Secrecy Preservation for Online Process Monitoring with Trusted Execution Environment

Davide Basile[1] and Claudio Di Ciccio[2]

[1] Sapienza University of Rome, Italy, davide.basile@uniroma1.it
[2] Utrecht University, Netherlands, c.diciccio@uu.nl

Abstract. The online monitoring of collaborative business processes involves the integration of event data sourced from various information systems into a unified process state. The transmission of process records across organizational boundaries poses significant data secrecy and security challenges. In this context, organizations may be reluctant to outsource sensitive data with their collaborators to update a global process state. In this paper, we propose ProMTEE, a novel framework designed to tackle these challenges by resorting to trusted applications running in Trusted Execution Environments (TEEs). To showcase the potential of ProMTEE, we contextualize its four-staged interaction protocol within the realm of two online monitoring tasks, namely, control flow tracking and compliance monitoring. We verify the security of our solution through a threat analysis against a set of security requirements derived from a supply-chain scenario. In addition, we conduct an experimental assessment of our proof-of-concept implementation with tests on memory usage and latency using real-world datasets.

Keywords: Collaborative information systems · Process-aware information systems · Business Process Management · TEE · Confidential Computing

1 Introduction

Through online process monitoring, organizations gain valuable real-time insights into the health status of running process instances with respect to reference models, performance metrics, and operational objectives [5]. Traditional online monitoring techniques focus on mono-organizational settings, in which a single firm or institute monitors the events of its internal business processes via proprietary information systems. However, in domains such as supply chain management, healthcare, and manufacturing, collaborative business processes are by now a standard setting [17]. To gain an exhaustive understanding of the global business process, parties need to transmit process execution data beyond their organizational boundaries and aggregate this information into a comprehensive view. In this context, preventing sensitive information from being disclosed becomes a primary requirement [13]. Organizations may hesitate to expose business-critical information to safeguard competitive advantages and uphold customer trust. Also, data privacy regulations such as the European GDPR[3] exert legal constraints on data treatment.

[3] European Parliament, Council of EU: Regulation 2016/679, data.europa.eu/eli/reg/2016/679/oj

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-02867-9_15

Secrecy concerns pose significant challenges for collaborative process monitoring. Most research in the BPM domain addresses secrecy preservation through pre-elaboration techniques which obfuscate sensitive data in event logs to prevent information reconstruction [7,8,2,6]. While effective for offline analysis, these methods are computationally intensive (thus determining processing delays) and introduce noise into the original data (increasing the risk of observing non-existing behavior). Recent approaches leverage hardware-backed solutions for secrecy-preserving offline analyses directly on original data [9,18]. Although mitigating the data distortion issue, these solutions remain suited for ex-post analysis, causing non-negligible delays when applied to online settings: if anything, monitoring tasks can only begin when a process instance terminates.

To address these limitations, we extend the state of the art by introducing ProMTEE, a novel framework for online process monitoring with guarantees on the secrecy of the shared process events. We contextualize ProMTEE within two intra-instance monitoring tasks [5,20]: control flow tracking (i.e., monitoring the process instance progression in the control flow) and compliance monitoring (i.e., monitoring the fulfillment of business rules within process instances). Our solution resorts to *trusted applications* operating within Trusted Execution Environments (TEEs), namely hardware-secured execution contexts ensuring data secrecy and code integrity [22]. ProMTEE supports a four-staged interaction protocol enabling trusted applications to receive events and update their view of the process state. As a result, organizations can securely share process execution data and access the global process state, representing the condition of the entire collaborative process, while the original private data of individual events remains encrypted within TEEs. We pursue fast responsiveness with respect to process execution (as is expected in a runtime setting) and low memory consumption (to redress the constrained capacity of TEEs). We evaluate the security of ProMTEE by verifying the fulfillment of four secrecy-related requirements derived from a supply chain scenario. Also, we validate our prototype implementation through experiments observing memory usage and latency.

The remainder of the paper is as follows. In Sect. 2, we set our objectives against the state of the art. Section 3 describes the requirements we aim to meet, within a motivating scenario. Section 4 details the design and deployment of ProMTEE. Section 5 focuses on our four-staged interaction protocol. Section 6 evaluates security and performance of our approach. Section 7 concludes the paper outlining limitations and future work.

2 Objectives and State of the Art

Our work firstly pursues the objective of preserving **secrecy** (**O1**) in process data treatment. Several methodologies addressing data confidentiality propose pre-processing techniques that obfuscate sensitive information within process data before transmitting it to untrusted entities. Fahrenkrog-Petersen et al. [8] introduce PRETSA, a family of prefix-based log sanitization algorithms designed to achieve k-anonymity and t-closeness. Similarly, Batista et al. [2] conceptualize a microaggregation approach to mitigate re-identification risks. PRIPEL, introduced in [7], is a three-step algorithm to anonymize contextual information in event logs by applying differential privacy principles. Elkoumy et al. [6] propose a method based on Secure Multi-Party Computation to produce pre-elaborations of Directly-Follows Graphs from event logs, adding Laplacian noise to protect privacy

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-02867-9_15

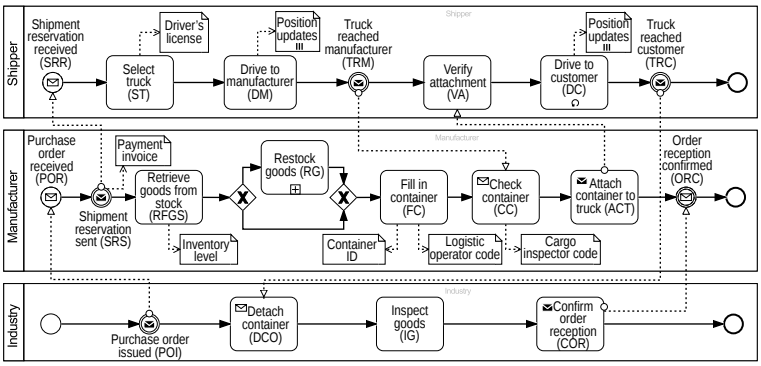


Fig. 1: The BPMN diagram of a collaborative process in the supply-chain domain

before transmission. These methodologies offer effective solutions for aggregation and generalization in offline settings. However, their computational complexity can significantly impact the responsiveness of monitoring activities, resulting in considerable lags between the generation of a process event and its computation. Moreover, the noise addition of these techniques makes them hardly applicable to case-level monitoring, as non-existing behavior may be observed. Therefore, our approach aims to yield Obj. O1 yet processing **unaltered data** (**O2**).

