Preserving Data Secrecy in Decentralized Inter-organizational Process Mining

Valerio Goretti[a], Davide Basile[a], Luca Barbaro[a], Claudio Di Ciccio[b]

[a]*Sapienza University of Rome, Via Regina Elena 295, 00161, Rome, Italy*
[b]*Utrecht University, Princetonplein 5, 3584 CC, Utrecht, The Netherlands*

Abstract

Inter-organizational business processes involve multiple independent organizations collaborating to achieve mutual interests. Process mining techniques have the potential to allow these organizations to enhance operational efficiency, improve performance, and deepen the understanding of their business based on the recorded process event data. However, inter-organizational process mining faces substantial challenges, including topical secrecy concerns: The involved organizations may not be willing to expose their own data to run mining algorithms jointly with their counterparts or third parties. In this paper, we introduce CONFINE, a novel approach that unlocks process mining on multiple actors' process event data while safeguarding the secrecy and integrity of the original records in an inter-organizational business setting. To ensure that the phases of the presented interaction protocol are secure and that the processed information is hidden from involved and external actors alike, our approach resorts to a decentralized architecture comprised of trusted applications running in Trusted Execution Environments (TEEs). We show the feasibility of our solution by showcasing its application to a healthcare scenario and evaluating our implementation in terms of memory usage and scalability on real-world event logs.

Keywords: Collaborative information

systems architectures, Trusted execution environments, Confidential computing

1. Introduction

In today's business landscape, organizations constantly seek ways to enhance operational efficiency, increase performance, and gain valuable insights to improve their processes. Process mining offers techniques to discover, monitor, and improve business processes by extracting knowledge from chronological records known as *event logs* [1]. Process-aware information systems record events referring to activities and interactions within a business process. The vast majority of process mining contributions consider *intra-organizational* settings, in which business processes are executed inside individual organizations. However, organizations increasingly recognize the value of collaboration and synergy in achieving operational excellence. *Inter-organizational* business processes involve several independent organizations cooperating to achieve a shared objective [2]. Despite the advantages of transparency, performance optimization, and benchmarking that companies can gain, inter-organizational process mining raises challenges that hinder its application. The major issue concerns confidentiality. Companies are reluctant to share private information required to execute process mining algorithms with their partners [3]. Indeed, letting sensitive operational data traverse organizational boundaries introduces concerns about data privacy, security, and compliance with internal regulations [4]. *Trusted Execution Environments* (TEEs) [5] can serve as fundamental enablers to balance the need for insights with the need to protect sensitive information in inter-organizational settings. TEEs offer secure contexts that guarantee code integrity and data confidentiality before, during, and after its utilization.

In this paper, we propose CONFINE, a novel approach and tool aimed at enhancing collaborative information system architectures with secrecy-preserving process mining capabilities in a decentralized fashion. It resorts to *trusted applications* running in TEEs to preserve the secrecy and integrity of shared data. To pursue this aim, we design a decentralized architecture for a four-staged protocol: *(i)* The initial exchange of preliminary metadata, *(ii)* the attestation of the miner entity, *(iii)* the secure transmission and privacy-preserving merge of encrypted information segments amid multiple parties, *(iv)* the isolated and verifiable computation of process discovery algorithms on joined data. We evaluate our proof-of-concept implementation against synthetic and real-world-based data with a convergence test followed by experiments to assess the scalability of our approach.

The remainder of this paper is as follows. Sect. 2.1 provides an overview of related work. In Sect. 3, we introduce a motivating use-case scenario in healthcare. We present the CONFINE approach in Sect. 5. We describe the implementation of our approach in Sect. 7. In Sect. 8, we report on the efficacy and efficiency tests for our solution. Finally, we conclude our work and outline future research directions in Sect. 9.

2. Background and Related Work

2.1. Background

2.1.1. Inter-organizational Process Mining

2.1.2. Trusted Execution Environments

A Trusted Execution Environment (TEE) is an tamper-proof processing environment that operates on a separation kernel [**?**]. By integrating both software and hardware techniques, it segregates the execution of code from the operating system. The separation kernel method guarantees distinct execution between two environments. TEEs were initially proposed by Rushby **?**], enable multiple systems with different security requirements to coexist on a single platform. Owing to kernel separation, the system is divided into numerous segments, ensuring robust isolation between them. TEEs ensure the authenticity of the executed code, the integrity of the runtime states, and the privacy of the code and data preserved in persistent memory. The content produced by the TEE is dynamic, with data securely updated and stored. Consequently, TEEs are fortified against both software and hardware attacks, precluding the exploitation of even backdoor security vulnerabilities [5]. Numerous TEE providers exist, differing in terms of the software system and, more specifically, the processor on which they operate. In this study, we utilize the Intel Software Guard Extensions (Intel SGX)[1]. Intel SGX comprises a set of CPU-level instructions that enable applications to establish enclaves. An enclave is a secure section of the application that ensures the confidentiality and integrity of the data and code within it. These guarantees are also effective against malware with administrative privileges [**?**]. The presence of one or more enclaves within an application can minimize the application's potential attack surfaces. An enclave is unaffected to external read or write operations. Only the enclave itself can modify its secrets, regardless of Central Processing Unit (CPU) privileges employed. Indeed, enclave access is not feasible by manipulating registers or the stack. Each call to the enclave necessitates a new instruction that conducts checks to safeguard the data that are exclusively accessible through the enclave code. In addition to being difficult to access, the data within the enclave is encrypted. Accessing the Dynamic Random Access Memory (DRAM) modules would yield encrypted data [**?**]. The cryptographic key undergoes alterations each time the system is restarted following a shutdown or hibernation [17].

[1] https://www.intel.co.uk/content/www/uk/en/architecture-and-technology/software-guard-extensions.html. Accessed: 24/01/2024.

remote attestation in sgx, epid dicap. ARM trustzone, AMD sev, Multizon security trusted execution environment

2.2. *Related Work*

Despite the relative recency of this research branch across process mining and collaborative information systems, scientific literature already includes noticeable contributions to inter-organizational process mining. The work of Müller et al. [6] focuses on data privacy and security within third-party systems that mine data generated from external providers on demand. To safeguard the integrity of data earmarked for mining purposes, their research introduces a conceptual architecture that entails the execution of process mining algorithms within a cloud service environment, fortified with Trusted Execution Environments. Drawing inspiration from this foundational contribution, our research work seeks to design a decentralized approach characterized by organizational autonomy in the execution of process mining algorithms, devoid of synchronization mechanisms involvement taking place between the involved parties. A notable departure from the framework of Müller et al. lies in the fact that here each participating organization retains the discretion to choose when and how mining operations are conducted. Moreover, we bypass the idea of fixed roles, engineering a peer-to-peer scenario in which organizations can simultaneously be data provisioners or miners. Elkoumy et al. [7, 8] present Shareprom. Like our work, their solution offers a means for independent entities to execute process mining algorithms in inter-organizational settings while safeguarding their proprietary input data from exposure to external parties operating within the same context. Shareprom's functionality, though, is confined to the execution of operations involving event log abstractions [9] represented as directed acyclic graphs, which the parties employ as intermediate pre-elaboration to be fed into secure multiparty computation (SMPC) [10]. As the authors remark, relying on this specific graph representation imposes constraints that may prove limiting in various process mining scenarios. In contrast, our approach allows for the secure, ciphered transmission of event logs to process mining nodes as a whole. Moreover, SMPC-based solutions require computationally intensive operations and synchronous cooperation among multiple parties, which make these protocols challenging to manage as the number of participants scales up [11]. In our research work, individual computing nodes run the calculations, thus not requiring synchronization with other machines once the input data is loaded.

We are confronted with the imperative task of integrating event logs originating from different data sources and reconstructing consistent traces that describe collaborative process executions. Consequently, we engage in an examination of methodologies delineated within the literature, each of which offers insights into the merging of event logs within inter-organizational settings. The work of Claes et al. [12] holds particular significance for our research efforts. Their seminal study introduces a two-step mechanism operating at the structured data level, contingent upon the configuration and subsequent application of merging rules. Each such rule indicates the relations between attributes of the traces and/or the activities that must hold across distinct traces to be combined. In accordance with their principles, our research incorporates a structured data-level merge based on case references and timestamps as merging attributes. The research by Hernandez et al. [13] posits a methodology functioning at the raw data level. Their approach represents traces and activities as *bag-of-words* vectors, subject to cosine similarity measurements to discern links and relationships between the traces earmarked for combination. An appealing aspect of this approach lies in its capacity to generalize the challenge of merging without necessitating a-priori knowledge of the underlying semantics inherent to the logs under consideration. However, it entails computational overhead in

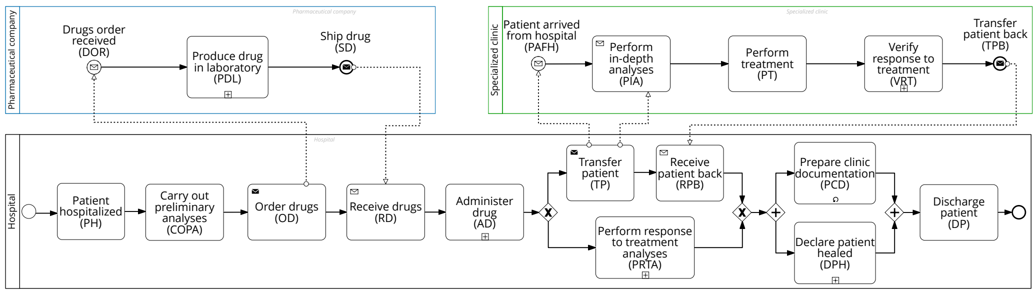


Figure 1: A BPMN collaboration diagram of a simplified healthcare scenario

Table 1: Events from cases 312 (Alice) and 711 (Bob) recorded by the hospital, the specialized clinic, and the pharmaceutical company

