Usage Control for Process Discovery through a Trusted Execution Environment

Valerio Goretti[1], Sabrina Kirrane[2], and Claudio Di Ciccio[3]

[1] Sapienza University of Rome, Italy, valerio.goretti@uniroma1.it
[2] Vienna University of Economics and Business, Austria, sabrina.kirrane@wu.ac.at
[3] Utrecht University, The Netherlands, c.diciccio@uu.nl

Abstract. Process mining provides valuable insights into workflows based on recorded execution data. The registered event logs often contain sensitive information as they may bear personal details about working individuals or reveal organizations' know-how and their operations' history. Therefore, confidentiality and privacy requirements are high priority for process mining. In this paper, we present ProMiSe, a software framework and service that allows users to control information usage for process mining, in particular, for the task of process discovery, from the input provision, through the log processing, to the output consultation. Usage control rules are expressed by means of policies. ProMiSe turns them into dedicated applications running within a Trusted Execution Environment (TEE) to enforce those rules. We put our solution to the test with real-world event logs to assess its computational resource consumption and service time.

Keywords: Confidential computing · Confidentiality · Privacy · TEE

1 Introduction

Process mining offers a powerful approach to analyzing workflows by extracting useful information from event data recorded in information systems [2]. It enables organizations to visualize real-world process executions, identify inefficiencies, and discover opportunities for improvement. Process mining is based on data-driven techniques applied to records that potentially exhibit sensitive information, against secrecy and privacy concerns. Thus, organizations may be reluctant to submit their event logs to process mining services.

In order to protect data from unwanted information leakage, pseudonymization and encryption techniques that protect sensitive information are often used [8, 9]. Nevertheless, studies have shown that even when data is obfuscated, it is still possible to single out individuals or infer information about the original input [6, 19]. Other solutions apply perturbation techniques that alter the original event log to protect sensitive information [15, 17]. These techniques ensure that knowledge about the input can be reconstructed from the output, but the resulting loss of data veracity can compromise the effectiveness of subsequent analyses. More recently, approaches have emerged that leverage hardware-backed solutions to run process mining in computing enclaves [18, 21]. Although mitigating the data distortion issue, they focus on the sole data processing phase, disregarding the treatment of input and output.

To overcome the above limitations, this paper presents ProMiSe, a framework that enables organizations to leverage process mining services under data governance. In particular, ProMiSe focuses on supporting process discovery as a subset of the broader process

mining domain. ProMiSe is designed to be integrated into the infrastructure of process discovery service providers and serves as a trusted platform for client organizations. ProMiSe enables data owners to define usage policies that control who can access the data, how the data is processed, and what outcome can be inspected, ensuring that process discovery analyses comply with regulatory requirements and organizational constraints. Summarizing our contributions, we: *(i)* introduce a lightweight policy language designed to express constraints to preserve secrecy and confidentiality during the input, processing, and output phases of process discovery; *(ii)* present a framework that interprets and enforces these policies, enabling the secure and compliant execution of process discovery operations on sensitive data; and *(iii)* provide an open-source prototypical implementation of our solution, which we use for an empirical evaluation with data stemming from public benchmarks.

The remainder of the paper is structured as follows. Section 2 provides an overview of related approaches and techniques. Section 3 presents a motivating scenario in the healthcare domain and illustrates the system requirements our solution aims to meet. Section 4 presents the architecture of ProMiSe. An evaluation of our approach is shown in Sect. 5. Section 6 concludes the paper with an outlook for future research endeavors.

2 Background and state of the art

In this section, we first introduce the background concepts that our solution is built upon, and outline work related to our endeavor.

2.1 Background

Usage control extends traditional access control by enforcing policies not only before but also during and after resource access [3]. Unlike simple authorization models, usage control introduces obligations (actions users must perform) and conditions (environmental constraints), as formalized in the UCON$_{ABC}$ model [22]. UCON$_{ABC}$ supports decision continuity and attribute mutability, enabling dynamic policy enforcement. A policy specifies the rules governing how data can be accessed and used. To enable automated and flexible enforcement, policies are often represented in RDF, a graph-based format that encodes rules as triples. Usage control frameworks govern the conditions under which data can be accessed and processed. This approach is particularly relevant to process mining, where multiple analytical techniques are applied, and stringent control mechanisms ensure compliance with proper data usage throughout the analysis. **Process mining** is a discipline across data science and process management aimed at analyzing real-world process executions using event logs recorded by information systems [12]. It helps understand how processes are actually performed, verify compliance with expected behavior, assess performance, and compare different execution outcomes. In this paper we will focus on the process discovery task that gives a record of events in input, where each case (process instance) is represented by a trace, i.e., a sequence of timestamped events tied to specific activities, returns a model that identifies the behavior of the process. For example, foundational techniques for process discovery (such as Heuristic Miner and Alpha Miner) use temporal and ordering patterns in the events of the log [13, 24]. These logs typically follow the XES (eXtensible Event Stream) standard format [1]. The output is a process model or analysis that reveals behavior, deviations, or inefficiencies. A **Trusted Execution Environment (TEE)** is a secure processor area that protects code and

data inside from disclosure or tampering, even by privileged software like the operating system [23]. Frameworks like the Intel Software Guard Extensions (SGX) utilizes *enclaves* to encapsulate the so-called Trusted Applications (TAs) in isolated memory regions and thus securely execute sensitive operations. Data within an enclave is encrypted and inaccessible to external software. Enclaves support sealing, a process that allows data to be securely encrypted and stored outside the enclave, bound to the enclave's identity, ensuring that only the same enclave instance can later decrypt and access the sealed data. Each enclave is uniquely identified by a so-called cryptographic *measurement* computed by the CPU at deployment time [4]. This identity allows external software or services to verify that an enclave runs trusted code on untampered hardware through a mechanism that goes under the name of *remote attestation*. Specifically, the enclave generates a signed remote attestation report that includes its measurement and platform information. This report can be verified by Intel's attestation service [26], establishing trust and enabling secure interaction with sensitive data even in untrusted environments [7]. Since TEEs provide a secure basis for protecting sensitive computation, they are well-suited technologies for enforcing usage control policies that ensure continuous regulation of data access and operations.

