A TEE-based Approach for Preserving Data Secrecy in Process Mining with Decentralized Sources

Davide Basile[a], Valerio Goretti[a], Luca Barbaro[a], Hajo A. Reijers[b], 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

Process mining techniques enable organizations to gain insights into their business processes through the analysis of execution records (event logs) stored by information systems. While most process mining efforts focus on intra-organizational scenarios, many real-world business processes span multiple independent organizations.

Inter-organizational process mining, though, faces significant challenges, particularly regarding confidentiality guarantees: The analysis of data can reveal information that the participating organizations may not consent to disclose to one another, or to a third party hosting process mining services. To overcome this issue, this paper presents CONFINE, an approach for secrecy-preserving inter-organizational process mining. CONFINE leverages Trusted Execution Environments (TEEs) to deploy trusted applications that are capable of securely mining multi-party event logs while preserving data secrecy. We propose an architecture supporting a four-stage protocol to secure data exchange and processing, allowing for protected transfer and aggregation of unaltered process data across organizational boundaries. To avoid out-of-memory errors due to the limited capacity of TEEs, our protocol employs a segmentation-based strategy, whereby event logs are transmitted to TEEs in smaller batches. We conduct a formal verification of our approach's correctness alongside a security analysis on the guarantees provided by the TEE core. We test our implementation using real-world and synthetic data to assess memory usage. Our experiments show that an incremental approach to segment processing in discovery and conformance checking is preferable over non-incremental strategies as the former maintains memory usage trends within a narrow range at runtime, whereas the latter exhibit high peaks towards the end of the execution. Furthermore, our results confirm that our prototype can handle real-world workloads without out-of-memory failures. The scalability tests reveal that memory usage grows logarithmically as the event log size increases. Memory consumption grows linearly with the number of provisioning organizations, indicating potential scalability limitations and opportunities for further optimizations.

Keywords: Secrecy, Process Mining, Confidential Computing, Business Process Management

1. Introduction

In the management landscape, the notion of *business process* (or process for short) conceptualizes and gives structure to the interplay of events, activities, and decision points to achieve organizations' operational objectives [1]. With the rise of process automation, information systems' data have become a precious information source to enhance processes. The discovery, monitoring, and improvement of processes via the extraction of knowledge from chronological execution records is the focus of the research field of process mining [2]. Input data is typically stored in structures known as *event logs* [3], which collect multiple process instance execution traces (named *cases*) as sequences of instantaneous records (named *events*).

To date, the vast majority of process mining contributions consider *intra-organizational* settings, in which business processes are executed within individual organizations. However, organizations 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 [4]. Through inter-organizational

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

process mining, organizations can gain a deeper understanding of their shared business processes, which would be missed by only considering fragmented views of distinct executions [5]. Such a comprehensive perspective supports the identification of bottlenecks entailed by the cross-organizational interactions, providing qualitative and quantitative insights into the effectiveness of the collaboration [6]. In the context of supply-chain management, this broader view enhances the analysis of how goods, information, and decisions circulate across multiple organizations, thereby supporting the optimization of logistic workflows, reducing operational costs, and improving delivery performance [7]. Additionally, inter-organizational process mining can reveal behavioral dependencies and structural mismatches arising from erroneous process enactment practices, thus being instrumental to root-cause analysis of failures. In the manufacturing industry, for example, it can identify the causes of recurrent production anomalies by uncovering incorrectly modeled dependencies between activities carried out across different organizational units [8]. Inter-organizational process mining also provides a shared ground for jointly verifying conformance with standards that span multiple organizational contexts and regulate cooperation [9]. This becomes particularly valuable in domains such as healthcare, where stringent regulations oversee the administration of patient pathways that involve multiple autonomous institutions [10, 11].

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 secrecy. Companies are reluctant to share private information, as required to execute process mining algorithms with their partners [12]. Indeed, letting sensitive operational data traverse organizational boundaries introduces concerns about data privacy, security, and compliance with internal regulations [13]. To address this issue, the majority of research endeavors have focused thus far on the alteration of input data or of intermediate analysis by-products, with the aim to impede the reconstruction of the original information sources from the counterparts [14, 15, 16, 17]. These preemptive solutions have the merit of neutralizing information leakage by malicious parties *a priori*. Nevertheless, they entail an ex-ante information loss, thus compromising downstream process mining accuracy [14, 15], or require the use of intermediary log representations, which restricts the range of applicable algorithms [16, 17].

To improve over the state of the art, we propose CONFINE, a novel approach and toolset aimed at enhancing collaborative information system architectures with secrecy-preserving process mining capabilities. Inspired by the vision of decentralized data ownership in Solid [18, 19], our solution decentralizes data provenance across multiple autonomous provisioning organizations, each retaining ownership of their (partitions of the) event logs. Individual logs embody the organizations' own partial view on the shared inter-organizational process.

To secure information secrecy during the exchange and elaboration of data, our solution resorts to *Trusted Execution Environments* (TEEs) [20]. These are hardware-secured contexts, which guarantee code integrity and data confidentiality before, during, and after their use. Owing to these characteristics, CONFINE lets information be securely transferred beyond an organization's borders. Therefore, computing nodes other than the information provisioners can aggregate and elaborate the original, unaltered process data in a secure, externally inaccessible vault.

The architecture of CONFINE supports a four-staged protocol: *(i)* The initial exchange of preliminary metadata, *(ii)* the attestation of the mining entities, *(iii)* the protected transmission and secrecy-preserving merge of encrypted information segments among multiple organizations, *(iv)* the computation of process discovery and conformance checking algorithms on joined data in a hardware-secured enclave. We evaluate CONFINE proving the soundness and completeness of our protocol in its most crucial phase of information transmission and merge *(iii)*. Also, we discuss the security properties brought by the integration of TEEs at the core of our approach against a set of security threats.

Since TEEs operate with limited sections of the main memory shielded from access by external entities (operating system included), exceeding these limits may expose the system to potential data leakage. Therefore, we endow our experiments with an analysis of memory usage. Given its direct impact on data secrecy, we analyze how memory usage scales by increasing the number of *(i)* processed events, *(ii)* processed cases, and *(iii)* participating organizations. The results indicate that memory consumption exhibit a logarithmic increase as the volume and length of process instances grow, thereby reducing the likelihood of out-of-memory conditions even with large amounts of processed data. However, when increasing the number of participating

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

organizations, the experiments show a linear growth of memory consumption, thereby delineating an avenue for future optimization work.

In this article, we extend and enhance our earlier work [21, 22] with the following main contributions: *(i)* we formally define the core concepts underpinning our approach (Sect. 4); *(ii)* we endow the description of our CONFINE protocol with a formal specification with pseudocode, and provide an in-depth exploration of the main components' behavior (Algs. 1 to 4 in Sect. 6); *(iii)* we discuss the security guarantees provided by the TEE to frame and motivate its role within our solution (Sect. 7.1); *(iv)* we demonstrate the soundness and completeness of the data-transmission and merge phases within the CONFINE protocol (Sect. 7.2); *(v)* we extend the performance assessment on memory usage by introducing novel scalability experiments (Sects. 8.3.1 and 8.3.2); *(vi)* we discuss the practical implications inherent to a real-world deployment of CONFINE (Sect. 9.1).

The remainder of this paper is as follows. Sect. 2 provides an overview of the background and the related work. In Sect. 3, we introduce a motivating use-case scenario in healthcare. In Sect. 4, we propose the formal notations characterizing our work. Sect. 5 presents the functional viewpoint of CONFINE and Sect. 6 describes the realization of CONFINE alongside its four-stage protocol. In the latter sections, we highlight the assumptions under which our solution operates as the discourse unfolds. In Sect. 7, we report on formal validation of our approach's correctness and security. Sect. 8 presents an empirical validation of our implementation with tests on memory usage and scalability. In Sect. 9, we discuss the practical implications and limitations, and outline future research directions beginning with a revisit of the assumptions mentioned in Sects. 5 and 6. We conclude our work in Sect. 10.

2. Background and Related Work

In what follows, we provide an overview of the background literature and relevant existing research, laying the foundation for our study.

2.1. Background

Here we introduce the theoretical concepts pertinent to the core areas of our work, i.e., process mining and trusted execution environments.

2.1.1. Process Mining

Process mining refers to a suite of techniques designed to uncover, monitor, and refine processes by analyzing event logs to offer analytical insights into business operations [23]. Events within an event log typically possess attributes providing descriptive details such as the executed task, its timestamp, and the involved human resources. Event logs organize events related to the same process instances into *cases*. The process mining literature delineates three primary families of techniques: *discovery*, *conformance checking*, and *enhancement* algorithms [2]. Discovery techniques generate a process model (i.e., a depiction of the process) from an event log without relying on any initial information [24]. The *HeuristicsMiner* is a seminal process discovery algorithm [24]. It constructs a dependency graph by computing the frequency of directly subsequent occurrences between activities within the same cases and deriving dependency measures capturing their causal relations. It then filters relations below a significance threshold to remove noise, and employs the remaining ones to build a weighted graph. This graph is finally transformed into a *causal net*, a process model supporting a basic set of core workflow patterns including concurrency and (short) loops [25].

Conformance checking algorithms verify whether a given event log aligns with an input process model and vice versa [26]. In this paper, we consider the *DeclareConformance* algorithm as a noticeable example [27]. It verifies the compliance of an event log with a declarative process specification, i.e., a set of behavioral constraints that capture dependencies among activities. The algorithm first projects the event log into activity sequences on a per-trace basis and then systematically checks each constraint of the model against these sequences. Detected violations are used to compute a fitness score for each trace, which the algorithm subsequently aggregates into a global score representing the overall adherence of the event log to the declarative

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

specification.Enhancement techniques extend or refine an existing process model using information gathered from the actual process recorded in an event log [2, 28].

Conventionally, companies apply process mining techniques to analyze their internal business processes within *intra-organizational* settings. However, the occurrence of scenarios where business processes span across multiple independent organizations, with each organization maintaining its own event log, is becoming increasingly common [12]. We refer to this application of process mining as *inter-organizational process mining*. Inter-organizational process mining classifies interactions into two basic settings: *collaboration* and *exploiting commonality*. In collaboration, business processes are distributed among collaborators, who cooperate to achieve a shared objective. In the commonality exploitation setting, organizations share common processes and may compete, learn from each other, or exchange experiences, knowledge, and infrastructure. Hence, the involved organizations need not collaborate as each of them carries out the same tasks independently. In both inter-organizational scenarios, multiple data sources are involved, posing significant challenges in data aggregation and subsequent algorithm application [4].

In this work, we concentrate on the privacy-preserving aspects of collaborative scenarios, recognizing organizations' need to share event logs across organizational boundaries while safeguarding data confidentiality. To tackle this challenge, our solution draws on the theoretical foundations of confidential computing, whose core concepts are elucidated as follows.

2.1.2. Confidential Computing and Trusted Execution Environments

Confidential computing encompasses a suite of techniques aimed at safeguarding data while it is in use. These methods leverage hardware-backed protection of highly sensitive computations within attested *Trusted Execution Environments* (TEEs). According to the Confidential Computing Consortium (CCC) [1], TEEs are secure execution contexts that provide assurance regarding three key properties: (*data confidentiality*) unauthorized entities cannot view data while it is in use within the TEE, (*data integrity*) unauthorized entities cannot add, remove, or alter data while it is in use within the TEE, and (*code integrity*) unauthorized entities cannot add, remove, or alter code executing in the TEE [20]. TEE technologies are diverse, and their features may differ depending on the specific implementation of the underlying Central Processing Unit (CPU).

TEEs differ primarily in the extension of their trusted surface, that is, the hardware-enforced boundary defining the extent of isolation. Two main classes can be distinguished based on this criterion [29]. The first includes application-level (or process-level) TEEs, which protect a restricted portion of an application's address space. In such architectures, implemented among others by Intel SGX[2] and ARM TrustZone[3], only specific regions of memory explicitly designated as trusted are protected with confidentiality and integrity guarantees, whereas the remaining system components are treated as untrusted [30]. The second class comprises virtual-machine level TEEs, such as AMD SEV[4] and Intel TDX,[5] which extend the isolation boundary to an entire guest virtual machine. These TEEs protect memory and CPU state belonging to the virtual machine from inspection or tampering by (even higher-privileged) external software [31].

Another distinguishing characteristic among TEE technologies lies in whether the memory regions allocated to the isolated environment are protected through hardware-based encryption. In TEE technologies such as Intel SGX, AMD SEV, and Intel TDX, the CPU encrypts and protects the integrity of data within the trusted domain. This feature relies on dedicated sections of a CPU capable of handling encrypted data within a reserved section of the main memory [32]. This protected memory section is encrypted by the CPU using a random key derived at every power cycle. In contrast, technologies like ARM TrustZone mainly rely on access control layers between trusted and untrusted execution modes, with no hardware-based encryption involved.

[1] confidentialcomputing.io. Accessed: December 2, 2025.
[2] software-guard-extensions/overview. Accessed: December 2, 2025.
[3] arm.com/technologies/trustzone. Accessed: December 2, 2025.
[4] amd.com/sev. Accessed: December 2, 2025.
[5] intel.com/trust-domain-extensions/overview. Accessed: December 2, 2025.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

A further essential feature characterizing certain TEE technologies, including Intel SGX, Intel TDX, and AMD SEV, is *attestability*. Through this feature, the CPU can produce a cryptographic proof, typically referred to as *attestation evidence*, to demonstrate the trusted status of the TEE to a local or remote verifier. Following the Remote ATtestation procedureS (RATS) standard [33], remote attestation protocols generally consist of the following steps: (1) The *verifier* establishes a communication channel with the *attester* (which may correspond to a trusted application or a virtual machine) and requests the generation of the attestation evidence. (2) The attester then instructs the CPU to produce the attestation evidence, which serves as cryptographic proof of its trusted state. In most protocols, the attestation evidence takes the form of a digital signature derived from hardware-bound secrets and includes a *measurement*, namely a cryptographic hash representing the state of the virtual machine or the trusted application's code. The attestation evidence is signed using a hardware-protected attestation key embedded in the processor. Depending on the TEE technology, the verifier may rely on the CPU manufacturer to validate this signature using the corresponding public endorsement credentials and to confirm the integrity of the reported measurement.. (3) Upon receiving the attestation evidence, the verifier validates its authenticity by relying on an *endorser* (typically the TEE manufacturer) responsible for providing the necessary public credentials and reference values. The verifier subsequently compares the reported measurement against one or more reference values to reach a trust decision. The successful completion of this process attests to the integrity and authenticity of the entity providing the attestation evidence.

Attestable TEE implementations enable trusted virtual machines and applications to embed custom data into the attestation evidence at the time of its generation (see step 2 above). This feature allows verifiers to apply tailored appraisal policies by evaluating the included information. Furthermore, this custom data field can serve to authenticate the trusted origin of cryptographic material, which may subsequently be leveraged to establish secure communication channels once the attestation evidence has been successfully verified [34]. While CPU manufacturers such as Intel and AMD provide web services supplying the validation material required to verify attestation evidence generated by their TEE-enabled processors, they delegate to system designers the responsibility of defining appropriate trust-decision policies and establishing secure communication channels [35].

In this work, we consider application-level TEEs that provide hardware-backed memory encryption and support remote attestation. Accordingly, our contribution implicitly assumes TEEs endowed with these three essential characteristics.

2.2. Related Work

Despite the recency of this research branch across process mining and collaborative information systems, scientific literature already includes noticeable contributions to inter-organizational process mining. Before discussing specific contributions, we categorize existing secrecy preservation methodologies into three main classes, depending on whether they are based on (I) event log sanitization, (II) Secure Multiparty Computation (SMPC), and (III) TEE. In Tab. 1, we summarize and compare the pros and cons of each approach. (I) *Sanitization-based approaches* apply obfuscation mechanisms to transform event logs prior to their analysis. The main advantage of these approaches lies in preserving privacy while maintaining compatibility with existing process mining algorithms, as the sanitized output remains a full-fledged event log. However, they introduce noise that may distort analytical results, hindering case-level mining capabilities, and exert no control over the subsequent process mining computation. (II) *SMPC-based approaches* employ cryptographic protocols that allow multiple organizations to jointly execute verifiable computations over encrypted data without revealing their original event logs to each other. These approaches ensure a high degree of control

Table 1: Comparison of secrecy-preserving approaches for process mining

Approach	**Zero noise**	**Algorithm compatibility**	**Hardware independence**	**Computational control**
Sanitization-based	✗	✓	✓	✗
SMPC-based	✗	✗	✓	✓
TEE-based	✓	✓	✗	✓

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

over the computation itself. Nevertheless, they typically limit the range of process mining algorithms supported and may introduce noise in the final results. (III) *TEE-based approaches* leverage hardware-backed cryptographical methodologies to enable the secure execution of process mining algorithms within isolated environments. This paradigm achieves zero-noise processing (as data remains encrypted even during processing, avoiding distortions), control over the computation, and seamless compatibility with existing process mining algorithms, as TEEs allow the execution of Turing-complete code either as-is or with only minimal adaptations. However, unlike sanitization and SMPC-based approaches, these solutions depend on specific hardware platforms, thereby imposing constraints on the underlying technology.

Next, we discuss existing research contributions, grouped according to the three aforementioned categories.

(I) **Sanitization-based approaches.** Fahrenkrog-Petersen et al. [14, 15] introduce the PRETSA algorithms family: a set of event log sanitization techniques that perform step-wise transformations of prefix-tree event log representation into a sanitized output ensuring *k-anonimization* and *t-closeness*. Batista et al. [36] present the k-PPPM algorithm, designed to ensure the protection of identities within an input event log. This objective is achieved through the clustering of the sub-logs corresponding to each individual and the subsequent approximation by selecting representative centroids. While these two approaches effectively minimize information loss, they introduce approximations within the original event log, which may compromise the exactness of process mining results or inhibit mining tasks. In contrast with the latter two techniques, our research proposes a secure architecture wherein secure computational vaults collect event logs devoid of upstream alterations and protect them at runtime, thus generating results derived directly from the original information source.

(II) **SMPC-based approaches.** Elkoumy et al. [16, 17] present Shareprom, an SMPC-based approach. Akin to 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 organizations operating within the same context. Shareprom's functionality, though, is confined to the execution of operations involving event log abstractions [37] represented as directed acyclic graphs, which the organizations employ as intermediate pre-elaboration to be fed into SMPC routines [38]. The final outputs are then modified through the addition of Laplacian noise. This approach empowers data owners to retain a degree of control over the process mining techniques applied to their event logs. However, as the authors note, the reliance on the graph representation introduces constraints that may be limiting in various process mining scenarios. In particular, the resulting graph abstraction is compatible with only a subset of discovery and conformance checking algorithms. In contrast, our approach supports the encrypted transmission and analysis of event logs without relying on any intermediate abstraction. This enables the faithful reproduction of existing process mining techniques within computationally isolated environments, thereby providing greater flexibility.

(III) **TEE-based approaches.** The seminal work of Müller et al. [39] is the first to envision the adoption of TEEs within the process mining domain. In their paper, the authors propose a conceptual architecture in which each participating organization encrypts its local event log, publishes it to a public repository, and delegates the mining computation to a remote trusted application executed inside a TEE. The interaction between miners and the public repository is coordinated through a blockchain-based control plane, while a Secret Management System (SMS) running in a separate TEE is responsible for storing the decryption keys and releasing them to the miner after successful attestation. While this work represents a foundational contribution and an essential source of inspiration for our research, it leaves several challenges unaddressed that our work explicitly aims to address.