Hospital					
Case	Timestamp	Activity	Case	Timestamp	Activity
312	2022-07-14T10:36	PH	312	2022-07-15T22:06	TP
312	2022-07-14T16:36	COPA	711	2022-07-16T00:55	PRTA
711	2022-07-14T17:21	PH	711	2022-07-16T00:55	PCD
312	2022-07-14T17:36	OD	711	2022-07-16T02:55	DPH
711	2022-07-14T23:21	COPA	711	2022-07-16T04:55	DP
711	2022-07-15T00:21	OD	312	2022-07-16T07:06	RPB
711	2022-07-15T18:55	RD	312	2022-07-16T09:06	DPH
312	2022-07-15T19:06	RD	312	2022-07-16T09:06	PCD
711	2022-07-15T20:55	AD	312	2022-07-16T11:06	DP
312	2022-07-15T21:06	AD	312	2022-07-16T11:06	DP

Pharmaceutical company		
HospitalCaseID	Timestamp	Activity
312	2022-07-15T09:06	DOR
711	2022-07-15T09:30	DOR
312	2022-07-15T11:06	PDL
711	2022-07-15T11:30	PDL
312	2022-07-15T13:06	SD
711	2022-07-15T13:30	SD

Specialized clinic		
TreatmentID	Timestamp	Activity
312	2022-07-16T00:06	PAFH
312	2022-07-16T01:06	PIA
312	2022-07-16T03:06	PT
312	2022-07-16T04:06	VRT
312	2022-07-16T05:06	TPB

the treatment of data that can interfere with the overall effectiveness of our approach.

$$T_{312} = \langle \text{PH, COPA, OD, DOR, PDL, SD, RD, AD, TP, PAFH, PIA, PT, VRT, TPB, RPB, DPH, PCD, DP} \rangle$$
$$T_{711} = \langle \text{PH, COPA, OD, DOR, PDL, SD, RD, AD, TP, DPH, PCD, DP} \rangle$$

3. Motivating Scenario

For our motivating scenario, we focus on a simplified hospitalization process for the treatment of rare diseases. The process model is depicted as a BPMN diagram in Fig. 1 and involves the cooperation of three parties: a hospital, a pharmaceutical company, and a specialized clinic. For the sake of simplicity, we describe the process through two cases, recorded by the information systems as in Table 3. Each patient in the hospital is associated with an id which would be the identifier of the case in the hospital log. Alice's journey begins when she enters the hospital for the preliminary examinations (patient hospitalized, PH). The hospital then places an order for the drugs (OD) to the pharmaceutical company for treating Alice's specific condition. Afterwards, the pharmaceutical company acknowledges that the drugs order is received (DOR), proceeds to produce the drugs in the laboratory (PDL), and ships the drugs (SD) back to the hospital. Upon receiving the medications, the hospital administers the drug (AD), and conducts an assessment to determine if Alice can be treated internally. If specialized care is required, the hospital transfers the patient (TP) to the specialized clinic. When the patient arrives from the hospital (PAFH), the specialized clinic performs in-depth analyses (PIA) and proceeds with the treatment (PT). Once the specialized clinic had completed the evaluations and verified the response to the treatment (VRT), it transfers the patient back (TPB). The hospital receives the patient back (RPB) and prepares the clinical documentation (PCD). If Alice has successfully recovered, the hospital declares the patient as healed (DPH). When Alice's treatment is complete, the hospital discharges the patient (DP). Bob enters the hospital a few hours later than Alice. His hospitalization process is similar to Alice's. However, he does not need specialized care, and his case is only treated by the hospital. Therefore, the hospital performs the response to treatment analyses (PRTA) instead of transferring him to the specialized clinic.

Both the National Institute of Statistics of the country in which the three organizations reside and the University that hosts the hospital wish to uncover information on this inter-organizational process for reporting and auditing purposes [14] via process analytics. The involved organizations share the urge for such an analysis and wish to be able to repeat the mining task also in-house. The hospital, the specialized clinic, and the pharmaceutical company have a partial view of the overall unfolding of the inter-organizational process as they record the events stemming from the parts of their pertinence.

In Table 1, we show Alice and Bob's cases (identified by the **312** and **711** codes respectively) recorded by the by the hospital (i.e., T^H_{312} and T^H_{711}), the specialized clinic (i.e., T^S_{312} and T^S_{711}), and the pharmaceutical company (i.e., T^C_{312} and T^C_{711}). The hospital stores the identifier of these cases in the *case* id attribute of its event log. Differently, the specialized clinic and the pharmaceutical company employees a different case denomination and stores the cross-organizational identifiers in other attributes (*TreatmentID* and *HospitalCaseID* respectively). The partial traces of the three organizations are projections of the two combined ones for the whole inter-organizational process: $T_{312} = \langle$PH, COPA, OD, DOR, PDL, SD, RD, AD, TP, PAFH, PIA, PT, VRT, TPB, RPB, DPH, PCD, DP$\rangle$ and $T_{711} = \langle$PH, COPA, OD, DOR, PDL, SD, RD, AD, TP, DPH, PCD, DP$\rangle$. Results stemming from the analysis of the local cases would not provide a full picture. Data should be merged. However, to preserve the privacy of the people involved and safeguard the confidentiality of the information, the involved parties cannot give open access to their traces to other organizations. The diverging interests (being able to conduct process mining on data from multiple sources without giving away the local event logs in-clear) motivate our research. In the following, we describe the design of our solution.

4. Preliminaries

Given a finite set of events $\hat{\mathrm{E}}$ and a total-order relation $\preceq$ subset of $\hat{\mathrm{E}} \times \hat{\mathrm{E}}$, we identify an event log as the totally ordered set $\left(\hat{\mathrm{E}}, \preceq\right)$. In the example, . . . Let $\widehat{\mathrm{IID}}$ be a finite non-empty set of symbols such that $|\widehat{\mathrm{IID}}| \leqslant |\hat{\mathrm{E}}|$. We assume that every event be associated with a *case identifier* $iid \in \widehat{\mathrm{IID}}$ via a total surjective function $\mathfrak{iid} : \hat{\mathrm{E}} \to \widehat{\mathrm{IID}}$ such that the restriction $\prec_{iid} = \preceq \cap \{e \in \hat{\mathrm{E}} : \mathfrak{iid}(e) = iid\}^2$ of total order $\preceq$ on all events

Add example

mapped to the same *iid* is strict (i.e., if $e \preceq e'$ with $e \neq e'$ and $\mathfrak{iid}(e) = \mathfrak{iid}(e')$ then $e' \npreceq e$). In the example, ... In other words, $\mathfrak{iid}$ acts as an equivalence relation partitioning $\widehat{\mathrm{E}}$ into $\left\{\widehat{\mathrm{E}}_{iid}\right\}_{iid \in \widehat{\mathrm{IID}}} \subseteq 2^{\widehat{\mathrm{E}}}$ based on the *iid* to which the events $e \in \widehat{\mathrm{E}}_{iid}$ map, and imposing that events are linearly ordered by the restriction of $\preceq$ on every $\widehat{\mathrm{E}}_{iid}$. Every pair $\left(\widehat{\mathrm{E}}_{iid}, \prec_{iid}\right)$ thus represents a finite linearly totally ordered set (or *loset* for brevity) with $\widehat{\mathrm{E}}_{iid} \subseteq \widehat{\mathrm{E}}$ and $\prec_{iid} \subseteq \widehat{\mathrm{E}}_{iid} \times \widehat{\mathrm{E}}_{iid} \subseteq \preceq \subseteq \widehat{\mathrm{E}} \times \widehat{\mathrm{E}}$. Let $\left(\widehat{\mathrm{E}}, \prec\right)$ be a loset and $(\widehat{\mathrm{E}}', \prec')$, $(\widehat{\mathrm{E}}'', \prec'')$ two (sub-)losets such that $\widehat{\mathrm{E}}' \cup \widehat{\mathrm{E}}'' \subseteq \widehat{\mathrm{E}}$ and $\widehat{\mathrm{E}}' \cap \widehat{\mathrm{E}}'' = \varnothing$, with $\prec'$ and $\prec''$ being the restrictions of $\prec$ on $\widehat{\mathrm{E}}'$ and $\widehat{\mathrm{E}}''$, respectively. We define the order-preserving union $\bigoplus : \widehat{\mathrm{E}}^3 \times \widehat{\mathrm{E}}^3 \to \widehat{\mathrm{E}}^3$ of losets as follows: $(\widehat{\mathrm{E}}', \prec') \bigoplus (\widehat{\mathrm{E}}'', \prec'') = \left(\widehat{\mathrm{E}}' \cup \widehat{\mathrm{E}}'', \prec \cap (\widehat{\mathrm{E}}' \cup \widehat{\mathrm{E}}'')^2\right)$. We can thus derive the notion of case C_{iid} given a $iid \in \widehat{\mathrm{IID}}$ as a loset of events mapping to the same *iid* and ordered by the linear restriction $\prec$ of $\preceq$ over the events in C_{iid}: $iid = (\widehat{\mathrm{E}}_{iid}, \prec)$ where $C_{iid} = \langle e_1, \ldots, e_{|C_{iid}|} \rangle$ where $\mathfrak{iid}(e_i) = iid \in \widehat{\mathrm{IID}}$ for every i s.t. $1 \leqslant i \leqslant |C_{iid}|$ and $e_i \prec e_j$ for every $i \leqslant j \leqslant |C_{iid}|$.[2] Notice that the cardinality of $\widehat{\mathrm{C}}$ and $\widehat{\mathrm{IID}}$ coincide. Events are also the domain of a function $\mathfrak{p} : \widehat{\mathrm{E}} \to \widehat{\mathcal{P}}$ mapping events to log provisioners. In the example, ... We shall denote with $C_{iid}^{\mathcal{P}}$ the loset consisting of every event $e \in C_{iid}$ such that $\mathfrak{p}(e) = \mathcal{P}$, with the restriction of the strict total order of C_{iid} on those events. In the example, ...

