Opacity Enforcement Using Nondeterministic Publicly Known Edit Functions

Yiding Ji, *Student Member, IEEE*, Xiang Yin, *Member, IEEE*, and Stéphane Lafortune, *Fellow, IEEE*

***Abstract*—This note investigates enforcement of opacity by nondeterministic edit functions. The edit functions alter the system's output by inserting fictitious events or erasing observed events. The edit decisions are randomly made while not known by the outside environment *a priori*. There is an intruder characterized as a passive observer with malicious goals to infer the secrets of the system. We require that opacity be enforced when the intruder may or may not know the implementation of edit functions. This requirement is termed as private and public safety. We also restrict the operation of edit functions by defining edit constraints. Then, the opacity enforcement problem is transformed to a three-player game among the edit function, the environment, and a dummy player, which helps to determine edit decisions. A game structure called the all edit structure (AES) is introduced to characterize the interaction among those players. It embeds all privately safe edit functions and may also embed publicly safe edit functions. Based on the AES, we present an algorithm that provably synthesizes nondeterministic edit functions that satisfy both private safety and public safety.**

***Index Terms*—Discrete event systems (DES), edit function, nondeterminism, opacity enforcement, privacy.**

I. Introduction

Opacity is an information-flow-based security property that characterizes whether a system can defend its secrets from being inferred by an outside intruder with malicious intentions. The external intruder is often modeled as an observer that knows the structure of the system and attempts to infer secrets of the system by passively observing the output of the system. A system is called opaque if the intruder fails to determine the secrets unambiguously from its observations.

Opacity has been extensively studied in computer security community, starting with [1], and in discrete event systems (DES). In finite-state automaton models, various opacity notions have been studied to capture different privacy requirements, e.g., language-based opacity [13], current-state opacity (CSO) [15], initial-state opacity [16], K-step opacity, and infinite-step opacity [22]. Opacity has also been extended to other types of system models, including infinite-state systems [5], Petri nets [18], modular systems [14], and timed systems [3]. Recently, opacity has also been evaluated quantitatively in stochastic settings, e.g., [6] and [12]. The readers are referred to the survey paper [8] for a comprehensive review of results on opacity in DES.

Opacity may not always hold so that the problem of opacity enforcement arises, which has been investigated under various mechanisms. One common approach is supervisory control [17], [21], where a supervisor disables some behaviors before the disclosure of secrets. While [23] also lies in this category and solves the problem from the complementary perspective of maximal information release. Another framework is sensor activation [4], where the observation of events are dynamically changed while the system's operation is not interrupted. Alternatively, Falcone and Marchand [7] propose a runtime technique.

In contrast to the aforementioned approaches, Wu and Lafortune [19] introduce insertion functions as a new method, which insert fictitious events into the system's output to obfuscate the intruder. The insertion functions serve as an interface between the system's output and the intruder's observation. After that, Ji *et al.* [10] investigate opacity enforcement under the assumption that the intruder may or may not know the implementation of the insertion functions. To capture this situation, two concepts of private safety and public safety are defined and studied for evaluating the performance of insertion functions. Furthermore, Ji *et al.* [11] discuss opacity enforcement by insertion functions under quantitative constraints. Wu *et al.* [20] proceed to extend insertion functions to edit functions, which modify the system's output by inserting, erasing, or replacing events. All these works enforce opacity in a deterministic setting, i.e., any string is mapped to a unique string.

In this note, we assume that the edit function's implementation is known to the intruder and discuss how to defend secrets under such an adversary. We improve [9], [10], [19], and [20] by considering opacity enforcement using nondeterministic (ND) edit functions, whose outcome is randomly chosen from a precalculated set and the intruder does not know the result *a priori*. Both private safety and public safety are defined for edit functions to characterize their performance. Although ND edit functions seem to release more information to the intruder by allowing more potential outcomes, they essentially provide the system more plausible denial of secret disclosure, which contributes to opacity enforcement. It is shown that an ND edit function may still achieve private and public safety even when its deterministic counterpart fails to do so. To the best of our knowledge, this note is the first to consider nondeterminism of the defender in opacity enforcement. We introduce a three-player game structure termed *all edit structure (AES)* to embed edit functions. An algorithm is developed to synthesize privately and publicly safe ND edit functions based on the AES.[1]

The remaining sections are organized as follows. Section II presents the system model. Section III formally introduces the notions of ND edit functions, private safety, and public safety. Section IV defines the three-player observer (TPO), discusses its properties and introduces edit constraints. Section V defines a special TPO called AES and presents its construction algorithm. Section VI develops an algorithm for syn-

Manuscript received October 4, 2018; revised January 17, 2019; accepted January 27, 2019. Date of publication February 5, 2019; date of current version September 25, 2019. This work was supported in part by the National Science Foundation under Grant CCF-1138860 (Expeditions in Computing Project ExCAPE: Expeditions in Computer Augmented Program Engineering) and Grant CNS-1421122, in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and in part by the National Natural Science Foundation of China under Grant 61803259 and Grant 61833012. Recommended by Associate Editor K. Cai. *(Corresponding author: Yiding Ji.)*

Y. Ji and S. Lafortune are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48105 USA (e-mail: jiyiding@umich.edu; stephane@umich.edu).

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

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAC.2019.2897553

[1] A preliminary version of a subset of the results in this note, for deterministic edit functions only, appears in [9].

thesizing ND publicly and privately safe edit functions based on the reachability tree of the AES. Finally, Section VII concludes this note.

II. System Model

We consider opacity in the framework of DES modeled as deterministic finite-state automata [2]

$$G = (X, E, f, x_0)$$

where X is the finite set of states, E is the finite set of events, $f : X \times E \to X$ is the partial state transition function, and $x_0 \in X$ is the initial state. Specifically, we denote by $X_S \subset X$ the set of *secret states*. The transition function is extended to domain $X \times E^*$ in the standard manner [2]. Given two strings s, u, we denote by $s \preceq u$ if s is a prefix u and $t \in s$ if t is a substring of s. The language generated by G is defined as $\mathcal{L}(G) = \{s \in E^* : f(x_0, s)!\}$ where ! means "is defined."

For simplicity, we write $x \xrightarrow{e} x'$, if $x' = f(x, e)$ for $x, x' \in X$ and $e \in E$. Given system G, a *run* is a sequence of alternating states and events $x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} x_n$, where $\forall i \leq n, x_i \in X$ and $e_i \in E$. A run contains a *cycle* if $\exists 1 \leq i < j \leq n$, s.t. $x_i = x_j$.

The system is partially observed with the event set E 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. Given a string $t \in E^*$, its natural projection $P : E^* \to E_o^*$ is recursively defined as $P(t) = P(t'e) = P(t')P(e)$, where $t' \in E^*$ and $e \in E$. The projection of an event is $P(e) = e$ if $e \in E_o$ and $P(e) = \epsilon$ if $e \in E_{uo} \cup \{\epsilon\}$, where ϵ is the empty string. Then, by the standard technique in [2], the *observer* of G is defined as: $\text{Obs}(G) = (X_{\text{obs}}, E_o, \delta, x_{\text{obs},0})$, where $X_{\text{obs}} \subseteq 2^X$ is the state space, E_o is the set of observable events, $\delta : X_{\text{obs}} \times E_o \to X_{\text{obs}}$ is the transition function, and $x_{\text{obs},0} \in X_{\text{obs}}$ is the initial state. An observer state can be viewed as an *estimate* of the system's current states. Therefore, the observer is often called "state estimator" in the literature, e.g., [19].

III. Edit Functions and Opacity Notions

In this section, we formally define *ND* edit functions and discuss the edit mechanism. We also define *private safety* and *public safety* to further characterize how the edit function defends the secrets of the system against intruders with different knowledge.

A. Edit Mechanism

We first review the concept of deterministic edit function in [9]: $f_e : E_o^* \times E_o \to E_o^* E_o^\epsilon$, where $E_o^\epsilon = E_o \cup \{\epsilon\}$. Given $s \in P[\mathscr{L}(G)]$, $e_o \in E_o$, $f_e(s, e_o) = s_I e_o$ if s_I is inserted before e_o; $f_e(s, e_o) = \epsilon$ if e_o is erased; $f_e(s, e_o) = s_I$ if s_I is inserted and e_o is erased.

By definition, the outcome of a deterministic edit function is unique. Then, we extend it and define an *ND edit function*: $f_{\text{ne}} : E_o^* \times E_o \to 2^{E_o^* E_o^\epsilon}$ that outputs a string nondeterministically from a set of potential outcomes. Its output is based on the past observed string and the current observed event. Given an observable string $s \in P[\mathscr{L}(G)]$ and an observable event $e_o \in E_o$, a potential outcome of an ND edit function may be $s_I e_o$ if s_I is inserted before e_o or ϵ if e_o is erased or s_I if s_I is inserted and e_o is erased. In contrast to deterministic edit functions in [9], the outcome is not precalculated and is chosen randomly when the ND edit function is implemented. Notice that s_I may be ϵ so that nothing is inserted. The outcome of such a function is not known by the intruder before it is observed. With a slight abuse of notation, we also define a string based ND edit function f_{ne} recursively as: $f_{\text{ne}}(\epsilon) = \{\epsilon\}$ and $f_{\text{ne}}(se_o) = \{l_p l_s \in E_o^* : l_p \in f_{\text{ne}}(s), l_s \in f_{\text{ne}}(s, e_o)\}$.

