Differentially Private Inductive Miner

Max Schulze*
Institute for IT Security
Universität zu Lübeck
max.schulze@student.uni-luebeck.de

Yorck Zisgen*
Chair of Business Informatics and Process Analytics
University of Bayreuth
yorck.zisgen@uni-bayreuth.de

Moritz Kirschte
Institute for IT Security
Universität zu Lübeck
m.kirschte@uni-luebeck.de

Esfandiar Mohammadi
Institute for IT Security
Universität zu Lübeck
esfandiar.mohammadi@uni-luebeck.de

Agnes Koschmider
Chair of Business Informatics and Process Analytics
University of Bayreuth, Fraunhofer FIT
agnes.koschmider@uni-bayreuth.de

2024 6th International Conference on Process Mining (ICPM) | 979-8-3503-6503-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICPM63005.2024.10680684

***Abstract*—Protecting personal data about individuals, such as event traces in process mining, is an inherently difficult task since an event trace leaks information about the path in a process model that an individual has triggered. Yet, prior anonymization methods of event traces like k-anonymity or event log sanitization struggled to protect against such leakage, in particular against adversaries with sufficient background knowledge. In this work, we provide a method that tackles the challenge of summarizing sensitive event traces by learning the underlying process tree in a privacy-preserving manner. We prove via the so-called Differential Privacy (DP) property that from the resulting summaries no useful inference can be drawn about any personal data in an event trace. On the technical side, we introduce a differentially private approximation (DPIM) of the Inductive Miner. Experimentally, we compare our DPIM with the Inductive Miner on 14 real-world event traces by evaluating well-known metrics: fitness, precision, simplicity, and generalization. The experiments show that our DPIM not only protects personal data but also generates faithful process trees that exhibit little utility loss above the Inductive Miner.**

***Index Terms*—Process mining, Differential Privacy, Process Discovery, Privacy Utility Trade-off**

I. Introduction

Privacy risks in process mining on potentially sensitive event logs impede the valuable extraction of insights from real-world event logs, as extracted process trees can provide significant transparency into business processes. From a privacy protection perspective, however, event logs are particularly challenging: in the worst case every trace is fully associated with only one individual; hence, the footprint of an individual on an event log is very high. A study on re-identification risks in event logs showed that there are significant privacy leakages in the vast majority of the event logs used widely by the process mining community [7], [15].

The literature contains proposals for protecting privacy for event logs or in process mining, respectively [8]–[11], [13]. Yet, four of these prior approaches [8]–[11] do not provide strong provable privacy guarantees for a process mining algorithm against attackers with strong background knowledge as summarized in Sec. III. Advancements in privacy-preserving computations have demonstrated that techniques, such as k-anonymity or event log sanitization, falter when an adversary possesses sufficient background knowledge [14]. Specifically, these methods struggle to provide substantial guarantees against future adversaries. A state-of-the-art privacy notion considering a strong attacker is differential privacy (DP), which requires that the impact of single traces on the final process model is limited; in particular, DP guarantees imply limited impact of outliers and, as a result, significantly mitigate re-identification risks. While there is work on querying traces in a differentially private manner [13], query-based information extraction only works for a limited number of queries to guarantee privacy. With every query, additional information about underlying sensitive traces is leaked. As an alternative, if the mining strategy itself guarantees differential privacy, the resulting process representation could be arbitrarily used, e.g., for generating synthetic event traces, without causing any additional privacy leakage.

This paper introduces a novel privacy-preserving process mining algorithm called `Differentially Private Inductive Miner` (DPIM) which produces a process tree (PST) based on an event log. DPIM replaces privacy-leaking operations of the Inductive Miner on single traces with privacy-compliant operations on sets of traces. These operations are designed such that we show strong differential privacy guarantees while approximating the functionality of the Inductive Miner as shown by the common quality measures (i.e., fitness, precision, simplicity, and generalization). We evaluated our algorithm against the Inductive Miner based on 14 real-world event logs in terms of accuracy and privacy.[1] The trade-off between privacy gain and data utility loss depends on the chosen degree of ε (lower means more privacy), event log complexity (simpler is better), and event log size (larger is better). In Fig. 4 we quantify this trade-off and show that differential privacy can be obtained on real-world event logs with process models that keep a fitness of 0.95, precision of 0.9, simplicity of 0.7, and generalization of 0.8.

Structure. Sec. II defines the problem on an exemplary use

*These authors contributed equally to this work.

[1]Code, data, and evaluations are available at github.com/Schulze-M/DPIM

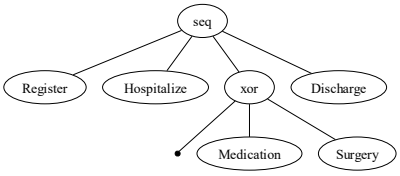


Fig. 1: PST on Trace Variants 1 to *3*

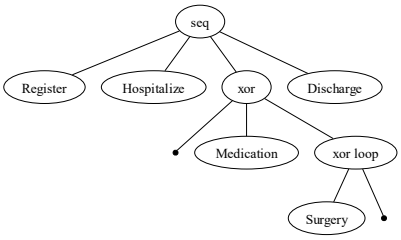


Fig. 2: PST on Variants 1 to *4*

Variant	Count	Trace
1	63x	⟨ R, H, M, D ⟩
2	25x	⟨ R, H, S, D ⟩
3	12x	⟨ R, H, D ⟩
4	1x	⟨ R, H, S, S, D ⟩
5	0x	⟨ R, M, D ⟩

TABLE I: Exemplary Trace Log

case. Sec. III compares our approach to related approaches. Sec. IV presents how to incorporate privacy guarantees into the Inductive Miner. Sec. V describes our *DPIM*. Evaluation results are summarized in Sec. VI, and a conclusion is drawn in Sec. VII.

II. Problem Statement

Process mining algorithms extract insights about an organization's processes from recorded and potentially sensitive event log data. Yet, it carries the inherent risk that what is disclosed may be private. We assume a medical treatment process in a hospital (Fig. 1) where patients are *registered*, *hospitalized*, and in slight medical cases can be *discharged* immediately, while in more severe medical cases they either receive *medication* or undergo *surgery*, before being *discharged*. Each trace represents a possibly unique path in the treatment process that can be directly linked to the health situation of an individual patient. Thus, anonymizing an event log can not keep the original traces unmodified as the traces pose a re-identification risk otherwise [15].