Add example

Continue here revising the definition of order preserving union

Add example

Add example

Gotta move this one earlier when we introduce the example with the partitioned event log. (Section 4.2??). Clarify the difference between segmentation (given a segsize, i.e., a segment of a case-part in a sublog) and partitioning (of a log into case-parts of sublogs. Then, prove that the pipeline of partitioning and segmentation has its inverse in the union and merge for soundness.

[2] We employ the angular-bracket notation here for the sake of simplicity, although it is typically used for sequences. Unlike sequences, cases do not allow for the same event to occur more than once.

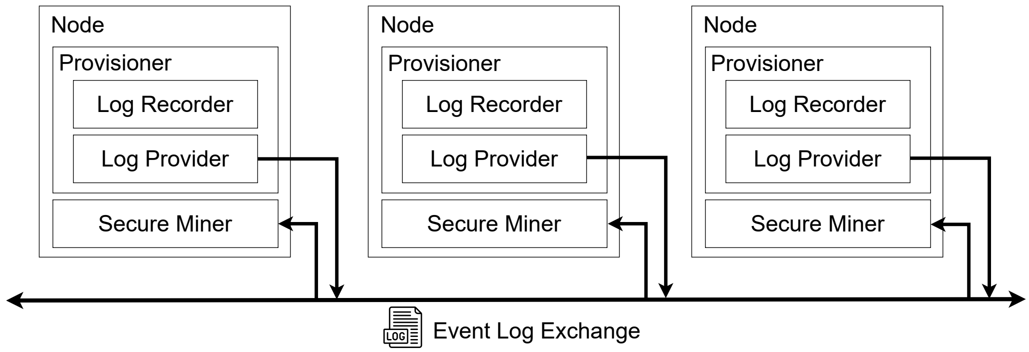


Figure 2: The CONFINE high-level architecture

5. Design

In this section, we present the high-level architecture of the CONFINE framework. We consider the main functionalities of each component, avoiding details on the employed technologies discussed in the next sections.

The CONFINE architecture at large. Our architecture involves different information systems running on multiple machines. An organization can take at least one of the following roles: **provisioning** if it delivers local event logs to be collaboratively mined; **mining** if it applies process mining algorithms using event logs retrieved from provisioners. In Fig. 2, we propose the high-level schematization of the CONFINE framework. In our solution, every organization hosts one or more nodes hosting components (the names of which will henceforth be formatted with a `teletype` font). Depending on the played role, nodes come endowed with a `Provisioner` or a `Secure Miner` component, or both. The `Provisioner` component consists of the following two main sub-components. The `Log Recorder` registers the events taking place in the organizations' systems. The `Log Provider` delivers on-demand data to mining players. The hospital and all other parties in our example record Alice and Bob's cases using the `Log Recorder`. The `Log Recorder` is queried by the `Log`

Provider for event logs to be made available for mining. The latter controls access to local event logs by authenticating data requests by miners and rejecting those that come from unauthorized parties. In our motivating scenario, the specialized clinic, the pharmaceutical company, and the hospital leverage Log Providers to authenticate the miner party before sending their logs. The Secure Miner component shelters external event logs inside a protected environment to preserve data confidentiality and integrity. Notice that Log Providers accept requests issued solely by Secure Miners. Next, we provide an in-depth focus on the latter.

The Secure Miner. The primary objective of the Secure Miner is to allow miners to securely execute process mining algorithms using event logs retrieved from provisioners such as the specialized clinic, pharmaceutical company, and the hospital in our example. Secure Miners are isolated components that guarantee data inalterability and confidentiality. In Fig. 3, we show a schematization of the Secure Miner, which consists of four sub-components: *(i)* Log Requester; *(ii)* Log Receiver; *(iii)* Log Manager; *(iv)* Log Elaborator. The Log Requester and the Log Receiver are the sub-components that we employ during the event log retrieval. Log Requesters send authenticable data requests to the Log Providers. The Log Receiver collects event logs sent by Log Providers and entrusts them to the Log Manager, securing them from accesses that are external to the Secure Miner. Miners of our motivating scenario, such as the university and the national institute of statistics, employ these three components to retrieve and store Alice and Bob's data. The Log Elaborator merges the event data locked in the Secure Miner to have a global view of the inter-organizational

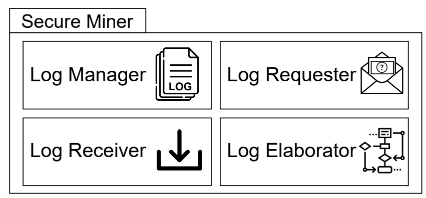


Figure 3: Sub-components of the Secure Miner

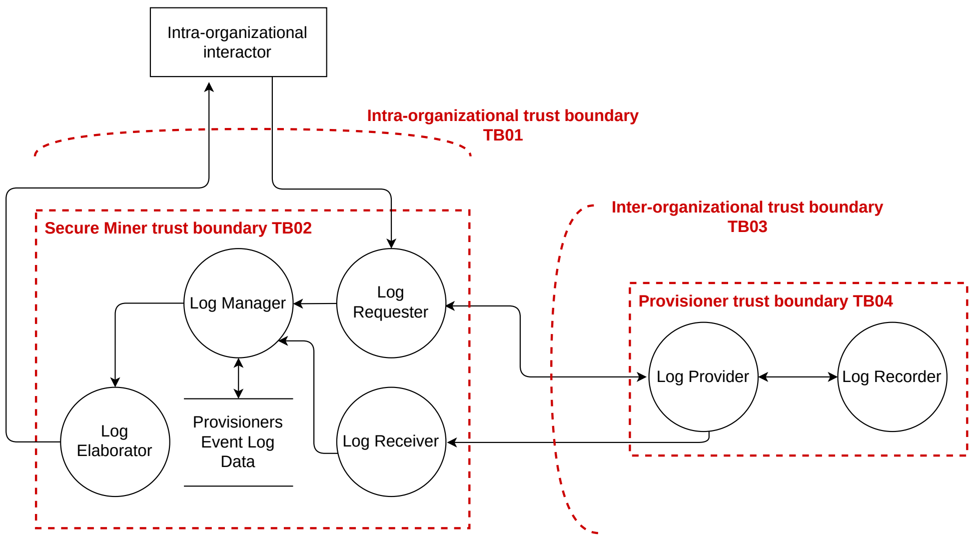


Figure 4: Data flow diagram employed for our threat model

process comprehensive of activities executed by each involved party. Thereupon, it executes process mining algorithms in a protected environment, inaccessible from the outside computation environment. In our motivating scenario, the `Log Elaborator` combines the traces associated with the cases of Alice (i.e., T^H_{312}, T^S_{312}, and T^C_{312}) and Bob (i.e, T^H_{711}, T^S_{711}, and T^C_{711}), generates the chronologically sorted traces T_{312} and T_{711}, and feeds them into the mining algorithms (see the bottom-right quadrant of Sect. 2.2).

6. Threat Model

In the following section we identify the threats that can jeopardize the confidentiality of provisioners' event logs in CONFINE. Our threat analysis is based upon the theoretical foundation of the *STRIDE* framework [**?**]. This model groups the threats in six cathegories: spoofing (i.e, the impersonation a legitimate entity), tampering (i.e., the modification of data to alter its integrity), repudiation (i.e., the

Table 2: Vulnerabilities in the CONFINE architecture

ID	Trust boundary	Type	Threat description
T01	TB01	Spoofing	The attacker impersonates a legitimate interactor to use the `Secure Miner`
T02		Tampering	The intra-organizational interactor sends malicious input to the `Secure Miner`
T03		Information disclosure	The attacker sniffs the messages between the interactor and the `Secure Miner`
T04	TB02	Information disclosure	The attacker accesses the `Secure Miner`'s memory location to leak the event logs
T05		Tampering	The attacker meddles the source code of the `Secure Miner` or its event log data
T06		Elevation of privileges	The attacker gains the rights to run in the same environment of the `Secure Miner`
T07		Denial of service	The `Secure Miner` crashes, halts or stops
T08	TB03	Spoofing	The attacker impersonates a `Secure Miner` to gain access to the `Provisioner`'s log
T09		Spoofing	The attacker impersonates a `Provisioner` to communicate with the `Secure Miner`
T010		Denial of service	The `Secure Miner` floods the `Provisioner` with log requests
T010		Denial of service	The `Secure Miner` floods the `Provisioner` with log requests
T011		Information disclosure	The attacker sniffs the `Provisioner`'s log sent to the `Secure Miner`
T012		Tampering	The attacker alters the data flow between the `Provisioner` and the `Secure Miner`

denial of performing a particular action), information disclosure (i.e., the exposure of sensistive data), denial of service (i.e., the disruption or degradation of availability) and elevation of priviledges (i.e., the misappropriation of higher level of rights).

