Dynamic Secret Protection in Discrete Event Systems via Reinforcement Learning

Jie Ren, Ruotian Liu, Agostino Marcello Mangini, and Maria Pia Fanti

***Abstract*— This paper proposes a reinforcement learning framework for dynamic secret protection in discrete event systems. We formulate the secret protection problem as a reinforcement learning task over automata. Compared with classical supervisory control theory, the proposed framework provides an automated approach for dynamic secret protection in the systems without requiring the explicit construction of a security automaton. During training, the agent explores protection and non-protection actions at each state. State-action pairs are penalized when secret states are reached with insufficient protection, and rewarded when security requirements are satisfied, while protection costs are also taken into account. Through interaction with the environment, the agent gradually learns an optimal protection policy. We implement and compare Q-learning and SARSA algorithms. Experimental results show that both methods achieve a high success rate and produce feasible protection policy. However, SARSA demonstrates higher training efficiency and better robustness.**

I. Introduction

Driven by the rapid advancement of digital technologies, information security has become a critical concern in areas such as cybersecurity and industrial automation [1]. Many practical cyber-physical systems can be modeled as discrete event systems (DES), where sensitive information or functionalities are accessible only after satisfying specific security requirements. For example, mobile payment systems require users to complete multiple authentication steps, such as passwords, biometric verification, or one-time codes, before accessing financial operations. This reflects a fundamental security principle: sensitive states should be reached only with sufficient protection effort.

Opacity is a fundamental concept in DES for protecting privacy and secret information, ensuring that secret states cannot be inferred by external observers from observed event sequences [2]. While opacity focuses on concealing secret states, it does not explicitly regulate access to them. This limitation motivates the notion of secret protection, which addresses the problem of safeguarding sensitive states in DES. In this context, the system is modeled as an automaton in which different states are assigned various secret levels. The goal of secret protection is to safeguard certain events such that any execution sequence reaching a secret state passes through at least the required number of designated protection measures. By enforcing these requirements, sensitive states can only be accessed after sufficient protection measures are applied, corresponding to the specified security level. This framework ensures that secrets remain practically safe while providing a structured way to balance security enforcement and operational efficiency.

Most existing studies on secret protection are developed within the framework of supervisory control theory (SCT) [3]. Ma et al. [4] propose a layered security automaton to optimize protection strategies. Further studies analyze the secret protection problem with minimum cost and prove that the decision version of secret protection is NP-hard, which motivates heuristic and metaheuristic approaches, including cost-weighted centrality and genetic algorithms [5]. The notion of disruptiveness is introduced to evaluate the impact of protection policy on normal system operations, leading to strategies that balance protection cost and operational interference. However, existing approaches of classical SCT either necessitate the explicit construction of security automata or rely on optimization techniques to identify optimal protection strategies. A significant drawback of these optimization-based methods is delayed feedback, as the fitness of a strategy can only be evaluated after an entire sequence is executed. This lack of immediate feedback hinders their effectiveness in addressing the dynamic, state-based decision requirements of complex systems. Consequently, the high computational complexity and requirement to build security automata in classical SCT motivate the use of reinforcement learning (RL) for large-scale systems.

RL is well suited for DES, as it can handle large-scale state spaces and generate control decisions online without requiring a complete system model. Consequently, several studies have explored the integration of RL with DES. For instance, in the research of [6], RL is employed to construct supervisors for partially observed DES by encoding specifications as rewards rather than explicit formal languages. Similarly, the authors of [7] propose a synthesis method for supervisors in decentralized DES based on RL under imprecise specifications and uncertain environments. More recently, hybrid frameworks combining RL and SCT have been investigated. The study in [8] proposes a hybrid SCT-RL control framework to handle stochastic events in DES, providing deterministic safety guarantees together with flexible decisions based on the system context. Moreover, the work in [9] investigates the scheduling problem of identifying minimal control sequences while ensuring deadlock avoidance in Petri nets. In addition, model-driven deep RL frameworks have been developed to enable nonblocking coordination in large-scale distributed DES [10]. However, to our best knowledge, the use of RL for enforcing secret

Jie Ren, Ruotian Liu, Agostino Marcello Mangini, and Maria Pia Fanti are with the Department of Electrical and Information Engineering, Polytechnic University of Bari, 70125 Bari, Italy (e-mails: j.ren@phd.poliba.it, ruotian.liu@poliba.it, agostinomarcello.mangini@poliba.it, mariapia.fanti@poliba.it).

protection in DES has not yet been explored.