2.2 State of the art

The interest in secrecy- and privacy preservation for process mining has been concretized in different research outputs over the last few years. A body of solutions focus on the preprocessing of input data to shield unintentional information revealing. PRIPEL [16] anonymizes contextual information in event logs by applying differential privacy principles. Fahrenkrog et al. [17] later proposed BF-PRETSA, a sanitization method that preserves control-flow semantics while enforcing k-anonymity and t-closeness, aiming to maintain utility despite structural data modifications. Batista and Solanas present u-PPPM [6], a privacy-preserving process mining technique that addresses privacy threats stemming from distributional and location-based attacks by uniformizing event distributions to limit re-identification risks, while minimizing information loss. Although these techniques enhance privacy by transforming the data, they alter the event log and may compromise the accuracy and interpretability of the discovered models. In contrast, ProMiSe operates on unmodified logs, enforcing compliance through policies that govern data access, processing, and output. Another line of research investigates software and hardware-backed protocols to support cross-organizational process mining while preventing any exposure of the underlying data, revealing only the outcome of the computation. Elkumy et al. [15] leverage Secure Multi-Party Computation to allow different parties to jointly discover process models while keeping data confidential. Recent work leveraging TEEs for secure process mining [5,18,21] demonstrates how mining can be performed on sensitive data without prior anonymization. However, these solutions focus mainly on secure computation and do not address policy-based control over input and output usage or access rights, thus overlooking end-to-end governance. ProMiSe extends this paradigm by integrating policy enforcement across the full process discovery lifecycle, including input control, algorithm selection, and output restrictions.

3 Motivating scenario and requirements

To motivate our research, we introduce a use case scenario inspired by a known case in process mining for healthcare [20]. Hospitals and other entities, such as Italy's

Table 1: GDPR Based Requirements for a trusted process discovery system

ID	Requirement	GDPR Chapter
R1	**Lawful processing with restricted purposes.** Personal data must be processed lawfully, for defined purposes, and in a transparent manner respecting user consent.	2
R2	**Data minimization and retention.** Access to and retention of data must be limited to what is strictly necessary for the defined purpose and duration.	3, 8
R3	**Security, protection by design, and accountability.** The system must protect data by default through technical and organizational measures ensuring confidentiality and integrity. It must also enable demonstration of compliance with the GDPR through comprehensive logging, traceability, and auditability mechanisms	4
R4	**Controlled data transfer and jurisdictional awareness**. Personal data must only be transferred to third countries or international organizations if appropriate safeguards are in place or specific derogations apply. The system must support geofencing and jurisdiction based policy enforcement.	5
R5	**Facilitate supervisory.** The system must support the export of data protection audit logs and policy definitions in a standardized, human readable format to aid supervisory authority reviews and ensure transparency for compliance inspections.	6
R6	**Regulatory cooperation in cross border contexts.** The system must support documentation and traceability mechanisms that facilitate cooperation between supervisory authorities in different countries by ensuring consistent application of GDPR requirements in cross border processing scenarios.	7
R7	**Sensitive processing adaptation.** When processing occurs in special contexts such as scientific research, the system must ensure that it is not possible to reconstruct the profiles of data owners.	9

National Agency for Regional Health Services (AGENAS), the United States' Joint Commission on Accreditation of Healthcare Organizations (JCAHO), and India's National Accreditation Board for Hospitals & Healthcare Providers (NABH)[4] play a critical role in assessing the quality, safety, and efficiency of healthcare processes. These organizations support continuous improvement through standardized evaluations and data driven performance analysis. In our running example, we consider a hospital specialized in the treatment of sepsis that aims to collaborate with an external provider of process discovery services (henceforth, ProcessLock), to analyze and improve its clinical workflows. While outsourcing to ProcessLock could bring valuable expertise and computational resources, it raises significant concerns regarding the confidentiality and protection of highly sensitive medical data. Medical event logs contain detailed records, including timestamps of clinical activities, laboratory test results, admissions and discharges, diagnostic procedures, and personal information of patients and medical staff. Moreover, ensuring secure and controlled data sharing among trusted stakeholders, such as cooperating hospitals and healthcare quality assurance bodies like AGENAS, NABH, and JCAHO, is essential to enable cooperative evaluation and continuous clinical improvement.

Table 1 summarizes the set of requirements we derived from the General Data Protection Regulation (GDPR), by addressing all chapters that introduce obligations pertaining to a process mining context. They reflect the core regulatory principles, such as lawful and transparent processing, data minimization, security, and accountability, aimed at ensuring the protection of users' rights and interests, which we aim to fulfill with our solution. Next, we individually contextualize them in our motivating scenario.

When handling sensitive data, such as healthcare information, the risk of misuse is high if processing is allowed without restrictions on its intended purpose. Req. R1 is needed to prevent uncontrolled or secondary uses of personal data, ensuring that processing remains tied to the original, legitimate intent defined by the data owner.

[4] AGENAS: www.agenas.gov.it, JCAHO: www.jointcommission.org; NABH: nabh.co. Accessed: 29/09/2025

Example 1 (Req. R1: Purpose-restricted process discovery in sepsis care). A hospital may share event logs exclusively to analyze delays in antibiotic administration through process discovery. To prevent unintended uses, other analyses are excluded.