Introduce Fig. 4 adoption;
Describe each boundary box with their adversary type (Provisioner –> honest, Secure Miner and input sources–> semi-honest)
Explain TB02 AND TB01;
For each TBi: describe all the STRICE threats that are in our scope

7. Realization

In this section, we outline the technical aspects concerning the realization of our solution. Therefore, we first present the enabler technologies through which we instantiate the design principles presented in Sect. 5. After that, we discuss the CONFINE interaction protocol. Finally, we show the implementation details.

7.1. Deployment

Figure 5 depicts a UML deployment diagram [15] to illustrate the employed technologies and computation environments. We recall that the `Miner` and `Provisioner` nodes are drawn as separated, although organizations can host both. In our motivating

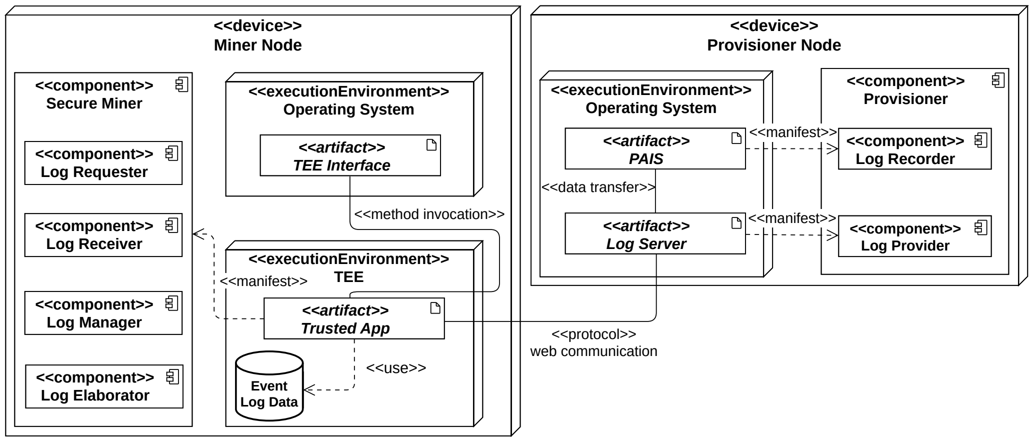


Figure 5: UML deployment diagram of the CONFINE architecture

scenario, e.g., the hospital can be equipped with machines aimed for both mining and provisioning.

`Provisioner Nodes` host the `Provisioner`'s components, encompassing the `Log Recorder` and the `Log Provider`. The Process-Aware Information System (`PAIS`) manifests the `Log Recorder` [16]. The `PAIS` grants access to the `Log Server`, enabling it to retrieve event log data. The `Log Server`, on the other hand, embodies the functionalities of the `Log Provider`, implementing web services aimed at handling remote data requests and providing event log data to miners.

The `Miner Node` is characterized by two distinct *execution environments*: the `Operating System` (`OS`) and the Trusted Execution Environment (`TEE`) [5]. `TEE`s establish isolated contexts separate from the `OS`, safeguarding code and data through hardware-based encryption mechanisms. This technology relies on dedicated sections of a CPU capable of handling encrypted data within a reserved section of the main memory [17]. By enforcing memory access restrictions, TEEs aim to prevent one application from reading or altering the memory space of another, thus enhancing the overall security of the system. This dedicated areas in memory are, however, limited. Once the limits are exceeded, TEEs have to scout around in outer memory areas, thus conceding the opportunity to malicious reader to understand the saved data based on the memory reads and writes. To avoid this risk, TEE implementations often raise errors that halt the program execution when the memory demand goes beyond the available space. Therefore, the design of secure systems that resort to TEEs must take into account that memory consumption must be kept under control. We leverage the security guarantees provided by `TEEs` [18] to protect a `Trusted App` responsible for fulfilling the functions of the `Secure Miner` and its associated sub-components. The `TEE` ensures the integrity of the `Trusted App` code, protecting it against potential malicious manipulations and unauthorized access by programs running within the `Operating System`. Additionally, we utilize the isolated environment of `TEEs` to securely store event log data (e.g., Alice and Bob's cases). The `TEE` retains a private key in the externally inaccessible section of memory, paired with a public key in a Rivest-Shamir-Adleman (RSA) [19] scheme for attestation (only the owner of the private key can sign messages that are verifiable via the public key) and secure message encryption (only the owner of the private key can decode messages that are encrypted with the corresponding public key). The private key associated with the `TEE`'s hardware remains inaccessible, even to users possessing administrative privileges on the `Miner Node`. In our solution, access to data located in the `TEE` is restricted solely to the `Trusted App`. Users interact with the `Trusted App` through the `Trusted App Interface`, which serves as the exclusive communication channel. The `Trusted App` offers secure methods, invoked by the `Trusted App Interface`, for safely receiving information from the `Operating System` and outsourcing the results of computations.

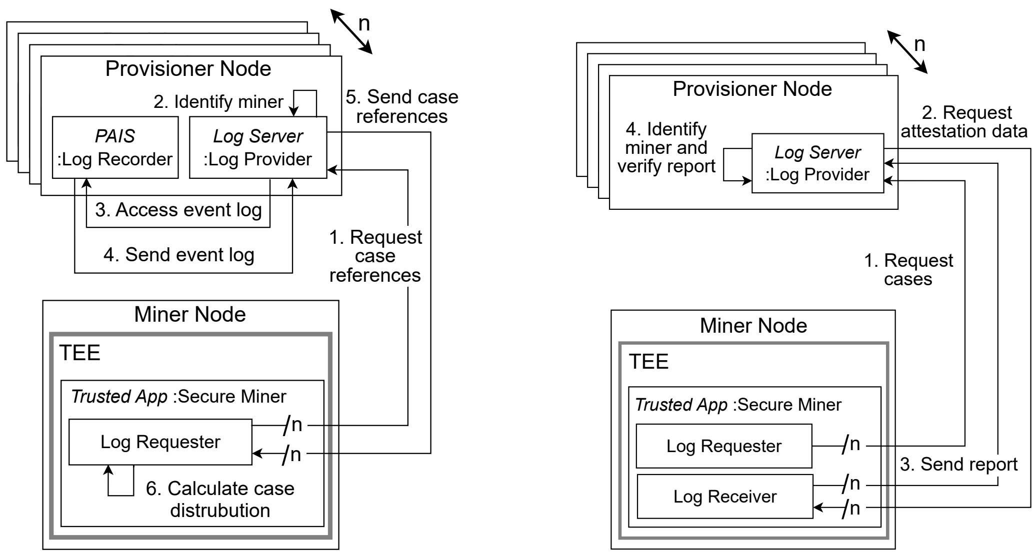
7.2. The CONFINE protocol

We orchestrate the interaction of the components in CONFINE via a protocol. We separate it in four subsequent stages, namely *(i) initialization*, *(ii) remote attestation*, *(iii) data transmission*, and *(iv) computation*. These stages are depicted in Figs. 6(a), 6(b), 7(a) and 7(b), respectively. Our protocol involves two primary entities: a `Secure Miner` (hereafter referred to as $\mathcal{M}$) and one or more `Provisioners` ($\mathcal{P}_1,\ldots,\mathcal{P}_n \in \widehat{\mathcal{P}}$).

The behavioral descriptions for $\mathcal{M}$ and any $\mathcal{P}_i \in \widehat{\mathcal{P}}$ are outlined in Alg. 1 and Alg. 2, respectively. These specifications adhere to the syntax for distributed algorithms detailed in [20].[3] We assume that communication between `Secure Miners` and `Log Provisioners` occurs through an *Authenticated Point-to-Point Perfect Link* [20]. This communication abstraction guarantees: *(i) reliable delivery* (i.e., if a correct process sends a message m to a correct process q, then q eventually delivers m), *(ii) no duplication* (i.e., no message is delivered by a correct process more than once), and *(iii) authenticity* (i.e., if some correct process q delivers a message m with sender p and process p is correct, then m was previously sent to q by p). We posit the assumption that communications transmitted throughout protocol execution are safeguarded by end-to-end encryption. Therefore, the content of the messages is discernible solely to the designated sender and receiver.

In Alg. 1, the `Secure Miner` is provided with the following input: the list of `Provisioners`' references ($\mathcal{P}_1,\ldots,\mathcal{P}_n$), namely descriptors all the necessary information to locate and identify provisioners, and a segment size *seg_size* employed for the log segmentation during the *data transmission* phase.

[3] In order to enhance clarity, we adapt the original syntax of the DELIVER and SEND expressions to emphasize message senders (preceded by the symbol '≪') and receivers (preceded by '≫') respectively.