To address these limitations and fill the research gap, we propose a RL-based framework that dynamically interacts with the automaton environment, learns state-based optimal actions online, and gradually synthesizes a secret protection policy. Substantial penalties are imposed for insufficient protection, while conditional rewards reinforce successful protection. This approach avoids explicit construction of security automata and balances security requirement with operational cost.

This paper makes two primary contributions to the field of DES. First, we propose a RL-based framework for secret protection that avoids the explicit construction of security automata, thereby alleviating the state explosion and high computational complexity inherent in classical SCT. Second, we develop an online and dynamic protection policy in which an agent interacts step by step with nondeterministic finite automata (NFA) environment, enabling autonomous protection decisions that balance security requirements and operational cost.

The remainder of this paper is organized as follows. Section II reviews the preliminaries. It defines secret protection in DES and provides the fundamental theories of RL. Section III presents the RL-based secret protection framework. This section illustrates the problem formulation and details the configuration of the state space, action space, and reward function of training. Section IV introduces a case study to explore secret protection policy and provide a systematic comparison of the Q-learning and SARSA algorithms based on experimental results. Finally, Section V concludes the paper and discusses directions for future research.

II. Preliminaries

A. Language and Automata

Let E be a set of events. E^* is the set of all finite strings composed of elements of E, including the empty string ε. A language $L \subseteq E^*$ is a set of finite-length strings of labels in E^*. For a string t, $|t|$ denotes the number of elements in t, where $|\varepsilon| = 0$. For a string $s \in E^*$, a string s' is a prefix of s if $(\exists s'' \in E^*)s's'' = s$. Note that two strings s and t are equal if they have the same length (i.e., $|s| = |t|$) and the same elements. The set of all prefixes of s is written as $\bar{s}$ or $\{\bar{s}\}$.

A NFA [11] is a four-tuple $\mathcal{N} = (Q, E, \delta, q_0)$ where $Q = \{q^{(1)}, q^{(2)}, \ldots, q^{(N)}\}$ is a finite set of states, $E = \{\sigma^{(1)}, \sigma^{(2)}, \ldots, \sigma^{(K)}\}$ is a finite set of inputs, $\delta : Q \times E \to 2^Q$ is the next-state transition function, and $q_0 \subseteq Q$ is the set of initial states.

B. Secret Protection

In a system $\mathcal{N} = (Q, E, \delta, q_0)$, certain states are designated as secret states containing confidential information. To prevent unauthorized access, security measures are enforced by protecting a subset of events, denoted as $E_p \subseteq E$. Specifically, these protectable events can be secured through mechanisms such as password verification. We define the execution of such a protected event as a security check. By this means, the system ensures that any user, whether authorized or not, must undergo a required number of security checks before reaching a secret state.

Formally, the event set E is partitioned into protectable events E_p and unprotectable events E_{up}, i.e., $E = E_p \cup E_{up}$. A security requirement is defined as a function $\beta : Q \to \mathbb{N}$ that assigns each state a security level, where $\beta(q)$ represents the minimum number of security checks required to reach q from the initial state q_0. The set of secret states is given by $Q_s = \{q \in Q \mid \beta(q) > 0\}$. In this paper, we assume that the initial state is non-secret, i.e., $\beta(q_0) = 0$.

