Algorithm 3.

We briefly discuss the computational complexity of Algorithm 3. Recall that the time complexity and space complexity for building AIS are  $O(|X_{\mathcal{E}}|^3)$  and  $O((1 + |E_o|)|X_{\mathcal{E}}|^2)$ , respectively, [24], [30]. To build  $\mathcal{E}_d^{\text{PP}}$ , Algorithm 1 needs a breadth-first search on the AIS, which requires  $O(|X_{\mathcal{E}}|^2)$  time complexity. Also, a breadth-first search on the

AIS is performed in Algorithm 2 for synthesizing  $f_{I,\text{greedy}}$ . In all, the computational complexity of Algorithm 3 is  $O(|X_{\mathcal{E}}|^3)$ .

*Remark 1:* We would like to point that Algorithm 3 can also be used to synthesize PP-enforcing insertion functions for infinite-step opacity and  $K$ -step opacity<sup>4</sup> in [10] and [11] by replacing  $\mathcal{E}_d$  in line 1 in Algorithm 3 with *desired infinite-step estimator*  $\mathcal{E}_d^{\text{inf}}$  and *desired  $K$ -step estimator*  $\mathcal{E}_d^K$  obtained in our work [30], respectively.

## V. CONCLUSION

This article investigates opacity enforcement when the insertion mechanism is also publicly known. First, the notion of PP-enforceability has been revised for accuracy, and an improved GM-criterion is proposed for synthesizing PP-enforcing insertion functions. We have solved the soundness problem by showing that there always exist PP-enforcing insertion functions when the AIS is not empty. A necessary and sufficient condition is derived for PP-enforceability of the system. It would be interesting to extend the enforcement method in this article to other system dynamics, such as cyber-physical systems [32], [33].

## APPENDIX

*Material for the Proof of Proposition 7*

Suppose that there exists a string set  $A \subseteq L_{\mathcal{E}}^1$  such that  $\forall s \in A$ , we have  $M_p(s) \subseteq A$ . Then, for  $A$ , we have that  $\overline{A} \setminus A \subseteq L_{\mathcal{E}}^2$  and  $\forall st \in L_{\mathcal{E}}^1$ , if  $s \in A$ , then  $st \in A$ . We define  $A_L = \{s \in \overline{A} \setminus A : \exists e \in E_o \text{ s.t. } se \in A\}$  and  $A_F = \{se \in A : e \in E_o \forall s \in A_L\}$ .  $A_F$  is the set of strings  $s$  in  $A$  such that any strict-prefix of  $s$  does not belong in  $A$ . Clearly,  $A_L \cap A = \emptyset$ ,  $A_L \subseteq \overline{A} \setminus A \subseteq L_{\mathcal{E}}^2$  and  $A_F \subseteq A \subseteq L_{\mathcal{E}}^1$ . Any string  $s \in A$  can be uniquely partitioned as  $s = s'es''$  such that  $s' \in A_L, e \in E_o, s'e \in A_F$ ; then, the *longest-safe* prefix of  $s$ , denoted by  $s^{\text{LP}}$ , is  $s'$ , i.e.,  $s^{\text{LP}} = s' \in A_L$  and  $s^{\text{LP}}e \in A_F$ .  $A_L$  is a collection set that contains all the longest-safe prefixes of strings in  $A$ .

Suppose that  $\overline{A}$  generated by a finite-state deterministic subautomaton  $H$  of estimator  $\mathcal{E}$ , denoted  $H = (Q, E_o, \delta, q_0)$  with  $Q \subseteq X_{\mathcal{E}}$  and  $q_0 = x_{\mathcal{E},0}$ . Define a state-event pair set  $\Theta$  as  $\Theta = \{(\delta(q_0, s), e) \in Q \times E_o : s \in A_L, e \in E_o, se \in A_F\}$ . Suppose that there are  $m$  state-event pairs in  $\Theta$ , then we have  $m = |\Theta| < |X_{\mathcal{E}}| \times |E_o|$ . For any string  $s$  in  $A$ , we can find a unique prefix  $s'e \preceq s$  and a unique state-event pair  $(q', e') \in \Theta$  such that  $s' = s^{\text{LP}} \in A_L, s'e \in A_F, \delta(q_0, s') = \delta(q_0, s^{\text{LP}}) = q'$  and  $e = e'$ . Or, we say that any string in  $A$  must “pass through” a state-event pair in  $\Theta$ .

