Secret Protection in Discrete-Event Systems with Generalized Confidentiality Requirements

Ziyue Ma, *Senior Member, IEEE*, Kai Cai, *Senior Member, IEEE*

***Abstract*— In this paper we propose a general framework to design optimal secret protection policies in discrete-event systems. The system is modeled by an automaton in which several states are *secret* and assigned with different confidentiality requirement. Events in the system can be protected to verify the identity of the user, and a user who successfully executes/passes a protected event gains some authorization. Our purpose is to design an event-protecting policy such that any user, either legal or unauthorized, who visits a secret state must have an authorization that satisfy the requirement of confidentiality of the state. We consider the criteria of optimality on protecting policies as to protecting policies with a minimum degree of disturbance to legal users' normal operations. To this aim, we use Moore machines to model the dynamics of the clearance level of users when using the system. Then, we develop an auxiliary data structure called the *generalized secret automaton*, based on which we propose a method to design a protecting policy using the classical supervisory control theory. The minimally disruptive protecting policy is then represented by an automaton called the *secret enforcer* whose state size is polynomial both in the number of the plant states and the number of secret states in the plant.**

***Index Terms*— Discrete-event systems, automata, secret protection, supervisory control, security, cyber-physical systems**

I. Introduction

Security issues in *cyber physical systems* have drawn much attention in recent years [1]–[3]. *Secret protection* [4]–[7] is to ensure that the *secrets* of a system are not exposed to unauthorized external intruders. To this aim, any operational sequence that allows a user to reach a secret state must contain a number of security checks for which a user must perform an identity verification to pass. For example, a user of a mobile phone must enter a password to unlock the phone and then pass a two-step verification (i.e., receiving an SMS code from the server) to prove his/her identity before getting access to some sensitive information such as the credit card numbers. In such a case, an unauthorized intruder who cannot legally pass the identity checks must pay some efforts to hack through them by stealing the password or forging the identity tokens. If the effort of hacking these security checks is high enough, an attack towards the secrets can be considered practically prevented.

In theory, the secret protection can be done by *protecting* all events at all times in a system, e.g., we associate an identity check for all events in the system. However, such a strategy is possibly too costly and will surely annoy legal users if each click triggers a password check. So, a series of work has been done to develop protecting policies (which determine the events to be protected next) from the prospective of both *cost* and *disruptiveness*. The cost criterion means the minimization of the physical expenditure on implementing the protecting policy. In [4], [5], the set of protectable events is partitioned into distinct levels, and the objective is to minimize the maximum of the levels of protected events. In [6] it is proved that the problem of minimizing the cost of secret protection belongs to complexity class P if all transitions are distinctly labeled (using a construction of s-t min-cut in the secret automaton), and the problem is weakly NP-hard if shared labels are allowed [7]. The other direction is to minimize the *disruptiveness* to users, which means a minimum degree of disturbance to the normal operations of legal users, which we believe even more important since the users' experience is strongly (negatively) correlated with the customer attrition. In [6] it is proved that the synthesis of such an optimal protecting policy can be transformed into a supervisory control problem in an auxiliary structure called the *secret automaton*. A protecting policy that minimally disturbs the users can then be obtained by achieving the maximally permissive supervisor in the corresponding secret automaton. The work in [8] generalizes the approach in [6] in which a particular type of dynamic clearance levels called the k-*lifespan clearance* is considered.

In this paper we follow the direction of [8] and study the secret protection problem with generalized confidentiality requirements. We consider systems modeled by a fully observable finite state automata[1] in which some states are defined as *secrets*. Each state is associated with a *confidentiality requirement*. An event is said to be *protected* if a security

This work was supported in part by the National Natural Science Foundation of China under Grant No. 62373313, JSPS KAKENHI Grant nos. 21H04875 and 22KK0155, and by Open Foundation of the State Key Laboratory of Fluid Power and Mechatronic Systems (Grant No. GZKF-202324).

Z. Ma is with School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China, and also with the State Key Laboratory of Fluid Power and Mechatronic Systems, Zhejiang University, Hangzhou 310027, China (email: maziyue@xidian.edu.cn).

K. Cai is with Department of Core Informatics, Osaka Metropolitan University, Osaka 558-8585, Japan (e-mail: cai@omu.ac.jp).

[1] In this work, we consider that we are the system administrator who aims to design secret protecting strategies in the system's design stage. Thus, it is reasonable to assume that the plant is fully observable to the administrator. Secret protection with partially observation may occur if the plant is pre-designed and our aim is to design an external validating agent. Such a problem is challenging and is part of our future work.

check is associated with it, and a user can gain/increase his/her clearance level by passing/executing protected events. A secret protection problem (SPP) is to design an event-protecting policy to enforce a confidentiality requirement such that every sequence from the initial state and reaching a secret state must grant the user a clearance that is not lower than the required confidentiality requirement of the secret.

Comparing with existing works, this paper focus on a generalized framework from two aspects. The first generalization is that in the paper we propose a general formalism to model the dynamical change of the clearance levels (based on which a solution is developed). In the aforementioned works [4]–[7], the clearance level of a user is granted once and forever. That is, when a user passes a protected event, his/her security clearance level monotonically increases and will not decrease for any actions thereafter. This means that after passing a sufficient number of security checks, a user is granted a full authority to the entire system permanently. However, we find that such a model may be oversimplified to describe real systems. For example, a user of a cellphone is granted the access to the bank statement after passing a face verification. Nevertheless, after a long period of idle time, when the user wants to access bank statement again, it is often necessary to ask for another identity check since the actual *person* who is currently using the phone may not be the same one who passes the face verification long time ago. In other words, it is meaningful to consider the case where the clearance level may both increase and decrease (or more complex logical dynamics) when operating the system, which is done in this paper. The aforementioned works such as [6]–[8] can be viewed as a particular case of the systems considered here.

The second generalization done in this work is that we consider multi-type of clearances. In [8] and other previous work, although not explicitly stated, the model has a unique identity check counter so that (i) a user can infinitely increase his/her clearance level by repeatedly passing low-level identity checks, and (ii) if a user has the access to high-level secrets, he/she is also accessible to all low-level secrets. However, such an access control logic is also oversimplified since a user may repeatedly enter the password to get a very high clearance level so that all secrets are accessible to him/her, which we do not want happen. In practice, it may be desired that some secrets must pass some particular types of identity checks, e.g., we may require that in order to access the bank statement a face recognition is mandatory regardless how many times the user has entered a password. This motivates us to introduce multi-type of clearances in this paper.

When generalizing the secret protecting problem from our previous work [6] to multi-type dynamic clearance conditions, several challenges are encountered. First, the secret specification for a single, monotonically increasing clearance level is defined by "a number of protected events on each trajectory" in [6], which is not suitable for describing multi-type dynamic clearance levels. Hence, new formal methods have to be developed to model the generalized clearance specification, and the monolithic model of systems with multi-type of clearance levels needs to be established. Moreover, multi-type clearance levels usually lead to a complex monolithic model, the structural complexity of the secret protecting policy needs to be optimized. These challenges, both conceptual and technical, are effectively addressed in this work, and our main contributions are summarized in the following.

1) First, we model the dynamic changes of a clearance level as a Moore machine called *clearance Moore machine* (CM) in which the relation of a plant sequence executed by a user and the resulting clearance level is encoded. If the system has multi-type of clearances, n-type of clearances can be modeled by n Moore machines. This framework is fairly general which provides a convenient way to model confidentiality requirements in practice. In the aforementioned works such as [6]–[8], the clearance level is hard-coded as "the number of protected events in a sequence" which can be viewed as a particular type of Moore machines introduced here.
2) We define an operator called the *clearance parallel synchronization* that is applied on a plant and the associated CMs, which returns a *generalized secret automaton* (GSA) in which the information of all possible protecting decisions are encoded. This is a new operator which is mandatory to solve our generalized problem since in [6] the monotonically increasing clearance level is hard-coded into the SA, which cannot be straightforwardly extended to multi-type clearance formulation. Thanks to this newly developed secret parallel synchronization, we prove that secret protecting problems can be reduced to a standard supervisory control problem on the structure of GSA. Hence, a minimally disruptive protecting policy can be obtained by first computing the maximally permissive supervisor based on the GSA, followed by converting it to its corresponding protecting policy.
3) Finally, the supervisor obtained by solving the supervisory control problem based on GSA turns out to have redundancy, which is not amenable to be used directly as a secret protecting enforcer. Therefore, a trimming procedure is newly proposed to remove the redundancy in the supervisor computed from GSA, resulting in a compact automaton called the *secret enforcer* to describe the synthesized protecting policy. It is shown that the scale of the secret enforcer is polynomial to the scale of the secret protection problem.

A closely related notion in cyber-security, called *opacity* [3], [9], [10], is also extensively studied in discrete-event systems. We point out that opacity is different from secret protection studied in this paper. In brief, opacity assumes that an intruder can passively observe the events and infer some secrets. In our problem, however, we assume that an intruder may disguise as a legal user to access the plant, which may not be recognized by an operator. Since we require that the plant be usable to legal users, the dynamics and the output of a plant are not allowed to be modified. For example, we cannot disable an event in the plant using supervisory control [11]–[13] or modify the output of the system [14], [15] as in opacity, since

legal users may still need the event. Other related perspective of research include *intrusion detection* [16]–[21] and *attack-resilience (tolerance) control* [22]–[27]. Since we assume that an intruder may disguise as a legal user and does not act beyond the nominal functionality of the system (e.g., to corrupt sensor readings), no abnormal behavior can be observed. Hence, we do not try to detect the existence of intruders. Instead, we set protective measures on the operational routines to increase the difficulty of unauthorized access to the secrets, which may practically prevent potential intruders.

The rest of this paper is organized in seven sections. Basic notions of automata are recalled in Section II. In Section III, the problem of secret protection is formulated, and the notions of the clearance Moore machines are proposed. In Section IV, the generalized security automaton are proposed. In Section V, we develop a method to compute a minimally disruptive protecting policy using supervisory control In Section VI we synthesize the secret enforcer automaton to encode the minimally disruptive protecting policy. Section VII draws our conclusions.

II. Preliminaries

A. Finite State Automaton

A *finite state automaton* (*automaton* for short) is a four-tuple $G = (Q, \Sigma, \delta, q_0)$, where Q is a set of *states*; Σ is a set of *events*; $\delta : Q \times \Sigma \to Q$ is the partial transition function; and $q_0 \in Q$ is the initial state.