Healthcare data often includes large volumes of information, much of which may be irrelevant for a given analysis. Without careful limitation, there is a significant risk of overexposure or unnecessary retention. Req. R2 addresses the problem of unmotivated access and long-term storage by setting boundaries to what is accessed and for how long.

Example 2 (Req. R2: Minimized and time-bound access to sepsis treatment data). To analyze delays in antibiotic administration, only essential data, such as timestamps, lab results, and drug administration records should be accessed. All other data must be excluded, and retention limited to prevent overexposure and unnecessary storage.

Given the sensitivity of medical logs and the involvement of external service providers, ensuring the confidentiality, integrity, and traceability of data processing is critical. Req. R3 is essential to prevent unauthorized access and to support accountability through verifiable evidence of who accessed which data and under what conditions.

Example 3. The hospital shares event logs covering a 9 day period with ProcessLock for the analysis of sepsis workflows. Access is restricted to authorized hospital staff and AGENAS. All data interactions are securely logged, enabling both the hospital and AGENAS to verify who accessed the data, when, and for what processing purpose.

When data is shared across borders, differences in legal protections may expose individuals to greater privacy risks. Req. R4 addresses this by requiring control mechanisms on the location from which data are accessed.

Example 4 (Req. R4: Geographic restriction of data access). The hospital requires that all processing of sepsis event logs by ProcessLock take place within Italy, prohibiting access to user located abroad.

In regulated environments like healthcare, organizations must be able to prove compliance when requested by authorities. Without accessible documentation and records, oversight becomes ineffective. Req. R5 ensures that policy definitions and audit trails can be reviewed in case of inspections or investigations.

Example 5 (Req. R5: Transparent documentation for regulatory inspection). During an inspection by AGENAS, the hospital exports usage policies and access logs showing how sepsis event data were processed, enabling verification of compliance with national healthcare regulations.

For collaborative analysis, the system should support documentation and evidence that can be shared between supervisors and auditors, to help them resolve any differences in regulatory interpretation. Req. R6 is necessary to maintain consistency in such settings, preventing conflicts between differing legal expectations.

Example 6 (Req. R6: Facilitating multi-authority cooperation in joint processing scenarios). An Italian hospital participates in a cross-border sepsis research initiative involving hospitals in India. Event logs are shared among participants to enable process discovery across institutions. To ensure consistent regulatory interpretation, the hospital provides all research partners, including oversight bodies such as AGENAS and NABH, with structured, human-readable policy documentation.

Table 2: Non-functional requirements

ID	Requirement
NR1	**Memory consumption.** The system must optimize the resources available to prevent memory depletion during the execution of tasks [14, 18].
NR2	**Runtime overhead.** The system shall ensure that the runtime overhead is minimized compared to baseline execution without security mechanisms [10, 25, 27].
NR3	**Scalability.** The system shall efficiently handle multiple concurrent clients by maintaining continuous service availability [10, 11].

Medical information requires special protection due to its sensitivity and potential impact on individuals. In contexts such as research, where data is reused or aggregated, it is essential to implement strict safeguards. Req. R7 specifically addresses the need to prevent re-identification and unintended disclosures while still enabling valuable analysis.

Example 7 (Req. R7: Privacy-preserving use of clinical logs in sepsis research). The hospital collaborates with other institutions to study the impact of variations in antibiotic administration timing on sepsis outcomes. To support this research, process discovery techniques are applied to identify treatment patterns. Access to the data is strictly limited to prevent the exposure of identifying information related to patients or medical staff.

To ensure that the system remains safe and practically usable, we define a set of non-functional requirements based on evaluation practices for systems that leverage TEEs. These requirements are defined from the perspective of the service provider, focusing on aspects that impact system performance, scalability, and robustness. Table 2 shows the non-functional requirements defined for this paper. The requirements have been defined on the basis of commonly adopted assessment criteria reflecting concerns arising in the practical deployment of safe applications. Due to the limited memory capacity allocated to each TA within TEEs, surpassing this limit may lead to system failures and service disruptions or break confidentiality guarantees due to the use of external memory. It is thus essential that the system be optimized to complete the required operations within available resources. Therefore, the objective of NR1 is to assess the capability of performing process discovery operations within a TEE by verifying, on real-world logs, whether the required executions can complete successfully without exceeding the memory available to the enclaves. Req. NR2 focuses on minimizing runtime overhead, as enclaves introduce performance penalties due to restricted system calls, alongside encryption and decryption operations in memory, which are essential to ensure a secure and isolated execution environment [10, 25, 27]. Since ProMiSe is intended to operate as a continuous service for process discovery, excessive memory usage can threaten its availability. The scalability assessment therefore aims to evaluate the system's ability to maintain stable operation and predictable performance under increasing levels of concurrent use [10, 27]. Req. NR3 addresses the need to ensure that resource consumption, particularly memory, remains within acceptable bounds, while preserving sufficient responsiveness to support realistic multi-user scenarios without compromising overall system reliability.

Example 8 (Reqs. NR1 to NR3: Performance and scalability requirements). The hospital submits sepsis treatment event logs to ProcessLock for process discovery. ProcessLock is required to execute discovery algorithms within the memory constraints of the enclave and within acceptable processing times. Furthermore, as the hospital authorizes multiple external stakeholders to access the results, the system must be scalable to handle concurrent requests without degrading performance or availability.