(a) Initialization

(b) Remote attestation

Figure 6: Unfolding example for the initialization, remote attestation phases of the CONFINE protocol

Controlla che la seg size e la segmentazione sono state introdotte. la seg size ha come dimensione minima la grandezza della traccia piu grande nel log, come dimensione massima la portata della TEE. Assumiamo che la somma dei pezzi di traccie sia piu grande della dimensione massima consentita diviso il numero di

Similarly, the `Provisioner`'s specification in Alg. 2 considers as input the list of references to miners ($\mathcal{M}_1,\ldots,\mathcal{M}_s$) for which event log access is enabled. According to the underlying syntax, $\mathcal{M}$ and $\mathcal{P}$ execute code prompted by events mutually exclusively, implying that they do not concurrently manage two events. For the sake of clarity, we omit any explicit representation of this feature in the pseudo-codes under discussion.

In the following, we describe each protocol phase in detail.

Initialization. The objective of the initialization stage is to inform the miner about the distribution of cases related to a business process among the `Provisioner Nodes`. At the onset of this stage, the `Log Requester` within the `Trusted App` issues n requests, one per `Log Server` component, to retrieve the list of case references they

Algorithm 1: Secure Miner's behavior in CONFINE.

Input: $\widehat{\mathcal{P}} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, the (references to) n log provisioners;
seg_size, the maximum size of the log segment to be transmitted by the log provisioners.

Data: *IIDMap* : $\widehat{\mathrm{IID}} \to 2^{\widehat{\mathcal{P}}}$, a map from case references *iid* $\in \widehat{\mathrm{IID}}$ to the set of log provisioners in $\widehat{\mathcal{P}}$;
PMap : $\widehat{\mathcal{P}} \to 2^{\widehat{\mathrm{IID}}}$, a map from log provisioners $\mathcal{P} \in \widehat{\mathcal{P}}$ to the set of references to their cases in $\widehat{\mathrm{IID}}$;
Cases : $\widehat{\mathrm{IID}} \to \widehat{\mathrm{C}}$, a map from case references *iid* $\in \widehat{\mathrm{IID}}$ to a set of cases in $\widehat{\mathrm{C}}$.

Implements: `SecureMiner`, **instance** $\mathcal{M}$.

Uses: *AuthenticatedPerfectPointToPointLink*, **instance** al.

1 **upon event** $\langle \mathcal{M}, \textsc{Init} \mid \widehat{\mathcal{P}}, seg\_size \rangle$ **do** // The `Log Requester` of $\mathcal{M}$ starts the CONFINE protocol – *initialization* phase in Fig. 6(a))
2   **foreach** $\mathcal{P} \in \widehat{\mathcal{P}}$ **do** // For every `Provisioner` $\mathcal{P}$
3   **trigger** $\langle al, \textsc{Send} \gg \mathcal{P} \mid \textsc{CasesRefReq} \rangle$ // Request $\mathcal{P}$'s case references (see Alg. 2, line 1)
4   **upon** $|\widehat{\mathcal{P}}| = |\mathrm{dom}(PMap)|$ **do** // Once all `Provisioners` have answered with their case references
5   **foreach** $\mathcal{P} \in \widehat{\mathcal{P}}$ **do** // For every `Provisioner` $\mathcal{P}$
6   **foreach** $iid \in PMap[\mathcal{P}]$ **do** // For every case declared by $\mathcal{P}$
7   **if** $WMap[iid] > seg\_size$ **then** $PMap[\mathcal{P}] \leftarrow PMap[\mathcal{P}] \setminus \{iid\}$ // If the weight the of case *iid* is above the *seg_size* do not consider *iid*
8   **trigger** $\langle al, \textsc{Send} \gg \mathcal{P} \mid \textsc{CasesReq}, seg\_size, PMap[\mathcal{P}] \rangle$ // Request the cases of $\mathcal{P}$via al (see Alg. 2, line 4)

9 **upon event** $\langle al, \textsc{Deliver} \ll \mathcal{P} \mid \textsc{CasesRefRes}, WMap' \rangle$ **such that** $\mathcal{P} \in \widehat{\mathcal{P}}$ **do** // $\mathcal{M}$'s `Log Requester` gets $\mathcal{P}$'s case references via al (Alg. 2, line 3)
10   **foreach** $iid, wh \in WMap'$ **do** // For every case reference *iid* received in *IIDs*
11     $IIDMap[iid] \leftarrow IIDMap[iid] \cup \{\mathcal{P}\}$ // Add $\mathcal{P}$ to the set of provisioners for case *iid* in *IIDMap*
12     $WMap[iid] \leftarrow WMap[iid] + wh$
13   $PMap[\mathcal{P}] \leftarrow PMap[\mathcal{P}] \cup IIDs$ // Register the references of the cases provided by $\mathcal{P}$ in *PMap*

14 **upon event** $\langle al, \textsc{Deliver} \ll \mathcal{P} \mid \textsc{CasesRes}, S \rangle$ **such that** $\mathcal{P} \in \widehat{\mathcal{P}}$ **do** // $\mathcal{M}$'s `Log Receiver` gets a segment from $\mathcal{P}$ via al (Alg. 2, line 8))
15   **foreach** $C^{\mathcal{P}}_{iid} \in S$ **do** // For every $C^{\mathcal{P}}_{iid}$ in the delivered segment S, each associated with a *iid*– *data transmission* phase in Fig. 7(a)
16   **if** $iid \in PMap[\mathcal{P}]$ **then** // If $\mathcal{P}$ has declared the ownership of *iid* (see line 9)
17       $PMap[\mathcal{P}] \leftarrow PMap[\mathcal{P}] \setminus \{iid\}$ // Remove *iid* from the set of case references to be provided by $\mathcal{P}$
18       $IIDMap[iid] \leftarrow IIDMap[iid] \setminus \{\mathcal{P}\}$ // Remove $\mathcal{P}$ from the set of *iid* provisioners
19       $\mathcal{M}$.`LogManager.mergeAndStore`$\left(Cases, C^{\mathcal{P}}_{iid}\right)$ // Update the case via $\bigoplus$ and store the result in *Cases*

20 **upon** $IIDMap[iid] = \varnothing$ for some $iid \in \mathrm{dom}(IIDMap)$ **do** // When all the pieces of some *iid* have arrived to $\mathcal{M}$'s `Log Manager`
21   $\mathrm{dom}(IIDMap) \leftarrow \mathrm{dom}(IIDMap) \setminus \{iid\}$ // Remove *iid* from the domain of cases which still needs to be processed
22   **yield** $Cases[iid]$ **to** $\mathcal{M}$.`LogElaborator` // Forward the case *iid* to the `Log Elaborator` of $\mathcal{M}$ for mining – *computation* phase in Fig. 7(b)

record (step 1 in Fig. 6(a) and Alg. 1, line 3). Following sender authentication (2), each `Log Server` retrieves the local event log from the `PAIS` (3, 4) and subsequently responds to the `Log Requester` by providing a list of its associated case references (5 and Alg. 2, line 3). After collecting these n responses (Alg. 1, line 4), the `Log Requester` delineates the distribution of cases. In the context of our motivating scenario, by the conclusion of the initialization, the miner gains knowledge that the case associated with Bob, synthesized in the traces T^H_{711} and T^C_{711}, is exclusively retained by the hospital and the specialized clinic. In contrast, the traces of Alice's case, denoted as T^H_{312}, T^C_{312}, and T^S_{312}, are scattered across all three organizations.

Remote attestation. The remote attestation serves the purpose of establishing trust

Algorithm 2: Provisioner's behavior in CONFINE.

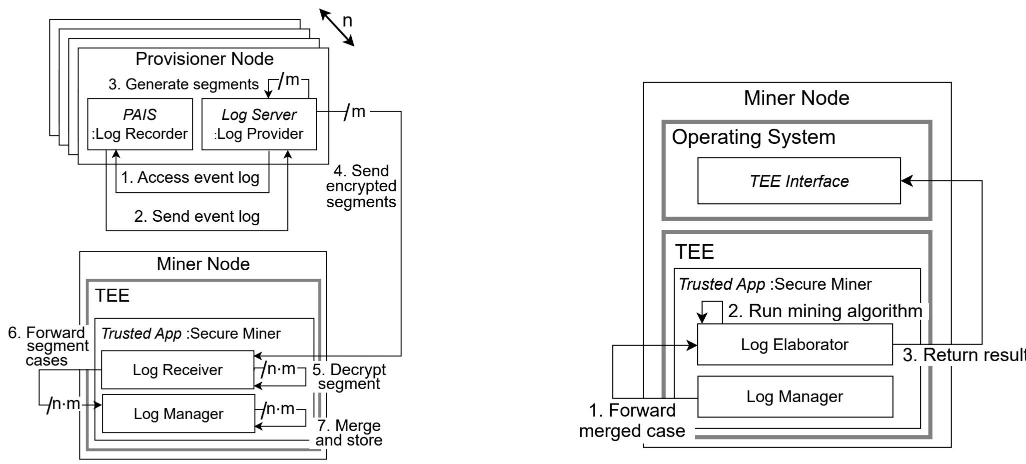
```
Input: M^ = {M_1,...,M_s}, the (references to) s miners.
Implements: Provisioner, instance P.
Uses: AuthenticatedPerfectPointToPointLink, instance al.
1 upon event ⟨al, Deliver ≪ M | CasesRefsReq⟩ such that M ∈ M^ do   // P receives the request for case references from M (see Alg. 1, line 3)
2     WMap ← P.LogRecorder.accessCaseReferences()                   // Access the case references via Log Recorder
3     trigger ⟨al, Send ≫ M | CasesRefRes, WMap⟩                    // send the case references to M (see Alg. 1, line 9)

4 upon event ⟨al, Deliver ≪ M | CasesReq, seg_size, IIDs⟩ such that M ∈ M^ do   // P gets the case request from M (see Alg. 1, line 8)
5     if M.LogReceiver.getAttestationReport(P) is valid then     // Get and verify the attestation report of M – remote attestation in Fig. 6(b)
6         {S_1,...,S_m} ← P.LogProvider.segmentEventLog(P.LogRecorder.accessEventLog(IIDs), seg_size)   // Segment the event log
7         foreach i ∈ {1,...,m} do                                  // For every split segment S_i
8             trigger ⟨al, Send ≫ M | CasesRes, S_i⟩   // send the segment S_i to M (see Alg. 1, line 14) – data transmission phase in Fig. 7(a)
```