On a process model based on unmodified traces, we can construct the following attack: assume an attacker, e.g. a data controller or business analyst, with authorized access to query aggregated data such as process models, but no direct access to the underlying data. The attacker has the background knowledge to perform a *difference attack* [4], e.g. the attacker queries the process model at a timepoint where the trace variants *1* to *3* in Table I are included (*R:Register, H:Hospitalize, M:Medication, S:Surgery, D:Discharge*) (cf. Fig. 1). After learning that Jane visited the hospital, the attacker queries again and finds out that the process model changed as seen in Fig. 2 and deducts that a trace variant like number *4* in Table I caused that difference. Thus, the attacker learns that Jane must have undergone surgery at least twice.

To counter privacy attacks without significantly compromising utility, we seek to anonymize traces by summarizing as many common behaviors as possible, e.g. by working on the trace variants or directly-follows relation. Intuitively, the more persons exhibit a common behavior, like having the same trace or directly following activities, the better a single person can blend in the crowd. Thus, the more we summarize, the stronger the privacy becomes. We can accelerate this advantage with strong privacy protection mechanisms like differential privacy (DP), where a high-utility DP mechanism has a privacy protection level $\varepsilon \in \mathcal{O}(1/n)$ where n is the number of summarized elements.

Based on summarized traces, we propose to construct a process structure tree (PST) where it is possible to synthesize an event log or build a Petri net. To create a privacy-preserving PST, we propose a method called *DPIM* that is based on the Inductive Miner and uses DP. DP guarantees that the resulting PST is protected against the addition, removal, or change triggered by any person's trace. Thus, even those attackers with unlimited background knowledge are unable to infer the influence of a single trace on the process model and thereby recreate the nature of the trace.

III. Related Work

Closely related to our approach are approaches that focus on differentially private event log sanitization [8]–[11], [13]. Yet, two of them [8], [9] admit to privacy limitations which leaves the privacy protection unclear; hence we do not compare our utility to their approach. For the other two papers [10], [11], we found a counterexample of a component of their approaches in the form of a privacy attack that violates the differential privacy property. Hence, it is unclear which degree of privacy protection these papers achieve, rendering a direct utility comparison unfair.

Elkoumy et al. [8] aims to generate a directly-follows graph (DFG) differentially private by determining the amount of noise needed and then noising the weight of the arcs. However, they state that they do not add or delete arcs. Yet, as per our example in Table. I the person with trace variant *4* can alter whether the DFG contains the activity pair $\langle S, S \rangle$. Thus this person can add or remove an arc in a DFG since any activity pair is represented in a DFG. Hence, their protection method is not DP, as an attacker can observe in the DFG whether *4* was present.

Elkoumy et al. [9] anonymizes event log timestamps and removes some traces, assuming limited background knowledge of an attacker. This removal is based on prior knowledge. However, they state that no new trace variants are introduced. Yet, as per our example in Table I and Fig. 2, the person with trace variant *5* can alter the process by introducing a new unforeseen trace that is not present in the data, e.g. the activity pair $\langle R, M \rangle$. This indicates that their method does

not meet differential privacy standards, as an attacker with extensive background knowledge can detect the presence of trace variant *5* in the anonymized data.

Fahrenkrog et al. [10] propose a DP algorithm that releases an anonymized trace variant distribution using the exponential mechanism in combination with Laplace noise. From the trace variant distribution, their algorithm builds a DFR. Although they add noise to selected activity pair frequencies of the DFR, the algorithm preselects a limited number of activity pairs based on the event log, which is data-dependent. This preselection, driven by semantic correctness or a k-follows score, undermines differential privacy. As new trace variants can add a huge number of new plausible activity pairs which put previously distant activities closer and leads to a lower k-follows score, a huge number of new activity pairs are accepted. Thus, an attacker can infer whether the new trace variant is used by how many activity pairs are released.

The PRIPEL framework [11] anonymizes traces using a random timestamp shift and noise added to the count of trace variants. The usefulness of these anonymized traces is improved by selecting only those similar to the original traces via the Levensthein distance. Yet, as per our example in Table I, the person with trace variant *4* could influence this selection if its trace is similar to an anonymized one. This suggests that the method does not ensure differential privacy, as an attacker can infer the presence of trace variant *4* from the selection process.

Mannhardt et al. [13] is similar to our approach as it introduces a method to generate the frequency of all directly-follows relation (DFR) or a prefix-tree differentially private. Our DPIM concentrates on generating a faithful process tree (PST), which includes a parallel and loop cut selection.

Further related to our work are research areas that focus on privacy measures that do not provide an equally strong guarantee of privacy as DP or are parallel to our work by protecting the mining process cryptographically. K-anonymity methods to protect event logs have been proposed [18], [19]. However, these are not robust against unlimited background knowledge like differential privacy (DP) and are hence vulnerable to intersection attacks [3]. Rafiei et al. [16], [17] suggests disclosure risk quantification measures. These do not provide techniques to guarantee privacy. Burattin et al. [2] suggest outsourcing process mining while maintaining the confidentiality of event logs and discovered process models using encryption and cryptography, yet the decrypted model itself is not protected with DP.

IV. Preliminaries

Differential Privacy (DP) [5] is a mathematical framework that enables the analysis and sharing of sensitive data while preserving the privacy of individual records within the data set. It guarantees privacy by introducing a controlled amount of noise to the data, ensuring that the presence or absence of any individual's data does not significantly influence the results of queries. This privacy guarantee is quantified by a privacy parameter ε, with lower values offering stronger privacy protection at the expense of reduced data utility. The *post-processing theorem* [6] states that no further processing of the output of a DP mechanism can increase privacy leakage.