An edit function is an interface between the system's output and the outside world, which includes the intruder eavesdropping on the system. The edit function works as follows: upon observing a string, it makes a decision to insert fictitious events before the last observed event or to erase the last observed event; then, the edited string is emitted as the actual output. We assume that all observable events E_o are allowed to be inserted or erased, and the intruder cannot distinguish between an inserted event and its genuine counterpart. We define $E_o^r = \{e_o \to \epsilon : e_o \in E_o\}$ to be the set of "event erasure" events. In this note, if we concatenate an "event erasure" event $e_o \to \epsilon$ with the observable event e_o, the result is simply ϵ.

Given an ND edit function f_{ne}, the intruder infers secrets from its *current state estimate* $\mathcal{E}_{f_{\text{ne}}} : P[\mathcal{L}(G)] \to 2^{X_{\text{obs}}}$ and $\mathcal{E}_{f_{\text{ne}}}(s) = \{x_{\text{obs}} \in X_{\text{obs}} : \exists t \in f_{\text{ne}}(s), \text{ s.t. } x_{\text{obs}} = \delta(x_{\text{obs},0}, t)\}$. Since f_{ne} is ND, $\mathcal{E}_{f_{\text{ne}}}(s)$ is generally a set of states in X_{obs}.

B. Private Safety and Public Safety

In this section, we first review the well-studied concept of *CSO* and then derive two concepts from it.

Definition 1 (Current-state opacity): System G is CSO w.r.t. projection P and secret states X_S if $\forall t \in L_S := \{t \in \mathcal{L}(G) : f(x_0, t) \in X_S\}$, $\exists t' \in L_{\text{NS}} := \{t \in \mathcal{L}(G) : f(x_0, t) \in (X \backslash X_S)\}$ s.t. $P(t) = P(t')$.

A system is current-state opaque if for every string reaching a secret state, there exists another string reaching a nonsecret state and both strings share the same projection. CSO can be verified by building the observer and checking whether any observer state contains solely secret states. If CSO is violated, an edit function may be used to enforce opacity, which is the problem studied in this note.

Based on CSO, we define the *safe language* [19] as: $L_{\text{safe}} = P[\mathcal{L}(G)] \backslash \{[P[\mathcal{L}(G)] \backslash P(L_{\text{NS}})] E_o^*\}$, which is prefix closed, whereas the *unsafe language* is $L_{\text{unsafe}} = P[\mathcal{L}(G)] \backslash L_{\text{safe}}$. Intuitively, we view all observable continuations of $P[\mathcal{L}(G)] \backslash P(L_{\text{NS}})$ as "unsafe." If we delete all states violating CSO from $\text{Obs}(G)$, i.e., all observer states that solely contain secret states, and then take the accessible part, the resulting automaton just generates L_{safe}. We call it *desired observer*: $\text{Obs}_d(G) = (X_{\text{obs}d}, E_o, \delta_d, x_{\text{obs}d,0})$, see [10] and [19] for more details.

Inspired by *private safety* and *public safety* of insertion functions in [10], we redefine those two concepts for ND edit functions and call them *ND private safety* and *ND public safety*, respectively.

Definition 2 (ND private safety): Consider system G with P, L_{safe}, and $Obs_d(G)$, an ND edit function f_{ne} is privately safe, if $\forall s \in P[\mathscr{L}(G)]$, $f_{ne}(s) \subseteq L_{\text{safe}}$.

If f_{ne} is privately safe, we denote it by $f_{\text{ne}} \vDash \varphi_{\text{ndpri}}$, where φ_{ndpri} stands for ND private safety. ND private safety is based on the assumption that the intruder does not know about the implementation of edit functions. Thus, as long as for a given string s and an edit function f_{ne}, every element in $f_{\text{ne}}(s)$ is also in L_{safe}, then, the intruder's state estimate would never reveal the secrets of the system.

Definition 3 (ND public safety): Consider a system G, L_{safe}, and L_{unsafe}, an ND edit function f_{ne} is publicly safe, if $\forall s \in L_{\text{unsafe}}, \forall \tilde{s} \in f_{ne}(s)$, $\exists t \in L_{\text{safe}}$, s.t. $\tilde{s} \in f_{ne}(t)$.

If f_{ne} is publicly safe, we denote it by $f_{\text{ne}} \vDash \varphi_{\text{ndpub}}$, where φ_{ndpub} stands for ND public safety. ND public safety is based on the assumption that the implementation of edit functions is known to the intruder. A sophisticated intruder may learn the implementation of the edit function and potentially does some reverse engineering to infer the source of the edited string. Thus, for ND public safety, we require that no matter how an unsafe string is edited, it should share the same edited behavior with some safe string. As the intruder does not know how a string is edited before it makes an observation, ND public safety and ND private safety guarantee that the system's secrets are never disclosed. A ND edit function f_{ne} is *ND-public-private enforcing* (ND-PP-enforcing), denoted by $f_{\text{ne}} \vDash \varphi_{\text{ndpp}}$, if $f_{\text{ne}} \vDash \varphi_{\text{ndpri}}$ and $f_{\text{ne}} \vDash \varphi_{\text{ndpub}}$. In this note, we require that an edit function should be able to map every string in $P[\mathcal{L}(G)]$ to some strings and we term this property as *admissibility*.

IV. Three-Player Observer

In this section, we propose the *TPO*, which is a three-player game structure that provides a systematic way of embedding edit functions and evaluating their performance. Then, we discuss some properties of the TPO and define *edit constraints*.

The TPO is an *information-state-based* structure, whose *current* state contains enough information for analysis of opacity enforcement and no *future* information is necessary. We denote the set of *information states* as I. The formal definition is as follows.

Definition 4 (Three-player observer): Given a system G, its observer $Obs(G)$ and desired observer $Obs_d(G)$, let $I \subseteq X_{obsd} \times X_{obs}$ be the set of information states. A TPO is the tuple $T = (Q_Y, Q_Z, Q_W, E_o, E_o^r, \Theta, f_{yz}, f_{zz}, f_{zw}^{in}, f_{zw}^{er}, f_{wy}^{in}, f_{wy}^{er}, y_0)$, where we have the following.

1) $Q_Y \subseteq I$ is the set of information states.
2) $Q_Z \subseteq I \times E_o$ is the set of information states augmented with observable events. Let $I(z)$ and $E(z)$ denote the information state component and observable event component of $z \in Q_Z$, respectively, so that $z = (I(z), E(z))$.
3) $Q_W \subseteq I \times (E_o \cup E_o^r)$ is the set of information states augmented with observable events or event erasure events. Let $I(w)$ and $A(w)$ denote the information state component and edit action component of $w \in Q_W$, respectively, so that $w = (I(w), A(w))$.
4) $E_o \subseteq E$ is the set of observable events.
5) E_o^r is the set of "event erasure" events.
6) $\Theta \subseteq E_o \cup \{\epsilon\} \cup E_o^r$ is the set of edit decisions at Q_Z-states.
7) $f_{yz} : Q_Y \times E_o \to Q_Z$ is the transition function from Q_Y-states to Q_Z-states. For $y = (x_d, x_f) \in Q_Y$, $e_o \in E_o$, we have
$$f_{yz}(y, e_o) = z \Rightarrow [\delta(x_f, e_o)!] \wedge [I(z) = y] \wedge [E(z) = e_o].$$
8) $f_{zz} : Q_Z \times \Theta \to Q_Z$ is the transition function from Q_Z-states to Q_Z-states. For $z = ((x_d, x_f), e_o) \in Q_Z$, $\theta \in \Theta$, we have
$$f_{zz}(z, \theta) = z' \Rightarrow [\theta \in E_o] \wedge [I(z') = (x_d', x_f)] \wedge [x_d' = \delta_d(x_d, \theta)] \wedge [E(z') = e_o].$$
9) $f_{zw}^{in} : Q_Z \times \Theta \to Q_W$ is the ϵ-insertion transition from Q_Z-states to Q_W-states. For $z = ((x_d, x_f), e_o) \in Q_Z$, $\theta \in \Theta$ we have
$$f_{zw}^{in}(z, \theta) = w \Rightarrow [\theta = \epsilon] \wedge [I(w) = I(z)] \wedge [A(w) = e_o] \wedge [\delta_d(x_d, e_o)!] \wedge [\delta(x_f, e_o)!].$$
10) $f_{zw}^{er} : Q_Z \times \Theta \to Q_W$ is the event erasure transition from Q_Z-states to Q_W-states. For $z = ((x_d, x_f), e_o) \in Q_Z$, $\theta \in \Theta$, we have
$$f_{zw}^{er}(z, \theta) = w \Rightarrow [\theta = e_o \to \epsilon] \wedge [I(w) = I(z)] \wedge [A(w) = e_o \to \epsilon] \wedge [\delta(x_f, e_o)!].$$
11) $f_{wy}^{in} : Q_W \times E_o \to Q_Y$ is the transition function from Q_W-states whose edit action component is in E_o to Q_Y-states. For $w = ((x_d, x_f), e_o) \in Q_W$, we have
$$f_{wy}^{in}(w, e_o) = y \Rightarrow [y = (x_d', x_f')] \wedge [x_d' = \delta_d(x_d, e_o)] \wedge [x_f' = \delta(x_f, e_o)].$$
12) $f_{wy}^{er} : Q_W \times E_o \to Q_Y$ is the transition function from Q_W-states whose edit action component is in E_o^r to Q_Y-states. For $w = ((x_d, x_f), e_o \to \epsilon) \in Q_W$, we have
$$f_{wy}^{er}(w, e_o) = y \Rightarrow [y = (x_d, x_f')] \wedge [x_f' = \delta(x_f, e_o)].$$
13) $y_0 \in Q_Y$ is the initial Q_Y-state, where $y_0 = (x_{obsd,0}, x_{obs,0})$. $x_{obsd,0}$ and $x_{obs,0}$ are the initial states of $Obs_d(G)$ and $Obs(G)$, respectively.

