On the Secret Protection Problem in Discrete-Event Systems

Tomáš Masopust and Jakub Večeřa

Faculty of Science, Palacky University Olomouc, Czechia

Abstract

The secret protection problem (SPP) seeks to synthesize a minimum-cost policy ensuring that every execution from an initial state to a secret state includes a sufficient number of protected events. The problem is solvable in polynomial time under the assumption that transitions are uniquely labeled. When this assumption is relaxed, the problem becomes weakly NP-hard. We first strengthen the result by showing that the problem is strongly NP-hard even if all parameters are restricted to binary values. We then propose a formulation of SPP as an integer linear programming (ILP) problem, and empirically evaluate the scalability and effectiveness of the ILP-based approach on relatively large systems. Finally, we examine the complexity of a variant of SPP in which only distinct protected events contribute to clearance and show that its decision version is Σ_2^P-complete.

Key words: Discrete event systems, secret protection problem, finite automata, verification, complexity.

1 Introduction

Modern systems frequently store and process sensitive information, from personal data to financial credentials, which must remain confidential to unauthorized users. This concern led to the formulation of the *Secret Protection Problem* (SPP), a framework for guaranteeing that users complete a predefined number of security checks before gaining access to sensitive parts of a system, see Matsui and Cai (2019) and Ma and Cai (2022, 2025) for further details, including discussions on the motivation and applications.

In the SPP framework, a system is modeled as a finite automaton with a designated set of *secret states*. Each secret state is associated with a *security level*, defined as a non-negative integer quantifying the minimum number of security checks required to reach that state. The event set is partitioned into *protectable* and *unprotectable* events. Each protectable event is assigned a non-negative *cost*, representing the implementation effort of the corresponding security check, and a *clearance level*, indicating the number of clearance units granted upon its execution. An event is *protected* if a security check has been placed on it. The objective of SPP is to synthesize a *protecting policy* of minimal cost such that every execution path from an initial state to a secret state includes a number of protected events that meets or exceeds the security level assigned to the respective secret state. This problem is referred to as Parikh-SPP in this paper.

If all transitions are labeled with unique events, Ma and Cai (2022) showed how to solve Parikh-SPP via a reduction to a supervisory control problem or to a max-flow network problem, providing thus a polynomial-time algorithm. If the assumption is relaxed, Ma et al. (2024) proved that Parikh-SPP becomes weakly NP-hard. However, they left two questions open: *Is the problem strongly NP-hard?* and *What is the complexity of a special case of Parikh-SPP, called the uniform Parikh-SPP (Parikh-SPP-U) in which all events have a uniform clearance level of one?*

We answer both problems by proving that Parikh-SPP-U is strongly NP-hard, and that it remains NP-hard even if the ranges of the cost function, the clearance function, and the security requirement function are restricted to be binary. Furthermore, we show that no sub-exponential-time algorithm exists for Parikh-SPP unless the Exponential Time Hypothesis of Impagliazzo and Paturi (2001) fails.

These intractability results raise a fundamental question: *How can Parikh-SPP be practically solved for real-world systems?* Although Ma et al. (2024) proposed a polynomial-time heuristic algorithm, its practical applicability remains limited—instances with only 100 states can require up to five minutes to solve, and the algorithm offers no guarantees on the quality of the solution. This gap highlights the need for a robust and efficient method to solve Parikh-SPP. We address this challenge by formulating Parikh-SPP as an integer linear programming (ILP) problem and demonstrating its scal-

⋆ Corresponding author: T. Masopust. Supported by the Palacky University under grants IGA PrF 2025 018 and IGA PrF 2026 026.

Email addresses: tomas.masopust@upol.cz (Tomáš Masopust), jakub.vecera01@upol.cz (Jakub Večeřa).

ability and effectiveness in finding exact solutions for relatively large systems across an extensive set of benchmarks, including both real-world and randomly generated data.

Finally, we analyze the complexity of a variant of SPP in which only distinct protected events contribute to the clearance, referred to as Indicator-SPP. This variant is motivated by real-world multi-factor authentication schemes. We prove that its decision version is Σ_2^P-complete, placing it at the second level of the polynomial-time hierarchy. In contrast to Parikh-SPP, the existence of a practically efficient algorithm for Indicator-SPP remains an open problem.

2 Preliminaries

We assume that the reader is familiar with automata theory (Sipser, 2012). The sets of non-negative integers and non-negative real numbers are denoted by $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$, respectively. For a set S, the cardinality of S is denoted by $|S|$, and its power set by 2^S. If $S = \{x\}$ is a singleton, we often write x instead of $\{x\}$. An alphabet Σ is a finite nonempty set of *events*. A string over Σ is a finite sequence of events from Σ. The set of all strings over Σ is denoted by Σ^*, and the empty string is denoted by ε. A language L over Σ is a subset of Σ^*. For a string $u \in \Sigma^*$ and an event $a \in \Sigma$, the number of occurrences of a in u is denoted by $|u|_a$.

A *nondeterministic finite automaton* (NFA) is a quintuple $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of states, Σ is an input alphabet, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of accepting states, and $\delta\colon Q \times \Sigma \to 2^Q$ is a transition function that can be inductively extended to $\delta\colon 2^Q \times \Sigma^* \to 2^Q$. The language generated by $\mathcal{A}$ is the set $L(\mathcal{A}) = \{w \in \Sigma^* \mid \delta(I, w) \neq \emptyset\}$, and the language accepted by $\mathcal{A}$ is the set $L_m(\mathcal{A}) = \{w \in \Sigma^* \mid \delta(I, w) \cap F \neq \emptyset\}$.

We briefly review the concepts from complexity theory, see Arora and Barak (2009). A *decision problem* is a yes–no question defined over instances of a problem. The problem is *decidable* if there is an algorithm that determines, for every instance, whether the answer is yes or no. Decidable problems are categorized into complexity classes based on the time or space required to solve them. The classes P and NP consist of problems solvable by deterministic and nondeterministic polynomial-time algorithms, respectively. A problem is NP-complete if it satisfies two conditions: (i) Membership—the problem belongs to NP, and (ii) Hardness—every problem in NP can be reduced to it in deterministic polynomial time. While P $\subseteq$ NP, whether the inclusion is strict is one of the most important open problems in computer science.

An NP-complete problem is *weakly* NP-complete if it is solvable in pseudo-polynomial time, i.e., in time polynomial in the numeric value of the input rather than in the length of the input (the number of bits representing it). A problem is *strongly* NP-complete if it remains NP-hard even if the data are encoded in unary, i.e., independent of the numerical values of the data, see Garey and Johnson (1979). Hereafter, NP-complete means strongly NP-complete unless stated otherwise.

The levels of the *polynomial-time hierarchy* (PH) are defined as follows: $\Sigma_0^P = \Pi_0^P = \mathrm{P}$, and for $k \geq 0$, the class Σ_{k+1}^P contains problems solvable by a nondeterministic polynomial-time Turing machine with access to an oracle for a problem in Σ_k^P. Analogously, Π_{k+1}^P is the class of problems whose complements are in Σ_{k+1}^P. In particular, $\Sigma_1^P = \mathrm{NP}$ and $\Pi_1^P = \mathrm{coNP}$, that is, coNP is the class of problems whose complements are in NP. It is widely believed that the hierarchy is strict, see Stockmeyer (1976) for more details.