Existing contributions in the field of BPM propose approaches based on Trusted Execution Environments (TEEs) to process original event data in a secured computing vault. The seminal work of Müller et al. [18] presents a hybrid architecture integrating TEEs and blockchain technologies to execute verified mining tasks within centralized trusted applications. Goretti et al. [9] introduce CONFINE, a TEE-based framework for secrecy-preserving process mining in decentralized environments. Their research defines an interaction protocol that enables the secure transmission, combination, and processing of event logs within trusted applications. These works provide solutions for executing process mining techniques on original, unaltered event logs while minimizing the risk of information leakage. However, they are designed for off-line processing and require the transmission of completed cases. As a result, applying these approaches to online process monitoring would introduce significant lags –after all, data could be processed only upon the conclusion of one or more cases. In this paper, we pursue the objective of achieving **fast responsiveness** (**O3**) to ongoing process executions, in addition to Objs. O1 and O2.

The aforementioned objectives drive our design choices and evaluation criteria. In the next section, we will provide a motivating scenario, which will serve as a running example and to illustrate the security requirements stemming from Obj. O1.

3 Motivating Scenario and Security Requirements

Our running example is inspired by supply chain motivating scenarios in the BPM field [25,23,14]. Figure 1 and Tab. 1 present the collaborative process as a BPMN diagram and its corresponding regulatory business rules, respectively.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-02867-9_15

Table 1: Business constraints in the supply-chain scenario of Fig. 1

ID	Informal description	Activities	Data involved
C1	The selected driver must have five years of experience in exceptional transport	ST	Driver's license
C2	The cost of the shipment in euro is up to 3 times the total distance in kilometers	RS, DM, DC	Payment invoice Position updates
C3	The logistics operator and the cargo inspector must differ	FC, CC	Logistics operator code Cargo inspector code
C4	The inspection of goods must happen within one hour from the delivery	DCO, IG	Timestamps
C5	A restock must be executed when the good units are below 1000	RGFS, RG	Inventory level

Table 2: Security requirements of ProMTEE

ID	Requirement description	GDPR[3] articles
R1	Monitors must prove their trustworthiness to receive and process event data	24, 30
R2	Exchanged events should be hidden from external observers during transmission	32
R3	Parties should be able to monitor process executions without accessing sensitive data in single events	25
R4	Process monitors must remain pristine in the course of monitoring	5

It begins with an industry issuing a purchase order (POI) to the manufacturer (POR), who then reserves a shipment (SRS), registering the payment invoice. Upon confirmation of the reservation (SRR), the shipper selects a truck (ST) recording the driver's license data. The selected driver must have five years of experience in exceptional transport (Rule C1 in Tab. 1). The shipper's truck drives to the manufacturer (DM), providing real-time information on its position. Meanwhile, the manufacturer retrieves goods from stock (RGFS), registering its inventory status. If stock levels are below 1000 units, a restocking (RG) must be triggered (Rule C5). Then, the manufacturer fills the container (FC) recording the code of the logistics operator who carried the activity. When the truck arrives (TRM), the manufacturer checks (CC) and attaches the container (ACT), documenting its cargo inspector's code. The logistics operator and the cargo inspector's codes must always differ (Rule C3). The courier verifies the attachment (VA) before driving to the customer (DC) and reaching it (TRC). The final cost of the shipment specified in the initial invoice must be up to 3 times the total distance in km of the position updates (Rule C2). The industry then detaches the container (DCO), inspects the goods (IG), and confirms their receipt (COR). Therefore, the timestamp of IG must be within one hour from TRC (Rule C4).

As is typical of supply chain scenarios [11], organizations enter into a contract with an insurance company to cover additional expenses in the case of disruptions. All parties, including the insurance company, monitor the status of each process instance and verify that the execution of activities adheres to the predefined standard [5]. To this end, they are also granted access to the process specifications (see Fig. 1 and Tab. 1).

However, monitoring requires the processing of data records that retain sensitive information. Therefore, organizations are reluctant to openly share them with the other parties [13]. From Obj. O1, we thus establish four related security requirements (Reqs. R1 to R4), which we root in GDPR regulations.[3] Table 2 provides a brief description for each of those, alongside references to the related GDPR articles.(R1) Monitoring parties must prove trustworthiness; for example, the shipper should verify how the insurance company handles its data, including the applied control-flow tracking technique, the

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-02867-9_15

logic of the monitored rules, and any up/downstream manipulation. (R2) In addition, data must be safeguarded from external observers to prevent any information leakage during transmission. (R3) Each organization should be able to monitor process executions, such as the fulfillment status of Rule C3, without directly accessing sensitive data of individual events from other organizations (e.g., the logistics operator and inspector's code). (R4) Finally, the behaviour of monitoring entities (e.g., the verification mechanism of Rule C3) must remain unaffected by malicious alterations, ensuring that reported information reflects the actual process state. Our research aims to enable all involved organizations to monitor the process state reflecting the operations carried out by all the other parties, and to ensure compliance with the above requirements.

4 Design

ProMTEE aims to (O2) enable organizations to share process execution records with external monitors while (O1) minimizing the risk of disclosing sensitive information. To operate on process data in real-time (O3) while meeting Objs. O1 and O2, we introduce the Process Vault, a dedicated verifiable, protected, and tamper-proof component. Hereafter, we provide a *functional viewpoint* [21] of ProMTEE applied in the context of two key monitoring tasks: Control flow tracking [1] and compliance monitoring [20]. Next, we bridge ProMTEE's functional view and its deployment by introducing TEEs.

4.1 Functional viewpoint

Figure 2 presents our decentralized functional architecture. Multiple organizations capture events for each activity performed throughout the process via their own Event Stream Engine components. As is common in process mining contexts, we assume that events are associated with a unique triplet of *case ID* (e.g., "PO24#1"), *activity name* (e.g., "Fill in Container (FC)"), and *timestamp* (e.g., 2024-11-04T16:05:06), alongside domain-specific attributes encoding artifacts generated in the course of the process (e.g., the operator's code in Fig. 1). Every Process State Agent transmits events from the organization's Event Stream Engine to the Process Vaults via a secured channel, so as to avoid undesired disclosures (as per Req. R2). Multiple Process Vaults can be deployed for the same business process to allow organizations to host their own. Organizations interested, such as the insurance company in our example, can do monitoring in-house. Decentralizing this component diminishes the risks of a single point of failure, thereby improving service availability. However, we also postulate that *(i)* Process State Agents can remotely verify the Process Vault's security properties before transmitting their events (to cater for Req. R1), and *(ii)* Process Vaults are tamper-proof and operate on events in a secure context (in light of Req. R4) to hold and update what we call the *process state*.

