mathematics

MDPI

Article

Almost *k*-Step Opacity Enforcement in Stochastic Discrete-Event Systems via Differential Privacy

Rong Zhao [1], Murat Uzam [2] and Zhiwu Li [1,*]

[1] Institute of Systems Engineering, Macau University of Science and Technology, Macau SAR, China; 2109853zmi30002@student.must.edu.mo
[2] Department of Electrical and Electronics Engineering, Faculty of Engineering and Architecture, Yozgat Bozok University, 66900 Yozgat, Turkiye; murat.uzam@yobu.edu.tr
* Correspondence: zwli@must.edu.mo

Abstract: This paper delves into current-state opacity enforcement in partially observed discrete event systems through an innovative application of differential privacy, which is fundamental for security-critical cyber–physical systems. An opaque system implies that an external agent cannot infer the predefined system secret via its observational output, such that the important system information flow cannot be leaked out. Differential privacy emerges as a robust framework that is pivotal for the protection of individual data integrity within these systems. Motivated by the differential privacy mechanism for information protection, this research proposes the secret string adjacency relation as a novel concept, assessing the similarity between potentially compromised strings and system-generated alternatives, thereby shielding the system's confidential data from external observation. The development of secret string differential privacy is achieved by substituting sensitive strings. These substitution strings are generated by a modified Levenshtein automaton, following exponentially distributed generation probabilities. The verification and illustrative examples of the proposed mechanism are provided.

Keywords: discrete event system; finite state automaton; *k*-step opacity; differential privacy

MSC: 93-08

Academic Editor: Ben Niu

Received: 2 March 2025
Revised: 2 April 2025
Accepted: 8 April 2025
Published: 10 April 2025

Citation: Zhao, R.; Uzam, M.; Li, Z. Almost *k*-Step Opacity Enforcement in Stochastic Discrete-Event Systems via Differential Privacy. *Mathematics* **2025**, *13*, 1255. https://doi.org/10.3390/math13081255

1. Introduction

Opacity, an information flow property, has garnered significant attention in recent years on discrete-event systems (DESs) that are a mathematical generalization of various contemporary technological systems characterized by computer integration and critical safety [1–3]. Typical discrete-event systems include various cyber–physical systems, concurrent computing systems and programming, logistics, etc., where the state transition is triggered by the occurrences of events, i.e., system evolution is driven by events not time, in contrast to the conventional continuous-variable dynamic systems. Due to their inherent characteristics in structure, function, and organization of DESs, privacy and security are paramount concerns in system design and development. For example, current-state opacity characterizes an information flow, originally proposed in the community of computer science, which provides a guarantee that an external observer or intruder cannot decide or infer whether a system's current state is a secret. Several types of opacity have been proposed, including, in addition to current-state opacity, language-based opacity, initial-and-final-state opacity, *k*-step opacity, and infinite-step opacity.

As a common DES model, finite state automata are instrumental in modeling, analyzing, and controlling contemporary cyber–physical systems, which are interconnected via computer or communication networks, thereby making them vulnerable to external malicious cyber-attacks [4,5]. External attackers can potentially uncover sensitive information, such as secrets, by observing the generated strings from a DES. Roughly speaking, opacity serves as a security property that assesses whether outside attackers can infer sensitive data or information from a system, assuming that they know the system's structure.

As noted, k-step opacity [6] is introduced to determine whether an attacker is able to deduce that a system is in a secret state within a certain number of steps before a specific moment. As the parameter k increases, the attacker's ability to discern the system's secret within k observation steps becomes restricted. When k approaches infinity, k-step opacity transitions into infinite-step opacity [7,8], signifying that the system prevents the attacker from ever definitively ascertaining the occurrence of a secret state at any point of time, regardless of how far back they inspect the system's behavior.

The enforcement of infinite-step opacity and k-step opacity in DESs by using an insertion function is proposed [9], which acts as a monitoring interface to insert fictitious events into the output as needed. Two new automata are developed to recognize safe languages for infinite-step and k-step opacity, ensuring that the output is secure. The insertion function leverages these automata to determine the appropriate points for event insertion. This method extends the insertion mechanism from current-state opacity [10] to infinite-step and k-step opacity.

Recent studies have explored weak and strong k-step opacity as extensions of both current-state opacity and infinite-step opacity [11]. An algorithm for verifying weak k-step opacity, independent of k, has been proposed. These works examine strong k-step opacity by reducing it to weak k-step opacity.

The traditional definitions of infinite-step opacity and k-step opacity offer a binary assessment, classifying systems as either opaque or non-opaque. However, in practical scenarios, a non-opaque system might still possess a low probability of violation, which could be acceptable in many real-world applications. Recognizing this case that is of practical value, recent research has shifted towards quantitatively evaluating opacity by leveraging probabilistic models [12,13], particularly in the context of stochastic DESs.

By incorporating probabilistic considerations, the related works aim to provide a more nuanced understanding of system opacity, allowing for a quantitative assessment of the likelihood of opacity breaches. This approach enables a finer-grained analysis, facilitating the identification of acceptable levels of opacity for different applications and system requirements. By accurately modeling the transition probabilities within the system, it becomes possible to assess the likelihood of security vulnerabilities in a more nuanced manner, rather than relying solely on binary classifications, thus providing a detailed evaluation of the system's susceptibility to breaches.

Existing research on opacity analysis in a stochastic DES [13,14] primarily focuses on current-state-type opacity, neglecting the consideration of delayed information in evaluating infinite-step opacity and k-step opacity. While initial-state opacity can be considered a current-state-type property when incorporating initial states into the system's state space, the evaluation of opacity becomes more complex when future information influences our understanding of the system's current status.

To avoid privacy leakage to an attacker with full knowledge of a security-critical system that is usually networked, the notion of differential privacy has been proposed as a method to mathematically safeguard individual privacy [15–17]. Initially, differential privacy was used to solve pricing and auction problems [18]. Shortly, it was extended in a

widespread way to protect sensitive data by adding random noise, thereby preventing user privacy from being leaked to attackers. The classic differential privacy mechanisms include the Laplace mechanism [19], the exponential mechanism [20], the Gauss mechanism [21], and the geometric mechanism [22]. These mechanisms achieve differential privacy by satisfying different distributions of query results. The probabilities of two datasets outputting the same results are approximate when an attacker conducts a differentiated attack by using two datasets that differ by only one record.

In traditional differential privacy, adding noise does not apply to non-numerical or symbolic data. In [23,24], differential privacy frameworks for symbolic systems are presented to protect trajectories generated by symbolic systems. The new differential privacy mechanisms in [23,24] approximate a sensitive word using a random word that is likely to be near it. However, in the framework of DESs, the approximate word generated in this way may not belong to the system language. An attacker who knows the structure of a system can detect the existence of the protection mechanism by discovering an output that does not belong to the language of the system.

In recent advancements, differential privacy has been increasingly applied to DESs to enhance state protection. One notable development is the introduction of state-protecting differential privacy [25,26], which primarily focuses on safeguarding state data, particularly the initial state privacy in systems with multiple starting states. For system architectures that do not inherently satisfy state differential privacy requirements, supervisory control can be implemented to enforce privacy protection.

Another method for achieving differential privacy in DESs is to design a mechanism that processes system observations [27]. It ensures that potentially sensitive observations and randomly generated ones are modified probabilistically, making their output probabilities nearly equal to obscure sensitive information.

This paper focuses on almost k-step opacity enforcement in stochastic DESs while considering data statistics and analysis. A potentially malicious attacker who monitors a system is modeled as a passive observer. A system is deemed opaque if the probability of strings that violate k-step opacity is smaller than a given threshold that is a real number less than one but greater than zero.

We aim to thwart external attackers from deducing the presence of a secret state within a system by reducing the probability of strings that breach k-step opacity. Simultaneously, we prioritize preserving the information utility of observed strings, ensuring that they adhere to a differential privacy mechanism. This dual objective allows us to enhance system opacity while maintaining the integrity of shared data for statistical analysis and other purposes. Our focus lies in elevating system opacity to meet stringent standards while safeguarding the statistical properties of strings and facilitating secure and efficient data-sharing practices. The main contributions of this work are summarized as follows:

- The almost k-step opacity enforcement problem in stochastic systems is formalized by leveraging differential privacy. This novel privacy mechanism addresses the need in the context of DESs to balance privacy preservation with utility in the context of a dynamic and uncertain environment.
- A probability mechanism is constructed to adhere to almost k-step differential privacy requirements. This mechanism utilizes a modified Levenshtein automaton, offering a practical approach to ensuring privacy while maintaining the integrity and usefulness of the data.
- A policy is reported for enforcing almost k-step opacity by reducing the occurrence probability of strings that violate k-step opacity, thus enhancing system security while preserving data utility.

The remaining sections of this paper are organized as follows. Section 2 introduces the basic notions of stochastic system models, the Levenshtein automata, and differential privacy. The notion of almost k-step opacity and problem formulation is given in Section 3. Section 4 formally defines a utility function and a sensitivity bound. In Section 5, a substitution string generation mechanism is proposed. Section 6 presents the construction of modified Levenshtein automaton and the policy of enforcing almost k-step opacity. Finally, Section 7 concludes this work.

2. Preliminaries

This section reviews the basic notions of finite state automata, Levenshtein automata, and differential privacy. The set of whole numbers (non-negative integers) is denoted by $\mathbb{N} = \{0, 1, 2, \ldots\}$.

2.1. System Model

In this paper, a DES is modeled as a finite state automaton $A = (Q, \Sigma, \eta, q_0, Q_m)$, where Q is the set of finite states, Σ is the set of events (called alphabet that is usually finite), $\eta : Q \times \Sigma \to Q$ is the (partial) transition function, q_0 is the initial state, and Q_m is the set of marked states that is of no interest in this paper unless otherwise stated. The transition function can be extended from the domain $Q \times \Sigma$ to the domain $Q \times \Sigma^*$. For a string $s \in \Sigma^*$ and an event $\sigma \in \Sigma$, the extended transition function can be recursively defined as $\eta(q, \varepsilon) = q$ and $\eta(q, s\sigma) = \eta(\eta(q, s), \sigma)$, where ε is the empty string containing no event. We write the transition function from the initial state to a certain state through a string $s \in \Sigma^*$ as $\eta(s) = \eta(q_0, s)$, where Σ^*, called the Kleene closure, is the collection of finite strings defined over Σ, including the empty string ε. Note that $\eta \subseteq Q \times \Sigma^* \times Q$ admits a relation representation of the transition function.

