Your Secret Is Safe With Me: Federated Directly-Follows Graph Discovery

Christian Rennert✉ , Julian Albers , Sander J. J. Leemans , and Wil van der Aalst
Department of Computer Science, RWTH Aachen University, Aachen, Germany
rennert@pads.rwth-aachen.de, julian.albers@rwth-aachen.de, leemans@bpm.rwth-aachen.de, wvdaalst@pads.rwth-aachen.de

***Abstract*—Business processes may span multiple organizations. For instance, cattle may move through several organizations in an agricultural supply chain, or patients may be seen by multiple healthcare providers as part of a treatment process. Optimizing these cross-organizational processes is an aim of process mining, however, process mining efforts may be challenged by commercially sensitive data or privacy laws, which may prevent the involved organizations from sharing recorded process data with one another. Federated process mining aims to perform inter-organizational analyses without information being shared across organizational borders. In this paper, we propose a federated technique to discover directly-follows graphs (DFGs), using homomorphic encryption, while keeping timestamps and activities secret. We evaluate the feasibility of our technique using real-life event logs to discover their DFGs and discuss potential attacks.**

***Index Terms*—federated process mining, event data, data privacy, homomorphic encryption, directly-follows graph**

I. Introduction

Organizations collaborate to produce and distribute goods, to offer services, and to manage payments in business processes. A process may span multiple organizations. For instance, the care pathway of a single patient may span multiple healthcare providers such as a hospital, a general practitioner, a physio, a pharmacy, and a health insurer. Similarly, an agricultural supply chain may span multiple farms specialized in different live stages, an abattoir, and multiple government agencies. In such cross-organizational settings, applying process mining may reveal system-wide insights, such that the entire front-to-end process can be optimized. For instance, an entire healthcare system can be optimized to reduce pressure on hospitals and to focus on prevention [1], and for agricultural supply chains, sector-wide food safety and animal welfare can be improved.

If organizations are allowed to share recorded process behavior with one another, standard process mining techniques can be applied. However, in some cases, they may not be allowed or may not want to share data with one another, due to, e.g., commercial sensitivity or privacy laws [2]. *Federated process mining* provides techniques for such inter-organizational settings, *without requiring organizations to share data in readable form with anyone*. Figure 1 illustrates the context of federated process discovery, which aims to obtain a process model: two organizations, A and B, both execute process steps (*activities*) for cases (cow, patient, ...), and cases may perform some events with A and some with B. For instance, a patient is subject to a surgical procedure in a hospital, revalidates with a physio, and attends regular check-ups at both the hospital and with a general practitioner. Both A and B record their own events in *partial event logs*. Their aim is to obtain a directly follows graph (DFG) of their combined process, without revealing their logs to one another.

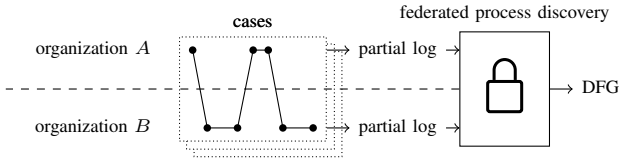


Fig. 1: Context of federated process discovery.

Existing approaches require public handover information [3], [4], only guarantee data confidentiality if at least three non-colluding, in majority honest organizations are involved [5], or pose restrictions, such as that no two organizations can execute the same activity [4].

In this paper, we introduce a federated process discovery protocol that does not place restrictions nor require prior knowledge on the workflow, and that does not require a trusted third party nor trust towards the partner organization. The protocol requires the organizations to be able to link cases (i.e. have a common case ID, such as a social security number), and requires a pre-negotiation on which activities are to be included in the final DFG. Then, the organizations communicate using fully homomorphic encryption (FHE) to obtain a DFG of their combined cross-organizational process.

In our evaluation, we study the complexity of our approach on real-life logs in terms of time and FHE operations. We also evaluate the correctness of our approach.

The remainder of the paper is structured as follows: Section II describes the context of the problem, Section III discusses related work, Section IV introduces concepts and notation, Section V introduces our approach, Section VI evaluates it, and Section VII concludes the paper.

II. Context

In federated process discovery, two organizations, A and B, run a single business process together. That is, each case may perform some process steps with A and some steps with B.

The authors gratefully acknowledge the financial support by the Federal Ministry of Education and Research (BMBF) for the joint project AIStudyBuddy (grant no. 16DHBKI016).

We assume that A and B (i) have consistent case IDs, (ii) have a timestamp for each event, (iii) share a common clock, and (iv) both keep their own private event logs, which we call *partial* logs.

Example 1 (Partial event logs): For instance, consider two partial event logs, representing two traces. Note that the trace with case ID 003 spans both A and B:

$$\text{partial log } L_A = [\langle a^{@1}, b^{@2} \rangle^{002}, \langle b^{@2}, a^{@3} \rangle^{003}]$$
$$\text{partial log } L_B = [\langle c^{@1}, b^{@4} \rangle^{003}]$$
$$\text{combined log} = [\langle a^{@1}, b^{@2} \rangle^{002}, \langle c^{@1}, b^{@2}, a^{@3}, b^{@4} \rangle^{003}]$$

We do not pose any further restrictions: activities may be executed by A and/or B, and traces may have events within A and/or B.