between miners and provisioners in the context of fulfilling data requests. This phase adheres to the overarching principles outlined in the RATS RFC standard [21] serving as the foundation for several TEE attestation schemes (e.g., Intel EPID,[4] and AMD SEV-SNP[5]). Remote attestation has a dual objective: *(i)* to furnish provisioners with compelling evidence that the data request for an event log originates from a `Trusted App` running within a TEE; *(ii)* to confirm the specific nature of the `Trusted App` as an authentic `Secure Miner` software entity. This phase is triggered when the `Log Requester` sends a new case request to the `Log Server`(step 1 in Fig. 6(b) and Alg. 2, line 5), specifying: *(i)* the segment size (henceforth, *seg_size*), and *(ii)* the set of the requested case *IIDs*. Both parameters will be used in the subsequent *data transmission* phase. Each of the n `Log Servers` commences the verification process by requesting the necessary information from the `Log Receiver` to conduct the attestation (2). Subsequently, the `Log Receiver` generates the attestation report containing the so-called *measurement* of the `Trusted App`, which is defined as the hash value of the combination of its source code and data. Once this report is signed using the attestation private key associated with the `TEE`'s hardware of the

[4]`sgx101.gitbook.io/sgx101/sgx-bootstrap/attestation`. Accessed: 24/01/2024.

[5]`amd.com/en/processors/amd-secure-encrypted-virtualization`. Accessed: 24/01/2024.



(a) Data transmission

(b) Computation

Figure 7: Unfolding example for the data transmission and computation phases of the CONFINE protocol

`Miner Node`, it is transmitted by the `Log Receiver` to the `Log Servers` alongside the attestation public key of the `Miner Node` (3). The `Log Servers` authenticate the miner using the public key and decrypt the report (4). In this last step, the `Log Servers` undertake a comparison procedure in which they juxtapose the measurement found within the decrypted report against a predefined reference value associated with the source code of the `Secure Miner`. If the decrypted measurement matches the predefined value, the `Miner Node` gains trust from the provisioner.

Data transmission. Once the trusted nature of the `Trusted App` is verified, the `Log Servers` proceed with the transmission of their cases. To accomplish this, each `Log Server` retrieves the event log from the `PAIS` (steps 1 and 2 in Fig. 7(a)), and filters it according to the case reference set specified by the miner. Given the constrained workload capacity of the `TEE`, it is imperative for `Log Servers` to partition the filtered event log into distinct segments. Consequently, each `Log Server` generates m log segments comprising a variable count of entire cases (3 and Alg. 2, line 6). The cumulative size of these segments is governed by the threshold parameter specified by the miner in the initial request (step 1 of the remote attestation phase, Fig. 6(b)). As an illustrative example from our motivating scenario, the `Log Server` of the hospital may structure the segmentation such that T_{312}^{H} and T_{711}^{H} reside within the same segment, whereas the specialized clinic might have T_{312}^{S} and T_{711}^{S} in separate segments. Subsequently, the n `Log Servers` transmit their m encrypted segments to the `Log Receiver` of the `Trusted App` (4 and Alg. 2, line 8). The `Log Receiver`, in turn, collects the $n \times m$ responses in a queue, processing them one at a time. After decrypting the processed segment (5), the `Log Receiver` forwards the cases contained in the segment to the `Log Manager` (6 and Alg. 1, line 19). To reconstruct the process instance, cases belonging to the same process instance must be merged by the `Log Manager` resulting in a single trace (e.g., T_{312} for Alice case) comprehensive of all the events in the partial traces (e.g., T_{312}^{H}, T_{312}^{S} and T_{312}^{C} for Alice case). During this operation, the `Log Manager` applies a specific *merging schema* (i.e., a rule specifying the attributes that link two cases during the merge) as stated in [12]. In our illustrative scenario, the merging schema to combine the cases of Alice is contingent upon the linkage established through their case identifier (i.e., 312). We underline that our proposed solution facilitates the incorporation of diverse merging schemas encompassing distinct trace attributes. The outcomes arising from merging the cases within the processed segment are securely stored by the `Log Manager` in the `TEE`.

Computation. The `Trusted App` requires all the provisioners to have delivered cases referring to the same process instances. For example, when the hospital and the other organizations have all delivered their information concerning case 312 to the `Trusted App`, the process instance associated with Alice becomes eligible for computation. Upon meeting this condition (Alg. 1, line 20), the `Log Manager` forwards the case earmarked for computation to the `Log Elaborator` (step 1 in Fig. 7(b) and Alg. 1, line 22). Subsequently, the `Log Elaborator` proceeds to input the merged case into the process mining algorithm (2). Ultimately, the outcome of the computation is relayed by the `Log Elaborator` from the `TEE` to the `TEE Interface` running atop the `Operating System` of the `Miner Node` (3). The CONFINE protocol does not impose restrictions on the post-computational handling of results. In our motivating scenario, the University and the National Institute of Statistics, serving as miners, disseminate the outcomes of computations, generating analyses that benefit the provisioners (though the original data are never revealed in clear). Furthermore, our protocol enables the potential for provisioners to have their proprietary `Secure Miner`, allowing them autonomous control over the computed results.

7.3. Implementation

We implemented the `Secure Miner` component as an Intel SGX[6] trusted application, encoded in Go through the EGo framework.[7] We resort to a TLS [22] communication channel between miners and provisioners over the HTTP web protocol to secure the information exchange. To demonstrate the effectiveness of our framework, we re-implemented and integrated the *HeuristicsMiner* discovery algorithm [23] within the `Trusted Application`. Our implementation of CONFINE, including the *HeuristicsMiner* in Go, is openly accessible at the following URL: `github.com/Process-in-Chains/CONFINE/`.

[6]`sgx101.gitbook.io/sgx101/`. Accessed: 24/01/2024.
[7]`docs.edgeless.systems/ego`. Accessed: 24/01/2024.

Table 3: Event logs used for our experiments

Name	Type	Activities	Cases	Max events	Min events	Avg. events	Organization $\mapsto$ Activities
Motivating scenario	Synthetic	19	1000	18	9	14	$\mathcal{O}^P \mapsto 3$, $\mathcal{O}^C \mapsto 5$, $\mathcal{O}^H \mapsto 14$
Sepsis [24]	Real	16	1050	185	3	15	$\mathcal{O}^1 \mapsto 1$, $\mathcal{O}^2 \mapsto 1$, $\mathcal{O}^3 \mapsto 14$
BPIC2013 [25]	Real	7	1487	123	1	9	$\mathcal{O}^1 \mapsto 6$, $\mathcal{O}^2 \mapsto 7$, $\mathcal{O}^3 \mapsto 6$

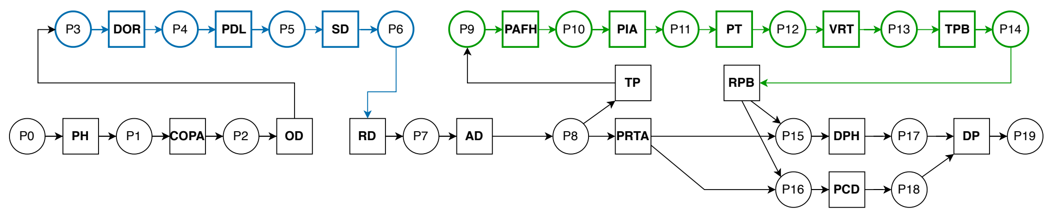
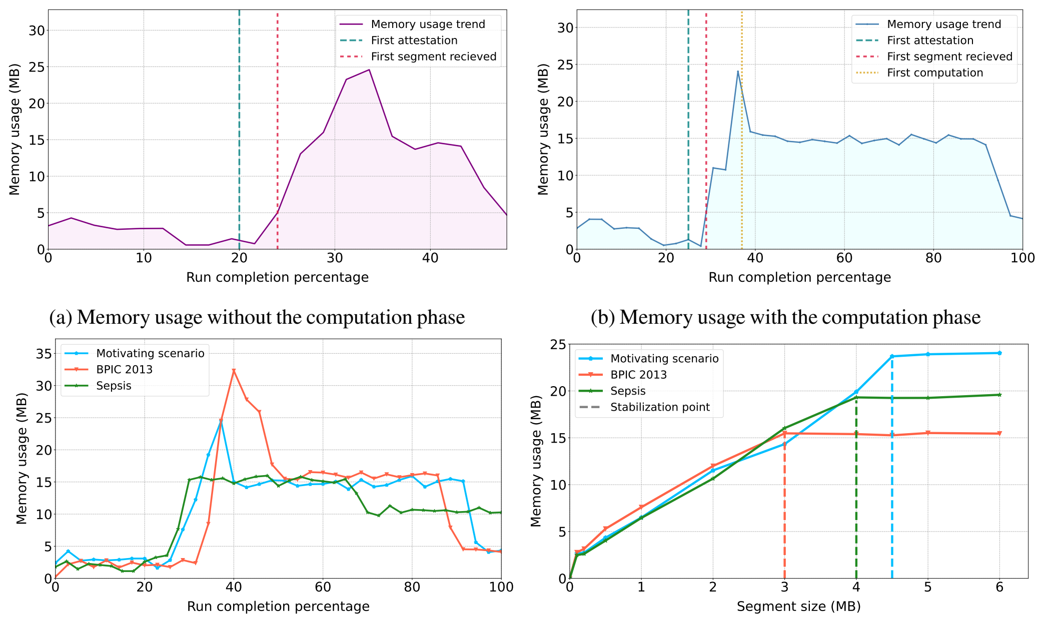