Given an automaton $A = (Q, \Sigma, \eta, q_0, Q_m)$, we denote the operation of deleting all the states that are not accessible from q_0 and not coaccessible to Q_m by $Trim(A)$ [28]. For any string $s \in \Sigma^*$, let $|s|$ be the length of s and σ_s^i be the i-th event in s. The language generated by an automaton A is defined as $L(A) = \{s \in \Sigma^* \mid \eta(q_0, s) \text{ is defined}\}$. A language defined over Σ is a subset of Σ^*, i.e., $L \subseteq \Sigma^*$. If there exists $v \in \Sigma^*$ and $uv = s$, $u \in \Sigma^*$ is said to be a prefix of s, denoted by $u \preceq s$. The set of all prefixes of s is $[s] = \{u \in \Sigma^* | u \preceq s\}$. Given a language $L \subseteq \Sigma^*$, the prefix of L is defined as $\bar{L} = \cup_{s \in L}[s]$. Given a letter $t \in \Sigma$ and a string $s \in \Sigma^*$, write $t \in s$ if t appears in s at least once. We denote by $t \leq s$ if $t \in s$ and $t < s$ if $t \in [s] \setminus \{s\}$. For any prefix $t \leq s$ of s, we denote s/t as the post string of t in s, i.e., $t(s/t) = s$, where $t(s/t)$ is the concatenation of t and s/t.

In general, a system is partially observable. The event set Σ can be partitioned into an observable set Σ_o and an unobservable set Σ_{uo}, i.e., $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. The natural projection $P : \Sigma^* \to \Sigma_o^*$ is defined as follows:

(1) $P(\varepsilon) = \varepsilon$;
(2) For any event $\sigma \in \Sigma$, $P(\sigma) = \sigma$ if $\sigma \in \Sigma_o$, and $P(\sigma) = \varepsilon$ if $\sigma \in \Sigma_{uo}$;
(3) For any string $s \in \Sigma^*$ and any event $\sigma \in \Sigma$, $P(s\sigma) = P(s)P(\sigma)$. We also define the inverse projection $P^{-1} : \Sigma_o^* \to 2^{\Sigma^*}$ as $P^{-1}(t) = \{s \in \Sigma^* \mid P(s) = t\}$.

Definition 1. *(Observer automaton): The standard observer automaton [28] of an automaton $A = (Q, \Sigma, \eta, q_0)$ that can be established by subset construction is defined as $O(A) = (Q_o, \Sigma_o, \eta_o, q_{0,o})$, where $Q_o \subseteq 2^Q$ is the set of states, Σ_o is the set of observable events, $\eta_o : Q_o \times \Sigma_o^* \to Q_o$ is the transition function, and $q_{0,o} \in Q_o$ is the initial state.*

In this paper, a stochastic discrete-event system is modeled as a probabilistic finite-state automaton (PFA) (G, p), where $G = (Q, \Sigma, \eta, q_0, Q_m)$ is a finite state automaton and

$p : Q \times \Sigma \to [0, 1]$ is the transition probability function. For any $q \in Q, \sigma \in \Sigma$, we write $p(\sigma|q)$ as the probability that event σ occurs in state q. For every state $q \in Q$, the sum of the probabilities of all possible events in this state $\sum_{\sigma \in \Sigma} p(\sigma|q) = 1$. And for every state $q \in Q$, if $\sigma \in \Sigma$, then $p(\sigma|q) > 0$ means that transition $\eta(q, \sigma)$ exists or $\eta(q, \sigma)$ is defined. Given a string $s \in \Sigma^*$, the probability of string s is denoted by $Pr(s)$, i.e., $Pr(s\sigma) = Pr(s)p(\sigma|\eta(s))$ where $\sigma \in \Sigma$.

2.2. Levenshtein Distance and Levenshtein Automaton

To compare two strings with the same length, we introduce the concept of the Levenshtein distance to measure the difference or edit distance between two strings [24]. The Levenshtein distance indicates the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another.

Definition 2. *(Levenshtein distance): Given two strings $s_1, s_2 \in \Sigma^*$ with $|s_1| = |s_2|$, $s_1 = \sigma_{s_1}^1 \dots \sigma_{s_1}^n$, and $s_2 = \sigma_{s_2}^1 \dots \sigma_{s_2}^n$, the Levenshtein distance between the strings s_1 and s_2, denoted as $d(s_1, s_2)$, is the count of events $\sigma_{s_1}^j \neq \sigma_{s_2}^j$, where $j \in \{1, \dots, n\}$.*

In this paper, we consider a restricted version of the Levenshtein distance that only accounts for substitution operations on strings of the same length. Under this condition, it is equivalent to the Hamming distance [23]. In order to correspond to the Levenshtein automaton below, it is still called the Levenshtein distance. Based on this setting, given two strings $s_1, s_2 \in \Sigma^*$, we conclude that $d(s_1, s_2)$ is the minimum number of changes that convert s_1 to s_2. Moreover, it is feasible to extend the Levenshtein distance for two strings with different lengths. For example, given two strings s_1 and s_2 with $|s_1| = n$, $|s_2| = m$, and $n > m$. Write $s_1 = s_1' s_1''$ with $|s_1'| = m$. Then, we can define $d(s_1, s_2) = d(s_1', s_2) + (n - m)$.

Example 1. *Given an alphabet $\Sigma = \{a, b, c, d, e\}$, the Levenshtein distance between the strings abcdd and abbae is 3, since the third event c is substituted to b, the fourth event d is substituted to a, and the fifth event d is substituted to e.*

Given a string s and a specific Levenshtein distance ω, a Levenshtein automaton can be used to determine whether the Levenshtein distance between the string s and another string is within ω [29]. Note that the Levenshtein automaton used in this work is a restricted version specifically tailored for substitution operations on equal-length strings.

Definition 3. *(Levenshtein automaton): Given a string $s \in \Sigma^*$ and an integer $\omega \in \mathbb{N}$, a Levenshtein automaton (with respect to the string s and integer ω) is a finite state automaton $A_{s,\omega} = (Q_{|s|,d}, \Sigma, \eta_{s,\omega}, q_{0,0})$, where $L(A_{s,\omega}) = \{s' \in \Sigma^* | d(s, s') \leq \omega\}$, $Q_{|s|,d}$ is the set of states, Σ is the set of events, $\eta_{s,\omega}$ is the transition function, and $q_{0,0}$ is the initial state.*

Example 2. *Consider $\Sigma = \{a, b, c\}$, a string bbc, and a distance $\omega = 2$. The Levenshtein automaton $A_{bbc,2}$ is shown in Figure 1. The horizontal arrow at the bottom line represents the original events in the string, and the slanted arrow represents the events that can be used to substitute the original event. For the slanted arrow from the initial state 0^0 to the state 1^1, the events a and c can substitute the first event b in the string bbc. The path from the initial state to another state constitutes the possible output string of the Levenshtein automaton, i.e., the output string can be ab, cb, ba or bc in the state 2^1.*

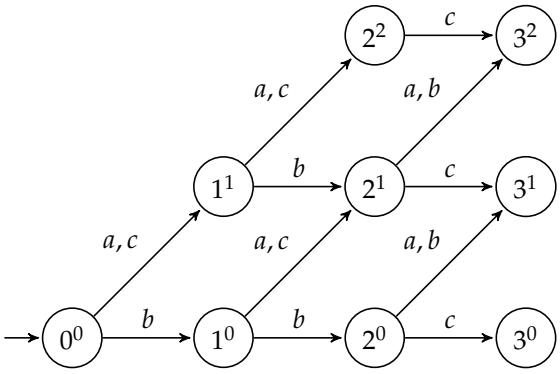


Figure 1. A Levenshtein automaton $A_{bbc,2}$.

2.3. Differential Privacy

Differential privacy provides a query method to protect records, maximizing the accuracy of the overall record while preventing the risk of leakage of single records. From its definition, differential privacy is enforced by a mechanism (i.e., a randomized function) that necessarily produces outputs with almost equal probabilities for adjacent sensitive data specified by an adjacency relation [30].

A randomized function M satisfies ϵ-differential privacy if for all $\mathcal{S} \subseteq range(M)$ and all adjacent sensitive data D_1 and D_2:

$$Pr[M(D_1) \in \mathcal{S}] \leq e^{\epsilon} Pr[M(D_2) \in \mathcal{S}]. \tag{1}$$

where $rang(M)$ is the range of function M, $Pr[M(D_i) \in \mathcal{S}]$ is the probability that the output of algorithm M under the input D_i ($i = 1, 2$) falls into $\mathcal{S}$, e is the natural constant, and the parameter ϵ, which is usually a real number, depends on a privacy policy and affects the degree of privacy protection. With the increase in ϵ, the data availability becomes higher, while the privacy becomes lower. Typically, ϵ is set to a small constant, such as 0.1 to $\ln(3)$.

In some scenes, Markov Decision Processes (MDPs) [31] are useful for privacy–utility trade-offs in optimal control problems, as they model state transitions probabilistically and optimize decision-making based on predefined reward functions. However, MDPs require carefully designed reward structures tailored to specific systems, making them less adaptable across different applications. In contrast, differential privacy provides a formal and universal privacy guarantee without relying on system-specific rewards, ensuring protection through controlled randomness.

2.4. Word Differential Privacy

In the traditional supervisory control theory of discrete-event systems, system behavior is represented by formal languages. A language is a subset of the Kleene closure of the alphabet of a system, which is a set of finite strings or words composed of the event labels of the system. As stated previously, differential privacy is maintained by a mechanism that acts as a randomized function; for similar sensitive data, the mechanism produces roughly distinguishable outputs. The notion of "similarity" is defined by an adjacency relation, such as word adjacency in [23,24]. We recall the notions of word adjacency and word differential privacy in [24] as follows.

Definition 4. *(Word adjacency): Given a length $n \in \mathbb{N}$ and a distance $\omega \in \mathbb{N}$, the word adjacency relation with two strings s_1 and s_2 on Σ^n with respect to ω is defined as*

$$AR_{n,\omega} = \{(s_1, s_2) \in \Sigma^n \times \Sigma^n | d(s_1, s_2) \leq \omega\}. \tag{2}$$

where $\Sigma^n = \{s \in \Sigma^* \mid |s| = n\}$ *is the set of strings with length n.*

Given a parameter ω, two strings that have the same length are similar (or adjacent) with respect to ω if the Levenshtein distance between them is less than or equal to ω. Further, any pair (s_1, s_2) in $AR_{n,\omega}$ contains two similar strings. It is obvious that $AR_{n,\omega}$ is a tolerance relation over Σ^n.

Definition 5. *(Word differential privacy [24]): Consider a probability space* $(\Omega, \mathcal{F}, Pr)$ *that is a measure space such that the measure of the whole space is equal to one, where, as conventionally defined,* Ω *is a sample space, collecting all possible outcomes,* $\mathcal{F}$ *is a set of events, and Pr is a probability function that assigns each event in the event space a probability, which is a number between 0 and 1. Given a length* $n \in \mathbb{N}$*, a distance* $\omega \in \mathbb{N}$*, and a privacy parameter* $\epsilon > 0$*, a mechanism* $M : \Sigma^n \times \Omega \to \Sigma^n$ *is word* ϵ*-differentially private if for all* $L \subseteq \Sigma^n$ *and all* $(s_1, s_2) \in AR_{n,\omega}$*:*

$$Pr_\Omega[M(s_1) \in L] \leq e^\epsilon Pr_\Omega[M(s_2) \in L]. \tag{3}$$

Note that the elements belonging to Ω in the mechanism M are omitted in the paper. As defined, if the mechanism M meets word ϵ-differential privacy, the mechanism M can approximate similar strings with randomized versions, i.e., M should ensure that the randomized outputs of two adjacent strings are approximately indistinguishable for any attacker such that the attacker is not able to identify or estimate the real output of a system. Such an idea will be formalized in the following sections of the paper.