Algorithm 1 DP Rejection Sampling [12, Algorithm 1]

Input: threshold t, probability $\gamma \leqslant 1$, privacy budget $\varepsilon_0 \leqslant 1$, number of steps $T \geqslant \max\left(\frac{1}{\gamma}\ln\frac{2}{\varepsilon_0}, 1+\frac{1}{e\gamma}\right)$, ε_1-DP mechanism $M(D)$
for $j = 1, \ldots, T$ **do**
  draw $(x, q) \sim M(D)$
  if $q \geqslant t$ **then return** (x, q)
  flip γ-biased coin s.t. with probability γ: **return** $\perp$
return $\perp$

Definition 1 (**Bounded Differential Privacy**). *Two event logs L, L' are neighboring (written $L \sim L'$) if they differ in at most one trace. Let $\mathcal{R}$ be the set of random variables over some set O. A randomized mechanism $\mathcal{M} : \mathcal{L} \to \mathcal{R}$ is ε-differentially private, with $\varepsilon > 0$, if for all $\mathcal{S} \subseteq O$, for all n, and all neighboring event logs L, $L' \in \mathcal{L}$ with $n = |L| = |L'|$:* $\Pr[M(L) \in \mathcal{S}] \leqslant \exp(\varepsilon)\Pr[\mathcal{M}(L') \in S]$.

The Laplace Mechanism $\mathcal{M}_{L,q,\varepsilon}$ [5] is ε-DP and takes a database D as input and outputs a noised Δ_q-sensitivity bounded query q: $\mathcal{M}_{L,q,\varepsilon}(D) \mapsto q(D) + \mathrm{Lap}(0, \frac{\Delta_q}{\varepsilon})$. The sensitivity Δ_q describes how much the output could change in the worst-case if a single data point is exchanged and $\mathrm{Lap}(\Delta_q/\varepsilon)$ is defined as $\mathrm{Lap}(\Delta_q/\varepsilon)[o] := \frac{\varepsilon}{2\Delta_q}\exp(-|o-\mu|\varepsilon/\Delta_q)$. Thus, a smaller ε or a larger sensitivity Δ_q corresponds to noise with a larger standard deviation.

Theorem 1 (The Laplace Mechanism is DP). *For a Δ_q-bounded ($\Delta_q \in \mathbb{R}_+$) counting query q and an $\varepsilon > 0$, the Laplace Mechanism $\mathcal{M}_{L,q,\varepsilon}$ is ε-DP.*

Report Noisy Max (ReNoM) [6] is an ε-DP algorithm that takes a set of m counting queries and returns the index of the query with the highest noisy count. For that, we add independently sampled noise $\mathrm{Lap}(1/\varepsilon)$ to each query (e.g. the frequency of activity pairs in a directly-follows relation).

Rejection Sampling (RejSamp) as in Alg. 1 is a technique to privately select candidates from a mechanism $M(D)$ that outputs x (e.g. a PST) and a score q (e.g. a noisy fitness). It is ε-DP with $\varepsilon = 2\cdot\varepsilon_1 + \varepsilon_0$ if $M(D)$ is ε_1-DP [12, Algorithm 1]. With a pre-defined threshold t, we accept and return (x, q) if $q \geqslant t$, otherwise we repeat the process at most T rounds until either the output is accepted or a γ biased coin is positive.

V. Approach

Our novel privacy-preserving process miner *DPIM* produces a process tree (PST) based on an event log. A key ingredient of many process mining algorithms is the directly-follows relation, which describes which pair of activities directly follow each other in the traces. DPIM (cf. Algorithm 2) leverages two key insights: it suffices to build a representative PST (cf. Algorithms 3 and 4) to solely operate on (i) a directly-follows

relation where each activity pair is annotated with its frequency in the event log and (ii) the first and last activities of any loop or parallel cut (cf. Algorithm 5). In Section V-C, we prove that DPIM is differentially private.

Algorithm 2 DPIM: Creates a differentially private PST.

1: **Input:** EventLog L, fitness threshold t, probability γ, privacy budgets $\varepsilon, \varepsilon_0$, budget share (r_1, r_2, r_3), lower and upper bound lb, ub

2: $\varepsilon_1 \leftarrow 0.5 \cdot (\varepsilon - \varepsilon_0)$ ▷ Budget build PST & fitness

3: $T \geqslant \max\{\frac{1}{\gamma} \ln \frac{2}{\varepsilon_0}, 1 + \frac{1}{e\gamma}\}$ ▷ #iterations for RejSamp

▷ Initialize DFR dict incl. dummy start & end activities

4: DFR $\leftarrow \{\{(a,b)\colon 0\} \mid (a,b) \in (A \cup \{\text{START}\}) \times (A \cup \{\text{END}\})\}$

▷ Count activity pairs in DFR: in how many traces it occurs

5: **for** dfr **in** DFR **do**

6: **for** trace **in** L **do**

7: **if** dfr $\in$ trace **then** dfr.value $\leftarrow$ dfr.value $+ 1$

▷ RejSamp until the PST meets the fitness threshold

8: **for** $1, \ldots, T$ **do** ▷ ... for at most T rounds

9: n $\leftarrow \text{Uniform}(lb, ub)$

▷ Chose n DFR with the highest noisy count (ReNoM)

10: DP-DFR $\leftarrow \{\ \}$

11: **for** $1, \ldots,$ n **do**

12: dfr $\leftarrow$ DFR[ReNoM(DFR $\setminus$ DP-DFR, $\frac{\varepsilon_1 \cdot r_1}{2 \cdot n}$)]

13: dfr.value $\leftarrow$ dfr.value $+ \text{Laplace}(0, \frac{2 \cdot \text{n}}{\varepsilon_1 \cdot r_1})$

14: DP-DFR $\leftarrow$ DP-DFR $\cup \{\text{dfr}\}$

15: DP-PST $\leftarrow$ buildT(DP-DFR, L, $\sqrt{8} \cdot \frac{2 \cdot \text{n}}{\varepsilon_1 \cdot r_1}$, $\varepsilon_1 \cdot r_2$, DP-S $=\emptyset$, DP-E$=\emptyset$) ▷ Algorithm 3

16: **if** DP-fit $\geqslant t$ **then return** (DP-PST, DP-fit)