The aim of *federated process discovery* is to let A and B construct a DFG of their combined business process, without revealing any more information to one another than what can be derived from the DFG. Information that can be derived from a DFG includes the number of traces (by counting the start activities) and events (the number of incoming edges for each node) in the combined process. If there are only two organizations involved, this allows them to derive the number of traces and events in the other organization's partial log. Similarly, the activities and their number of occurrences can be derived.

A. Adversaries

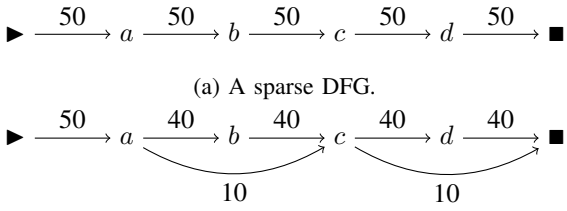
We assume that A and B are both willing to engage in the protocol, but either may be a malicious adversary [6]. That is, either organization may attempt to break encryption, alter events, alter cases, manipulate case IDs, manipulate timestamps, or add identifiable details to coerce the other organization into revealing information. Consequently, the protocol must ensure that no matter how each party manipulates the other, no information beyond what can be derived from the final DFG is revealed.

Please note that, inherent to the problem setting, integrity of the returned DFG cannot be guaranteed: A and B could simply alter their own partial log to sabotage the resulting DFG. Thus, we assume that A and B have a common interest in obtaining a correct DFG, but, nevertheless, must treat each other as malicious adversaries.

B. Secrets

The organizations A and B would like to obtain the DFG that would result from them merging their partial event logs; however, without sharing any more information than the final DFG. In particular, A and B would like the following information to remain secret:

- *S1* Whether a given case is in their partial log;
- *S2* The number of events for any case in the partial logs;
- *S3* Any timestamp in their partial log;
- *S4* Any bound on any timestamp in their partial log;
- *S5* Whether a given case has a particular activity in their partial log;



(a) A sparse DFG.

(b) A less sparse DFG.

Fig. 2: For sparse DFGs, information may leak.

Second, if the DFG is sparse, it may be possible to derive information on the partial logs from the DFG. As a rather extreme example, consider Figure 2a, which shows a DFG of a process that consists of a single sequence. Suppose that A has a partial trace $\langle a, c \rangle$, then it is not hard for A to derive that B must have executed the events b and d for this case (secrets *S1*, *S2*, and *S5*). Furthermore, A can derive that the timestamp of b must lie between the timestamps of a and c (secret *S4*), though, assuming infinite precision, no exact timestamp can be derived (secret *S3* is kept). Again, for three or more involved organizations, it may be possible to keep these secrets for sparse DFGs, but this is beyond the scope of this paper.

We give another example to show that adding even a few arcs will prevent the secrets from being leaked, and that, in order to leak information, the required sparseness is considerable. In Figure 2b, two arcs have been added. Organization A again knows the partial trace $\langle a, c \rangle$, but the DFG does not allow A to distinguish between several options: (i) B may not have this case ID in its partial log at all, (ii) B may have the partial trace $\langle b, d \rangle$, (iii) B may have the partial trace $\langle b \rangle$, or (iv) B may have the partial trace $\langle d \rangle$. As A cannot distinguish between these options, secrets *S1*, *S2*, *S4* and *S5* are kept. Given that in real-life cases DFGs are rarely sparse [7], we conclude that in practice, DFGs by themselves leak little information. A potential rule of thumb would be for an organization not to engage in federated process discovery if the DFG of its partial log has activities with fewer than two outgoing arcs. Alternatively, noise could be added to avoid revealing secrets.

III. Related Work

Recently, works on the crossroads of inter-organizational process mining, privacy, differential privacy, and confidentiality have received considerable research interest; we refer to [8], [9] and [10] for an elaborate overview. The need for inter-organizational process mining was identified in [11], and inter-organizational process mining has recently been covered in an extensive literature review [12].

An overview of related work in inter-organizational process discovery is given in Table I. In the following, we categorize approaches into those using abstractions, a third party, or decentralization. This section is concluded by a discussion of limitations and secret preservation between the related work and the newly introduced approaches, i.e., BF and TB. While we do not discuss information leaked by directly-follows graph, [13] covers such issues and mitigation strategies.