3. Problem Formulation

3.1. Intruder Model

In this section, we consider the problem of achieving differential privacy in stochastic DESs under partial observation to enforce almost k-step opacity. To address this problem, we define string adjacency and differential privacy with respect to the secrets of a system. Denote a probabilistic finite automaton by $(G, p) = (Q, \Sigma, \eta, q_0, p, Q_s)$, where $Q_s \subseteq Q$ is the set of secret states, and $p : Q \times \Sigma \to [0, 1]$, as defined before, is the transition probability function.

An attacker can infer from the observed strings at which state the system might be upon observation. Formally, let $\alpha \in P(L(G, p))$ be an observed string (an observation). Then, the current state estimate upon the occurrence of α is defined by $\hat{Q}_{G,p}(\alpha) = \{q \in Q \mid \exists s \in L(G) : P(s) = \alpha, q = \eta(s)\}$. The current state estimate can be computed by building an observer automaton (current state estimator) [28].

In some situations, an attacker is also interested in knowing which state the system could be in for some particular previous instant. Assume that an observation $\alpha \in P(L(G, p))$ is obtained, and we are interested in the state estimate of the system for the instant when only $\beta \preceq \alpha$ is executed. We define the delayed state estimate at the time instant when $\beta \preceq \alpha$ is exactly observed by $\hat{Q}_{G,p}(\beta|\alpha) = \{q \in Q \mid \exists s, t \in \Sigma^* : st \in L(A), P(s) = \beta, P(st) = \alpha, q = \eta(s)\}$. Intuitively, $\hat{Q}_{G,p}(\beta|\alpha)$ estimates the state of the system $|\alpha| - |\beta|$ steps prior to the instant when α is observed. Write $\hat{Q}_{G,p}(\beta|\alpha)$ as $\hat{Q}_G(\beta|\alpha)$ if p is of no concern.

3.2. Almost k-Step Opacity

In the k-step opacity problem in the context of DESs, a secret refers to a set of states that need to be protected and kept hidden from external observers. In practical applications, some strings (i.e., observations) can be observed by controllers and external observers, which can facilitate the supervisory control of a system or meet the needs for information disclosure. However, sharing information with third parties can create some security

issues. Once a secret state is visited, in the next k steps, strings generated by the system may lead attackers to infer that the system is or was in a secret state. For an automaton $G = (Q, \Sigma, \eta, q_0)$, the language as the set of strings that violate k-step opacity is defined as $L_k = \{s \in L(G) \mid \exists \alpha \preceq P(s) : |P(s)| - |\alpha| \leq k, \hat{Q}_G(\alpha | P(s)) \subseteq Q_s\}$.

Then, the k-step opacity requires $L_k = \emptyset$.

The definition of almost k-step opacity [14] is proposed to refine privacy protection for stochastic DESs based on the notion of almost current state opacity [13]. The concept of almost k-step opacity addresses the limitation that k-step opacity only offers binary classifications of system opacity without quantifying it, extending opacity evaluation to stochastic systems under partial observation. This extension provides a quantitative measure of the opacity, enhancing its applicability in various security-sensitive applications.

Note that once a string exposes the system's secret, any subsequent continuation of that string will also divulge the secret. We define the language (a subset of L_k) $L_k^P = \{s \in L_k \mid \forall t \prec s : t \notin L_k\}$ as the set of strings that violate k-step opacity for the first time. In essence, almost k-step opacity necessitates that the combined probability of strings violating k-step opacity logic is below a specified threshold θ. Almost k-step opacity introduces a probability threshold into k-step opacity, refining the concept for stochastic DESs.

Definition 6. *(Almost k-step Opacity): Let (G, p) be a PFA, $\Sigma_o \subseteq \Sigma$ be the set of observable events, $Q_s \subseteq Q$ be a set of secret states, $k \in \mathbb{N}$ be an integer, and $\theta < 1$ be a threshold value. The PFA (G, p) is said to be almost k-step opaque if $\sum_{s \in L_k^P} Pr(s) < \theta$.*

In this paper, we mainly guard against attackers capable of inferring that a system is or was in a secret state by observing the output strings. In a logical DES, almost k-step opacity can be verified by building the observer of a system that is represented by $O(G) = (Q_o, \Sigma_o, \eta_o, q_{0,o})$ [28]. In the setting of this paper, an attacker can partially observe the system (G, p), and all the observations belong to $P[L(G, p)]$, i.e., $L(O(G, p))$, where $O(G, p)$ is the observer of (G, p).

The strings observed after a secret state is reached within k steps, for which there exist no equivalent strings bypassing a secret state in k-steps, are called sensitive strings. We define $L_s = \{s \in L(G) \mid \exists \alpha \preceq P(s) : |P(s)| - |\alpha| = k, \hat{Q}_G(\alpha | P(s)) \subseteq Q_s\}$ be the set of all the sensitive strings in $L(O(G, p))$. To ensure that a system is almost k-step opaque, we need to protect the sensitive strings with a mechanism that satisfies differential privacy.

Note that in this paper, observations are divided into input observations (i.e., the projections of the strings generated by a system) and output observations (i.e., the strings observed by an external attacker). Let us abuse the terminology to say that these two types of observations correspond to the inputs and outputs of a system, respectively. A non-numerical query function f is proposed to describe the relation between the input and output of a system. If a system generates a string s, an observer will observe $f(P(s))$ (i.e., an output observation) instead of $P(s)$ as shown in Figure 2. Generally, if no mechanism that can affect the output observations exists, we assume that $f(P(s)) = P(s)$.

Example 3. *As shown in Figure 3a, there is a probabilistic automaton $(G, p) = (Q, \Sigma, \eta, q_0, p, Q_s)$ with two secret states marked in yellow, i.e., $Q_s = \{3, 8\}$. The probability of an event occurring in a certain state is marked in red. Let $\Sigma = \{a, b, c, d\}$ and $\Sigma_o = \{a, b, d\}$. In the observer of (G, p) as depicted in Figure 3b, there exists a secret state 3. In an almost k-step opacity problem, given $\omega = 1$, we can obtain the sensitive string $s_s = abd$, i.e., an attacker will be sure that secret state 4 was reached in one step before the current state when $s_s \in L_k$ is observed. The string $ab \in L_k^P$ that leads the system to the secret state 3 is included in string $s_s = abd$ (as a prefix of $s_s = abd$). Therefore, there is no need to deal with it separately. For secret state 8, string $ba = P(bcac)$ is not a secret*

since string bca with $P(bca) = ba$ exists such that non-secret state 7 is reached. The probability of occurrence of the string that violates k-step opacity is $\sum_{s \in L_k^P} Pr(s) = \frac{1}{2}$, and the system is almost k-step opaque if the given threshold θ is greater than 0.5.

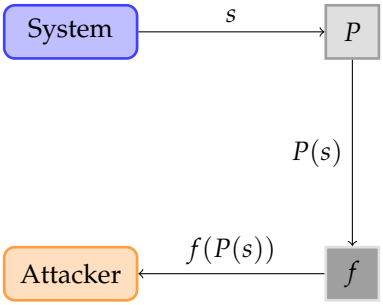
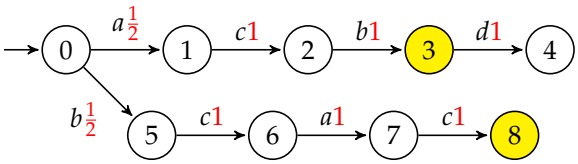
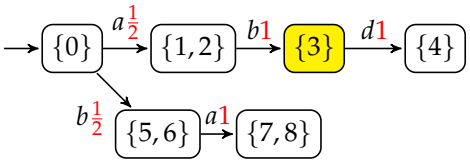


Figure 2. A system with a query function.



(**a**)



(**b**)

Figure 3. (**a**) A system A_{s_1} with secret states $\{4, 8\}$ and (**b**) the observer of A_{s_1}.

3.3. String Differential Privacy

For a symbolic system, word adjacency [23,24] explains the similarity of two words, and the word differential privacy mechanism ensures that the random outputs of two ω-adjacent words are indistinguishable (with respect to the parameter ω) for any recipient of their privatized forms. Based on word adjacency, we propose the sensitive string adjacency for partially observed stochastic systems to consider the similarity between sensitive strings and other available observations. Note that a "new" string that violates k-step opacity may appear; however, as long as the probability is small enough, the system can still be almost k-step opaque.

Definition 7. *(Sensitive string adjacency): Let (G, p) be a system containing secret states. Given a distance $\omega \in \mathbb{N}$ and a sensitive string $s_s \in L_s$ where L_s is the set of all sensitive strings, for a string s, the sensitive string adjacency relation with s_s is defined as*

$$AR_{s,\omega} = \{(s_s, s) \in L_s \times (\Sigma_o^* \setminus L_s) | d(s_s, s) \leq \omega, |s_s| = |s|\}. \tag{4}$$

For an attacker with knowledge of a system's structure, he/she may potentially infer that a secret state is or was reached by observing sensitive strings. Therefore, we devise a mechanism that adheres to differential privacy standards to generate similar output, thereby replacing the original sensitive strings. We propose a concept called string differential privacy to ensure that the random outputs of two ω-adjacent strings (one of them is a sensitive string) are similar for observers. In certain cases, some of these random outputs

can be strings that violate k-step opacity, i.e., the string $s \in L_k$, as long as their output probability is less than the preset threshold θ.

Definition 8. *(String differential privacy): Given a system $(G, p) = (Q, \Sigma, \eta, q_0, p, Q_s)$ with $Q_s \subseteq Q$, a probability space $(\Omega, \mathcal{F}, Pr)$, a distance $\omega \in \mathbb{N}$, and a privacy parameter $\epsilon > 0$, a mechanism $M_s : \Sigma^* \times \Omega \to \Sigma^*$ is string-ϵ-differentially private if for all $(s_1, s_2) \in AR_{s,\omega}$*

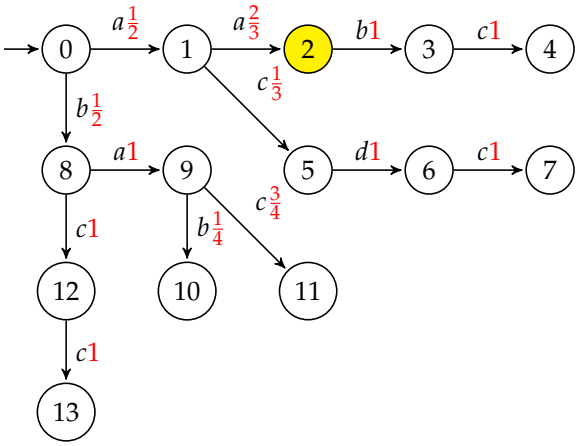
$$Pr_\Omega[M_s(s_1) \in \Sigma_o^*]] \leq e^\epsilon Pr_\Omega[M_s(s_2) \in \Sigma_o^*]. \tag{5}$$

Our primary goal is to provide privacy protection for the sensitive strings (i.e., input observations) by reducing the probability of strings that violate k-step opacity in output observations, and the length of the output observations remains unchanged after being substituted. We study the problem of designing a mechanism that satisfies string differential privacy to protect privacy and preserve the statistical value of the output for analysis.