A key distinction from the work of Müller et al. concerns the event log provisioning protocol, which constitutes a key contributions of our work. In [39], a specific provisioning strategy was not designed, and event logs are transmitted and processed as single and monolithic datasets. This can lead to confidentiality breaches as large event logs may exceed the limited memory available within TEEs and result in crashes or untrusted memory allocations [40, 41]. To overcome these issues, our research introduces a segmentation-based provisioning protocol in which event logs are partitioned into smaller batches that are individually transmitted and merged within the TEE. Building on this segmentation mechanism, we further propose an incremental processing model that allows the miner running within the TEE to update a partial result as new segments arrive from different organizations during transmission. Therefore, the process mining output converges to

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

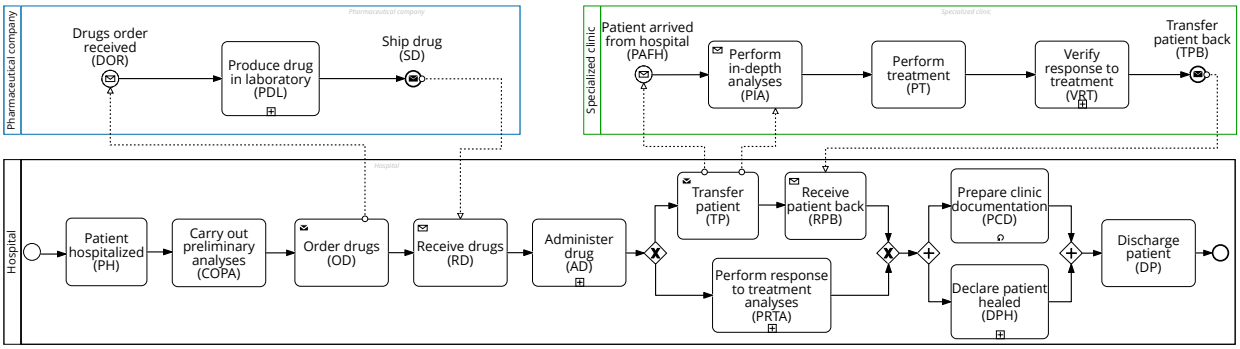


Figure 1: A BPMN collaboration diagram of a simplified healthcare scenario

the same result as a full merged log computation, while avoiding the risks associated with memory peaks that may occur with monolithic processing. From an architectural standpoint, our approach reduces the attack surface inherent in that seminal work, which arises from the inclusion of utility components operating outside organizational boundaries. Although designed to store encrypted data, the event log repository proposed in the research of Müller et al. remains vulnerable to malicious inspection or cryptanalysis attacks due to its public nature. Similarly, the SMS running within the TEE may entail vulnerabilities. Since TEEs are inherently non-persistent environments, any data residing in their volatile memory is lost once the TEE's machine is turned off. Consequently, the previous approach would require additional mechanisms to anchor the cryptographic material in persistent memory outside the TEE. However, this practice inevitably extends the trusted boundary and may introduce rollback attacks [42], in which an adversary with control over the host system can revert a sealed state to a previous version, thereby re-enabling outdated or revoked authorizations. To overcome these limitations, we propose a peer-to-peer model in which miners operating within a TEE retrieve event logs directly from native organizations, which, in turn, act as decentralized data sources [18] that independently handle the encrypted data transmission. Additionally, within every execution of our protocol, we generate ephemeral encryption keys on demand and use them solely for the current transmission session, without requiring any permanent external key storage.

3. Motivating Scenario

To provide a running example and motivating scenario for our investigation, we propose a simplified hospitalization process for the treatment of rare diseases. Our running example draws inspiration from

Table 2: Events from cases 312 (Alice) and 711 (Bob) recorded by the hospital, the specialized clinic, and the pharmaceutical company

Hospital

	Case	Timestamp	Activity
e_1	312	2022-07-14T10:36	PH
e_2	312	2022-07-14T16:36	COPA
e_3	711	2022-07-14T17:21	PH
e_4	312	2022-07-14T17:36	OD
e_5	711	2022-07-14T23:21	COPA
e_6	711	2022-07-15T00:21	OD
e_7	711	2022-07-15T18:55	RD
e_8	312	2022-07-15T19:06	RD
e_9	711	2022-07-15T20:55	AD
e_{10}	312	2022-07-15T21:06	AD

	Case	Timestamp	Activity
e_{11}	312	2022-07-15T22:06	TP
e_{12}	711	2022-07-16T00:55	PRTA
e_{13}	711	2022-07-16T01:55	PCD
e_{14}	711	2022-07-16T02:55	DPH
e_{15}	711	2022-07-16T04:55	DP
e_{16}	312	2022-07-16T07:06	RPB
e_{17}	312	2022-07-16T09:06	DPH
e_{18}	312	2022-07-16T10:06	PCD
e_{19}	312	2022-07-16T11:06	DP

Pharmaceutical company

	HospitalCaseID	Timestamp	Activity
e_{20}	312	2022-07-15T09:06	DOR
e_{21}	711	2022-07-15T09:30	DOR
e_{22}	312	2022-07-15T11:06	PDL
e_{23}	711	2022-07-15T11:30	PDL
e_{24}	312	2022-07-15T13:06	SD
e_{25}	711	2022-07-15T13:30	SD

Specialized clinic

	TreatmentID	Timestamp	Activity
e_{26}	312	2022-07-16T00:06	PAFH
e_{27}	312	2022-07-16T01:06	PIA
e_{28}	312	2022-07-16T03:06	PT
e_{29}	312	2022-07-16T04:06	VRT
e_{30}	312	2022-07-16T05:06	TPB

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

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

two notable contributions in the domain of healthcare processes: the work of Aguilar et al. [43], which illustrates modeling practices for representing healthcare workflows, and the work of Sampson et al. [44], which emphasizes coordination principles among independent healthcare institutions. The process model of our running example is represented as a Business Process Model and Notation (BPMN) diagram in Fig. 1. BPMN is among the most widely adopted standards for process modeling, describing processes in terms of interacting activities, events, and control-flow constructs like decision and concurrent execution gateways.[6]

Our scenario involves three cooperating organizations: 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 Tab. 2. 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. Afterward, 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 [45] 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 Tab. 2, 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} =⟨PH, COPA, OD, DOR, PDL, SD, RD, AD, TP, PAFH, PIA, PT, VRT, TPB, RPB, DPH, PCD, DP⟩ and T_{711} =⟨PH, COPA, OD, DOR, PDL, SD, RD, AD, PRTA, PCD, DPH, DP⟩. 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 organizations *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.

4. Definitions and Formal Properties

Thus far, we have intuitively introduced the concepts of events, cases, event logs and related aspects. In this section, we delve deeper into these notions from a formal standpoint. The definitions provided henceforward pave the ground for properties and aspects to which we will resort in the design and realization of our solution.

[6] https://www.omg.org/spec/BPMN/2.0. Accessed: December 2, 2025.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

Definition 4.1 (Event). Let $\widehat{\mathrm{E}} \ni e$ be a finite non-empty set of symbols. We refer to e as *event* and to $\widehat{\mathrm{E}}$ as the *universe of events*. An *attribute* is a function having the universe of events as its domain.

Referring to the example of Alice and Bob's hospitalization process (see Sect. 3), each event denoted as e_i with $1 \leqslant i \leqslant 30$ in Tab. 2 is contained in the universe of events $\widehat{\mathrm{E}}$.

Due to the general nature of elements that can be linked to events via attribute functions (in the example, e.g., timestamps and activity labels), we leave the range unspecified, akin to the definition of finite sequence as a function in [46]. We assume the following attributes as mandatory as in the classical process mining literature [47].

Postulate 4.1 (Mandatory attributes)**.** *The following attributes are total on the universe of events* $\widehat{\mathrm{E}}$*:* **instance identifier** *(or iid for short), a total surjective function* $\mathfrak{iid} : \widehat{\mathrm{E}} \to \widehat{\mathrm{IID}}$*, where* $\widehat{\mathrm{IID}}$ *is a finite non-empty set we name* instance set*;* **activity label**, *a total function* $\mathfrak{act} : \widehat{\mathrm{E}} \to \widehat{\mathrm{A}}$*, where* $\widehat{\mathrm{A}}$ *is a finite non-empty set we name* activity set*;* **timestamp**, *a total function* $\mathfrak{tms} : \widehat{\mathrm{E}} \to \mathbb{N}$*, where* $\mathbb{N}$ *is the set of positive integers.*

In the context of our motivating scenario (see Tab. 2), $\mathfrak{iid}(e_1) = 312$. This instance identifier serves the purpose of collating the events associated with Alice's hospitalization across the three organizations. Furthermore, $\mathfrak{act}(e_1)$ and $\mathfrak{tms}(e_1)$ map e_1 to its activity label (i.e., `PH`) and timestamp (i.e., `2022-07-14T10:36`), respectively. For the sake of readability, we henceforth format the timestamps in our examples using the ISO 8601 string standard. Nonetheless, they stem from an integer representation (e.g., UNIX time), as per the above definition.

Definition 4.2 (Event log). Given the universe of events $\widehat{\mathrm{E}}$ as per Def. 4.1, let $\preceq \subseteq \widehat{\mathrm{E}} \times \widehat{\mathrm{E}}$ be a total-order relation defined over $\widehat{\mathrm{E}}$. The pair $\mathrm{L} = \left(\widehat{\mathrm{E}}, \preceq\right)$ is an *event log*.

In our example, the event log $\mathrm{L} = \left(\widehat{\mathrm{E}}, \preceq\right)$ is the totally ordered set consisting of all the events $e_1, \ldots, e_{30}$ recorded by the hospital, the pharmaceutical company, and the specialized clinic. We assume that the total order of events can be reconstructed based on the timestamp associated with events, which we express with the following postulate.

Postulate 4.2 (Timestamp as an order isomorphism)**.** *Given an event log* $\mathrm{L} = \left(\widehat{\mathrm{E}}, \preceq\right)$*, the timestamp attribute* $\mathfrak{tms} : \widehat{\mathrm{E}} \to \mathbb{N}$ *is an order isomorphism, i.e., given* $e, e' \in \widehat{\mathrm{E}}$ *with* $e \neq e'$*,* $e \preceq e'$ *if and only if* $\mathfrak{tms}(e) \leqslant \mathfrak{tms}(e')$*.*

Examining the events e_{30} and e_{16} in our motivating scenario, we can deduce from the order relation established based on their timestamps (i.e., `2022-07-16T05:06` and `2022-07-16T07:06`, respectively) that e_{30} precedes e_{16}, i.e., $e_1 \preceq e_{30}$. This rationale holds true for any generic pair of events selected from the event log.

Definition 4.3 (Process mining function). Given an event log L, a *process mining function* $\underset{\hbar_1,\ldots,\hbar_k}{prom}$ is a function having as its arguments L and a tuple of hyperparameters $\hbar_1, \ldots, \hbar_k$, with $k \in \mathbb{N}$.

We leave the above definition purposefully underspecified to cater for the width of process mining techniques ranging over discovery, conformance checking, and process enhancement. Those categories all have in common that the event log is the input, whereas the hyperparameters and the output vary. A process discovery technique such as the *heuristic miner*, e.g., takes a set of thresholds to pruning out noise as its hyperparameters and returns a heuristic net [24]. A consequence of covering process mining techniques under the umbrella of a function is that we assume them to be deterministic.

In our setting, events may be recorded by different entities. We name these entities *provisioners* and formally define them as follows.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

Definition 4.4 (Provisioner). Let $\widehat{\mathcal{P}} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ be a non-empty finite set of entities (with $n \in \mathbb{N}$) named *provisioner*s. Given the universe of events $\widehat{\mathrm{E}}$ as per Def. 4.1, $\xrightarrow{\mathcal{P}}$ is a total surjective function having $\widehat{\mathrm{E}}$ as the domain and $\widehat{\mathcal{P}}$ as its range. Function $\xrightarrow{\mathcal{P}}$ defines an equivalence relation $\overset{\mathcal{P}}{\sim}$ on $\widehat{\mathrm{E}}$ (i.e., $e \overset{\mathcal{P}}{\sim} e'$ iff $e \xrightarrow{\mathcal{P}} \mathcal{P}_i$ and $e' \xrightarrow{\mathcal{P}} \mathcal{P}_i$ for some $\mathcal{P}_i \in \widehat{\mathcal{P}}$, $1 \leqslant i \leqslant n$) and hence partitions $\widehat{\mathrm{E}}$ into equivalence classes we denote with $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_i}$ for any $\mathcal{P}_i \in \widehat{\mathcal{P}}$.

In the depicted scenario, we introduce three provisioners: the hospital, the pharmaceutical company, and the specialized clinic, henceforth referred to as $\mathcal{P}_{\mathrm{Hsp}}$, $\mathcal{P}_{\mathrm{PhC}}$, and $\mathcal{P}_{\mathrm{SpC}}$. Events e_1 and e_{19} in Tab. 2 fall within the same equivalence class $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_{\mathrm{Hsp}}}$ as they are both recorded by the hospital. Consequently, the relation $e_1 \overset{\mathcal{P}}{\sim} e_{19}$ holds true. Conversely, e_1 and e_{30} are recorded by the hospital and the specialized clinic, respectively, placing them in distinct equivalence classes, namely $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_{\mathrm{Hsp}}}$ and $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_{\mathrm{SpC}}}$. Therefore, the relation $e_1 \overset{\mathcal{P}}{\sim} e_{30}$ does not hold.

Definition 4.5 (Log partition). Given an event log $\mathrm{L} = \left(\widehat{\mathrm{E}}, \preceq\right)$ as per Def. 4.2, provisioner $\mathcal{P} \in \widehat{\mathcal{P}}$ and the equivalence class $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}} \subseteq \widehat{\mathrm{E}}$ as per Def. 4.4, a *log partition* is a pair $\mathrm{L}_{\mathcal{P}} = \left(\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}}, \preceq_{\mathcal{P}}\right)$, where $\preceq_{\mathcal{P}}$ is a total order restricting $\preceq$ on $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}}$, i.e., $\preceq_{\mathcal{P}}$ is $(\preceq) \cap \left(\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}} \times \widehat{\mathrm{E}}^{\sim}_{\mathcal{P}}\right)$.

In our setting, we posit that each organization independently maintains its own log partition. The events $e_1, e_2, \ldots, e_{19}$ in the $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_{\mathrm{Hsp}}}$ equivalence class are contained in the hospital's log partition $\left(\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_{\mathrm{Hsp}}}, \preceq_{\mathcal{P}_{\mathrm{Hsp}}}\right)$. Similarly, the totally ordered sets defined over $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_{\mathrm{PhC}}} = \{e_{20}, e_{21}, \ldots, e_{25}\}$ and $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_{\mathrm{SpC}}} = \{e_{26}, e_{27}, \ldots, e_{30}\}$ are the pharmaceutical company and the specialized clinic's log partitions, respectively – i.e., $\left(\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_{\mathrm{PhC}}}, \preceq_{\mathcal{P}_{\mathrm{PhC}}}\right)$ and $\left(\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_{\mathrm{SpC}}}, \preceq_{\mathcal{P}_{\mathrm{SpC}}}\right)$.

We recall that the restriction of a totally ordered set (like the event log) is a total order. Therefore, the log partition is a totally ordered set, too, and hence an event log. In the remainder of this paper, we shall name as **sublog** a restriction of an event log (omitting the reference to the event log whenever clear from the context). For instance, since $e_1 \preceq e_2$, then $e_1 \preceq_{\mathcal{P}_{\mathrm{Hsp}}} e_2$.

Definition 4.6 (Segment). Given a log partition $\left(\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}}, \preceq_{\mathcal{P}}\right)$ defined over the equivalence class $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}}$ by provisioner $\mathcal{P} \in \widehat{\mathcal{P}}$ as per Defs. 4.4 and 4.5, and a subset $\{\mathit{iid}_1, \ldots, \mathit{iid}_k\} \subseteq \widehat{\mathrm{IID}}$ (see Post. 4.1) with $1 \leqslant k \leqslant |\widehat{\mathrm{IID}}|$, $k \in \mathbb{N}$, a *segment* over $\{\mathit{iid}_1, \ldots, \mathit{iid}_k\}$ is a pair $S = \left(\widehat{\mathrm{E}}^{\sim}_{S}, \preceq_{S}\right)$ wherein events $e_S \in \widehat{\mathrm{E}}^{\sim}_{S} \subseteq \widehat{\mathrm{E}}^{\sim}_{\mathcal{P}}$ are such that $\mathrm{iid}(e_S) \in \{\mathit{iid}_1, \ldots, \mathit{iid}_k\}$ and $\preceq_S$ is a total order restricting $\preceq_{\mathcal{P}}$ on $\widehat{\mathrm{E}}^{\sim}_{S}$, i.e., $(\preceq_{\mathcal{P}}) \cap \left(\widehat{\mathrm{E}}^{\sim}_{S} \times \widehat{\mathrm{E}}^{\sim}_{S}\right)$.

In light of the above, the segment is a totally ordered set, too. Therefore, a segment is an event log (or sublog, depending on the context). Also, notice that a segment can be the whole log partition if the given set of *iid*s contains all the *iid*s to which the events in the log partition map. Finally, notice that all events that map to a given *iid* in a log partition are in the same segment. In the context of our example, e.g., consider the pharmaceutical company's log partition $\left(\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_{\mathrm{PhC}}}, \preceq_{\mathcal{P}_{\mathrm{PhC}}}\right)$ and Alice's hospitalization (*iid* 312). As $\mathrm{iid}(e_{20}) = \mathrm{iid}(e_{22}) = \mathrm{iid}(e_{24}) = 312$ and $e_{20}, e_{22}, e_{24} \in \widehat{\mathrm{E}}^{\sim}_{\mathcal{P}_{\mathrm{PhC}}}$, all the three aforementioned events reside within the same segment. In other words, Def. 4.6 precludes any scenario wherein two events pertaining to Alice's case in the pharmaceutical company's partition are allocated to different segments (e.g., it cannot happen that e_{20} is in S' and e_{22}, e_{24} are in S" if $S' \neq S''$).