We define the *process state* as a compact representation of the running process instances in terms of their execution stage or adherence to the given specification. Following Req. R3, we distinguish two separate sections of the process state: A *secret* section storing event-specific data (e.g., the operator's code of "FC" events), and an *accessible* section exposing monitoring results. The accessible state section is comprised of separate fields, each aligned with a specific monitoring perspective. The control flow

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-02867-9_15

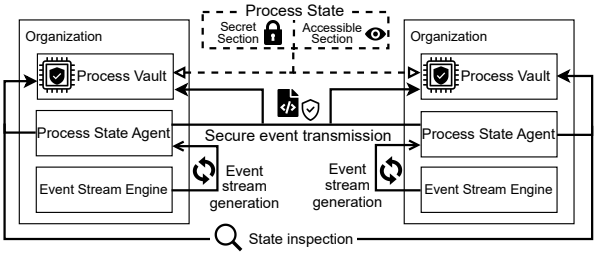


Fig. 2: Functional viewpoint of ProMTEE

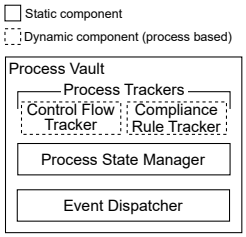


Fig. 3: Process Vault

field captures the set of activities enabled at runtime [1] (for example, after "FC" is carried out within instance "PO24#1", the associated control flow field records that "CC" is enabled). We identify the following three core values for a case's control flow status: *running* (i.e., the instance is in execution without registered errors), *completed* (i.e., the instance reaches the end with no errors), *disrupted* (i.e., a non-enabled activity was observed). The compliance rule field registers the current satisfaction status of the business rules. As in [20], the status of a process instance's rule can assume either of the following four values: *temporarily satisfied* (i.e., a running instance has not violated the rule, but breaches are still possible), *temporarily violated* (i.e., the rule is breached, but compensation activities to satisfy it are still possible), *satisfied* (i.e., the instance fulfills the rule), or *violated* (i.e., the instance breaches the rule).

Process State Agents generate state inspections to the Process Vault, which grants access to the accessible state section but hides the input events and data within the secret section. Figure 3 illustrates the sub-components of a Process Vault. The Event Dispatcher receives the events and forwards them to the Process State Manager. The latter securely handles the process state within the Process Vault. Each event processed by the Process State Manager results in an update of the process state. The Process State Manager updates it by coordinating the joint action of the Process Trackers, serving different monitoring tasks. We integrate state-of-the-art techniques to support control flow and compliance monitoring tasks via the Control Flow Tracker and the Compliance Rule Tracker, respectively. Being them well-established techniques, we shall not go into the details but provide an overview of them. Further information can be found in [1,15], respectively. As in [1], the Control Flow Tracker bases execution semantics upon workflow net firing mechanisms. We encode the routing logic through input and output activity matrices. Intuitively, an input matrix records, for every activity, the conditions enabling it in terms of predecessor activities. It encodes, e.g., that "CC" is enabled only after "FC" is executed. The output matrix records the successor activities enabled by an execution. It registers, e.g., that "CC" enables "ACT". As new events arrive, the Control Flow Tracker updates the control flow field of the process state by progressing the execution sequence (e.g., by juxtaposing the occurrence of "FC") and altering the enablement status of transitions accordingly (e.g., by enabling "CC"). For the Compliance Rule Tracker, we adopt the fluent model schema of the Mobucon approach [15]. In brief, we design the Compliance Rule Tracker as a composition of: *(i)* a set of declarative predicates expressing the business rules of the process (such as those in Tab. 1); *(ii)* a collection of finite state machines (FSMs,

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-02867-9_15

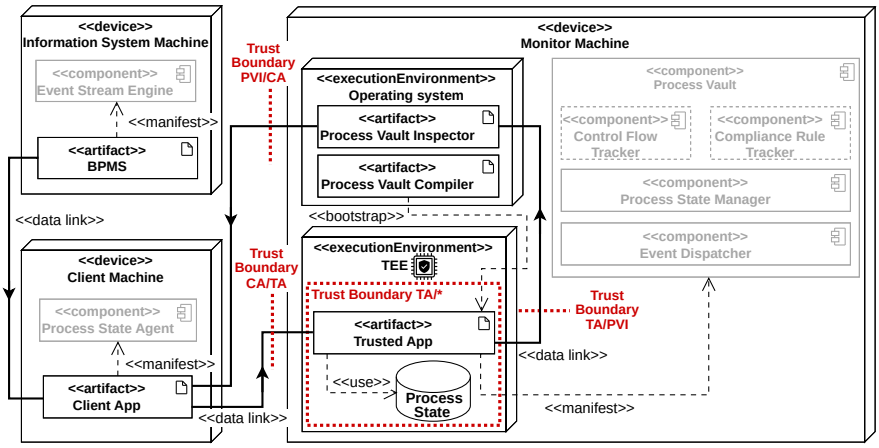


Fig. 4: UML deployment diagram of ProMTEE with trust boundaries

represented as transition matrices), each modeling the lifecycle of a business rule, and *(iii)* a reasoner evaluating the satisfaction of each predicate based on the state reached by the corresponding FSM during the instance execution. For example, upon processing "CC" after "FC", the tracker determines the evaluation of Rule C3 depending on the cargo inspector code: from a *temporarily satisfied* status into either *satisfied* (if the cargo inspector code differs) or *violated* (if it is the same).

The Event Dispatcher and Process State Manager are static components (represented as solid boxes in Fig. 3), meaning their logic remains fixed for all processes. In contrast, Process Trackers are dynamic components (see the dashed boxes in Fig. 3) that are built to fit the process being monitored. Once a Process Vault is deployed, its sub-components remain immutable and are not susceptible to external interference to meet Req. R4.

4.2 Deployment

Figure 4 shows a UML deployment diagram associating ProMTEE's aforementioned functional components (i.e., the light-grey boxes in Fig. 4) with *artifacts* (i.e., the black boxes) via the *manifest* relationship. We add to the standard UML notation *(i)* the flow direction for data-links for clarity, and *(ii)* the trust boundaries to represent transitions of trust levels (as per the STRIDE framework,[4] which we will employ later for the threat analysis in Sect. 6.1).

We identify three distinct types of *device* nodes: the Information System Machine, the Client Machine, and the Monitor Machine. Though logically separated, they may reside within the same computing node. The Information System Machine hosts the Business Process Management System (BPMS) [5] of an organization, enabling the data generation functionalities of the Event Stream Engine. BPMSs transmit events via the Client App, through which we manifest the Process State Agent component.

[4] learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-02867-9_15

Organizations interested in monitoring the business process in-house host the Process Vault component in their Monitor Machine. We distinguish two *execution environment*s in it, namely, the Operating System (OS) and the Trusted Execution Environment (TEE). To achieve Objs. O1 and O2, and elaborate original events sheltered from the unauthorized disclosures, we leverage TEEs' guarantees on data integrity, data confidentiality, and code integrity [10]. TEE-enabled CPUs can operate on encrypted data in a reserved memory section, not available for the OS [22]. However, TEE-allocated memory is limited. Exceeding these limits entails out-of-memory crashes that can severely affect availability (the service is interrupted) and data secrecy (encrypted data can be swapped to outer memory regions). Efficient memory management is thus paramount to Objs. O1 and O2.