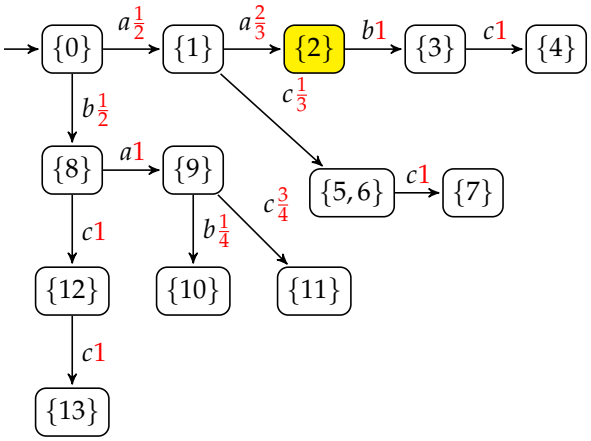
Problem 1. *Consider a probability space $(\Omega, \mathcal{F}, Pr)$. Given a system (G, p) and a sensitive string s_s, design a policy for generating substitution strings to enforce the system to be almost k-step opaque with a generation mechanism $M_s : \Sigma^* \times \Omega \to \Sigma^*$ that satisfies string differential privacy.*

To prevent sensitive strings from being exposed to attackers, we integrate a differential privacy mechanism into opacity enforcement and devise a protective policy. The differential privacy mechanism not only guarantees that the random output of a sensitive string is indistinguishable from that of another input observation but also ensures that the output maintains similarity to the sensitive string. Consequently, attackers are unable to discern the exact input observation (i.e., sensitive strings), and it becomes exceedingly improbable for them to determine whether the system has reached a secret state within k steps of the current state.

Example 4. *As shown in Figure 4a, there is a PFA $(G, p) = (Q, \Sigma, \eta, q_0, p, Q_s)$ with a secret state 2 marked in yellow, i.e., $Q_s = \{2\}$. Let the set of observable events be $\Sigma_o = \{a, b, c\}$. The probability of an event occurring in each state is marked in red. The observer of (G, p) is shown in Figure 4b. Since there exists the state $2 \in Q_s$ in the observer, given $k = 1$, string $s_s = aab$ is a sensitive string, and string aa belongs to L_k^P. We then have $\sum_{s \in L_k^P} Pr(s) = Pr(aa) = \frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}$. If the given θ is less than $\frac{1}{3}$, the system is not almost k-step opaque, and we need to enforce almost k-step opacity. Given the parameter $\omega = 2$, the sensitive string aab can be replaced with 12 substitution strings, and its generation probability mechanism satisfies string differential privacy. There are four substitution strings that belong to $P[L(G, p, s)]$, i.e., acc, bac, bab, and bcc, and other substitution strings belong to L_k. To enforce almost k-step opacity, we need to design a policy such that the mechanism for generating substitution strings satisfies string differential privacy, and the substitution strings satisfy $\sum_{s \in L_k^P} Pr(s) < \theta$.*



(**a**)



(**b**)

Figure 4. (**a**) A system A_{s_2} with a secret state 2 and (**b**) the observer of A_{s_2}.

4. Utility and Bound

Typically, a method leveraging differential privacy introduces random noise to a dataset, rendering it challenging or impossible for attackers to single out individual records. However, for non-numerical data like text or categorical data, preserving privacy while retaining data utility through noise addition may not be viable [32]. We adopt an exponential mechanism to uphold differential privacy for strings that fall under non-numerical data.

4.1. Information Utility

The exponential mechanism [20] selects outputs that balance privacy and utility in a principled manner, ensuring a trade-off between the privacy of the output and its utility. This approach utilizes the edit distance to quantify the deviation between two strings and privatizes the input string by randomly outputting a string close to the input. Generally, the information value of the output string is inversely proportional to its distance from the input string. A substitution string (i.e., output observations) that closely resembles the sensitive string (i.e., input observations) carries a higher information value. The utility function quantifies the information value of possible output strings generated by the exponential mechanism.

In a symbolic system, the utility of a private output string, as outlined in [23], is defined as the negative of the *Hamming distance* between the given input string and its corresponding output string. The utility function is defined as $u(s_i, s_o) = -d(s_i, s_o)$, where

s_i is the input string, and s_o is the output string. However, a negative utility does not conform to general cognition. In this paper, we define a utility function based on two positive numbers and the Levenshtein distance between an input string and an output string as follows.

Definition 9. *(Information utility): Given two positive numbers $\alpha > 0$ and $\beta > 0$ and an input string $s_i \in P[L(G,p)]$, the information utility of an output string $s_o \in P[L(G,p)]$ (with respect to α, β, and s_i) is*

$$u_{\alpha,\beta}(s_i, s_o) = \frac{1}{d(s_i, s_o) + \beta} + \alpha. \tag{6}$$

From Equation (6), the utility $u_{\alpha,\beta}(s_i, s_o)$ is a positive number due to $\alpha > 0$ and $\beta > 0$, and a larger value of α will give a larger value of $u_{\alpha,\beta}(s_i, s_o)$ given s_i and s_o. The value of α can be set to any positive number to act as an offset, affecting the overall utility of an output string s_o regardless of the Levenshtein distance between s_i and s_o. When the length of the string is not critical information, we can set β to a smaller value close to 0.

For different systems, we can use the parameter β to adjust the effect of the Levenshtein distance on utility. When α is a small value close to 0, the information utility is mainly affected by β. A larger value of β makes the effect of the Levenshtein distance on utility smaller. When both α and β are close to 0, as the disparity between the input and output strings widens, the information utility diminishes rapidly.

In terms of substitution operations, we inherently attribute higher utility to a substitution string (i.e., an output observation) that closely aligns with the sensitive string (i.e., an input observation). The information utility facilitates versatile comparisons and evaluations across various systems, ensuring equitable comparisons between strings.

4.2. Sensitivity Bound

In an exponential mechanism, the sensitivity bound of a utility function offers a framework to set boundaries that delineate the acceptable level of dissimilarity between two input strings [16]. By defining sensitivity bounds, we can efficiently restrict the potential disclosure of private data while holding the information utility.

The utility of two adjacent strings can be calculated by the utility function $u_{\alpha,\beta}(\cdot,\cdot)$. For the sake of simplicity, write $u_{\alpha,\beta}(\cdot,\cdot)$ as $u_\beta(\cdot,\cdot)$ or simply u_β. Given two adjacent input strings $s_1, s_2 \in P[L(G,p)]$ and an output string $s_o \in P[L(G,p)]$, the sensitivity bound [24] of u_β is defined as

$$\Delta u_\beta = \max_{s_o \in P[L(G,p)]} \max_{(s_1,s_2) \in AR_{s,\omega}} |u_\beta(s_1, s_o) - u_\beta(s_2, s_o)|. \tag{7}$$

Given $\beta > 0$ and $\omega \in \mathbb{N}$, the sensitivity bound of u_β satisfies

$$\Delta u_\beta \leq \frac{\omega}{\beta(\omega + \beta)}. \tag{8}$$

The proof is similar to that in [24].

Proof. Let $s_1, s_2 \in P[L(G,p)]$ be two adjacent input strings. Assume $d(s_2, s_o) = d(s_1, s_o) + c$. Since $d(s_1, s_2) \leq k$, we conclude $0 \leq c \leq k$. By this assumption, s_1 is closer to s_o than s_2, i.e., the utility of s_1 is higher than the utility of s_2. By removing the absolute value signs in Equation (7), it gives

$$\Delta u_\beta = \max_{s_o \in P[L(G,p)]} \max_{(s_1,s_2) \in AR_{s,k}} u_\beta(s_1, s_o) - u_\beta(s_2, s_o). \tag{9}$$

Combining this with Equation (8), we have

$$\begin{aligned}\Delta u_\beta = & \max_{s_o \in P[L(G,p)]} \max_{(s_1,s_2)\in AR_{s,k}} \frac{1}{d(s_1,s_o)+\beta} + \alpha \\ & - \left(\frac{1}{d(s_2,s_o)+\beta} + \alpha\right) \\ = & \max_{s_o \in P[L(G,p)]} \max_{s_1 \in P[L(G,p)]} \frac{1}{d(s_1,s_o)+\beta} \\ & - \frac{1}{d(s_1,s_o)+c+\beta}\end{aligned} \tag{10}$$

where $c \in \{0,1,\ldots,k\}$. For any $s_o \in P[L(G,p)]$, Δu_β is maximized as

$$\Delta u_\beta = \max_{c\in\{0,1,\ldots,k\}} \frac{c}{\beta(c+\beta)} \tag{11}$$

where $s_1 = s_o$ holds. If $c = k$, the right-hand side can be maximized, and we can obtain Inequality (8). □

Sensitivity bounds are pivotal in highlighting potential discrepancies in utility between two adjacent input strings. A broader sensitivity bound suggests a greater probability of generating a substitution string that significantly diverges from the sensitive string. This increased deviation reduces the useful information accessible to attackers, thereby bolstering system security.

The exponential mechanism determines the probability distribution of possible output strings by leveraging the sensitivity bound. A sensitivity bound ensures that the generated output strings fall within an acceptable deviation range. It is crucial to strike a balance, as an excessive deviation in the substitution string may lead to a misinterpretation by all observers.

Example 5. *Consider again the system in Figure 4a. Given the parameter $\omega = 2$, string acc is adjacent to the sensitive string aab with the Levenshtein distance of 2 ($\omega = 2$). We set $\alpha = \frac{1}{2}$ and $\beta = 1$ and let aab and acc be two input observations s_1 and s_2, respectively. If the string acc is selected as an output substitution string, we find $u_\beta(aab, acc) = \frac{5}{6}$ and $u_\beta(acc, acc) = \frac{3}{2}$. Then, the sensitivity bound is $\Delta u_\beta = \frac{3}{2} - \frac{5}{6} = \frac{2}{3}$, which is equal to $\frac{\omega}{\beta(\omega+\beta)}$. Supposing we choose a substitution string with a Levenshtein distance of 1 from the sensitive string as the output string, the utility difference will be less than the sensitivity bound Δu_β.*

5. Substitution String Generation Mechanism

5.1. String Exponential Mechanism

Leveraging both the utility function and the sensitivity bound, we will propose a string exponential mechanism to obtain the probability of each possible output string. This mechanism ensures that the output probability of strings is proportional to the utility.