Figure 8: *HeuristicsMiner* output in CONFINE

8. Evaluation

In this section, we evaluate our approach through the testing of our tool implementation. We begin with a convergence analysis to demonstrate the correctness of the collaborative data exchange process. Subsequently, we gauge the memory usage with synthetic and real-life event logs, to observe the trend during the enactment of our protocol and assess scalability. We recall that we focus on memory utilization since the availability of space in the dedicated areas is limited as we discussed in Sect. 7.1. We discuss our experimental results in the following. For the sake of reproducibility, we make available all the testbeds and results in our public code repository (linked above).

Output convergence. To experimentally validate the correctness of our approach in the transmission and computation phases (see Sect. 7), we run a *convergence* test. To this end, we created a synthetic event log consisting of 1000 cases of 14 events on average (see Table 3) by simulating the inter-organizational process of



(a) Memory usage without the computation phase

(b) Memory usage with the computation phase

(c) Memory usage comparison with three event logs

(d) Segment size impact on memory usage

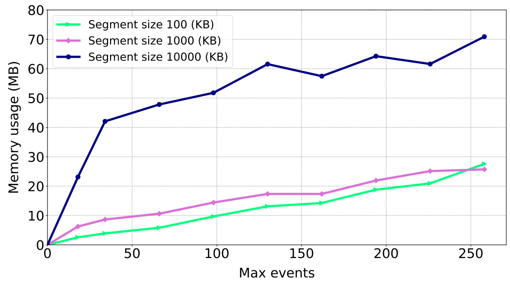
Figure 9: Memory usage test results

our motivating scenario (see Fig. 1)[8] and we partitioned it in three sub-logs (one per involved organization), an excerpt of which is listed in Sect. 2.2. We run the stand-alone *HeuristicsMiner* on the former, and processed the latter through our CONFINE toolchain. As expected, the results converge and are depicted in Fig. 8 in the form of a workflow net [26]. For clarity, we have colored activities recorded by the organizations following the scheme of Table 3 (black for the hospital, blue for the pharmaceutical company, and green for the specialized clinic).

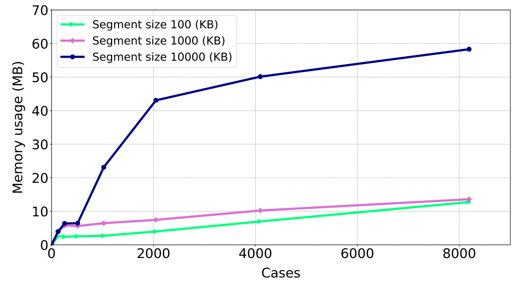
Memory usage. Figures 9(a) and 9(b) display plots corresponding to the runtime memory utilization of our CONFINE implementation (in MegaBytes).

[8]We generated the event log through BIMP (`https://bimp.cs.ut.ee/`). We filtered the generated log by keeping the sole events that report on the completion of activities, and removing the start and end events of the pharmaceutical company and specialized clinic's sub-processes.

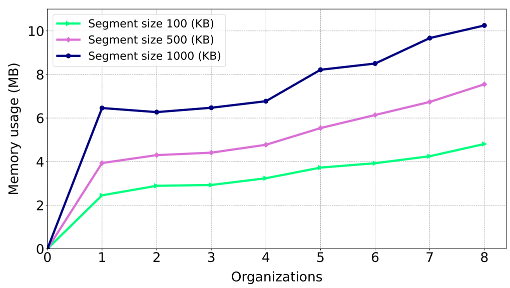
Differently from Fig. 9(b), Fig. 9(a) excludes the computation stage by leaving the *HeuristicsMiner* inactive so as to isolate the execution from the mining-specific operations. The dashed lines mark the starting points for the remote attestation, the data transmission and the computation stages. We held the *seg_size* constant at 2000 KiloBytes. We observe that the data transmission stage reaches the highest peak of memory utilization, which is then partially freed by the subsequent computation stage, steadily occupying memory space at a lower level. To verify whether this phenomenon is due to the synthetic nature of our simulation-based event log, we also gauge the runtime memory usage of two public real-world event logs too (Sepsis [24] and BPIC 2013 [25]). The characteristics of the event logs are summarized in Table 3. Since those are *intra-organizational* event logs, we split the contents to mimic an *inter-organizational* context. In particular, we separated the Sepsis event log based on the distinction between normal-care and intensive-care paths, as if they were conducted by two distinct organizations. Similarly, we processed the BPIC 2013 event log to sort it out into the three departments of the Volvo IT incident management system. Figure 9(c) depicts the results. We observe that the processing of the BPIC 2013 event log demands more memory, particularly during the initial stages, probably owing to its larger size. Conversely, the Sepsis event log turns out to entail the least expensive run. To verify whether these trends are affected by the dimension of the exchanged data segments, we conducted an additional test to examine the trend of memory usage as the *seg_size* varies with all the aforementioned event logs. Notably, the polylines displayed in Fig. 9(d) indicate a linear increment of memory occupation until a breakpoint is reached. After that, the memory in use is steady. These points, marked by vertical dashed lines, correspond to the *seg_size* value that allows the provider's segments to be contained in a single data segment.



(a) Results of scalability test I



(b) Results of scalability test II



(c) Results of scalability test III

Test	**Seg.size**	R^2_{lin}	$\widehat{\beta}$	$R^2_{\log}$
Max events	100	0.9847	0.0980	0.8291
	1000	0.9544	0.0821	0.9043
	10000	0.7357	0.1518	0.9386
Number of cases	100	0.9896	0.0013	0.6822
	1000	0.9629	0.0010	0.8682
	10000	0.7729	0.0068	0.9303
Provisioning organizations	100	0.9770	0.3184	0.8577
	500	0.9602	0.5174	0.7902
	1000	0.9066	0.6102	0.6977

(d) Scalability measurements

Figure 10: Scalability test results

Scalability. In this subsection, we examine the scalability of the `Secure Miner`, focusing on its capacity to efficiently manage an increasing workload in the presence of limited memory resources. We implemented three distinct test configurations gauging runtime memory usage as variations of our motivating scenario log. In particular, we considered (I) the maximum number of events per case, (II) the number of cases $|\widehat{\mathrm{IID}}|$, and (III) the number of provisioning organizations $|\widehat{\mathcal{O}}|$ as independent integer variables. To conduct the test on the maximum number of events, we added a loop back from the final to the initial activity of the process model, progressively increasing the number of iterations $2 \leqslant x_{\circlearrowleft} \leqslant 16$ at a step of 2, resulting in $18+16\cdot(x_{\circlearrowleft}-1)$ events. Concerning the test on the number of cases, we simulated additional process instances so that $|\widehat{\mathrm{IID}}| = 2^{x_{iid}}$ having $x_{iid} \in \{7,8,\dots,13\}$. Finally, for the assessment of the number of organizations, the test necessitated the distribution of the process model activities' into a variable number of pools, each representing a different organization ($|\widehat{\mathcal{O}}| \in \{1,2,...,8\}$). We parameterized the above configurations with three segment sizes (in KiloBytes): *seg_size* $\in \{100, 1000, 10000\}$ for tests I and II, and *seg_size* $\in \{100,500,1000\}$ for test III (the range is reduced without loss of generality to compensate the partitioning of activities into multiple organizations). To facilitate a more rigorous interpretation of the output trends across varying *seg_size* configurations, we employ two well-known statistical measures. As a primary measure of goodness-of-fit, we employ the coefficient of determination R^2 [27], which assesses the degree to which the observed data adheres to the linear (R^2_{lin}) and logarithmic (R^2_{log}) regressions derived from curve fitting approximations. To further delve into the analysis of trends with a high R^2_{lin}, we consider the slope $\widehat{\beta}$ of the approximated linear regression [28].