Abstraction-based discovery. [14] is an early work using homomorphic encryption (HE) that introduces a confidentiality framework for single-organization privacy-preserving process discovery and uses abstractions. [15] proposes an inter-organizational Alpha Miner in settings without handover-of-work between organizations, sharing footprint matrices as abstractions. Having no handover-of-work is a limitation that does not occur for us and that reduces the complexity of the inter-organizational setting. On the opposite, [4] necessitates handover information to merge local DFGs. Necessitating handover-of-work is also a limitation that is not given in our context and that expects a message flow to be transparent and pre-defined, thus leaking *S1*, *S4*, and *S5*. Similarly, [16] discovers and connects private BPMN models using message tasks. While activity ordering is abstracted using bag-of-words, their frequency per case is publicly shared, thus leaking *S1*, *S2*, *S4*, and *S5*.

With a third party. In [5], using HE an encrypted time-annotated DFG is constructed that can be used for performance analysis queries. Further, the approach requires a third party, is vulnerable to collusion attacks that break confidentiality, and requires activities between organizations to be disjoint, whereas our approach is not susceptible to collusion and allows any distribution of activities over participating organizations. On the other hand, it would be interesting to consider [5] for our approaches, potentially providing FHE-based confidentiality between two or more parties, while allowing for protected performance queries.

[3] proposes a differentially private discovery using the Heuristics miner. Similar to [15], [3] requires no handover-of-work as a limitation to the setting. While not particularly designed for confidentiality, [17] discovers and optimizes deterministic finite frequency automata on partial logs to merge them on a server, obtaining DFGs that they define as interesting, potentially deviating from traditionally discovered DFGs.

Decentralized Discovery. In [18], trusted execution environments, which are dedicated hardware, are used for decentralized process mining. In a multi-party computation fashion, organizations together compute a BPMN using Heuristics Miner and thus do not need a third party.

Limitations. In Table I, we compare our protocols and the related work in terms of their secrecy on the Secrets *S1* - *S5*, defined in Section II, and further restrictions by design.

The newly introduced trace-based TB approach compares to the works in [5] and [18] in terms of confidentiality. Compared to TB, [5] is not robust in encryption as their multi-party computation further expects an honest majority. [18] is limited by the dedicated hardware for the trusted execution environments. Overall, our TB approach eliminates the need for trust in the correct execution of the collaborating organizations as they cannot corrupt the computations using corrupted trusted execution environments or convincing a dishonest majority in the multi-party computation due to the nature of our suggested two-party fully homomorphically encrypted computation scenario.

To the best of our knowledge, the newly introduced brute-force BF approach is the only approach that covers all secrets to remain private as the approaches in [17], [15] or [3] either do not guarantee rediscovering traditional process models that are identical to as if the data was shared or as they only aggregate independent event logs, i.e., event logs of the same process but that do not contain interactions across organizations, respectively.

IV. Preliminaries

a) Basic Notations.: A multiset generalizes a set, allowing multiple occurrences of an element, e.g., $[x, y, z, x, x, z] = [x^3, y, z^2]$. By $\mathbb{M}(X)$, we denote the set of all multisets over a set X. A sequence allows for multiple occurrences of the same element while having a *total order* for all its elements, e.g., $\sigma = \langle x, x, y, x, z\rangle \in X^*$ is a sequence over $X = \{x, y, z\}$ with X^* as the set's Kleene closure. By $\sigma \cdot \sigma'$, we denote the trace concatenation of traces σ and σ', e.g., $\langle x, y, z\rangle \cdot \langle y, z\rangle = \langle x, y, z, y, z\rangle$. Let X be a set, $Y \in \mathbb{M}(X)$ be a multiset over X and $X' \subseteq X$ a subset of X. The multiset-set difference between Y and X' is $Y \setminus X' = [y \in Y \mid y \notin X']$, e.g., $[x^3, y, z^2] \setminus \{y, z\} = [x^3]$. The multiset union of X and X' is $X \uplus X'$, e.g., $[x^3, y, z^2] \uplus [y^2, z^3] = [x^3, y^3, z^5]$. By $X(x)$, we denote the frequency of the element $x \in X$ in the multiset X, e.g., $X(y) = 3$ for $X = [x^2, y^3, z^5]$.

b) Event Data.: In process mining, event data is used to keep track of what is happening within the execution of a process. In particular, an event includes an activity taking place for a single process instance at a specific time. Thus, we define $\mathcal{U}_c$ as the universe of case IDs, $\mathcal{U}_a$ as the universe of activities, and $\mathcal{U}_t$ as the universe of timestamps. An event e from the universe of events, i.e., $e \in \mathcal{U}_e$, is a tuple consisting of a case ID, an activity, and a timestamp, i.e., $e \in \mathcal{U}_c \times \mathcal{U}_a \times \mathcal{U}_t$. A trace $\sigma \in \mathcal{U}_e^*$ is a sequence of events that all share the same case ID c and that are ordered by their timestamps, e.g., $\sigma = \langle (3, a, 15), (3, b, 19), (3, a, 24), (3, c, 30)\rangle$ is a trace. We also use $\sigma = \langle a^{@15}, b^{@19}, a^{@24}, c^{@30}\rangle^{3}$ as a shorthand notation. We define $\Sigma\colon \mathcal{U}_e \to \mathcal{U}_a$ as the activity function, $t\colon \mathcal{U}_e \to \mathcal{U}_t$ as the timestamp function, and $id\colon \mathcal{U}_e \cup \mathcal{U}_e^* \to \mathcal{U}_c$ as the case ID function, e.g., for an event $e = (3, a, 15) \in \mathcal{U}_e$ we obtain $\Sigma(e) = a$, $t(e) = 15$, $id(e) = 3$, and $id(\sigma^{ex}) = 3$. An event log L is a set of traces, i.e., $L \subset \mathcal{U}_e^*$.