Definition 4.7 (Case). Let $\mathrm{L} = \left(\widehat{\mathrm{E}}, \preceq\right)$ be an event log as per Def. 4.2. Let $\mathrm{iid} : \widehat{\mathrm{E}} \to \widehat{\mathrm{IID}}$ be the total surjective function of the mandatory attribute *iid* (Post. 4.1) that defines an equivalence relation $\overset{\mathrm{iid}}{\sim}$ on $\widehat{\mathrm{E}}$ (i.e., $e \overset{\mathrm{iid}}{\sim} e'$ iff $\mathrm{iid}(e) = \mathit{iid}$ and $\mathrm{iid}(e') = \mathit{iid}$ for some $\mathit{iid} \in \widehat{\mathrm{IID}}$) and hence divides $\widehat{\mathrm{E}}$ into equivalence classes. Given

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

Table 3: Glossary of notation and summary of formal concepts

Symbol	Brief description	Reference
$e \in \widehat{\mathrm{E}}$	An event e in a universe of events $\widehat{\mathrm{E}}$	Def. 4.1
$\mathfrak{iid} : \widehat{\mathrm{E}} \to \widehat{\mathrm{IID}}$	The *iid* attribute mapping events to instance identifiers in $\widehat{\mathrm{IID}}$	Def. 4.1 and Post. 4.1
$\mathfrak{act} : \widehat{\mathrm{E}} \to \widehat{\mathrm{A}}$	The attribute mapping events to activity labels in $\widehat{\mathrm{A}}$	Def. 4.1 and Post. 4.1
$\mathfrak{tms} : \widehat{\mathrm{E}} \to \mathbb{N}$	The attribute mapping events to timestamps	Def. 4.1 and Post. 4.1
$\mathrm{L} = \left(\widehat{\mathrm{E}}, \preceq\right)$	The event log, a set of events totally ordered by $\preceq \subseteq \widehat{\mathrm{E}} \times \widehat{\mathrm{E}}$. The timestamp is an order isomorphism to $\preceq$.	Def. 4.2 and Post. 4.2
$\underset{\hbar_1,\ldots,\hbar_k}{prom}$	A process mining function with k hyperparameters $\hbar_1, \ldots, \hbar_k$	Def. 4.3
$\mathcal{P} \in \widehat{\mathcal{P}}$	A provisioner $\mathcal{P}$ in a universe of provisioners $\widehat{\mathcal{P}}$	Def. 4.4
$\xrightarrow{\mathcal{P}}: \widehat{\mathrm{E}} \to \widehat{\mathcal{P}}$	The function mapping events to provisioners in $\widehat{\mathcal{P}}$	Def. 4.4
$\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}}$	The equivalence class of the events in $\widehat{\mathrm{E}}$ related to the same provisioner $\mathcal{P}$	Def. 4.4
$e \overset{\mathcal{P}}{\sim} e'$	The equivalence relation between two events in $\widehat{\mathrm{E}}$ related to the same provisioner $\mathcal{P}$	Def. 4.4
$\mathrm{L}_{\mathcal{P}} = \left(\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}}, \preceq_{\mathcal{P}}\right)$	The log partition, a set of events in $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}}$ ordered by $\preceq_{\mathcal{P}}$. $\preceq_{\mathcal{P}}$ is a total order restricting $\preceq$ to $\widehat{\mathrm{E}}^{\sim}_{\mathcal{P}}$	Def. 4.4
$S = \left(\widehat{\mathrm{E}}^{\sim}_{S}, \preceq_{S}\right)$	The segment, a subset of a log partition collecting all events whose *iid* belongs to a given subset of $\widehat{\mathrm{IID}}$ ordered by the restriction $\preceq_S$ of $\preceq_{\mathcal{P}}$	Def. 4.6
$e \overset{\mathfrak{iid}}{\sim} e'$	The equivalence relation between two events related to the same *iid*	Def. 4.7
$\widehat{\mathrm{E}}^{\sim}_{iid}$	The equivalence class of the events in $\widehat{\mathrm{E}}$ related to the same *iid*	Def. 4.7
$C_{iid} = \left(\widehat{\mathrm{E}}^{\sim}_{iid}, \preceq_{iid}\right)$	The case, a set of events in $\widehat{\mathrm{E}}$ related to the same *iid* ordered by $\preceq_{iid}$. $\preceq_{iid}$ is a total order restricting $\preceq$ on $\overset{\mathfrak{iid}}{\sim}$	Def. 4.7
$\left(\widehat{\mathrm{E}}_{\mathrm{L}'}, \preceq_{\mathrm{L}'}\right) \oplus \left(\widehat{\mathrm{E}}_{\mathrm{L}''}, \preceq_{\mathrm{L}''}\right)$	The merge between two event logs, an event log containing the set of all the events in $\widehat{\mathrm{E}}_{\mathrm{L}'}$ and $\widehat{\mathrm{E}}_{\mathrm{L}''}$ ordered by a superset of $(\preceq_{\mathrm{L}'} \cup \preceq_{\mathrm{L}''})$. A merge is *safe* if the output is totally ordered and the timestamp attribute is an order isomorphism.	Def. 4.8 and Prop. 4.1

an instance identifier $iid \in \widehat{\mathrm{IID}}$, a *case* C_{iid} is a pair $\left(\widehat{\mathrm{E}}^{\sim}_{iid}, \preceq_{iid}\right)$, wherein $\widehat{\mathrm{E}}^{\sim}_{iid} \subseteq \widehat{\mathrm{E}}$ is the equivalence class for *iid* by the equivalence relation $\overset{\mathfrak{iid}}{\sim}$, and $\preceq_{iid}$ is a total order restricting $\preceq$ on $\widehat{\mathrm{E}}^{\sim}_{iid}$, i.e., $\preceq_{iid}$ is $(\preceq) \cap \left(\widehat{\mathrm{E}}^{\sim}_{iid} \times \widehat{\mathrm{E}}^{\sim}_{iid}\right)$. Given a *iid* and L, $\mathcal{C}$ is the function returning C_{iid} from L as above, i.e., $C_{iid} = \mathcal{C}(\mathrm{L}, iid)$. We shall omit the *iid* subscript from C_{iid} when clear from the context. We denote with $\widehat{\mathrm{C}} \ni C$ the set of all cases in L, i.e., $\widehat{\mathrm{C}} = \bigcup_{iid \in \widehat{\mathrm{IID}}} \left\{\left(\widehat{\mathrm{E}}^{\sim}_{iid}, \preceq_{iid}\right)\right\}$.

Considering our motivating scenario, we can distinguish Alice and Bob's cases, referred to as $C_{312} = \mathcal{C}(\mathrm{L}, 312)$ and $C_{711} = \mathcal{C}(\mathrm{L}, 711)$, respectively. Events e_1 and e_{30} belong to the same equivalence class $\widehat{\mathrm{E}}^{\sim}_{312}$ (i.e., the class associated with C_{312}), given that $\mathfrak{iid}(e_1) = \mathfrak{iid}(e_{30}) = 312$. Therefore, the relation $e_1 \overset{\mathfrak{iid}}{\sim} e_{30}$ holds. In contrast, e_1 and e_{21} fall in different equivalence classes (i.e., $\widehat{\mathrm{E}}^{\sim}_{312}$ and $\widehat{\mathrm{E}}^{\sim}_{711}$, respectively), as $\mathfrak{iid}(e_1) = 312$ and $\mathfrak{iid}(e_{21}) = 711$. Thus, $e_1 \overset{\mathfrak{iid}}{\sim} e_{30}$ does not hold true.

Intuitively, a case is derived from an event log by taking all events that share the same *iid* in that log. Also partitions, segments, and sublogs in general are logs, thus the composition of cases depend on the structure from which they are extrapolated. Furthermore, since a case is per se a totally ordered set of events, it is an event log itself. Notice that a case is a restriction of an *event log* to events having the same *unique instance identifier* in common. A segment, instead, is a restriction of a *log partition* to events that have a non-empty *set of instance identifiers* in common. In other words, segments can contain events mapping to different *iid*'s, whilst cases cannot. On the other hand, cases can contain events stemming from different provisioners, whilst segments cannot.

We conclude this section with an operation defined on event logs (hence sublogs, i.e., segments, partitions, and cases as well) we name *merge*, and an important property that a merge should enjoy to be declared *safe*.

Definition 4.8 (Merge). Let $\mathrm{L}' = \left(\widehat{\mathrm{E}}_{\mathrm{L}'}, \preceq_{\mathrm{L}'}\right)$ and $\mathrm{L}'' = \left(\widehat{\mathrm{E}}_{\mathrm{L}''}, \preceq_{\mathrm{L}''}\right)$ be two event logs as per Def. 4.2. The *merge* of two event logs $\mathrm{L}' \bigoplus \mathrm{L}''$ is a pair $\left(\widehat{\mathrm{E}}_{\mathrm{L}'} \cup \widehat{\mathrm{E}}_{\mathrm{L}''}, \preceq_{\oplus}\right)$ where $\preceq_{\oplus}$ is a superset of $(\preceq_{\mathrm{L}'} \cup \preceq_{\mathrm{L}''})$. A merge is *safe* if the two following conditions hold:

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

1. $\preceq_{\bigoplus}$ is a total order over $\widehat{\mathrm{E}}_{\mathrm{L}'} \cup \widehat{\mathrm{E}}_{\mathrm{L}''}$;
2. the timestamp attribute $\mathfrak{tms} : \widehat{\mathrm{E}}_{\mathrm{L}'} \cup \widehat{\mathrm{E}}_{\mathrm{L}''} \to \mathbb{N}$ is an order isomorphism as per Post. 4.2.

Notice that the merge of two event logs (or sublogs) still is a totally ordered set (and hence an event log, or a sublog) if and only if the merge is safe. In other words, event logs are closed under the safe-merge operation. For instance, a non-safe merge between L' and L'' could be computed as $\left(\widehat{\mathrm{E}}_{\mathrm{L}'} \cup \widehat{\mathrm{E}}_{\mathrm{L}''}, \preceq_{\mathrm{L}'} \cup \preceq_{\mathrm{L}''}\right)$. Take, e.g., the segment over *iid* 312 in the hospital's partition as L' and the segment over *iid* 312 in the pharmaceutical company's partition as L''. With a non-safe merge like the one we just mentioned, the order would not be defined for either of e_{11} (in L') and e_{20} (in L''), among others.

Proposition 4.1. *Let* L', L'', L''' *be three event logs. If* $\bigoplus$ *is a safe merge, it enjoys the properties of (i) associativity* $((\mathrm{L}' \bigoplus \mathrm{L}'') \bigoplus \mathrm{L}''') = (\mathrm{L}' \bigoplus (\mathrm{L}'' \bigoplus \mathrm{L}'''))$ *and (ii) commutativity* $(\mathrm{L}' \bigoplus \mathrm{L}'') = (\mathrm{L}'' \bigoplus \mathrm{L}')$*, and (iii) has* $(\{\}, \{\})$*, i.e., a log with no events, as its identity element.*

Proof. (Sketch) It follows from the definition of $\bigoplus$, employing the associative and commutative operator $\cup$ having $\{\}$ as its identity element, and the safety condition imposing a deterministic assignment of the order relation based on the events' timestamps over $\preceq_{\bigoplus}$. □

For the sake of readability, we recapitulate the formal concepts introduced thus far in a glossary of notation in Tab. 3, listing them in their order of appearance within this section. Equipped with these notions, we provide an architectural overview of our solution in the next section.

5. Design

Our goal is to enable the secure aggregation and elaboration of original, unaltered event logs from decentralized sources in dedicated environments that potentially lie beyond the individual organizations' information perimeter. With this objective in mind, we devise the `Secure Miner` component, which is capable of securing data merge and processing by running certified code in an isolated execution vault. Thus, we decouple provisioning from treatment, and the two tasks can be carried out by distinct computing nodes. In this section, we outline the high-level architecture of CONFINE.

5.1. Architectural overview

Here, we introduce CONFINE's key components, with a special focus on the `Secure Miner` (Sect. 5.1.2) after an outline of the overall architecture (Sect. 5.1.1).

5.1.1. 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 mined (as per Def. 4.4); **miner** if it applies process mining algorithms (see Def. 4.3) using event logs retrieved from provisioners. In our adversarial model, we assume that provisioning organizations behave as *honest* entities (**Assumption A1**), whereas miner organizations may act in a *malicious* manner (**Assumption A2**). This scenario characterizes situations in which the members of a consortium cooperate towards a common business objective and are thus motivated to gain benefits from constructive behavior. Consequently, we do not expect provisioning organizations to deviate from the prescribed protocol or manipulate the transmitted event logs. By contrast, miner organizations may attempt to exploit their role by inferring sensitive information from the received event logs.

In Fig. 2, we propose the high-level schematization of the CONFINE framework. Henceforth, we will refer to provisioning organizations' event logs as *log partitions*, according to Def. 4.5. In our solution, every organization hosts one or more nodes realizing the architectural components. 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*

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

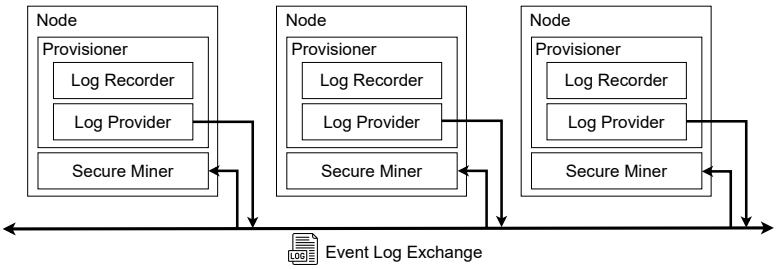


Figure 2: The CONFINE high-level architecture

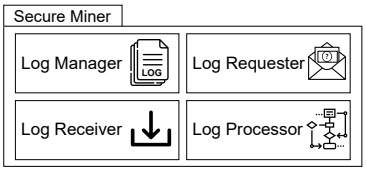


Figure 3: The Secure Miner

(introduced in Def. 4.1) taking place in the organizations' systems, and builds the local log partitions. The `Log Provider` delivers on-demand log partitions to mining players.

As formalized in Post. 4.1, we assume events in log partitions are comprised of three mandatory attributes, namely, the *iid*, the *activity* label and the *timestamp* (**Assumption A3**; see Post. 4.1). We further assume that iids are synchronized across organizations so that they consistently refer to the same process instance (**Assumption A4**). The `Log Recorder` of the hospital and all other organizations in our example refer to Alice and Bob's cases (see Def. 4.7) with iids 312 and 711, respectively. The `Log Recorder` is queried by the `Log Provider` for log partitions to be made available for mining. The latter controls access to local log paritions by authenticating data requests by miners and rejecting those that come from unauthorized organizations. In our motivating scenario, the specialized clinic, the pharmaceutical company, and the hospital leverage `Log Providers` to authenticate the miner party before sending their log partition. The `Secure Miner` component shelters external log partitions 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.

5.1.2. The Secure Miner

The primary objective of the `Secure Miner` is to allow miners to securely execute process mining algorithms using decentralized log partitions 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 Processor`. The `Log Requester` and the `Log Receiver` are the sub-components that we employ during the log partition retrieval. `Log Requesters` send authenticable data requests to the `Log Providers`. The `Log Receiver` collects log partitions 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 Manager` combines the event data locked in the `Secure Miner` to have a global view of the inter-organizational process, encompassing activities executed by each involved provisioning organization. We refer to this operation as *merge*, according to Def. 4.8. In the context of the merge, we assume that the events contained in the individual log partitions expose timestamp attributes that are mutually synchronized across organizations (**Assumption A5**). The `Log Processor` executes process mining algorithms using merged event logs in a protected environment, inaccessible from the outside computation environment. A `Log Processor` implements a process mining function $\underset{\hbar_1,\ldots,\hbar_k}{prom}$ following Def. 4.3.In our motivating scenario, the `Log Manager` 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}) and generates the chronologically sorted traces T_{312} and T_{711} preserving the events' temporal sequence (as described in Post. 4.2). Subsequently, the `Log Processor` feeds these cases into the mining algorithms.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

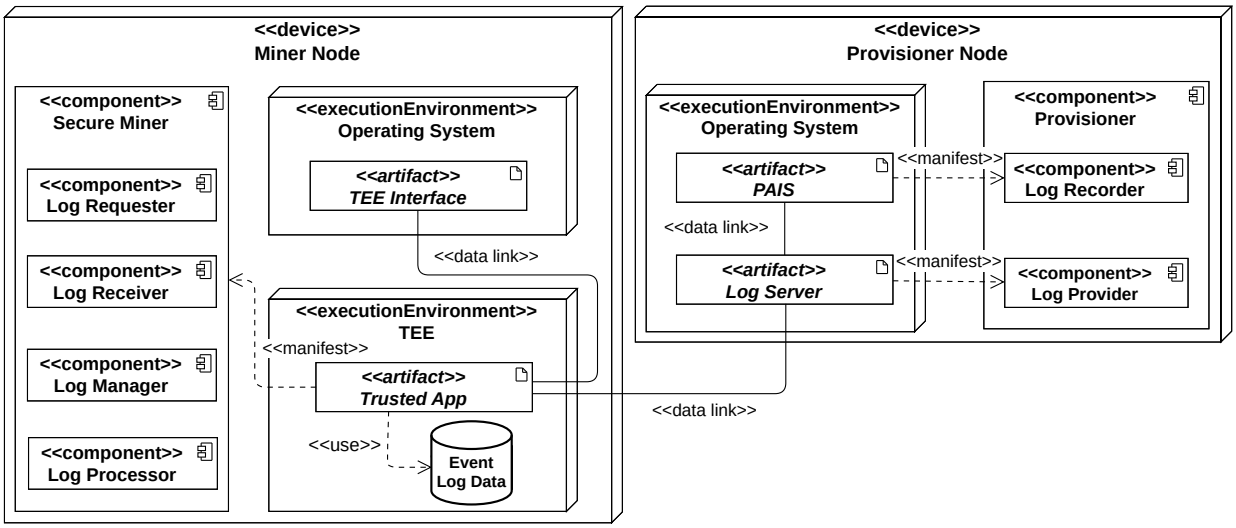


Figure 4: UML deployment diagram of the CONFINE architecture

6. 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 architectural principles presented in Sect. 5. Afterwards, we discuss the CONFINE interaction protocol.

6.1. Deployment

Fig. 4 depicts a Unified Modeling Language (UML) deployment diagram [48] 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 scenario, e.g., the hospital can be a provisioner (as per Def. 4.4) and a miner at the same time. Our deployment setting assumes the establishment of Identity and Access Management (IAM) services capable of issuing, for each node, an *identity proof* that authenticates its owning organization (**Assumption A6**).

`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` [1]. 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 log partitions (introduced in Def. 4.5) to miners.

The `Miner Node` is characterized by two distinct *execution environments*: the `Operating System` (`OS`) and the Trusted Execution Environment (`TEE`) [20]. `TEE`s establish isolated contexts separate from the `OS`, safeguarding code and data through hardware-based encryption mechanisms.

As introduced in Sect. 2.1.2, we assume our `TEE` profile is characterized by the following distinctive features: *(i)* application-level trusted surface, *(ii)* hardware-based memory encryption, and *(iii)* remote attestability (**Assumption A7**). Through the application-level trusted surface, we execute a `Trusted App` responsible for implementing the core functions of the `Secure Miner` and its associated sub-components within the `TEE`'s isolated domain. The `TEE` preserves the integrity of the `Trusted App` code by shielding it from malicious tampering and preventing unauthorized access by processes running in the `Operating System`.

Through hardware-based memory encryption, the `TEE` ensures that the log partitions processed by the `Secure Miner` remain encrypted throughout their computation, confining them to dedicated and protected regions of the main memory. By enforcing memory access restrictions, the `TEE` prevent one trusted application from reading or altering the memory space of another. These dedicated areas in memory are limited, though.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

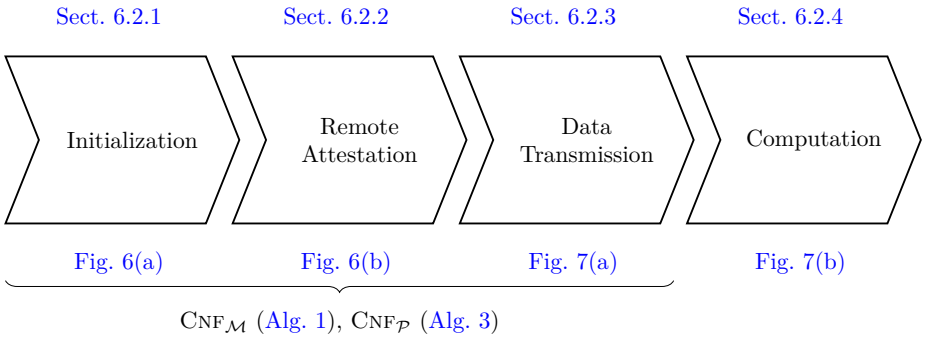


Figure 5: Phases of the CONFINE protocol with references to the related sections, figures, and algorithms in this article

Once the limits are exceeded, TEEs have to scout around in outer memory areas, thus conceding the opportunity to malicious readers 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 memory demand goes beyond the available space. Within the CONFINE protocol, we mitigate this issue by directing the `Secure Miner` to elaborate log partitions in a smaller batch of cases (defined in Def. 4.7) we refer to as *segment* (see Def. 4.6).