Definition 1. *Given a system $G = (Q, E, \delta, q_0)$, a state-based secret protection policy is defined as a mapping*

$$\vartheta : Q \to 2^{E_p}, \tag{1}$$

where $\pi(q) \subseteq E_p$ denotes the set of protectable events that are enforced with protection when the system is at state $q \in Q$. Equivalently, the policy can be represented by a state-dependent protection function $\vartheta : Q \times E_p \to \{0, 1\}$, where $\vartheta(q, \sigma) = 1$ (resp., 0) indicates that event $\sigma \in E_p$ is protected (resp., unprotected) when the system is at state $q \in Q$. □

Definition 2 (Valid state-based secret protection policy)**.** *Given a security level requirement $\beta : Q \to \mathbb{N}$, a state-based protection policy $\vartheta : Q \times E_p \to \{0, 1\}$ is said to be* valid *if, for every secret state $q \in Q_s$ and for all event sequences $w = \sigma_1\sigma_2 \cdots \sigma_n \in L(G)$ such that $\delta(q_0, w) = q$, the following condition holds:*

$$\sum_{i=1}^{n} \vartheta(q_{i-1}, \sigma_i) \geq \beta(q), \tag{2}$$

where $q_{i-1} = \delta(q_0, \sigma_1 \cdots \sigma_{i-1})$. □

In other words, a policy is valid if it guarantees that all secret states are protected. This means that for any path ending in a secret state, the accumulated security checks must not be less than the state's predefined security requirement.

Definition 3 (Security Automaton for NFA)**.** *[4] Given an NFA plant $\mathcal{N} = (Q, E, \delta, q_0)$ with $\delta : Q \times E \to 2^Q$ and a security requirement $\beta : Q \to \mathbb{N}$, the SA of G is a nondeterministic finite automaton $H = (Q_H, E_H, \delta_H, q_{0,0})$, where $Q_H \subseteq Q \times \{0, 1, \ldots, \beta_{max}\}$ is the augmented state space, and $h \in \{0, \ldots, \beta_{max}\}$ is the security counter tracking cumulative protection actions. The event set is $E_H = E_a \cup E_b$, where E_a and E_b represent unprotection and protection of events, respectively. Starting from $q_{0,0}$ with $h = 0$, the transition function $\delta_H : Q_H \times E_H \to 2^{Q_H}$ is defined such that for any $\delta(q_i, E_j) = Q'$, it holds that $\delta_H(q_{i,h}, b_j) = \{q_{k,h} \mid q_k \in Q'\}$ and $\delta_H(q_{i,h}, a_j) = \{q_{k,h+1} \mid q_k \in Q'\}$ for all $h \leq \beta_{max}$,.*

Example 1. *As shown in Fig. 1, we consider the NFA plant $\mathcal{N}$ consists of 20 states, denoted by q_0–q_{19}. Among them, q_{10}, q_{14}, and q_{19} are designated as secret states, with required security levels of 2, 3, and 4, respectively. The event set*

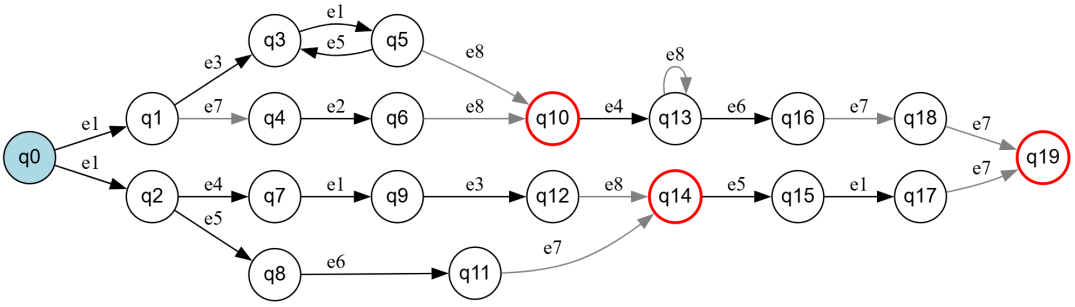

Fig. 1. An NFA model for secret protection.