We use Σ^* to denote the *Kleene closure* of Σ, consisting of all finite sequences composed by the events in Σ (including the *empty sequence* ε). Given a sequence $s \in \Sigma^*$, $|s|$ denotes the *length* of s. The transition function δ is extended to $\delta^* : Q \times \Sigma^* \to Q$ by recursively defining $\delta^*(q, \varepsilon) = q$ and $\delta^*(q, s\sigma) = \delta(\delta^*(q, s), \sigma)$, where $s \in \Sigma^*$ and $\sigma \in \Sigma$. The *language* of G, denoted by $L(G)$, is defined as $L(G) = \{s \in \Sigma^* \mid \delta^*(q_0, s) \in Q\}$.

We use $\Gamma_G(q) = \{\sigma \in \Sigma \mid \delta(q, \sigma) \text{ is defined}\}$ to denote the set of events that are *enabled* at state $q \in Q$ in G, and we use $\Gamma_G(s) = \Gamma_G(\delta^*(q_0, s))$ to denote the set of events that are *enabled* after sequence s.

Given an automaton $G = (Q, \Sigma, \delta, q_0)$, the *accessible part* of G, denoted as $Ac(G)$, is the automaton $G' = (Q', \Sigma, \delta', q_0)$ obtained from G by removing all unreachable states and their corresponding transitions. Precisely speaking, $Q' = \{q \in Q \mid (\exists s \in L(G))\ \delta^*(q_0, s) = q\}$, and δ' is the restriction of δ to $Q' \times \Sigma \to Q'$.

A sequence $\bar{s} \in \Sigma^*$ is a *prefix* of a sequence $s \in \Sigma^*$ if $s = \bar{s}s'$ where $s' \in \Sigma^*$, which is denoted as $\bar{s} \prec s$. We use $\bar{s}_k$ (where $0 \le k \le |s|$) to denote the prefix of s of length k, i.e., $s = \bar{s}_k s'$ where $|\bar{s}_k| = k$ and $s' \in \Sigma^*$. The *prefix closure* of a language $L \subseteq \Sigma^*$ is the set $\overline{L} = \{s \in \Sigma^* \mid \exists s' \in \Sigma^*, ss' \in L\}$.

B. Supervisory Control in Discrete-Event Systems

Supervisory control theory of discrete-event systems was first proposed by Ramadge and Wonham [28]. For a plant automaton $G = (Q, \Sigma, \delta, q_0)$, the event set Σ is partitioned into two disjoint subsets $\Sigma = \Sigma_c \cup \Sigma_{uc}$ where Σ_c is the set of *controllable events* and Σ_{uc} is the set of *uncontrollable events*.

In [28], the control objective, called the *(language) specification*, is defined by a regular language $K \subseteq \Sigma^*$. A supervisor S that dynamically disables events of the plant such that the closed-loop language of S over G is restricted within K. Here we use S/G to denote the closed-loop system composed by the plant G under the supervision of S, and we use $L(S/G)$ to denote the language of S/G. A supervisor S runs in parallel with the plant and, when a plant generates a sequence $s \in L(G)$, makes a control decision $\xi(s) \subseteq \Sigma_c$ that allows all events in $\xi(s)$ to execute (or equivalently, to disable all controllable events not in $\xi(s)$). Note that a supervisor cannot disable any uncontrollable $\sigma \in \Sigma_{uc}$ in any case. A language K is said to be *controllable* with respect to $L(G)$ if $\overline{K}\Sigma_{uc} \cap L(G) \subseteq \overline{K}$. If K is not controllable, a supervisor S enforcing K can be obtained by computing the *supremal controllable sublanguage* [29] of $L(G)$ with respect to K, i.e.:

$$L(G)^{\uparrow K} = \bigcup \{H \subseteq L(G) \mid H \text{ is controllable to } L(G)\}.$$

by iterative manipulations on regular languages.

A *state specification* defines a set of forbidden states $Q_l \subseteq Q$ that requires that the plant does not reach any state in Q_l. A supervisor S that enforces Q_l can be similarly obtained by first converting the state specification Q_l into its equivalent language specification $K = \{s \in L(G) \mid \delta^*(q_0, s) \notin Q_l\}$ followed by computing the corresponding $L(G)^{\uparrow K}$.

C. Moore Machine

A *Moore machine* is a six-tuple $M = (Q, \Sigma, \delta, q_0, O, \Sigma_O)$, where (Q, Σ, δ, q_0) is an automaton and $O : Q \to \Sigma_O$ is the *output function* which maps each state to the *output alphabet* Σ_O (a finite set). In this paper, the output alphabet used is $\mathbb{N}$.[2] Hence, we omit the output alphabet and simply denote a Moore machine as $M = (Q, \Sigma, \delta, q_0, O)$ where $O : Q \to \mathbb{N}$.

III. Secret Protecting Problem Formulation

A. Plant Automata and Clearance Moore Machines

We consider a plant modeled by an automaton $G = (Q, \Sigma, \delta, q_0)$; at a subset of states $Q_s \subset Q$ some secrets, such as credit numbers or crucial personal data, are stored. By reaching these states, secret information may be obtained. To protect the secrets from being accessed by unauthorized intruders, some events of the plant should be protected such that any user (legal or unauthorized) who accesses the plant must have a clearance no less than the security requirement of the secret state he/she attempts to visit. In the previous works, a simplified access control logic is used such that (i) a user can infinitely increase his/her clearance level by repeatedly passing low-level identity checks, and (ii) if a user has the access to high-level secrets, he/she is also accessible to all low-level secrets. In this work, we consider a general and more practical

[2]Precisely speaking, the output alphabet we will use is the finite subset of consecutive natural numbers $\mathbb{N}_b = \{0, 1, 2, \ldots, b-1, b\}$ where b is an integer bound dependent on the secret protecting problem instance. Thus the output alphabet is finite.

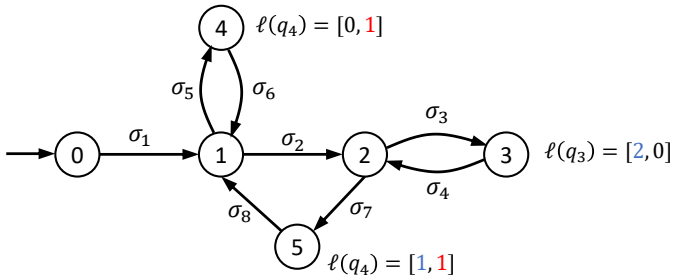


Fig. 1. A plant automaton that contains three secret states (q_3, q_4, q_5) and two clearance types (Type-1 in blue, Type-2 in red).

scenario of access control by allowing multiple independent types of clearances. Hence, we define the *security requirement* as the following.

Definition 1: Given a system G with n types of clearances, a *confidentiality requirement* is a function $\ell : Q \to \mathbb{N}^n$ ($\mathbb{N} = \{0, 1, 2, \ldots\}$) that assigns a *confidentiality level* $\ell(q) = [l_1(q), l_2(q), \ldots, l_n(q)]$ to each state q, which means that to reach state q a user must be granted for each clearance type i a clearance level u_i no less than $l_i(q)$. ♢

In plain words, a security requirement assigns each state q a vector $[l_1(q), l_2(q), \ldots, l_n(q)]$ so that a user who visits state q must hold clearance type-1 no less than level $l_1(q)$, clearance type-2 no less than level $l_2(q)$, and so on. Such multi-type of clearance is very common in practice. For example, in an airport a staff must hold different badges to enter different zones, where in general the authorization of the two badges are not inclusive. As another example, to access the bank statement on a phone a face recognition is mandatory regardless how many times the user has entered a password.

Example 1: Consider the automaton in Figure 1 which represents a computer network. For readability, each state q_i is denoted as "i" in the state circle. A user who visits the system is first initialized at the initial state q_0. The user can connect to the system from the router via σ_1 and then can switch between two servers q_1 and q_2 via σ_2 and σ_3. Several secret files are located at database q_3, q_4, and q_5 in the system, and the access control consists of two types of clearances. To visit secret q_3 one need level 2 of clearance type-1 while to visit secret q_4 one need level 1 of clearance type-2. On the other hand, to access secret q_5 one must hold both types of clearances with level 1. For readability, clearance type-1 is marked in blue and clearance type-2 in red. Our aim is to design a proper protecting strategy so that any user who visits any of the three secret states must holds both the clearance types and levels required by the corresponding state. ♢

B. Protecting Policies and Dynamics of Clearance Levels

We use ϑ to denote a *protecting policy* whose definition is given below. Intuitively, for each sequence $s \in L(G)$ observed, ϑ decides which events are protected after s.

Definition 2: Given a plant $G = (Q, \Sigma, \delta, q_0)$, a protecting policy is a function $\vartheta : L(G) \to 2^\Sigma$ such that after observing an event sequence $s \in L(G)$, the set of protected events following s is $\vartheta(s) \subseteq \Sigma$. ♢

Remark 1: In previous works, it is assumed that some events are *unprotectable*, i.e., $\Sigma = \Sigma_p \cup \Sigma_{up}$ where only events in Σ_p can be protected. In this work we follow this setting: only a subset of the events can be protected. However, we do not explicitly announce the set of un/protected events. The (un)protectability of events are implicitly encoded into the clearance Moore machines which will be introduced shortly.

In a system with n types of clearances, a user's total clearance can be characterized by an n-dimensional vector $\mathbf{u} = [u_1, u_2, \ldots, u_n]$ where u_i ($1 \le i \le n$) denotes the level of the type-i clearance. When the user executes a sequence of events s in $L(G)$, eventually he/she will be granted a clearance level of $[u_1, u_2, \ldots, u_n]$ depending on the action of protecting events in s made by ϑ. To characterize how ϑ affects sequence s, for each event σ_i in Σ, we introduce two new events λ_i and α_i to denote whether event σ_i is protected or unprotected, respectively. Hence, given a sequence $s = \sigma_1 \cdots \sigma_k \in L(G)$ and a protecting policy ϑ, we map s into a new sequence in a way such that each σ_i ($i = 1, \ldots, k$) is successively replaced by α_i (resp., λ_i) if it is unprotected (resp., protected) by ϑ after $\bar{s}_{i-1}$. We use Σ_λ and Σ_α to denote the sets of λ- and α-events, respectively. That is, for $\Sigma = \{\sigma_1, \ldots, \sigma_l\}$, we have $\Sigma_\lambda = \{\lambda_1, \ldots, \lambda_l\}$ and $\Sigma_\alpha = \{\alpha_1, \ldots, \alpha_l\}$.