A *Boolean formula* consists of variables, logical connectives ($\wedge$, $\vee$, $\neg$), and parentheses. A *literal* is a variable or its negation. A *clause* is a disjunction of literals. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. If each clause contains at most k literals, the formula is in k-CNF. Analogously, a *conjunct* is a conjunction of literals, and a formula is in *disjunctive normal form* (DNF) if it is a disjunction of conjuncts. If each conjunct contains at most k literals, the formula is in k-DNF. A formula is *satisfiable* if there is an assignment of 1 and 0 to the variables such that the formula evaluates to 1. For $k \geq 1$, the k-CNF *Boolean satisfiability problem* (k-SAT) asks whether a given formula in k-CNF is satisfiable. If a formula in k-CNF has n variables, enumerating possible truth assignments results in an $O(2^n n^k)$-time algorithm for k-SAT. *Quantified Boolean formulas* (QBFs) extend Boolean formulas by allowing quantification over variables.

3-SAT is NP-complete, and despite extensive efforts, no sub-exponential-time algorithm has been found to date. The lack of progress led to the formulation of the exponential time hypothesis (ETH), which posits that 3-SAT cannot be solved in sub-exponential time $2^{o(n)}$, where n is the number of variables in the 3-CNF formula; recall that $o(g(n))$ denotes the class of functions $f(n)$ that grow asymptotically much slower than $g(n)$, i.e., $\lim_{n\to\infty} f(n)/g(n) = 0$.

Hypothesis 1 (ETH, Impagliazzo and Paturi (2001))**.** There is a $\lambda > 0$ such that 3-SAT cannot be solved in time $O(2^{\lambda n})$, where n is the number of variables in the formula.

Note that ETH allows for algorithms that solve 3-SAT in time $O(c^n)$ for $c < 2$. In fact, the current fastest algorithm by Paturi et al. (2005), improved by Hertli (2014), runs in time $O^*(1.30704^n)$; we use the soft big-O notation O^* to disregard polynomial factors $n^{O(1)}$ in the big-O notation; e.g., we can write the complexity $O(2^n n^k)$ as $O^*(2^n)$.

3 The Secret Protection Problem

In this paper, the formulation of the problem is intentionally simplified for clarity of our generalization. For the original treatment, we refer to Ma and Cai (2022) or Ma et al. (2024).

Let $\mathcal{A} = (Q, \Sigma, \delta, I, S)$ be an NFA. States of S are called *secret states*. A *security requirement* is a function $\ell \colon Q \to \mathbb{N}$ that assigns a non-negative security level $\ell(q)$ to each secret state $q \in S$. Intuitively, the security level specifies the minimum number of *protected events* that must be traversed to reach state q from any initial state. We assume that initial states are not secret, i.e., $I \cap S = \emptyset$.

The event set Σ is partitioned into *protectable events* Σ_p and *unprotectable events* Σ_{up}; we use the notation $\Sigma = \Sigma_p \uplus \Sigma_{up}$. The protectable events are characterized by two functions:

- The *clearance function* $\gamma \colon \Sigma_p \to \mathbb{N}$ specifying the number of clearance units granted upon the execution of a protectable event.
- The *cost function* $c \colon \Sigma_p \to \mathbb{R}_{\geq 0}$ assigning a cost to the implementation of protection for each protectable event.

For every protectable event $a \in \Sigma_p$, let $\chi_a \colon \Sigma^* \to \mathbb{N}$ be a function that assigns a natural number to each string over Σ. We denote by $\chi = \{\chi_a \mid a \in \Sigma_p\}$ the family of these functions. In this paper, we focus on two specific instances:

(1) the *Parikh function* $\chi_a(w) = |w|_a$, which counts every occurrence of a in w, and
(2) the *indicator function*

$$\chi_a(w) = \begin{cases} 1 & \text{if } a \text{ occurs in } w, \\ 0 & \text{otherwise,} \end{cases}$$

which counts each protectable event in w only once.

A *protecting policy* $\mathcal{P} \subseteq \Sigma_p$ is a subset of protectable events. A policy $\mathcal{P}$ is χ*-valid* if, for every string $w \in L(\mathcal{A})$ that leads to a secret state $q \in S$,

$$\sum_{a \in \mathcal{P}} \Big(\gamma(a) \cdot \chi_a(w) \Big) \geq \ell(q) \,.$$

The *cost of a policy* $\mathcal{P}$, denoted by $C(\mathcal{P})$, is the sum of the costs of all events included in the policy:

$$C(\mathcal{P}) = \sum_{a \in \mathcal{P}} c(a) \,.$$

A policy is χ*-optimal* if it is χ-valid and no other χ-valid policy has a lower cost. The *secret protection problem under* χ*-optimality* is defined as follows.

Definition 2 (χ-SPP)**.** Given a system $\mathcal{A} = (Q, \Sigma, \delta, I, S)$ with $\Sigma = \Sigma_p \uplus \Sigma_{up}$, along with a security requirement ℓ, a clearance function γ, and a cost function c, the objective is to determine a χ-optimal protection policy $\mathcal{P} \subseteq \Sigma_p$.

A special case of χ-SPP, the *uniform* χ*-SPP* (χ-SPP-U), simplifies the clearance mechanism by assuming that the clearance level for every protectable event is uniformly set to one, i.e., $\gamma(a) = 1$ for all $a \in \Sigma_p$.

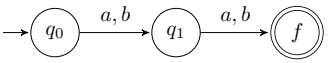

Figure 1. An instance of Parikh-SPP but not of Indicator-SPP.

We now formulate the decision versions, which generalize the decision version presented by Ma et al. (2024).

Definition 3 (BC-χ-SPP)**.** The *budget-constrained* χ*-SPP* (BC-χ-SPP) asks whether, given a system $\mathcal{A}$ over $\Sigma = \Sigma_p \uplus \Sigma_{up}$, a security requirement ℓ, a clearance function γ, a cost function c, and a budget limit $W \in \mathbb{N}$, there is a valid protecting policy $\mathcal{P} \subseteq \Sigma_p$ such that the cost of the policy satisfies $C(\mathcal{P}) \leq W$. Analogously, we define BC-χ-SPP-U.

If χ consist purely of Parikh (resp. indicator) functions, we call the problem Parikh-SPP (resp. Indicator-SPP).

The concept of Indicator-SPP is intended to differentiate between distinct authentication methods, rather than to account for repeated use of the same method as used in Parikh-SPP. The following example highlights the difference between the two notions.

Example 4. We consider the automaton of Figure 1, where state f is the only secret state, with $\ell(f) = 2$, and both events a and b are protectable, with $c(a) = c(b) = \gamma(a) = \gamma(b) = 1$. Let the protecting policy be $\mathcal{P} = \{a, b\}$. Then $\mathcal{P}$ is Parikh-valid and, in fact, Parikh-optimal, since there are four sequences reaching state f, namely aa, ab, ba, and bb, and for each sequence $w \in \{aa, ab, ba, bb\}$, we have $\gamma(a) \cdot |w|_a + \gamma(b) \cdot |w|_b = 2 \geq \ell(f)$. On the other hand, no indicator-valid protecting policy exists for this scenario. Indeed, as shown by the following calculation $\gamma(a) \cdot \chi_a(aa) + \gamma(b) \cdot \chi_b(aa) = 1 < \ell(f)$, the sum of the clearance values is less than the required security threshold $\ell(f)$. Consequently, even the largest possible policy, $\mathcal{P} = \{a, b\}$, fails to meet the indicator-validity condition. Therefore, no protecting policy yields sufficient clearance to satisfy the security requirement under the indicator function. ⋄

4 Complexity of Parikh-SPP

We now show that Parikh-SPP(-U) is NP-hard even under the restricted setting where both the cost and clearance functions are uniformly set to one, and the security requirement is binary, with exactly one state having a non-zero requirement. To this end, we prove NP-hardness of the corresponding decision problem. Specifically, we show that BC-Parikh-SPP belongs to NP and that BC-Parikh-SPP-U is NP-hard.