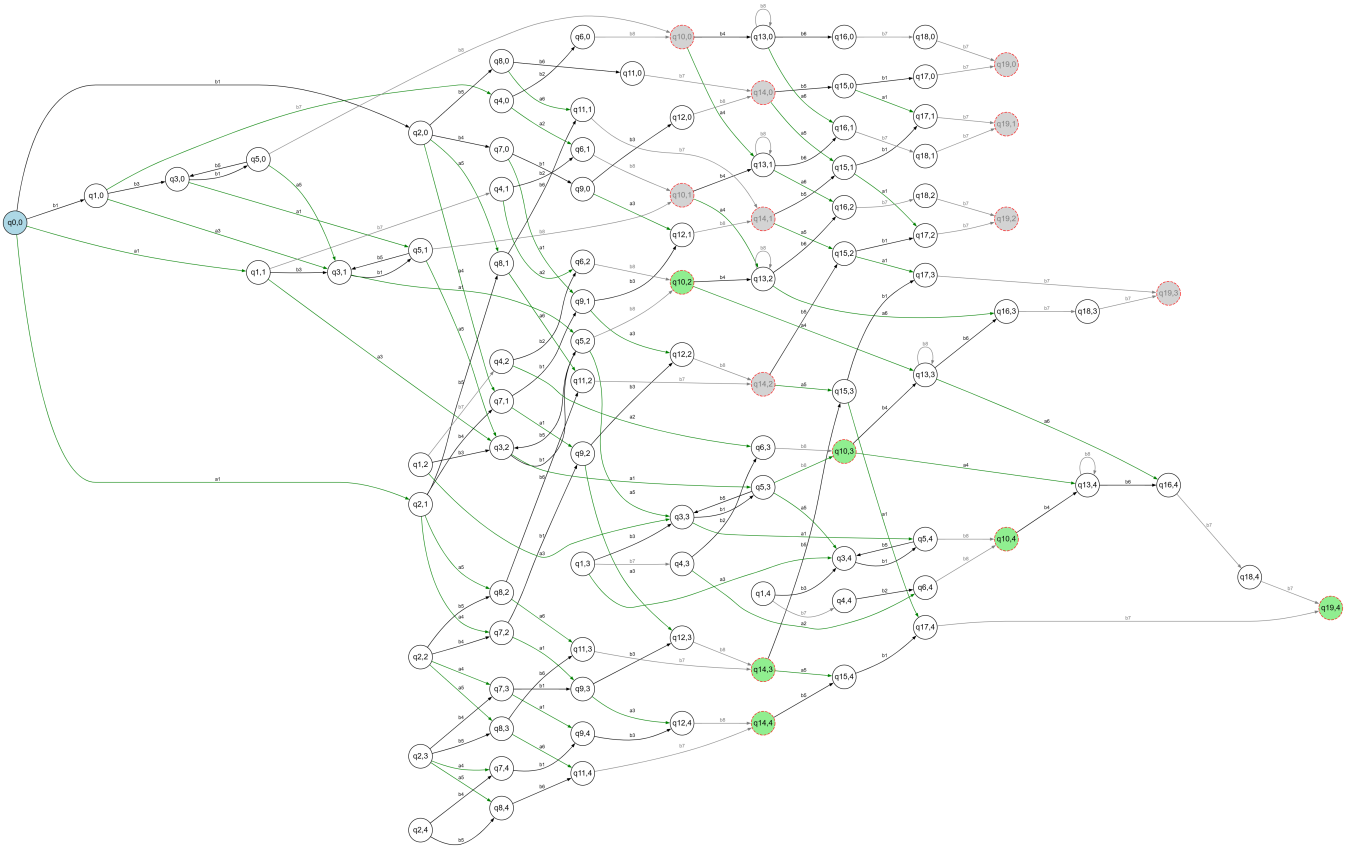

Fig. 2. Possible path traversal routes.

is given by $\{e_1, e_2, \ldots, e_8\}$*, where* e_1–e_6 *are protectable events, while* e_7 *and* e_8 *are unprotectable.*

Under the classical SCT framework, the construction of the security automaton for N *is required. Fig. 2 depicts partial possible path traversal routes. In this figure, the green transitions represent sub paths where events are protected or unprotected, and the shaded states identify secret states that do not satisfy the security requirement* $\beta(q)$*. Constructing the security automaton is a challenging task due to the significant expansion of the state space, even for this 20-state automaton. For larger systems, constructing it becomes practically impossible. We propose a RL based framework for secret protection which bypasses the complex automaton construction and synchronization required by SCT. The following sections introduce the details of the framework and provide experimental results for this case study.*

C. Reinforcement Learning

Our objective is to develop a RL-based approach for solving the problem of secret protection in DES while minimizing protection cost. The secret protection problem in DES can be effectively modeled as a Markov Decision Process (MDP) with five-tuple: $\langle S, A, P, R, \gamma \rangle$, where S is the state set, A is the action set, P is the state transition probability, R is the reward function, and γ is the discount factor, with $0 \leq \gamma \leq 1$.