Finally, we rely on the `TEE`'s feature of remote attestability, which enables `Provisioners` to verify the trustworthiness of the `Secure Miner`. In particular, we leverage the customizable portion of the attestation evidence to embed ephemeral cryptographic means generated at runtime by our `Secure Miner`. Those cryptographic tools are then used to establish a secure communication channel for the encrypted transmission of log partitions.

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 `TEE Interface`, for safely receiving information from the `Operating System` and outsourcing the results of computations.

6.2. The CONFINE protocol

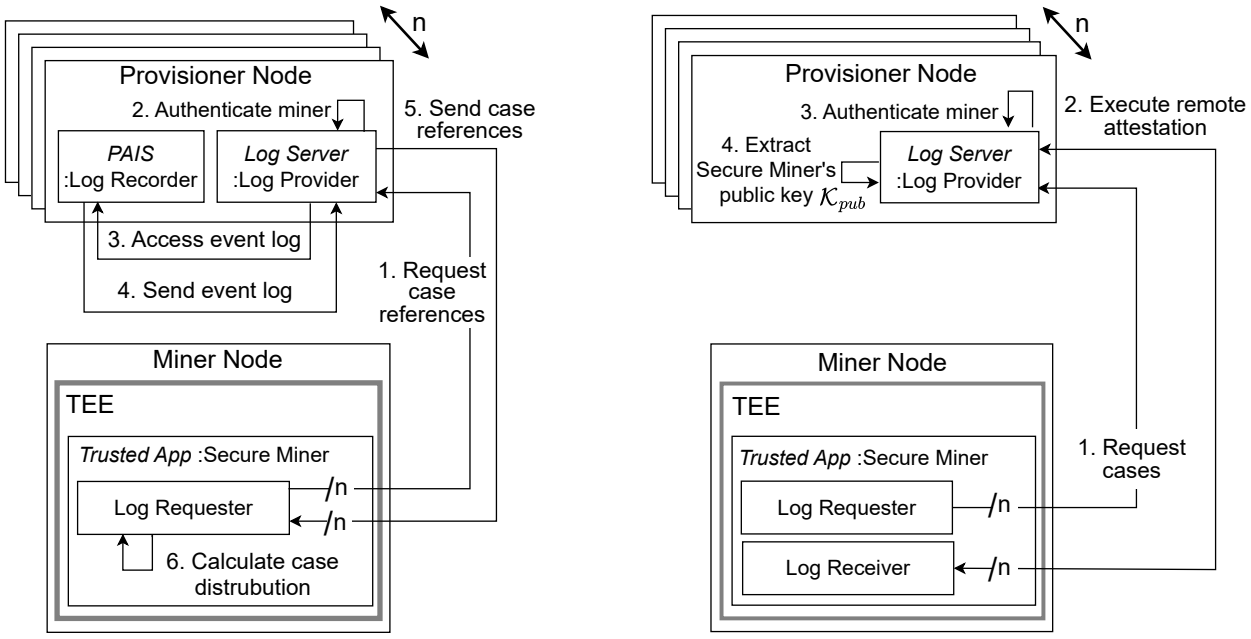
We orchestrate the interaction of the components in CONFINE via a protocol. We separate it in four subsequent stages, namely *(i) initialization*, *(ii) remote attestation*, *(iii) data transmission*, and *(iv) computation*. The initialization phase aims to exchange preliminary metadata, enabling the `Secure Miner` to acknowledge the distribution of cases among the participating `Provisioners`. During the remote attestation phase, the `Provisioners` verify the security properties of the `Secure Miner` prior to the externalization of their log partitions. In the subsequent data transmission phase, the `Provisioners` transfer their event logs in smaller batches of cases according to a segmentation strategy. Finally, the `Secure Miner` executes the selected process mining algorithm over the merged cases received from the `Provisioners`.

In Fig. 5, we associate these phases with the sections, figures, and algorithms explicitly addressing them throughout the text. The protocol's phases are individually 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 $\mathcal{M}$ (which we refer to as CNF$_{\mathcal{M}}$) and any $\mathcal{P}_i \in \widehat{\mathcal{P}}$ (i.e., CNF$_{\mathcal{P}}$) are outlined in Algs. 1 and 3, respectively. These specifications adhere to the syntax for distributed algorithms detailed in [49].[7] We assume that communication between `Secure Miners` and `Log Provisioners` occurs through an *Authenticated Point-to-Point Perfect Link* (**Assumption A8**) [49]. 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),

[7] 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.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381



(a) Initialization

(b) Remote attestation

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

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). In addition, we assume that `Provisioners` implement an access control policy that authorizes or denies requests issued by `Secure Miners` based on the identity of the requesting organization (**Assumption A9**) [50].

$\textsc{Cnf}_{\mathcal{M}}$ accepts as input the list of `Provisioners`' references $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$, namely, descriptors of all the necessary information to locate and identify provisioners. In the same specification Alg. 1, the `Secure Miner` is provided with the following hyperparameters: the *do_yield_cases* flag that specifies whether the computation is incrementally yielded on single complete cases or on the entire event log, and a segment size *seg_size* employed for the generation of the log *segments* (formally introduced in Def. 4.6) during the *data transmission* phase. We posit that the segment size range spans from the largest case (see Def. 4.7) size among all the merged cases, as the minimum value, to the maximum memory capacity of the `TEE`, as the maximum value (**Assumption A10**). Referring to our example, in which the merged Alice's case (comprehensive of the all the three organizations' events) constitutes the largest case with a cumulative size of 100 KB, and considering the capacity of the `TEE` to be 128,000 KB, it follows that the segment size employed by the miner consistently falls between these two values.

Similarly, the `Provisioner`'s specification $\textsc{Cnf}_{\mathcal{P}}$, documented in Alg. 3, 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 manage two or more events/messages concurrently. For the sake of clarity, we omit any explicit representation of this feature in the pseudo-codes under discussion. Furthermore, to preserve the comprehensibility of the components' behavior, we leave out of the proposed pseudo-codes the validity checks and the encryption operations that are addressed in the text and in Figs. 6(a), 6(b), 7(a) and 7(b). Also, we leave out of Alg. 1 the behaviour of the `Secure Miner` $\mathcal{M}$ during the computation phase, as our approach is designed to be integrable with generic process mining algorithms (see Def. 4.3).

For the sake of readability, we provide in Tab. 4 an overview of the ephemeral keys generated by the CONFINE components and referred to in the remainder of this section. In the following, we describe each protocol phase in detail.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

Algorithm 1: $\textsc{Cnf}_{\mathcal{M}}$, Secure Miner's behavior in CONFINE.

Input: $\widehat{\mathcal{P}} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, the (references to) n log provisioners.
Hyperparameters: *seg_size*, the maximum size of the log segment to be transmitted by the log provisioners;
do_yield_cases, indicating whether single cases (if assigned with *true*) or only the full log at the end (if *false*)) should be yielded.
Data: $IIDMap : \widehat{\text{IID}} \to 2^{\widehat{\mathcal{P}}}$, a map from case references $iid \in \widehat{\text{IID}}$ to the set of log provisioners in $\widehat{\mathcal{P}}$;
$PMap : \widehat{\mathcal{P}} \to 2^{\widehat{\text{IID}}}$, a map from log provisioners $\mathcal{P} \in \widehat{\mathcal{P}}$ to the set of references to their cases in $\widehat{\text{IID}}$;
$Cases : \widehat{\text{IID}} \to \hat{\mathbb{C}}$, a map from case references $iid \in \widehat{\text{IID}}$ to cases in $\hat{\mathbb{C}}$.
Implements: `SecureMiner`, **instance** $\mathcal{M}$.
Uses: *AuthenticatedPerfectPointToPointLink*, **instance** *al*.

1 **upon event** $\langle \mathcal{M}, \textsc{Init} \mid \widehat{\mathcal{P}}, seg_size, do_yield_cases \rangle$ **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. 3, ln. 1)
4 **upon** $|\widehat{\mathcal{P}}| = |\text{dom}(PMap)|$ **do** // Once all Provisioners have answered with their case references
5 **foreach** $\mathcal{P} \in \widehat{\mathcal{P}}$ **do trigger** $\langle al, \textsc{Send} \gg \mathcal{P} \mid \textsc{CasesReq}, seg_size, PMap[\mathcal{P}] \rangle$ // Request their cases via *al* (see Alg. 3, ln. 5)
6 **upon event** $\langle al, \textsc{Deliver} \ll \mathcal{P} \mid \textsc{CasesRefRes}, IIDs \rangle$ **such that** $\mathcal{P} \in \widehat{\mathcal{P}}$ **do** // $\mathcal{M}$'s Log Requester gets $\mathcal{P}$'s case references via *al* (Alg. 3, ln. 4)
7 **foreach** $iid \in IIDs$ **do** // For every received case reference *iid* in *IIDs*
8 $IIDMap[iid] \leftarrow IIDMap[iid] \cup \{\mathcal{P}\}$ // Add $\mathcal{P}$ to the set of provisioners for case *iid* in *IIDMap*
9 $PMap[\mathcal{P}] \leftarrow PMap[\mathcal{P}] \cup IIDs$ // Register the references of the cases provided by $\mathcal{P}$ in *PMap*
10 **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 segment $S = \left(\hat{\mathrm{E}}^{\sim}_{S}, \prec_S\right)$ from $\mathcal{P}$ (Alg. 3, ln. 11)
11 **foreach** $e_S \in \hat{\mathrm{E}}^{\sim}_{S}$ **do** // For every e_S in the delivered segment S, each associated with an *iid* – see the *data transmission* phase in Fig. 7(a)
12 **if** $\text{iid}(e_S) \in PMap[\mathcal{P}]$ **then** // If $\mathcal{P}$ has declared the ownership of *iid* (see ln. 6)
13 $PMap[\mathcal{P}] \leftarrow PMap[\mathcal{P}] \setminus \{iid\}$ // Remove *iid* from the set of case references to be provided by $\mathcal{P}$
14 $IIDMap[iid] \leftarrow IIDMap[iid] \setminus \{\mathcal{P}\}$ // Remove $\mathcal{P}$ from the set of *iid* provisioners
15 $C^{\mathcal{P}}_{iid} = \left(\hat{\mathrm{E}}^{\sim}_{C_{\mathcal{P}}}, \prec_{C_{\mathcal{P}}}\right) \leftarrow \mathcal{C}\left(\left(\hat{\mathrm{E}}^{\sim}_{S}, \prec_S\right), iid\right)$ // Collect in $C^{\mathcal{P}}_{iid}$ the case extracted from the partition with case reference *iid* (see Def. 4.7)
16 $Cases[iid] \leftarrow \mathcal{M}.\texttt{LogManager.merge}\left(Cases[iid], C^{\mathcal{P}}_{iid}\right)$ // Merge the case and store the result in *Cases* – see Alg. 2
17 $\hat{\mathrm{E}}^{\sim}_{S} \leftarrow \hat{\mathrm{E}}^{\sim}_{S} \setminus \hat{\mathrm{E}}^{\sim}_{C_{\mathcal{P}}}$; $\prec_S \leftarrow \prec_S \setminus \prec_{C_{\mathcal{P}}}$ // Remove the events and order relation's tuples of case $C^{\mathcal{P}}_{iid}$ from segment S
18 **upon** $IIDMap[iid] = \emptyset$ for some $iid \in \text{dom}(IIDMap)$ **do** // When all the provisioners have delivered some *iid* to $\mathcal{M}$'s Log Manager
19 $\text{dom}(IIDMap) \leftarrow \text{dom}(IIDMap) \setminus \{iid\}$ // Remove *iid* from the domain of cases which still needs to be processed
20 **if** *do_yield_cases* **then** // If *do_yield_cases* flag is set to true
21
22 **yield** $Cases[iid]$ **to** $\mathcal{M}$.LogProcessor // Forward the case *iid* to the Log Processor of $\mathcal{M}$ for mining – *computation* phase in Fig. 7(b)
23 **else** // If *do_yield_cases* flag is set to false
24 **if** $\text{dom}(IIDMap) = \{\}$ **then** // If all the cases have been collected
25 $\mathrm{L} \leftarrow (\{\}, \{\})$ // Initialize an empty event log L
26 **foreach** $C \in Cases$ **do** // For each complete case C stored in *Cases*. . .
27 $\mathrm{L} \leftarrow \mathcal{M}.\texttt{LogManager.merge}\,(\mathrm{L}, C)$ // Add the case C to the event log L via merge
28 **yield** L **to** $\mathcal{M}$.LogProcessor // Forward the event log L to the Log Processor of $\mathcal{M}$ for mining – *computation* phase in Fig. 7(b)

6.2.1. Initialization

The protocol starts with the initialization stage whose objective is to inform the `Secure Miner` about the distribution of cases in the log partitions (see Def. 4.5) of the `Provisioners`. At the onset of this stage, the `Log Requester` within the `Secure Miner` issues n requests, one per `Log Server` component, to retrieve the list of case references they record (step 1 in Fig. 6(a) and (Alg. 1, ln. 3). This data request contains an identity proof associated with the miner organization, which the `Provisioners` evaluate in the context of their authorization policies over their log partitions. Following miner authentication (step 2), each `Log Server` retrieves the local log partition from the `PAIS` (steps 3 and 4) and subsequently responds to the `Log Requester` by providing the list of iids *IIDs* within the owned log partition (step 5 and Alg. 3, ln. 4). After collecting these n responses (Alg. 1, ln. 4), the `Log Requester` delineates the distribution of *IIDs*. In our motivating scenario, by the conclusion of the initialization, the `Secure Miner` gains knowledge that Bob's case, with *iid* 711 and synthesized in the traces T^H_{711} and T^C_{711}, is exclusively retained by the hospital and the specialized clinic. In contrast, the trace of Alice's case having *iid* 312, denoted as T^H_{312}, T^C_{312}, and T^S_{312}, is scattered across all three organizations.

In this phase, we solely rely on standard authentication of the `Secure Miner`'s organization since obtaining the list of case identifiers (*IIDs*) typically does not reveal log partitions' sensitive data (it might be a random number or a hashcode as well, for instance). Our design aims to perform the TEE attestation when the `Secure Miner` begins processing sensitive data, hence in the subsequent phase.

6.2.2. Remote attestation

The remote attestation stage has a threefold objective: *(i)* to furnish provisioners with compelling evidence that the data request for a log partition originates from a `Trusted App` running within a TEE; *(ii)* to confirm

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

Algorithm 2: The `merge` function employed by the Secure Miner in CNF$_{\mathcal{M}}$

Input: $\mathrm{L}' = \left(\hat{\mathrm{E}}', \preceq'\right)$, an event log; $\mathrm{L}'' = \left(\hat{\mathrm{E}}'', \preceq''\right)$, another event log.
Output: The merge of L' and L''

Function merge$\left(\mathrm{L}', \mathrm{L}''\right)$ **is**
  $\hat{\mathrm{E}} \leftarrow \left(\hat{\mathrm{E}}' \cup \hat{\mathrm{E}}''\right)$; $\preceq \leftarrow \left(\preceq' \cup \preceq''\right)$; $\mathrm{L} \leftarrow \left(\hat{\mathrm{E}}, \preceq\right)$
  foreach $e'' \in \hat{\mathrm{E}}''$ **do** // For each event in L''
    $\preceq \leftarrow \preceq \cup \left\{\left(e', e''\right) : e' \in \hat{\mathrm{E}}' \text{ and } \mathrm{tms}\left(e'\right) \leqslant \mathrm{tms}\left(e''\right)\right\}$ // For every e' in $\hat{\mathrm{E}}'$ s.t. $\mathrm{tms}\left(e'\right) \leqslant \mathrm{tms}\left(e''\right)$ add to $\preceq$ that e' precedes e''
    $\preceq \leftarrow \preceq \cup \left\{\left(e'', e'\right) : e'' \in \hat{\mathrm{E}} \text{ and } \mathrm{tms}\left(e''\right) \leqslant \mathrm{tms}\left(e'\right)\right\}$ // For every e' in $\hat{\mathrm{E}}'$ s.t. $\mathrm{tms}\left(e''\right) \leqslant \mathrm{tms}\left(e'\right)$ add to $\preceq$ that e'' precedes e'
  return L

Algorithm 3: CNF$_{\mathcal{P}}$, Provisioner's behavior in CONFINE.

Input: $\widehat{\mathcal{M}} = \{\mathcal{M}_1, \ldots, \mathcal{M}_s\}$, the (references to) s miners.
Hyperparameters: /
Implements: `Provisioner`, **instance** $\mathcal{P}$.
Uses: *AuthenticatedPerfectPointToPointLink*, **instance** *al*.

upon event $\langle al,$ Deliver $\ll \mathcal{M} \mid$ CasesRefsReq$\rangle$ **such that** $\mathcal{M} \in \widehat{\mathcal{M}}$ **do** // $\mathcal{P}$ receives the request for case references from $\mathcal{M}$ (see Alg. 1, ln. 3)
  $\left(\hat{\mathrm{E}}^{\sim}_{\mathcal{P}}, \prec_{\mathcal{P}}\right) \leftarrow \mathcal{P}$.`LogRecorder.accessEventLog()` // Access the log partition via `Log Recorder`
  $IIDs \leftarrow \left\{\mathrm{iib}(e'_{\mathcal{P}}) : e'_{\mathcal{P}} \in \hat{\mathrm{E}}^{\sim}_{\mathcal{P}}\right\}$ // Get the set of case references in the log partition
  trigger $\langle al,$ Send $\gg \mathcal{M} \mid$ CasesRefRes, $IIDs\rangle$ // Send the case references to $\mathcal{M}$ (see Alg. 1, ln. 6)

upon event $\langle al,$ Deliver $\ll \mathcal{M} \mid$ CasesReq, seg_size, $IIDs\rangle$ **such that** $\mathcal{M} \in \widehat{\mathcal{M}}$ **do** // $\mathcal{P}$ gets the case request from $\mathcal{M}$ (see Alg. 1, ln. 5)
  $rep \leftarrow \mathcal{M}$.`LogReceiver.getAttestationReport(`$\mathcal{P}$`)` // Get the attestation report of $\mathcal{M}$ – *remote attestation* phase in Fig. 6(b)
  if $\mathcal{P}$.`LogProvider.executeRemoteAttestation(`rep`)` is successful **then** // Execute the remote attestation of the report via `Log Provider`
    $\left(\hat{\mathrm{E}}^{\sim}_{\mathcal{P}}, \prec_{\mathcal{P}}\right) \leftarrow \mathcal{P}$.`LogRecorder.accessEventLog()` // Access the log partition via `Log Recorder`
    $\{S_1, \ldots, S_m\} \leftarrow \mathcal{P}$.`LogProvider.segmentEventLog`$\left(\left(\hat{\mathrm{E}}^{\sim}_{\mathcal{P}}, \prec_{\mathcal{P}}\right), IIDs, seg_size\right)$ // Segment the event log – see Alg. 4
    foreach $i \in \{1, \ldots, m\}$ **do** // For every segment S_i
      trigger $\langle al,$ Send $\gg \mathcal{M} \mid$ CasesRes, $S_i\rangle$ // Send the segment S_i to $\mathcal{M}$ (see Alg. 1, ln. 10) – *data transmission* phase in Fig. 7(a)

the specific nature of the `Trusted App` as an authentic `Secure Miner` software entity; *(iii)* to authorize the organization of the `Secure Miner` application.