17: flip a γ-biased coin s.t. with prob. γ: **return** $\perp$

18: **return** $\perp$

A. High-level Description of DPIM

DPIM (cf. Algorithm 2) generates a process tree (PST) and – for the rejection sampling step – a fitness score in a differentially private manner. We use rejection sampling (cf. Alg. 1) to accept only those PSTs with a high fitness. DPIM works as follows: it annotates in the directly-follows relation DFR in how many traces of the event log L any two activities directly follow each other (Alg. 2, lines 5-7). Note that for privacy reasons, we only use a binary count of whether two activities directly follow. We consider any permutations of activity pairs, including dummy start and end activities (line 4). Table II provides sample DFRs with counts based on Table I. As this DFR contains sensitive data (counts based on the number of traces), we modify this relation in the next steps to make it differentially private. We perform at most T rounds of rejection sampling, which rejects a generated PST (cf. Algorithm 3) that does not have a fitness of at least t. Noise induced by differential privacy increases the variance of the PSTs, which can result in PSTs with low fitness. As the fitness depends on the sensitive event log, we noise the fitness with Laplace noise (cf. Theorem 1). The scale parameter of the Laplace noise that is added to the fitness, i.e. $1/|L| \cdot \varepsilon_1 \cdot r_3$, directly correlates with the expected deviation on how much the fitness threshold is missed. We design DPIM in a way that the PST generation in buildT (cf. Algorithm 3) operates on the annotated directly-follows relation (DP−DFR) that is obtained differentially private. The post-processing theorem of differential privacy [6] states that any operation on this relation does not incur additional privacy leakage. We select the top-n frequency-annotated activity pair in DP−DFR in a differentially private way: in each iteration of rejection sampling, the algorithm first samples the variable n uniformly at random from lower- and upper-bound hyperparameters lb, ub (lines 8-9). Directly choosing n as the number of activity pairs that have a frequency count larger than 0 would have privacy leakage. Due to rejection sampling, the effect of a suboptimal n selection is limited to how much including non-existent or excluding existing activity pairs affects the fitness. Next, we select the index of the directly-follows relation with the largest noisy count using the differentially private Report Noisy Max mechanism (cf. Section IV) and also release the noisy count which we formally noise again. We repeat this *argmax* process n times where we exclude the previously selected indices in each round. As a result, we obtain n-many frequency-annotated activity pairs in DP−DFR that occur probably the most in the event log (lines 11-14). For example, when having $n = 5$, we select (START, R), (R, H), (H, S), (S, D) and (S, S), which have highest noisy count in Table II, even though *(S, S)* is not part of the first three trace variants. These selected DFRs are then used to build the PST (line 15) (cf. Algorithm 3).

DFR	Count	
	Raw	Noisy
(START, R)	100	105.69
(R, H)	100	97.23
(R, END)	0	5.99
...	...	...
(H, S)	25	22.31
(S, S)	0	7.64
(S, D)	25	31.02
(START, S)	0	-2.16

TABLE II: DFR based on trace variants *1* to *3* of Table I

B. Detailed Description of the Subalgorithms

buildT (Algorithm 3) buildT analyses the differentially privately obtained DFR (DP−DFR) and returns a process tree (PST) by determining sequential ($\rightarrow$), exclusive or ($\otimes$), parallel ($\oplus$), and loop ($\circlearrowleft$) cuts in DP−DFR. In this step, we follow the Inductive Miner [20]. buildT works as follows:

Initially, the algorithm determines the start and end activities (DP−S and DP−E), counts the initial event log size (DP−ESize), and counts activity occurrences at the start of a DFR (DP−ActC) (lines 3-8). These operate on DP−DFR and do not incur additional privacy leakage due to the post-processing theorem.

DP-S, DP-E. The start and end counts are necessary for loop detection and to distinguish parallel from loop cuts (lines 3-4).

DP-ESize. We count the event log size to exclude simple loops for an improved parallel cut detection and to determine

Algorithm 3 BuildT: Builds a PST by detecting cuts in a DFR.

1: **Input:** DP-DFR, args={L, std, $\varepsilon_{\text{start\_end}}$, DP-S, DP-E}
2: **if** first recursive round **then**
   ▷ calculate start & end activities and event log size DP
3: DP-S ← $\{b \mid a = \text{START}\}$ $\forall\{(a,b):p\} \in$ DP-DFR
4: DP-E ← $\{a \mid b = \text{END}\}$ $\forall\{(a,b):p\} \in$ DP-DFR
5: DP-ESize ← $\sum_{\{(a,b):p\} \in \text{DP-DFR}} p \cdot 1[a = \text{START}]$
6: DP-ActC ← $\{a\colon 0 \mid a \neq \text{START} \wedge b \neq \text{END}\}$ $\forall\{(a,b):p\} \in$ DP-DFR
7: **for** $\{(a,b):p\}$ **in** DP-DFR **do** ▷ activity count
8: DP-ActC[a] ← DP-ActC[a] + p

9: seqSet ← SEQUENCE(DP-DFR) ▷ cf. [20]
10: **if** len(seqSet) > 1 **then return** APPENDTREE(→, seqSet, DP-DFR, args, DP-ESize, DP-ActC) ▷ seq. cut, Alg. 4

11: xorSet ← XOR(DP-DFR) ▷ cf. [20]
12: **if** len(xorSet) > 1 **then return** APPENDTREE($\bigotimes$, xorSet, DP-DFR, args, DP-ESize, DP-ActC) ▷ xor cut

13: DP-DFR' = DP-DFR ▷ remove loops
14: **for** $\{(a,b):p\}$ and $\{(b,a):p'\}$ **in** DP-DFR **do**
15: **if** $p + p' \geqslant$ DP-ESize + std **then**
16: DP-DFR' ← delete $\{(a,b):p\}, \{(b,a):p'\}$

17: andSet ← AND(DP-DFR', DP-S, DP-E) ▷ cf. [20]
18: **if** len(andSet) > 1 **then return** APPENDTREE($\bigoplus$, andSet, DP-DFR', args, DP-ESize, DP-ActC) ▷ parallel cut

19: loopSet, DP-DFR-nL ← LOOP(DP-DFR, DP-S, DP-E)
20: **if** len(loopSet) > 1 **then return** APPENDTREE($\circlearrowleft$, loopSet, DP-DFR-nL, args, DP-ESize, DP-ActC) ▷ loop cut

21: **return** APPENDTREE(FLOWER, s, DP-DFR, args, DP-ESize, DP-ActC)

Algorithm 4 appendTree: Recursively appends a cut to a PST.