To show the nonexistence of  $A$ , we first pick up a string  $s_0 \in A$ . Then, according to Proposition 5, we can further find a sequence of strings  $s_i \in A, i = 1, 2, \dots, m$  such that  $s_i \in M_p(s_{i-1})|s_{i-1}^{\text{LP}}$ . Clearly, for such string sequences  $s_i, i = 0, 1, \dots, m$ , we also have  $|s_i| > |s_{i-1}|, s_{i-1} \in s_i$  and  $s_{i-1}^{\text{LP}} \preceq s_i^{\text{LP}}$ . Since there are at most  $m$  pairs in  $\Theta$ , then there must exist two indices  $k, l \in [0, m]$  with  $k < l$  and a state-event pair  $(q', e') \in \Theta$  such that  $s_k, s_l$  pass through the same pair  $(q', e')$ , i.e.,  $\delta(q_0, s_k^{\text{LP}}) = \delta(q_0, s_l^{\text{LP}}) = q', s_k^{\text{LP}}e' \preceq s_k$  and  $s_l^{\text{LP}}e' \preceq s_l$ . Since  $s_k^{\text{LP}} \preceq s_l^{\text{LP}}$  and  $s_i \in M_p(s_{i-1})|s_{i-1}^{\text{LP}}, i \in [k, l]$ , we have that  $s_k^{\text{LP}} = s_l^{\text{LP}}$  or  $\exists s_I \in E_o^* \text{ s.t. } s_k^{\text{LP}}s_I = s_l^{\text{LP}}$  and  $\delta(q', s_I) = q'$  (i.e.,  $s_I$  is a string that is obtained from a circle path that starts and ends at the same state  $q'$ ). However, both above situations imply that  $s_k^{\text{LP}}e' \in L_{\mathcal{E}}^2$ . Clearly, if  $s_k^{\text{LP}} = s_l^{\text{LP}}$ , then we have that  $s_k^{\text{LP}}e' = s_l^{\text{LP}}e'$  and  $s_k^{\text{LP}}e' \in L_{\mathcal{E}}^2$ ; if  $s_k^{\text{LP}}s_I = s_l^{\text{LP}}$  and  $\delta(q', s_I) = q'$  (i.e.,  $s_l = s_l^{\text{LP}}e'(s_I \setminus s_l^{\text{LP}}e')$ ), which means that based on  $s_l$  and  $s_k$ , we can simply delete the string  $s_I$  in  $s_l$  and obtain a newly modified string  $s' = s_k^{\text{LP}}e'(s_I \setminus s_l^{\text{LP}}e')$ ;

<sup>4</sup>More specifically, the notions of infinite-step opacity and  $K$ -step opacity in [10], [11] are also termed as weak infinite-step opacity and weak  $K$ -step opacity [21], respectively; strong infinite-step opacity and strong  $K$ -step opacity were introduced in [15] and [31] for a higher level of confidentiality, respectively.



this means that  $s_k^{LP} e' \in L_{\mathcal{G}}^2$ . Furthermore, recall that  $s_k^{LP} e' \in A_F \subseteq L_{\mathcal{G}}^1$ , which is a contradiction. Therefore, such subset  $A$  does not exist.