Theorem 1. *Given a sensitive string $s_s \in P[L(G,p)]$, a string exponential mechanism M_s satisfies sensitive string-ϵ-differential privacy if the system (G,p) under the mechanism M_s outputs a substitution string $s \in \Sigma_o^*$ with probability proportional to $\exp(\frac{\epsilon u_\beta(s_s,s)}{2\Delta u_\beta})$.*

Proof. Consider a probability space $(\Omega, \mathcal{F}, Pr)$. Given two adjacent strings $s_1, s_2 \in \Sigma_o^*$ with $(s_1, s_2) \in AR_{s,\omega}$, we consider the ratio of the probability that the exponential mechanism outputs $s \in \Sigma_o^*$. By

$$\begin{aligned}
\frac{Pr_\Omega[M_s(s_1) = s]}{Pr_\Omega[M_s(s_2) = s]} &= \frac{\left(\frac{\exp(\frac{\epsilon u_\beta(s_1,s)}{2\Delta u_\beta})}{\sum_{s' \in \Sigma_o^*} \exp(\frac{\epsilon u_\beta(s_1,s')}{2\Delta u_\beta})}\right)}{\left(\frac{\exp(\frac{\epsilon u_\beta(s_2,s)}{2\Delta u_\beta})}{\sum_{s' \in \Sigma_o^*} \exp(\frac{\epsilon u_\beta(s_2,s')}{2\Delta u_\beta})}\right)} \\
&= \exp\left(\frac{\epsilon(u_\beta(s_1,s') - u_\beta(s_2,s'))}{2\Delta u_\beta}\right) \cdot \left(\frac{\sum_{s' \in \Sigma_o^*} \exp(\frac{\epsilon u_\beta(s_2,s')}{2\Delta u_\beta})}{\sum_{s' \in \Sigma_o^*} \exp(\frac{\epsilon u_\beta(s_1,s')}{2\Delta u_\beta})}\right) \qquad (12) \\
&\leq \exp\left(\frac{\epsilon}{2}\right) \cdot \exp\left(\frac{\epsilon}{2}\right) \cdot \left(\frac{\sum_{s' \in \Sigma_o^*} \exp(\frac{\epsilon u_\beta(s_1,s')}{2\Delta u_\beta})}{\sum_{s' \in \Sigma_o^*} \exp(\frac{\epsilon u_\beta(s_1,s')}{2\Delta u_\beta})}\right) \\
&= \exp(\epsilon),
\end{aligned}$$

we have

$$Pr_\Omega[M_s(s_1) = s] = \exp(\epsilon) Pr_\Omega[M_s(s_2) = s].$$

The inequality is proved by substituting the inequality $u_\beta(s_1, s') \leq u_\beta(s_2, s') + \Delta u_\beta$ into the equation. Thus, the string exponential mechanism satisfies string differential privacy, which completes the proof. □

5.2. Probability Distribution of Output

The observed strings are the output under the string exponential mechanism for an output observer. A system under the proposed protection mechanism generates a privatized output observation with the same length as the input. Given a sensitive string s_s and a distance ω, a Levenshtein automaton derived from s_s, and ω is represented as $A_{s_s,\omega} = (Q_{s_s,\omega}, \Sigma_o, \eta_{s_s,\omega}, q_{0,s_s,\omega})$ based on Definition 3. For convenience, the set of states is written as $Q_{s_s,\omega} = \{q_{i,d} | 0 \leq i \leq |s_s|, 0 \leq d \leq k\}$, where the first subscript (index i) of a state $q_{i,d}$ denotes the length i of a substitution string, and the second indicates the distance (index d) between the output string and the input string with the length i.

Note that when a sensitive string $s_s \in P[L(G, p)]$ is treated as an input observation, we may abuse the symbols s_i and s_s. Given a sensitive string $s_i \in \Sigma_o^*$ (i.e., an input string) and a Levenshtein automaton $A_{s_s,\omega} = (Q_{s_s,\omega}, \Sigma_o, \eta_{s_s,\omega}, q_{0,s_s,\omega})$, we implement a randomized policy to the output string and make its probability satisfy $Pr_o(s_o) : Q_{s_s,\omega} \times \Sigma_o \to [0, Pr_i(s_i)]$. Moreover, the probability of an input string is equal to the product of the probabilities of each original event in the passing state, i.e.,

$$Pr_i(s_i) = \prod_{n=1}^{|s_i|} p(\sigma_{s_i}^n | \eta(\sigma_{s_i}^1 \sigma_{s_i}^2 \dots \sigma_{s_i}^{n-1})). \qquad (13)$$

In this way, we privatize the string while Δu_β achieves maximum value $\frac{\omega}{\beta(\omega+\beta)}$ if $\omega \leq |s_i|$. The Levenshtein automaton assists in bypassing the computation of the proportionality constant for the string exponential mechanism. This means that we can circumvent the distance calculation between a sensitive string and every potential output string. Instead, we establish the sensitivity bounds as the utility difference between the sensitive strings and the string s_i that has the distance ℓ from the sensitive strings.

According to the string exponential mechanism, the output strings with the same distance from s_i have the same output probability. The probability of outputting a string with the distance ℓ from s_i is equal to the ratio of its output probability to the probability of outputting all the strings with distance ℓ from s_i. A Levenshtein automaton ensures that a substitution string s_o (i.e., the output string) is within Levenshtein distance ω from the original string s_i (i.e., the input string). Under the condition that the distance between the substitution string and the input string is ℓ, we can compute the probability of selecting an output string s_o with the given distance ℓ from s_i as

$$Pr(s_o, s_i)[d(s_o, s_i) = \ell] = \frac{\exp(\frac{\epsilon\beta(\omega+\beta)}{2\omega} \cdot (\frac{1}{\ell+\beta} + \alpha))}{\sum_{\lambda=0}^{\ell} \exp(\frac{\epsilon\beta(\omega+\beta)}{2\omega} \cdot (\frac{1}{\lambda+\beta} + \alpha))}, \tag{14}$$

derived from Theorem 1.

For a Levenshtein automaton, given a distance ℓ, we prune the states and transitions that do not correspond to this Levenshtein distance and only retain the paths from the initial state to the state that makes the Levenshtein distance of the output string from the input string equal to ℓ. A distance-limited Levenshtein automaton is defined as follows. For the algorithm to construct this automaton, one can refer to [24].

Definition 10. *(Distance-limited Levenshtein automaton): Given a Levenshtein automaton* $A_{s,\omega} = (Q_{|s|,d}, \Sigma, \eta_{s,\omega}, q_{0,0})$ *and a distance* ℓ*, a distance-limited Levenshtein automaton is a five-tuple* $\mathcal{A}^{\ell} = (Q^{\ell}, \Sigma_o, \eta^{\ell}, q_{0,0}, q_{\ell})$*, where* Q^{ℓ} *is the set of states,* Σ_o *is the set of observable events,* $\eta^{\ell} : Q^{\ell} \times \Sigma_o \to Q^{\ell}$ *is the transition function,* $q_{0,0}$ *is the initial state, and* q_{ℓ} *is the state corresponding to the length* $|s_s|$ *and the distance* ℓ*.*

Example 6. *Consider the Levenshtein automaton* $A_{bbc,2}$ *shown in Figure 1. By limiting the distance to be* $\ell = 2$*, we prune the states and transitions that cannot reach the state* 3^2*. After this operation, the distance-limited Levenshtein automaton with distance* $\ell = 2$ *is shown in Figure 5.*

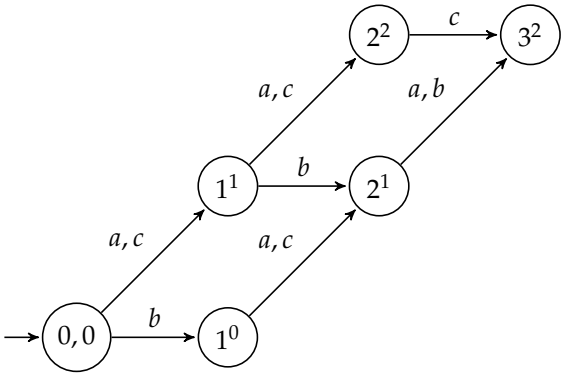


Figure 5. A distance-limited Levenshtein automaton $A_{bbc,2}$ with distance $\ell = 2$.

Based on the distance-limited Levenshtein automaton, we can obtain all possible output strings, and the sum of their probabilities is equal to the probability of the input string. According to Equation (14), we can obtain the output probabilities of strings sharing the same Levenshtein distance from an input string. The output probabilities of these strings can be calculated through the string exponential mechanism as

$$\begin{aligned} Pr_o(s_o) &= \frac{\exp(\frac{\epsilon\beta(\omega+\beta)}{2\omega} \cdot (\frac{1}{\ell+\beta} + \alpha))}{\sum_{\lambda=0}^{\ell} \exp(\frac{\epsilon\beta(\omega+\beta)}{2\omega} \cdot (\frac{1}{\lambda+\beta} + \alpha))} \cdot Pr_i(s_i) \\ &= \frac{\exp(\frac{\epsilon\beta(\omega+\beta)}{2\omega} \cdot (\frac{1}{\ell+\beta} + \alpha))}{\sum_{i=0}^{n} \binom{n}{i}(m-1)^i \exp(\frac{\epsilon\beta(\omega+\beta)}{2\omega} \cdot (\frac{1}{i+\beta} + \alpha))} \cdot Pr_i(s_i), \end{aligned} \tag{15}$$

where $Pr_i(s_i)$ is the probability of the input string and m is the number of the substitution strings permitted by the string exponential mechanism with the distance i from the input string. The number of all possible output strings is $\binom{n}{i}(m-1)^i$.

Based on the string exponential mechanism, we propose a substitution string generation mechanism in stochastic systems. The substitution string generation mechanism constructs a probabilistic control policy in the distance-limited Levenshtein automaton to achieve probability control of the output string to satisfy string-ϵ-differential privacy.

To determine the probability of outputting every observable event at a state $q \in Q^\ell$, we calculate the number of unique paths from this state to the state q_ℓ and denote this number by $V(q)$. Given a distance-limited Levenshtein automaton $\mathcal{A}^\ell = (Q^\ell, \Sigma_o, \eta^\ell, q_{0,0}, q_\ell)$, let $V(q_\ell) = 1$, and $q, q' \in Q^\ell$ be two states with $(q, \sigma, q') \in \eta^\ell$. For any $\sigma \in \Sigma_o$ such that $(q, \sigma, q') \in \eta^\ell$, we write the number of events that lead the state q to the state q' in $\mathcal{A}^\ell$ as $C_\sigma(q, q')$. Note that a distance-limited Levenshtein automaton is a deterministic automaton. For convenience, we sometime write $\eta^\ell(q, \sigma) = q'$ as $(q, \sigma, q') \in \eta^\ell$ for $q, q' \in Q^\ell$ and $\sigma \in \Sigma_o$, as a function can be represented by a relation. Given a distance-limited Levenshtein automaton $\mathcal{A}^\ell = (Q^\ell, \Sigma_o, \eta^\ell, q_{0,0}, q_\ell)$, we construct the probabilistic-control policy by using Algorithm 1 as follows.