1: **Input:** cutType c, cutSet s, DP-DFR, p={L, std, ε, DP-S, DP-E}, DP-ESize, DP-ActC
2: **if** c = LOOP **or** c = AND **then** $\varepsilon \leftarrow 0.5 \cdot \varepsilon$
3: DP-S, DP-E ← DETECTS_E(c, s, DP-DFR, p) ▷ Alg. 5
4: **if** c = LOOP **then** subtree[$\perp$] ← XOR(τ, $\perp$) ▷ Loops are always optional
5: subtree[$\perp$] ← c[...,$\perp$,...] ▷ Add cut to subtree
6: **for** acts **in** s **do**
7: **if** len(act) = 1 **then** ▷ create leaf node DP
8: **if** {(acts[0], acts[0]): p} ∈ DP-DFR **then**
9: subtree[$\perp$] ← LOOP(acts[0], τ)
10: **else if** DP-ActC[acts[0]] ⩽ DP-ESize - std **then**
11: subtree[$\perp$] ← XOR(τ, acts[0])
12: **else** subtree[$\perp$] ← acts[0]
13: **else if** len(act) = 0 **then**
14: subtree[$\perp$] ← τ
15: **else** ▷ Alg. 3 to cut remaining activities recusively
16: subtree[$\perp$] ← BUILDT(DP-DFR[acts], p)
17: **return** subtree
18: **if** $\sum_{\text{acts} \in s} \sum_{a \in \text{acts}}$ DP-ActC[a] ⩽ DP-ESize - std **then**
19: subtree[$\perp$] ← τ
20: **return** subtree

silent transitions τ. We use the design of DP−DFR and sum up all counts in DP−DFR that contain the dummy START activity (line 5). The original Inductive Miner uses a different technique, which performs privacy-leaking lookups on the sensitive event log. Our technique is differentially private but constitutes an overapproximation, which results in a higher amount of τ in the resulting PST. The reason is that DP−ESize does not change in a subtree, although the active event log size in this subtree does if, e.g., this subtree is branched by an XOR.

DP-ActC. We count how often each activity occurs at the start of a DFR which determines with DP−ESize whether a subtree is optional, i.e. XOR(subtree, τ). We sum over those counts of DP−DFR where each activity is at the beginning of the relation, excluding activity pairs that involve the dummy activities START or END (lines 6-8).

Approximating IM. Next, we detect the cuts using the DP−DFR for the APPENDTREE subroutine (cf. Algorithm 4). For SEQ and XOR cuts, we use the Inductive Miner (IM) [20] as these operate on a DFR and not on the event log (lines 9-12). For AND and LOOP cuts, we adapt the IM by using only the DP−DFR and the knowledge of the start and end activities within a LOOP and AND candidate, DP−S and DP−E. To simplify the parallel cut detection, the DPIM removes simple loops like LOOP(A,B) or LOOP(AND(A,B,C), τ) over activities A, B, and C a prior (lines 13-16). For instance, the directly-follow relation of LOOP(A,B) is similar to that of the parallel cut AND(A,B): (A,B) and (B,A). However, the frequency count differs: for a loop, count(A,B) + count(B,A) > EventSize, whereas for a parallel cut, count(A,B)+count(B,A) = EventSize. As the counts and event log size are noisy, we only remove loops that are considerably above the noisy event log size DP−ESize (factor: $\sqrt{8} \cdot$ LaplaceScale) (line 15-16).

appendTree (Algorithm 4) APPENDTREE performs the cut of Algorithm 3 and either returns a leaf in a PST if the cut separates a single activity (line 20) or a recursively built subtree on the remaining activities in a cut (line 16) (cf. BUILDT of Algorithm 3).

DETECTS_E spends some privacy budget ε to recalculate the start and end activities within a loop or parallel cut for a subsequent recursive round (lines 2-3). A priori the count of parallel or loop cuts is unknown. Therefore, we spend the privacy budget using a geometric series, where we first spend 0.5ε, then 0.25ε, etc (line 2).

The algorithm builds the PST recursively where we append cuts and activities to an empty leaf $\perp$ with the notation "subtree[$\perp$] ←". In particular, "subtree[$\perp$] ← c[...,$\perp$,...]" denotes that we add the cut c with as many empty leaves as needed (line 5), and "subtree[$\perp$] ← c[acts[0]]" marks a leaf

Algorithm 5 DetectS_E: Detects the next start & end activities.

1: **Input:**cutType c, cutSet s, DP-DFR, p={DP-S, DP-E, L, _, ε}
2: **if** c = $\circlearrowleft$ **or** c = $\bigoplus$ **then**
3: **for** trace **in** L **do** ▷ only keep event log of subtree
4: trace ← del. all activities in trace that are not in s
5: cStart ← $\{\{a:0\}\ \forall a \in \text{acts}, \text{acts} \in s\}$
6: cEnd ← $\{\{a:0\}\ \forall a \in \text{acts}, \text{acts} \in s\}$
▷ count for each activity how often it starts and ends
7: **for** trace **in** L **do**
8: cStart[trace.firstAct] ← cStart[trace.firstAct] + 1
9: cEnd[trace.lastAct] ← cEnd[trace.lastAct] + 1
10: **for** { a : cnt } **in** cStart **and** { a : cnt } **in** cEnd **do**
11: cnt ← cnt + $\text{Laplace}(0, \frac{4}{\varepsilon})$ ▷ noise this count
12: DP-S ← $\{\{a:\text{cnt}\} \in \text{cStart} \mid \text{cnt} \geqslant \sqrt{8} \cdot \frac{4}{\varepsilon}\}$ ▷ keep significant count
13: DP-E ← $\{\{a:\text{cnt}\} \in \text{cEnd} \mid \text{cnt} \geqslant \sqrt{8} \cdot \frac{4}{\varepsilon}\}$
14: **return** DP-S, DP-E
15: **else if** c = $\bigotimes$ or c = $\rightarrow$ **then**
16: **for** acts **in** s **do**
17: **if** len(acts) = 1 **then**
18: DP-S ← replace acts[0] with DP-DFR-succ(acts[0])
19: DP-E ← replace acts[0] with DP-DFR-pred(acts[0])
20: **return** DP-S, DP-E

with one activity where no subsequent cuts can be appended (i.e, line 12). We overapproximate loops by making them optional (line 4).