We manifest the Process Vault and its components as a Trusted App running on top of a TEE. In light of Obj. O2, the events transmitted by a Client App (CA) are processed by the Trusted Apps (TAs), traversing the CA/TA trust boundary. Upon the delivery of new events, the Trusted Apps update the process state at runtime (Obj. O3). The components of the Process Vault comprising the Trusted App maintain the same trust level and handle event data securely within the TA/* trust boundary. The Trusted App outsources updates of the accessible state section (introduced in Sect. 4.1) to the Process Vault Inspector (PVI) running within the Monitor Machine's OS, crossing the TA/PVI trust boundary. As per Req. R3, this transmission excludes sensitive data of individual events, which are exclusively processed within the TEE. For example, the insurance company can use its Client App to interact with a Process Vault Inspector (either its own or one managed by another organization) to verify whether "FC" and "CC" activities are performed by different operators, without accessing their ID codes (Rule C3). Monitoring outcomes accessed by the Client App traverse the PVI/CA trust boundary. To fulfill Req. R1, we consider attestable TEEs [3], meaning that the TEE's CPU can produce a cryptographic proof (referred to as *attestation evidence*) for the trusted status of Process Vaults using a secret hardware-bounded encryption key. This evidence includes the hash footprint of the Trusted App's source code (the so-called *measurement*) alongside context-related additional data. For example, the industry can verify that it is transmitting events to insurance company's expected Process Vault, running in a functioning TEE. Process Vaults' source code and measurement are bootstrapped in the TEE by the Process Vault Compiler running in the OS of the Monitor Machine.

5 The ProMTEE protocol

The aforementioned components interact across four phases of our protocol, in which we assume remote messages are exchanged via an *authenticated perfect point-to-point link* [4]. We present each phase of the ProMTEE protocol in the following paragraphs.

Bootstrapping. The bootstrapping phase enables organizations to launch a Process Vault via the Process Vault Compiler, which generates the dynamic Process Trackers' components and installs them into the TEE. To this end, the organizations feed input to the compiler *(i)* a control flow specification for the Control Flow Tracker (e.g., the BPMN model in Fig. 1), *(ii)* a set of compliance rules for the Compliance Rule Tracker (in our implementation, a

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-02867-9_15

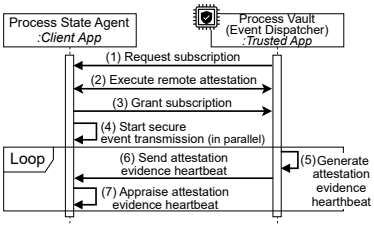


Fig. 5: Trusted subscription phase

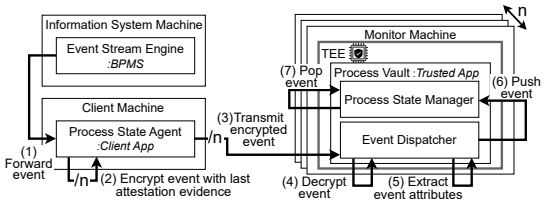


Fig. 6: Secure event transmission phase

Rego[5] encoding of business rules like the ones in Tab. 1), and *(iii)* an extraction manifest (a notion inspired by [12]) for the fetching instructions of the Event Dispatcher.

The Process Vault Compiler uses those inputs as follows. From the *(i)* control flow specification, it extracts the input/output matrices described in Sect. 4.1. Those matrices are embedded into the Control Flow Tracker to represent the process's routing logic. The compiler processes the *(ii)* compliance rules to derive a set of declarative predicates along with their corresponding FSMs to be employed by the Compliance Rule Tracker's reasoner (see Sect. 4.1). Finally, *(iii)* the manifest provides directives for parsing raw event attributes and indicating which of them should be kept in the TEE's secret state section. As shown in Fig. 7, e.g., the manifest instructs to retain the logistics operator code, so as to evaluate Rule C3 over the occurrences of "FC" and "CC".

Once bootstrapped, the TEE makes the Process Vault incorruptible. We leverage this property to ensure Req. R4. Organizations can verify the authenticity of the Process Vault by comparing the corresponding Trusted App's identifying measurement (see Sect. 4.2) with the one obtained by locally reproducing the deterministic compilation process.

Trusted Subscription. In compliance with Req. R1, Process Vaults must prove their status of Trusted App to subscribe to Process State Agents' event streams. Therefore, we designed a dedicated phase that we call *trusted subscription*, built following the RATS RFC [3]. Figure 5 schematizes it in the form of a UML sequence diagram. We describe it below. (1) Initially, the Event Dispatcher requests a new subscription to the Process State Agent, which, in turn, executes a TEE *remote attestation*, as in [3]. (2) Through this established mechanism, the Process State Agent requests and receives from the Process Vault an attestation evidence signed by the CPU containing its measurement (see Sect. 4.2). We customize the structure of the RATS attestation evidence by incorporating a symmetric encryption key randomly generated by the Process Vault, which we refer to as the *provisioning key*. By decrypting the evidence (via CPU producer according to [3]) and comparing its measurement with the expected one, the Process State Agent appraise the integrity of both the Process Vault's source code and the provisioning key (thus meeting Req. R1). (3) If the evidence appraisal is successful, the Process State Agent grants the subscription by choosing and sending a time interval to the Event Dispatcher. (4) Then, it concurrently starts the secure event transmission with the newly subscribed Process Vault using the evidence's provisioning key (we describe this phase in the next paragraph). To keep receiving events, (5) the Process Vault must produce and (6) periodically transmit (based on the aforementioned time interval) an attestation evidence heartbeat to the

[5] openpolicyagent.org/docs/latest/policy-language.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-02867-9_15

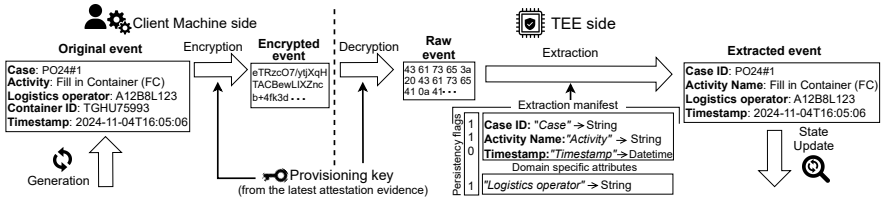


Fig. 7: Example of the trusted subscription and secure event transmission phases

Process State Agent containing a new randomly generated provisioning key. (7) As in step 2, the Process State Agent appraises the evidence (following RATS [3]), and in case of a positive outcome, it extends the subscription duration. Failures or timeouts in the evidence appraisal determine the deactivation of the secure transmission. To achieve Obj. O3, we parallelize the Process Vault attestation and event transmission, ensuring that the latter is not delayed by the appraisal processes (thus impacting responsiveness). Consequently, parties in our example can verify the insurance company's Process Vault, avoiding interruptions in the event transmission.