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. 1, ln. 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 Server`s initiates a `TEE` remote attestation with the `Log Receiver` within the `Secure Miner` (step 2), as outlined in Sect. 2.1.2. During this procedure, the `Log Receiver` leverages the customizable data field of the attestation evidence (the inclusion of custom data is discussed in Sect. 2.1.2) to embed: *(i)* an identity proof of the `Secure Miner`'s organization, and *(ii)* an ephemeral public key $\mathcal{K}_{\text{pub}}$, generated at runtime by the `Log Receiver` for the current protocol session and paired with a private key $\mathcal{K}_{\text{priv}}$, which remains sealed within the `TEE`.

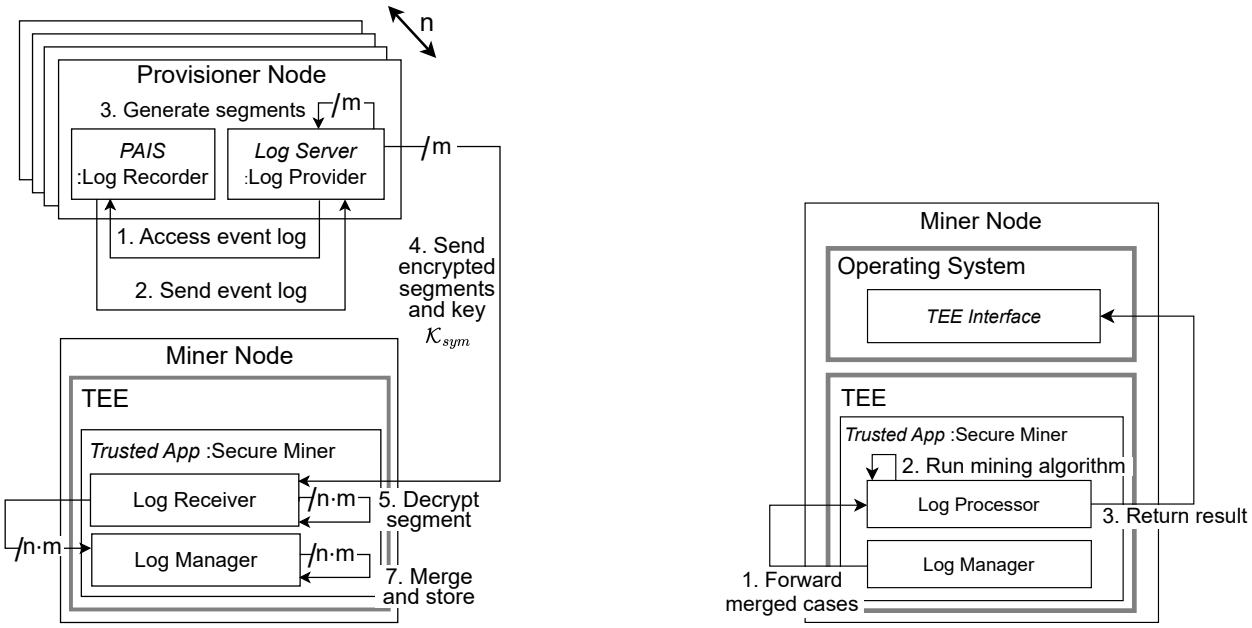
Upon successful completion of the remote attestation and verification of the `Secure Miner`'s workload, the n `Log Server`s inspect the content of the custom data field of the validated attestation evidence. Using the embedded identity evidence, they authenticate the miner organization and, based on this assessment,

Algorithm 4: The `segmentEventLog` function of the Provisioner in CNF$_{\mathcal{P}}$.

Input: $\mathrm{L}_{\mathcal{P}} = \left(\hat{\mathrm{E}}^{\sim}_{\mathcal{P}}, \prec_{\mathcal{P}}\right)$, the log partition of provisioner $\mathcal{P}$; *IIDs*, a set of case references; *seg_size*, a segment size.
Output: $\{S_1 \ldots S_m\}$ a set of m segments

Function segmentEventLog($\mathrm{L}_{\mathcal{P}}$, *IIDs*, *seg_size*) **is**
  $SegSet \leftarrow \{\}$ // Initialize the segment set to return
  $\hat{\mathrm{E}}^{\sim}_{S} \leftarrow \{\}$; $\prec_S \leftarrow \{\}$; $S \leftarrow \left(\hat{\mathrm{E}}^{\sim}_{S}, \prec_S\right)$ // Initialize an empty segment
  foreach *iid* $\in$ *IIDs* **do** // For each case in the log partition
    $C^{\mathcal{P}}_{iid} = \left(\hat{\mathrm{E}}^{\sim}_{C}, \prec_C\right) \leftarrow \mathcal{C}\left(\left(\hat{\mathrm{E}}^{\sim}_{\mathcal{P}}, \prec_{\mathcal{P}}\right), iid\right)$ // Collect in $C^{\mathcal{P}}_{iid}$ the case extracted from the partition with case reference *iid*
    if `SizeOf`(S) + `SizeOf`$\left(C^{\mathcal{P}}_{iid}\right) >$ *seg_size* **then** // If $C^{\mathcal{P}}_{iid}$ do not fit the remaining space of the segment $\hat{\mathrm{E}}^{\sim}_{S}$
      $SegSet \leftarrow SegSet \cup \{S\}$ // Put the segment $\left(\hat{\mathrm{E}}^{\sim}_{S}, \prec_S\right)$ in *SegSet*
      $\hat{\mathrm{E}}^{\sim}_{S} \leftarrow \{\}$; $\prec_S \leftarrow \{\}$; $S \leftarrow \left(\hat{\mathrm{E}}^{\sim}_{S}, \prec_S\right)$ // Empty the segment
    $\prec_S \leftarrow \prec_S \cup \prec_C$ // Add the case's order rel. to the current segment
    $\prec_S \leftarrow \prec_S \cup \left(\prec_{\mathcal{P}} \cap \left(\hat{\mathrm{E}}^{\sim}_{C} \times \hat{\mathrm{E}}^{\sim}_{S} \cup \hat{\mathrm{E}}^{\sim}_{S} \times \hat{\mathrm{E}}^{\sim}_{C}\right)\right)$ // Add the order rel. from the case's events to the current segment's ones and vice-versa
    $\hat{\mathrm{E}}^{\sim}_{S} \leftarrow \hat{\mathrm{E}}^{\sim}_{S} \cup \hat{\mathrm{E}}^{\sim}_{C}$ // Add the case's events to the current segment

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381



(a) Data transmission

(b) Computation

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

render a final trust decision (step 3). If authentication succeeds, the `Log Servers` extract $\mathcal{K}_{\text{pub}}$ from the validated evidence (step 4), after which the protocol proceeds to the data transmission phase.

6.2.3. Data transmission

Once the trusted nature of the `Secure Miner` is verified, the `Log Servers` proceed with the transmission of their cases. To accomplish this, each `Log Server` retrieves the log partition from the `PAIS` (steps 1 and 2 of Fig. 7(a)), and filters it according to the *IIDs* specified by the miner. Given the constrained workload capacity of the `TEE`, it is imperative for `Log Servers` to split the filtered log partition into distinct segments (as per Def. 4.6). Consequently, each `Log Server` generates m log segments comprising a variable count of entire cases (step 3 and Alg. 3, ln. 9). The cumulative size of these segments is governed by the *seg_size* parameter specified by the miner in the initial request (as in step 1 of Fig. 6(b)). In our motivating scenario, the `Log Server` of the hospital may structure the segmentation such that T^H_{312} and T^H_{711} reside within the same segment, whereas the specialized clinic might have T^S_{312} and T^S_{711} in separate segments. In Alg. 4, we propose the specification of one of the possible segmentation strategies. Our pseudocode takes as input a log partition $\mathrm{L}_{\mathcal{P}}$, a set of instance identifiers *IIDs*, and a segment size *seg_size* and it produces a set of m segments S of $\mathrm{L}_{\mathcal{P}}$. After the segmentation, the n `Log Servers` generate a random symmetric encryption key, which we call $\mathcal{K}_{\text{sym}}$ (see Tab. 4). Therefore, each `Log Server` encrypts its m segments with $\mathcal{K}_{\text{sym}}$ and transmits the encrypted data to the `Log Receiver` (step 4 and Alg. 3, ln. 11). Each transmission includes a cryptographic proof of the `Provisioner`'s identity, computed over the forwarded segments, along with the ciphertext obtained by encrypting $\mathcal{K}_{\text{sym}}$ with the public key $\mathcal{K}_{\text{pub}}$ found within the attestation evidence

Table 4: Ephemeral encryption keys generated during the a session of the CONFINE protocol

Key name	Generated by	Objective
$\mathcal{K}_{\text{pub}}$	`Secure Miner`	Used by the `Provisioners` to encrypt the key $\mathcal{K}_{\text{sym}}$ during the segment transmission
$\mathcal{K}_{\text{priv}}$		Used by the `Secure Miner` to decrypt the key $\mathcal{K}_{\text{sym}}$ within the `TEE`
$\mathcal{K}_{\text{sym}}$	`Provisioner`	Used by the `Provisioner` to encrypt segments transmitted to the `Secure Miner`

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

of the previous phase. Subsequently, the `Log Receiver` collects the $n \times m$ transmissions in a queue and processes them sequentially. It first uses $\mathcal{K}_{\text{priv}}$ to decrypt $\mathcal{K}_{\text{sym}}$, then uses the latter to decrypt the received segment (step 5). Subsequently, the `Log Receiver` forwards the cases contained in the segment to the `Log Manager` (step 6 and Alg. 1, ln. 16). To reconstruct the process instance, cases with the same *iid* 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^H_{312}, T^S_{312} and T^C_{312} for Alice case) following the definition of *safe merge* in Def. 4.8. In Alg. 2, we outline the pseudocode of the merge operation. This specification takes as input two event logs L' and L'' and returns the merged event log $\mathrm{L} = \mathrm{L}' \bigoplus \mathrm{L}''$. 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 [51]). In our illustrative scenario, the merging schema to combine the cases of Alice is contingent upon the linkage established through their *iid* attributes, namely, *'Case'* (in the hospital's log partition), *'HospitalCaseID'* (in the pharmaceutical company's log partition), and *'TreatmentID'* (in the specialized clinic's log partition). The outcomes arising from merging the cases within the processed segment are securely stored by the `Log Manager` in the `TEE`, until the subsequent computation phase.

6.2.4. Computation

The `Secure Miner` requires all the `Provisioner`s to have delivered cases referring to the same *iid* value (as per Post. 4.1). 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, ln. 18), the `Log Manager` forwards the case earmarked for computation to the `Log Processor` (step 1 in Fig. 7(b) and Alg. 1, ln. 22, Alg. 1, ln. 28). This batch of cases may correspond either to the entire merged event log or to a subset thereof. In Alg. 1, we describe both the complete event log and the single case computation settings (Alg. 1, ln. 23 and Alg. 1, ln. 20, respectively). The former entails a single computation routine, thus saving computation time but requiring a larger memory buffer in the `TEE`, whereas the latter necessitates multiple consecutive elaborations with a lower demand for space. Subsequently, the `Log Processor` proceeds to input the merged case/log, sorted according to their timestamp attributes, as per Post. 4.2, into the process mining algorithm $\underset{\hbar_1,\ldots,\hbar_k}{prom}$ (step 2), according to Def. 4.3. Notice that the above choice on the buffering of cases impacts the selection of $\underset{\hbar_1,\ldots,\hbar_k}{prom}$. If we elaborate subsequent batches, each containing a part of all merged cases, $\underset{\hbar_1,\ldots,\hbar_k}{prom}$ must support incremental processing, enriching the output as new batches come along. An example of this class of algorithms is the HeuristicsMiner [24]. Otherwise, incrementality is not required. Ultimately, the outcome of the computation is relayed by the `Log Processor` from the `TEE` to the `TEE Interface` running atop the `Operating System` of the `Miner Node` (step 3). The CONFINE protocol does not impose restrictions on the post-computational handling of results, once they leave the `TEE`. 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 event log 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. System Guarantees

Our solution provides security guarantees that are built upon an architecture leveraging TEEs as its core, compounded with a protocol we specifically devise for data provisioning and treatment. In Sect. 7.1, we analyze how data secrecy threats are mitigated by the integration of TEE's features in our architecture. For the latter, we conduct a formal analysis of our protocol for segmentation and merge of event data stemming from heterogeneous sources in Sect. 7.2, Specifically, we focus on the safety and liveness of CONFINE in its data transmission phase by formal verification of the underpinning algorithms' soundness and completeness.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

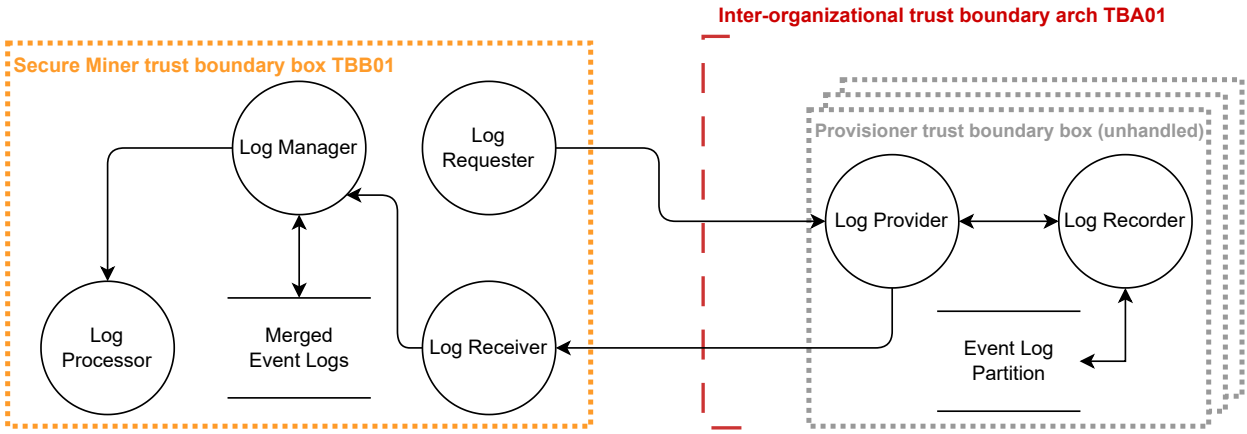


Figure 8: Data flow diagram and trust boundaries of our architecture

7.1. TEE-based Security Risk Mitigation

Here we pursue a twofold objective: *(i)* to provide a clear rationale for the integration of the `TEE` within our architecture, and *(ii)* to examine the security guarantees that the `TEE`-related mechanisms offer in the context of our solution. To this end, we build upon the STRIDE framework for the definition of a set of data-secrecy threats. Thereafter, we clarify how the `TEE`-based measures counteract these adversarial scenarios.

In Fig. 8, we present the Data Flow Diagram (DFD) which captures the components of the CONFINE architecture and the information exchange between them. Through this representation, we demarcate the *trust boundaries* among the CONFINE components. In our DFD, a trust boundary box (the orange dotted square) groups entities with mutual trust. Differently, trust boundary arches (red dashed lines) mark the transitions of trust level on the occasion of data exchanges (the black arrows) between untrusted parties. We identify three trust boundaries: the `Secure Miner`'s trust boundary box (**TBB01**), the inter-organizational trust boundary arch (**TBA01**), and the `Provisioner`'s trust boundary box.

In Tab. 5, we summarize the identified threats, indicating for each the trust boundary it targets and its corresponding STRIDE category: *spoofing*, *tampering*, *repudiation*, *information disclosure*, *denial of service*, and *elevation of privileges*. In our examination, we exclude the security considerations pertaining to `Provisioner`'s trust boundary boxes (the gray dotted square in Fig. 8), as menaces targeting event logs in their native organizational domain fall beyond the scope of this study. In what follows, we examine how the `TEE` mechanisms embedded in our architecture mitigate each of these threats.

(T01) The attacker accesses the Secure Miner's memory location to leak the event logs. As introduced in Sect. 6.1, the `TEE` profile we consider in CONFINE provides hardware-based memory encryption. Therefore, data and code belonging to the `Secure Miner` trusted app are managed by the CPU in a hardware-encrypted region of the main memory [32]. The cryptographic key for decrypting this data randomly changes each time the machine undergoes a power cycle, and it is ingrained within the hardware of the CPU. Through a hardware-based access control mechanism, the processor interacts with reserved event log data solely through the execution of designated instructions delineated in the source code of the `Secure Miner`. In the

Table 5: TEE-related threats in CONFINE

ID	Trust boundary	Type	Threat description
T01	TBB01	Information disclosure	The attacker accesses the `Secure Miner`'s memory location to leak the event logs
T02		Tampering	The attacker meddles with the source code of the `Secure Miner` or its event log data
T03		Elevation of privileges	The attacker gains the rights to run in the same environment of the `Secure Miner`
T04	TBA01	Spoofing	The attacker impersonates a `Secure Miner` to gain access to the `Provisioner`'s log
T05		Information disclosure	The attacker sniffs the data exchanged between the `Provisioner` and the `Secure Miner`

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

event of a physical leakage attack, where attackers attempt to compromise sensitive information by gaining direct access to this private storage, they would only acquire ciphered data.

(T02) The attacker meddles with the source code of the Secure Miner or its event log data. Log partitions (defined in Def. 4.5) residing within the `TEE` can solely be manipulated by the source code of our `Secure Miner` trusted app. Any unauthorized modifications to the `Secure Miner`'s code after its deployment in memory would result in an altered measurement (i.e., the hash identifier for a trusted app source code). The processor possesses the capability to detect such alterations by comparing the measurement calculated on the tampered `Trusted App` with the one computed during the deployment phase. Through this mechanism, the `TEE` can forestall any malevolent modifications to event logs.

(T03) The attacker gains the rights to run in the same environment of the Secure Miner. In our approach, we consider application-level `TEE`s. Therefore, each `Secure Miner` application operates within its own dedicated `TEE` instance. This implies that reserved data can uniquely be accessed by a specific source code (in our case, the one of the `Secure Miner`). Any other trusted application software running on the same device is allocated a separate `TEE` instance, effectively rendering privilege escalation attacks infeasible.

(T04) The attacker impersonates a Secure Miner to gain access to the Provisioner's log. In CONFINE, we consider attestable `TEE`s. Therefore, during the attestation phase of our protocol, `Provisioner`s engage in the assessment of a hardware-signed attestation evidence to determine the authenticity of the sender, which purportedly represents a `Secure Miner` software entity operating within a `TEE` provided by an authorized organization. Depending on the specific TEE technology, `Provisioner`s may enlist an endorser [52], serving as the fundamental authority (i.e., the CPU manufacturer in the majority of TEE instances). This authority is tasked with validating the report by confirming its signature with a valid attestation key. Attestation keys are securely embedded within the hardware and cannot be accessed by any entity other than the processor itself. Once the `Provisioner` validates the report, it scrutinizes the measurement, which must correspond to the reference value associated with the `Secure Miner`'s source code. In case an unknown trusted app triggers the report generation, the CPU computes a software measurement that differs from the `Secure Miner`'s reference value, thus yielding a failure in the attestation process.