Definition 3: Given a plant $G = (Q, \Sigma, \delta, q_0)$ and a sequence $s = \sigma_1 \cdots \sigma_k \in L(G)$, the *protected sequence* of s with respect to a protecting policy ϑ is a new sequence $s' = \sigma'_1 \cdots \sigma'_k \in (\Sigma_\alpha \cup \Sigma_\lambda)^*$ such that:

$$\sigma'_i = \begin{cases} \alpha_i & \text{if } \sigma_i \notin \vartheta(\bar{s}_{i-1}) \\ \lambda_i & \text{if } \sigma_i \in \vartheta(\bar{s}_{i-1}). \end{cases}$$

This is denoted as $s_{\uparrow\vartheta} := s'$, and s is said to be the *original sequence* of s'. The protected language of $L(G)$ over ϑ is defined as: $L_\vartheta(G) = \{s_{\uparrow\vartheta} \mid s \in L(G)\}$. ♢

It is reasonable to assume that a user who visits the system from the entry (i.e., the initial state q_0) is granted a zero clearance level for all types of clearances, i.e., $\mathbf{u}_0 = \mathbf{0} = [0, \ldots, 0]$. After the execution of each event σ, the clearance level $\mathbf{u}$ may vary, depending on the current and the history of the protecting decision. The dynamics of each type i clearance level are usually problem-dependent. Notice that for each protection sequence $\vartheta s \in (\Sigma_\alpha \cup \Sigma_\lambda)^*$, the final level of each type of clearance level is deterministic. For each type i clearance level, we define a *clearance function*

$$\mathcal{C}_i : L_\vartheta(G) \to \mathbb{N}, \quad 1 \le i \le n. \tag{1}$$

That is, given G and ϑ, if a user executes s from state q_0 to some state q, then the resulting clearance level of type i is $\mathcal{C}_i(s_{\uparrow\vartheta})$. The total clearance level vector after s is given by a total function $\mathcal{C} : L_\vartheta(G) \to \mathbb{N}^n$:

$$\mathcal{C}(s) = [\mathcal{C}_1(s), \ldots, \mathcal{C}_n(s)].$$

Therefore, a protecting policy ϑ is valid if, after executing a sequence s to yield state q, the final level for each type of clearance is no fewer than the confidentiality requirement $l_i(q)$ of state q.

This article has been accepted for publication in IEEE Transactions on Automatic Control. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAC.2024.3481030

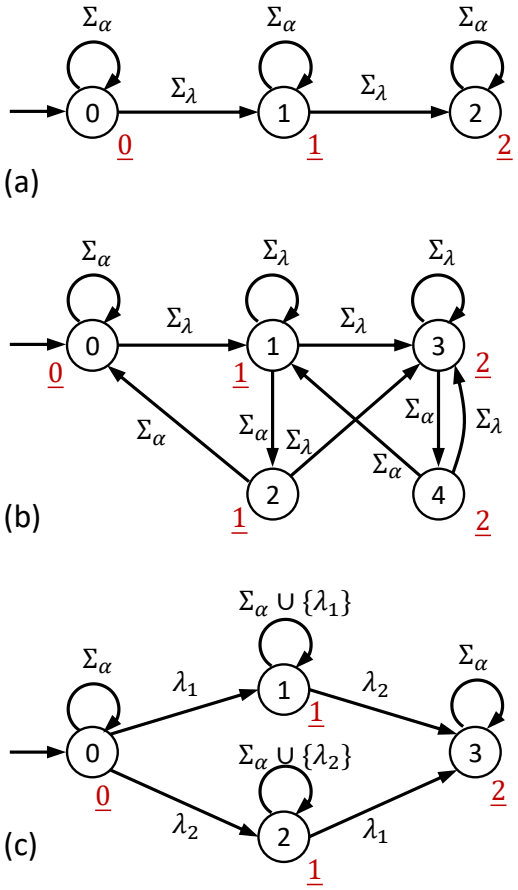


Fig. 2. Several clearance Moore machines. The output of each state is located next to the state with an underline "_", e.g., in (a) $\boldsymbol{O(q_2) = 2}$.

Definition 4: Given a security requirement ℓ and clearance functions $\mathcal{C}_i$ for each type i of clearance, a protecting policy ϑ is *valid* if for any sequence $s \in L(G)$ that yields a state $q \in Q$, $\mathcal{C}(s) \geq \ell(q)$ holds, i.e., $\mathcal{C}_i(s_{\uparrow\vartheta}) \geq l_i(q)$ holds for all i. ♢

The specific total function $\mathcal{C}$ and all $\mathcal{C}_i$ in Eq (1) are problem-dependent. In this paper we consider $\mathcal{C}_i$'s that can be described by Moore machines.

Definition 5: A clearance Moore machine (CM) for S_i is a five-tuple $M_i = (Q_{M_i}, \Sigma_{M_i}, \delta_{M_i}, q_{M_i,0}, O_i)$ where:

- $(Q_{M_i}, \Sigma_{M_i}, \delta_{M_i}, q_{M_i,0})$ is an automaton;
- the alphabet is $\Sigma_{M_i} \subseteq (\Sigma_\alpha \cup \Sigma_\lambda)$;
- the output function is $O_i : Q_{M_i} \to \mathbb{N}$.

Thus M_i simulates a clearance function $\mathcal{C}_i : L_\vartheta(G) \to \mathbb{N}$ such that:

$$(\forall s \in L(M_i))\, O_i(\delta_{M_i}(q_{M_i,0}, s)) = \mathcal{C}_i(s). \tag{2}$$

♢

In a clearance Moore machine, the state transition describes how the clearance level changes when executing a sequence of protected/unprotected events. Normally, the alphabet of M_i contains all unprotected events, i.e., $\Sigma_{M_i} \supseteq \Sigma_\alpha$, since "not to protect an event" is always an option. On the other hand, it may happen that some events in Σ_λ do not appear in Σ_{M_i} because of the following reasons.

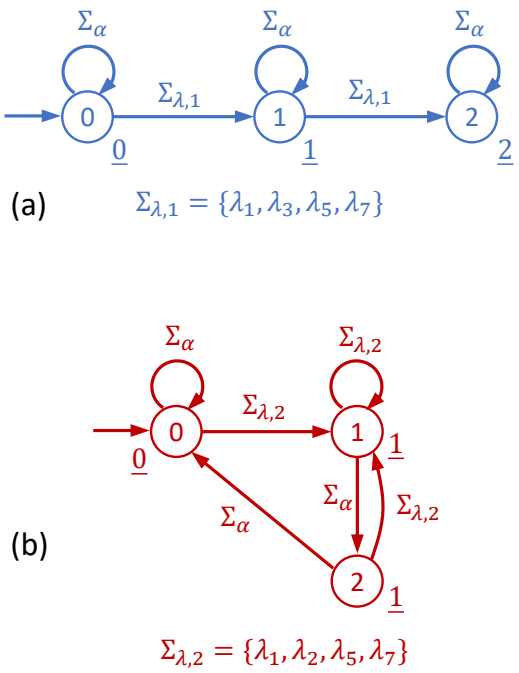


Fig. 3. The clearance automaton in Example. 2. The output of each state is located next to the state with an underline "_".

- If protecting an event σ_j does not affect the output $O_i(\delta_{M_i}(q_{M_i,0}, s))$ for all s, i.e., event σ_j does not affect the clearance level, the corresponding λ_j does not appear in the alphabet of M_i. For example, the clearance type for accessing the bank statement is related to face recognition but unrelated to password check. In such a case transitions which are physically associated with password checks do not appear in the CM for that clearance type.
- If an event σ_j is not protectable (which means that we cannot set an identification check on that event), the corresponding λ_j does not appear in all CMs.

The dynamics of clearance levels used in the literature can be considered as particular cases of CMs in Definition 5. For example, the monotonically nondecreasing level considered in [6] can be modeled by the Moore machine in Figure 2(a), while the "k-lifespan security function" in [8] as that in Figure 2(b). Figure 2(c) models another situation such that: executing protected event σ_1 (i.e., λ_1) alone or event σ_2 (i.e., λ_2) alone can only increase the clearance level to 1 at most, while executing both λ_1 and λ_2 can increase the clearance level to 2. Hence, Moore machines are expressive models for capturing many different evolutions of clearance levels in practice. In the running example of this paper, we consider the two CMs in Figure 3 that model the dynamics of the two types of clearance levels of the plant in Figure 1.

Example 2: Recall the plant in Figure 1 in which the confidentiality requirements contains two types (type-1 in blue and type-2 in red) of clearances which are associated with two types of users. Type-1 clearance is with students who can pay the fee at q_3. Once the user's student identity is verified, the clearance is permanently granted. On the other hand, type-2 clearance is with teachers who give scores of

This article has been accepted for publication in IEEE Transactions on Automatic Control. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAC.2024.3481030

exams at q_4. Hence, a more strict rule to regularly recheck the teacher's identity is applied. The corresponding clearance Moore machine

$$M_1 = (Q_{M_1}, \Sigma_{M_1}, \delta_{M_1}, q_{M_1,0}, O_1)$$

and

$$M_2 = (Q_{M_2}, \Sigma_{M_2}, \delta_{M_2}, q_{M_2,0}, O_2)$$

are depicted in Figure 3. Moore machine M_1 models a standard nondecreasing clearance change: executing protected events $\lambda_1, \lambda_3, \lambda_5, \lambda_7$ increases the type-1 clearance level by 1, and the maximal clearance level of type-1 is 2. Moore machine M_2 models a standard nondecreasing clearance change: executing protected events $\lambda_1, \lambda_2, \lambda_5, \lambda_7$ increases the clearance level by 1, and the maximal clearance level of type-2 is 1. Moreover, if two unprotected events are executed consecutively, the type-2 clearance level decreases by 1, i.e., to zero, meaning that the authorization of type-2 clearance has expired.