Lemma 5. *BC-Parikh-SPP belongs to NP.*

Proof. Given an instance $\mathcal{A}$ over $\Sigma = \Sigma_p \uplus \Sigma_{up}$ of BC-Parikh-SPP, there are $2^{|\Sigma_p|}$ possible protecting policies. We can nondeterministically guess a correct policy $\mathcal{P}$ and verify, in polynomial time, that $\mathcal{P}$ is valid and satisfies the constraint $C(\mathcal{P}) \leq W$. To verify validity, we apply Dijkstra's

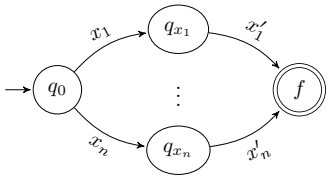

Figure 2. The first step of the reduction.

algorithm to compute shortest paths from each initial state to all secret states, taking into account the clearances of the events specified by the policy $\mathcal{P}$. If the clearance requirements of the secret states are met along the shortest path, then they are met along all paths, see Ma et al. (2024). □

Now, we prove NP-hardness.

Lemma 6. *BC-Parikh-SPP-U is NP-hard. It remains NP-hard even if the cost and clearance functions are constant with values one, and the security requirement satisfies $\ell(x) \in \{0, 1\}$ where only a single state is secret, i.e., has a non-zero security requirement.*

Proof. To establish NP-hardness, let φ be a Boolean formula in 3-CNF with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$. In each clause, we replace every literal $\neg x$ with x'. Without loss of generality, we may assume that the literals within each clause are distinct, that is, we may treat each clause as a set.

We construct an automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, f)$ with a single initial state q_0 and a single accepting (secret) state f, as follows. For each variable x, we add to $\mathcal{A}$ two events, x and x', both of which are protectable, along with two transitions

$$(q_0, x, q_x) \text{ and } (q_x, x', f)\,,$$

where q_x is a new state, see Figure 2. This step introduces n new states and $2n$ transitions.

Now, for each clause $C_i = \{\lambda_{i,1}, \ldots, \lambda_{i,k_i}\}$, where $1 \le i \le m$ and $1 \le k_i \le 3$, we add k_i transitions and $k_i - 1$ new states to $\mathcal{A}$. Specifically, we construct the sequence of transitions

$$(q_0, \lambda_{i,1}, q_{i,1}), \ldots, (q_{i,k_i-1}, \lambda_{i,k_i}, f)\,,$$

where $q_{i,1}, \ldots, q_{i,k_i-1}$ are newly introduced intermediate states, see Figure 3 for an illustration, as well as Examples 7 and 8 below. This step introduces at most $2m$ new states and $3m$ transitions.

We define the cost and clearance functions uniformly by setting $c(a) = 1$ and $\gamma(a) = 1$ for every event $a \in \Sigma$, meaning that all events in $\mathcal{A}$ are protectable. The security requirement function is defined by setting $\ell(f) = 1$ for a single designated secret state f, while all other states have security requirement zero, that is, they are not secret. Finally, we set the budget limit W to be the number of variables, that is, $W = n$.

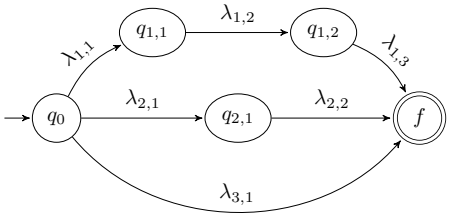

Figure 3. The encoding of three clauses $C_1 = \{\lambda_{1,1}, \lambda_{1,2}, \lambda_{1,3}\}$, $C_2 = \{\lambda_{2,1}, \lambda_{2,2}\}$, and $C_3 = \{\lambda_{3,1}\}$.

We now show that the formula φ is satisfiable if and only if there is a Parikh-valid protection policy $\mathcal{P}$ with $C(\mathcal{P}) \le n$.

If φ is satisfiable, then there is a truth assignment to the variables that satisfies all clauses. Define $\mathcal{P}$ as the set of all literals that are assigned the value 1. For each variable x, the set $\mathcal{P}$ contains exactly one of x and x', ensuring that the paths in Figure 2 satisfy the security requirement of state f. Since the assignment satisfies every clause of φ, the set $\mathcal{P}$ contains at least one literal from each clause, thereby ensuring that the paths in Figure 3 also meet the security requirement of state f. Therefore, the set $\mathcal{P}$ forms a protecting policy of cardinality n, and thus satisfies the cost constraint $C(\mathcal{P}) \le n$.

Conversely, suppose that $\mathcal{P}$ is a Parikh-valid protecting policy with $C(\mathcal{P}) \le n$. Since $\mathcal{P}$ must secure every path from q_0 to f, the structure in Figure 2 ensures that $\mathcal{P}$ contains at least one of the literals x or x' for each variable x. As there are n such paths and the total cost is bounded by n, it follows that $\mathcal{P}$ contains exactly n events. Therefore, for each variable x, the policy includes exactly one of x and x', but not both. This selection induces a consistent truth assignment for the variables of the formula φ: inclusion of x in $\mathcal{P}$ corresponds to assigning $x = 1$, while inclusion of x' corresponds to $x = 0$. Moreover, since $\mathcal{P}$ intersects all paths from q_0 to f in Figure 3, which correspond to the clauses of φ, the assignment satisfies every clause. Therefore, the policy $\mathcal{P}$ encodes a satisfying truth assignment for φ. □

The following examples illustrate the construction from the proof of Lemma 6. In the first example, we consider a satisfiable formula.

Example 7. Let $\varphi = (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$ be a formula over variables x, y, z. The assignment $x = y = z = 1$ satisfies both clauses, and therefore φ is satisfiable. We construct an automaton $\mathcal{A}$ with events $\Sigma = \{x, x', y, y', z, z'\}$, all of which are protectable, i.e., $\Sigma_p = \Sigma$, and with the set of states $Q = \{q_0, q_x, q_y, q_z, q_{1,1}, q_{1,2}, q_{2,1}, q_{2,2}, f\}$. For the variables x, y, z, we define the transitions (q_0, x, q_x), (q_x, x', f), (q_0, y, q_y), (q_y, y', f), (q_0, z, q_z), (q_z, z', f), and for the clauses $C_1 = \{x, y', z\}$ and $C_2 = \{x', y, z'\}$, we define the transitions $(q_0, x, q_{1,1})$, $(q_{1,1}, y', q_{1,2})$, $(q_{1,2}, z, f)$, and $(q_0, x', q_{2,1})$, $(q_{2,1}, y, q_{2,2})$, $(q_{2,2}, z', f)$, see Figure 4 for an illustration. The cost and clearance functions are defined by $c(a) = \gamma(a) = 1$ for every $a \in \Sigma_p$, and the security requirement is given by $\ell(f) = 1$ and $\ell(q) = 0$ for all $q \neq f$. The

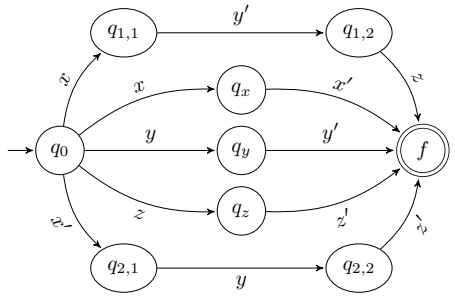

Figure 4. The encoding of $\varphi = (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$.