The TPO is defined to describe the game among a "dummy" player, "edit function," and "system/environment." All three players have *complete information* in the sense that they know exactly the actions of each other at any moment of the game.

A Q_Y-state (Y-state) is an information state, from which the "dummy" player executes observable events. A Y-state contains both the intruder's estimate and the system's estimate. Actually, the events from Y-states do not really occur and they are the events to be observed by the edit function player. f_{yz} is defined only to help determine what edit decisions can be made by the edit function in the next step. That is why we call this player a dummy player.

A Q_Z-state (Z-state) is an information state augmented with the event executed by the dummy player, where the edit function makes decisions. If the edit function chooses to insert an event, a succeeding Z-state will be reached under an f_{zz} transition. If another event is inserted following the last inserted event, then another succeeding Z-state is reached until the edit function stops inserting. This corresponds to insertion of multiple events. If the edit function keeps inserting events, we can expect that a cycle of Z-states and f_{zz} transitions is formed in the TPO. When an event is inserted, only the intruder's estimate is updated, whereas the system's estimate remains the same, which is reflected in defining f_{zz}. This is consistent with the edit function's mechanism as the edit function serves as an interface to modify the intruder's observation but does not interfere with the system's operation. When the edit function decides to stop insertion or to erase the last observed event, the turn of the game is passed to the system/environment player by f_{zw}^{in} and f_{zw}^{er} transitions. We denote by $f_{zw} = f_{zw}^{\text{in}} \cup f_{zw}^{\text{er}}$, where f_{zw}^{in} stands for ϵ-insertion (termination of insertion) and f_{zw}^{er} stands for erasure of the observable event executed by the dummy player. We will use f_{zw} for simplicity in the following discussion if there is no confusion. There may be multiple transitions defined out of a Z-state, i.e., multiple edit decisions, and we let $\Theta(z)$ be the set of edit decisions defined at $z \in Q_Z$ in a TPO.

A Q_W-state (W-state) is an information state augmented with an observable event or an "event erasure" event, from which the system plays. If a W-state contains an observable event that means that the edit function player has inserted ϵ from its preceding Z-state. When that event is executed, it will be observed by the intruder. Thus, a f_{wy}^{in} transition leads to a Y-state, whose first- and second-state components are both updated. If a W-state contains an "event erasure" event that means that the edit function has decided to erase the observable event. So, when the event is executed, it will not be observed by the intruder. Hence, an f_{wy}^{er} transition leads to a Y-state, whose first-state component (intruder's estimate) is updated, whereas the second state component (system's estimate) remains unchanged. We just denote by $f_{wy} = f_{wy}^{\text{in}} \cup f_{wy}^{\text{er}}$ and will use f_{wy} when there is no confusion.

Given two TPOs T_1 and T_2, T_1 is a *subsystem* of T_2, denoted by $T_1 \sqsubseteq T_2$, if $Q_Y^{T_1} \subseteq Q_Y^{T_2}$, $Q_Z^{T_1} \subseteq Q_Z^{T_2}$, $Q_W^{T_1} \subseteq Q_W^{T_2}$, and $\forall y \in Q_Y^{T_1}, \forall z, z' \in Q_Z^{T_1}, \forall w \in Q_W^{T_1}, \forall e_o \in E_o, \forall \theta, \theta' \in \Theta$, we have: first, $f_{yz}^{T_1}(y, e_o) = z \Rightarrow f_{yz}^{T_2}(y, e_o) = z$; second, $f_{zz}^{T_1}(z, \theta) = z' \Rightarrow f_{zz}^{T_2}(z, \theta) = z'$; third, $f_{zw}^{T_1}(z, \theta') = w \Rightarrow f_{zw}^{T_2}(z, \theta') = w$; fourth, $f_{wy}^{T_1}(w, e_o) = y \Rightarrow f_{wy}^{T_2}(w, e_o) = y$.

A run in a TPO is of the form: $r = y_0 \xrightarrow{e_0} z_0^1 \xrightarrow{\theta_0^1} z_0^2 \xrightarrow{\theta_0^2} \cdots \xrightarrow{\theta_0^{m_0-1}} z_0^{m_0} \xrightarrow{\theta_0^{m_0}} w_0 \xrightarrow{e_0} y_1 \xrightarrow{e_1} z_1^1 \xrightarrow{\theta_1^1} z_1^2 \xrightarrow{\theta_1^2} \cdots z_1^{m_1} \xrightarrow{\theta_1^{m_1}} w_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_n} z_n^1 \xrightarrow{\theta_n^1} \cdots z_n^{m_n} \xrightarrow{\theta_n^{m_n}} w_n \xrightarrow{e_n} y_{n+1}$, where y_0 is the initial state of T, $e_i \in E_o$, $\theta_i^j \in \Theta(z_i^j)$, $\forall 0 \le i \le n$, $1 \le j \le m_i$, and $n \in \mathbb{N}$, $m_i \in \mathbb{N}^+$. It characterizes the information flow in a TPO and we denote the set of runs in a TPO T by Run(T). We also write $y_i \in r$ ($z_i \in r$ or $w_i \in r$) if y_i (z_i or w_i) is a state in r. A run corresponds to an unedited string and an edited string, then we have the following definitions.

Definition 5 (String generated by a run): Given a run $r = y_0 \xrightarrow{e_0} z_0^1 \xrightarrow{\theta_0^1} z_0^2 \xrightarrow{\theta_0^2} \cdots \xrightarrow{\theta_0^{m_0-1}} z_0^{m_0} \xrightarrow{\theta_0^{m_0}} w_0 \xrightarrow{e_0} y_1 \xrightarrow{e_1} z_1^1 \xrightarrow{\theta_1^1} z_1^2 \xrightarrow{\theta_1^2} \cdots z_1^{m_1} \xrightarrow{\theta_1^{m_1}} w_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_n} z_n^1 \xrightarrow{\theta_n^1} \cdots z_n^{m_n} \xrightarrow{\theta_n^{m_n}} w_n \xrightarrow{e_n} y_{n+1}$, the string generated by r is defined as: $l_g(r) = \theta_0^1\theta_0^2\cdots\theta_0^{m_0-1}\theta_0^{m_0} e_0\theta_1^1\cdots\theta_1^{m_1}e_1\cdots e_{n-1}\theta_n^1\cdots\theta_n^{m_n}e_n$, where $\forall i \le n$, $\theta_i^{m_i}e_i = \epsilon$ if $\theta_i^{m_i} = e_i \to \epsilon$.

Definition 6 (Edit projection): In a TPO T, given a run $r = y_0 \xrightarrow{e_0} z_0^1 \xrightarrow{\theta_0^1} z_0^2 \xrightarrow{\theta_0^2} \cdots \xrightarrow{\theta_0^{m_0-1}} z_0^{m_0} \xrightarrow{\theta_0^{m_0}} w_0 \xrightarrow{e_0} y_1 \xrightarrow{e_1} z_1^1 \xrightarrow{\theta_1^1} z_1^2 \xrightarrow{\theta_1^2} \cdots z_1^{m_1} \xrightarrow{\theta_1^{m_1}} w_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_n} z_n^1 \xrightarrow{\theta_n^1} \cdots z_n^{m_n} \xrightarrow{\theta_n^{m_n}} w_n \xrightarrow{e_n} y_{n+1}$, edit projection $P_e : Run(T) \to P[\mathcal{L}(G)]$ is defined such that $P_e(r) = e_0e_1\cdots e_n$.