Secure event transmission. To meet Req. R2, Process State Agents transmit events to subscribed Process Vaults over an encrypted channel. Figure 6 depicts an unfolding schematization of this phase involving n subscribed Process Vaults. Figure 7 illustrates an example of event transmission with the "FC" activity from Fig. 1. (1) First, the Event Stream Engine forwards the original event (e.g., the manufacturer's "FC" activity) to the Process State Agent of the same organization. (2) The Process State Agent encrypts the data using the provisioning keys of the latest attestation evidence obtained from the n subscribed Process Vaults during the trusted subscription phase. Since these random keys are secretly generated in the TEE (see steps 2 and 4 of the trusted subscription phase), the newly encrypted event can only be decrypted by the Process Vault which sent the evidence. (3) Thereafter, the Process State Agent transmits the ciphered event to the n subscribed Process Vaults. The insurance company and industry's Process Vaults, subscribed to the manufacturer's Process State Agent, receive the "FC" event encrypted with different provisioning keys. (4) Upon the event arrival within the TEE, the Process Vault decrypts the message using the provisioning key of its latest evidence heartbeat, and (5) extracts its attributes from the raw event according to the extraction manifest. In the example of Fig. 7, the manifest specifies the parsing directives to extract the case ID, the activity name, the timestamp, and the logistics operator code (required to verify Rule C3), while keeping out extra information like the container ID attribute. (6) Finally, the Event Dispatcher pushes the event to the Process State Manager for the state update phase (discussed next). The Process State Manager operates as an *asynchronous job handler* [4], which serves as a mechanism to fetch events to be processed following a First-In First-Out queueing model. Therefore, if the events associated with the "FC" and "CC" activities from Fig. 1 are delivered in close succession, we ensure "CC" is processed only after "FC", avoiding inconsistent updates of the process state due to concurrent processing.

State Update. The Process State Manager updates the process state in the isolated context of the TEE. Algorithm 1 presents the pseudocode for this phase, while Fig. 8 illustrates

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-02867-9_15

Algorithm 1: State update phase

Input: e, an event; Ps, a process state; Em, an extraction manifest
Implements: Process State Manager
Uses: Control Flow Tracker CFT; Compliance Rule Tracker CRT; Process Vault Inspector PVI; Trusted Execution Environment TEE;

1 $h_e \leftarrow \text{hash}(e)$ #Compute the hash h_e of the new event e
2 **foreach** *pt in* {*CFT, CRT*} **do** #For each Process Tracker *pt*. . .
3 $Upds \leftarrow pt.\text{update}(e, Ps)$ #*pt* updates *Ps* with e and returns *Upds*
4 **foreach** *upd in Upds* **do** #For each computed state update *upd* . . .
5 $h_{upd} \leftarrow \text{hash}(upd, now, h_e)$ #Compute the hash h_{upd} of *upd*, *now*, h_e
6 $evd \leftarrow$ TEE.getAttestationEvidence(h_{upd}) #Get TEE evidence for h_{upd}
7 **yield** *upd, now,* h_e*, evd* **to** PVI #Outsource the update *upd* to PVI
#Store in *Ps* the attributes of e flagged as persistent in Em
8 storePersistentAttributes(e, Em, Ps)

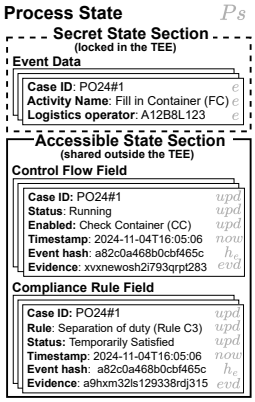


Fig. 8: Process state

the process state after handling the FC event of Fig. 7. In Fig. 8, the variables mentioned in Alg. 1 are written in gray next to the assigned value.

When processing a new event e, the Process State Manager employs the Process Trackers to update the process state *Ps* (Alg. 1, ln. 3). Each Process Tracker *pt* produces a set of updates *Upds*. As discussed in Sect. 4.1, the Control Flow Tracker reacts by updating the enablement of activities. For example, it updates the control flow field of *Ps*, setting "CC" as enabled and "FC" (i.e., the activity of e) as disabled for case "PO24#1", as per the input and output matrices. Since the execution step is encoded in the routing logic and the case is not concluded, the status remains *running*. Concurrently, the Compliance Rule Tracker updates the compliance rule field of *Ps*. To this end, it triggers a status transition within the FSM associated with each rule (as outlined in Sect. 4.1). In our example, the activity of e is "FC", which triggers the evaluation of Rule C3. The tracker assigns its status with *temporarily satisfied*. Upon the eventual occurrence of "CC", the status will transition to *satisfied* or *violated* depending on whether the new cargo inspector code differs from the logistics operator ID born by e or not, respectively.

For each update *upd*, the Process State Manager generates a hash footprint h_{upd}, including the current timestamp *now* and the new event's hash h_e (ln. 5). It then instructs the TEE to sign an attestation evidence *evd* (as in the RATS standard [3]) in which we include h_{upd} (ln. 6). Finally, it transmits *upd* alongside *now*, h_e, and *evd* from the TEE to the outer Process Vault Inspector (ln. 7) , as part of the accessible state section. To meet Req. R4, when the parties of our example access the process state via the Process Vault Inspector, they can verify that *upd*, *now*, h_e are intact and originates from the expected Process Vault by attesting *evd* (as in [3]) and evaluating the decrypted information found within it (i.e., the measurement and h_{upd}). Finally, the Process State Manager stores the attributes of e flagged as persistent in the extraction manifest Em within the secret section of *Ps*, ensuring availability for future updates (ln. 8). By recalling the example manifest in Fig. 7, the case ID, activity name and logistics operator attributes (flagged as persistent) are cached in the secret section and never disclosed to the Process Vault Inspector outside the TEE (as shown in Fig. 8).

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-02867-9_15

Table 3: Vulnerabilities mitigated in ProMTEE. Requirements under threat refer to Tab. 2.

ID	Trust boundary ID	STRIDE type	Threat description	Against
T1	CA/TA	Spoofing	The attacker pretends to be a Process Vault to receive event data	R1
T2	CA/TA	Information disclosure	The attacker discloses sensitive data during the secure event transmission	R2
T3	TA/*	Information disclosure	The attacker leaks event data from the Process Vault's secret state section	R3
T4	TA/*, TA/PVI, PVI/CA	Tampering	The attacker meddles with a Process Vault to alter monitoring outcomes	R4

6 Evaluation

We center our evaluation around two core aspects. (1) In Sect. 6.1, we verify that the security requirements Reqs. R1 to R4 are met by ProMTEE by design. (2) In Sect. 6.2, we introduce our proof-of-concept implementation of ProMTEE and report on the experiments we conducted to gauge its feasibility in terms of computation memory and time, alongside a discussion of the results with respect to Objs. O1 to O3.