(T05) The attacker sniffs the Provisioner's log sent to the Secure Miner. By leveraging attestation, the `TEE` enables the creation of a secure communication channel in which transmitted log partitions can only be decrypted inside the `TEE` itself. Following the successful attestation of the `Secure Miner`'s workload, the `Provisioner`s extract from the customizable section of the attestation evidence a public key that is randomly generated by the `Secure Miner` and whose corresponding private key is securely sealed within the `TEE`. This private key is inaccessible to any external actors, including users with administrative privileges over the mining machine. Using this key material, the `Provisioner` and the `Secure Miner` can establish the encrypted communication channel. Within this channel, segments are encrypted using an ephemeral symmetric key selected by the `Provisioner`. The symmetric key is transmitted in encrypted form using the aforementioned public key, together with the encrypted segments. Decryption of the symmetric key is only possible using the sealed private key within the `TEE`. Consequently, any adversary intercepting the transmission can access only ciphertext, thereby preserving the confidentiality of the transmitted information.

7.2. Soundness and Completeness

Hereafter, we formally prove the safety and liveness of our system [53] in the execution of its data transmission phase by verifying soundness and completeness of the underlying algorithms. With the former, we ensure that if the protocol terminates, it returns a correct result. With the latter, we guarantee that the a result is always eventually returned. Specifically, our objective is to prove that the ensemble of $\textsc{Cnf}_{\mathcal{M}}$ and $\textsc{Cnf}_{\mathcal{P}}$ (Algs. 1 and 3, respectively) always produces a safe merge as per Def. 4.8 under the assumption of authenticated perfect point-to-point link used as a communication channel and honest `Provisioner`s. First, we prove soundness and completeness for our realization of the pivotal `merge` function (Alg. 2). Thereupon, we prove the general statements pertaining to the protocol. Finally, we build upon the previous statements to prove the result's convergence with respect to a generic process mining algorithm (as per Def. 4.3).

To begin with, we prove that our realization of the `merge` function (Alg. 2) is *safe*.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

Lemma 7.1 (Safety of `merge`). *Let* $\mathrm{L}' = \left(\widehat{\mathrm{E}}', \preceq'\right)$ *and* $\mathrm{L}'' = \left(\widehat{\mathrm{E}}'', \preceq''\right)$ *be two event logs as per Def. 4.2 whose total orders* $\preceq'$ *and* $\preceq''$ *are induced by the timestamp attribute as per Post. 4.2. The output of* `merge` $(\mathrm{L}', \mathrm{L}'')$ *is always* $\mathrm{L} = (\widehat{\mathrm{E}}, \preceq) = \mathrm{L}' \bigoplus \mathrm{L}''$ *wherein* $\bigoplus$ *is safe as per Def. 4.8.*

Proof. (Base case) Let L' and L'' consist of one event each: $\widehat{\mathrm{E}}' = \{e'\}$ (hence $e' \preceq' e'$), $\widehat{\mathrm{E}}'' = \{e''\}$ (hence $e'' \preceq'' e''$). Alg. 2, ln. 2 initializes $\widehat{\mathrm{E}}$ as $\{e', e''\}$ and $\preceq$ as $\{(e', e'), (e'', e'')\}$. It remains to establish whether $e' \preceq e''$ or vice-versa. The execution of Alg. 2, ln. 4 and Alg. 2, ln. 5 verify the statement by direct application of the isomorphism postulated in Post. 4.2 since it employs the events' timestamps as the criterion for establishing the order. *(Inductive case)* Let $\widehat{\mathrm{E}}'$ and $\widehat{\mathrm{E}}''$ be of cardinality $k' > 1$ and $k'' > 1$, respectively. Let $e^{(k)} \in \widehat{\mathrm{E}}''$ be the event fetched by the cycle on Alg. 2, ln. 3 at the k-th iteration, with $1 < k \leqslant k''$. As per Post. 4.2, the isomorphism is guaranteed by Alg. 2, ln. 4 and Alg. 2, ln. 5. Totality can only be violated if it was not preserved at the $(k-1)$-th iteration, which contradicts the induction over the base case. □

Equipped with this result underpinning the basic operation backing the merging of event log partitions stemming from diverse `Provisioner`s, we can proceed to prove the general statement confirming the soundness and completeness of our protocol.

Theorem 7.1 (Soundness and completeness). *Let* $\mathcal{M}$ *be a* `Secure Miner` *following* $\textsc{Cnf}_{\mathcal{M}}$ *(Alg. 1) with input references to* `Provisioners` $\mathcal{P}_1, \ldots, \mathcal{P}_n$*, whereby the latter are honest, follow* $\textsc{Cnf}_{\mathcal{P}}$ *(described in Alg. 3), and provide log partitions* $\mathrm{L}_{\mathcal{P},1}, \ldots, \mathrm{L}_{\mathcal{P},n}$ *with* $n \in \mathbb{N}$*. Upon the completion of the data transmission phase,* $\mathcal{M}$ *always (completeness) yields an event log* $\mathrm{L} = \mathrm{L}_{\mathcal{P},1} \bigoplus \cdots \bigoplus \mathrm{L}_{\mathcal{P},n}$ *wherein* $\bigoplus$ *is safe (soundness).*

Proof sketch. The communication channel (henceforth *al*) is an Authenticated Point-to-point Perfect Link [49]. For the sake of brevity, we begin this proof sketch with the `Log Manager`'s assemblage of cases (Alg. 1, ln. 16) following `Log Receiver`'s receipt of segments (see Alg. 1, ln. 10 and Alg. 3, ln. 11). We leave out the analysis of the preceding interactions (i.e., the initialization and remote attestation phases, the subsequent gathering of case references and polishing of internal data structures), since their soundness and completeness can be verified by inspection of the algorithms (see the code blocks starting at Alg. 1, ln. 1, Alg. 3, ln. 5, Alg. 3, ln. 1, and Alg. 1, ln. 6) given the channel's guarantees, the `Provisioners`' honesty, and the secure miner's authenticity.
(Soundness). Assume that every log partition $\mathcal{P}_i$ consists of exactly one segment, S_i, with $1 \leqslant i \leqslant n$. Let S_i contain (part of) a case identified by *iid*, $C^{\mathcal{P}}_{iid}$, and *Cases*[*iid*] be the current copy of the case identified by *iid* in $\mathcal{M}$. Notice that the building of *Cases*[*iid*] is based on Alg. 1, ln. 15, wherein all events associated to *iid* in $\mathcal{M}$ are extracted and put in a total order relation that restricts the segment's total order to those events. The assemblage of cases is carried out by invoking `merge` (Alg. 2) on *Cases*[*iid*] and $C^{\mathcal{P}}_{iid}$ (Alg. 1, ln. 16). By definition, a case is an event log (see Def. 4.7). Thus, *Cases*[*iid*] and $C^{\mathcal{P}}_{iid}$ can be considered as such, too. Therefore, Lem. 7.1 applies, and the merge of *Cases*[*iid*] and $C^{\mathcal{P}}_{iid}$ is safe. Since a safe merge is associative and commutative (Prop. 4.1), the number of events in the segment and the processing order thereof do not affect the correctness of the above statement. Also, this operation is repeated for every case in the segment as all events in it are visited (Alg. 1, ln. 11) and every case is checked once (as guaranteed by Alg. 1, ln. 12, Alg. 1, ln. 13, and Alg. 1, ln. 14). Therefore, the statement holds regardless of the number of cases in the segment. If a log partition consists of more than one segment, the `Secure Miner` reiterates the same above procedure for every such segment (Alg. 1, ln. 10), and does so in a mutual exclusive fashion, thus impeding faulty over-writings on *Cases* (Alg. 1, ln. 16). The same conclusion can be drawn extending the discourse to a setting with multiple `Provisioners` delivering their own segments. Again, the processing order for segments does not influence the outcome due to the associativity and commutativity of the safe merge (see Prop. 4.1) that the `merge` function realizes (as per Lem. 7.1).
(Completeness). Now, we show that, operating under our assumptions, no segment gets lost during the transmission phase (thus, blocking the protocol execution and undermining the completeness). Assuming every $\mathcal{P}$ is honest, they transmit all and only the segments of which their partitions consist following $\textsc{Cnf}_{\mathcal{P}}$ (Alg. 3). Since the communication channel resorts to an Authenticated Perfect Point-to-Point Perfect Link, all broadcast segments are delivered once, and no package is lost. Therefore, every event in a received

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

Table 6: 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	$\mathscr{O}^P \mapsto 3$, $\mathscr{O}^C \mapsto 5$, $\mathscr{O}^H \mapsto 14$
Sepsis [54]	Real	16	1050	185	3	15	$\mathscr{O}^1 \mapsto 1$, $\mathscr{O}^2 \mapsto 1$, $\mathscr{O}^3 \mapsto 14$
BPIC2013 [55]	Real	7	1487	123	1	9	$\mathscr{O}^1 \mapsto 6$, $\mathscr{O}^2 \mapsto 7$, $\mathscr{O}^3 \mapsto 6$

segment (Alg. 1, ln. 10) is visited by the routine starting at Alg. 1, ln. 11, hence all the cases therein. All merged cases are thus finally yielded with the procedure starting at Alg. 1, ln. 18, thus completing the data transmission phase. □

In light of the above proof, we can finally prove the convergence of CONFINE.

Theorem 7.2 (Convergence)**.** *Let* L *be an event log such that* $\mathrm{L} = \mathrm{L}_{\mathcal{P},1} \bigoplus \cdots \bigoplus \mathrm{L}_{\mathcal{P},n}$ *wherein* $\bigoplus$ *is safe as per Def. 4.8. Let* $\mathrm{L}' = \textsc{Cnf}_{\mathcal{M}}(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ *be the event log yielded as a result of the data transmission phase* ($\textsc{Cnf}_{\mathcal{M}}$) *by a* `Secure Miner` $\mathcal{M}$ *following the protocol in Alg. 1 with references to* `Provisioners` $\mathcal{P}_1, \ldots, \mathcal{P}_n$*, whereby the latter follow the protocol described in Alg. 3 and provide logs* $\mathrm{L}_{\mathcal{P},1}, \ldots, \mathrm{L}_{\mathcal{P},n}$ *with* $n \in \mathbb{N}$*. Let* $\underset{\hbar_1,\ldots,\hbar_k}{\mathit{prom}}$ *be a process mining function as per Def. 4.3. The application of* $\underset{\hbar_1,\ldots,\hbar_k}{\mathit{prom}}$ *on the event log* L *and on* L' *leads to the same result, i.e.,* $\underset{\hbar_1,\ldots,\hbar_k}{\mathit{prom}}(\mathrm{L}) = \underset{\hbar_1,\ldots,\hbar_k}{\mathit{prom}}(\mathrm{L}')$.

Proof sketch. By contradiction, suppose that $\underset{\hbar_1,\ldots,\hbar_k}{\mathit{prom}}(\mathrm{L}) \neq \underset{\hbar_1,\ldots,\hbar_k}{\mathit{prom}}(\mathrm{L}')$. Given that the process mining algorithm $\underset{\hbar_1,\ldots,\hbar_k}{\mathit{prom}}$ is deterministic (as per Def. 4.3) and all its parameters are given, this would imply that L is not correctly reconstructed during the execution of the protocol, i.e., $\textsc{Cnf}_{\mathcal{M}}(\mathcal{P}_1, \ldots, \mathcal{P}_n) \neq \mathrm{L}$. In other words, $\mathcal{M}$ computes $\mathrm{L}_{\mathcal{P},1} \bigoplus \cdots \bigoplus \mathrm{L}_{\mathcal{P},n} = \mathrm{L}'$, with $\mathrm{L}' \neq \mathrm{L}$. We assume that each `Provisioner` $\mathcal{P}_i$ behaves honestly and that the communication channel is an Authenticated Perfect Point-to-Point channel. Therefore, the incorrect reconstruction of L' would contradict the merge safety property demonstrated in Lem. 7.1, and more generally, it would violate the soundness of the protocol as established in Thm. 7.1. □

8. Empirical Evaluation

In this section, we report on our empirical evaluation of CONFINE. First, we introduce our implementation in Sect. 8.1. Then, we propose an experimental verification of Thm. 7.2, empirically demonstrating the output convergence in Sect. 8.2. Finally, we evaluate our implementation with runtime tests on memory usage to show that our approach determines low memory demand, compatible with the constrained capacity of TEEs (Sect. 8.2).

8.1. Implementation

We implemented the `Secure Miner` component as an Intel SGX [56] trusted application, encoded in Go through the EGo framework.[8] We opted for an Intel SGX TEE for our prototype because its features align with the TEE profile outlined in Sect. 2.1.2 and offers a well-documented developer ecosystem. These characteristics made Intel SGX a practical choice for implementing our research prototype aimed at demonstrating the feasibility of CONFINE. However, we highlight that recent studies have reported vulnerabilities [57, 58], and real-world deployments may therefore prefer more robust alternatives or incorporate tailored mitigation strategies to address these concerns [59].

We encoded the provisioner's `Log Server` in GO. The implementation of our `Secure Miner` component facilitates input reception from users via a console application (i.e., the `TEE Interface` in Sect. 6.1). We store

[8] `https://docs.edgeless.systems/ego`. Accessed: December 2, 2025.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

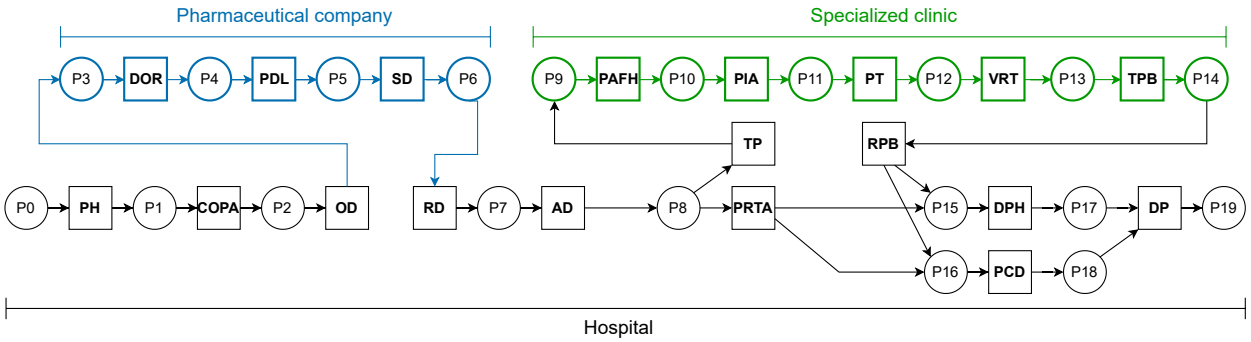


Figure 9: *HeuristicsMiner* output in CONFINE

the descriptive information of the `Trusted App` (i.e., heap size, embedded files, and environment variables) in a JavaScript Object Notation (JSON) file. In our implementation, the `Secure Miner` app maintains references to provisioners' `Log Servers` in a dedicated JSON file, alongside the label of the *iid* attribute in their respective log partitions. We embed in the `TEE` during deployment. We resort to a Transport Layer Security (TLS) [60] communication channel between miners and provisioners over the HyperText Transfer Protocol (HTTP) web protocol to secure the information exchange and authenticate the miner's organization. The control data structures *IIDMap* and *PMap* (refer to Alg. 1) generated by the `Secure Miner` during the CONFINE protocol are formatted as JSON files.

We integrate two established process mining algorithms within our `Secure Miner` implementation: *HeuristicsMiner* [24] and *DeclareConformance* [27] from the *pm4py* library.[9] In Sect. 2.1.1, we provided an overview of their core mechanisms. The selected algorithms are representative of process mining techniques along two complementary dimensions. First, *HeuristicsMiner* and *DeclareConformance* perform the core process mining tasks of process discovery and conformance checking, respectively. Second, they reflect the two predominant modeling paradigms in process mining, namely *imperative* (as in the case of *HeuristicsMiner*) and *declarative* approaches (as in the case of *DeclareConformance*) [61, 62]. Through the integration of these algorithms, we demonstrate that our implementation is applicable to distinct process mining tasks and accommodates different methodological paradigms.

Upon receiving each complete case (see Def. 4.7), our *HeuristicsMiner* implementation produces a causal net, which is subsequently transformed into the corresponding workflow net of the analyzed process. The `TEE` outputs the resulting models in the Petri Net Markup Language (PNML) format, a standard for representing workflow nets.[10] Similarly, our *DeclareConformance* implementation computes fitness scores, which are returned from the `TEE` in the form of JSON files. For both algorithms, we provide the following two versions. **Incremental:** In this version, the algorithm takes batches of the complete cases as an input and updates a partial result while data transmission is still ongoing; **Non-incremental:** The algorithm is triggered at the end of the data transmission, once all the cases are collected. Our implementation of CONFINE, including the *HeuristicsMiner* and the *DeclareConformance* in Go, is openly accessible at the following URL: `github.com/Process-in-Chains/CONFINE/`.

8.2. Convergence

To experimentally validate Thm. 7.2, we run a *convergence* test. We created a synthetic event log consisting of 1000 cases (as per Def. 4.7) of 14 events on average (see Tab. 6) by simulating the inter-organizational process of Fig. 1.[11] and we partitioned it into three sub-logs (one per involved organization), an excerpt

[9] `github.com/process-intelligence-solutions/pm4py`. Accessed: December 2, 2025.
[10] `www.pnml.org`. Accessed: December 2, 2025.
[11] We generated the event log through BIMP (`bimp.cs.ut.ee`; accessed: December 2, 2025). We filtered the log by retaining the sole events that report on the completion of activities, and removed the start and end events of the pharmaceutical company and specialized clinic's sub-processes.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

of which is listed in Tab. 2. We run the stand-alone *HeuristicsMiner* on the former and process the latter through our CONFINE toolchain. As expected, the results converge and are depicted in Fig. 9 in the form of a workflow net [63]. For the sake of clarity, we color activities recorded by the organizations following the scheme of Tab. 6 (black for the hospital, blue for the pharmaceutical company, and green for the specialized clinic). Additionally, we add labels to specify the organization involved in separate fragments of the workflow net.

8.3. Performance Assessment

In what follows, we report on our experiments conducted to put our tool implementation to the test. As discussed in Sect. 6.1, the availability of space in the dedicated TEE areas is subject to hardware limitations. Therefore, we focus on memory consumption since exceeding those limits could diminish the level of security guaranteed by TEEs. We gauge the memory usage with synthetic and real-life event logs, to observe the trend during the enactment of our protocol. In particular, we focus on runtime consumption (Sect. 8.3.1), the effect that the segment size hyperparameter discussed in Sect. 6.2 has on efficiency (Sect. 8.3.2), and scalability with respect to the event log size and the number of provisioners (Sect. 8.3.3). The raw execution data alongside the scripts to plot the obtained results are openly accessible at `github.com/Process-in-Chains/CONFINE/tree/main/evaluation/`.