Algorithm 1: Construction of a probabilistic control policy.

input : A distance-limited Levenshtein automaton $\mathcal{A}^\ell = (Q^\ell, \Sigma_o, \eta^\ell, q_{0,0}, q_\ell)$, an input string s_i, and the probability of each event in it

output: Probabilistic-control policy p

1 $V(q_\ell) \leftarrow 1$;
2 $Q' \leftarrow Q^\ell$;
3 $Q_c \leftarrow \emptyset$;
4 $d \leftarrow 0$;
5 **while** $d < \ell$ **do**
6 **for** *each* $q \in Q^\ell$ *such that* $(q, \sigma, q') \in \eta^\ell$ *and* $q' \in Q'$ **do**
7 $C_\sigma(q, q') \leftarrow |\{\sigma \in \Sigma_o | (q, \sigma, q') \in \eta^\ell\}|$;
8 $V(q) \leftarrow \sum_{(q,\sigma,q') \in \eta^\ell} C_\sigma(q, q') \cdot V(q')$;
9 $Q_c \leftarrow Q_c \cup \{q\}$;
10 **end**
11 **for** *each* $q \in Q_c$ *such that* $(q, \sigma, q') \in \eta^\ell$ **do**
12 $p(\sigma_{s_o}^n | q) \leftarrow \frac{V(q')}{V(q)} \cdot p(\sigma_{s_i}^n | \eta(\sigma_{s_i}^1 \dots \sigma_{s_i}^{n-1}))$;
13 **end**
14 $Q' \leftarrow Q_c$;
15 $Q_c \leftarrow \emptyset$;
16 $d \leftarrow d + 1$;
17 **end**

In steps 1–10, for any state $q \in Q^\ell$ that is coaccessible to state q_ℓ, we find the number of unique paths from it to q_ℓ by searching backward. Let $V(q_\ell) = 1$ for any q_ℓ and $Q' = Q^\ell$. Iteratively, $V(q)$ is obtained by considering a one-step transition at a time, and the states

that have been reached at each iteration are collected in Q_c. Then, the probability of outputting any event $\sigma \in \Sigma_o$ at q can be calculated in steps 11–13. Finally, we reset Q' to Q_c and proceed to the next iteration until the initial state is reached. The computational complexity of constructing the policy is $\mathcal{O}(|\ell||\Sigma_o||Q^\ell|^2)$.

Note that the values of $V(q)$ and $p(\sigma_{s_o}^n|q)$ are derived from Theorem 2 as follows.

Theorem 2. *Given a distance-limited Levenshtein automaton $\mathcal{A}^\ell = (Q^\ell, \Sigma_o, \eta^\ell, q_{0,0}, q_\ell)$, the number of unique paths from a state $q \in Q^\ell$ to the state q_ℓ $V(q)$ is the product of $V(q')$ and $C_\sigma(q, q')$, i.e.,*

$$V(q) = \sum_{(q,\sigma,q') \in \eta^\ell} C_\sigma(q, q') \cdot V(q'), \tag{16}$$

and the probability of outputting the event $\sigma_{s_o}^n \in \Sigma_o$ at the state $q = \eta(\sigma_{s_o}^1 \sigma_{s_o}^2 \dots \sigma_{s_o}^{n-1}) \in Q^\ell$ is product of $p(\sigma_{s_i}^n | \eta(\sigma_{s_i}^1 \sigma_{s_i}^2 \dots \sigma_{s_i}^{n-1}))$ and the ratio of $V(q')$ to $V(q)$, i.e.,

$$p(\sigma_{s_o}^n|q) = \frac{V(q')}{V(q)} \cdot p(\sigma_{s_i}^n | \eta(\sigma_{s_i}^1 \sigma_{s_i}^2 \dots \sigma_{s_i}^{n-1})) \tag{17}$$

where $(q, \sigma_{s_o}^n, q') \in \eta^\ell$, and $s_i \in \Sigma_o^*$ is an input string.

Proof. For any possible output string $s_o = \sigma_{s_o}^1 \dots \sigma_{s_o}^n \in \Sigma_o^*$, the probability of outputting s_o is equal to the product of the probabilities of each event occurrence. According to Equation (17), we have

$$\begin{aligned} Pr_o(s_o) &= p(\sigma_{s_o}^1|q^0) \cdot p(\sigma_{s_o}^2|q^1) \cdot \dots \cdot p(\sigma_{s_o}^n|q^n) \\ &= Pr(s_o, s_i)[d(s_o, s_i) = \ell] \cdot \frac{1}{\binom{n}{\ell}(m-1)^\ell} \cdot Pr_i(s_i) \\ &= \frac{\exp(\frac{\epsilon\beta(\omega+\beta)}{2\omega} \cdot (\frac{1}{\ell+\beta} + \alpha))}{\sum_{i=0}^n \binom{n}{i}(m-1)^i \exp(\frac{\epsilon\beta(\omega+\beta)}{2\omega} \cdot (\frac{1}{i+\beta} + \alpha))} \cdot Pr_i(s_i). \end{aligned} \tag{18}$$

This equation holds since each possible output string with the same distance from s_i has the same probability. The probability of outputting substitution strings with the same distance from an input string can by calculated by Equation (18), consistent with Equation (15), which completes the proof. □

Example 7. *Given a Levenshtein distance $\ell = 2$, the distance-limited Levenshtein automaton for all strings of length three with a distance 2 from a string aab is obtained by the method as shown in Figure 6. We remove the generated strings whose Levenshtein distance is not 2 to aab. For state 2^1, events a and b can lead the system from state 2^1 to reach state 3^2, and we have $C_\sigma(2^1, 3^2) = 2$ and $V(2^1) = C_\sigma(2^1, 3^2) \cdot V(3^2) = 2 \cdot 1 = 2$. For the state 1^1, two reachable states need to be computed separately, i.e., $C_\sigma(1^1, 2^2) = 2$ and $C_\sigma(1^1, 2^1) = 1$. Then, we have $V(1^1) = C_\sigma((1^1, 2^2) \cdot V(2^2) + C_\sigma((1^1, 2^1) \cdot V(2^1) = 2 \cdot 1 + 1 \cdot 2 = 4$. In state 1^1, the output probability of event b is $p(b|1^1) = \frac{V(2^2)}{V(1^1)} \cdot p(a_{s_i}|\eta(a_{s_i})) = \frac{1}{4} \cdot \frac{2}{3} = \frac{1}{6}$. Similarly, the output probability of every event in every state can be computed, which is marked in red in Figure 6.*

This example applies Theorem 2 to compute the probability of each event occurring in each state of the distance-limited Levenshtein automaton shown in Figure 6. By analyzing these event probabilities, we can derive the probability of the output string generated through the substitution string generation mechanism. In many cases, the probability of outputting strings that violate k-step opacity is lower than that of the original system output, demonstrating the effectiveness of the proposed mechanism in enhancing privacy protection.

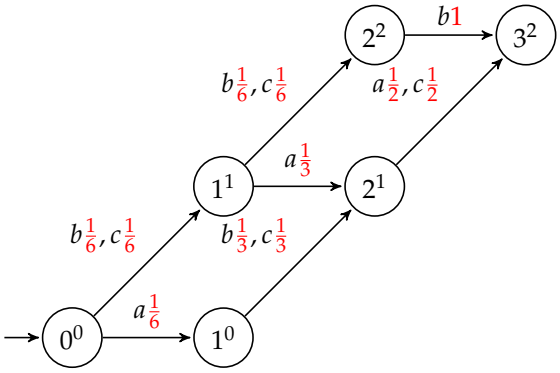


Figure 6. The distance-limited Levenshtein automaton for all strings of length 3 with a distance 2 from *aab*.

6. Almost *k*-Step Opacity Enforcement

6.1. Modified Levenshtein Automaton

Utilizing a substitution string generation mechanism for a system can reduce the probability of strings that violate k-step opacity being output since some strings produced by the substitution string generation mechanism conform to the system language, i.e., they do not breach k-step opacity. However, at some point, this reduction in probability is not enough to make $\sum_{s\in L_k^P} Pr(s) < \theta$. Therefore, a modified Levenshtein automaton is constructed to address this limitation.

Given the observer of a system $O(G,p) = (Q_o, \Sigma_o, \eta_o, q_{0,o})$ and a distance-limited Levenshtein automaton $\mathcal{A}^\ell = (Q^\ell, \Sigma_o, \eta^\ell, q_{0,0}, q_\ell)$, a modified Levenshtein automaton is defined as $\mathcal{A}_{a,o} = (Q_{a,o}, \Sigma_o, \eta_{a,o}, q_{0,a,o}, Q_m)$, where $Q_{a,o} = Q^\ell \times Q_o$ is the set of states, Σ_o is the set of observable events, $\eta_{a,o} : Q_{a,o} \times \Sigma_o \to Q_{a,o}$ is the (partial) transition function, $q_{0,a,o} = (q_{0,0}, q_{0,o})$ is the initial state, and $Q_m = \{q_m \in Q_{a,o} | q_m = (q_\ell, q_o)\}$ is a set of marked states with $q_o \in Q_o$.

We construct a modified Levenshtein automaton $\mathcal{A}_{a,o}$ by using Algorithm 2 as follows. Based on the transition function construction method of the product composition in [28], we obtain the transition function of $\mathcal{A}_{a,o}$ in steps 3–9. By using the trim operation [28] in step 10, the automaton $\mathcal{A}_{a,o}$ is constructed such that all the strings generated by $\mathcal{A}_{a,o}$ belong to $L(O(A_s))$, i.e., $P[L(A_s)]$. For a string s_s and a distance ω, given $\mathcal{A}^\ell$ with $\omega|s_s|$ states and $O(A_s)$ with $2^{|Q|}$ states, the automaton $\mathcal{A}_{a,o}$ has at most $\omega|s_s||\Sigma_o|2^{|Q|}$ transitions. Hence, the computational complexity of Algorithm 2 is $\mathcal{O}(\omega|s_s||\Sigma_o|2^{|Q|})$.

After obtaining the modified Levenshtein automaton, we use the probabilistic control policy in Algorithm 1 to ensure that the probabilities of outputting substitution strings meet the requirements of differential privacy. Note that there may be multiple states corresponding to the length $|s_s|$. For each $q_\ell \in Q^m$, we have $V(q_\ell) = 1$.