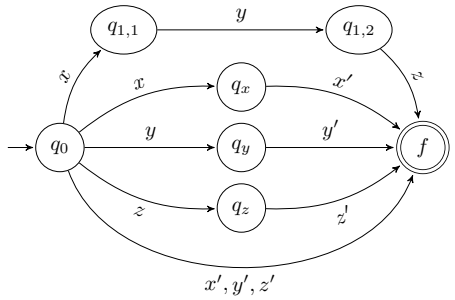

Figure 5. The encoding of $\psi = (x \vee y \vee z) \wedge (\neg x) \wedge (\neg y) \wedge (\neg z)$.

budget $W = 3$, the number of variables. Since φ is satisfiable, there exists a Parikh-valid protecting policy $\mathcal{P}$. Indeed, the set $\mathcal{P} = \{x, y, z\}$ intersects every path from q_0 to f and satisfies the cost constraint $C(\mathcal{P}) = 3 \leq W$. Therefore, $\mathcal{P}$ is a Parikh-valid protecting policy. ◇

In the second example, we consider an unsatisfiable formula.

Example 8. Let $\psi = (x \vee y \vee z) \wedge (\neg x) \wedge (\neg y) \wedge (\neg z)$. The automaton obtained by the construction is shown in Figure 5. The alphabet consists of six events: x, x', y, y', z, z'. The edge labeled with x', y', z' represents three parallel transitions from q_0 to f, corresponding to the three clauses $C_2 = \{x'\}$, $C_3 = \{y'\}$, and $C_4 = \{z'\}$. The states include the initial state q_0, the secret state f, the three states q_x, q_y, and q_z introduced for the variables, and the two intermediate states created for the first clause. Each event has cost and clearance 1, with f being the only secret state with $\ell(f) = 1$. The budget limit is $W = 3$. In this automaton, any protecting policy within the budget must include exactly one of the events x and x', exactly one of the events y and y', and exactly one of the events z and z'. This requirement leaves at least one path from q_0 to f that avoids the protected events. Consequently, no policy can simultaneously satisfy both the security requirement and the budget constraint. ◇

The preceding two lemmas yield the following main result.

Theorem 9. *BC-Parikh-SPP and BC-Parikh-SPP-U are NP-complete, and Parikh-SPP and Parikh-SPP-U are NP-hard. The problems remain NP-hard even if the system has a unique initial state, the cost and clearance functions are constant (i.e., $c(x) = 1$ and $\gamma(x) = 1$), and the security requirement satisfies $\ell(x) \in \{0, 1\}$, with exactly one state designated as secret.*

Proof. Lemma 5 shows that BC-Parikh-SPP, and hence also BC-Parikh-SPP-U, belong to NP, while Lemma 6 proves that the considered variants of the problem are NP-hard. Strong NP-hardness follows because the reduction of Lemma 6 produces instances in which all numerical data are constant. □

Lemma 5 suggests an algorithm whose runtime is exponential in the size of the alphabet of protectable events. We now show that, under current knowledge in complexity theory, this algorithm is optimal. Specifically, we show that if any of the considered budget-constrained secret protection problems could be solved in sub-exponential time, then 3-SAT could also be solved in sub-exponential time.

Theorem 10. *There is no algorithm that solves BC-Parikh-SPP(-U) in time $2^{o(N)}$, where N is the number of protectable events in the system, unless ETH fails.*

Proof. Consider the reduction of 3-SAT from the proof of Lemma 6. If the input 3-CNF formula contains n variables, then the constructed system has $N = 2n$ protectable events. Suppose that there exists an algorithm solving BC-Parikh-SPP(-U) in time $2^{o(N)} = 2^{o(n)}$. Then, by reducing any instance of 3-SAT to an instance of BC-Parikh-SPP(-U) in polynomial time and applying this algorithm, we would obtain a sub-exponential-time algorithm for 3-SAT, contradicting the exponential time hypothesis. □

Remark 11. It is worth noticing that BC-Indicator-SPP(-U) is also NP-hard. In particular, the proof of Lemma 6 shows that BC-Indicator-SPP-U is NP-hard, which follows from the assumption that all literals within each clause are pairwise distinct. Consequently, there is no algorithm that would solve BC-Indicator-SPP(-U) in time $2^{o(N)}$, where N is the number of protectable events, unless ETH fails. We analyze the exact complexity of BC-Indicator-SPP(-U) in Section 6.

The intractability results characterize the inherent difficulty of the most challenging instances and do not necessarily reflect the performance on realistic inputs. They raise a fundamental question: *Can SPP be solved efficiently in practice?*

While Ma et al. (2024) introduced a polynomial-time heuristic for Parikh-SPP, its scalability and robustness remain limited. In practice, instances with around 100 states may require several minutes to solve, and the approach offers no guarantees on solution quality. In the next section, we present an ILP-based algorithm for Parikh-SPP and show that it performs efficiently even on large-scale systems. In contrast, whether Indicator-SPP admits a practically efficient solution remains an open question.

5 Solving Parikh-SPP via ILP

Integer Linear Programming (ILP) is used to determine the optimal outcome of a linear objective function, subject to linear constraints. Although the integrality constraints render ILP problems NP-hard, advances in algorithms and solvers have made it practical to solve large and complex instances, see Schrijver (1999). As a result, ILP has become a widely used tool for addressing a broad range of NP-hard optimization problems.

We design an algorithm that solves Parikh-SPP using an ILP solver. Given a candidate policy, its validity can be verified in polynomial time using Dijkstra's shortest paths algorithm (Ma et al., 2024). In particular, if all shortest paths satisfy the security requirements of secret states, then all paths do as well. Thus, we can examine the policy that includes all protectable events and determine, in polynomial time, whether it satisfies the security constraints. If it does, the SPP instance is solvable; otherwise, it is not. Hence, without loss of generality, we assume that the input instance to Algorithm 1 is solvable.

Algorithm 1 Parikh-SPP solver using ILP

Require: A solvable instance of Parikh-SPP.
Ensure: Optimal protecting policy $\mathcal{P}$.

```
Define binary variables x_e for each e ∈ Σ_p.
Set objective function: Minimize ∑_{e∈Σ_p} c(e) · x_e.
while true do
    Solve the current ILP model.
    Get current values of x_e.
    for each initial state i do
        ▷ Verify the current solution. ◁
        Compute the shortest paths from i to all other states using Dijkstra's algorithm, considering current values of x_e to adjust edge weights based on event clearance γ.
        for each secret state s do
            d(i, s) ← shortest distance from i to s
            if d(i, s) < ℓ(s) then
                add the constraint ∑_{e∈π} |π|_e · γ(e) · x_e ≥ ℓ(s) to the ILP model.
        if a new constraint was added to the ILP model then
            ▷ A violation of the solution was found, repeat the while-loop. ◁
            break (from outer for-loop)
        else
            ▷ The found solution is a valid policy. ◁
            return the current solution and terminate
```

The algorithm begins by introducing a binary variable for each protectable event: a value of 1 indicates that the event is included in the policy, and 0 indicates that it is not. The objective function, to be minimized, is defined as the total cost of the events selected for the policy, which is initially empty. An ILP solver is then used to compute a solution to the current ILP model. This solution is verified using Dijkstra's algorithm to ensure that all shortest paths satisfy the specified security requirements. If this condition holds, the solution is valid and the algorithm terminates.