6.1 Threat Analysis

Here we discuss the mitigation of the STRIDE[4] threats endangering the fulfillment of Reqs. R1 to R4. Table 3 summarizes and classifies the menaces, and specifies the trust boundaries (introduced in Sect. 4.2) they jeopardize. Below, we describe our countermeasures based on the protocol phases we detailed in Sect. 5.

(T1) To mitigate the risk of spoofing attackers impersonating Process Vaults across the CA/TA trust boundary, we design the trusted subscription phase (Fig. 5). Process Vaults periodically transmit to Process State Agents attestation evidences (signed by the TEE's processor), including their measurement. By validating this evidence, data owners can ensure that *(i)* the evidence originates from a trusted app running in a TEE, and *(ii)* the trusted app sending the evidence is a Process Vault built from known process specifications. Failures in the evidence appraisal or timeouts in its transmission determine the immediate deactivation of the event transmission. **(T2)** During secure event transmission, an adversary may intercept events risking information disclosure at the CA/TA trust boundary. To mitigate this menace, we include within the attestation evidence an encryption key (i.e., the provisioning key in Fig. 7), randomly generated by the Process Vault and employed by Process State Agents for the encrypted transmission of events (Fig. 6, step 3). Validation of the attestation evidence's measurement ensures that the key was securely generated by the expected Process Vault. External observers exploiting eavesdropping attacks to intercept the transmission channel would only access ciphered data. Also, by updating the provisioning key randomly at each heartbeat (Fig. 6, step 5), we complicate cryptoanalysis attacks targeting recurrent patterns to infer events' sensitive information. **(T3)** To disclose sensitive information , a malicious actor might gain access to a Process Vault's memory space by exploiting a direct memory access attack against TA/* trust boundary. We counter this threat by leveraging the TEE's hardware-encryption mechanism. The TEE's CPU operates on Process Vault's data keeping it encrypted in a reserved section of the DRAM [10]. The decryption key is hardware-bounded and randomized at each power cycle. This ensures that the processor interacts with event data solely through the attested Process Vault's source code. Access attempts from adversary entities within the Monitoring Machine are systematically blocked by the CPU. In the

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-02867-9_15

Table 4: Event logs used for our experiments

Event log	Type	Cases	Events	Activities	Trace events			Business rules
					min	max	avg.	
Supply Chain Example (SC)	Synthetic	2000	40 740	19	18	27	20	5
Sepsis	Real-world	1050	15 214	16	3	185	14	2 [16]
BPIC12	Real-world	13 087	262 200	36	3	75	20	16 [14]
Road Traffic Fines (RTF)	Real-world	150 370	561 470	11	2	20	4	8 [24]

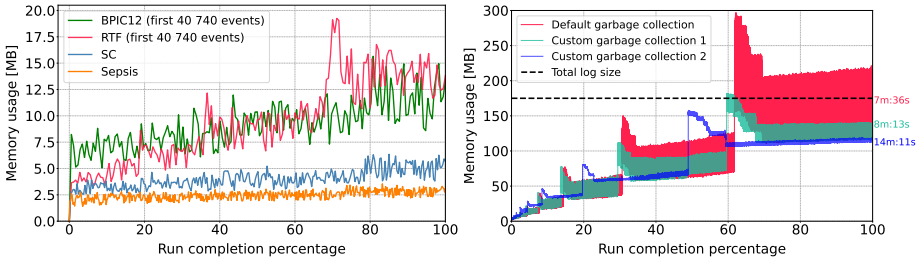
event of a cold-boot attack, adversaries extracting sensitive information by plugging the main memory on another machine only retrieve encrypted content. **(T4)** Attackers with direct access to the MonitoringMachine may employ code poisoning attacks targeting the components within TA/* trust boundary to introduce misbehaviors in the Process Vault. To mitigate this issue, we exploit TEE's access control mechanisms. Unauthorized modifications to the Process Vault's source alter its runtime measurement, which the CPU detects and reports. Thus, we ensure that events within the TEE are processed without post-deployment alterations of the Process Vault (e.g., print instructions leading to data exposure) and that the observed results align with the expected execution of the protocol. Also, we leverage remote attestations to detect downstream corruptions of the accessible state section shared outside the TEE across TA/PVI and PVI/CA. In Alg. 1, ln. 7, we include the hashed state update in a TEE-signed attestation evidence shared alongside the monitoring results. External observers can verify the update's authenticity via TEE remote attestation, ensuring integrity by detecting mismatched hash footprints.

6.2 Implementation and Experimental Evaluation

To experimentally validate our approach, we gauged our implementation's computation efforts in terms of space and time with a twofold objective: *(i)* Since our approach leverages TEEs to achieve Objs. O1 and O2, and these environments are prone to low memory availability, we analyze runtime memory consumption to prevent out-of-memory crashes (see Sect. 4.2 for potential security implications). *(ii)* As ProMTEE is designed to operate during the execution of processes, we quantify the induced latency and show that the overhead of our approach does not lead to diverging lags, thereby ensuring Obj. O3.
Implementation. We implemented our approach in GO, resorting to the EGo framework to deploy trusted apps in an Intel SGX TEE hosted by an Intel Xeon Gold 5415+ CPU.[6] We bootstrap our Process Vault prototype using a Python encoding of the Process Vault Compiler. In our tool, business rules are expressed in the form of Rego declarative policies.[5] Our implementation is publicly available at doi.org/10.5281/zenodo.15545154.
Experimental setup. For our tests, we employ real-world event logs retrieved from the 4TU.ResearchData repository alongside a synthetic dataset generated from Fig. 1 via the PLG2 toolset[7], based on our use case scenario described in Sect. 3. We will henceforth refer to the latter as Supply Chain Example (SC) log. Table 4 summarizes their main characteristics alongside the related business constraints we retrieved from

[6]EGo: edgeless.systems/products/ego; Intel SGX: sgx101.gitbook.io/sgx101.
[7]4TU.ResearchData: data.4tu.nl; PLG2: plg.processmining.it.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-02867-9_15



(a) Comparison across event logs (b) Comparison across GC policies (RTF log)

Fig. 9: Memory usage of the Process Vault

the scientific literature [24,16,14]. For each log, we bootstrapped a Process Vault instance. In our experimental setup, we stream the logs by sequentially emitting all the events in the original order at the maximum available bandwidth (modulo the throughput of the simulation engine). Also, we keep the accessible state section for all cases, to show the cumulative effect on memory utilization regardless of the order in which single cases are streamlined. In our repository, we collect data and scripts reproducing the test runs.