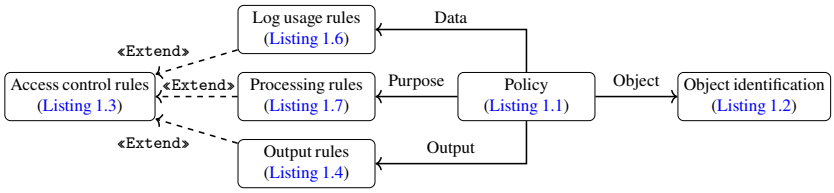


Fig. 1: Policy structure

4 Conceptual Design

In this section, we describe ProMiSe's policies, architecture, and internal data flow.

4.1 Policy

ProMiSe usage policies are modeled using RDF. To align the usage policies with the motivating scenario of the paper, we developed a domain specific policy language tailored to process mining. Our goal is to model the fundamental elements of process models and discovery directly within the policy structure, enabling precise and context-aware policy enforcement.

Figure 1 illustrates the structure of the policies adopted in our approach. Each policy consists of four main elements, which collectively regulate the secure and compliant use of event logs within the trusted environment. (i) The **Object Identification** element defines the data subject to the policy, specifying the event log along with its associated metadata, such as filename and format. (ii) The **Log Usage Rules** element specifies the conditions under which the event log may be accessed and used. These rules include constraints such as log expiration, geographic access restrictions, exclusion of sensitive attributes, permitted time windows for analysis, constraints on event inclusion or exclusion, and limits on the number of authorized accesses. (iii) The **Processing Rules** define which process discovery techniques and algorithms are permitted, thereby restricting a third-party platform to execute only authorized forms of analysis within the trusted environment. (iv) The **Output Rules** govern the handling of analysis results, specifying constraints on their accessibility, temporal validity, and geographical availability, thereby ensuring controlled dissemination and use of derived insights. The prefixes specified in the following example apply to all examples presented in this section.

```
@prefix ucon: <http://example.org/ucon#> .
@prefix eventLog: <http://example.org/eventLog#> .
@prefix pmt: <http://example.org/pmt#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix loc: <http://id.loc.gov/vocabulary/countries/> .
ucon:objectId [...];
ucon:logUsageRules [...];
ucon:processingRules [...];
ucon:outputRules [...];
```

Listing 1.1: Example of the policy structure

To identify the object to which the policy applies, the object identification element includes two properties: `fileName`, and `format`, indicating the name and format of the event log file respectively.

```
eventLog:fileName "eventLog.xes"^^xsd:string;
eventLog:format "XES";
```

Listing 1.2: The event log file named "eventLog.xes" is provided in XES format

Users can specify the identities authorized to access each phase, as well as the geographic locations from which access is permitted, within the three elements of the process discovery lifecycle: `logUsageRules`, `processingRules`, and `outputRules`.

```
ucon:allowedLocations loc:it;
ucon:accessControlRules "AGENAS";
```

Listing 1.3: Only AGENAS agents in Italy have the permission to access the event log

The policy also allows users to define expiration times and access limits for both the event log and the output models generated during processing. This can be achieved by specifying the properties "`ucon:expiration`" and "`ucon:maxAccessCount`", as illustrated in the following example.

```
ucon:expiration "2025-12-31T23:59:59Z"^^xsd:dateTime;
ucon:maxAccessCount "9"^^xsd:integer;
```

Listing 1.4: Log expiration set to the end of the year 2025, with at most 9 accesses

To impose temporal constraints on the event log, the policy may include the "`allowedTimeRange`" property. This property specifies a start and end date, thereby restricting the events considered during processing to those occurring within the defined interval. Additionally, it allows the user to indicate the specific event attribute that contains the timestamp, enabling accurate identification of the temporal dimension within the log.

```
ucon:allowedTimeRange [
    ucon:eventAttribute "time:timestamp";
    ucon:startDate "2020-01-01T00:00:00Z"^^xsd:dateTime;
    ucon:endDate "2025-01-01T00:00:00Z"^^xsd:dateTime ];
```

Listing 1.5: Only events timestamped between 2020 and 2025 can be accessed

The user, defining the "`logConstraints`" property, in the "`logUsageRules`" elements, defines conditions based on a specific event attribute (specified by the "`ucon:eventAttribute`" property) to determine which traces are included in the analysis. It requires that traces contain events with attribute values listed in "`mustInclude`" property and exclude traces containing events with values listed in "`mustExclude`" property.

```
ucon:logConstraints [
    ucon:eventAttribute "concept:name";
    ucon:mustInclude ( "ER Triage" );
    ucon:mustExclude ( "Leucocytes" "IV Antibiotics" ) ];
```

Listing 1.6: Include events with value "ER Triage" and exclude those with values "Leucocytes" and "IV Antibiotics."

Finally, within the "`processingRules`" element, the user can specify the types of operations permitted on the data. This is achieved by defining the "`allowedTechnique`" property, which contains one or more specifications of mining techniques. Each technique is described using two internal properties: "`pmt:techniqueType`", indicating the category of the technique, and "`pmt:algorithm`", specifying the exact algorithm allowed.

```
ucon:allowedTechnique (
    [ pmt:techniqueType pmt:AutomatedDiscovery;
        pmt:algorithm pmt:HeuristicMiner ],
    [ pmt:techniqueType pmt:AutomatedDiscovery;
        pmt:algorithm pmt:AlphaMiner ] )
```

Listing 1.7: Only the HeuristicMiner and AlphaMiner discovery algorithms are allowed