In the learning process, the agent's objective is to find an optimal policy such that the expected cumulative reward over

time is maximized through trial and error. The Bellman optimality equation for the state-action value function $Q^*(s,a)$ is used to evaluate the maximum expected cumulative reward when the agent takes action a in state s, which is given by

$$Q^*(s,a) = \sum_{s' \in S} P(s,a,s') \left[R(s,a,s') + \gamma \max_{a' \in A} Q^*(s',a') \right]$$

where $s \in S$ is the system state, and $a \in A$ is the selected action at state s.

The pursuit of the optimal policy is a central theme RL, supported by foundational approaches such as Q-learning [12] and State-Action-Reward-Stat-Action (SARSA) [13]. Building upon these, Deep Q-Networks [14] and related architectures have further demonstrated the ability to achieve optimality in more complex, large-scale environments.

III. RL-based Secret Protection Framework

Traditional secret protection strategies typically rely on constrcution of preconstructed security automata to make protection decisions by classical SCT. As a result, such approaches are prone to state-space explosion and often suffer from rigid policy with limited adaptability when applied to complex systems. To overcome these limitations, this work introduces a RL-based framework, in which the secret protection problem is formulated as a dynamic MDP, enabling adaptive policy synthesis and online optimization.

RL allows the system to learn optimal protection strategies through interaction with the environment by taking actions (such as performing policy of security checks) to protect secret states within the system, as illustrated in Fig. 3. Each time the agent randomly executes an action based on the current state, the environment provides immediate feedback in the form of a reward. Through continuous interaction with the environment, the agent autonomously learns to balance secret assurance and intervention cost minimization.

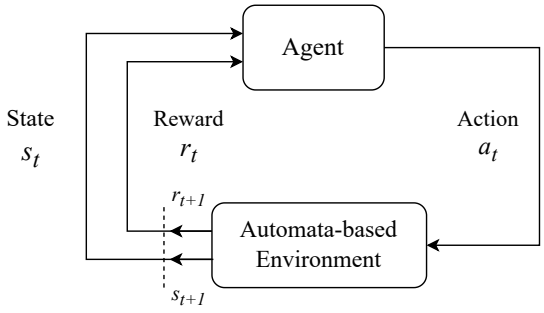

Fig. 3. RL-based secret protection framework.

A. Problem Formulation

An agent is trained in automata using RL. Starting from the initial state of the automaton, it follows a path through the system and, at each reached state, randomly generates an action to protect or unprotect with the protectable event. During RL training, the nondeterministic transitions of the NFA are modeled as stochastic state transitions in a MDP, where one of the possible states is sampled according to the transition dynamics. During learning, the framework continuously checks whether the protected action generated could allow an to reach secret states before enough security checks have been completed. If a potential secret state is detected, the environment penalizes the corresponding action, discouraging unsafe behavior during learning rather than explicitly forbidding it in advance. Through repeated interaction with the environment, actions that risk revealing secret states receive lower expected rewards and are gradually eliminated from the policy. As a result, the learning process implicitly converges to a secret protection strategy that balances secret protection performance and cost.

B. Construction of the State Set S

In an automaton $\mathcal{N} = (Q, E, \delta, q_0)$, for each $q_i \in Q$, $h(q_i)$ denotes the cumulative number of security checks applied at q_i. The agent's state at time t is a tuple:

$$s_t = (q_t, h_t) \in S \tag{3}$$

where $q_t \in Q$ is the current automaton state, and $h_t \in \mathbb{N}_0$ is the count of security checks performed along the execution path from the initial state $s_0 = (q_0, 0)$. This representation captures both the automaton state and the progress of security checks.