c) Federated Event Data.: Although a number of organizations can collaborate in an inter-organizational process, this paper focuses for simplicity on two organizations $A, B \in \mathcal{U}_o$, where $\mathcal{U}_o$ represents the universe of organization identifiers. An organization's partial event log $A \in \mathcal{U}_o$ is denoted as L_A. Example 1 shows two partial event logs and their combination.

d) Directly-Follows Graph (DFG).: Let $A \subseteq \mathcal{U}_a$ be a set of activities, ▶ and ■, for which $\{▶, ■\} \cap A = \emptyset$ holds, be a reserved start and end activity, respectively, and $F \in \mathbb{M}((A \cup \{▶\}) \times (A \cup \{■\}))$ be a multiset of arcs. We define the weighted directed graph resulting from the pair $G = (A, F)$ for which its frequencies match, i.e., $\forall_{a \in A \setminus \{▶, ■\}}\colon (\sum_{a' \in A} F(a', a) = \sum_{a' \in A} F(a, a'))$, as a directly-follows graph (DFG). An example directly-follows graph is shown in Figure 2b.

TABLE I: Overview of related work.

Work	discovered model	technique	*S1 S2 S3 S4 S5*	collaborative setting	further limitation
[15]	Petri net	abstraction, footprint matrix, α-Miner	✓ ✓ ✓ ✓ ✓	no handover-of-work	limited expressiveness of workflow
[4]	DFG	handover, abstraction	✗ ✓ ✓ ✗ ✗	message passing	
[16]	BPMN	bag-of-words, abstraction	✗ ✗ ✓ ✗ ✗	message passing	
[17]	uncertain DFG	DFFA, stochastic discovery, third party	✓ ✓ ✓ ✓ ✓	any	no correctness guarantee
[5]	time-annotated DFG	FHE, querying, third Party	✗ ✗ ✓ ✓ ✓	any	honest majority, $\geq$3 parties
[3]	Petri net	Heuristics Miner, third party	✓ ✓ ✓ ✓ ✓	no handover-of-work	independent event logs
[18]	Petri net	TEE, third Party	✗ ✗ ✓ ✓ ✓	any	TEE hardware
BF (this paper)	DFG	FHE	✓ ✓ ✓ ✓ ✓	any	
TB (this paper)	DFG	FHE	✗ ✗ ✓ ✓ ✓	any	

e) Homomorphic Encryption.: We secure federated event data with homomorphic encryption, converting plaintext into ciphertext. Homomorphic encryption allows for arithmetic and logical operations to be performed on ciphertext. *Fully homomorphic encryption* (FHE) is a type of homomorphic encryption, enabling an unlimited number of operations (arithmetical or logical) to be performed arbitrary often.

More formally, an encryption scheme is called *homomorphic* over an operation $\bigstar$ if $\forall_{m_1,m_2 \in M} \colon \mathbb{a}(m_1) \bigstar \mathbb{a}(m_2) = \mathbb{a}(m_1 \bigstar m_2)$ holds, where $\mathbb{a}$ is the encryption algorithm and M is the set of all possible messages. While we indicate the encryption function by $\mathbb{a}$ or an encrypted value v throughout the paper by $\mathbb{a}(v)$, we indicate the decryption function by 🔑, i.e., $🔑(\mathbb{a}(v)) = v$. For more details on homomorphic encryption in general, we refer the reader to [19].

This paper utilizes the TFHE scheme from [20], enabling operations like addition, evaluation of conditional *if then else* statements, the minimum and maximum functions $min(a, b)$ and $max(a, b)$ between either a ciphertext and a plaintext or two ciphertexts only. Introduced errors by applying the TFHE operations are mitigated using bootstrapping to avoid errors with a high probability. The security of the TFHE scheme is given by the properties that a single plaintext can be represented by multiple ciphertexts.

V. Federated Directly-Follows Graph Discovery

This section explains how to discover a directly-follows graph from federated event data without sharing event logs. We present two protocols: a highly secure but costly brute-force approach — BF, and a less costly, more information-leaking trace-based method — TB. Approach BF verifies case ID matches using encrypted comparisons, whereas TB makes case IDs transparent for all events in shared cases. That is, TB reveals *S1* and trades it for faster computations.