That is, the edit projection projects away the edit decisions in a run and "recovers" the unedited string, whereas the generated string of a run is just the string after considering the edit decisions.

From a given TPO, we may extract an edit function from it and we define the *edit function embedded in a TPO*. With a slight abuse of notation, we write $f_{\text{ne}} \in T$ if f_{ne} is embedded in T.

Definition 7 (ND edit function embedded in TPO): Given a TPO T, ND edit function f_{ne} is embedded in T if $\forall s \in P[\mathcal{L}(G)]$, $\forall \tilde{s} \in f_{ne}(s)$, $\exists r \in Run(T)$, s.t. $P_e(r) = s$, and $l_g(r) = \tilde{s}$.

In a TPO, $y \in Q_Y$ is a *terminating state* if $\nexists e_o \in E_o$, s.t. $f_{yz}(y, e_o)!$ and $w \in Q_W$ is a *deadlocking* state if $\nexists e_o \in E_o$, s.t. $f_{wy}(w, e_o)!$. Also, $z \in Q_Z$ is a *deadlocking* state if $\nexists\theta \in \Theta$, s.t. $f_{zz}(z, \theta)!$ or $f_{zw}(z, \theta)!$. We call a TPO *complete* if: first, there are no deadlocking W or Z states; second, $\forall s \in P[\mathcal{L}(G)]$, $\exists r \in \text{Run}(T)$, s.t. $P_e(r) = s$. In a complete TPO, all embedded edit functions are admissible and they can always make a decision no matter what event occurs; also the events executed by the system cannot be blocked from happening. From now on, we will only consider complete TPOs. Notice that a complete TPO only terminates at Y-states, being consistent with the definition of run.

In practice, the edit functions may be constrained by the outside environment or the preference of the system's designer so that certain edit decisions may not be taken and some Y-states may not be preferred. Thus, we introduce *constraints on edit decisions* and *constraints on Y-states*, both in a generic form.

Definition 8 (Constraints on edit decisions): The constraint on edit decisions is a binary function $\phi_{dec} : \Theta \to \{0, 1\}$ and an edit decision $\theta \in \Theta$ satisfies the constraint if $\phi_{dec}(\theta) = 1$.

Definition 9 (Constraints on Y-states): The constraint on Y-states is a binary function $\phi_y : Q_Y \to \{0, 1\}$ and a Y-state $y \in Q_Y$ satisfies the constraint if $\phi_y(y) = 1$.

Both constraints are problem-dependent and will be specified when a problem is discussed. They will reduce the state space of the TPO and bring in deadlocking states. In the following section, we will define the "largest" TPO satisfying both constraints.

V. All Edit Structure

In this section, we define a complete TPO such that: $[\forall y \in Q_Y : \phi_y(y) = 1] \wedge [\forall\theta \in \Theta : \phi_{\text{dec}}(\theta) = 1]$ and T is "as large as possible." We call this structure the AES. The property of being as large as possible is as follows: if T_1 and T_2 are two TPOs satisfying edit constraints, then their union, in the graph merging sense, is also a TPO satisfying edit constraints. The union of T_1 and T_2 is defined as: first, $Q_Y^{T_1\cup T_2} = Q_Y^{T_1} \cup Q_Y^{T_2}$, $Q_Z^{T_1\cup T_2} = Q_Z^{T_1} \cup Q_Z^{T_2}$, and $Q_W^{T_1\cup T_2} = Q_W^{T_1} \cup Q_W^{T_2}$; second, $\forall y \in Q_Y^{T_1\cup T_2}$, $\forall z, z' \in Q_Z^{T_1\cup T_2}$, $\forall w \in Q_W^{T_1\cup T_2}$, $\forall\theta, \theta' \in \Theta$, and $\forall e_o \in E_o$, we have: $f_{yz}^{T_1\cup T_2}(y, e_o) = z \Leftrightarrow \exists i \in \{1, 2\} : f_{yz}^{T_i}(y, e_o) = z$, $f_{zz}^{T_1\cup T_2}(z, \theta') = z' \Leftrightarrow \exists i \in \{1, 2\} : f_{zz}^{T_i}(z, \theta') = z'$, $f_{zw}^{T_1\cup T_2}(z, \theta) = w \Leftrightarrow \exists i \in \{1, 2\} : f_{zw}^{T_i}(z, \theta) = w$ and $f_{wy}^{T_1\cup T_2}(w, e_o) = y \Leftrightarrow \exists i \in \{1, 2\} : f_{zw}^{T_i}(w, e_o) = y$.

Definition 10 (All edit structure): Given system G, edit constraints ϕ_{dec} and ϕ_y, the AES is the largest complete TPO: $AES = (Q_Y^A, Q_Z^A, Q_W^A, E_o, E_o^r, \Theta, f_{yz}^A, f_{zz}^A, f_{zw}^A, f_{wy}^A, y_0)$, where $\forall y \in Q_Y^A : \phi_y(y) = 1$ and $\forall\theta \in \Theta : \phi_{dec}(\theta) = 1$. The largest TPO is such that: for all TPO T satisfying the above-mentioned two conditions, $T \sqsubseteq AES$.

Algorithm I shows a general procedure for constructing the AES and it calls Algorithms II and III in its operation. In Algorithm II, we

Algorithm I: Construction of the AES.

```
Input   :$Obs(G), Obs_d(G), E_o^r, \phi_{dec}, \phi_y$
Output  :$AES$
1: $Q_Y^A = \{y_0\} = \{(x_{obsd,0}, x_{obs,0})\}, Q_Z^A = \emptyset, Q_W^A = \emptyset$;
2: $AES_{pre} = DoDFS(y_0, \phi_{dec}, \phi_y, Obs(G), Obs_d(G), E_o^r)$;
3: $AES = Prune(AES_{pre})$;
```

Algorithm II: DoDFS.

```
Input   : $y, \phi_{dec}, \phi_y, Obs(G), Obs_d(G), E_o^r$
Output  : $AES_{pre}$
1  for $e_o \in E_o$, s.t. $f_{yz}(y, e_o)!$ by Definition 4 do
2      $z = ((x_{obsd}, x_{obsf}), e_o) = f_{yz}(y, e_o)$;
3      add transition $y \xrightarrow{e_o} z$ to $f_{yz}^A$;
4      if $z \notin Q_Z^A$ then
5          $Q_Z^A = Q_Z^A \cup \{z\}$;
6          $\Theta(z) = \emptyset$;
7          $Z_{ext}(z) = \{z\}$;
8          $EXTEND-Z(Z_{ext}(z), \phi_{dec})$;
9          for $z' \in Z_{ext}(z)$ do
10             if $\exists\theta \in \Theta$, s.t. $f_{zw}(z', \theta)!$ by Definition 4 and
               $\phi_{dec}(\theta) = 1$ then
11                 $w = f_{zw}(z', \theta)$;
12                 $\Theta(z') = \Theta(z') \cup \{\theta\}$;
13                 add transition $z' \xrightarrow{\theta} w$ to $f_{zw}^A$;
14                 if $w \notin Q_W^A$ then
15                     $Q_W^A = Q_W^A \cup \{w\}$;
16                 for $e_o \in E_o$, s.t. $f_{wy}(w, e_o)!$ by Definition 4 do
17                     $y' = f_{wy}(w, e_o)$;
18                     if $\phi_y(y') = 1$ then
19                         add transition $w \xrightarrow{e_o} y'$ to $f_{wy}^A$ ;
20                         if $y' \notin Q_Y^A$ then
21                             $Q_Y^A = Q_Y^A \cup \{y'\}$;
22                             $DoDFS(y', \phi_{dec}, \phi_y, Obs(G), Obs_d(G), E_o^r)$;
   Procedure: $EXTEND-Z(Z_{ext}(z), \phi_{dec})$
23 while $\exists z \in Z_{ext}(z)$, $\exists\theta \in \Theta$, s.t. $f_{zz}(z, \theta)!$ by Definition 4 and
   $\phi_{dec}(\theta) = 1$ do
24     $z' = f_{zz}(z, \theta)$;
25     $\Theta(z) = \Theta(z) \cup \{\theta\}$;
26     add transition $z \xrightarrow{\theta} z'$ to $f_{zz}^A$;
27     if $z' \notin Q_Z^A$ then
28         $Q_Z^A = Q_Z^A \cup \{z'\}$;
29         $\Theta(z') = \emptyset$;
30         $Z_{ext}(z) = Z_{ext}(z) \cup \{z'\}$;
31         $EXTEND-Z(Z_{ext}(z), \phi_{dec})$;
```

start searching from $y_0 = (x_{\text{obsd},0}, x_{\text{obs},0})$ and expand the state space recursively by computing all possible successors of the current state. We terminate searching on a path when a Y-state violates the edit constraint, i.e., $\phi_y(y) = 0$ or an edit decision is not allowed by the constraints, i.e., $\phi_{\text{dec}}(\theta) = 0$. This is an iterative procedure that allows us to build the whole reachable state space. We also add transitions in this process.