We can see that protecting event λ_2 only affects type-2 clearance and protecting event λ_3 only affects type-1 clearance, since λ_2 does not appear in M_1 while λ_3 does not appear in M_2. The protection on $\sigma_1, \sigma_5, \sigma_7$ affects both type of clearances, so $\lambda_1, \lambda_5, \lambda_7$ appears in both M_1 and M_2. Moreover, in this example we assume that events $\sigma_4, \sigma_6, \sigma_8$ are unprotectable. So, the corresponding $\lambda_4, \lambda_6, \lambda_8$ do not appear in neither CMs. ♢

C. Secret Protecting Problem and Minimally Disruptive Protecting Policy

Given a plant $G = (Q, \Sigma, \delta, q_0)$, a security requirement $\ell : Q \to \mathbb{N}^n$ such that n types of clearances are modeled by n CMs $M_1, \ldots, M_n$, our aim is determine a valid disruptive protecting policy ϑ. This can be done by choosing the most conservative protecting policy ϑ_{max} that protects all events at all times, i.e.:

$$(\forall s \in L(G)) \quad \vartheta(s) = \Sigma.$$

It is reasonable to assume that such ϑ_{max} is valid, otherwise the secret protecting problem has no solution. However, it is clear that such a conservative protecting policy will surely annoy legal users. For user's convenience, security checks should not be popped out when the user just wants to use calculator or check the weather, but should be required only if he/she wants to inspect the credit cards and other private information of the account. This motivates us to design a protecting policy such that it protects events only when necessary. In plain words, a protecting policy is minimally disruptive if it does not protect any events unless it has to.

Definition 6: [Minimal Disruptiveness] Given a plant $G = (Q, \Sigma, \delta, q_0)$, a security requirement $\ell : Q \to \mathbb{N}^n$ such that n types of clearances are modeled by n CMs $M_1, \ldots, M_n$, a protecting policy ϑ is *minimally disruptive* if there does not exist another protecting policy $\vartheta' \neq \vartheta$ and a sequence $s = \sigma_1 \cdots \sigma_m \in L(G)$ that satisfy the following conditions simultaneously:

$$\begin{cases} \vartheta(\sigma_1 \cdots \sigma_i) = \vartheta'(\sigma_1 \cdots \sigma_i), \forall i \in \{1, \ldots, m-1\} \\ \vartheta'(s) \subsetneq \vartheta(s). \end{cases} \tag{3}$$

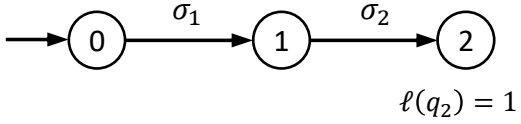


Fig. 4. An automaton for the illustration of minimal disruptiveness of protecting policies.

♢

In plain words, ϑ is minimally disruptive if there does not exist a different protecting policy ϑ' such that for a sequence s of length m, the first $m-1$ protecting decisions are the same while for the last decision ϑ protects more events than ϑ'.

Remark 2: We point out that in Definition 6 the equality of decisions of the first $m-1$ steps (i.e., the first line of Eq. (3)) is necessary. Suppose that we remove the first condition in Eq. (3) from Definition 6. Consider the plant in Figure 4 in which both σ_1 and σ_2 are protectable and two protecting policies:

$$\vartheta_1 : \vartheta_1(\varepsilon) = \{\sigma_1\}, \vartheta_1(\sigma_1) = \emptyset$$
$$\vartheta_2 : \vartheta_2(\varepsilon) = \emptyset, \vartheta_2(\sigma_1) = \{\sigma_2\}$$

Clearly, ϑ_2 is minimally disruptive since it does not protect σ_1 but σ_2. However, one can see $\vartheta_1(\varepsilon) \supseteq \vartheta_2(\varepsilon)$ and $\vartheta_1(\sigma_1) \subseteq \vartheta_2(\sigma_1)$, i.e., ϑ protects more events than ϑ_1 when observing σ_1. Hence, the condition "$\vartheta(\sigma_1 \cdots \sigma_i) = \vartheta'(\sigma_1 \cdots \sigma_i), \forall i \in \{1, \ldots, m-1\}$" rules out such a phenomenon: under such condition ϑ_2 is minimally disruptive while ϑ_1 is not. ♢

Now we are ready to formalize the *secret protection problem* (SPP) that will be studied in this paper.

Problem 1: [Secret Protection Problem] Given a plant $G = (Q, \Sigma, \delta, q_0)$, a security requirement $\ell : Q \to \mathbb{N}^n$ such that n types of clearances are modeled by n CMs $M_1, \ldots, M_n$, determine a valid, minimally disruptive protecting policy ϑ.

IV. Generalized Security Automata

In this section, we first introduce a structure called the security automaton in which both the plant behavior and the protecting actions are encoded. First, given n clearance Moore machines, we establish a monolithic Moore machine by applying the *parallel synchronization* on them [30].

Definition 7: [30] Given two Moore machines $M_1 = (Q_1, \Sigma_1, \delta_1, q_{M_1,0}, O_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_{M_2,0}, O_2)$, the parallel synchronization of M_1 and M_2 is a Moore machine $M = M_1 || M_2$ by:

$$M = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{M_1,0}, q_{M_2,0}), O_1 \times O_2),$$

where

$$\delta((q', q''), \sigma) = \begin{cases} (\delta_1(q', \sigma), \delta_2(q'', \sigma)), & \sigma \in \Sigma_1 \cap \Sigma_2 \\ (\delta_1(q', \sigma), q''), & \sigma \in \Sigma_1 \setminus \Sigma_2 \\ (q', \delta_2(q'', \sigma)), & \sigma \in \Sigma_2 \setminus \Sigma_1 \end{cases}$$

and

$$O(q', q'') = (O_1(q'), O_2(q''))$$

for all $(q', q'') \in Q_1 \times Q_2$. ♢

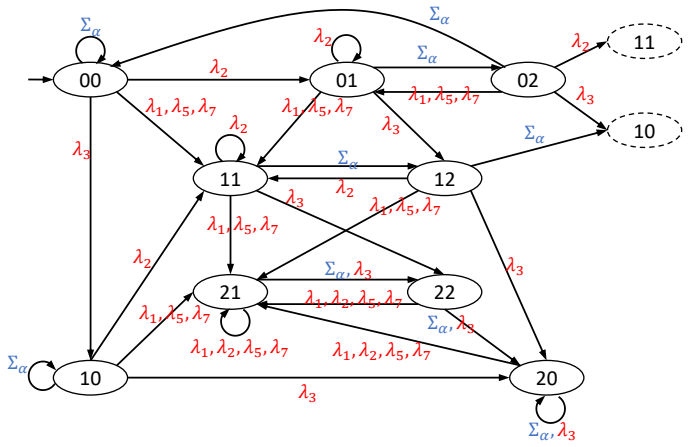


Fig. 5. The parallel synchronization $\boldsymbol{H} = \boldsymbol{M_1}||\boldsymbol{M_2}$ of the two clearance Moore machines $\boldsymbol{M_1}, \boldsymbol{M_2}$ in Figure 3. For readability state **11** and **10** are duplicated on the right.

Namely, the parallel synchronization $M_1||M_2$ is formed by running Moore machines M_1 and M_2 in parallel and giving outputs synchronously. One example is given in Figure 5. Parallel synchronization can be easily generalized to $n > 2$ Moore machines. Then, by doing parallel synchronization for n clearance Moore machines $M_1, \ldots, M_n$, we obtain a monolithic Moore machine $H = M_1||M_2||\cdots||M_n = (Q_H, \Sigma_H, \delta_H, h_0, O_H)$. Now we are ready to define a *secret parallel synchronization* function $||_S$ that applies on a plant and its corresponding clearance Moore machines.

Definition 8: Given a plant $G = (Q, \Sigma, \delta, q_0)$, a security requirement $\ell : Q \to \mathbb{N}^n$ modeled by n CMs $M_1, \ldots, M_n$, let $H = M_1||\cdots||M_n = (Q_H, \Sigma_H, \delta_H, h_0, O_H)$ be the monolithic clearance Moore machine. The *secret parallel synchronization* of G and H is a Moore machine $G_S = G||_S H = Ac(Q \times Q_H, \Sigma_\alpha \cup \Sigma_\lambda, \delta_S, (q_0, h_0), O_S)$ where

$$\begin{cases} \delta_S((q,h), \alpha_j) = (\delta(q, \sigma_j), \delta_H(h, \alpha_j)) \\ \delta_S((q,h), \lambda_j) = (\delta(q, \sigma_j), \delta_H(h, \lambda_j)) \end{cases}$$

in case that the corresponding $\delta(q, \sigma_j)$ and $\delta_H(h, \alpha_j)$ / $\delta_H(h, \lambda_j)$ are defined, and $O_S(q,h) = O_H(h)$. Here we write each state $h_i = [h_{i,1}, \ldots, h_{i,n}]$ as a vector to denote each Moore machine j is at its state i. Moore machine G_S is called the *generalized security automaton* (GSA) of G and H. ♢

In plain words, a GSA G_S is a Moore machine consisting of $|Q_H|$ copies of G, while only the accessible part of these copies is considered. The physical interpretation of a state (q_i, h_j) in GSA is that the plant G is at state q_i while the current clearance vector $\mathbf{u}$ of the user is $O_H(h_j)$ in H. Clearly, G_S contains no more than $|Q| \cdot |Q_H|$ states and $|Q| \cdot |Q_H| \cdot (2 \cdot |\Sigma|)$ arcs.

Example 3: Again consider the plant automaton G in Figure 1 and the clearance Moore machines M_1, M_2 in Figure 3. The corresponding GSA G_S is shown in Figure 6. For readability:

- the G-component of each state is shown in the state circle while the H-component $[a, b]$ is depicted as "$h = [a, b]$" in the grey box along with the corresponding output.
- A grey box with "$O = [a, b]$" indicates that all states in the box have the same output $[a, b]$.
- Blue and red transitions are α- and λ-transitions, respectively, while their labels (events) are omitted in the figure.

Although we omit the events on the transitions, they can be uniquely recognized from the leaving and entering states. For example, since in G the label of the transition from state q_0 to state q_1 is σ_1, the blue transition from state q_0 in box $h = [0, 0]$ to state q_1 in the *same* box has label α_1, while the red transition from state q_0 in box $h = [0, 0]$ to state q_1 in *another* box $h = [1, 1]$ has label λ_1.