After appending the cut, the algorithm appends for each element in the cut set either a leaf or a subtree (lines 6-17). For instance, for input `cutSet=[[R], [H], [S,D]]` and `cutType=SEQ` on activities R, H, S, and D, we append a sequence cut with two leaves with the R and H and a subtree over S and D. For appending each subtree we call Algorithm 3 recursively on the directly-follows relation `DP-DFR[acts]` relevant for the subtree (line 16). Note that the algorithm also handles a few special cases: first, a self-loop where we loop over one activity or τ (line 9); second, an optional activity represented by XOR(τ, activity) which only happens if this activity occurs considerably (factor: $\sqrt{8} \cdot$ LaplaceScale) less then `DP-ESize` (line 11); and third if all leaves of this cut are optional as each activity within it occurs considerably less than `DP-ESize` (lines 18-19). The algorithm also overapproximates τ-transitions in the second and third special cases, as it considers the overall event log size `DP-ESize` and not the active number of traces within this cut.

detectS_E (Algorithm 5) DETECTS_E determines the start and end activities within the current cut. For a sequence ($\rightarrow$) or exclusive or ($\bigotimes$) cut, we determine these by looking at the predecessor or successor of the start or end activities of the previous cut in `DP-DFR` (lines 15-20). This step is DP by the post-processing theorem [6]. For a parallel ($\bigoplus$) or loop cut ($\circlearrowleft$), we work on the event log where we spend a privacy budget roughly proportional to the number of ANDs and LOOPs (lines 2-14). This is due to the nature of ANDs and LOOPs where a successor or predecessor is not necessarily part of the next cut.

AND or LOOP case. First, we identify the correct start and end activities by deleting all activities in the event log L that are not part of the current cutSet (lines 3-4), e.g. assuming (S, S) and (S, D), the traces would be: $\langle D \rangle$ and $\langle S, D \rangle$, based on trace variants *1* to *3*. Thus, we only have to count in `cStart` and `cEnd` the first and last activity of each modified trace (lines 7-9). Second, we noise these counts using the Laplace Mechanism (lines 10-11) (cf. Section V-C). Each trace has a single start and end activity, so it can only influence one count per start and end. Thus, we apply parallel composition to prevent scaling the noise with the number of candidates. Third, we select those activities from the candidate sets `cStart` and `cEnd` as start and end activities `DP-S` and `DP-E`, which have a count significantly greater than zero (lines 12-13). As a significance level for this particular case, we use the doubled standard deviation $2\sigma = \sqrt{8} \cdot b$ of the added Laplace noise $\text{Laplace}(0, b)$, which corresponds to a $97\,\%$ significance level of sampling below the standard deviation.

C. Privacy Guarantee

Theorem 2. *The DP-Inductive-Miner in Algorithm 2 is ε-differentially private.*

Proof. By the post-processing theorem of differential privacy, it suffices that each operation that works on the sensitive event log is differentially private. These are the start and end count in Algorithm 5 and the DP-DFR creation, fitness calculation, and rejection sampling in Algorithm 2.

In Algorithm 5, we count based on the event log in `cStart` and `cEnd` for each activity how often it is at the start and the beginning of each trace. Each activity in `cStart` and `cEnd` has a sensitivity of 1, i.e. exchanging one trace increases each count by at most 1. By the Laplace Mechanism (Definition 1), adding Laplace noise to each 1-sensitivity-bounded count with a scale of $\frac{4}{\varepsilon_{\text{start\_end}}} = \frac{1}{0.25\varepsilon_{\text{start\_end}}}$ is $0.25\varepsilon_{\text{start\_end}}$-DP. As only two counts in either `cStart` and `cEnd` can increase by an exchanged trace, we can apply the parallel composition theorem [6] which means that noising all counts in either `cStart` and `cEnd` is $0.5\varepsilon_{\text{start\_end}}$-DP. As we noise both `cStart` and `cEnd`, we apply the sequential composition theorem such that this process is $\varepsilon_{\text{start\_end}}$-DP since $\varepsilon_{\text{start\_end}} = 0.5\varepsilon_{\text{start\_end}} + 0.5\varepsilon_{\text{start\_end}}$.

Algorithm 2 utilizes the rejection sampler [12] (cf. Section IV) which is ε-DP with $\varepsilon = 2\varepsilon_1 + \varepsilon_0$. Since we chose the number of activity pairs n uniformly at random, the mechanism is not dependent on the rejection iteration. Thus, it remains to show that the fitness calculation is ε_1-DP with $\varepsilon_1 = r_1\varepsilon_1 + r_2\varepsilon_1 + r_3\varepsilon_1$ for some positive shares r_1, r_2, r_3 s.t. $r_1 + r_2 + r_3 = 1$. We notate $r_2\varepsilon_1 = \varepsilon_{\text{start\_end}}$.

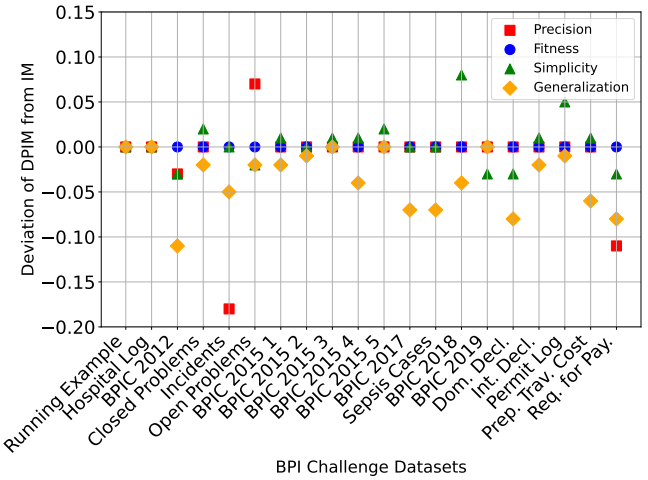


Fig. 3: IM vs. the non-DP DPIM: Deviation of metrics

Selecting the activity pair of DFR with the highest count using the Report Noisy Max is $0.5r_1\varepsilon_1$-DP. As this process is repeated n times, we apply sequential composition and rescale the privacy budget of the Report Noisy Max by $1/n$. Counting the frequency of dfr is $0.5r_1\varepsilon_1$-DP with a similar argumentation as for Algorithm 5: The counting process of each count is 1-sensitivity bounded, but here we apply sequential composition as each trace can alter all activity pairs. Thus, we have to scale the Laplace noise for each time we noise, i.e., by n. Counting the fitness works analogously and is $r_3\varepsilon_1$-DP, but here we have a sensitivity of $1/|L|$. Note that we assume that the event log size L is public knowledge, which does not change if we exchange a trace in the event log. Thus, by the post-processing theorem, the fitness score calculation is ε_1-DP, which concludes that creating a PST in Algorithm 2 is ε-differentially private. □

VI. Evaluation

This section summarizes the evaluation results in terms of process model accuracy and trade-off between accuracy and privacy. We applied DPIM on 14 event logs from the BPI Challenges. Table III shows for each evaluated event log the amount of traces, trace variants, events, and unique activities.