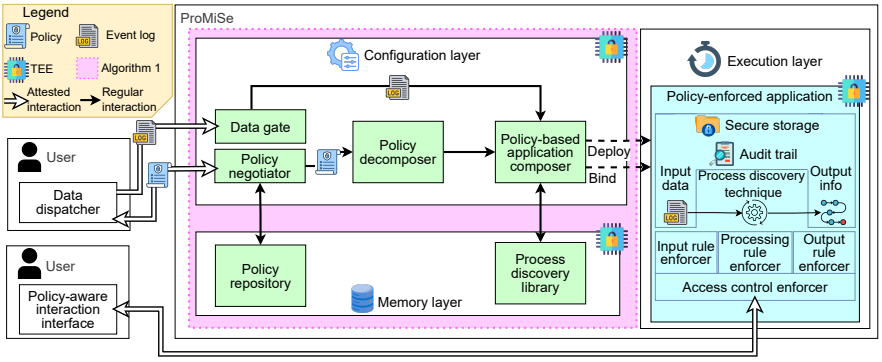


Fig. 2: An overview of the ProMiSe architecture

4.2 Architecture

Figure 2 provides an overview of the architecture of ProMiSe. We assume that a user can play two distinct roles: data provider or data consumer. When acting as a data provider, the user initiates the process by requesting the setup of a secure environment for conducting process discovery analysis within the ProMiSe system. To this end, the user submits to ProMiSe both the event log intended for analysis and the corresponding usage policy, which specifies the constraints and conditions under which the data may be accessed and processed. When acting as a data consumer, the user submits a request to ProMiSe to perform an operation on an event log that has already been provided and initialized by a data provider within a secure environment. To initiate such a request, the data consumer must first authenticate. ProMiSe then evaluates whether the requested operation is permitted by verifying compliance with the usage policy associated with the data. If all policy conditions are satisfied, the request is granted and the operation is executed, otherwise, access is denied. ProMiSe is structured into three main layers. The configuration layer is responsible for receiving input data, setting up the execution environment. The memory layer maintains copies of the policies submitted to ProMiSe as well as all known and applicable algorithms relevant to processing the data. The execution layer encompasses the generated execution environment where user data will be preserved, managed and processed. Since these layers handle sensitive user data, they operate entirely within a TEE. Consequently, all operations performed within these layers utilize TEE resources, and all data stored therein are encrypted through TEE's mechanisms (see Sect. 2).The figure depicts two types of arrows: double arrows represent attested interactions, while black arrows indicate regular interactions. Attested interactions involve remote attestation procedures and occur at two critical stages. The first attestation takes place when the data provider submits data to the system, verifying the authenticity and integrity of the system components involved in the interaction, to ensure they have not been compromised. The second attestation occurs when the data consumer requests to access or process the data, and it targets the output produced. This step confirms that the operations were performed by a trusted and unmodified system, thus validating the integrity and trustworthiness of the result. ProMiSe receives data from the user acting as a data provider through two components: The *Data gate*, which processes the submitted data and forwards it to the *Policy-based application composer*, and the *Policy negotiator*, which performs preliminary checks on the policy to ensure it is syntactically correct and compatible with the system. If the check fails, a request for modification is

Algorithm 1: Generation of policy-enforced applications

```
   Input: policyFile (a file with usage policies), and logFile (a file with the event log)
   Output: tApp (a trusted application for policy enforcement)
1  triples ← parseRDF(policyFile);
2  phases ← ⟨"logUsageRules", "processingRules", "outputRules"⟩;
3  foreach phase ∈ phases do
4      rules[phase] ← extractPhase(triples, "ucon:" + phase);
5  policyID ← hash(triples);
6  tApp ← retieveTA(policyID);
7  if tApp is null then
8      enforcers ← generateEnforcers(rules);
9      tApp ← generateTApp(policyID, enforcers);
10     deployTApp(tApp);
11 bindEventLogToTApp(tApp, logFile);
12 return tApp;
```

issued to the data provider. Once the policy passes these checks, the *Policy negotiator* stores a copy in the *Policy repository* and forwards the policy to the *Policy decomposer*. The *Policy repository* leverages sealing within the Trusted Execution Environment, encrypting policies so that they can only be accessed and decrypted by authorized components. The *Policy decomposer* component is responsible for extracting the relevant information from the submitted policy. Once extracted, this information is passed to the *Policy-based application composer*, which is in charge of the generation of the enforcers from the policy and to compose the trusted application responsible for enforcing the policy on the data. This trusted application is the *Policy-enforced application*, and it encapsulates both the policy logic and the operational constraints. The *Policy-based application composer* receives the decomposed policy and first checks whether its structure matches that of an existing, previously processed policy. If a *Policy-enforced application* corresponding to the same policy has already been generated, the *Policy-based application composer* reuses the existing instance by binding the new event log to it, avoiding redundant trusted application creation.

Algorithm 1 operates with the components highlighted by a dotted pink area in Fig. 2 and details the procedure for generating the application. This algorithm takes as input the policy and the event log, parses the policy, and identifies the phases for which the data provider can define authorized behaviors. The phases are named `logUsageRules`, `processingRules`, and `outputRules`. Each set of rules is extracted (line 3), and a unique application identifier is computed by hashing the policy structure (line 5). This identifier is used to check whether an application with an identical rule configuration already exists (line 7). If no such application is found, the system generates the necessary enforcers, which are then used to generate the trusted application (line 8). Once built, the application is deployed (line 9-10) and the event log is stored within its secure storage (line 11). In case a matching application already exists, the system reuses it and associates the new event log with the corresponding trusted application. Otherwise, it proceeds to generate a new *Policy-enforced application* that embeds all the constraints specified by the policy and the event log. Generating a new application requires building and deploying it within the TEE to guarantee confidentiality and data integrity. For all the rules regarding which techniques may be applied, the relevant methods are retrieved from the *Process discovery library* and incorporates them into the application. The *Process discovery library* acts as a repository for ProMiSe, providing both domain knowledge and details about available mining techniques. Thus, we have shown how ProMiSe dynamically generates *Policy-enforced application*s that implement the data provider's specified rules. A *Policy-enforced application* includes a *Secure storage*, wherein input and output data are retained in a controlled manner to ensure compliance with the