We use the left-top part of G_S as an example. In the left-top box, transition $\delta_S((q_0, [0, 0]), \alpha_1) = (q_1, [0, 0])$ indicated by a blue arrow means: if the plant is at state q_0, the user's current clearance level vector is $\mathbf{u} = [0, 0]$, and event σ_1 is not protected; then by executing σ_1 the plant moves to state q_1 while the clearance level vector remains $\mathbf{u}' = O_S(q_1, [0, 0]) = O_H([0, 0]) = [0, 0]$. On the other hand, the red transition $\delta_S((q_0, [0, 0]), \lambda_1) = (q_1, [1, 1])$ means that if σ_1 is protected, by executing σ_1 the plant moves from state q_0 to state q_1 while the clearance level vector is changed to $\mathbf{u}' = O_S(q_1, [1, 1]) = O_H([1, 1]) = [1, 1]$. ♢

According to Definition 8, it is not difficult to understand that a GSA can be used to simulate the evolution of a plant and the clearance level for any protecting policy ϑ. In fact, G_S can simulate G when any ϑ is given in a way that whenever G executes s, G_S executes $s_{\uparrow\vartheta}$. Precisely speaking:

- G and G_S are initialized at q_0 and (q_0, h_0) where $h_0 = \mathbf{0}$, respectively;
- Suppose that G and G_S are at q and (q, h), respectively. When G executes the next event σ_i with $\delta(q, \sigma_i) = q'$. If σ_i is protected, G_S executes event λ_i to reach $(q', \delta_H(h, \lambda_i))$; otherwise, if σ_i is not protected, G_S executes event α_i to reach $(q', \delta_H(h, \alpha_i))$.

Moreover, the following proposition shows that $O_S(q, h)$ is the final clearance level of s with respect to ϑ, where (q, h) is the state in G_S reached by executing $s_{\uparrow\vartheta}$.

Proposition 1: Given a plant $G = (Q, \Sigma, \delta, q_0)$, a security requirement $\ell : Q \to \mathbb{N}^n$ modeled by n CMs $M_1, \ldots, M_n$, let $H = M_1||\cdots||M_n = (Q_H, \Sigma_H, \delta_H, h_0, O_H)$ be the monolithic clearance Moore machine and its GSA be $G_S = G||_S H = Ac(Q \times Q_H, \Sigma_\alpha \cup \Sigma_\lambda, \delta_S, (q_0, h_0), O_S)$. For any protecting policy ϑ and any sequence $s \in L(G)$, it holds:

$$\mathcal{C}(s_{\uparrow\vartheta}) = O_S(\delta_S^*((q_0, h_0), s_{\uparrow\vartheta}))$$

Proof: We prove this proposition by induction. First, for $|s| = 0$ (i.e., $s = \varepsilon$), the statement holds.

Now, suppose that for all $s = \sigma_1 \cdots \sigma_m$ (i.e., $|s| = m$), the statement hold. Consider an arbitrary sequence $s\sigma_{m+1}$ with $\sigma_{m+1} \in \Sigma$ and $\delta^*(q_0, s) = q_i$, $\delta(q_i, \sigma_{m+1}) = q'$. By the assumption above, it holds that $\delta_S^*((q_0, h_0), s_{\uparrow\vartheta}) = (q, h)$ in G_S and $\mathcal{C}(s_{\uparrow\vartheta}) = O_S(\delta^*(s_{\uparrow\vartheta}))$. By the definition of the GSA, at state (q, h), events α_{m+1} and λ_{m+1} are both defined such that $\delta_S((q, h), \alpha_{m+1}) = (q', \alpha_{m+1})$, $\delta_S((q, h), \lambda_{m+1}) = (q', \delta_S(h, \lambda_{m+1}))$. Hence, if $\sigma_{m+1} \in \vartheta(s)$, at state (q, h) the GSA G_S can execute λ_{m+1} to reach state $(q', \delta_S(h, \lambda_{m+1}),$

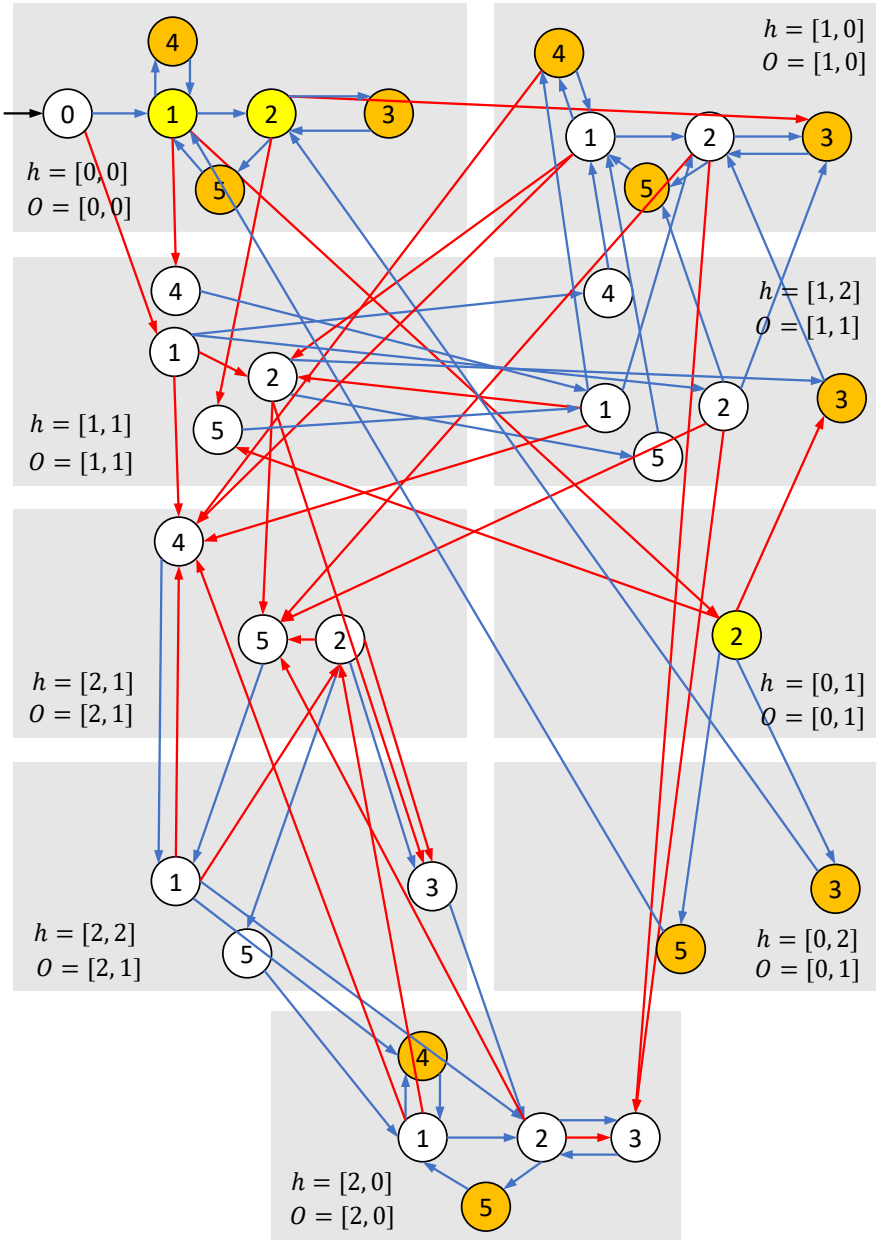


Fig. 6. Generalized security automaton $\boldsymbol{G_S = G||_S H}$ where $\boldsymbol{G}$ is the automaton in Figure 1. For readability, the $\boldsymbol{G}$-component of each state is shown in the state circle while the $\boldsymbol{H}$-component is depicted in the grey box along with the corresponding output.

otherwise it executes α_{m+1} to reach state $(q', \delta_S(h, \alpha_{m+1}))$. This indicates

$$\begin{aligned}\mathcal{C}((s\sigma_{m+1})_{\uparrow\vartheta}) &= O_S(\delta_S(h, \vartheta(s, \sigma_{m+1}))\\ &= O_S(\delta_S^*((s\sigma_{m+1})_{\uparrow\vartheta}))\end{aligned}$$

which concludes the proof. ■

V. Design of Minimally Disruptive Protecting Policies Using Supervisory Control

In this section we first recall the standard supervisory control in discrete-event systems.

Problem 2: Given a plant $G = (Q, \Sigma, \delta, q_0)$ with $\Sigma = \Sigma_c \cup \Sigma_{uc}$ where Σ_c denotes the set of controllable events and Σ_{uc} denotes the set of uncontrollable events, and given a state specification $Q_F \subseteq Q$ defining a set of forbidden states, determine a supervisor $\xi : L(G) \to 2^{\Sigma_c}$ such that G does not reach any state in Q_F when controlled by ξ.

It has been known that the supervisor ξ (if exists) can be represented by G_{Sup} a subautomaton of G obtained by removing all states in $Q \setminus Q_F$ and all states that can reach $Q \setminus Q_F$ via uncontrollable sequences in Σ_{uc}^* [29]. Moreover, a supervisor obtained via such a procedure is *maximally permissive* and unique. A maximally permissive supervisor is denoted as ξ_{max}. The control action for a sequence s is given by

$$\xi(s) = \Gamma_{\xi/G}(q) \cap \Sigma_c, \text{ where } q = \delta'(q_0', s),$$

i.e., all controllable events that are not defined after executing s are disabled.

In the sequel we show that Problem 1 can be transformed to a *supervisory control problem* in the corresponding GSA. Given a secret protection problem (i.e., SPP in Problem 1), we define its corresponding *supervisory control problem in the corresponding GSA* (SCP-GSA) as follows.

Problem 3: [SCP-GSA] Given a plant $G = (Q, \Sigma, \delta, q_0)$, a security requirement $\ell : Q \to \mathbb{N}^n$ modeled by n CMs $M_1, \ldots, M_n$, let $H = M_1 || \cdots || M_n = (Q_H, \Sigma_H, \delta_H, h_0, O_H)$ be the monolithic clearance Moore machine and its GSA be $G_S = G||_S H = Ac(Q \times Q_H, \Sigma_\alpha \cup \Sigma_\lambda, \delta_S, (q_0, h_0), O_S)$. Design a maximally permissive supervisor $\xi_{max} : L(G_S) \to 2^{\Sigma_c}$ for automaton G_S with respect to

$$\Sigma_c = \Sigma_\alpha, \quad \Sigma_{uc} = \Sigma_\lambda \tag{4}$$

and a state specification $Q_F = \{(q, h) \mid O_S(q, h) \not\geq \ell(q)\}$.

We then define the duality of a protecting policy ϑ of an SPP and a control policy ξ in the corresponding SCP-GSA. The physical interpretation of the duality of ϑ and ξ is that, if we run ϑ and ξ in parallel, ϑ protects event σ_i after a sequence $s \in L(G)$ if and only if ξ disables event α_i after a sequence $\tilde{s} = s_{\uparrow\vartheta} \in L(G_S)$.