## REFERENCES

- [1] R. Focardi and R. Gorrieri, "A taxonomy of trace-based security properties for CCS," in *Proc. Comput. Secur. Found. Workshop VII*, 1994, pp. 126–136.
- [2] S. Schneider and A. Sidiropoulos, "CSP and anonymity," in *Eur. Symp. Res. Comput. Secur.*, 1996, pp. 198–218.
- [3] R. Alur, P. Černý, and S. Zdancewic, "Preserving secrecy under refinement," in *Proc. Int. Colloq. Automata, Lang., Program.*, 2006, pp. 107–118.
- [4] C. N. Hadjicostis, "Estimation and inference in discrete event systems," in *Series Communications and Control Engineering*, Berlin, Germany: Springer, 2020.
- [5] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems.*, 3rd ed., Berlin, Germany: Springer, 2021.
- [6] A. Saboori and C. N. Hadjicostis, "Notions of security and opacity in discrete event systems," in *Proc. IEEE 46th Conf. Decis. Control*, 2007, pp. 5056–5061.
- [7] F. Lin, "Opacity of discrete event systems and its applications," *Automatica*, vol. 47, no. 3, pp. 496–503, 2011.
- [8] A. Saboori and C. N. Hadjicostis, "Verification of initial-state opacity in security applications of discrete event systems," *Inf. Sci.*, vol. 246, pp. 115–132, 2013.
- [9] Y.-C. Wu and S. Lafortune, "Comparative analysis of related notions of opacity in centralized and coordinated architectures," *Discrete Event Dynamic Syst.*, vol. 23, no. 3, pp. 307–339, 2013.
- [10] X. Yin and S. Lafortune, "A new approach for the verification of infinite-step and  $K$ -step opacity using two-way observers," *Automatica*, vol. 80, pp. 162–171, 2017.
- [11] A. Saboori and C. N. Hadjicostis, "Verification of infinite-step opacity and complexity considerations," *IEEE Trans. Autom. Control*, vol. 57, no. 5, pp. 1265–1269, May 2012.
- [12] Y. Ji, X. Yin, and S. Lafortune, "Optimal supervisory control with mean payoff objectives and under partial observation," *Automatica*, vol. 123, 2021, Art. no. 109359.
- [13] A. Saboori and C. N. Hadjicostis, "Opacity-enforcing supervisory strategies via state estimator constructions," *IEEE Trans. Autom. Control*, vol. 57, no. 5, pp. 1155–1165, May 2012.
- [14] Y. Xie, X. Yin, and S. Li, "Opacity enforcing supervisory control using non-deterministic supervisors," *IEEE Trans. Autom. Control*, vol. 67, no. 12, pp. 6567–6582, Dec. 2022.
- [15] Z. Ma, X. Yin, and Z. Li, "Verification and enforcement of strong infinite- and  $k$ -step opacity using state recognizers," *Automatica*, vol. 133, 2021, Art. no. 109838.
- [16] F. Cassez, J. Dubreil, and H. Marchand, "Synthesis of opaque systems with static and dynamic masks," *Formal Methods Syst. Des.*, vol. 40, no. 1, pp. 88–115, 2012.
- [17] X. Yin and S. Li, "Synthesis of dynamic masks for infinite-step opacity," *IEEE Trans. Autom. Control*, vol. 65, no. 4, pp. 1429–1441, 2020.
- [18] J. Balun and T. Masopust, "Comparing the notions of opacity for discrete-event systems," *Discrete Event Dyn. Syst.*, vol. 31, pp. 553–582, 2021.
- [19] Y. Tong, Z. Li, C. Seatzu, and A. Giua, "Verification of state-based opacity using Petri nets," *IEEE Trans. Autom. Control*, vol. 62, no. 6, pp. 2823–2837, Jun. 2017.
- [20] X. Han, K. Zhang, J. Zhang, Z. Li, and Z. Chen, "Strong current-state and initial-state opacity of discrete-event systems," *Automatica*, vol. 148, 2023, Art. no. 110756.
- [21] R. Jacob, J. Lesage, and J. Faure, "Overview of discrete event systems opacity: Models, validation, and quantification," *Annu. Rev. Control*, vol. 41, pp. 135–146, 2016.
- [22] Y.-C. Wu and S. Lafortune, "Synthesis of insertion functions for enforcement of opacity security properties," *Automatica*, vol. 50, no. 5, pp. 1336–1348, 2014.
- [23] Y.-C. Wu, V. Raman, B. C. Rawlings, S. Lafortune, and S. A. Seshia, "Synthesis of obfuscation policies to ensure privacy and utility," *J. Automated Reasoning*, vol. 60, no. 1, pp. 107–131, 2018.
- [24] Y. Ji, Y.-C. Wu, and S. Lafortune, "Enforcement of opacity by public and private insertion functions," *Automatica*, vol. 93, pp. 369–378, 2018.
- [25] Y. Ji, X. Yin, and S. Lafortune, "Opacity enforcement using nondeterministic publicly-known edit functions," *IEEE Trans. Autom. Control*, vol. 64, no. 10, pp. 4369–4376, Oct. 2019.
- [26] R. Liu, J. Lu, and C. N. Hadjicostis, "Opacity enforcement via attribute-based edit functions in the presence of an intended receiver," *IEEE Trans. Autom. Control*, early access, Nov. 8, 2022, doi: [10.1109/TAC.2022.3220557](https://doi.org/10.1109/TAC.2022.3220557).
- [27] X. Li, C. N. Hadjicostis, and Z. Li, "Extended insertion functions for opacity enforcement," *IEEE Trans. Autom. Control*, vol. 67, no. 10, pp. 5289–5303, Oct. 2022.
- [28] Y.-C. Wu and S. Lafortune, "Synthesis of opacity-enforcing insertion functions that can be publicly known," in *Proc. IEEE 54th Conf. Decis. Control*, 2015, pp. 3506–3513.
- [29] A. Wintenberg, M. Blischke, S. Lafortune, and N. Ozay, "Enforcement of  $K$ -step opacity with edit functions," in *Proc. IEEE 60th Conf. Decis. Control*, 2021, pp. 331–338.
- [30] R. Liu and J. Lu, "Enforcement for infinite-step opacity and  $K$ -step opacity via insertion mechanism," *Automatica*, vol. 140, 2022, Art. no. 110212.
- [31] Y. Falcone and H. Marchand, "Enforcement and validation (at runtime) of various notions of opacity," *Discrete Event Dyn. Syst.*, vol. 25, no. 4, pp. 531–570, 2015.
- [32] L. An and G. Yang, "Opacity enforcement for confidential robust control in linear cyber-physical systems," *IEEE Trans. Autom. Control*, vol. 65, no. 3, pp. 1234–1241, Mar. 2020.
- [33] X. Yin, M. Zamani, and S. Liu, "On approximate opacity of cyber-physical systems," *IEEE Trans. Autom. Control*, vol. 66, no. 4, pp. 1630–1645, Apr. 2021.