Otherwise, there is a shortest path π that violates the security requirement of a secret state s. The algorithm then augments the ILP model with an additional constraint enforcing the security requirement along this path. Specifically, it ensures that the sum of the clearances of the events on π meets or exceeds the required security level $\ell(s)$. This is expressed as $\sum_{e\in\pi} |\pi|_e \cdot \gamma(e) \cdot x_e \geq \ell(s)$, where $|\pi|_e$ denotes the number of occurrences of event e in path π. After adding all such constraints to the ILP model, the algorithm proceeds to the next iteration of the while-loop. Since the instance is assumed to be solvable, the loop is guaranteed to eventually terminate with a valid solution.

We implemented Algorithm 1 in Python, using *PuLP* with the *CBC* solver. All experiments were run on an Ubuntu 24.04.3 LTS machine with 32 Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20 GHz and 246 GB RAM. We ran up to 24 threads in parallel. We considered two data sources. (1) We derived Parikh-SPP instances from public automata benchmarks. In these instances, all events are protectable with $\gamma(\cdot) = c(\cdot) = 1$ (this setting already makes the problem NP-hard), and every accepting state is treated as a secret state with security level 1. Trivially unsolvable cases were filtered out. (2) We generated random NFAs using the Tabakov-Vardi model (Tabakov and Vardi, 2005), using the MATA library (Chocholatý et al., 2024), with q states and the alphabet size k. For each combination of $q \in \{100, 500, 1000, 5000, 10000, 50000, 100000\}$ and $k \in \{2, 3, 5, 10\}$, we generated 100 random instances, varying the transition density uniformly between 0.8 and 5.0. Each generated NFA was converted to a Parikh-SPP instance in which all events were protectable, and $\gamma(a)$, $c(a)$ and $\ell(f)$ were drawn at random from $\{1, ..., 10\}$. If an instance turned out to be trivially unsolvable, it was regenerated. In total, we solved 2800 random instances.

For each instance, we logged the number of states, the optimal policy cost, the solver's status, the number of iterations, and the runtime. All instances, data, and scripts are available in the git repository `https://apollo.inf.upol.cz:81/vecerajakub/SPP-automata`.

For the real-based instances, Figure 6 plots the runtime versus the number of states. All instances were solved quickly. Since the instances are not native Parikh-SPP data, they rather test structural difficulty than domain-specific Parikh-SPP semantics.

For the random instances, our results show that the approach scales to NFAs with hundreds of thousands of states. Table 1 summarizes the median runtime performance over 100 runs for each size and alphabet combination (in seconds). Even for the largest instances with 100,000 states and 10 events, the median time was about 19 minutes, and the slowest 5% of the instances were solved within 44 minutes. At moderate

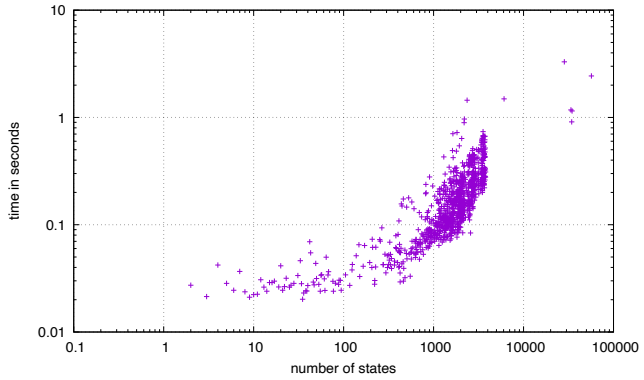

Figure 6. Runtime on real-derived Parikh-SPP instances.

Table 1
Median runtime in seconds across 100 runs.

q states	k=2	k=3	k=5	k=10
100	0.039	0.081	0.247	1.783
500	0.060	0.116	0.330	2.519
1,000	0.087	0.152	0.442	2.780
5,000	0.491	0.868	2.217	13.562
10,000	1.593	2.563	5.323	30.275
50,000	43.201	58.465	81.393	268.409
100,000	163.626	179.318	228.184	1133.861

sizes (e.g., 10,000 states with $k = 10$), the median runtime remained around 30 seconds.

For a fixed number of states, instances with a larger alphabet are generally harder, since more events and transitions increase the ILP size and the number of constraints generated. For example, for $q = 10{,}000$ the median runtime rises from 2.56s ($k = 3$) to 30.28s ($k = 10$), and for $q = 100{,}000$ from 179.32s ($k = 3$) to 1133.86s ($k = 10$). The few instances in the slowest 5% typically had the highest transition density, which leads to more complex shortest-path computations and more constraint-generation iterations.

One observation is the impact of multiple initial states on performance. For the randomly generated automata, the number of initial states grows with the model size (about 0.1% of states were initial in those models). We ran a separate experiment generating similar random instances but restricting each automaton to a single initial state. The performance on these instances was noticeably better, especially for large systems. Table 2 lists the median runtimes. We see that at 100,000 states and $k = 10$, the median runtime drops to about 175s under the single-initial-state restriction, compared to 1134s in the general case. This trend is illustrated in Figure 7, which compares the median running times for multi-initial vs. single-initial random instances. Intuitively, multiple initial states require the clearance constraints to be satisfied from each of those starting points, which yields a larger ILP and more constraints in the process of solving.

Table 2
Random instances with one initial state—median runtime in seconds across 100 runs.

q states	k=2	k=3	k=5	k=10
100	0.044	0.078	0.261	1.657
500	0.064	0.128	0.326	2.517
1,000	0.078	0.151	0.458	2.671
5,000	0.295	0.531	1.708	9.412
10,000	0.642	1.237	3.328	18.764
50,000	4.005	8.563	21.406	86.610
100,000	9.458	17.079	34.733	174.971

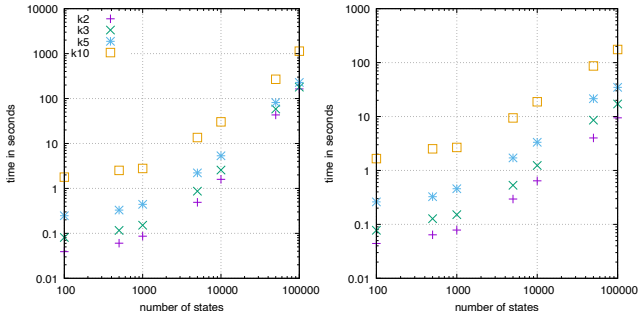

Figure 7. Median runtime on random instances with multiple initials (left) vs. a single initial state (right).

In summary, our ILP-based approach scales to 10^5 states and handles dense transitions. The runtime grows predictably with the number of states and alphabet size. Despite the underlying hardness of Parikh-SPP, these results indicate that Algorithm 1 provides an exact method for solving relatively large instances.

6 Complexity of Indicator-SPP

In this section, we discuss BC-χ-SPP where χ consists of indicator functions. We call this variant BC-Indicator-SPP, and prove that it is Σ_2^P-complete. To this end, we first show that verifying indicator-validity is computationally more challenging than verifying Parikh-validity.

Proposition 12. *Given an automaton $\mathcal{A}$ and a policy $\mathcal{P}$, to verify whether $\mathcal{P}$ is indicator-valid is coNP-complete.*

Proof. To verify indicator-validity is to check there is no secret state f and execution from an initial state to f whose set of symbols $S \subseteq \mathcal{P}$ along the execution satisfies $\sum_{a \in S} \gamma(a) < \ell(f)$. To verify the complement is an NP search: we guess a state f and a sequence w of length at most the number of states and compute the weight of w in polynomial time. Thus, to verify indicator-validity is in coNP.