Table 3: Dataset statistics

Name	Activities	Cases	Events		
			Max	Min	Avg.
Sepsis	16	1050	185	3	15
BPIC 2013	7	1487	123	1	9
BPIC 2012	36	13,087	75	3	20

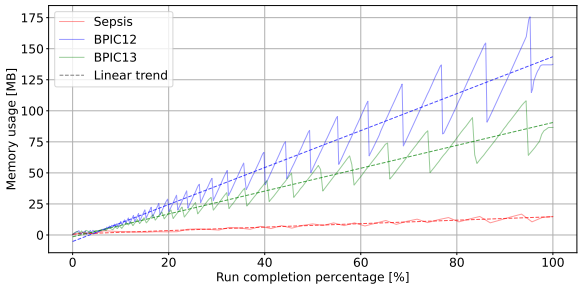


Fig. 3: Memory usage test

defined policies. In the *Secure storage* is also the audit trail file that each application has to show how the data are used by tracking all operations performed on them. The *Policy-enforced application* enforces rules specified by the data provider to govern the entire lifecycle of the shared data. These rules address the handling of input data, the processing logic applied during computation, and the management of output models generated after execution. At each stage (input, processing, and output), the *Policy-enforced application* applies access control mechanisms, if defined in the policy, to restrict access to authorized entities only.

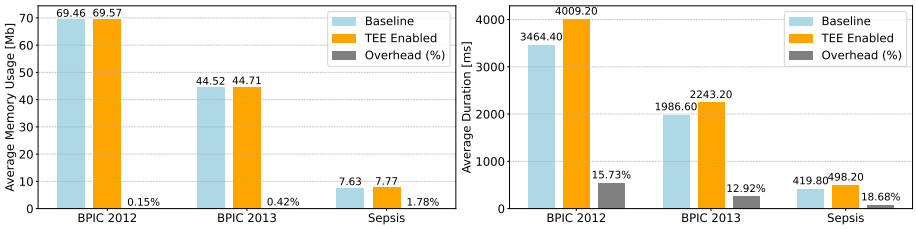
5 Evaluation

This section presents the evaluation of ProMiSe. We begin with a performance assessment to verify compliance with the non-functional requirements in Table 2 (Sects. 5.1 and 5.2). Then, we show how the system meets the functional requirements in Table 1 by design (Sect. 5.3).

5.1 Implementation and experimental setup

We implemented the components running within the TEE in Go using the EGo framework,deploying them within an Intel SGX TEE[5] on a machine equipped with an Intel Xeon Gold 5415+ CPU. The *Policy-enforced application*, representing the core services offered by ProMiSe, exposes a set of APIs to interact with user-submitted data. Communication with the application occurs via RPC-style calls, enabling remote invocation of only those functionalities explicitly exposed by the application. These reflect the rules defined in the corresponding policy. The remaining components of the system, including user interaction, policy management, and file handling, were developed in Python. The code is available at anonymous.4open.science/r/ProMise-816E. We used three real-world event logs summarized in Table 3. We performed performance tests on the execution of the process discovery algorithm, using the Heuristic Miner as a representative case due to its relatively high computational demands compared to other supported techniques. We employed a single policy for all logs, including the rules in Sect. 4.1 and encompassing user authentication, geographical access restrictions, log expiration, maximum access limits, and validation of the permitted processing technique. For this evaluation, a *Policy-enforced application* was generated based on it, and the above event logs were bound to it. We collected test data by issuing requests to the trusted application while triggering all relevant policy enforcement checks. We repeated every experiment five times per configuration.

[5]EGo: edgeless.systems/products/ego. Intel SGX: sgx101.gitbook.io/sgx101.



(a) Memory overhead test (b) Time overhead test

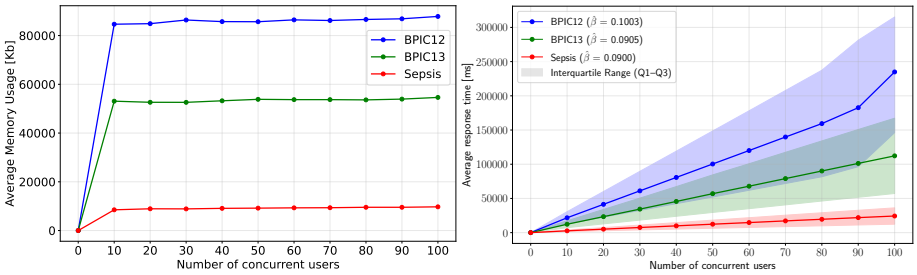
Fig. 4: Overhead test

5.2 Performance

We gauge the performance of our solution in terms of memory consumption (to address NR1), runtime overhead (NR2), and scalability (NR3).

Memory Consumption. Due to the limited memory available to each TA operating within a TEE, the experiment we describe next is of relevance to NR1. Figure 3 presents the results of the memory usage test performed during the execution of the process discovery algorithm. The plot illustrates the trend of memory consumption (in MB) as a function of the completion percentage of each test run. For all logs, memory usage exhibits a linear growth trend, with minor oscillations around the regression line that represent the actual consumption profile. The saw-tooth patterns observed in the curves reflect the activity of garbage collection mechanisms, which periodically release accumulated memory during execution. In addition, all the curves exhibit increasingly wide memory fluctuations as the execution progresses. We attribute this behavior to the standard garbage collection policy employed by the Go runtime environment, which is automatically triggered based on an adaptive memory threshold. The execution on the Sepsis log displays consistently low memory usage, remaining below $20\,\mathrm{MB}$ and showing a nearly flat growth curve, indicating minimal resource requirements. The execution with BPIC 2013 log demonstrates a moderate and constantly increasing memory footprint, peaking at approximately $90\,\mathrm{MB}$. In contrast, BPIC 2012, the largest and most complex log of the set, exhibits the highest memory demand, reaching up to $150\,\mathrm{MB}$ by the end of the execution. Despite this, the processing was successfully completed within the limits of the system used.