Algorithm 2: Construction of a modified Levenshtein automaton.

input : A system's observer $O(G,p)$, a sensitive string $s_s \in L(O(G,p))$, and a distance-limited Levenshtein automaton $\mathcal{A}^\ell = (Q^\ell, \Sigma_o, \eta^\ell, q_{0,0}, q_\ell)$

output: A modified Levenshtein automaton $\mathcal{A}_{a,o} = (Q_{a,o}, \Sigma_o, \eta_{a,o}, q_{0,a,o}, Q_m)$

$Q_{a,o} \leftarrow Q^\ell \times Q_o$;
$q_{0,a,o} \leftarrow (q_{0,0}, q_{0,o})$;
for *each* $(q_{i,d}, q_o), (q_{i',d'}, q'_o) \in Q_{a,o}$ **do**
  for *each* $\sigma_o \in \Sigma_o$ **do**
    if $(q_{i,d}, \sigma_o, q_{i',d'}) \in \eta^\ell$ *and* $(q_o, \sigma_o, q'_o) \in \eta_o$ **then**
      $\eta_{a,o} \leftarrow \eta_{a,o} \cup ((q_{i,d}, q_o), \sigma_o, (q_{i',d'}, q'_o))$;
    end
  end
end
$\mathcal{A}_{a,o} \leftarrow Trim((Q_{a,o}, \Sigma_o, \eta_{a,o}, q_{0,a,o}, Q_m))$, where Q_m is the set of marked states $\{(q_{i,d}, q_o) \in Q_{a,o} | i = |s_s|\}$;
return $\mathcal{A}_{a,o}$.

Theorem 3. *A system (G,p) using a modified Levenshtein automaton $\mathcal{A}_{a,o} = (Q_{a,o}, \Sigma_o, \eta_{a,o}, q_{0,a,o}, Q_m)$ to generate output strings is almost k-step opaque.*

Proof. Since the set of states $Q_{a,o}$ in the modified Levenshtein automaton is constructed by $Q^\ell \times Q_o$, $Q_{a,o} \subseteq Q \setminus Q_s$, all output strings s generated by the modified Levenshtein automaton belong to the language of the system, i.e., $s \in L(G,p)$. For all prefixes α of s, $\hat{Q}_G(\alpha | P(s)) \subseteq Q \setminus Q_s$, i.e., no outputs generated by the modified Levenshtein automaton violate k-step opacity. Therefore, we can obtain the probability of strings violating k-step opacity $\sum_{s \in L_k^p} Pr(s) = 0$, which must be less than the given threshold. According to Definition 6, the system can be determined to be almost k-step opaque. □

Example 8. *Consider system A_{s_2} in Figure 4a. Given the distance $\omega = 2$, we use Algorithm 2 to find a modified Levenshtein automaton $\mathcal{A}_{a,o}$ and apply a probabilistic control policy in Algorithm 1.*

The states $(3,2,\{7\}), (3,2,\{11\}) \in Q^\ell$ have $V(q_\ell) = 1$. For the state $(2,1,\{5,6\})$, only event c can drive the state to state $(3,2,\{7\})$, i.e., $C_\sigma((2,1,\{5,6\}),(3,2,\{7\})) = 1$, and we calculate $V((2,1,\{5,6\})) = C_\sigma((2,1,\{5,6\}),(3,2,\{7\})) \cdot V((3,2,\{7\})) = 1 \cdot 1 = 1$. For the initial state, there are two reachable states, and we have $V((0,0,\{0\})) = C_\sigma((0,0,\{0\}),(1,0,\{1\})) \cdot V((1,0,\{1\})) + C_\sigma((0,0,\{0\}),(1,1,\{8\})) \cdot V((1,1,\{8\})) = 1 \cdot 1 + 1 \cdot 1 = 2$. For event b that occurs at state $(0,0,\{0\})$, we obtain $p(b|(0,0,\{0\})) = \frac{V((1,1,\{8\}))}{V((0,0,\{0\}))} \cdot p(a_{s_i}|q_0) = \frac{1}{4}$. Similarly, the output probability of any possible event at all states can be calculated (marked in red) in Figure 7.

The probability of outputting acc as the substitution is

$$Pr_o(acc) = p(a|(0,0,\{0\})) \cdot p(c|(1,0,\{1\})) \cdot p(c|(2,1,\{5,6\})) \cdot Pr_i(abb) = \frac{1}{2} \cdot 1 \cdot 1 \cdot \frac{1}{3} = \frac{1}{6}$$

which is equal to the probability of outputting bac as the substitution $Pr_o(bac)$.

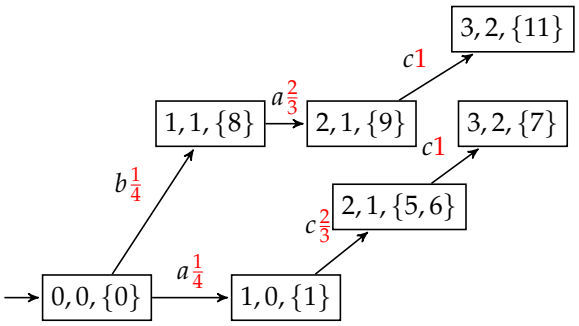


Figure 7. A distance-limited modified Levenshtein automaton $\mathcal{A}^{\ell}_{a,o}$ with a probabilistic control policy.

In this example, a probabilistic control policy from Algorithm 1 is applied to the distance-limited modified Levenshtein automaton in Figure 4a. Through this example, we can derive the probability distribution of the output strings under the given conditions, ensuring that no strings violating k-step opacity are included. Compared to the output string probabilities in Example 7 and Example 8, the proposed method provides stronger privacy protection, further enhancing the security of the distance-limited modified Levenshtein automaton.

6.2. Enforcement Policies

For some stochastic systems, the substitution string generation mechanism can be used to replace observable sensitive strings and enforce almost k-step opacity. Since a portion of the strings output by the substitution string generation mechanism belongs to the system language, they do not violate k-step opacity, i.e., the probability of outputting the strings that violate k-step opacity decreases.

Example 9. *As shown in Figure 6, the substitution strings that do not belong to the system language include the prefix-closure of ab, aca, bb, bb, bc, c, i.e.,* $ab, aca, bb, baa, c \in L_k^P$. *we can calculate* $\sum_{s\in L_k^P} Pr(s) = Pr(ab) + Pr(aca) + Pr(bb) + Pr(bc) + Pr(baa) + Pr(c) = \frac{1}{18} + \frac{1}{36} + \frac{1}{36} + \frac{1}{36} + \frac{1}{36} + \frac{1}{9} = \frac{5}{18}$. *If the given threshold* θ *is less than* $\frac{1}{3}$ *and greater than* $\frac{5}{18}$, *then using the substitution string generation mechanism output can make the system satisfy almost k-step opacity.*

However, for some systems, the output generated by a substitution string generation mechanism still does not meet the almost k-step opacity. For these systems, we introduce a modified Levenshtein automaton to ensure that all possible output strings belong to the system language and are not sensitive strings.

For a system, the method used depends on the value of θ. When the system requires higher privacy protection (corresponding to the situation where the given θ is very small), we use a modified Levenshtein automaton to ensure that an attacker will not observe any string that should not be generated.

In Example 9, when the given threshold θ is less than $\frac{5}{18}$, the substitution string generation mechanism will not work. Then, we need to use a modified Levenshtein automaton to determine the output to enforce almost k-step opacity on the system under the differential privacy mechanism.

In some cases, a system that fails to satisfy almost k-step opacity cannot achieve this standard because there may not always exist a safe observation that belongs to the system language and a sensitive string similar to the sensitive string for a given distance, i.e., there is no substitution string that does not violate k-step opacity. For example, consider the following scenario.

Example 10. *Consider again the automaton A_{s_1} in Figure 3. When the given threshold $\theta = 0.2$ is less than $\sum_{s \in L_k^P} Pr(s)$, the system does not satisfy almost k-step opacity. Given the parameter $\omega = 2$, all the strings that are adjacent to the sensitive string abd are not in $P[L(A_{s_1})]$, i.e., there does not exist any substitution string that allows the system to achieve almost k-step opacity.*

We enforce almost k-step opacity to systems that do not meet this criterion if there are substitution strings that do not violate k-step opacity. For systems with moderate security requirements or a substantial number of substitution strings that do not violate k-step opacity, we employ a substitution string generation mechanism to generate the output. In other cases, a modified Levenshtein automaton is utilized for systems with high privacy protection requirements or fewer substitution strings that do not violate k-step opacity.

Compared with the enforcement of infinite-step opacity and k-step opacity via the insertion mechanism [9], the proposed differential privacy mechanism protects system privacy while retaining the information value of the output. The degree of this retention can be adjusted as needed. The insertion function requires inserting fictitious events into the system output, whereas the differential privacy mechanism developed in this research keeps the length of the output string unchanged. We extend the scope of k-step opacity enforcement to stochastic systems, achieving refined system privacy protection through almost k-step opacity enforcement. Depending on the protection scenario, we use the substitution string generation mechanism for lightweight protection and the modified Levenshtein automaton for higher levels of privacy protection.

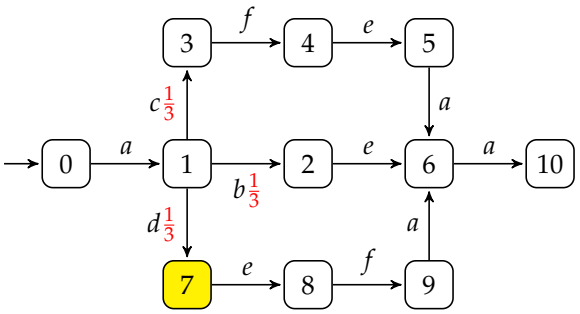
6.3. Application

In medical systems, patients' health data are continuously monitored to provide real-time insights to healthcare providers, enabling timely medical interventions. The health status of patients is modeled as a set of discrete states. Changes in physiological indicators and external events have a probabilistic chance of triggering state transitions, thus modeling the system as a probabilistic automaton. The medical system must allow managers to access data for analysis and count events such as examinations. The disease status of patients is highly sensitive, and external observers should not be able to identify whether a patient is or has been in a specific disease state. Directly altering the strings that lead to secret states or using insertion functions can result in data distortion. A differential privacy mechanism is more suitable for this type of stochastic system, as it preserves the privacy of patients' health data while maintaining the statistical integrity of the information.

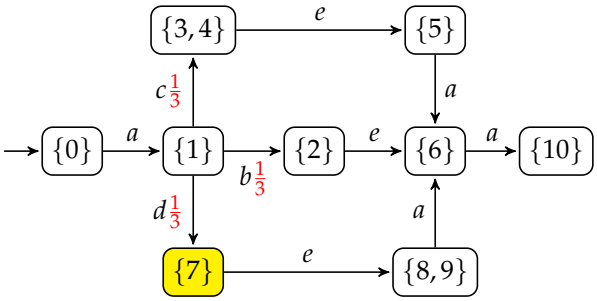
Here is a medical system of monitoring and treating the patient's condition; it has three possible developments modeled as shown in Figure 8a. State 7 marked in yellow, for which there is a one-third probability of being reached, is related to patients' privacy, i.e., this state is a secret state. The set of observable events is presented as $\Sigma_o = \{a, b, c, d, e\}$ and the event f is unobservable in the system. For outside observers, the observation of the modeling is shown in Figure 8b. In all figures of this section, the probability of an event occurring in each state is marked in red.