In the context of agent training, we identify certain states that are reached without a sufficient number of security checks. These are defined as *forbidden states*, denoted by $S_{fb} \subset S$, and are formulated as:

$$S_{fb} = \{(q,h) \in S \mid q \in Q_s, h < \beta(q)\} \tag{4}$$

where $Q_s \subseteq Q$ is the set of secret states and $\beta(q)$ is the pre-defined security requirement for each $q \in Q_s$. During the learning process, entering any state $(q,h) \in S_{fb}$ triggers a penalty to suppress such unsafe behaviors, ensuring the final policy adheres to the required security constraints. Any augmented state (q,h) where the cumulative security checks h fail to meet the required level $\beta(q)$ is strictly prohibited.

C. Definition of Actions in RL

The action space A is defined based on the protection ability of events of the system. For each protectable event $e \in E_p$, the agent randomly generates an action between a protection action a_e and an unprotection action b_e. For unprotectable events $e \in E_{up}$, only the release action b_e is available. For any specific event e occurring in the current state, the available actions $\vartheta(q)_e$ are constrained as follows:

$$A = \{\vartheta(q)_e\} = \begin{cases} \{a_e, b_e\}, & \text{if } e \in E_p \\ \{b_e\}, & \text{if } e \in E_{up} \end{cases} \tag{5}$$

In this paper, the action is defined as the random protection selection over the protectable events at the current state, rather than over all events in the system, which effectively reduces the action space.

D. Reward Function Design

To guide the agent toward an optimal secret protection policy, we define the reward function $R(s, a, s')$ to balance security requirements, operational costs, and system efficiency. The reward is formulated as follows:

$$R(s,a,s') = \begin{cases} R_{\text{fail}}, & s' \in S_{fb} \\ R_{\text{safe}} + R_{\text{cost}} + R_{\text{over}}, & q' \in Q_s \wedge h \geq \beta(q') \end{cases} \tag{6}$$

In this reward function, R_{fail} is defined as a significant negative penalty to strictly suppress security violations. Conversely, R_{safe} provides a substantial positive reinforcement to reward compliant access to secret states. Each security check a_e incurs an operational cost R_{cost}, which is assigned a small negative value to encourage the agent to minimize intrusive protection mechanisms without compromising safety. Furthermore, to eliminate redundant checks, an additional penalty R_{over} is applied when the security check h' exceeds the specific requirement security level $\beta(q')$. This design ensures the agent converges toward a cost-effective and precise protection policy.

E. Optimal Policies Learning

The agent starts from $s_0 = (q_0, 0)$ and explores the environment by deciding whether to protect or release each triggered event. These actions drive the system's evolution and update the security counter h. During training, the agent receives a severe penalty for entering forbidden states and a reward for compliant access to secret states. A minor penalty is also applied for excessive checks to ensure policy efficiency.

Algorithm 1: SARSA-based Secret Protection Strategy Training

Input: State space S, action space $A = \{a_e, b_e \mid e \in E\}$, learning rate α, discount factor γ, exploration rate ϵ
Output: Optimal policy $\pi^*(s) = \arg\max_a Q(s,a)$

Initialize $Q(s,a) = 0$ for all $s \in S$, $a \in A$
for *each episode* $ep = 1, \ldots, N$ **do**
  $s \leftarrow (q_0, 0)$
  Generate action a from s using the ϵ-greedy policy based on $Q(s, \cdot)$
  while s *is not a secret state* **do**
    Execute action a, observe reward R and next state s'
    Generate next action a' from s' using the ϵ-greedy policy based on $Q(s', \cdot)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[R + \gamma Q(s', a') - Q(s,a)\right]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  Update ϵ
return *Optimal policy* $\pi^*(s)$

To evaluate different learning mechanisms, we compare Q-learning and SARSA. Q-learning (off-policy) focuses on the maximum potential value to find an optimal policy. Conversely, SARSA (on-policy) updates based on actual executed actions, offering higher robustness in defense scenarios. The training procedure for our proposed protection policy using the SARSA algorithm is formalized as Algorithm 1. We analyze both algorithms regarding convergence speed, rewards, and policy precision in the next section about experiment to identify the most effective secret protection algorithm.