While they have to negotiate their roles upfront, without loss of generality, we assume that organization A holds the private key, which should never be shared, and organization B applies the computations on the encrypted data. Next, we introduce the brute-force protocol in Section V-A first and then present its adaptation to the trace-based protocol in Section V-C. In Section V-B, we present the subprotocol used in both protocols to obtain all encrypted directly-follows edges from an encrypted sequence of events from A and an unencrypted sequence of events from B. Section V-D analyses complexity.

The protocols are outlined as follows: (1) all activity counts are communicated, (2) for TB only, determine common case IDs, then repeat the next step for each case ID, and (3) using homomorphic operations, find directly-following events in a case, collect all encrypted directly-follows edges, send them over, and conclude their DFG.

A. Brute-Force Protocol (BF)

In this section, we introduce the brute-force protocol (BF). Intuitively, in BF, A creates a single encrypted sequence of its events and sends it to B. B then computes for each event in the sequences of A and B its successor using a nested *if then else* logic. The obtained encrypted edges are all stored within B. B shuffles all these encrypted edges and sends them to A, who decrypts them and obtains the DFG. Algorithm 1 shows the protocol, and we proceed with explaining its details.

First, A and B negotiate an activity mapping M, which maps the name of each activity to a unique natural number $[0, |\Sigma|)$, as well as an occurrence map $N \colon \Sigma \to \mathbb{N}$, which indicates how often each activity will appear in the final DFG (line 3). The parties should not engage in the protocol if N is too low for any activity, e.g., lower than 10, if $1/10$ provides enough uncertainty. This ensures that A has limited opportunity to insert additional activities to obtain information on B's traces, and that reversely B has no opportunity to insert additional activities to obtain information on A's traces (see *S1*, *S4* or *S5* in Section II).

Both A and B add artificial start ▶ and end ■ events to each trace and give these events appropriate extreme timestamps: the start event at a moment in the far history, and the end event in the far future. Then, both A and B create a list of events, where each event is represented by a tuple of its case ID, its timestamp and its activity. A encrypts this list, and both stably sort their list: first by case ID, and in case of equal case IDs, by timestamp (lines 4 and 6).

Example 2 (Preprocessing): Consider two partial event logs L_A and L_B from Example 1. Then, A and B negotiate an activity mapping $M = ▶ \mapsto 0, ■ \mapsto 1, a \mapsto 2, b \mapsto 3, c \mapsto 4$, and an occurrence mapping $N = a \mapsto 2, b \mapsto 3, c \mapsto 1$, after which they each construct their sorted list:

$$\overrightarrow{L_A} = \langle (\mathbb{a}(002), \mathbb{a}(-\infty), \mathbb{a}(M(▶))), (\mathbb{a}(002), \mathbb{a}(1), \mathbb{a}(M(a))), (\mathbb{a}(002), \mathbb{a}(2), \mathbb{a}(M(b))), (\mathbb{a}(002), \mathbb{a}(\infty), \mathbb{a}(M(■))), (\mathbb{a}(003), \mathbb{a}(-\infty), \mathbb{a}(M(▶))), (\mathbb{a}(003), \mathbb{a}(2), \mathbb{a}(M(b))), (\mathbb{a}(003), \mathbb{a}(3), \mathbb{a}(M(a))), (\mathbb{a}(003), \mathbb{a}(\infty), \mathbb{a}(M(■)))\rangle$$

$$\overrightarrow{L_B} = \langle (003, -\infty, M(▶)), (003, 1, M(c)), (003, 4, M(b)), (003, \infty, M(■))\rangle$$

After creation, in line 5 of Algorithm 1, $\overrightarrow{L_A}$ is sent to B. B sanitizes the activities sent by A to ensure they fall within the mapping M (line 7), thus preventing A from adding hidden information in the activities (see *S1* or *S4* in Section II).

Algorithm 1 Federated DFG Discovery - brute-force (executing organization is given at the start of each line and by color)