8.3.1. Runtime analysis

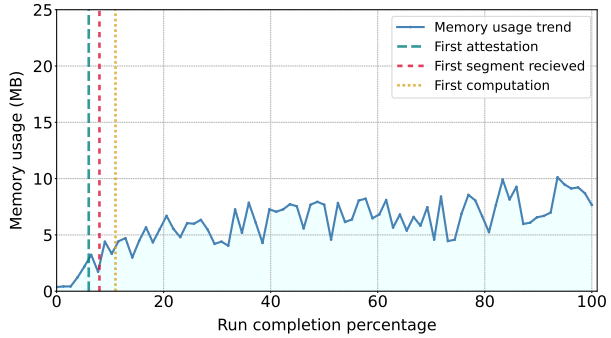
Figs. 10(a) to 10(d) display the runtime memory utilization of our `Secure Miner` implementation (in MegaBytes) using the synthetic log derived from our motivating scenario as input. Fig. 10(a) illustrates the memory utilization trend with the integration of the incremental variant of the HeuristicsMiner, as described in Sect. 6.2.4. Differently, Fig. 10(b) pictures the test outcome resulting from the execution of the protocol with the non-incremental variant of the algorithm. 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 100 KiloBytes. In both figures, we observe that after the initiation of the data transmission stage (the dashed red line), the memory utilization falls approximately between 5 and 10 MB. The major difference between the two trends concerns the computation phase. In Fig. 10(a), the HeuristicsMiner starts computing cases when the data transmission is still ongoing. This ensures the memory utilization trend remains within a constant range until the end of the run. In Fig. 10(b), the HeuristicsMiner is triggered after all the `Provisioners` have delivered their respective log partitions (as per Def. 4.5). Consequently, the final peak at the yellow dashed line signifies the computation of the entire event log, making the memory utilization diverge to a value of between 20 and 25 MB. This behavior is confirmed by Figs. 10(c) and 10(d), in which we consider the memory usage trend of the DeclareConformance algorithm variants. The tests indicate that the DeclareConformance demands a higher memory utilization than the HeuristicsMiner due to the involvement of the declarative process model in the conformance checking task.

To verify whether these behaviors are 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 (Sepsis [54] and BPIC 2013 [55]). The characteristics of the event logs are summarized in Tab. 6. 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. Figs. 10(e) and 10(f) depict the results with the integration of the incremental variants of HeuristicsMiner and DeclareConformance. We observe that the BPIC 2013 event log processing demands more memory probably owing to its larger size. Conversely, the Sepsis event log turns out to entail the least expensive run.

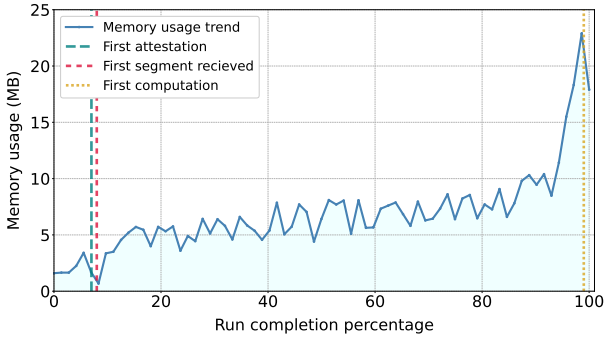
8.3.2. Segment Size

To verify whether these trends are affected by the dimension of the exchanged data segments, we conducted additional tests to examine the trend of memory usage and the overhead of the exchanged message as the *seg_size* varies with all the aforementioned event logs. Notably, the polylines displayed in Fig. 11(a) indicate

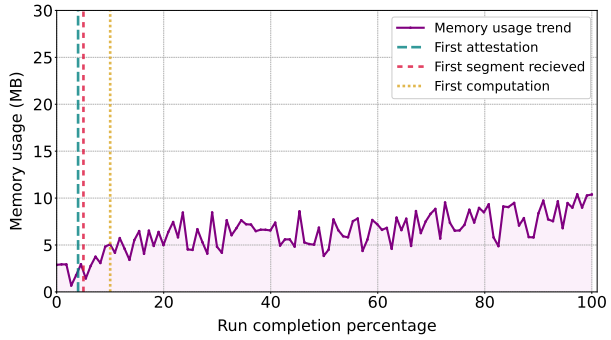
Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381



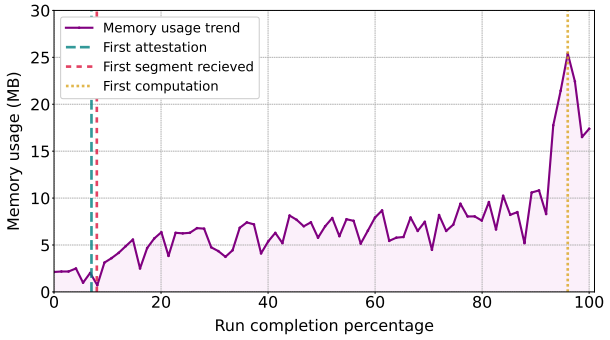
(a) Memory usage with incremental HeuristicsMiner



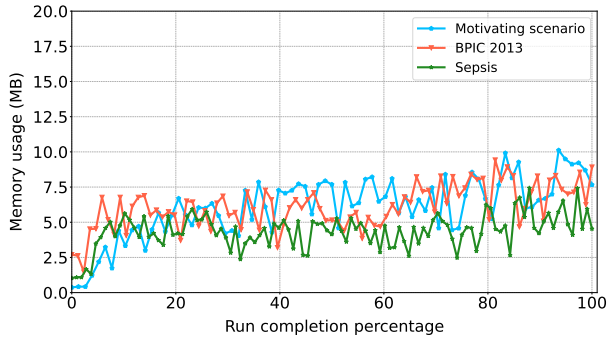
(b) Memory usage with the non-incremental HeuristicsMiner



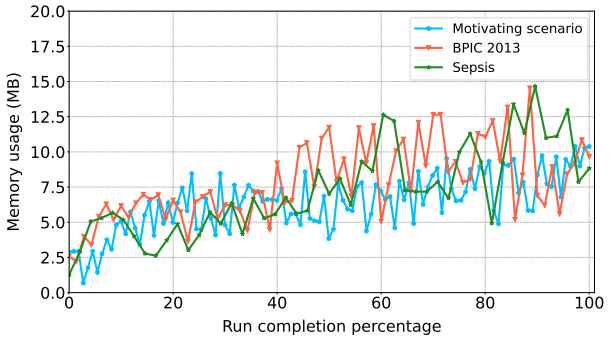
(c) Memory usage with incremental DeclareConformance



(d) Memory usage with non-incremental DeclareConformance



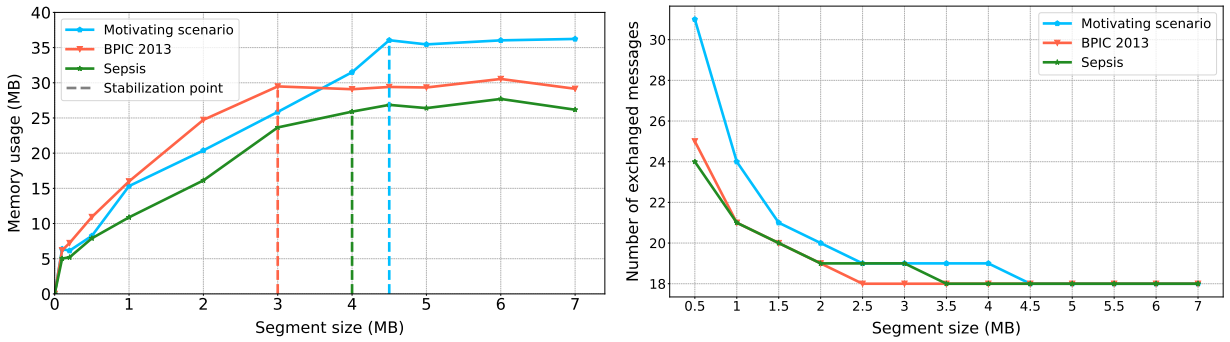
(e) Memory usage with three event logs and the incremental HeuristicsMiner



(f) Memory usage with three event logs and the incremental DeclareConformance

Figure 10: Memory usage test results

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381



(a) Segment size impact on memory usage

(b) Segment size impact on exchanged messages overhead

Figure 11: Segment size test result

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.

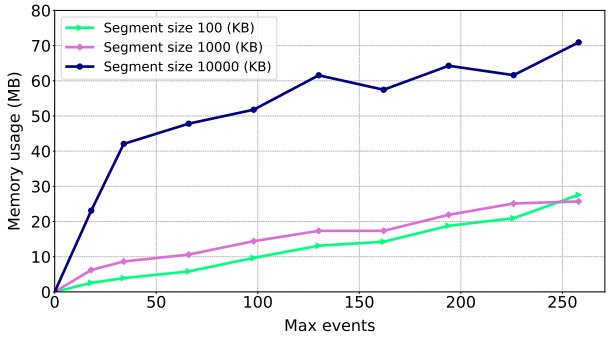
Fig. 11(b) depicts the relationship between segment size and the number of message exchanges through CONFINE. As the segment size increases, there is a decrease in the number of messages exchanged, indicating a reduction in communication overhead. This trend is consistent across different logs and segment sizes, with a stabilizing effect observed after the segment size of 4.5 MB suggesting an optimal range to minimize message exchanges.

8.3.3. Scalability

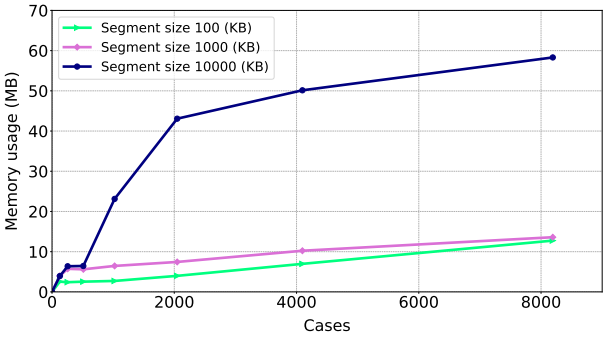
In the remainder of this section, we examine the scalability of the `Secure Miner`, focusing on its capacity to efficiently manage an increasing workload in the presence of limited memory resources. We implemented three distinct test configurations gauging the average runtime memory usage as variations of our motivating scenario's log. In particular, we considered (I) the maximum number of events per case, (II) the number of cases $|\widehat{\text{IID}}|$, 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 $|\widehat{\text{IID}}| = 2^{x_{iid}}$ having $x_{iid} \in \{7, 8, \ldots, 13\}$. 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 ($|\widehat{\mathcal{O}}| \in \{1, 2, \ldots, 8\}$). 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 [64], 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 [65].

Tab. 9(d) lists the measurements we obtained. We describe them to elucidate the observed patterns. Fig. 12(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

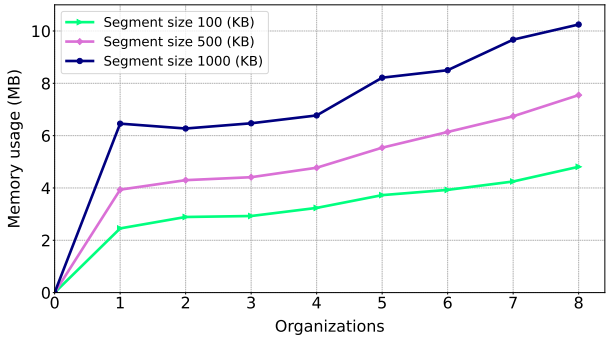
Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381



(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 12: Scalability test results

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. 12(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. 12(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 mitigation strategies.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

Table 7: Summary of the assumptions considered in this article

Reference	Description
Asm. A1	Provisioning organizations play the role of the *honest* party
Asm. A2	Miner organizations play the role of the *malicious* party
Asm. A3	Events are comprised of an *iid*, an *activity* label and a *timestamp*
Asm. A4	The iids are synchronized across the involved organizations
Asm. A5	The timestamps across organizations are temporally consistent
Asm. A6	Organizations can verify one another's identities through IAM services
Asm. A7	The TEE provides application-level protection, hardware-based memory encryption, and remote attestability
Asm. A8	The communication channel underlying CONFINE ensures reliable delivery, no-duplication, and authenticity
Asm. A9	`Provisioner`s retain authorization policies to decide whether requests from `Secure Miner`s are permitted
Asm. A10	The segment size range spans from the largest case size among the merged cases to the TEE's memory capacity

9. Discussion

Following the above empirical examination of our approach implementation, we discuss CONFINE by addressing the implications associated with its practical instantiation (Sect. 9.1) and outlining the limitations that pave the path for future work (Sect. 9.2).

9.1. Practical Implications

The applications of CONFINE span across an array of domains, including and beyond healthcare, in which our running example in Sect. 3 is rooted. It particularly applies to scenarios in which one or more organizations are interested in process analytics outcomes based on data they hold, but cannot be disclosed to others or to the miners. In healthcare, CONFINE can aid in analyzing patient pathways on multiple healthcare providers while maintaining the secrecy of patient information, thereby enabling the identification of bottlenecks and the optimization of treatment procedures [10, 11]. Supply chain management companies can benefit from CONFINE by securely analyzing the flow of goods and information between multiple partners, thereby streamlining operations, reducing costs, and improving delivery timelines. In the manufacturing domain, CONFINE aids in examining branching workflows across independent organizations, optimizing resource allocation, and maintaining operational data secrecy. However, the acquisition of competitive advantage out of knowledge leakage must be prevented [8].

Our solution enables organizations to differentiate the sources from which the `Secure Miner` retrieves the input data. This feature allows organizations to share event logs with others while locally enriching their own dataset with additional information for their internal CONFINE protocol executions [14]. The advantage is twofold. On the one hand, this scheme lets an organization cross-link information stemming from the overarching multi-party process with internal procedure records and thus enriches the output of the `Secure Miner`. On the other hand, information that is not relevant to the external organizations is not disclosed.

A crucial aspect for facilitating the adoption of our solution in real-world settings is the availability of accessible repositories hosting and distributing variants of the `Secure Miner`, each tailored to a specific TEE technology and equipped with different process mining algorithms (as in the the work of Soriente et al. [66]). The development of trusted software for application-level TEEs requires specific Software Development Kits (SDKs), which may pose a significant integration burden for organizations lacking specialized expertise. By relying on such repositories, organizations could select the `Secure Miner` trusted application that best matches their intended process mining tasks and infrastructural constraints. At the same time, we acknowledge that this approach introduces potential risks, including malicious modifications to the `Secure Miner` source code before its deployment in a TEE. Relying on a trusted third-party hosting services for distributing verified versions of the `Secure Miner`, and counterchecking the associated measurements before startup through code-review practices can mitigate these risks.

Tab. 7 collects all the assumptions underpinning this work, which pertains to the adversarial roles of the involved organizations (Asms. A1 and A2), the event log format (Asms. A3 to A5), the mechanisms for organizational trust and authorization (Asms. A6 and A9), the security guarantees provided by the TEE

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

(Asm. A7), and the protocol execution (Asms. A8 and A10). A real-world deployment of CONFINE should ensure that those assumptions are properly addressed.

IT engineers responsible for implementing the theoretical principles of CONFINE can instantiate the framework by following the steps outlined below. (1) They analyze the privacy requirements of the involved business entities and define their role within the framework (i.e., provisioner, miner, or both). (2) Organizations intending to participate in the CONFINE framework as miners select a TEE technology and equip themselves with a TEE-enabled processor. Cloud-based solutions may serve as an alternative for organizations lacking the necessary hardware. Different organizations may adopt distinct TEE implementations, provided that each conform to the TEE profile defined in Sect. 2.1.2. With such a heterogeneous setting, the specific TEE technology chosen by each organization should be known to the others to ensure interoperability. This is especially relevant for attestations, since they require the correct handling of different attestation formats and verification services. (3) Process mining vendors select the algorithms to be integrated into their `Secure Miner`s and either obtain a verified distribution of the corresponding trusted application (as discussed above) or build their own implementation. (4) The organizations exchange the credentials that are necessary for executing the CONFINE protocol, such as the `Provisioner`s' web references, organizational credentials, and the `Secure Miner`s' measurements (which depend on the specific process mining algorithms integrated into each instance). (5) Finally, the involved organizations can start up their `Secure Miner` and `Provisioner` components to execute the CONFINE protocol.

9.2. Limitations and Future Work

In Sect. 7.1, we delineated the security guarantees that the `TEE` core provides within CONFINE. However, recent studies have demonstrated that `TEE` technologies remain susceptible to three principal threat vectors: *(i)* side-channel attacks, *(ii)* physical attacks, *(iii)* root-of-trust attacks. Side-channel attacks exploit secondary information leakage such as memory access patterns [67], or timing variations [58], to undermine hardware-backed memory encryption and infer confidential data. Physical attacks arise when adversaries obtain direct access to the TEE's machine and manipulate hardware-level parameters, such as inducing voltage faults, to cause the processor to skip critical security instructions [57, 68]. Root-of-trust attacks stem from vendor-specific vulnerabilities that enable adversaries to compromise attestation keys at the production stage to forge attestation evidence or recover chip-endorsement keys [69].

In future work, we plan to revisit the assumptions listed in Tab. 7. First, we aim to incorporate mitigation strategies to address scenarios in which provisioners behave maliciously and inject manipulated event logs. We also intend to extend our protocol to operate over unreliable communication channels.

Our merge operations currently count on the availability of consistent instance identifier attributes to be used as merge keys. This presupposes prior knowledge of how each participant encodes its case identifiers. Moreover, we hypothesize the existence of a universal clock to ensure timestamp synchronization across log partitions. To overcome these limitations, future work will explore techniques for establishing or inferring a shared schema for case identifiers as well as synchronization-agnostic strategies for merging temporally misaligned log partitions.

In addition, we plan to provide a more explicit specification of the mechanisms governing organizational trust and authorization. We will investigate the integration of federated IAM solutions for cross-organization identity management and the adoption of attribute-based access control models to formalize authorization policies grounded in contextual process data [50]. Also, we plan to integrate usage control policies that regulate how event logs are processed by the `Secure Miner`. Specifically, building on prior blockchain-based approaches [70, 19], we plan to extend CONFINE with enforcement and monitoring mechanisms that represent usage policies as smart contracts, thus providing verifiable governance and auditable lifecycle management. In parallel, recent work on the dynamic generation of trusted applications [71] offers a complementary avenue for embedding policy-specific enforcement logic within TEEs at runtime. Integrating these two directions within CONFINE would enable organizations to dynamically generate `Secure Miner` services tailored to their usage policies while continuously monitoring policy compliance through a blockchain-based infrastructure. A threat to secrecy lies in the reconstruction of original data back from the mining outcome. Keeping this aspect in mind is crucial to determine the mining algorithm to be embedded in the `Secure Miner`. Studies in this

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

regard have been conducted, among others, in [72, 73]. Integrating the proposed recommendations with CONFINE paves the path for future investigations.

10. Conclusion

This article presented CONFINE, a framework to safeguard data secrecy in process mining. Unlike traditional approaches that either modify the input data or impose intermediate log representations to protect sensitive information, CONFINE leverages TEEs to securely process unaltered event logs beyond organizational boundaries. Compared to our previous work [21, 22], the research work that led to this article contributes several key advancements. We conduct a formal analysis of the CONFINE framework, including the pseudocode for the CONFINE protocol. We analyze the security guarantees that TEEs offer in the context of our solution. We extend the framework's applicability by incorporating a conformance checking algorithm among the supported process mining techniques. Finally, we enhance our evaluation with both qualitative insights and quantitative assessments. The new results demonstrate the formal correctness of our approach and provide empirical evidence for its memory feasibility with different process mining tasks.

Future research will focus on addressing many of the assumptions outlined in Tab. 7, and beyond. Among other endeavors, we plan to investigate strategies for managing temporal misalignment in distributed settings, handling heterogeneity in the identification of cases, mitigating the effects of maliciously altered event logs, and designing dedicated authentication policies for integration into collaborative environments. Furthermore, future work will focus on the governance of information exchange, usage policy enforcement for data treatment, and cross-organization identity management.

Data availability. The source code of our implementation and the experimental results are available at github.com/Process-in-Chains/CONFINE.

Acknowledgments. This work was partially funded by the Italian Ministry of University and Research under PRIN grant B87G22000450001 (PINPOINT), by Sapienza University of Rome under grant RG123188B3F7414A (ASGARD), and by the EU-NGEU under the Cyber 4.0 NRRP MIMIT programme (Health-e-Data).

References

[1] M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, Fundamentals of Business Process Management, Second Edition, Springer, 2018. doi:10.1007/978-3-662-56509-4.