By setting parameter $k = 2$, the string *adea* is the sensitive string that must be replaced. Given the distance $\omega = 1$, based on the substitution string generation mechanism, we can obtain 16 substitution strings as shown in Figure 9, among which strings *abea* and *acea* belong to the system language. It can be calculated that $\sum_{s \in L_k^P} Pr(s) = \frac{7}{24}$, i.e., an observer who knows all possible output languages of the system has this chance of finding that the system is in the secret state within two steps. The manager of the medical system will set the threshold according to the observer's situation. When the threshold θ is set to be less than 7/24, a modified Levenshtein automaton will be used to output strings.

We can obtain the modified Levenshtein automaton as shown in Figure 10 according to Algorithm 2. In this case, only stings *abea* and *acea* can be output.



(**a**)



(**b**)

Figure 8. (**a**) Modeling of the medical system and (**b**) the observer of (**a**).

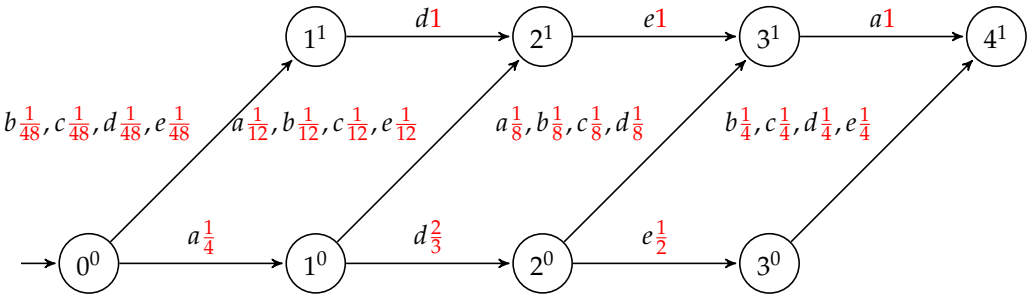


Figure 9. The distance-limited Levenshtein automaton of the medical system with a distance 1.

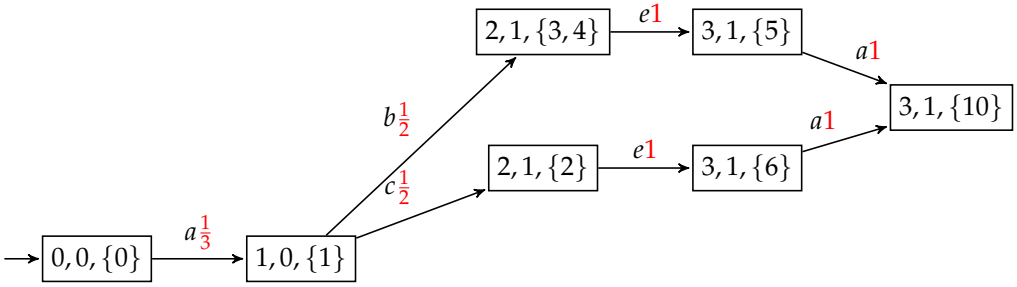


Figure 10. The modified Levenshtein automaton of the medical system.

In practical applications, the reported strategy in this paper optimizes privacy protection by tailoring enforcement methods to the specific security needs of a system. It provides a balanced approach between security and utility, providing customized measures for systems with varying security requirements. This fine-grained approach enhances system security and resilience.

7. Conclusions

In this paper, we tackle the challenge of enforcing almost k-step opacity in a stochastic partially observed discrete-event system, whose behavior is characterized by a language that is a collection of strings (along with the generation probability) defined over the alphabet of the system. The opacity enforcement is achieved by replacing the sensitive strings with substitution strings. We aim to reduce the likelihood that an attacker can deduce that the system has reached a secret state within k steps from an observed string. A differential privacy mechanism is introduced to enforce almost k-step opacity while maintaining the statistical significance of the output.

We devise a utility function and establish sensitivity bonds to evaluate the information value of substitution strings. A probabilistic control policy based on an exponential mechanism is proposed to ensure that the probabilities of substitution strings meet the requirements of differential privacy. We use a substitution string generation mechanism for lightweight protection and a modified Levenshtein automaton for higher levels. We enforce almost k-step opacity by reducing the probability of strings that violate k-step opacity and realize the refined protection of the system. To move forward, we will explore the use of this tailored differential privacy mechanism to enforce opacity in a decentralized architecture of a discrete event system, where different observational sites may have different substitution string generation mechanisms. Further, it is interesting to explore a system with multiple agents (observers) who may want to disclose each other's secrets while keeping their own concealed.

Author Contributions: Conceptualization, Z.L.; methodology, R.Z.; software, M.U.; validation, M.U. and Z.L.; writing—original draft preparation, R.Z.; writing—review and editing, M.U. and Z.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research is partially supported by the Science and Technology Fund, FDCT, Macau SAR, under Grant 0029/2023/RIA1.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bryans, J.W.; Koutny, M.; Mazaré, L.; Ryan, P.Y.A. Opacity generalized to transition systems. *Int. J. Inf. Secur.* **2008**, *7*, 421–435. [CrossRef]
2. Lin, F. Opacity of discrete event systems and its applications. *Automatica* **2021**, *47*, 496–503. [CrossRef]
3. Wu, Y.; Lafortune, S. Comparative analysis of related notions of opacity in centralized and coordinated structures. *Discret. Event Dyn. Syst. Theory Appl.* **2013**, *23*, 307–339. [CrossRef]
4. Zhou, S.; Yu, J.; Yin, L.; Li, Z. Security quantification for discrete event systems based on the worth of states. *Mathematics* **2023**, *11*, 3629. [CrossRef]
5. Liang, Y.; Liu, G.; El-Sherbeeny, A.M. Polynomial-time verification of decentralized fault pattern diagnosability for discrete-event systems. *Mathematics* **2023**, *11*, 3998. [CrossRef]
6. Saboori, A.; Hadjicostis, C.N. Verification of k-step opacity and analysis of its complexity. *IEEE Trans. Autom. Sci. Eng.* **2011**, *8*, 549–559. [CrossRef]
7. Saboori, A.; Hadjicostis, C.N. Verification of infinite-step opacity and complexity considerations. *IEEE Trans. Autom. Control* **2012**, *57*, 1265–1269. [CrossRef]
8. Saboori, A.; Hadjicostis, C.N. Verification of initial-state opacity in security applications of discrete-event systems. *Inf. Sci.* **2013**, *246*, 115–132. [CrossRef]
9. Liu, R.; Lu, J. Enforcement for infinite-step opacity and k-step opacity via insertion mechanism. *Automatica* **2022**, *140*, 110212. [CrossRef]
10. Ji, Y.; Wu, Y.; Lafortune, S. Enforcement of opacity by public and private insertion functions. *Automatica* **2018**, *93*, 369–378. [CrossRef]

11. Balun, J.; Masopust, T. Verifying weak and strong *k*-step opacity in discrete-event systems. *Automatica* **2023**, *155*, 111153. [CrossRef]
12. Bryans, J.W.; Koutny, M.; Mu, C. Towards quantitative analysis of opacity. In Proceedings of the Trustworthy Global Computing: 7th International Symposium, Newcastle upon Tyne, UK, 7–8 September 2012.
13. Saboori, A.; Hadjicostis, C.N. Current-state opacity formulations in probabilistic finite automata. *IEEE Trans. Autom. Control* **2014**, *59*, 120–133. [CrossRef]
14. Yin, X.; Li, Z.; Wang, W.; S. Li. Infinite-step opacity and *k*-step opacity of stochastic discrete-event systems. *Automatica* **2019**, *99*, 266–274. [CrossRef]
15. Dwork, C. The differential privacy frontier (extended abstract). In Proceedings of the 6th Theory of Cryptography Conference, Berlin, Heidelberg, Germany, 15–17 March 2009.
16. Dwork, C. Differential privacy: A survey of results. In Proceedings of the 5th International Conference on Theory and Applications of Models of Computation, Berlin, Heidelberg, Germany, 25–29 April 2008.
17. Dwork, C.; Lei, J. Differential privacy and robust statistics. In Proceedings of the 41st Annual ACM Symposium on Theory of Computing, 31 May–6 June 2009.
18. McSherry, F.; Talwar, K. Mechanism design via differential privacy. In Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, Providence, RI, USA, 21–23 October 2007.
19. Dwork, C.; Roth, A. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* **2013**, *9*, 211–406. [CrossRef]
20. Huang, Z.; Kannan, S. The Exponential mechanism for social welfare: Private, truthful, and nearly optimal. In Proceedings of the 53rd Annual Symposium on Foundations of Computer Science, New Brunswick, NJ, USA, 20–23 October 2012.
21. Liu, F. Generalized Gaussian mechanism for differential privacy. *IEEE Trans. Knowl. Data Eng.* **2001**, *31*, 747–756. [CrossRef]
22. Li, C.; Miklau, G.; Hay, M.; McGregor, A.; Rastogi, V. The matrix mechanism: Optimizing linear counting queries under differential privacy. *VLDB J.* **2015**, *24*, 757–781. [CrossRef]
23. Chen, B.; Leahy, K.; Jones, A.; Hale, M. Differential privacy for symbolic systems with application to Markov Chains. *Automatica* **2023**, *152*, 152–164. [CrossRef]
24. Jones, A.; Leahy, K.; Hale, M. Towards differential privacy for symbolic systems. In Proceedings of the IEEE 2019 American Control Conference, Philadelphia, PA, USA, 10–12 July 2019.
25. Al-Sarayrah, T.A.; Li, Z.; Zhu, G.; El-Meligy, M.A.; Sharaf, M. Verification and enforcement of (ϵ, ξ)-differential privacy over finite steps in discrete event systems. *Mathematics* **2023**, *11*, 4991. [CrossRef]
26. Teng, Y.; Li, Z.; Yin, L.; Wu, N. State-based differential privacy verification and enforcement for probabilistic automata. *Mathematics* **2023**, *11*, 1853. [CrossRef]
27. Zhang, J.; Dong, Y.; Yin, L.; Mostafa, A. M.; Li, Z. Opacity enforcement in discrete event systems using differential privacy. *Inf. Sci.* **2025**, *688*, 121284. [CrossRef]
28. Cassandras, C.G.; Lafortune, S. In *Introduction to Discrete Event Systems*, 2nd ed.; Springer: New York, NY, USA, 2008; pp. 53–143.
29. Schulz, K.U.; Mihov, S. Fast string correction with Levenshtein automata. *Int. J. Doc. Anal. Recognit.* **2002**, *5*, 67–85. [CrossRef]
30. Ny, J.L.; Pappas, G. Differentially private filtering. *IEEE Trans. Autom. Control* **2014**, *59*, 341–354.
31. Garcia, F.; Emmanuel, R. Markov decision processes. *Markov Decis. Processes Artif. Intell.* **2013**, 1–38.
32. Dwork, C.; Roth, A. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* **2014**, *9*, 211–407. [CrossRef]