Overhead. This subsection assesses the runtime overhead incurred by executing within a TEE. The objective is to demonstrate that the enhanced security guarantees provided by the enclave impose minimal performance degradation relative to an unprotected baseline environment. The overhead evaluation addresses Req. NR2, which concerns the performance impact of executing within a TEE compared to a baseline implementation without the TEE support. Figure 4 summarizes the experiments conducted to measure system overhead in terms of execution time and memory consumption. As depicted in Fig. 4(a), memory usage during TEE execution remains close to that of the baseline, with only a slight increase consistently observed across all logs. Time overheads are minimal, ranging from $0.11\,\mathrm{MB}$ for the BPIC 2012 log to $0.19\,\mathrm{MB}$ for the Sepsis log, indicating that the performance cost introduced by TEE execution remains under control. Figure 4(b) presents the time overhead observed during the same test runs used to measure memory usage. In this case, the use of the TEE results in approximately a $16\,\%$ increase in execution time. This overhead primarily stems from the cryptographic operations performed by



(a) Scalability on average memory usage (b) Scalability on average response time

Fig. 5: Scalability tests

the TEE during memory access. Additionally, remote attestation mechanisms contribute to this, as they require the TEE to generate a report based on the output.

Scalability. Since ProMise is intended to operate as a continuous service, we evaluate the scalability of our solution, thus addressing NR3. Our primary objective remains to avoid exceeding TEE memory limits, which can represent a serious threat to service continuity. As stated in Sect. 4, we integrate a queue-based mechanism within the *Policy-enforced application* to counteract possible transient memory overloads by serializing requests.

Figure 5 illustrates the results of the scalability tests. We simulated concurrent invocations of the same Trusted Application by increasing the number of parallel user calls from 10 to 100, by increments of 10. Figure 5(a) illustrates the scalability of the system in terms of average memory usage under concurrent execution. In all the test configurations, the memory consumption presents stable trend. The average memory usage remains approximately 1000 KB for Sepsis, around 5500 KB for BPIC 2013, and slightly above 8000 KB for BPIC 2012, consistently staying below 9000 KB. These results confirm that the system maintains predictable memory usage even under high concurrency. This queue-based strategy can affect responsiveness in favor of service availability. To quantify this trade-off, we also analyze the response latency induced by multiple concurrent processing requests. Figure 5(b) reports the average waiting time per test. For each curve, we shaded the area between the first and third quartiles using a gradient of the same color, representing the central distribution range of the observed waiting times. The results show a linear increase in waiting time across all logs tested. To quantify the slope of each growth trend, we computed the average $\hat{\beta}$ by fitting a linear regression over response times normalized to the initial value. In this way, each $\hat{\beta}$ expresses the average relative increase (in decimal form) in response time per additional concurrent user. This global trend estimation enables a consistent evaluation and comparison of the scalability behavior across the different logs. The executions on Sepsis and BPIC 2013 exhibit an average relative increase in response time of about 9 % ($\hat{\beta} = 0.09$ and $\hat{\beta} = 0.0905$, respectively) per additional concurrent user, whereas BPIC 2012 shows a slightly higher value of 10.03 % ($\hat{\beta} = 0.1003$). These results confirm that the queue-based implementation, while effective in controlling memory usage, contributes to increased latency under high concurrency, underlining the need for further optimizations to maintain responsiveness in concurrent scenarios.

5.3 Faithfulness

In this section, we discuss how ProMise fulfills the requirements by design, relying on its architecture and the technologies it adopts. As described in the design section, *Policy-enforced*

*application*s are generated based on a submitted policy. Thanks to this close link between policy and the trusted application, the constraints defined by the data owner are enforced, ensuring that only the operations explicitly allowed in the policy can be performed on the data. This design enables compliance with **R1**, since the RPC-based implementation of the generated trusted application exposes to users only the functionalities permitted for that specific event log. Likewise, the system satisfies **R2**, as the policy can specify which data attributes are not accessible and define retention rules such as expiration and maximum number of accesses. Each time a request is made, these rules are taken into account, along with the user's identity and the geographic location from which the request originates (as required by **R4**) in order to grant or deny access. By enabling control over the entire data processing flow, including input, processing, and output, ProMise also ensures compliance with **R7**, limiting the exposure of sensitive information produced by process discovery techniques. Furthermore, ProMise addresses **R3** through encryption and decryption mechanisms and remote attestation, ensuring high confidentiality on the data and verifiability on the environment that execute and store it. ProMise also facilitates cooperation across jurisdictions by providing, at any time, a structured version of the rules that will be enforced by the application upon data usage requests, in line with **R6**. Finally, thanks to integrated auditing and traceability mechanisms, every operation and request, along with its outcome and all submitted information, is recorded and stored in a retrievable audit trail file, ensuring compliance with **R5**.