To establish coNP-hardness of the indicator-validity verification, let φ be a Boolean formula in 3-CNF with n variables $X = \{x_1, \ldots, x_n\}$ and m clauses $C_1, \ldots, C_m$. We denote the negation of x by x'. Without loss of generality, we assume

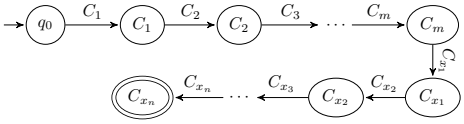

Figure 8. An illustration of the automaton $\mathcal{A}$.

that the literals in each clause are distinct, i.e., we may treat each clause as a set.

From φ, construct the automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, \{q_{m+n}\})$ with a single initial state, q_0, and the set of states consisting of all clauses and all two-element sets of literals corresponding to a variable, that is, $Q = \{C_1, \ldots, C_m\} \cup \{C_x = \{x, x'\} \mid x \in X\}$. To simplify the presentation, we choose a fixed order of the states of Q, so that we can write $Q = \{q_0\} \cup \{q_1, \ldots, q_{m+n}\}$, where q_0 is the initial state, q_i is the clause C_i, for $i = 1, \ldots, m$, and $q_{m+1}, \ldots, q_{m+n}$ are the sets $C_{x_1}, \ldots, C_{x_n}$, respectively. We set the state q_{m+n} secret with $\ell(q_{m+n}) = n + 1$; the other states are nonsecret. The transitions are defined as follows: for every $x \in q_{i+1}$, we add the transition (q_i, x, q_{i+1}). Notice that all paths from q_0 to q_{m+n} are of length $m + n$; see Figure 8.

We now show that φ is satisfiable if and only if the policy $\mathcal{P} = \bigcup_{x \in X} C_x$ is not indicator-valid.

If φ is satisfiable, then there is an assignment to the variables that satisfies all clauses. Let V be the set of literals that are assigned the value 1. Then $|V| = n$, and since V satisfies all clauses, it contains at least one literal of every clause. Therefore, there is a string $w \in V^{m+n}$ leading from q_0 to q_{m+n} that violates the requirement $\ell(q_{m+n}) = n + 1$, since w contains at most $|V| = n$ different events. Thus, $\mathcal{P}$ is not indicator-valid.

On the other hand, we assume that $\mathcal{P}$ is not indicator-valid, and consider a string w over $\mathcal{P}$ from q_0 to q_{m+n} that violates indicator-validity. Since w leads through the states corresponding to the sets $C_x = \{x, x'\}$, for every $x \in X$, the string w contains at least n different events. These events correspond to the literals of n different variables. Since w violates indicator-validity, it contains exactly n distinct events. However, these events must define a satisfying assignment for φ, because the string w also goes through all the states that correspond to the clauses $C_1, \ldots, C_m$, and hence the ith event of w satisfies the clause C_i. Hence, φ is satisfiable. □

Recall that the class Σ_2^P consists of problems that can be expressed as $L = \{w \mid \exists x \ \forall y \ R(w, x, y)\}$, where R is a polynomial-time computable predicate. Intuitively, a language L is in Σ_2^P if there is a polynomial-time verifier that, given a guessed string x, can query an NP oracle to verify that a condition over y holds. An oracle Turing machine is a polynomial-time machine that can make queries to another decision problem (the *oracle*) and receive instantaneous answers. A nondeterministic polynomial-time machine with access to an NP (or coNP) oracle, denoted by NP^{NP} (or NP^{coNP}), captures the computational power of Σ_2^P. We make use of this characterization in the proof of the main result of this section.

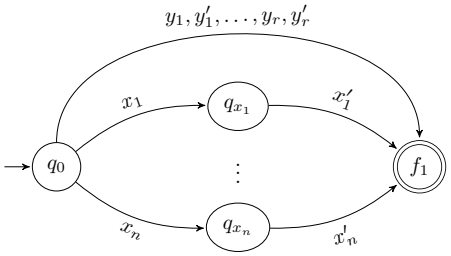

Figure 9. The first step of the reduction.

Theorem 13. *BC-Indicator-SPP(-U) is Σ_2^P-complete.*

Proof. To show that BC-Indicator-SPP belongs to Σ_2^P, we guess a policy $\mathcal{P} \subseteq \Sigma_p$ with the cost not exceeding the budget constraint, and verify that $\mathcal{P}$ is indicator-valid. Since the guess of $\mathcal{P}$ can be done in nondeterministic polynomial time, the cost can be computed in polynomial time, and the check whether $\mathcal{P}$ is indicator-valid is a coNP-complete problem by Proposition 12, we obtain that BC-Indicator-SPP belongs to $\text{NP}^{\text{coNP}} = \Sigma_2^P$.

To prove hardness, we reduce the Σ_2^P-complete problem of 3-QSAT_2 (Stockmeyer, 1976). The problem asks whether a formula of the form $\exists X \, \forall Y \, \varphi(X, Y)$, where $\varphi(X, Y)$ is a Boolean formula in 3-DNF and X, Y are disjoint sets of variables, is satisfiable.

To this end, let $\exists X \, \forall Y \, \varphi(X, Y)$ be a 3-QSAT_2 formula with $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_r\}$, and with m conjuncts $C_1, \ldots, C_m$. In each conjunct, we replace every literal $\neg z$ with a fresh symbol z'. Without loss of generality, we assume that the literals within each conjunct are distinct, allowing us to treat each conjunct as a set. Additionally, we denote by $\overline{X}$ the set of all literals over X, that is, $\overline{X} = \{x, x' \mid x \in X\}$. Analogously, we define the set $\overline{Y}$ of Y-literals.

We construct an automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, \{f_1, f_2\})$ with a single initial state q_0 and two secret states f_1 and f_2. The alphabet of $\mathcal{A}$ consists of all literals, that is, $\Sigma = \overline{X} \cup \overline{Y}$. The transition function of $\mathcal{A}$ is defined as follows. For each variable $x \in X$, we add two transitions (q_0, x, q_x) and (q_x, x', f_1), where q_x is a new state, and for each variable $y \in Y$, we add two transitions

$$(q_0, y, f_1) \text{ and } (q_0, y', f_1)\,,$$

see Figure 9 for an illustration. This step introduces n new states and $2(n + r)$ transitions.

For each conjunct $C_i = \{\lambda_{i,1}, \ldots, \lambda_{i,k_i}\}$, $1 \leq i \leq m$, we add a state C_i and k_i transitions from C_{i-1} to C_i under the

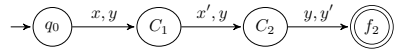

Figure 10. The encoding of the formula $\varphi = C_1 \vee C_2$ where $C_1 = \{x, y\}$, $C_2 = \{x', y\}$, $Y = \{y\}$, and $f_2 = C_y$.

literals of C_i, that is, we add the transitions

$$(C_{i-1}, \lambda_{i,1}, C_i), (C_{i-1}, \lambda_{i,2}, C_i), \ldots, (C_{i-1}, \lambda_{i,k_i}, C_i)\,,$$

where C_0 denotes the initial state q_0. Finally, for every $y_i \in Y$, we add a new state C_{y_i} together with two transitions from $C_{y_{i-1}}$ to C_{y_i} under y_i and y_i', that is, we add

$$(C_{y_{i-1}}, y_i, C_{y_i}) \text{ and } (C_{y_{i-1}}, y_i', C_{y_i})\,,$$

where $C_{y_0} = C_m$ and C_{y_r} is denoted by f_2, see Figure 10 for an illustration. This step introduces $m + r$ new states and $2r + \sum_{i=1}^{m} k_i$ transitions, which is polynomial in the size of the formula.

To complete the construction, we define the cost and clearance functions uniformly by setting $c(a) = 1$ and $\gamma(a) = 1$ for every event $a \in \Sigma$, meaning that all events in $\mathcal{A}$ are protectable. The security requirement function is defined by setting $\ell(f_1) = 1$ and $\ell(f_2) = r + 1$, while all the other states have security requirement zero. Finally, we set $W = 2r + n$.