Definition 9 ($\vartheta - \xi$ Duality): Consider an SPP for plant G and the corresponding SCP-GSA for the GSA G_S. A control policy ξ of the SCP-GSA and a protecting policy ϑ of the SPP are *dual* if for each sequence $s \in L(G)$ and each $\sigma_i \in \Sigma_p$, the following condition holds:

$$\sigma_i \in \vartheta(s) \quad \Leftrightarrow \quad \alpha_i \in \Gamma_{G_S}(s_{\uparrow\vartheta}) \setminus \xi(s_{\uparrow\vartheta}) \tag{5}$$

$\diamondsuit$

The duality of ξ and ϑ given by Definition 9 is analogous to the duality in [6] (Definition 4.3 in [6]). Such a property establishes the link between the solutions of the SPP and the SCP-GSA whose intuition is the following. Suppose that the plant is at state q while H is at state h, and G is about to execute event σ. Setting a protection on σ is associated with the disablement of the corresponding α-event at state (q, h) in the GSA $G_S = G||_S H$, which forces G_S to follow the corresponding λ-event. Similar to [6], we have the following theorem.

Theorem 1: A protecting policy ϑ is valid if and only if its dual supervisor ξ is a solution of the corresponding SCP-GSA.

Proof: ($\Rightarrow$) Let ϑ be an arbitrary valid protecting policy that enforces ℓ. We prove that its dual supervisor ξ is a solution to the SCP-GSA. The proof is by contradiction. Suppose that ξ permits a sequence s in G_S to reach a forbidden state (q, h) with $O_S(q, h) \not\geq \ell(q)$. Since $O_S(q, h) = \mathcal{C}(s_{\uparrow\vartheta}) \not\geq \ell(q)$, ϑ is not valid, which is a contradiction.

($\Leftarrow$) Let ξ be a supervisor of the SCP-GSA problem, whose supervisor automaton G_ξ is a subautomaton of G_S. We prove that its dual protecting policy ϑ is a valid protecting policy that enforces ℓ. The proof is again by contradiction. Suppose that there exists a sequence $s = \sigma_1 \cdots \sigma_m \in L(G)$ such that $\mathcal{C}(s_{\uparrow\vartheta}) \not\geq \ell(q), q = \delta^*(q_0, s)$. It implies that in the GSA by executing $s_{\uparrow\vartheta}$ a state (q, h) with $O_S(q, h) \not\geq \ell(q)$ is reached. Since each execution of unprotected event σ_i corresponds to a control action that permits α_i, it indicates that ξ permits $s_{\uparrow\vartheta}$ in G_S, i.e., the forbidden state (q, h) is reachable under the control of ξ, which is a contradiction. ■

Example 4: Again consider the GSA in Figure 6. We construct the corresponding SCP-GSA as follows. For safety requirement $\ell(q_3) = [2, 0]$, $\ell(q_4) = [0, 1]$, $\ell(q_5) = [1, 1]$, and $\ell(q_i) = \mathbf{0}$ for other states, the forbidden states are $Q_F = \{(q_i, h) \mid O_S(q_i, h) \not\geq \ell(q_i)\}$ and are depicted in orange in the same figure . The uncontrollable transitions are $\Sigma_{uc} = \Sigma_\lambda$ (in red). Then, such a problem can be solved by the Ramadge-Wonham paradigm to obtain the automaton G_{Sup} in Figure 7: it is the maximally permissive supervisor ξ_{max} enforcing Q_F for G. $\diamondsuit$

By Theorem 1, for each supervisor in an SCP-GSA, its dual protecting policy is a valid protecting policy of the original SPP. Now we show that for the maximally permissive supervisor of an SCP-GSA, its dual protecting policy is minimally disruptive. In other words, Problem 1 can be transformed into Problem 3 that is a standard supervisory control problem which can be solved by standard algorithms [29].

Theorem 2: Given a plant $G = (Q, \Sigma, \delta, q_0)$, a security requirement $\ell : Q \to \mathbb{N}^n$ modeled by n CMs $M_1, \ldots, M_n$, let $H = M_1 || \cdots || M_n = (Q_H, \Sigma_H, \delta_H, h_0, O_H)$ be the monolithic clearance Moore machine and its GSA be $G_S = G||_S H = Ac(Q \times Q_H, \Sigma_\alpha \cup \Sigma_\lambda, \delta_S, (q_0, h_0), O_S)$. Let ξ_{max} be the maximally permissive supervisor of the SCP-GSA. The protecting policy ϑ_{min} that is dual to ξ_{max} is minimally disruptive.

Proof: By contradiction, suppose that ϑ_{min} is not minimally disruptive. By Definition 6 there exists another valid protecting policy ϑ and a sequence $s = \sigma_1 \cdots \sigma_m$ such that $\vartheta(\bar{s}_i) = \vartheta_{min}(\bar{s}_i)$ holds for all $i \in \{1, \ldots, m-2\}$, and $\vartheta(\bar{s}_{m-1}) = \vartheta_{min}(\bar{s}_{m-1}) \setminus \{\sigma_m\}$ holds. In other words, the protecting decisions of ϑ_{min} and ϑ are the same for all events in s except the last one: ϑ_{min} protects the last event σ_m while ϑ does not protect σ_m. According to Theorem 1, the dual supervisor ξ of ϑ is a solution of the SCP-GSA and permits $(\bar{s}_{m-1})_{\uparrow\vartheta} \cdot \alpha_m$ in the GSA, which implies that $(\bar{s}_{m-1})_{\uparrow\vartheta} \cdot \alpha_m$ is valid. However, sequence $(\bar{s}_{m-1})_{\uparrow\vartheta} \cdot \alpha_m$ is forbidden by ξ_{max}. This contradicts the fact that ξ_{max} is maximally permissive. ■

The reduction from an SPP to its corresponding SCP-GSA is polynomial (since for a given plant the corresponding GSA can be constructed in polynomial time according to Definition 8), and the existence of solutions of SCP-GSA can be verified using existing supervisory control methods. Note that an SPP has a solution if and only if the corresponding SCP-GSA has a solution, i.e., ξ_{max} is not empty. Hence, the existence of a solution of an SPP can be verified by checking the emptiness of the corresponding supremal controllable sublanguage of the SCP-GSA. Moreover, notice that the maximally permissive supervisor ξ_{max} (whenever it exists) is unique. Therefore, the minimally disruptive protecting policy ϑ_{min} is also unique when it exists.

Corollary 1: The minimally disruptive protecting policy, if it exists, is unique.

Proof: The conclusion is immediate from the uniqueness of ξ_{max} and the duality of ξ_{max} and ϑ_{min} by Theorem 2. ■

This article has been accepted for publication in IEEE Transactions on Automatic Control. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAC.2024.3481030

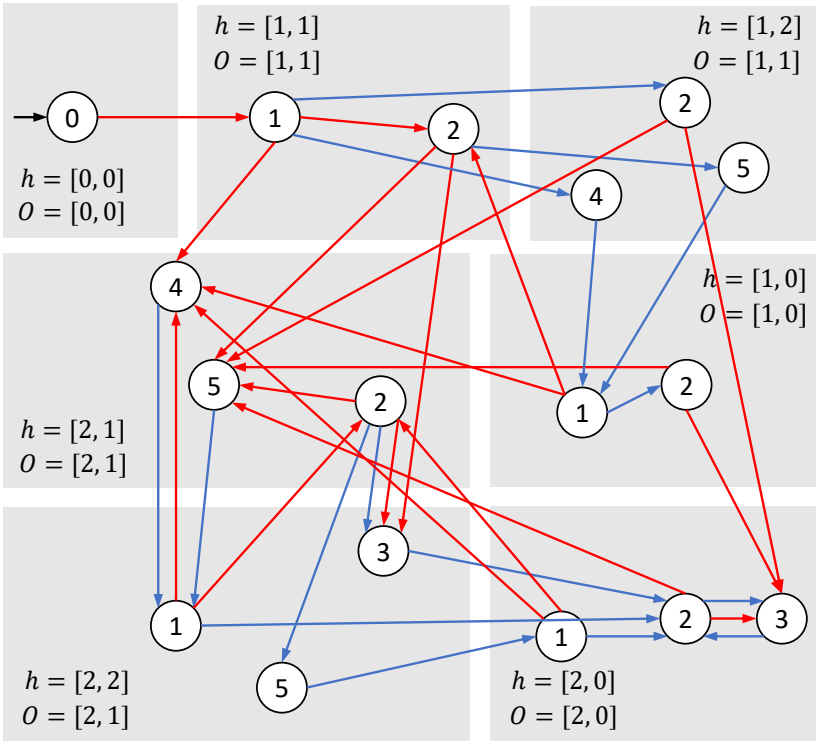


Fig. 7. The maximally permissive supervisor $\boldsymbol{G_{Sup}}$ of SCP-GSA $\boldsymbol{G_S = G||_S H}$ in Figure 6. For readability, the $\boldsymbol{G}$-component of each state is shown in the state circle while the $\boldsymbol{H}$-component is depicted in the grey box along with the corresponding output.

VI. Security Enforcer Automaton

Suppose that an SPP has a solution, i.e., the maximally permissive supervisor ξ_{max} for the corresponding SCP-GSA exists. The supervisor ξ_{max} can be viewed as a secret protecting policy ϑ_{min} by the duality in Definition 9, since in ξ_{max} all protecting decisions are encoded. Let ξ_{max} be represented by automaton $G_{sup} = (Q \times Q_H, \Sigma_\alpha \cup \Sigma_\lambda, \delta_S, (q_0, h_0), O_S)$. In this section we propose a trim procedure for G_{sup} to obtain a new compact automaton called the *enforcer*. The enforcer automaton functions the same as G_{sup} but in general has fewer states.

Notice that by the duality Eq. (5), the set of protected events in each step is unambiguously decided by the disablement of α-transitions (since ϑ_{min} protects event σ_i if and only if its dual supervisor ξ_{max} disables event α_i). Hence, if both events α_i and λ_i are defined at a state (q, h) in G_{Sup}, the λ_i-transition is redundant since the protecting decision at (q, h) is "not to protect σ_i". Hence, at state (q, h) the λ_i-transition is never executed. Hence, we can remove all such redundant transitions from G_{Sup} to obtain a concise automaton — which we call the security enforcer — to represent protecting policy ϑ_{min}.