Memory Usage. Here, we aim to test whether the runtime memory demand of the Process Vault for processing real-world workloads is compatible with the constrained capacity of TEEs (Objs. O1 and O2). In Fig. 9(a), we illustrate the runtime memory utilization trend of the Process Vault with the event logs in Tab. 4. For the sake of readability, we cut the figure's viewport to the number of observations collected with the SC log (40 740). As expected, due to the cumulative growth of the accessible state section, all the values display an upward trend. The processing of the Sepsis dataset (orange line) exhibits the lowest and most stable growth, consistently remaining below 3 MB throughout execution. The SC test (in blue) follows a similar trend, gradually rising from approximately 2.5 MB to 5.5 MB. In contrast, the runs with BPIC12 and RTF (in green and red, respectively) show higher memory demands due to a greater number of constraints associated with the logs (8 and 16, respectively, as per Tab. 4). From the start, BPIC12 requires higher memory allocation. However, RTF exhibits a steeper increase in memory consumption compared to BPIC12, diverging significantly after 60 % of the events are fetched. At its peak, the memory usage with RTF exceeds 15 MB. This behavior is attributable to the higher ratio of cases per parsed event (0.30 in RTF; 0.05 in BPIC12), which necessitates the allocation of additional data structures to manage more process constraints and control flow states simultaneously.

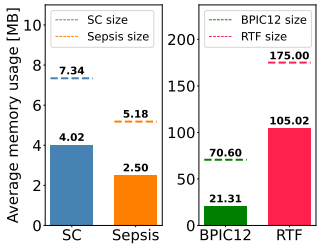


Fig. 10: Average memory usage

In this analysis, we observe the increasing amplitude of memory fluctuations as execution progresses. This effect becomes more pronounced as the process state grows larger, thus posing a potential threat to the robustness of our solution. Upon further investigation, we were able to relate this phenomenon to the low-level garbage collection policy of the Process Vault implementation. In Fig. 9(b), we present the full view on

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-02867-9_15

Table 5: Queueing-related metrics in real-world event log settings

Log	**Elapsed time** (avg.) [ms] λ^{-1}	**Arrival rate** [kHz] λ	**State update** (avg.) [ms] μ^{-1}	**Service rate** [kHz] μ	**Occupation rate** λ/μ	**Expected lag** [ms] $(\mu-\lambda)^{-1}$
Sepsis	3 266 601.06	3.06×10^{-7}	0.497	2.012	$\mathbf{1.52 \times 10^{-7}}$	**0.497 000 08**
BPIC12	54 582.86	1.83×10^{-5}	0.867	1.153	$\mathbf{1.59 \times 10^{-5}}$	**0.867 013 77**
RTF	756 638.03	1.32×10^{-6}	0.649	1.541	$\mathbf{8.58 \times 10^{-7}}$	**0.649 000 56**

the memory usage trend of the RTF log (the largest one) under three different garbage collection policies. The default one (the red line) operates adaptively based on the current memory utilization. This policy generates increasingly wide memory oscillations, exceeding the total log size threshold (the dashed black line) more frequently after 60 % of execution and peaking at around 270 MB. These peaks correspond to the dynamic expansion of in-memory data structures. To control these fluctuations, we implemented two custom garbage collection policies based on fixed time intervals: (1) every 50 ms, and (2) every 10 ms. The former reduces peak memory usage to approximately 180 MB, maintaining a more stable profile with an oscillation amplitude that stabilizes after 20 % of the run (the green line). The latter further lowers memory consumption, keeping it consistently below 120 MB with minimal fluctuations and peaking approximately to 150 MB (the blue line). These memory savings come at the cost of increased execution time (annotated in the plot to the right side, at the end of each poly-line). Custom policy 2 minimizes memory usage but increases runtime by about 90 % with respect to the default. Custom policy 1 strikes a balance between memory occupation and execution time.

Figure 10 presents the average runtime memory usage of the ProcessVault for each test. To provide context for these values, we compare them to the total size of the corresponding logs. The average memory consumption remains significantly lower than the log size. We also remark that no out-of-memory errors occurred during these experiments.

Responsiveness. To assess the responsiveness of our solution (Obj. O3), we first verify that the ProcessVault introduces minimal *observation lag* (i.e., the difference between the event processing time and its generation time) compared to the *elapsed time* (i.e., the time interval between two consecutive generic events) in real-world executions. We want to avoid, in other words, that the queue of unprocessed events gets out of control and leads to a damaging loss of responsiveness. Therefore, we model the event fetching from the EventDispatcher to the ProcessStateManager (Fig. 6, step 7) as an *M/M/1* queue, following Peters et al. [19]. We exclude considerations of networking overhead and packet loss as they fall outside the scope of this study. Table 5 displays the average elapsed time of the the event logs in Tab. 4. These values serve as a basis for the expected arrival rates (λ). As in [19], we assume that events within the logs occur adhering to a Poisson process, which effectively represents multiple-case arrivals. In the table, we also report the average state update latency, i.e., the mean time the ProcessStateManager takes to process an event. We use it as a basis to compute the service rate (μ) of the ProcessVault for each log. The BPIC12 test exhibits the highest state update latency (0.867 ms), attributable to the larger number of business rules compared to the other logs, followed by RTF (0.649 ms) and Sepsis (0.497 ms). Using the computed λ and μ, we apply the *M/M/1* occupation rate equation to determine the expected lag [19]. Table 5 displays the outcome. The ProcessVault achieves utilization rates in the order of 10^{-7} to 10^{-5},

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-02867-9_15

hence much closer to the theoretical minimum of 0 than to the divergence threshold of 1. This result evidences that the system remains stable and responsive without queue overloading. The expected lags are of the same magnitude as the state update latency, thus signifying that the processing time does not cumulatively increment over time. These findings suggest that the Process Vault manages real-world execution settings by maintaining observation lag at a minimum.

TEEs' hardware-based encryption introduces performance overhead due to its hardware-based encryption mechanisms [10]. Hence, we want to ensure that this overhead ranges within predictable intervals and does not lead to diverging lags. To this end, we timed the Process Vault running in a non-TEE environment (thus disabling the trusted subscription phase). Figure 11 plots the average lag for each tested log in both non-TEE and TEE settings (the blue and orange bars, respectively) alongside the delta (magenta). We observe that the overhead ranges between 0.17 ms to 0.25 ms (with Sepsis and BPIC12, respectively), i.e., with a delta of 20 % to 28 % (SC and Sepsis, respectively). The narrow range of these values suggests that the overhead introduced by our approach is not strictly dependent on the overall workload, indicating that the additional latency remains manageably bounded.