We now show that the formula $\exists X\, \forall Y\, \varphi(X, Y)$ is satisfiable if and only if there is an indicator-valid protection policy $\mathcal{P}$ such that $C(\mathcal{P}) \leq W$. To simplify the arguments, we state the following two claims.

Claim 14. *Every indicator-valid policy $\mathcal{P}$ with $C(\mathcal{P}) \leq W$ satisfies $C(\mathcal{P}) = 2r + n$ and has the form $\mathcal{P} = \overline{Y} \cup \{\text{exactly one of } x, x' \mid x \in X\}$.*

Proof. For a Y-literal z, a path from q_0 to f_1 is the single edge labelled z; since $\ell(f_1) = 1$, indicator-validity forces $z \in \mathcal{P}$. Hence $\overline{Y} \subseteq \mathcal{P}$, contributing cost $2r$. Analogously, for each $x \in X$, a path from q_0 to f_1 is $q_0 \to^{x} q_x \to^{x'} f_1$, which uses the two distinct events x, x'; as $\ell(f_1) = 1$, $\mathcal{P}$ contains at least one of them, contributing cost at least n in total. Thus, $C(\mathcal{P}) \geq 2r + n = W$, and the assumption $C(\mathcal{P}) \leq W$ implies that $\mathcal{P}$ contains exactly one literal of each pair $\{x, x'\}$. □

Let ν_X be an assignment to the X-variables, and let

$$\mathcal{P} = \overline{Y} \cup \{\lambda \in \overline{X} \mid \lambda \text{ is true under } \nu_X\}\,. \tag{1}$$

Claim 15. *The minimum, over all paths from q_0 to f_2, of the indicator clearance $\big|\{\text{events of the path}\} \cap \mathcal{P}\big|$ equals r if there is a Y-assignment ν_Y with $\varphi(\nu_X, \nu_Y) = 0$, and is at least $r + 1$ otherwise.*

Proof. A path from q_0 to f_2 chooses a literal $\lambda_i \in C_i$ for $i = 1, \ldots, m$ and a literal $b_j \in \{y_j, y_j'\}$ for $j = 1, \ldots, r$. Its clearance is $\big|\{\lambda_1, \ldots, \lambda_m, b_1, \ldots, b_r\} \cap \mathcal{P}\big|$. The r events $b_1, \ldots, b_r$ are pairwise distinct and all lie in $\mathcal{P}$; therefore, they contribute exactly r. Let ν_Y be the Y-assignment that makes each chosen b_j *false*. We claim that $\lambda_i \in \mathcal{P}$ and $\lambda_i \notin \{b_1, \ldots, b_r\}$ if and only if λ_i is *true* under $\nu_X \cup \nu_Y$: If λ_i is an X-literal, then $\lambda_i \notin \{b_1, \ldots, b_r\}$, and by (1) $\lambda_i \in \mathcal{P}$ iff λ_i is true under ν_X. If λ_i is a Y-literal, then $\lambda_i \in \mathcal{P}$; and $\lambda_i \notin \{b_1, \ldots, b_r\}$ iff λ_i is true under ν_Y. Hence the clearance equals r plus the number of distinct chosen literals λ_i that are true under $\nu_X \cup \nu_Y$. For a fixed ν_Y, pick $\lambda_i \in C_i$ to be a literal false under $\nu_X \cup \nu_Y$ whenever one exists. Such a false literal exists iff C_i is *not* satisfied by $\nu_X \cup \nu_Y$. Therefore, the minimal clearance for ν_Y is exactly r if every conjunct is unsatisfied, i.e. $\varphi(\nu_X, \nu_Y) = 0$, and is at least $r + 1$ otherwise. □

($\Rightarrow$) If the formula $\exists X\, \forall Y\, \varphi(X, Y)$ is satisfiable, then there is a truth assignment ν_X to the X-variables that satisfies the formula $\forall Y\, \varphi(X, Y)$. Let $\mathcal{P} = \overline{Y} \cup \{\lambda \in \overline{X} \mid \lambda \text{ is true under } \nu_X\}$, then $C(\mathcal{P}) = 2r + n = W$ and $\mathcal{P}$ contains every Y-literal and one literal of each pair $\{x, x'\}$, that is, it satisfies the secret state f_1. Since no ν_Y makes $\varphi(\nu_X, \nu_Y) = 0$, Claim 15 gives minimal clearance to f_2 at least $r + 1 = \ell(f_2)$, and hence $\mathcal{P}$ also satisfies f_2. Thus $\mathcal{P}$ is indicator-valid and within budget.

($\Leftarrow$) Suppose $\mathcal{P}$ is indicator-valid with $C(\mathcal{P}) \leq W$. By Claim 14, $\mathcal{P} = \overline{Y} \cup \mathcal{P}_X$, where $\mathcal{P}_X$ contains exactly one literal of each pair $\{x, x'\}$. Let ν_X be the assignment defined by $\nu_X(x) = 1$ if $x \in \mathcal{P}$ and $\nu_X(x) = 0$ if $x' \in \mathcal{P}$; by (1), $\mathcal{P}_X$ is exactly the set of X-literals true under ν_X. As $\mathcal{P}$ satisfies f_2, every path to f_2 has clearance at least $r + 1$, and hence by Claim 15 there is no ν_Y with $\varphi(\nu_X, \nu_Y) = 0$; that is, ν_X satisfies $\forall Y\, \varphi(X, Y)$, and the formula is satisfiable. □

The following examples illustrate the construction.

Example 16. Let $\varphi = \exists x_1, x_2 \forall y_1, y_2.(x_2 \wedge \neg y_1) \vee (x_1 \wedge y_2) \vee (\neg x_2 \wedge \neg y_2)$ with conjuncts $C_1 = \{x_2, y_1'\}$, $C_2 = \{x_1, y_2\}$, $C_3 = \{x_2', y_2'\}$. The assignment $\nu_X = (x_1 = 1, x_2 = 0)$ satisfies $\forall y_1, y_2\, \varphi$, and hence φ is satisfiable. Here $n = r = 2$, $m = 3$, and the budget is $W = 2r + n = 6$. The automaton produced by the construction is shown in Figure 11; all events have cost and clearance 1, $\ell(f_1) = 1$, and $\ell(f_2) = r + 1 = 3$. Consider the policy $\mathcal{P} = \{x_1, x_2', y_1, y_1', y_2, y_2'\}$ corresponding to ν_X with $C(\mathcal{P}) = 6 = W$. Under $\nu_X = (1, 0)$, we have $C_1 = x_2 \wedge \neg y_1 \equiv 0$, $C_2 = x_1 \wedge y_2 \equiv y_2$, $C_3 = \neg x_2 \wedge \neg y_2 \equiv \neg y_2$, and hence C_1 is never satisfied, while for every ν_Y exactly one of C_2, C_3 is satisfied, namely C_2 when $y_2 = 1$ and C_3 when $y_2 = 0$. The four universal assignments behave as discussed in Table 3. Hence every path from q_0 to f_2 has clearance at least $3 = \ell(f_2)$, that is, $\mathcal{P}$ is indicator-valid and meets the budget. ◇

Example 17. Let $\psi = \exists x_1, x_2 \forall y_1, y_2.(x_1 \wedge y_1) \vee (x_2 \wedge \neg y_1 \wedge y_2) \vee (\neg x_1 \wedge \neg x_2 \wedge y_2)$ with conjuncts $C_1 = \{x_1, y_1\}$, $C_2 = \{x_2, y_1', y_2\}$, $C_3 = \{x_1', x_2', y_2\}$. No assignment to x_1, x_2

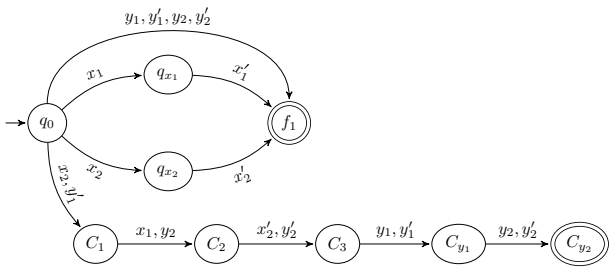

Figure 11. The automaton for $\varphi = \exists x_1, x_2\ \forall y_1, y_2.\ (x_2 \wedge \neg y_1) \vee (x_1 \wedge y_2) \vee (\neg x_2 \wedge \neg y_2)$.