6 Conclusion and Future Work

In this paper, we presented ProMiSe, a policy enforcement framework implemented as a service to support process discovery. By enabling data owners to specify fine-grained usage policies, ProMiSe enforces strict control over data access, processing, and output, ensuring adherence to predefined constraints. Our future work will focus on four main directions. First, we aim to verify the extensibility of the defined policy language, evaluating its ability to support additional constraints and evolving requirements over time. Secondly, we plan to integrate automated policy verification and validation mechanisms, ensuring that any data submitted by the data owner complies with the expected constraints and usage rules. We intend to expand the scope of supported process mining tasks, moving beyond process discovery to include additional techniques such as conformance checking, performance mining, and variant analysis, in order to offer broader analytical capabilities. Finally, based on the results of the scalability tests, which revealed increased latency under high concurrency, we will improve request management through two complementary strategies. The first involves implementing a load balancing mechanism across multiple instances of the trusted application, in order to achieve better request distribution, preserve memory isolation, and enhance overall scalability. The second consists in supporting the processing of large log files by dividing them into smaller chunks. This strategy enables the trusted application to handle extensive datasets without exceeding the memory limitations imposed by the secure execution environment.

References

1. IEEE Standard for eXtensible Event Stream (XES) for achieving interoperability in event logs and event streams (Nov 2023). https://doi.org/10.1109/IEEESTD.2023.10267858

2. van der Aalst, W.M.P., Carmona, J. (eds.): Process Mining Handbook. Springer (2022)
3. Akaichi, I., Kirrane, S.: Usage control specification, enforcement, and robustness: A survey. arXiv preprint arXiv:2203.04800 (2022)
4. Anati, I., Gueron, S., Johnson, S., et al.: Innovative technology for cpu based attestation and sealing. In: Proceedings of the 2nd international workshop on HASP. vol. 13 (2013)
5. Basile, D., Di Ciccio, C.: Secrecy preservation for online process monitoring with trusted execution environment. In: BPM. pp. 235–254 (2025)
6. Batista, E., Solanas, A.: A uniformization-based approach to preserve individuals' privacy during process mining analyses. P2P Networking and Applications **14**(3), 1500–1519 (2021)
7. Birkholz, H., Thaler, D., Richardson, M., et al.: Rfc 9334: Remote attestation procedures (rats) architecture (2023)
8. Borovits, N., Bardelloni, G., Tamburri, D.A., van den Heuvel, W.: Anonymization-as-a-service: The service center transcripts industrial case. In: ICSOC. pp. 261–275 (2023)
9. Burattin, A., Conti, M., Turato, D.: Toward an anonymous process mining. In: FiCloud. pp. 58–63. IEEE Computer Society (2015)
10. Chen, L., Li, J., Ma, R., et al.: Enclavecache: A secure and scalable key-value cache in multi-tenant clouds using intel sgx. In: Middleware. pp. 14–27 (2019)
11. Chouksey, A., et al.: Oblix: An efficient oblivious search index. In: IEEE Symposium on Security and Privacy (SP) (2020)
12. De Weerdt, J., Wynn, M.T.: Foundations of process event data. In: van der Aalst and Carmona [2], pp. 193–211
13. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of business process management. Springer (2013)
14. El-Hindi, M., Ziegler, T., Heinrich, M., et al.: Benchmarking the second generation of intel sgx hardware. In: Proceedings of the 18th International Workshop on DaMoN. pp. 1–8 (2022)
15. Elkoumy, G., Fahrenkrog-Petersen, S.A., Dumas, M., et al.: Secure multi-party computation for inter-organizational process mining. In: BPMDS 2020. vol. 387, pp. 166–181 (2020)
16. Fahrenkrog-Petersen, S.A., van der Aa, H., Weidlich, M.: PRIPEL: privacy-preserving event log publishing including contextual information. In: BPM 2020 (2020)
17. Fahrenkrog-Petersen, S.A., van der Aa, H., Weidlich, M.: Optimal event log sanitization for privacy-preserving process mining. Data & Knowledge Engineering **145**, 102175 (2023)
18. Goretti, V., Basile, D., Barbaro, L., et al.: Trusted execution environment for decentralized process mining. In: CAiSE. pp. 509–527. Springer (2024)
19. Kirchmann, H., Fahrenkrog-Petersen, S.A., Mannhardt, F., et al.: Control-flow reconstruction attacks on business process models. In: EDOC. pp. 60–77. Springer (2024)
20. Mannhardt, F.: Sepsis cases – event log (2016). https://doi.org/10.4121/UUID:915D2BFB-7E84-49AD-A286-DC35F063A460
21. Müller, M., Simonet-Boulogne, A., Sengupta, S., et al.: Process mining in trusted execution environments: Towards hardware guarantees for trust-aware inter-organizational process analysis. In: ICPM Workshops. pp. 369–381 (2021)
22. Park, J., Sandhu, R.: The UCONABC usage control model. ACM transactions on information and system security (TISSEC) **7**(1), 128–174 (2004)
23. Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: What it is, and what it is not. In: IEEE Trustcom/BigDataSE/Ispa. vol. 1, pp. 57–64 (2015)
24. Schönig, S., Di Ciccio, C., Maggi, F.M., Mendling, J.: Discovery of multi-perspective declarative process models. In: ICSOC. pp. 87–103 (2016)
25. Shen, Y., Tian, H., Chen, Y., et al.: Occlum: Secure and efficient multitasking inside a single enclave of intel sgx. In: Proceedings of the ASPLOS 25. pp. 955–970 (2020)
26. Wang, J., Hong, Z., Zhang, Y., Jin, Y.: Enabling security-enhanced attestation with intel sgx for remote terminal and iot. IEEE TCAD **37**(1), 88–96 (2017)
27. Weichbrodt, N., Aublin, P.L., Kapitza, R.: sgx-perf: A performance analysis tool for Intel SGX enclaves. In: Middleware. pp. 201–213 (2018)