Specifically, at a newly added Z-state, we need to determine feasible edit decisions. There may be consecutive Z-states between a Y-state and a W-state. Then, we search them in the procedure $EXTEND - Z$, which is also a depth-first search process. In $EXTEND - Z$, we add succeeding Z-states until no more f_{zz} transitions are defined and no more insertions are made. In this process, for each $z \in Q_Z^A$, we define $Z_{\text{ext}}(z)$ to be the set of Z-states that can be reached from z through f_{zz} transitions. We keep growing $Z_{\text{ext}}(z)$ until no more Z-states are added

Algorithm III: Prune.

```
Input  : A three-player observer
Output: A three-player observer without deadlocking states
1 while there exist deadlocking $W$-states or $Z$-states do
2     for deadlocking $W$-state $w$ do
3         remove $w$ from the structure
4     for deadlocking $Z$-state $z$ do
5         if there exist $Y$-state $y$ and $e_o \in E_o$, s.t. $z$ is reachable
          from $y$ through $e_o$ then
6             remove $y$ and $z$ from the structure;
7         else
8             remove $z$ from the structure;
9     take the accessible part of the structure;
```

Algorithm IV: Build Labeled Reachability Tree of the AES.

```
Input    : $AES$
Output   : $AES_t$
1 $Q_Y^{AT} = \{y_0\}$, $Q_Z^{AT} = Q_W^{AT} = \emptyset$;
2 $AES_t^{pre} = DoBFS(y_0, AES)$;
3 call Algorithm 3, $Prune(AES_t^{pre})$;
4 for $Y$-state $y$ in the remaining structure do
5     specify the run $r$ from $y_0$ to $y$ in the remaining structure;
6     use $(l(r), P_e(r))$ to label $y$;
7 return $AES_t$;
  Procedure: $DoBFS(q, AES)$
8 while there exists state $q$ in AES that has not been examined do
9     evaluate all transitions defined at $q$ in $AES$;
10    if no transition is defined at $q$ in $AES$ then
11        terminate searching on the current path from $q$;
12    else
13        for a transition defined at $q$ in $AES$ do
14            add state $q'$ reached by the transition as a new state
              in the tree $AES_t^{pre}$;
15            if $q'$ equals a state on the path from $y_0$ to $q$ then
16                stop searching from on the current path $q'$;
```

and no new f_{zz} transitions are defined at states in $Z_{\text{ext}}(z)$. Consecutive Z-states may form a cycle in the AES, which indicates that a loop is inserted by the edit function. Since the information state component of a Z-state comes from $2^X \times 2^X$ and its event component comes from E_o, both of which are finite sets, then only a finite number of Z-states is added in each iterate and $EXTEND - Z$ always terminates. Similarly, the information state components of Y-states and W-states also come from $2^X \times 2^X$, whereas the edit action components of W-states come from E_o or E_o^r. All of them are finite sets. Overall, only finite states will be added to AES_{pre} until some states or transitions violate the edit constraints. Thus, Algorithm II terminates after a finite number of steps and returns a finite structure.

We denote the output of Algorithm II by AES_{pre}, which may contain deadlocking states since edit constraints preclude transitions out of them or their succeeding states. We prune away deadlocking states as well as their predecessor states in Algorithm III in an iterative manner until the structure converges. If a state is deadlocking, then the edit decisions leading to it should not be considered for synthesizing edit functions. Thus, we also prune away its preceding states. This process is similar to calculating the supremal controllable sublanguage in nonblocking supervisory control under full observation [2], by viewing the deadlocking states as undesired marked states and f_{yz}^A and f_{wy}^A transitions as uncontrollable, whereas f_{zz}^A and f_{zw}^A transitions as controllable. Algorithm III also terminates after a finite number of steps when no more states are to be removed, then it returns the AES after it is called in Algorithm I. The following theorem

Algorithm V: Synthesize PP-enforcing Edit Functions.

```
Input  : $AES_t$
Output: Nondeterministic PP-enforcing edit function
1 for $l_i \in L_{leaf}^u$ do
2     collect $Q_{Y-leaf}^{AT1}(l_i)$, suppose $Q_{Y-leaf}^{AT1}(l_i)$ has $m_i$ elements;
3     for $j = 1 : m_i$ do
4         consider $y_j^1(l_i) = ((x_d, x_f), (t, l_i)) \in Q_{Y-leaf}^{AT1}(l_i)$;
5         if $\nexists y^2(l') = ((x_d', x_f'), (t', l')) \in Q_{Y-l}^{AT2}$, s.t. $t \preceq t'$ then
6             remove $y_j^1(l_i)$ from the $\text{AES}_t$
7 $AES_t^r = Prune(AES_t)$;
8 for $l_i \in L_{leaf}^u$ do
9     denote by $Q_{Y-re}^{AT1}$ ($Q_{Y-re}^{AT2}$) the $Y$-states in $AES_t^r$ with
      (without) unsafe string components, then define
      $S_{pp}^r(l_i) = \{((x_d, x_f), (t, l_i)) \in Q_{Y-leaf}^{AT1}(l_i) \cap Q_{Y-re}^{AT1} : \exists y^2 =$
      $((x_d', x_f'), (t', l')) \in Q_{Y-l}^{AT2} \cap Q_{Y-re}^{AT2}$, s.t. $t \preceq t'\}$;
10    if $S_{pp}^r(l_i) = \emptyset$ then
11        nondeterministic PP-enforcing edit functions do not
          exist, terminate the algorithm;
12 return the nondeterministic edit function embedded in $AES_t^r$;
```

reveals the correctness and completeness of the AES, namely, the AES embeds all ND privately safe edit functions satisfying the edit constraints.

Theorem 1: Given system G, an ND edit function f_{ne} is ND privately safe if and only if $f_{ne} \in AES$.

Proof: $(\Rightarrow)$ By contradiction. Suppose $f_{\text{ne}} \vDash \varphi_{\text{ndpri}}$ but $f_{\text{ne}} \notin \text{AES}$. Then, there should exist a TPO T such that $f_{\text{ne}} \in T$. This means that $\exists s \in P[\mathcal{L}(G)]$, $\exists r \in \text{Run}(T)$, s.t. $P_e(r) = s$, $l_g(r) \in f_{\text{ne}}(s)$ but $r \notin \text{Run}(\text{AES})$. Thus, there are some states or transitions in r that are not in the AES. However, this implies that the union of T and the AES is strictly larger than the AES, which contradicts with the definition that the AES is the largest TPO satisfying the edit constraints.

$(\Leftarrow)$ Suppose that $f_{\text{ne}} \in \text{AES}$, then $\forall s \in P[\mathcal{L}(G)]$, $\forall \tilde{s} \in f_{\text{ne}}(s)$, $\exists r \in \text{Run}(T)$, s.t. $P_e(r) = s \wedge l_g(r) = \tilde{s}$. Since $\forall y = (x_d, x_f) \in r$, $x_d \in X_{\text{obsd}}$, we know $f_{\text{ne}}(s) \subseteq \mathcal{L}(\text{Obs}_d(G)) = L_{\text{safe}}$ and f_{ne} is privately safe. ■

Remark 1: We briefly analyze the complexity of constructing the AES. First, we evaluate the complexity of Algorithm II. Here, we define $Q_Z^{ent} = \{z \in Q_Z^A : \exists y \in Q_Y^A, \exists e_o \in E_o \text{ s.t. } f_{yz}(y, e_o) = z\}$ as the Z-states that can be reached from certain Y-states by f_{yz} transitions. Given system G with $|X|$-states, its observer $Obs(G)$ has at most $|X_{obs}| = 2^{|X|}$ states. Since $Q_Y^A \subseteq X_{obsd} \times X_{obs}$, $|Q_Y^A| \leq |X_{obs}|^2$. Also, each Y-state can execute at most $|E_o|$ observable events in line 1, so $|Q_Z^{ent}| \leq |E_o||X_{obs}|^2$. In $DoDFS$, we apply procedure $EXTEND - Z$ at each Q_Z^{ent} in line 8 to determine edit choices step by step. This procedure creates at most $(|X_{obs}| - 1)$-states for each Q_Z^{ent}-state. Thus, $|Q_Z^A| \leq |E_o||X_{obs}|^2(|X_{obs}| - 1 + 1) = |E_o||X_{obs}|^3$. Furthermore, every Z-state may lead to a W-state by f_{zw} transition, so $|Q_W^A| \leq |E_o||X_{obs}|^3$. Thus, the state space complexity of AES_{pre} is $O(|X_{obs}|^3)$. The complexity of Algorithm III is quadratic in the size of AES_{pre} as one state is visited at most once in an iteration. Overall, the space complexity of constructing the AES is polynomial in terms of $|X_{obs}|$.