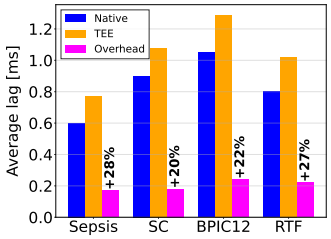


Fig. 11: TEE overhead

7 Conclusion

We introduced ProMTEE, a novel framework for executing online process monitoring tasks while preserving the secrecy of the original information. ProMTEE leverages a TEE-based architecture supported by a four-staged interaction protocol designed to securely transmit and process event data within protected environments.

The findings presented in this paper indicate multiple research directions. At present, we assume organizations to countercheck Process Vaults' correctness through manual inspection of shared input specifications; we envision a systematic approach to assess the correctness of running instances, even with partially or fully undisclosed specifications. Also, we aim to widen the monitoring scope from the intra to inter-instance dimension involving synchronization mechanisms for case identification and timestamping. Considering the inter-instance context, we want to investigate aggregation mechanisms for monitoring outputs across independent Process Vaults, each tracking distinct process segments. Finally, we aim to relax the assumption of perfect point-to-point links [4], thereby addressing potential inconsistencies due to faults, reboots, and packet loss.

Acknowledgments. This work was partially funded by the Italian Ministry of University and Research under PRIN grant B87G22000450001 (PINPOINT) and by Sapienza University of Rome under grant RG123188B3F7414A (ASGARD).

References

1. van der Aalst, W.M.P.: The application of petri nets to workflow management. J. Circuits Syst. Comput. **8**(1), 21–66 (1998)

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-02867-9_15

2. Batista, E., Martínez-Ballesté, A., Solanas, A.: Privacy-preserving process mining: A microaggregation-based approach. J. Inf. Secur. Appl. **68**, 103235 (2022)
3. Birkholz, H., Thaler, D., Richardson, M., et al.: Remote attestation procedures (RATS) architecture. RFC (2023)
4. Cachin, C., Guerraoui, R., Rodrigues, L.E.T.: Introduction to Reliable and Secure Distributed Programming (2. ed.). Springer (2011)
5. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
6. 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)
7. Fahrenkrog-Petersen, S.A., van der Aa, H., Weidlich, M.: PRIPEL: privacy-preserving event log publishing including contextual information. In: BPM 2020 (2020)
8. Fahrenkrog-Petersen, S.A., van der Aa, H., Weidlich, M.: Optimal event log sanitization for privacy-preserving process mining. Data Knowl. Eng. **145**, 102175 (2023)
9. Goretti, V., Basile, D., Barbaro, L., Di Ciccio, C.: Trusted execution environment for decentralized process mining. In: CAiSE 2024. vol. 14663, pp. 509–527 (2024)
10. Han, Z., Hu, H.: ProDB: A memory-secure database using hardware enclave and practical oblivious RAM. Inf. Syst. **96**, 101681 (2021)
11. Kang, K., Qi, H., Lu, J., et al.: Dual-channel supply chain disruption model and analysis under cargo transportation insurance. IEEE Access pp. 114953–114967 (2020)
12. Klinkmüller, C., Ponomarev, A., Tran, A.B., et al.: Mining blockchain processes: Extracting process mining data from blockchain applications. In: BPM 2019. vol. 361, pp. 71–86 (2019)
13. Liu, C., Li, Q., Zhao, X.: Challenges and opportunities in collaborative business process management: Overview of recent advances and introduction to the special issue. Inf. Syst. Frontiers (3), 201–209 (2009)
14. Ly, L.T., Maggi, F.M., Montali, M., et al.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015)
15. Maggi, F.M., Montali, M., van der Aalst, W.M.P.: An operational decision support framework for monitoring business constraints. In: FASE 2012. vol. 7212, pp. 146–162 (2012)
16. Mannhardt, F., Blinde, D.: Analyzing the trajectories of patients with sepsis using process mining. In: RADAR+ EMISA 2017. vol. 1859, pp. 72–80 (2017)
17. Mincuzzi, N., Falsafi, M., Modoni, G., et al.: Managing logistics in collaborative manufacturing: the integration services for an automotive application. In: PRO-VE 2019. pp. 355–362 (2019)
18. 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)
19. Peters, S., Kerner, Y., Dijkman, R.M., et al.: Fast and accurate quantitative business process analysis using feature complete queueing models. Inf. Syst. **104**, 101892 (2022)
20. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer (2012)
21. Rozanski, N., Woods, E.: Software Systems Architecture. Addison-Wesley Professional (2005)
22. Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: What it is, and what it is not. In: TrustCom 2015. pp. 57–64. IEEE (2015)
23. Soffer, P., Hinze, A., Koschmider, A., et al.: From event streams to process models and back: Challenges and opportunities. Inf. Syst. **81**, 181–200 (2019)
24. Völzer, H., Zerbato, F., Sulzer, T., et al.: A fresh approach to analyze process outcomes. In: ICPM 2023. pp. 97–104 (2023)
25. Weber, I., Xu, X., Riveret, R., et al.: Untrusted business process monitoring and execution using blockchain. In: BPM 2016. vol. 9850, pp. 329–347. Springer (2016)

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-02867-9_15

This document is a pre-print copy of the manuscript
(Basile and Di Ciccio 2025)
published by Springer (available at link.springer.com).

The final version of the paper is identified by DOI: 10.1007/978-3-032-02867-9_15

References

Basile, Davide and Claudio Di Ciccio (2025). "Secrecy Preservation for Online Process Monitoring with Trusted Execution Environment". In: *BPM*. Ed. by Arik Senderovich, Cristina Cabanillas, Irene Vanderfeesten, and Hajo A. Reijers. Vol. 16044. Lecture Notes in Computer Science. Springer, pp. 235–254. ISBN: 978-3-032-02866-2. DOI: 10.1007/978-3-032-02867-9_15.

BibTeX

```
@InProceedings{   Basile.DiCiccio/BPM2025:SecrecyPreservationOnline,
  author        = {Basile, Davide and Di Ciccio, Claudio},
  booktitle     = {BPM},
  title         = {Secrecy Preservation for Online Process Monitoring with
                   Trusted Execution Environment},
  year          = {2025},
  pages         = {235--254},
  crossref      = {BPM2025},
  doi           = {10.1007/978-3-032-02867-9_15},
  keywords      = {Collaborative information systems; Process-aware
                   information systems; Business Process Management; TEE;
                   Confidential Computing}
}
@Proceedings{     BPM2025,
  title         = {Business Process Management - 23rd International
                   Conference, {BPM} 2025, Seville, Spain, August 31 -
                   September 5, 2025, Proceedings},
  year          = {2025},
  editor        = {Arik Senderovich and Cristina Cabanillas and Irene
                   Vanderfeesten and Hajo A. Reijers},
  isbn          = {978-3-032-02866-2},
  publisher     = {Springer},
  series        = {Lecture Notes in Computer Science},
  volume        = {16044}
}
```