BPI Challenge Event Log	Traces	Variants	Events	Activities
Closed Problems	1,487	327	6,660	7
Domestic Declarations	10,500	99	53,437	17
Incidents	7,554	2,278	65,533	13
International Declarations	6,449	753	72,151	34
Open Problems	819	182	2,531	5
Prepaid Travel Costs	2,099	202	18,246	29
Request for Payment	6,886	89	36,796	19
Sepsis	1,050	846	15,214	16

TABLE III: Evaluation Event Log Statistics

Evaluation Setup We used PM4Py [1] to compute fitness, precision, simplicity, and generalization. As mentioned in Section V, we modify some parts of PM4Pys cut detection to allow a privacy-preserving generation of a PST. As hyperparameters of Algorithm 2, we have chosen $r_1 = 0.65$, $r_2 = 0.25$, $r_3 = 0.1$, $\varepsilon_0 = 0.01$, $\gamma = 0.01$, and $t = 0.95$. The lower and upper bound hyperparameters vary per event log. In our evaluation, we determined both bounds using the number of activity pairs in the directly-follows relation with a frequency count above 0. We subtracted 15 from this frequency count for the lower bound and added 15 for the upper bound. Both values were then rounded to the next by 5 divisible number. Moreover, the lower bound is never smaller than the number of activities in the event log L, and the upper bound is always smaller than the square of the number of activities.

Evaluating Correctness We ran DPIM in a non-DP mode (simulated by $\varepsilon = 100{,}000$) on all event logs and compared the fitness, precision, simplicity, and generalization of the PSTs with the results of the Inductive Miner (IM). We found that DPIM approximates IM closely; nearly all metrics of the DPIM are within $\pm$ 0.10 of the respective IM values (Fig. 3). For instance on the 2013 Incidents event log, Figs. 5a and 5b show that process trees discovered by IM and DPIM are close.

Evaluating Privacy Gain The strength of the DP guarantee is expressed with ε where a lower value means better privacy. By design, a higher ε introduces more noise, implicating how drastically a PST can be altered. Comparing Figs. 5b and 5c shows that the discovered PST changes significantly, hiding the exact nature of the underlying process. Thereby, the influence of single traces on the process model has been obscured, showing that the introduction of DP, as proven in Theorem 2, alters the PST with limited utility loss (cf. Fig. 4).

Evaluating Utility Loss Fig. 4 illustrates the fitness, precision, simplicity, and generalization of the 14 event logs of the IM and DPIM. The IM values are denoted as dots on the Y-axis. The privacy-preservation DPIM results are shown with varying ε values (3.75, 1.25, and 0.125). For **fitness**, most process models achieve a value of 0.95, where only the process model of the *Open Problems* event log falls below 0.9. This robustness of fitness results from the rejection sampling method, as the DPIM is designed to preserve a high fitness to still be representative of the event log. For **precision**, the introduction of differential privacy increases the metric value for 10 out of 14 event logs. As the ε decreases, we see two effects. While some process models slightly decline in precision (*Open Problems*) as the noised process model allows for more behavior not seen in the event log, some also increase (*Closed Problems*). The latter can be explained by noise removing choices from the original process, thus allowing for fewer unseen process behaviors. **Simplicity** initially increases across all event logs compared to the IM. With a further increase in privacy, simplicity also further increases (*Open Problems*, *Prepaid Travel Cost*) while for most process models simplicity remains stable. The increase in simplicity comes from noise removing subtrees, leading to shorter and simpler process trees. **Generalization** declines across all event logs when introducing privacy, and mostly declines further when strengthening privacy.