IV. Experimental Study

This section presents an experimental study to validate the proposed RL-based approach for secret protection strategies. A representative example is used to demonstrate how the agent learns to enforce secret protection through interaction with the augmented environment. The example presentation, parameter settings, experimental results are reported and analyzed in detail.

A. Parameter Settings

All experiments were implemented in Python 3.10 using NumPy, PyTorch, and the OpenAI Gym interface. Experiments were conducted on a computer with an Intel Core Ultra 7 processor and 32 GB RAM. To ensure a fair comparison, Q-learning and SARSA used identical hyperparameters. Detailed reward settings and training parameters are listed in Table I.

TABLE I
HYPERPARAMETER CONFIGURATIONS FOR THE RL-BASED SECRET PROTECTION ALGORITHMS

Parameter (Symbol)	Value/Setting
Failure Penalty (R_{fail})	-5000
Success Reward (R_{safe})	500
Action Cost (R_{cost})	-10
Over-protection Penalty (R_{over})	-20
Number of Episodes (N_{ep})	20,000
Number of Paths per Episode (N_{path})	30
Discount Factor (γ)	0.99
Initial Exploration (ϵ_{start})	1.0
Final Exploration (ϵ_{end})	0.01
Exploration Decay Rate (ϵ_{decay})	0.9997

B. Experimental Results and Discussions

Both algorithms show excellent convergence after 20,000 episodes of training. The final success rate for secret protection reaches 100 percent for both methods. The final success rate for secret protection reaches 100 percent for both methods, as shown in Fig. 4. The policies generated by both Q-learning and SARSA are feasible and effective, as they both satisfy the safety requirements $\beta(q)$. However, the output policy differ. The Q-learning policy applies protection at $\vartheta(q_0) = e_1$, $\vartheta(q_3) = e_1$, $\vartheta(q_4) = e_2$, $\vartheta(q_7) = e_1$, $\vartheta(q_9) = e_3$, $\vartheta(q_2) = e_5$, $\vartheta(q_8) = e_6$, $\vartheta(q_{10}) = e_4$, $\vartheta(q_{13}) = e_6$, and $\vartheta(q_{15}) = e_1$. In contrast, the SARSA policy chooses protection at $\vartheta(q_0) = e_1$, $\vartheta(q_1) = e_3$, $\vartheta(q_4) = e_2$,

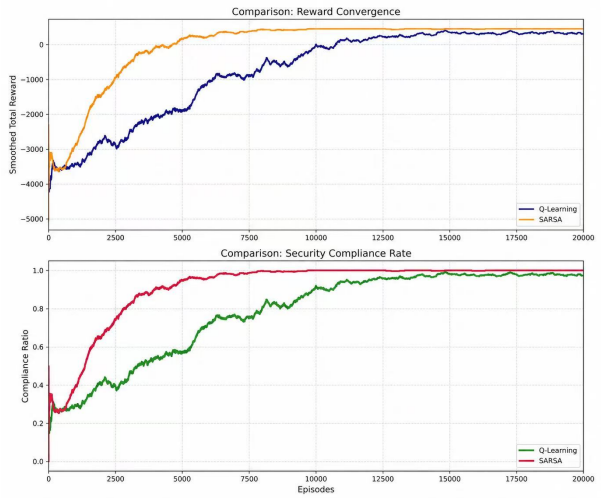

Fig. 4. Training performance comparison between Q-learning and SARSA.

$\vartheta(q_2) = e_4$, $\vartheta(q_7) = e_1$, $\vartheta(q_2) = e_5$, $\vartheta(q_8) = e_6$, $\vartheta(q_{10}) = e_4$, $\vartheta(q_{13}) = e_6$, and $\vartheta(q_{14}) = e_5$.

The difference between SARSA and Q-learning arises from their update logic. SARSA updates values based on the action actually taken, making it more conservative and directly incorporating the risk of exploration penalties into Q values, which leads to safer and more stable learning. In contrast, Q-learning assumes the best possible future action, encouraging risk-taking and higher penalties during the early stages of training.