1: **procedure** JOINT PROTOCOL BF(partial log L_A, partial log L_B) between Organization A and Organization B
2: A: sends its public key to B
3: $A \leftrightarrow B$: $M, N \leftarrow$ negotiate activity mapping and occurrence map
4: A: $\overrightarrow{L_A} \leftarrow \langle(\text{🔒}(\text{id}(\sigma)), \text{🔒}(t(e)), \text{🔒}(M(\Sigma(e)))) \mid e \in \sigma, \sigma \in L_A\rangle \cdot \langle(\text{🔒}(\text{id}(\sigma)), \text{🔒}(-\infty), \text{🔒}(\blacktriangleright)), (\text{🔒}(\text{id}(\sigma)), \text{🔒}(\infty), \text{🔒}(\blacksquare)) \mid \sigma \in L_A\rangle$ stably sorted by $\text{id}(\sigma)$ and $t(e)$
5: A: sends $\overrightarrow{L_A}$ to B
6: B: $\overrightarrow{L_B} \leftarrow \langle(\text{id}(\sigma), t(e), M(\Sigma(e))) \mid e \in \sigma, \sigma \in L_B\rangle \cdot \langle(\text{id}(\sigma), -\infty, \blacktriangleright), (\text{id}(\sigma), \infty, \blacksquare) \mid \sigma \in L_B\rangle$ stably sorted by $\text{id}(\sigma)$ and $t(e)$
7: B: $\overrightarrow{L_A} \leftarrow \langle(\text{🔒}(id), \text{🔒}(t), \min(\text{🔒}(a), |M|) \mid (\text{🔒}(id), \text{🔒}(t), \text{🔒}(a)) \in \overrightarrow{L_A}\rangle$ ▷ sanitize $\overrightarrow{L_A}$
8: B: $DFG \leftarrow$ DFG EDGES($\overrightarrow{L_A}$, $\overrightarrow{L_B}$) ▷ compute encrypted edges
9: B: shuffles DFG and sends it to A
10: A: $DFG \leftarrow [(\text{🔑}(\text{🔒}(a)), \text{🔑}(\text{🔒}(b))) \mid (\text{🔒}(a), \text{🔒}(b)) \in DFG]$ ▷ decrypt
11: A: $DFG \leftarrow DFG \setminus \{(\blacktriangleright, \blacktriangleright), (\blacksquare, \blacktriangleright), (\blacksquare, \blacksquare)\}$ ▷ remove artifacts
12: A: $DFG \leftarrow [(M^{-1}(\min(a, |M|)), M^{-1}(\min(a', |M|))) \mid (a, a') \in DFG]$ ▷ sanitize
13: A: verify $\forall_{a \in \Sigma} \sum_{a' \in \Sigma \cup \{\blacktriangleright\}} DFG((a', a)) = \sum_{a' \in \Sigma \cup \{\blacksquare\}} DFG((a, a')) = N(a)$
14: A: **return** DFG
15: **end procedure**

Next, B computes the DFG edges of the two lists $\overrightarrow{L_A}$ and $\overrightarrow{L_B}$ (line 8), using the subroutine that is described in Section V-B. B randomly shuffles the encrypted DFG edges to prevent A from deducing their order (line 9) and sends them to A (line 9) for decryption with its private key (line 10).

A performs some control steps: edges between ends, from end to start or between starts are removed (line 11), and the edges are sanitized to give B reduced opportunity to insert activities and deduce information on A's events (line 12). The last check performed by A is whether each activity appears in the DFG precisely as many times as agreed upon beforehand in the occurrence map negotiation (lines 3 and 13). If this is not the case, then either the DFG is incorrect, or B attempted to perform an attack. Finally, the DFG is returned.

Example 3: Continuing with our example, Figure 5a shows the DFG first obtained by A, and Figure 5c shows the DFG after filtering, which is shared with B.

B. Subroutine of BF and TB: Constructing DFG Edges

Next B, the public key organization, transforms the two lists of case-ID-timestamp-activity tuples $\overrightarrow{L_A}$ and $\overrightarrow{L_B}$. These lists together form a partial order: the lists themselves are totally ordered, but the order between the inter-elements is unknown. Nevertheless, the combined list forms an unknown total order, thus, every event in the list has exactly one outgoing DFG edge (except the last one).

Consider an event tuple (id_i, t_i, a_i) and its successor $(id_{i+1}, t_{i+1}, a_{i+1})$ in $\overrightarrow{L_A}$ with $|\overrightarrow{L_A}| = l$. Then, this event tuple was followed directly by precisely one other event: either $(id_{i+1}, t_{i+1}, a_{i+1})$ or any $(id'_j, t'_j, a'_j) \in \overrightarrow{L_B}$ with $|\overrightarrow{L_B}| = l'$. We define a recursive function φ to determine the next activity following the position i of an event tuple, starting with $j = 0$:

$$\varphi(l, l') = \blacksquare$$

$$\varphi(i, l') = \begin{cases} a_{i+1} & \text{if } id_i = id_{i+1} \\ \blacksquare & \text{otherwise} \end{cases}$$

$$\varphi(l, j) = \begin{cases} a'_j & \text{if } t'_j > t_l \wedge id_i = id'_j \\ \varphi(l, j+1) & \text{otherwise} \end{cases}$$

$$\varphi(i, j) = \begin{cases} a'_j & \text{if } t'_j > t_i \wedge t'_j < t_{i+1} \wedge id_i = id'_j \\ a_{i+1} & \text{if } t'_j > t_i \wedge t'_j \geq t_{i+1} \wedge id_i = id_{i+1} \\ \varphi(i, j+1) & \text{otherwise} \end{cases}$$

Note: encryption indicators have been removed for readability as everything from $\overrightarrow{L_A}$ is encrypted. The outgoing edge of a_i in (id_i, t_i, a_i) is obtained by evaluating $(a_i, \varphi(i, 0))$. For the trace-based protocol, case IDs are already agreed upon upfront and therefore, for the φ function, they are evaluated to be agreeing if absent. φ' is symmetric to φ for event tuples of B.