Remark 2: It can be shown by induction on the length of strings that if the AES is not empty, then all edit functions embedded in it are admissible. This is a consequence of the pruning process in Algorithm III and we omit the proof here. By the same argument, no admissible edit function exists if the AES is empty. Hence, we will rule out this situation in the remainder of the note.

Example 1: We show an AES. The observer of system G is depicted in Fig. 1. All events $\{a, b, c, d\}$ are observable and observer state 4 is solely composed of secret states from G. The desired observer $Obs_d(G)$ is simply without state 4 and we omit its figure here. To begin with, we follow the first two steps of Algorithm I and build

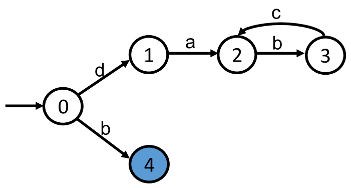

Fig. 1. Observer in Example 1.

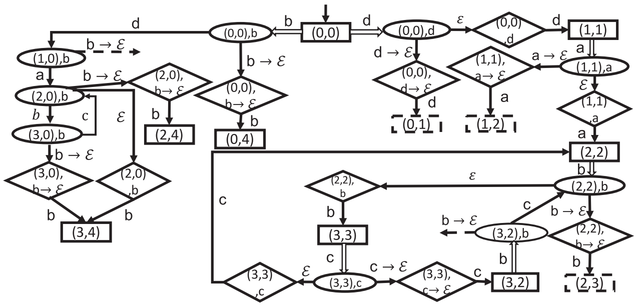

Fig. 2. AES$_{\text{pre}}$ in Example 1 (without dashed states and transitions).

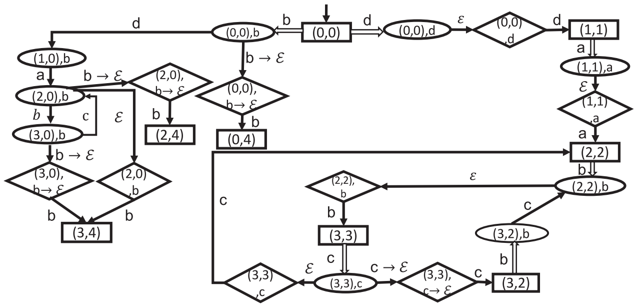

Fig. 3. AES in Example 1.

AES_{pre} in Fig. 2, where squared states, oval states, and diamond states stand for Y, Z, and W states, respectively.

The game is initialized at $y_0 = (0, 0)$, where the dummy player executes b and d since both events are defined at state 0 in $Obs(G)$. If b is executed, Z-state $((0, 0), b)$ is reached, where the edit function plays and there are two edit decisions. At $((0, 0), b)$, if the edit function chooses to erase b, then the system plays at W-state $((0, 0), b \to \epsilon)$; if the edit function inserts d, then Z-state $((1, 0), b)$ is reached since $\delta_d(0, d) = 1$. If a is also inserted after d is inserted, then another Z-state $((2, 0), b)$ is reached. Then, at Z-state $((2, 0), b)$, if the edit function decides to stop inserting, W-state $((2, 0), b)$ is reached. When the system plays, say, at $((0, 0), b \to \epsilon)$, b occurs and leads to Y-state (0, 4), since b is not observed by the intruder and the first-state component is not updated. When the system plays at $((2, 0), b)$, b occurs and leads to Y-state (3, 4) since $\delta_d(2, b) = 3$ and $\delta(0, b) = 4$. The whole structure is interpreted in a similar way.

In this example, the edit constraints prohibit the edit function from erasing b at $((1, 0), b)$ and $((3, 2), b)$, also $\phi_y((0, 1)) = \phi_y((1, 2)) = \phi_y((2, 3)) = 0$. We use dashed lines in Fig. 2 to indicate the transitions and states that violate edit constraints. Those transitions/states are not in AES$_{pre}$. In Fig. 2, there are some deadlocking W-states, such as $((0, 0), d \to \epsilon)$, $((1, 0), a \to \epsilon)$, and $((2, 2), b \to \epsilon)$ and no deadlocking Z-states exist. Then, we prune away those deadlocking states by Algorithm III and finally obtain the AES in Fig. 3.

Then, it is natural to ask when there exists an ND-PP-enforcing edit function in the given AES. The key point is *every unsafe string shares the same edited behavior with some safe string*. However, the state information in the AES is insufficient to verify this condition as a Y-state may appear in multiple runs and different strings may be edited to the same one by different edit decisions. Therefore, additional analysis is necessary, which is discussed in the following section.

VI. Synthesis of ND Privately Safe and Publicly Safe Edit Functions

In this section, we synthesize ND-PP-enforcing edit functions. From Theorem 1, any edit function embedded in the AES is ND privately safe so we only need to consider ND public safety. Unfortunately, we cannot only consider the state information in the AES for synthesis. Thus, we introduce the *reachability tree* of the AES, which is the "unfolded" AES with respect to unedited strings and edited strings. Then, we have access to strings before/after edit and develop a synthesis algorithm based on the tree.[2]

A. Reachability Tree of the AES

The reachability tree of the AES is denoted by AES$_t$ $= (Q_Y^{AT}, Q_Z^{AT}, Q_W^{AT}, E_o, E_o^r, \Theta, f_{yz}^{AT}, f_{zz}^{AT}, f_{zw}^{AT}, f_{wy}^{AT}, y_0)$ and constructed in Algorithm VI. It is built by unfolding the state space in a breadth-first search manner in line 2. The AES$_t$ is an acyclic structure by construction, so all its runs are finite. The transitions in the AES$_t$ are defined in a similar way as in the AES. Within $DoDFS$, if an examined state is visited again, we stop searching on the current path and know there is a cycle in the AES. Since the number of states in the AES is finite, $DoBFS$ stops after a finite number of steps when all states in the AES are examined. In line 3, we call Algorithm III and achieve two goals: first, all leaf states in the AES$_t$ are Y-states; second, no deadlocking states exist in the AES$_t$. We denote by $Q_{Y-\text{leaf}}^{AT}$ the leaf states in the AES$_t$. Since states are completely split in terms of state and string components, there is a unique run from the root y_0 to every state in the AES$_t$. Finally, we label each Y-state in the tree with both the edited string and the original string in line 6.

Edit functions embedded in the AES$_t$ only make finite insertion choices. However, this does not compromise the performance of edit functions in opacity enforcement. We use Example 1 to illustrate this point. If we build the reachability tree for this example, the cycle between Z-states $((3, 0), b)$ and $((2, 0), b)$ is broken and the transition c is removed. Thus, if we consider edit functions embedded in the AES$_t$, then string b can only be mapped to dab. However, all strings of the form $da(bc)^n b$ where $n \geq 1$ reach state 2. It does not really matter whether b is edited to a string containing a loop or not.

In the following discussion, we let the edit function make the same decisions every time a Z-state in the AES is reached. Hence, if there exists a cycle in the AES, the edit function does not change decisions whenever the cycle is visited. Therefore no information is lost if we consider edit functions embedded in the AES$_t$ and repeat the same edit decisions when two states share the same state components.

Remark 3: We briefly analyze the space complexity of the AES$_t$. First, let $Q = \max\{|Q_Y^{AT}|, |Q_Z^{AT}|, |Q_W^{AT}|\}$. The number of nodes reached by the initial state in one step transition in the AES is at most Q. Also, each node may have at most Q succeeding nodes by one step transition in the AES. Thus, the number of states reached by y_0 by two transitions is at most Q^2. The same process goes on and we know that there may be at most $|Q_Y^{AT}| + |Q_Z^{AT}| + |Q_W^{AT}|$ states between the root y_0 and any leaf state in the tree. Thus, the number of states in the AES$_t$ is at most in the order of $Q^{|Q_Y^{AT}|+|Q_Z^{AT}|+|Q_W^{AT}|+1}$. From aforementioned section's discussion, we know that the complexities of Q and $|Q_Y^{AT}| + |Q_Z^{AT}| + |Q_W^{AT}|$ are both of the order $(O(|X_{obs}|^3))$. Therefore, the complexity of the AES$_t$ does not exceed $O(|X_{obs}|^{3(|X_{obs}|^3+1)})$.

[2]The terminology of *reachability tree* is from the Petri net literature; it is employed here as it is well suited to the construction procedure in this note.

In the AES_t, some Y-states are labeled by an unsafe string and a safe string, whereas others by two safe strings. We partition Y-states as: $Q_Y^{AT1} = \{((x_d, x_f), (t, s)) \in Q_Y^{AT} : t \in L_{\text{safe}}, s \in L_{\text{unsafe}}\}$ and $Q_Y^{AT2} = \{((x_d, x_f), (t, s)) \in Q_Y^{AT} : t, s \in L_{\text{safe}}\}$.