Fig. 4 shows training results for a 20-state NFA. Both algorithms achieve increasing rewards and security compliance rate over time, reflecting the agent's ability to reach secret states while respecting security rules. SARSA reaches 100% compliance and stable rewards after 7,500 episodes with smooth curves, whereas Q-learning converges more slowly, stabilizing only after 15,000 episodes and exhibiting significant fluctuations, indicating frequent entry into forbidden states during exploration. This comparison highlights SARSA's superior efficiency and reliability for learning secret protection policies in this setting.

Classical SCT requires the explicit construction of security automata, which guarantees optimality but is NP-hard and grows exponentially with system size. As illustrated in Fig. 2, even for a system with only 20 states, the security automaton is already considerable, demonstrating the practical burden of explicit construction. In contrast, RL avoids building security automata and instead synthesizes dynamic secret protection policies fast through direct interaction with the environment. In our experiments, training on the 20-state NFA took 20 seconds, demonstrating a favorable trade-off between offline training cost and online scalability. These results confirm that RL can effectively handle large-scale DES, providing a practical solution to the secret protection problem.

V. CONCLUSIONS

This paper presents a RL framework for dynamic secret protection in DES. The agent makes step by step action at each state to balance security and operational costs while getting rewards. This interaction allows the agent to navigate the automaton space and avoid forbidden states where security requirements are not met. The final result is an optimal protection policy that ensures safety with minimum operational cost. We compared Q-learning and SARSA algorithms. Experimental results show that while both reach a high success rate, SARSA is more efficient and robust. Therefore, SARSA is better suited for training this secret protection framework. Future research will address more realistic challenges. We will study cases with varying costs for security checks, including scenarios where different secret states or events incur different operational penalties. We also plan to investigate secret protection in incomplete or partially observed models, where the agent has limited knowledge of the system state. These areas are challenging for classical supervisory control theory and will provide further insight into practical secret protection strategies.

REFERENCES

[1] R. Liu, T. Li, S. Hu, A. M. Mangini, and M. P. Fanti, "Securing networked discrete event systems for diagnosability under attacks," *IEEE Transactions on Automation Science and Engineering*, vol. 23, pp. 7409–7423, 2026.

[2] A. Saboori and C. N. Hadjicostis, "Notions of security and opacity in discrete event systems," in *46th IEEE Conference on Decision and Control*, 2007, pp. 5056–5061.

[3] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[4] Z. Ma and K. Cai, "Optimal secret protections in discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 67, no. 6, pp. 2816–2828, 2021.

[5] 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, no. 7, pp. 4380–4395, 2023.

[6] T. Ushio and T. Yamasaki, "Supervisory control of partially observed discrete event systems based on a reinforcement learning," in *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483)*, vol. 3. IEEE, 2003, pp. 2956–2961.

[7] T. Yamasaki and T. Ushio, "Decentralized supervisory control of discrete event systems based on reinforcement learning," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 88, no. 11, pp. 3045–3050, 2005.

[8] K. M. Zielinski, L. V. Hendges, J. B. Florindo, Y. K. Lopes, R. Ribeiro, M. Teixeira, and D. Casanova, "Flexible control of discrete event systems using environment simulation and reinforcement learning," *Applied Soft Computing*, vol. 111, p. 107714, 2021.

[9] R. Liu, A. M. Mangini, and M. P. Fanti, "Deep reinforcement learning for near-optimal control sequence in discrete event systems," *IEEE Transactions on Automation Science and Engineering*, vol. 23, pp. 8086–8096, 2026.

[10] J. Yang, K. Tan, L. Feng, and Z. Li, "A model-based deep reinforcement learning approach to the nonblocking coordination of modular supervisors of discrete event systems," *Information Sciences*, vol. 630, pp. 305–321, 2023.

[11] C. N. Hadjicostis, *Estimation and inference in discrete event systems*. Springer, 2020.

[12] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[13] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT Press Cambridge, 1998, vol. 1, no. 1.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.