Table 9(d) lists the measurements we obtained. We describe them to elucidate the observed patterns. Figure 10(a) depicts the results of test I, focusing on the increase of memory utilization when the number of events in the event logs grows. We observe that the memory usage for *seg_size* 100 and 1000 (depicted by green and lilac lines, respectively) are quite similar, whereas the setting with *seg_size* $10{,}000$ (blue line) exhibits significantly higher memory usage. For the settings with *seg_size* 100 and 1000, R^2_{lin} approaches 1, signifying an almost perfect approximation of the linear relation, against lower R^2_{log} values. In these test settings, $\widehat{\beta}$ is very low yet higher than 0, thus indicating that memory usage is likely to continue increasing as the number of max events grows. The configuration with *seg_size* $10{,}000$ yields a higher R^2_{log} value, thus suggesting a logarithmic trend, hence a greater likelihood of stabilizing memory usage growth rate as the number of maximum events increases. In Fig. 10(b), we present the results of test II, assessing the impact of the number of cases on the memory consumption. As expected, the configurations with *seg_size* set to 100 and 1000 exhibit a trend of lower memory usage than settings with *seg_size* $10{,}000$. The R^2_{lin} score of the trends with *seg_size* 100 and 1000 indicate a strong linear relationship between the dependent and independent variables compared to the trend with *seg_size* $10{,}000$, which is better described by a logarithmic regression ($R^2_{\text{log}} = 0.9303$). For the latter, the R^2_{log} value is higher than the corresponding R^2_{lin} thus suggesting that the logarithmic approximation is better suited to describe the trend. Differently from test I, the $\widehat{\beta}$ score associated with the linear approximations of the trends with *seg_size* 100 and 1000 approaches 0, indicating that the growth rate of memory usage as the number of cases increases is negligible. In Fig. 10(c), we present the results of test III, on the relation between the number of organizations and the memory usage. The chart shows that memory usage trends increase as provisioning organizations increase for all three segment sizes. The R^2_{lin} values for the three *seg_size*s are very high, indicating a strong positive linear correlation. The test with *seg_size* 100 exhibits the slowest growth rate, as corroborated by the lowest $\widehat{\beta}$ result (0.3184). For the configuration with *seg_size* 500, the memory usage increases slightly faster ($\widehat{\beta} = 0.5174$). With *seg_size* 1000, the overall memory usage increases significantly faster than the previous configurations ($\widehat{\beta} = 0.6102$). We derive from these findings that the `Secure Miner` may encounter scalability issues when handling settings with a large number of provisioning organizations. Further investigation is warranted to determine the precise cause of this behavior and identify potential mitigation strategies.

In the next section, we conclude our work and outline future research directions based upon our current findings and the limitations of our approach.

9. Conclusion and Future Work

Confidentiality is paramount in inter-organizational process mining due to the transmission of sensitive data across organizational boundaries. Our research investigates a decentralized secrecy-preserving approach that enables organizations to employ process mining techniques with event logs from multiple organizations while ensuring the protection of privacy and confidentiality. Our solution offers a number of directions to walk along for improvement. We operate under the assumption of fair conduct by data provisioners and do not account for the presence of injected or maliciously manipulated event logs. In addition, we assume that miners and provisioners exchange messages in reliable communication channels where no loss or bit corruption occurs. Our approach relies on certain assumptions about event log data, including the existence of a universal clock for event timestamps, which may not be realistic in situations where organizations are not perfectly synchronized. We aim at enhancing our approach to make it robust to the relaxation of these constraints. Our future work encompasses the integration of usage control policies that specify rules on event logs' utilization. We plan to design policy enforcement and monitoring mechanisms to achieve this goal following the principles already addressed in [29, 30]. Our solution embraces process mining techniques in a general way. However, we believe the presented approach is compatible with declarative model representations [31]. Therefore, trusted applications could compute and store the entire set of rules representing a business process, and users may interact with them via trusted queries. Finally, in our implementation, we have focused on process discovery tasks. Nevertheless, our approach has the potential to seamlessly cover a wider array of process mining functionalities such as *conformance checking*, and *performance analysis* techniques. Implementing them and showing their integrability with our approach paves the path for future research endeavors.

Acknowledgments. The authors thank Giuseppe Ateniese for the fruitful discussion and insights. This research work was partly funded by MUR under PRIN grant B87G22000450001 (PINPOINT), by the Latium Region under PO FSE+ grant B83C22004050009 (PPMPP), and by the EU-NGEU under the NRRP MUR grant PE00000014 (SERICS).

EXTENSION PLAN:

Extended introduction

Add Background Section

Add more related work

Modify the motivating scenario (and the design alongside the implementation) with more attributes.

Add Notation and formalization of event logs, merging, partitioning and segmentation

Add Soundness and completeness theorems

Add threat model $\bigvee$

Full pseudocode of the protocol $\bigvee$

More real-world event logs (plus two, at least) with associated tests

Add communication overhead v. segment size charts: elab time vs segment size; total time (incl. network) vs segment size.

Integrate declarative conformance checking (let it be with Janus or MINERful).

Modify conclusion

References

[1] W. M. P. van der Aalst, et al., Process mining manifesto, in: BPM Workshops, 2012, pp. 169–194.

[2] W. M. P. van der Aalst, Intra-and inter-organizational process mining: Discovering processes within and between organizations, in: PoEM, 2011, pp. 1–11.

[3] C. Liu, Q. Li, X. Zhao, Challenges and opportunities in collaborative business

process management: Overview of recent advances and introduction to the special issue, Inf. Syst. Front. 11 (2009) 201–209.

[4] M. Müller, N. Ostern, Koljada, et al., Trust mining: analyzing trust in collaborative business processes, IEEE Access (2021) 65044–65065.

[5] M. Sabt, M. Achemlal, A. Bouabdallah, Trusted execution environment: What it is, and what it is not, in: 2015 IEEE TrustCom/BigDataSE/ISPA, 2015, pp. 57–64.

[6] M. Müller, A. Simonet-Boulogne, S. Sengupta, O. Beige, Process mining in trusted execution environments: Towards hardware guarantees for trust-aware inter-organizational process analysis, in: ICPM, 2021, pp. 369–381.

[7] G. Elkoumy, S. A. Fahrenkrog-Petersen, et al., Shareprom: A tool for privacy-preserving inter-organizational process mining, in: BPM (PhD/Demos), 2020, pp. 72–76.

[8] G. Elkoumy, S. A. Fahrenkrog-Petersen, et al., Secure multi-party computation for inter-organizational process mining, in: BPMDS/EMMSAD, 2020, pp. 166–181.

[9] W. M. P. van der Aalst, Federated process mining: Exploiting event data across organizational boundaries, in: SMDS 2021, 2021, pp. 1–7.

[10] R. Cramer, I. Damgård, J. B. Nielsen, Secure Multiparty Computation and Secret Sharing, Cambridge University Press, 2015.

[11] C. Zhao, S. Zhao, Zhao, et al., Secure multi-party computation: Theory, practice and applications, Inf. Sci. 476 (2019) 357–372.

[12] J. Claes, G. Poels, Merging event logs for process mining: A rule based merging method and rule suggestion algorithm, Expert Syst. Appl. 41 (2014) 7291–7306.

[13] J. D. Hernandez-Resendiz, E. Tello-Leal, H. M. Marin-Castro, et al., Merging event logs for inter-organizational process mining, in: New Perspectives on Enterprise Decision-Making Applying Artificial Intelligence Techniques, Springer, 2021, pp. 3–26.

[14] M. Jans, M. Hosseinpour, How active learning and process mining can act as continuous auditing catalyst, Int. J. Accounting Inf. Systems 32 (2019) 44–58.

[15] N. Koch, A. Kraus, The expressive power of UML-based web engineering, in: IWWOST02, volume 16, 2002, pp. 40–41.

[16] M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, Fundamentals of Business Process Management, Second Edition, Springer, 2018.

[17] V. Costan, S. Devadas, Intel SGX explained, Cryptology ePrint Archive (2016).

[18] P. Jauernig, A.-R. Sadeghi, E. Stapf, Trusted execution environments: Properties, applications, and challenges, IEEE Secur. Priv. 18 (2020) 56–60.

[19] R. L. Rivest, A. Shamir, L. M. Adleman, A method for obtaining digital signatures and public-key cryptosystems (reprint), Commun. ACM 26 (1983) 96–99.

[20] C. Cachin, R. Guerraoui, L. E. T. Rodrigues, Introduction to Reliable and Secure Distributed Programming (2. ed.), Springer, 2011.

[21] H. Birkholz, D. Thaler, M. Richardson, et al., Remote ATtestation procedureS (RATS) Architecture, 2023.

[22] S. A. Thomas, SSL and TLS Essentials: Securing the Web, Wiley, 2000.

[23] A. J. M. M. Weijters, W. M. P. van der Aalst, A. K. Alves De Medeiros, Process mining with the HeuristicsMiner algorithm, 2006.

[24] F. Mannhardt, Sepsis cases - event log, 2016. doi:10.4121/UUID:915D2BFB-7E84-49AD-A286-DC35F063A460.

[25] W. Steeman, BPI challenge 2013, incidents, 2013. doi:10.4121/UUID:500573E6-ACCC-4B0C-9576-AA5468B10CEE.

[26] W. M. P. van der Aalst, Verification of workflow nets, in: ICATPN, 1997, pp. 407–426.

[27] J. P. Barrett, The coefficient of determination—some limitations, The American Statistician (1974) 19–20.

[28] N. Altman, M. Krzywinski, Simple linear regression, Nature Methods (2015) 999–1000.

[29] D. Basile, C. Di Ciccio, V. Goretti, S. Kirrane, Blockchain based resource governance for decentralized web environments, Frontiers in Blockchain (2023) 1141909.

[30] D. Basile, C. Di Ciccio, V. Goretti, S. Kirrane, A blockchain-driven architecture for usage control in solid, in: ICDCSW, 2023, pp. 19–24.

[31] C. Di Ciccio, M. Montali, Declarative process specifications: Reasoning, discovery, monitoring, in: Process Mining Handbook, Springer, 2022, pp. 108–152.