Definition 10: Given an SPP and let ξ_{max} be represented by automaton $G_{sup} = (Q \times Q_H, \Sigma_\alpha \cup \Sigma_\lambda, \delta_S, (q_0, h_0), O_S)$. The *security enforcer* is an automaton

$$G_E = (Q_E, \Sigma_\alpha \cup \Sigma_\lambda, \delta_E, (q_0, h_0), O_E)$$

that is a subautomaton of G_{Sup} obtained by removing all λ_i transitions at all states (q, h) in G_{Sup} if α_i transitions are defined at the same state (q, h), followed by taking its accessible part. This is denoted as $G_E = Trim_\lambda(G_{sup})$. ♢

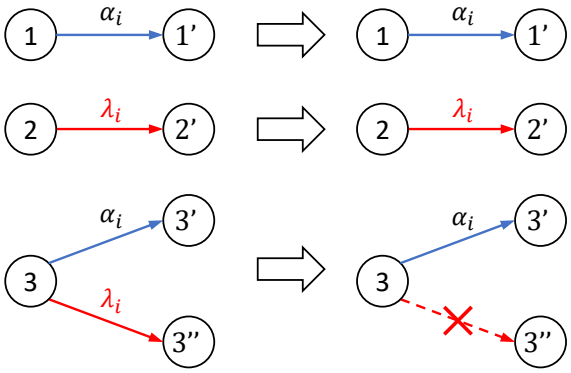


Fig. 8. The illustration of the trimming of $\boldsymbol{G_{Sup}}$ to obtain $\boldsymbol{G_E}$.

The construction of G_E from G_{Sup} is illustrated in Figure 8. Precisely speaking, automaton G_{Sup} is trimmed as follows:

- for each state (q, h) in G_{sup} and each event $\sigma_i \in \Sigma_i$, if both $\delta_S((q, h), \alpha_i)$ and $\delta_S((q, h), \lambda_i)$ are defined, remove transition $\delta_S((q, h), \lambda_i)$;
- taken the accessible part of the resulting automaton to obtain G_E.

Such a procedure can be done in polynomial time (by checking $|Q| \times |Q_H| \times 2 \cdot |\Sigma|$ transitions). The number of states in the enforcer automaton G_E in the worst case[3] is the same as that in G_S but the number of transitions is reduced to $|Q| \times |Q_H| \cdot |\Sigma|$ since at each state in G_E for each event σ_i either λ_i or α_i (or

[3]It is also notable that, as we show in Example 5, the state in G_E can be much less than the corresponding G_{sup}.

TABLE I
ILLUSTRATION FOR THE EXECUTION OF SEQUENCE $\boldsymbol{\sigma_1\sigma_5\sigma_6\sigma_2\sigma_3\sigma_4\sigma_7}$ IN EXAMPLE 5.

s	$s_{\uparrow\vartheta}$	$\vartheta(s)$	$\mathbf{u}$
ε	ε	σ_1	[0, 0]
σ_1	λ_1	$\emptyset$	[1, 1]
$\sigma_1\sigma_5$	$\lambda_1\alpha_5$	$\emptyset$	[1, 1]
$\sigma_1\sigma_5\sigma_6$	$\lambda_1\alpha_5\alpha_6$	σ_5	[1, 0]
$\sigma_1\sigma_5\sigma_6\sigma_2$	$\lambda_1\alpha_5\alpha_6\alpha_2$	σ_3, σ_7	[1, 0]
$\sigma_1\sigma_5\sigma_6\sigma_2\sigma_3$	$\lambda_1\alpha_5\alpha_6\alpha_2\lambda_3$	$\emptyset$	[2, 0]
$\sigma_1\sigma_5\sigma_6\sigma_2\sigma_3\sigma_4$	$\lambda_1\alpha_5\alpha_6\alpha_2\lambda_3\alpha_4$	σ_7	[2, 0]
$\sigma_1\sigma_5\sigma_6\sigma_2\sigma_3\sigma_4\sigma_7$	$\lambda_1\alpha_5\alpha_6\alpha_2\lambda_3\alpha_4\lambda_7$	$\emptyset$	[2, 1]

neither) is defined, but not both. The security enforcer G_E runs in parallel with G and outputs the decision of event protection:

$$\vartheta(s) = \{\sigma_i \in \Sigma \mid \Gamma_{G_E}(\delta_E((q_0, h_0), s_{\uparrow\vartheta})) \cap \Sigma_\lambda\}. \quad (6)$$

i.e., G_E protects all events σ_i's such that their corresponding λ_i's are defined while does not protect any events σ_j's such that their corresponding α_j's are defined. The reason is that according to the $\xi - \vartheta$ duality, all transitions deleted from G_{Sup} will never been executed. Hence, G_E with Eq. (6) and G_{Sup} with Eq. (5) produces the same protecting policy.

Example 5: Consider the supervisor G_{Sup} in Figure 7. Then, to obtain the security enforcer G_E, we remove all transitions λ_i in Σ_λ from state (q, h) if the corresponding unprotected event α_i is defined at the same state. For example, since both events α_1 and λ_1 are defined at state $(q_1, [1, 1])$ (in the top-middle box), the transition λ_1 from $(q_0, [0, 0])$ to $(q_2, [1, 1])$ is removed. By taking the accessible part of the resulting automaton, we obtain G_E that is depicted in Figure 9.

Now we illustrate how the secret enforcer G_E works. Assume that a user executes an event sequence $\sigma_1\sigma_5\sigma_6\sigma_2\sigma_3\sigma_4\sigma_7$ by which he/she visits states q_4, q_3, q_5 sequentially. According to the secret enforcer G_E in Figure 9, the minimally disruptive protecting policy is: $\vartheta(\varepsilon) = \{\sigma_1\}$, $\vartheta(\sigma_1) = \emptyset$, $\vartheta(\sigma_1\sigma_5) = \emptyset$, $\vartheta(\sigma_1\sigma_5\sigma_6) = \{\sigma_5\}$, $\vartheta(\sigma_1\sigma_5\sigma_6\sigma_2) = \{\sigma_3, \sigma_7\}$, $\vartheta(\sigma_1\sigma_5\sigma_6\sigma_2\sigma_3) = \emptyset$, $\vartheta(\sigma_1\sigma_5\sigma_6\sigma_2\sigma_3\sigma_4) = \{\sigma_7\}$, $\vartheta(\sigma_1\sigma_5\sigma_6\sigma_2\sigma_3\sigma_4\sigma_7) = \emptyset$. The detailed protecting decision made by G_E for each step is summarized in Table I. ♢

We recap the procedure of the developed approach for solving an SPP. Given a plant G with security requirement ℓ that contains n types of clearances, the entire algorithm includes the following steps:

1) Compose n clearance Moore machines $M_1, \ldots, M_n$ into a monolithic clearance automaton H;
2) Use secret parallel synchronization to compute the GSA $G_S = G||_S H$;
3) Solve the SCP-GSA in G_S to obtain the maximally permissive supervisor G_{sup} (if G_{sup} does not exist, the algorithm terminates since there is no solution for the original SPP);
4) Trim G_{Sup} to finally obtain the secret enforcer automaton G_E whose protecting policy ϑ_{min} is given by Eq. (6).

The roadmap of the algorithm above is illustrated in Figure 10.

At the end of this section we discuss the complexity of the proposed approach. Given a plant G with $|Q|$ states, $|\Sigma|$ events, and n types of clearance levels modeled by n Moore machines with $|Q_{c,i}|$ states, the corresponding GSA G_S has at most $|Q| \times \prod_{i=1}^{n} |Q_{c,i}|$ states and $|Q| \times 2|\Sigma| \times \prod_{i=1}^{n} |Q_{c,i}|$ events. The number of states in the enforcer automaton G_E in the worst case is the same as that in G_S that is $|Q| \times \prod_{i=1}^{n} |Q_{c,i}|$, while the number of transitions in G_E reduces to $|Q| \times |\Sigma| \times \prod_{i=1}^{n} |Q_{c,i}|$ since at each state in G_E for each event σ_i either λ_i or α_i (or neither) is defined, but not both. When $n = 1$, the secret enforcer contains $|Q| \times |Q_c|$ states and $|Q| \times |\Sigma| \times |Q_c|$ events which is also smaller than the security automaton in [6] (which is $|Q| \times |Q_c| \times 2 \cdot |\Sigma|$). Hence, the structural complexity of G_E is polynomial to the scale of the secret protection problem.

Remark 3: The exponential complexity with respect to number of types of clearances n when constructing monolithic model is unavoidable, like in many other properties in discrete-event systems such as fault diagnosis with n multi-type of faults, opacity estimation with n multi-site observers, etc. Possible approaches to reduce the complexity include supervisor decomposition/localization techniques, which will be part of our future work.

On one hand, we believe that in practice the types of clearance n is are typically not too many, since each type of clearance is associated with a basic, individual type of users of the system. For example, in Example 2, the two basic roles who can access the database are *students* and *teachers*. So, we need two types of clearance levels l_1 and l_2. For *administrators* who can access both types of data, we do not need to create a new clearance type for them but define their authority as a combined security specification $[l_1, l_2]$ in the system. Therefore, state q_5 in Figure 1 is a location to store data that can only be visited by administrators, not students nor teachers. ♢

VII. CONCLUSIONS

In this paper, we have introduced a secret protection problem in discrete-event systems modeled by automata with multi-type of clearances modeled by Moore machines. We have proposed a *secret parallel synchronization* operator on the plant and its clearance automaton, based on which an auxiliary data structure called the *generalized secret automaton* is obtained. Based on the generalized secret automata, we have proposed a method to design a minimally disruptive protecting policy using the supervisory control theory. The minimally disruptive protecting policy we have designed is represented by a secret enforcer automaton that is polynomial in the size of the given secret protection problem.

Our future work includes extending the current centralized setup to a decentralized one with multiple subsystems containing local secrets, as well as applying supervisor localization to decompose the centralized protecting policies into local protection policies.