Given that the event tuples in $\overrightarrow{L_A}$ are encrypted, B cannot make the necessary comparisons directly. Instead, B constructs a tree of conditional (*if then else*) expressions according to φ, which evaluates to the correct, encrypted edge. On the one hand, B cannot know the resulting edge, as this is encrypted. On the other hand, A only obtains the edge, not the conditional tree it was derived from. Since the conditional operations are homomorphic, no termination criteria can be formulated, requiring always the evaluation of the full recursion.

For given $\overrightarrow{L_A} = \langle(id_0, t_0, a_0), \ldots (id_n, t_n, a_n)\rangle$ and $\overrightarrow{L_B} = \langle(id'_0, t'_0, a'_0), \ldots (id'_m, t'_m, a'_m)\rangle$, B apply:

$$\text{DFG EDGES}(\overrightarrow{L_A}, \overrightarrow{L_B}) = [(\text{🔒}(a_i), \varphi(i, 0)) \mid 0 \leq i \leq n] \uplus [(\text{🔒}(a'_j), \varphi'(0, j)) \mid 0 \leq j \leq m] \quad (1)$$

Example 4 (B applies DFG EDGES($\overrightarrow{L_A}, \overrightarrow{L_B}$)*):* B has obtained $\overrightarrow{L_A}$ and $\overrightarrow{L_B}$ of Example 2, and these two lists yield a partial order, as shown in Figure 3. For each event tuple in both lists, B computes its successor, i.e., the next node in the combined trace and thus the outgoing directly-follows edge.

For instance, consider the second event tuple in $\overrightarrow{L_A}$: $(\text{🔒}(002), \text{🔒}(1), \text{🔒}(a))$. The subsequent event can either be the third event tuple of $\overrightarrow{L_A}$, or any event tuple of $\overrightarrow{L_B}$.

The FHE expression of the corresponding DFG edge is $(\text{🔒}(a), \varphi(1, 0)))$ with:

$\varphi(1, 0) =$ *if* $t'_0 > \text{🔒}(t_1) \wedge t'_0 < \text{🔒}(t_2) \wedge \text{🔒}(id_i) = id'_0$ then $\text{🔒}(a'_0)$
else if $t'_0 > \text{🔒}(t_1) \wedge t'_0 \geq \text{🔒}(t_2) \wedge \text{🔒}(id_1) = id'_0$ then $\text{🔒}(a_2)$
else if $t'_1 > \text{🔒}(t_1) \wedge t'_1 < \text{🔒}(t_2) \wedge \text{🔒}(id_1) = id'_1$ then $\text{🔒}(a'_1)$
else if $t'_1 > \text{🔒}(t_1) \wedge t'_1 \geq \text{🔒}(t_2) \wedge \text{🔒}(id_1) = id'_1$ then $\text{🔒}(a_2)$
else if $\ldots$

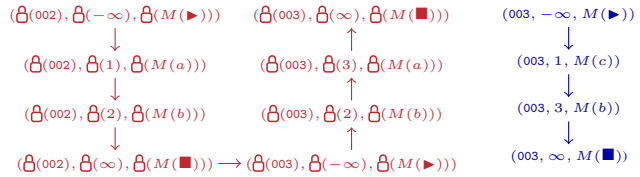


Fig. 3: Example of a partial order of events on our example.

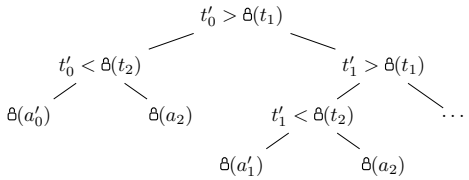


Fig. 4: Internal decision tree when evaluating $\varphi(1, 0)$.

This expression evaluates to $\mathfrak{A}(b)$, from the third event tuple of $\overrightarrow{L_A}$. The FHE expression is represented more visually in Figure 4. The full multiset of edges is shown in Figure 5a. After filtering, A obtains the DFG shown in Figure 5c.

C. Trace-Based Protocol (TB)

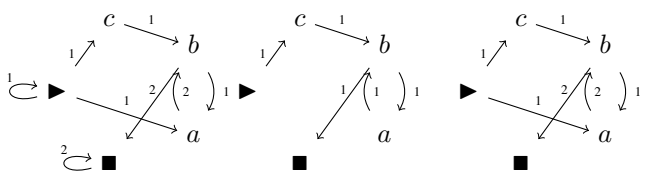
Next, we present a more efficient protocol, which can be applied if the context does not require the secrets *S1* or *S2* to be kept. That is, if the common case IDs and the number of events in each case may be revealed. Intuitively, in the trace-based protocol (TB), first the common case IDs are computed, after which each common case is considered in isolation.