Next, we define the *last preserved* Q_Y^{AT2} *state* as: $Q_{Y-lp}^{AT2} = \{y_t^2 \in Q_Y^{AT2} : \exists y_t^1 \in Q_Y^{AT1}, \exists \theta_1, \dots \theta_m \in \Theta, \exists e_o \in E_o, \text{ s.t. } f_{wy}^{AT}(f_{zw}^{AT}(f_{zz}^{AT} \dots (f_{zz}^{AT}(f_{yz}^{AT}(y_t^1, e_o), \theta_1), \cdots \theta_{m-1}), \theta_m), e_o) = y_t^2\}$, which serves as the "boundary" between Q_Y^{AT1} and Q_Y^{AT2} states.

Define $Q_{Y-\text{leaf}}^{AT1} = Q_{Y-\text{leaf}}^{AT} \cap Q_Y^{AT1}$ and $Q_{Y-\text{leaf}}^{AT2} = Q_{Y-\text{leaf}}^{AT} \cap Q_Y^{AT2}$ as leaf states that contain and do not contain unsafe string components. Besides, we define $Q_{Y-l}^{AT2} = Q_{Y-\text{leaf}}^{AT2} \cup Q_{Y-lp}^{AT2}$. Then, we define

$$L_{\text{leaf}}^u = \{l \in L_{\text{unsafe}} : \exists y_{\text{leaf}}^1 = ((x_d, x_f), (t, s)) \in Q_{Y-\text{leaf}}^{AT1}, \text{ s.t. } s = l\}$$

$$L_{\text{leaf}}^s = \{l \in L_{\text{safe}} : \exists y_{\text{leaf}}^2 = ((x_d, x_f), (t, s)) \in Q_{Y-\text{leaf}}^{AT2}, \text{ s.t. } s = l\}$$

$$L_{lp}^s = \{l \in L_{\text{safe}} : \exists y_{lp}^2 = ((x_d, x_f), (t, s)) \in Q_{Y-lp}^{AT2}, \text{ s.t. } s = l\}$$

as the set of unsafe strings appearing in $Q_{Y-\text{leaf}}^{AT1}$, the set of safe strings appearing in $Q_{Y-\text{leaf}}^{AT2}$ and Q_{Y-lp}^{AT2}, respectively. We further group some Y-states by their components of original strings (safe or unsafe)

$$Q_{Y-\text{leaf}}^{AT1}(l) = \{((x_d, x_f), (t, s)) \in Q_{Y-\text{leaf}}^{AT1} : s = l \in L_{\text{leaf}}^u\}$$

$$Q_{Y-\text{leaf}}^{AT2}(l) = \{((x_d, x_f), (t, s)) \in Q_{Y-\text{leaf}}^{AT2} : s = l \in L_{\text{leaf}}^s\}$$

$$Q_{Y-lp}^{AT2}(l) = \{((x_d, x_f), (t, s)) \in Q_{Y-lp}^{AT2} : s = l \in L_{lp}^s\}$$

$$Q_{Y-l}^{AT2}(l) = \{((x_d, x_f), (t, s)) \in Q_{Y-l}^{AT2} : s = l \in L_{lp}^s \cup L_{\text{leaf}}^s\}.$$

In this note, we assume that events are inserted or erased one by one, so observed one at a time. Also, both the observer's language and the safe language are prefix closed. Therefore, if a string s is mapped to string l, then all the prefixes of s are mapped to some prefixes of string l. This result is formally stated as follows.

Lemma 1: Consider an ND edit function f_{ne}, if $s, t \in P[\mathcal{L}(G)]$ satisfy $f_e(s) \subseteq f_e(t)$, then $\forall s' \preceq s, \exists t' \preceq t$, s.t. $f_e(s') \subseteq f_e(t')$.

This lemma has the implication that we can restrict attention to unsafe strings in L_{leaf}^u since all the other unsafe strings in the AES_t, being their prefixes, can be mapped to safe strings if strings in L_{leaf}^u can be mapped to safe strings. Besides, we can focus on safe strings in $L_{lp}^s \cup L_{\text{leaf}}^s$ for opacity enforcement as the other safe strings in the AES_t are their prefixes. This result further justifies why we build the reachability tree AES_t: since the AES_t explicitly contains unsafe strings in some of its leaf states, we can evaluate those leaf states and determine how those unsafe strings are edited.

B. Synthesis Algorithm

We proceed to synthesize ND-PP-enforcing edit functions based on the AES_t. We will give a condition for verifying the existence of ND-PP-enforcing edit functions and show that the verification problem is closely related with the synthesis problem. Then, we will solve these two problems together. To begin with, we derive the following result from Theorem 1, which shows that ND private safety is always ensured by the AES.

Lemma 2: If the AES is not empty, then there exists a privately safe ND edit function.

The ND public safety case is more challenging and we start by evaluating the unsafe strings in the leaf states of the AES_t. For each unsafe string $l_i \in L_{\text{leaf}}^u$, we define the set of *PP-enforcing candidate states* as $S_{\text{pp}}(l_i) = \{((x_d, x_f), (t, l_i)) \in Q_{Y-\text{leaf}}^{AT1}(l_i) : \exists y^2 = ((x_d', x_f'), (t', l')) \in Q_{Y-l}^{AT2}, \text{ s.t. } t \preceq t'\}$. That is, we search through AES_t to find $((x_d', x_f'), (t', l'))$ where some prefix of the edited string t' is just t, whereas the unedited unsafe string is also l_i. So, if the edit function reaches those states, it will be publicly safe by definition. On the other hand, if $S_{\text{pp}}(l_i) = \emptyset$ for some l_i, then, we know we cannot find a safe string that shares the same edited behavior with unsafe string l_i, in which case no ND-PP-enforcing edit function exists.

Besides, we call states in $Q_{Y-\text{leaf}}^{AT1}(l_i) \backslash S_{\text{pp}}(l_i)$ *bad candidate states* since the edited behaviors of l_i indicated in those states cannot be matched with edited behaviors of any other safe string. Thus, if those states are reached by the edit function, ND public safety cannot be achieved. Those states are expected to be avoided when synthesizing ND-PP-enforcing edit functions.

Based on those concepts, we propose Algorithm V for synthesis. First, we group the leaf states by their unsafe string components $l_i \in L_{\text{leaf}}^u$ in line 2. Each state in $Q_{Y-\text{leaf}}^{AT1}(l_i)$ corresponds to a potentially different edited behavior of l_i. Then, we search through the AES_t to find bad candidate states and remove them from the AES_t. As the removal of those states may bring in deadlocking states, we apply Algorithm III to resolve deadlocking states in line 7 and denote the remaining structure by AES_t^r. In this process, some states in $S_{\text{pp}}(l_i)$ may also be removed. We use Q_{Y-re}^{AT1} and Q_{Y-re}^{AT2} to denote the Y-states with and without unsafe string components in the AES_t^r, respectively. For unsafe string l_i, we define $S_{\text{pp}}^r(l_i)$ in line 5 as the set of PP-enforcing candidate states remaining in the AES_t^r after pruning. We claim that if $S_{\text{pp}}^r(l_i)$ is not empty for each l_i, then there exist ND-PP-enforcing edit functions in the AES_t. Finally, we may extract the edit function by following transitions in the AES_t^r

Theorem 2: Given the AES_t^r, ND-PP-enforcing edit functions exist if and only if $\forall l_i \in L_{\text{leaf}}^u, S_{pp}^r(l_i) \neq \emptyset$.

Proof: ($\Rightarrow$) By contradiction. Suppose $\exists f_{\text{ne}} \in AES_t^r$, $f_{\text{ne}} \vDash \varphi_{\text{ndpp}}$, and $\exists l_i \in L_{\text{leaf}}^u$, s.t. $S_{\text{pp}}^r(l_i) = \emptyset$. Then, we can find $s \in f_{\text{ne}}(l_i)$, s.t. $\nexists t \in L_{\text{safe}}$ and $s \in f_{\text{ne}}(t)$, which contradicts $f_{\text{ne}} \vDash \varphi_{\text{ndpp}}$.