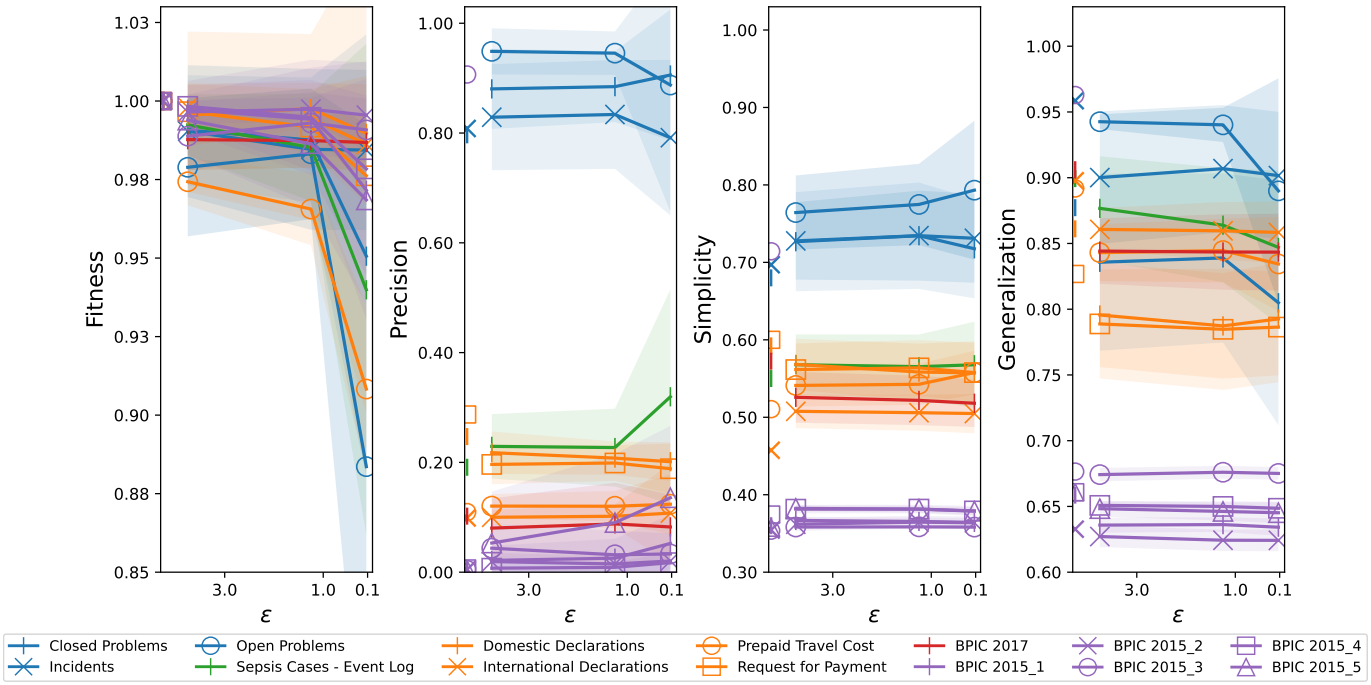
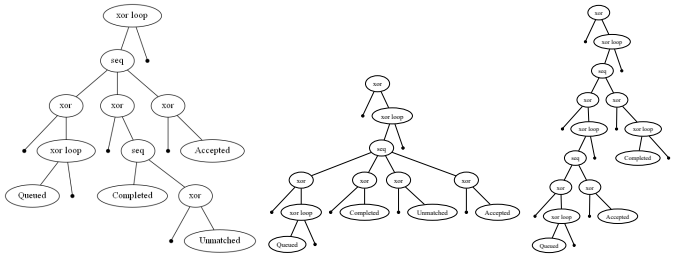


Fig. 4: Fitness, precision, simplicity, and generalization (higher is better) of DPIM for 8 benchmark event logs and 3 privacy parameters (lower ε means stronger privacy). The dots at the Y-axis indicate the respective performance of the IM.



(a) IM (b) ε = 100,000 (c) ε = 0.125

Fig. 5: Comparing IM (a) and DPIM (b+c) on *Incidents* data.

VII. Conclusion

This paper presented the differentially private discovery algorithm based on the Inductive Miner (IM) called *DPIM*. We have proven ε-DP of *DPIM* and evaluated it on 14 real-world event logs. A comparison between the process models discovered by DPIM and the original IM shows that strong privacy can be given with a slight loss of utility. The choice of the budget ε quantifies the privacy guarantee. The loss of utility is quantified via the difference in quality metrics such as fitness, precision, simplicity, and generalization between the results of IM and DPIM.

References

[1] Alessandro Berti, Sebastiaan van Zelst and Daniel Schuster, "PM4Py: A process mining library for Python", Software Impacts, vol. 17, 2023.

[2] A. Burattin, M. Conti, and D. Turato, "Toward an Anonymous Process Mining," in FiCloud, 2015.

[3] Cohen, Aloni and Nissim, Kobbi, "Towards formalizing the GDPR's notion of singling out", PNAS, vol. 117, no. 15, 2020.

[4] C. Dwork, "A firm foundation for private data analysis", Commun. ACM, Bd. 54, Nr. 1, S. 86–95, Jan. 2011

[5] C. Dwork, F. McSherry, K. Nissim, and A. Smith "Calibrating noise to sensitivity in private data analysis," in Theory of Cryptography, 2006.

[6] C. Dwork and A. Roth, "The Algorithmic Foundations of Differential Privacy," Foundations and Trends® in TCS, vol. 9, no. 3–4, 2014.

[7] G. Elkoumy et al., "Privacy and Confidentiality in Process Mining: Threats and Research Challenges," ACM TMIS, vol. 13, 2022.

[8] G. Elkoumy, A. Pankova, and M. Dumas, "Privacy-Preserving Directly-Follows Graphs: Balancing Risk and Utility in Process Mining." arXiv, 2020.

[9] G. Elkoumy, A. Pankova, and M. Dumas, "Differentially Private Release of Event Logs for Process Mining." arXiv, 2022.

[10] S. A. Fahrenkrog-Petersen, M. Kabierski, H. van der Aa, and M. Weidlich, "Semantics-aware mechanisms for control-flow anonymization in process mining", Inf. Syst., Bd. 114, S. 102169, 2023.

[11] S. A. Fahrenkrog-Petersen, H. van der Aa, and M. Weidlich, "PRIPEL: Privacy-Preserving Event Log Publishing Including Contextual Information," in BPM 2020.

[12] Jingcheng Liu and Kunal Talwar, "Private Selection from Private Candidates", ACM SIGACT Symposium on Theory of Computing, 2018.

[13] F. Mannhardt, A. Koschmider, N. Baracaldo, M. Weidlich, and J. Michael, "Privacy-Preserving Process Mining," BISE, vol. 61, no. 5., pp. 595–614, 2019.

[14] A. Narayanan and V. Shmatikov, "Robust De-anonymization of Large Sparse Datasets", in IEEE S&P, 2008.

[15] S. Nuñez von Voigt et al., "Quantifying the Re-identification Risk of Event Logs for Process Mining," in CAiSE, 2020.

[16] M. Rafiei, G. Elkoumy, and W. M. P. van der Aalst, "Quantifying Temporal Privacy Leakage in Continuous Event Data Publishing," in Coop. Inf. Syst., 2022.

[17] M. Rafiei and W. M. P. van der Aalst, "Towards Quantifying Privacy in Process Mining," in Process Mining Workshops, 2021.

[18] M. Rafiei and W. M. P. van der Aalst, "Group-based privacy preservation techniques for process mining," Data Knowl. Eng., vol. 134, 2021.

[19] M. Rafiei, M. Wagner, and W. M. P. van der Aalst, "TLKC-Privacy Model for Process Mining," in RCIS, vol. 385. 2020.

[20] W. M. P. van der A. S.J.J. Leemans D. Fahland, "Discovering Block-Structured Process Models from Event Logs - A Constructive Approach," 2013.