Table 3
The four universal assignments.

(y_1, y_2)	(b_1, b_2)	Satisfied conjunct	Clearance
$(0, 0)$	(y_1, y_2)	C_3 (x_2', y_2' true)	2 + 1 = 3
$(0, 1)$	(y_1, y_2')	C_2 (x_1, y_2 true)	2 + 1 = 3
$(1, 0)$	(y_1', y_2)	C_3	2 + 1 = 3
$(1, 1)$	(y_1', y_2')	C_2	2 + 1 = 3

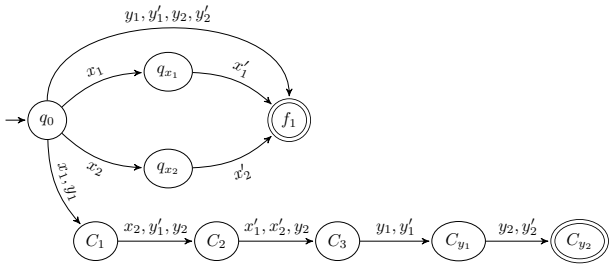

Figure 12. The automaton for $\psi = \exists x_1, x_2, \forall y_1, y_2.(x_1 \wedge y_1) \vee (x_2 \wedge \neg y_1 \wedge y_2) \vee (\neg x_1 \wedge \neg x_2 \wedge y_2)$.

satisfies ψ for all y_1, y_2, that is, ψ is false. Here $n = r = 2$, $m = 3$, $W = 6$, $\ell(f_1) = 1$, and $\ell(f_2) = 3$; the automaton is shown in Figure 12. Every indicator-valid policy $\mathcal{P}$ with $C(\mathcal{P}) \leq W$ has the form $\overline{Y} \cup \{\lambda \in \overline{X} \mid \lambda \text{ is true under } \nu_X\}$ for some assignment ν_X to the X-variables. We show that no indicator-valid policy within the budget exists by exhibiting a path to f_2 of clearance $2 < \ell(f_2)$. All four possible policies fail in the same way; a witnessing universal assignment and the resulting clearance are listed below:

$\nu_X = (x_1, x_2)$	$\mathcal{P} \setminus \overline{Y}$	Witness (y_1, y_2)	Clearance
$(1, 1)$	$\{x_1, x_2\}$	$(0, 0)$	2
$(1, 0)$	$\{x_1, x_2'\}$	$(0, 0)$	2
$(0, 1)$	$\{x_1', x_2\}$	$(1, 0)$	2
$(0, 0)$	$\{x_1', x_2'\}$	$(0, 0)$	2

For instance, for $\nu_X = (1, 1)$ and $(y_1, y_2) = (0, 0)$, C_1 is falsified through y_1, C_2 through y_2, and C_3 through $x_1' \notin \mathcal{P}$, resulting in the path $y_1 y_2 x_1' y_1 y_2$ giving clearance $|\{y_1, y_2\}| = 2$. Since none of these policies is valid, the construction correctly reports ψ as unsatisfiable. ◇

7 Conclusion

We addressed the computational complexity of two variants of the secret protection problem in discrete-event systems—Parikh-SPP and Indicator-SPP. While previous results showed that Parikh-SPP is solvable in polynomial time under the assumption of uniquely labeled transitions, we extended the understanding of its hardness by considering more general and practically relevant scenarios.

We showed that relaxing the uniqueness constraint on event labels makes Parikh-SPP NP-hard. Moreover, we proved that the problem remains NP-hard even if the cost and clearance functions are constant, and the security requirement is binary with only a single secret state. These results closed the gap left open in the literature and established NP-hardness for the discussed variants of the problem.

We further showed that no sub-exponential-time algorithm can solve Parikh-SPP or Indicator-SPP unless the exponential time hypothesis fails. This result implies that, under current complexity-theoretic assumptions, exhaustive enumeration of all subsets of protectable events is essentially optimal.

Given that the decision version of Parikh-SPP is NP-complete, we developed an ILP-based algorithm and conducted an extensive empirical evaluation demonstrating its scalability and efficiency on benchmark instances comprising up to hundreds of thousands of states.

Finally, we analyzed the computational complexity of the decision version of Indicator-SPP, where only distinct protected events contribute to clearance. We proved that this problem is Σ_2^P-complete, placing it at the second level of the polynomial-time hierarchy. In contrast to Parikh-SPP, for which an ILP-based approach proved practically efficient, developing a practically efficient algorithm for Indicator-SPP remains an open challenge. Since the worst-case complexity need not reflect performance on realistic inputs, it remains open whether suitable techniques or heuristics could solve relevant instances of Indicator-SPP efficiently.

References

Arora, S., Barak, B., 2009. Computational Complexity - A Modern Approach. Cambridge University Press.

Chocholatý, D., Fiedor, T., Havlena, V., Holík, L., Hruska, M., Lengál, O., Síc, J., 2024. Mata: A fast and simple finite automata library, in: TACAS, pp. 130–151.

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman.

Hertli, T., 2014. 3-SAT faster and simpler—unique-SAT bounds for PPSZ hold in general. SIAM Journal on Computing 43, 718–729.

Impagliazzo, R., Paturi, R., 2001. On the complexity of k-SAT. Journal of Computer and System Sciences 62, 367–375.

Ma, Z., Cai, K., 2022. Optimal secret protections in discrete-event systems. IEEE Trans. Automatic Control 67, 2816–2828.

Ma, Z., Cai, K., 2025. Secret protection in discrete-event systems with generalized confidentiality requirements. IEEE Trans. Automatic Control 70, 2321–2333.

Ma, Z., Jiang, J., Cai, K., 2024. Secret protections with costs and disruptiveness in discrete-event systems using centralities. IEEE Trans. Automatic Control 69, 4380–4395.

Matsui, S., Cai, K., 2019. Secret securing with multiple protections and minimum costs, in: 58th IEEE Conference on Decision and Control, CDC 2019, Nice, France, December 11-13, 2019, pp. 7635–7640.

Paturi, R., Pudlák, P., Saks, M.E., Zane, F., 2005. An improved exponential-time algorithm for k-SAT. Journal of the ACM 52, 337–364.

Schrijver, A., 1999. Theory of linear and integer programming. Wiley-Interscience series in discrete mathematics and optimization.

Sipser, M., 2012. Introduction to the Theory of Computation. Cengage Learning.

Stockmeyer, L.J., 1976. The polynomial-time hierarchy. Theoretical Computer Science 3, 1–22.

Tabakov, D., Vardi, M.Y., 2005. Experimental evaluation of classical automata constructions, in: Logic for Programming, Artificial Intelligence, and Reasoning, pp. 396–411.