($\Leftarrow$) Given the AES_t and the AES_t^r, it is sufficient to consider unsafe strings in L_{leaf}^u and safe strings in $L_{\text{leaf}}^s \cup L_{lp}^s$ for synthesis. Besides, we only need to check ND public safety since the AES is not empty. If $\forall l_i \in L_{\text{leaf}}^u$, $S_{\text{pp}}^r(l_i) \neq \emptyset$, we know $\forall y^1(l_i) = ((x_d, x_f), (t, l_i)) \in S_{\text{pp}}^r(l_i)$, $\exists y^2 = ((x_d', x_f'), (t', l')) \in Q_{Y-l}^{AT2} \cap Q_{Y-re}^{AT2}$, s.t. $t \preceq t'$. Since $f_{\text{ne}} \in AES_t^r$, $f_{\text{ne}} \in$ AES also holds. We let all the players make the same decisions specified at states in the AES_t^r whenever a state is reached again in the AES. So, we can design an edit function f_{ne} such that $f_{\text{ne}}(l_i) = \{t : \exists y^1(l_i) = ((x_d, x_f), (t, l_i)) \in S_{\text{pp}}^r(l_i)\}$ and $t' \in f_{\text{ne}}(l')$. Since $t \preceq t'$, we know $f_{\text{ne}}(l_i) \subseteq L_{\text{safe}}$, $\forall l_i \in L_{\text{leaf}}^u$. Therefore, f_{ne} is both privately safe and publicly safe. ■

Theorem 2 gives a necessary and sufficient condition for verifying the existence of ND-PP-enforcing edit functions. It also shows the completeness and soundness of Algorithm V, so the synthesis of ND-PP-enforcing edit functions is reduced to finding $S_{\text{pp}}^r(l_i)$ for every $l_i \in L_{\text{leaf}}^u$ in the AES_t^r. When running Algorithm V, we collect all edited strings appearing in states from $S_{\text{pp}}^r(l_i)$ and include them as the potential edited behavior of $l_i \in L_{\text{leaf}}^u$. In that way, the synthesized ND edit function is "most permissive" in the sense that it preserves all feasible edit decisions to achieve ND private safety and ND public safety.

Remark 4: Compared with deterministic edit functions, ND edit functions perform better at enforcing public safety. The intuition is as follows. Consider the case when a safe string is edited to multiple (safe) strings, which may be the edited behaviors of several unsafe strings. In the deterministic case, every string is mapped to a *unique* one so in the above-mentioned case, we are only able to guarantee that one unsafe string shares the same edited behavior with a safe string, hence, public safety is violated. Thus, a deterministic PP-enforcing edit function may not always exist. However, if nondeterminism is allowed, as long as we find an edited string whose edited behaviors correspond to the edited behaviors of (potentially multiple) unsafe strings, then ND public safety is satisfied. The above-mentioned argument further justifies why we explore ND edit functions, given that both deterministic and ND edit functions may enforce private safety.

Example 2: Let the observer in Fig. 4 be with $E_o = \{a, b, c, d\}$, states 7 and 8 are composed of only secret states from the system. We omit the steps of building the AES and the AES_t, instead we directly show the AES_t in Fig. 5 . While we only label leaf states with strings here and $Q_{Y-\text{leaf}}^{AT1}$ states are marked in red (those states contain an unsafe

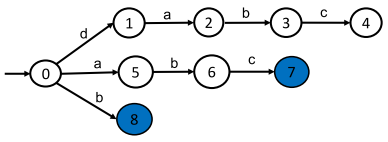

Fig. 4. Observer in Example 2.

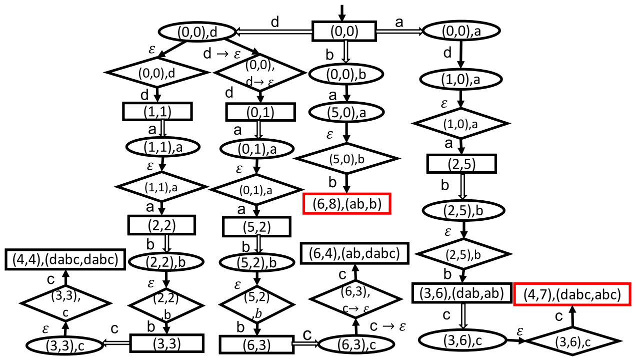

Fig. 5. AES$_t$ in Example 2.

string label). Due to the edit constraints (not explicitly stated here), the edit function can only make decisions and reach states as indicated in the AES$_t$. We can see that $((6,8),(ab,b))$ shares the first string component with $((6,4),(ab,dabc))$, $((4,7),(dabc,abc))$ shares the first string component with $((4,4),(dabc,dabc))$. Also, unsafe string b is edited to ab, unsafe string abc is edited to $dabc$, and safe string $dabc$ is edited to $dabc$ or ab.

It is interesting to notice that if we let the edit function be deterministic, i.e., every string is mapped to a unique one, then no PP-enforcing edit functions exist here since unsafe strings b and abc cannot share the same modified behavior with safe string $dabc$ simultaneously. However, an ND-PP-enforcing edit function exists by Algorithm V. No states are removed from the AES$_t$ and we have $S^r_{pp}(b) = \{((6,4),(ab,dabc)\}$ and $S^r_{pp}(abc) = \{((4,4),(dabc,dabc)\}$. So, the edit function inserts a before event b occurs from state 0; inserts d before event a occurs from state 0; inserts nothing before event d occurs from state 0 or just erases that d. This example reveals that introducing nondeterminism to edit functions may contribute to opacity enforcement by allowing more plausible denial for the intruder's inference, compared with the deterministic counterpart.

VII. Conclusion

We discussed opacity enforcement by edit functions in ND settings. Based on the knowledge of the adversary, we defined private safety and public safety of ND edit functions and then investigated their enforcement. This note is the first to apply ND edit functions to enforce opacity. The concept of edit constraint was introduced to restrict the choices of edit functions. Then, we reformulated the problem as a three-player game and proposed the AES, which embedded all privately safe edit functions satisfying edit constraints. Finally, an algorithm was presented for synthesizing ND-PP-enforcing edit functions based on the reachability tree of the AES.

References

[1] J. W. Bryans, M. Koutny, and P. Y. A. Ryan, "Modelling opacity using Petri nets," *Electr. Notes Theor. Comput. Sci.*, vol. 121, pp. 101–115, 2005.
[2] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*, 2nd ed. Berlin, Germany: Springer, 2008.
[3] F. Cassez, "The dark side of timed opacity," in *Int. Conf. Inf. Secur. Assurance*, 2009, pp. 21–30.
[4] 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.
[5] S. Chédor, C. Morvan, S. Pinchinat, and H. Marchand, "Diagnosis and opacity problems for infinite state systems modeled by recursive tile systems," *Discrete Event Dyn. Syst., Theory Appl.*, vol. 25, no. 1/2, pp. 271–294, 2015.
[6] J. Chen, M. Ibrahim, and R. Kumar, "Quantification of secrecy in partially observed stochastic discrete event systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 1, pp. 185–195, 2017.
[7] Y. Falcone and H. Marchand, "Enforcement and validation (at runtime) of various notions of opacity," *Discrete Event Dyn. Syst., Theory Appl.*, vol. 25, no. 4, pp. 531–570, 2015.
[8] R. Jacob, J.-J. Lesage, and J.-M. Faure, "Overview of discrete event systems opacity: Models, validation, and quantification," *Annu. Rev. Control*, vol. 41, pp. 135–146, 2016.
[9] Y. Ji and S. Lafortune, "Enforcing opacity by publicly known edit functions," in *Proc. 56th IEEE Conf. Decis. Control*, 2017, pp. 4866–4871.
[10] Y. Ji, Y.-C. Wu, and S. Lafortune, "Enforcement of opacity by public and private insertion functions," *Automatica*, vol. 93, pp. 369–378, 2018.
[11] Y. Ji, X. Yin, and S. Lafortune, "Opacity enforcement by insertion functions under energy constraints," in *Proc. 14th Int. Workshop Discrete Event Syst.*, 2018, pp. 291–297.
[12] C. Keroglou and C. N. Hadjicostis, "Probabilistic system opacity in discrete event systems," *Discrete Event Dyn. Syst., Theory Appl.*, vol. 28, pp. 289–314, 2018.
[13] F. Lin, "Opacity of discrete event systems and its applications," *Automatica*, vol. 47, no. 3, pp. 496–503, 2011.
[14] T. Masopust and X. Yin, "Complexity of detectability, opacity and A-diagnosability for modular discrete event systems," *Automatica*, vol. 101, pp. 290–295, 2019.
[15] A. Saboori and C. N. Hadjicostis, "Notions of security and opacity in discrete event systems," in *Proc. 46th IEEE Conf. Decis. Control*, 2007, pp. 5056–5061.
[16] 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.
[17] S. Takai and Y. Oka, "A formula for the supremal controllable and opaque sublanguage arising in supervisory control," *SICE J. Control, Meas., Syst. Integr.*, vol. 1, no. 4, pp. 307–311, 2008.
[18] 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.
[19] 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.
[20] Y.-C. Wu, V. Raman, B. C. Rawlings, S. Lafortune, and S. A. Seshia, "Synthesis of obfuscation policies to ensure privacy and utility," *J. Autom. Reason.*, vol. 60, no. 1, pp. 107–131, 2018.
[21] X. Yin and S. Lafortune, "A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems," *IEEE Trans. Autom. Control*, vol. 61, no. 8, pp. 2140–2154, Aug. 2016.
[22] 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.
[23] B. Zhang, S. Shu, and F. Lin, "Maximum information release while ensuring opacity in discrete event systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 3, pp. 1067–1079, Jul. 2015.