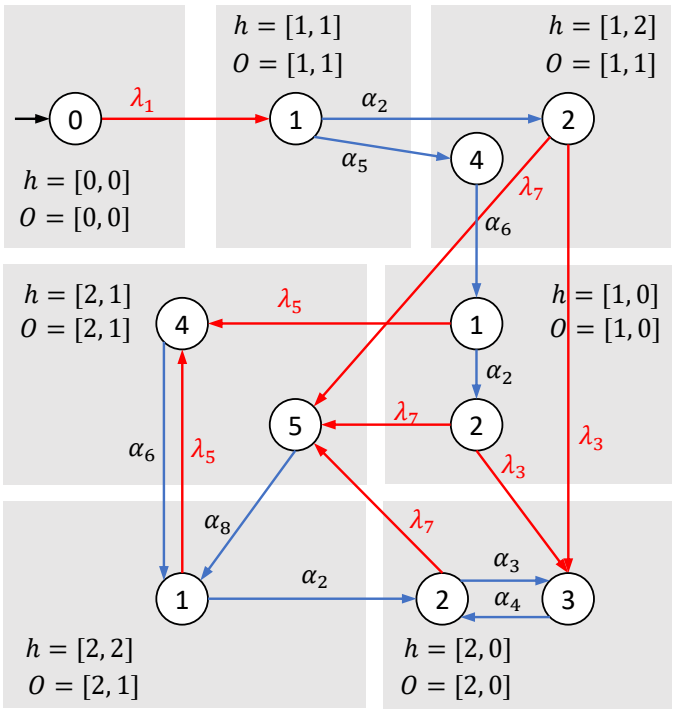


Fig. 9. The security enforcer in Example 5. The G-component of each state is shown in the state circle while the H-component is depicted in the grey box along with the corresponding output.

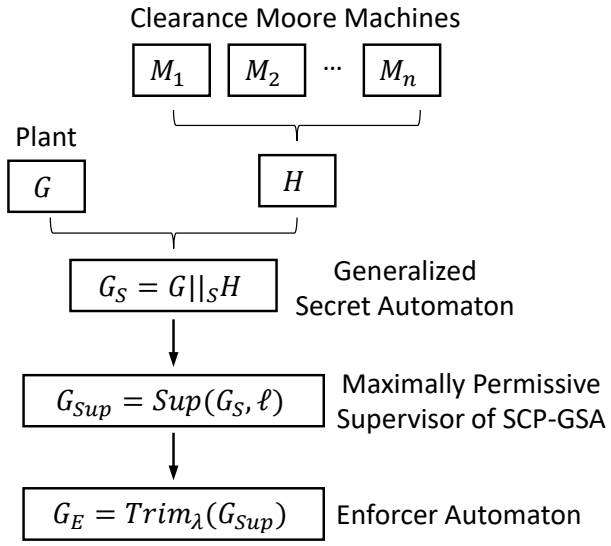


Fig. 10. The roadmap (algorithm) of the approach in this work.

[1] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 42–57, 2013.

[2] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," *ACM Computing Surveys*, vol. 42, no. 1, pp. 1–31, 2009.

[3] S. Lafortune, F. Lin, and C. Hadjicostis, "On the history of diagnosability and opacity in discrete event systems," *Annual Reviews in Control*, vol. 45, pp. 257–266, 2018.

[4] S. Matsui and K. Cai, "Secret securing with multiple protections and minimum costs," in *Proceedings of the 58th IEEE Conference on Decision and Control*, 2019, pp. 7635–7640.

[5] ——, "Usability aware secret protection with minimum cost," *Nonlinear Analysis: Hybrid Systems*, vol. 43, p. 101111, 2021.

[6] Z. Ma and K. Cai, "Optimal secret protections in discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 67, no. 6, pp. 2816–2828, 2022.

[7] Z. Ma, J. Jiang, and K. Cai, "Secret protections with costs and disruptiveness in discrete-event systems using centralities," *IEEE Transactions on Automatic Control*, vol. 69, p. to appear, 2024.

[8] Z. Ma and K. Cai, "Optimal secret protection in discrete event systems with dynamic clearance levels," in *Proceedings of the 22nd IFAC World Congress (IFAC WC'23)*, 2023, pp. 3579–3584.

[9] R. Jacob, J.-J. Lesage, and J.-M. Faure, "Overview of discrete event systems opacity: Models, validation, and quantification," *Annual Reviews in Control*, vol. 41, pp. 135–146, 2016.

[10] F. Lin, "Opacity of discrete event systems and its applications," *Automatica*, vol. 47, no. 3, pp. 496–503, 2011.

[11] X. Yin and S. Lafortune, "A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 8, pp. 2140–2154, 2016.

[12] A. Saboori and C. N. Hadjicostis, "Opacity-enforcing supervisory strategies via state estimator constructions," *IEEE Transactions on Automatic Control*, vol. 57, no. 5, pp. 1155–1165, 2012.

[13] J. Dubreil, P. Darondeau, and H. Marchand, "Supervisory control for opacity," *IEEE Transactions on Automatic Control*, vol. 55, no. 5, pp. 1089–1100, 2010.

[14] Y. Ji, X. Yin, and S. Lafortune, "Enforcing opacity by insertion functions under multiple energy constraints," *Automatica*, vol. 108, p. 108476, 2019.

[15] 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.

[16] R. Fritz and P. Zhang, "Modeling and detection of cyber attacks on discrete event systems," in *Proceedings of the 14th IFAC Workshop on Discrete Event Systems*, Sorrento, Italy, 2018, pp. 285–290.

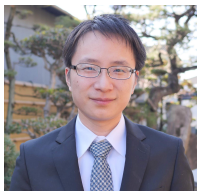
[17] M. Agarwal, "Rogue twin attack detection: A discrete event system paradigm approach," in *Proceedings of the 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, Oct 2019, pp. 1813–1818.

[18] L. K. Carvalho, Y. C. Wu, R. Kwong, and S. Lafortune, "Detection

and mitigation of classes of attacks in supervisory control systems," *Automatica*, vol. 97, pp. 121–133, 2018.
[19] Y. Tong, Y. Wang, and A. Giua, "A polynomial approach to verifying the existence of a threatening sensor attacker," *IEEE Control Systems Letters*, vol. 6, pp. 2930–2935, 2022.
[20] M. R. Alves, P. N. Pena, and K. Rudie, "Discrete-event systems subject to unknown sensor attacks," *Discrete Event Dynamic Systems*, vol. 32, no. 1, pp. 143–158, 2022.
[21] C. Gao, C. Seatzu, Z. Li, and A. Giua, "Multiple attacks detection on discrete event systems," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 2352–2357.
[22] R. Meira-Goes, E. Kang, R. Kwong, and S. Lafortune, "Synthesis of sensor deception attacks at the supervisory layer of cyber-physical systems," *Automatica*, vol. 121, p. Article 109172, 2020.
[23] M. Wakaiki, P. Tabuada, and J. P. Hespanha, "Supervisory control of discrete-event systems under attacks," *Dynamic Games and Applications*, vol. 9, pp. 965–983, 2019.
[24] L. Lin and R. Su, "Synthesis of covert actuator and sensor attackers," *Automatica*, vol. 130, p. Article 109714, 2021.
[25] R. Su, "About existence of resilient supervisors against smart sensor attacks," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, 2022, pp. 4263–4269.
[26] P. M. Lima, M. V. S. Alves, L. K. Carvalho, and M. V. Moreira, "Security of cyber-physical systems: Design of a security supervisor to thwart attacks," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 2030–2041, 2022.
[27] R. Su, "On decidability of existence of nonblocking supervisors resilient to smart sensor attacks," *Automatica*, vol. 154, p. 111076, 2023.
[28] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of IEEE*, vol. 77, no. 1, pp. 81–98, 1989.
[29] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*. Springer, 2019.
[30] J. Moerman, "Learning product automata," in *International Conference on Grammatical Inference*. PMLR, 2019, pp. 54–66.

Ziyue Ma (Senior Member, IEEE) received the B.Sc. degree and the M.Sc. degree in Chemistry from Peking University, Beijing, China, in 2007 and 2011, respectively. In 2017 he got the Ph.D degree in cotutorship between the School of Electro-Mechanical Engineering of Xidian University, China (in Mechatronic Engineering), and the Department of Electrical and Electronic Engineering of University of Cagliari, Italy (in Electronics and Computer Engineering). He joined Xidian University in 2011, where he is currently an Associate Professor in the School of Electro-Mechanical Engineering. His current research interests include control theory in discrete event systems, automata and Petri net theories, fault diagnosis/prognosis, resource optimization, and information security.

Dr. Ma is a member of Technical Committee of IEEE Control System Society on Discrete Event Systems and IFAC Technical Committee 1.3 on Discrete Event and Hybrid Systems. He is serving as the Associate Editor of the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING and is serving/has served in the Conference Editorial Board of IEEE Conference on Automation Science and Engineering (CASE'17–'24), European Control Conference (ECC'19–'24), and IEEE International Conference on Systems, Man, and Cybernetics (SMC'19–'24). He is/was the Track Committee Member of the International Conference on Emerging Technologies and Factory Automation (ETFA'17–'24). In 2016 and 2022 he received the Outstanding Reviewer Award from the IEEE TRANSACTIONS ON AUTOMATIC CONTROL and IEEE CONTROL SYSTEM LETTERS, respectively.



Kai Cai (Senior Member, IEEE) received the B.Eng. degree in Electrical Engineering from Zhejiang University, Hangzhou, China, in 2006; the M.A.Sc. degree in Electrical and Computer Engineering from the University of Toronto, Toronto, ON, Canada, in 2008; and the Ph.D. degree in Systems Science from the Tokyo Institute of Technology, Tokyo, Japan, in 2011. He is currently a Full Professor at Osaka Metropolitan University. Previously, he was an Associate Professor at Osaka City University (2014–2020), an Assistant Professor at the University of Tokyo (2013–2014), and a Postdoctoral Fellow at the University of Toronto (2011–2013).

Dr. Cai's research interests include cooperative control of multi-agent systems, discrete-event systems, and cyber-physical systems. He is the co-author with Z. Lin of "Directed Cooperation" (KDP 2023), and with W.M. Wonham of "Supervisory Control of Discrete-Event Systems" (Springer 2019) and "Supervisor Localization" (Springer 2016). He is serving as a Senior Editor for Nonlinear Analysis: Hybrid Systems. He was an Associate Editor for IEEE Transactions on Automatic Control (2017–2023), the Chair for IEEE CSS Technical Committee on Discrete Event Systems (2019–2022), and a member of IEEE CSS Conference Editorial Board (2017–2022). He received the Pioneer Award of SICE in 2021, the Best Paper Award of SICE in 2013, the Best Student Paper Award of IEEE Multi-Conference on Systems & Control in 2010, and the Young Author's Award of SICE in 2010.