[2] W. M. P. van der Aalst, Process mining: Overview and opportunities, ACM Transactions on Management Information Systems 3 (2012) 7:1–7:17. doi:10.1145/2229156.2229157.

[3] J. De Weerdt, M. T. Wynn, Foundations of process event data, in: [74], 2022, pp. 193–211. doi:10.1007/978-3-031-08848-3_6.

[4] W. M. P. van der Aalst, Intra- and inter-organizational process mining: Discovering processes within and between organizations, in: The Practice of Enterprise Modeling (IFIP) 2011, volume 92 of *Lecture Notes in Business Information Processing*, Springer, 2011, pp. 1–11. doi:10.1007/978-3-642-24849-8_1.

[5] F. Oberdorf, M. Schaschek, S. Weinzierl, N. Stein, M. Matzner, C. M. Flath, Predictive end-to-end enterprise process network monitoring, Business Information Systems Engineering 65 (2023) 49–64. doi:10.1007/S12599-022-00778-4.

[6] M. Morales-Sandoval, J. A. Molina, H. M. Marín-Castro, J. L. G. Compeán, Blockchain support for execution, monitoring and discovery of inter-organizational business processes, PeerJ Computer Science 7 (2021) e731. doi:10.7717/PEERJ-CS.731.

[7] T. Ho, A. Kumar, N. Shiwakoti, Supply chain collaboration and performance: an empirical study of maturity model, SN Applied Sciences 2 (2020) 726. doi:10.1007/s42452-020-2468-y.

[8] K. H. Tan, W. P. Wong, L. Chung, Information and knowledge leakage in supply chain, Information Systems Frontiers 18 (2016) 621–638. doi:10.1007/s10796-015-9553-6.

[9] W. Fdhila, D. Knuplesch, S. Rinderle-Ma, M. Reichert, Verifying compliance in process choreographies: Foundations, algorithms, and implementation, Information Systems 108 (2022) 101983. doi:10.1016/J.IS.2022.101983.

[10] C. Liu, H. Li, S. Zhang, L. Cheng, Q. Zeng, Cross-department collaborative healthcare process model discovery from event logs, IEEE Transactions on Automation Science and Engineering 20 (2023) 2115–2125. doi:10.1109/TASE.2022.3194312.

[11] J. Jeon, J. Kim, M. Shin, M. Kim, A blockchain-based trust model for supporting collaborative healthcare data management, Computer Systems Science and Engineering 46 (2023) 3403–3421. doi:10.32604/CSSE.2023.036658.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

[12] C. Liu, Q. Li, X. Zhao, Challenges and opportunities in collaborative business process management: Overview of recent advances and introduction to the special issue, Information Systems Frontiers 11 (2009) 201–209. doi:10.1007/S10796-008-9089-0.
[13] M. Müller, N. Ostern, D. Koljada, K. Grunert, M. Rosemann, A. Küpper, Trust mining: analyzing trust in collaborative business processes, IEEE Access 9 (2021) 65044–65065. doi:10.1109/ACCESS.2021.3075568.
[14] S. A. Fahrenkrog-Petersen, H. van der Aa, M. Weidlich, Optimal event log sanitization for privacy-preserving process mining, Data Knowledge Engineering 145 (2023) 102175. doi:10.1016/J.DATAK.2023.102175.
[15] S. A. Fahrenkrog-Petersen, H. van der Aa, M. Weidlich, PRETSA: event log sanitization for privacy-aware process discovery, in: International Conference on Process Mining (ICPM) 2019, IEEE, 2019, pp. 1–8. doi:10.1109/ICPM.2019.00012.
[16] G. Elkoumy, S. A. Fahrenkrog-Petersen, M. Dumas, P. Laud, A. Pankova, M. Weidlich, Shareprom: A tool for privacy-preserving inter-organizational process mining, in: Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Track at BPM 2020, CEUR-WS.org, 2020, pp. 72–76. URL: https://ceur-ws.org/Vol-2673/paperDR02.pdf.
[17] G. Elkoumy, S. A. Fahrenkrog-Petersen, M. Dumas, P. Laud, A. Pankova, M. Weidlich, Secure multi-party computation for inter-organizational process mining, in: Enterprise, Business-Process and Information Systems Modeling EMMSAD 2020, volume 387 of *Lecture Notes in Business Information Processing*, Springer, 2020, pp. 166–181. doi:10.1007/978-3-030-49418-6_11.
[18] E. Mansour, A. V. Sambra, S. Hawke, M. Zereba, S. Capadisli, A. Ghanem, A. Aboulnaga, T. Berners-Lee, A demonstration of the solid platform for social web applications, in: International Conference on World Wide Web (WWW) 2016, ACM, 2016, pp. 223–226. doi:10.1145/2872518.2890529.
[19] D. Basile, C. Di Ciccio, V. Goretti, S. Kirrane, A blockchain-driven architecture for usage control in solid, in: International Conference on Distributed Systems (ICDCS) 2023 - Workshops, IEEE, 2023, pp. 19–24. doi:10.1109/ICDCSW60045.2023.00009.
[20] M. Sabt, M. Achemlal, A. Bouabdallah, Trusted execution environment: What it is, and what it is not, in: International Conference on Trust, Security and Privacy in Computing and Communications TrustCom 2015, IEEE, 2015, pp. 57–64. doi:10.1109/TRUSTCOM.2015.357.
[21] V. Goretti, D. Basile, L. Barbaro, C. Di Ciccio, Trusted execution environment for decentralized process mining, in: CAiSE 2024, Springer, 2024, pp. 509–527. doi:10.1007/978-3-031-61057-8_30.
[22] V. Goretti, D. Basile, L. Barbaro, C. Di Ciccio, CONFINE: preserving data secrecy in decentralized process mining, in: Doctoral Consortium and Demo Track ICPM 2024, volume 3783, 2024. URL: https://ceur-ws.org/Vol-3783/paper_324.pdf.
[23] W. M. P. van der Aalst, A. Adriansyah, A. K. A. de Medeiros, F. Arcieri, T. Baier, T. Blickle, R. P. J. C. Bose, P. van den Brand, R. Brandtjen, J. C. A. M. Buijs, et al., Process mining manifesto, in: F. Daniel, K. Barkaoui, S. Dustdar (Eds.), International Conference on Business Process Management Workshops (BPM) 2011, volume 99 of *Lecture Notes in Business Information Processing*, Springer, 2011, pp. 169–194. doi:10.1007/978-3-642-28108-2_19.
[24] A. J. M. M. Weijters, W. M. P. van der Aalst, A. K. Alves De Medeiros, Process mining with the HeuristicsMiner algorithm, 2006. URL: https://pure.tue.nl/ws/portalfiles/portal/2388011/615595.pdf.
[25] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, A. P. Barros, Workflow patterns, Distributed and Parallel Databases 14 (2003) 5–51. URL: http://dx.doi.org/10.1023/A:1022883727209.
[26] W. M. P. van der Aalst, Conformance Checking, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 243–274. doi:10.1007/978-3-662-49851-4_8.
[27] I. Donadello, F. Riva, F. M. Maggi, A. Shikhizada, Declare4py: A python library for declarative process mining, in: Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Trackco-located with International Conference on Business Process Management (BPM) 2022, volume 3216 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 117–121.
[28] F. A. Yasmin, F. A. Bukhsh, P. de Alencar Silva, Process enhancement in process mining: A literature review, in: P. Ceravolo, M. T. Gómez-López, M. van Keulen (Eds.), International Symposium on Data-driven Process Discovery and Analysis (SIMPDA) 2018, volume 2270 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018, pp. 65–72. URL: https://ceur-ws.org/Vol-2270/short4.pdf.
[29] P. Jauernig, A.-R. Sadeghi, E. Stapf, Trusted execution environments: Properties, applications, and challenges, IEEE Security and Privacy 18 (2020) 56–60. doi:10.1109/MSEC.2019.2947124.
[30] P. Antonino, W. A. Woloszyn, A. W. Roscoe, Guardian: Symbolic validation of orderliness in SGX enclaves, in: Cloud Computing Security Workshop (CCSW@CCS) 2021, ACM, 2021, pp. 111–123. doi:10.1145/3474123.3486755.
[31] B. Mei, S. Xia, W. Wang, D. Lin, Cabin: Confining untrusted programs within confidential vms, in: Information and Communications Security (ICICS) 2024, volume 15056 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 165–184. doi:10.1007/978-981-97-8798-2_9.
[32] V. Costan, S. Devadas, Intel SGX explained, IACR Cryptology ePrint Archive (2016) 86. URL: https://eprint.iacr.org/2016/086.pdf.
[33] H. Birkholz, D. Thaler, M. Richardson, N. Smith, W. Pan, Remote attestation procedures (RATS) architecture, RFC 9334 (2023) 1–46. doi:10.17487/RFC9334.
[34] S. C. Helble, I. D. Kretz, P. A. Loscocco, J. D. Ramsdell, P. D. Rowe, P. Alexander, Flexible mechanisms for remote attestation, ACM Transactions on privacy and security 24 (2021) 29:1–29:23. doi:10.1145/3470535.
[35] J. Ménétrey, C. Göttel, A. Khurshid, M. Pasin, P. Felber, V. Schiavoni, S. Raza, Attestation mechanisms for trusted execution environments demystified, in: Distributed Applications and Interoperable Systems (DAIS) 2022, volume 13272, 2022, pp. 95–113. doi:10.1007/978-3-031-16092-9_7.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

[36] E. Batista, A. Martínez-Ballesté, A. Solanas, Privacy-preserving process mining: A microaggregation-based approach, Journal of Information Security and Application 68 (2022) 103235. doi:10.1016/J.JISA.2022.103235.
[37] W. M. P. van der Aalst, Federated process mining: Exploiting event data across organizational boundaries, in: IEEE International Conference on Smart Data Services (SMDS) 2021, IEEE, 2021, pp. 1–7. doi:10.1109/SMDS53860.2021.00011.
[38] R. Cramer, I. Damgård, J. B. Nielsen, Secure Multiparty Computation and Secret Sharing, Cambridge University Press, 2015. doi:doi.org/10.1017/CBO9781107337756.
[39] 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: International Conference on Process Mining ICPM 2021 Workshops„ volume 433 of *Lecture Notes in Business Information Processing*, Springer, 2021, pp. 369–381. doi:10.1007/978-3-030-98581-3_27.
[40] P. Yuhala, C. Göttel, J. Ménétrey, V. Schiavoni, D. Kozhaya, P. Felber, On real-time guarantees in intel SGX and TDX, in: Euromicro Conference on Real-Time Systems ECRTS 2025, volume 335 of *LIPIcs*, 2025, pp. 8:1–8:25. doi:10.4230/LIPICS.ECRTS.2025.8.
[41] Y. Sun, S. Wang, H. Li, F. Li, Building enclave-native storage engines for practical encrypted databases, Proceedings of the VLDB Endowment 14 (2021) 1019–1032. doi:10.14778/3447689.3447705.
[42] M. Brandenburger, C. Cachin, M. Lorenz, R. Kapitza, Rollback and forking detection for trusted execution environments using lightweight collective memory, in: International Conference on Dependable Systems and Networks (DSN) 2017, IEEE Computer Society, 2017, pp. 157–168. doi:10.1109/DSN.2017.45.
[43] E. R. Aguilar, F. García, F. Ruiz, M. Piattini, L. Calahorra, M. García, R. Martin, Process modeling of the health sector using BPMN: A case study, in: International Conference on Health Informatics (HEALTHINF), INSTICC - Institute for Systems and Technologies of Information, Control and Communication, 2008, pp. 173–178. doi:10.5220/0001042201730178.
[44] S. E. Sampson, G. Schmidt, J. W. Gardner, J. Van Orden, Process coordination within a health care service supply network, Journal of Business Logistics 36 (2015) 355–373. doi:10.1111/jbl.12106.
[45] 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. doi:10.1016/j.accinf.2018.11.002.
[46] E. Mendelson, Introduction to Mathematical Logic, Chapman and Hall, 2015.
[47] W. M. P. van der Aalst, Process Mining - Data Science in Action, Second Edition, Springer, 2016. doi:10.1007/978-3-662-49851-4.
[48] N. Koch, A. Kraus, The expressive power of UML-based web engineering, in: Workshop on Web-Oriented Software Technology (IWWOST) 2002, volume 16, Citeseer, 2002, pp. 40–41. doi:doi.org/10.1007/3-540-36208-8_5.
[49] C. Cachin, R. Guerraoui, L. E. T. Rodrigues, Introduction to Reliable and Secure Distributed Programming (2. ed.), Springer, 2011. doi:10.1007/978-3-642-15260-3.
[50] J. Crampton, C. Morisset, Ptacl: A language for attribute-based access control in open systems, in: Principles of Security and Trust (POST) 2012, volume 7215 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 390–409. doi:10.1007/978-3-642-28641-4_21.
[51] J. Claes, G. Poels, Merging event logs for process mining: A rule based merging method and rule suggestion algorithm, Expert Systems with Applications 41 (2014) 7291–7306. doi:10.1016/J.ESWA.2014.06.012.
[52] H. Birkholz, D. Thaler, M. Richardson, N. Smith, W. Pan, Remote attestation procedures (RATS) architecture, volume 9334, 2023, pp. 1–46. doi:10.17487/RFC9334.
[53] L. Lamport, Proving the correctness of multiprocess programs, IEEE Transactions on Software Engineering 3 (1977) 125–143. doi:10.1109/TSE.1977.229904.
[54] F. Mannhardt, Sepsis cases - event log, 2016. doi:10.4121/UUID:915D2BFB-7E84-49AD-A286-DC35F063A460.
[55] W. Steeman, BPI challenge 2013, incidents, 2013. doi:10.4121/UUID:500573E6-ACCC-4B0C-9576-AA5468B10CEE.
[56] K. Bagher, S. Lai, Sgx-stream: A secure stream analytics framework in sgx-enabled edge cloud, Journal of Information Security and Application 72 (2023) 103403. doi:10.1016/J.JISA.2022.103403.
[57] K. Murdock, D. F. Oswald, F. D. Garcia, et al., Plundervolt: Software-based fault injection attacks against intel SGX, in: IEEE Symposium on Security and Privacy (SP) 2020, IEEE, 2020, pp. 1466–1482. doi:10.1109/SP40000.2020.00057.
[58] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, et al., Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution, in: USENIX 2018, USENIX Association, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck.
[59] S. Fei, Z. Yan, W. Ding, H. Xie, Security vulnerabilities of SGX and countermeasures: A survey, ACM Computing Surveys 54 (2022) 126:1–126:36. doi:10.1145/3456631.
[60] S. A. Thomas, SSL and TLS Essentials: Securing the Web, Wiley, 2000. URL: https://dl.acm.org/doi/10.5555/519018.
[61] P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, H. A. Reijers, Imperative versus declarative process modeling languages: An empirical investigation, in: Business Process Management Workshops (BPM) 2011, volume 99 of *Lecture Notes in Business Information Processing*, Springer, 2011, pp. 383–394. doi:10.1007/978-3-642-28108-2_37.
[62] C. Di Ciccio, M. Montali, Declarative process specifications: Reasoning, discovery, monitoring, in: [74], 2022, pp. 108–152. doi:10.1007/978-3-031-08848-3_4.
[63] W. M. P. van der Aalst, Verification of workflow nets, in: International Conference on Applications and Theory of Petri Nets and Concurrency (ICATPN) 1997, 1997, pp. 407–426. doi:10.1007/3-540-63139-9_48.
[64] J. P. Barrett, The coefficient of determination—some limitations, The American Statistician (1974) 19–20. doi:10.1080/00031305.1974.10479056.
[65] X. Su, X. Yan, C.-L. Tsai, Linear regression, Wiley Interdisciplinary Reviews: Computational Statistics 4 (2012) 275–294. doi:doi.org/10.1002/wics.1198.
[66] C. Soriente, G. Karame, W. Li, S. Fedorov, Replicatee: Enabling seamless replication of SGX enclaves in the cloud, in: IEEE

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

European Symposium on Security and Privacy (EuroS&P) 2019, IEEE, 2019, pp. 158–171. doi:10.1109/EUROSP.2019.00021.
[67] M. Morbitzer, M. Huber, J. Horsch, S. Wessel, Severed: Subverting amd's virtual machine encryption, in: European Workshop on Systems Security (EuroSec@EuroSys) 2018, ACM, 2018, pp. 1:1–1:6. doi:10.1145/3193111.3193112.
[68] P. Qiu, D. Wang, Y. Lyu, G. Qu, Voltjockey: Breaching trustzone by software-controlled voltage manipulation over multi-core frequencies, in: Conference on Computer and Communications Security (CCS) 2019, ACM, 2019, pp. 195–209. doi:10.1145/3319535.3354201.
[69] R. Buhren, C. Werling, J. Seifert, Insecure until proven updated: Analyzing AMD sev's remote attestation, in: Communications Security (CCS) 2019, ACM, 2019, pp. 1087–1099. doi:10.1145/3319535.3354216.
[70] D. Basile, C. Di Ciccio, V. Goretti, S. Kirrane, Blockchain based resource governance for decentralized web environments, Frontiers Blockchain 6 (2023). doi:10.3389/FBLOC.2023.1141909.
[71] V. Goretti, S. Kirrane, C. Di Ciccio, Usage control for process discovery through a trusted execution environment, in: ICSOC, 2025, pp. 268–286. doi:10.1007/978-981-95-5015-9_20.
[72] S. N. von Voigt, S. A. Fahrenkrog-Petersen, D. Janssen, A. Koschmider, F. Tschorsch, F. Mannhardt, O. Landsiedel, M. Weidlich, Quantifying the re-identification risk of event logs for process mining - empiricial evaluation paper, in: Advanced Information Systems Engineering (CAiSE) 2020, volume 12127 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 252–267. doi:10.1007/978-3-030-49435-3_16.
[73] S. A. Fahrenkrog-Petersen, M. Kabierski, H. van der Aa, M. Weidlich, Semantics-aware mechanisms for control-flow anonymization in process mining, Information Systems 114 (2023) 102169. doi:10.1016/J.IS.2023.102169.
[74] W. M. P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, volume 448 of *Lecture Notes in Business Information Processing*, Springer, 2022.

Pre-print copy of the manuscript published by Elsevier
identified by DOI: 10.1016/j.jisa.2026.104381

This document is a pre-print copy of the manuscript
(Basile et al. 2026)
published by Elsevier.

The final version of the paper is identified by DOI: 10.1016/j.jisa.2026.104381

References

Basile, Davide, Valerio Goretti, Luca Barbaro, Hajo A. Reijers, and Claudio Di Ciccio (2026). "A TEE-based approach for preserving data secrecy in process mining with decentralized sources". In: *Journal of Information Security and Applications* 98, p. 104381. ISSN: 2214-2126. DOI: 10.1016/j.jisa.2026.104381.

BibTeX

```
@Article{         Basile.etal/JISA2026:TEEforProcessMining,
  author        = {Basile, Davide and Goretti, Valerio and Barbaro, Luca and
                   Reijers, Hajo A. and Di Ciccio, Claudio},
  journal       = {Journal of Information Security and Applications},
  title         = {A {TEE}-based approach for preserving data secrecy in
                   process mining with decentralized sources},
  year          = {2026},
  issn          = {2214-2126},
  pages         = {104381},
  volume        = {98},
  doi           = {10.1016/j.jisa.2026.104381},
  keywords      = {Secrecy; Process mining; Confidential computing; Business
                   process management},
  publisher     = {Elsevier}
}
```