Algorithm 2 shows the protocol. We explain the lines that differ from BF. A private set intersection [21] is applied to obtain the common case IDs (line 3). Non-common cases are inserted by A and B without communication (lines 5 and 15). Furthermore, the event lists do not need to contain start activities, as these can be added to the DFG directly (lines 11) (end activities are added by DFG EDGES).

Example 5: Using TB on our running example, A obtains the DFG (Fig. 5a) and returns the appended DFG (Fig. 5c).

D. Analysis

Both protocols may return a DFG that is not entirely equivalent to the DFG of the combined process: if both A and B have an event with the same timestamp for the same case ID, the protocols have no way to determine which one of these events happened first, in case an ordering exists, and thus they may be swapped, which introduces up to three incorrect edges



(a) BF: As A gets it. (b) TB: As A gets it. (c) As A returns it.

Fig. 5: Steps towards a DFG for our example.

in the DFG. If this situation does not occur, by construction, the correct DFG of the combined process is returned.

For the complexity of TB we only focus on the function DFG EDGES. Let $|L_A| = n$ and $|L_B| = m$. Let k be an upper bound such that for each trace $\sigma \in L_A \cup L_B$, $|\sigma| \leq k$. The complexity of the TB approach is in the order of $O(k \cdot (m+n))$.

Let $|\overrightarrow{L_A}| = l$ and $|\overrightarrow{L_B}| = l'$, then the complexity of the BF approach is in the order of $O(l \cdot l')$.

We identified one attack on both protocols: the *adding-activities attack*. Say that B is a malicious organization, which convinces A to include an additional activity x into Σ_B. For each such added activity x, B can insert one event of x into one of its traces, and perform the protocol (BF or TB). In the resulting DFG, x will have exactly one incoming and one outgoing edge in the DFG, and thus B may be able to derive which activities were executed by A before and after x for that particular case (*S3*, *S4*, *S5*). B can also derive whether A executed any event for the trace (*S1*).

For this attack, B needs to (i) convince A to insert x, (ii) guess a case ID that A has in its partial log, and (iii) guess a timestamp for x. Note that the activity x can be used only for one trace by B, thus the amount of leaked information is limited. This attack is mitigated through a pre-negotiation step in which the number of activities should be minimized, and both parties should not engage in the protocol if there are activities that appear less than a handful of times (lines 3 and 13 in Algorithm 1 and lines 4 and 16 in Algorithm 2).

VI. EVALUATION

In this section, we evaluate our approach threefold: we describe its implementation, we illustrate its keeping of secrets, and we show its practical computational complexity.

A. Implementation

TB (Section V-C) was implemented using [22] and is publicly available (https://github.com/cRennert/Federated-Directly-Follows-Graph-Discovery). To save memory in organization B, the traces are processed in batches of 100 traces, such that B only has to keep the encrypted edges of 100 traces in memory at any one time. The implementation parallelizes computations wherever possible and uses the TFHE-rs FHE library (see https://github.com/zama-ai/tfhe-rs) with 128 bits of security and an error probability of 2^{-128}.

B. Illustration

As a use case, imagine a general practitioner (GP) and a hospital that want to learn the process for their shared patients. This is interesting for cases in which patients may retake examinations or in which they do not take an examination or a treatment that is prescribed. Upfront, they agree on the personal identification number (e.g., the BSN in the Netherlands) as their case ID. The hospital and the GP apply BF. Fig. 6 illustrates this: both create a total order of all their events, which the hospital sends to the GP encrypted. Then, the GP computes its parts of the protocol and sends them back to the hospital, still encrypted. The hospital then decrypts and

Algorithm 2 Federated DFG Discovery - trace-based (executing organization is given at the start of each line and by color)

procedure JOINT PROTOCOL TB(partial log L_A, partial log L_B) between Organization A and Organization B
  A: sends its public key to B
  $A \leftrightarrow B$: $C_{\text{pub}} \leftarrow$ private set intersection on case IDs
  $A \leftrightarrow B$: $M, N \leftarrow$ negotiate activity mapping and occurrence map
  B: $DFG \leftarrow [(\text{🔒}(a), \text{🔒}(a')) \mid \langle \ldots a, a', \ldots \rangle = \sigma, \sigma \in L_B, \text{id}(\sigma) \notin C_{\text{pub}}] \uplus [(\text{🔒}(\blacktriangleright), \text{🔒}(a)) \mid \langle a, \ldots \rangle = \sigma, \sigma \in L_B, \text{id}(\sigma) \notin C_{\text{pub}}]$
    $\uplus [(\text{🔒}(a), \text{🔒}(\blacksquare)) \mid \langle \ldots, a \rangle = \sigma, \sigma \in L_B, \text{id}(\sigma) \notin C_{\text{pub}}]$ ▷ B's own DFG edges
  for $\sigma_A \in L_A, \sigma_B \in L_B, \text{id}(\sigma_A) = \text{id}(\sigma_B)$ **do** ▷ for each common trace
    A: $\overrightarrow{L_A} \leftarrow \langle (\text{🔒}(t(e)), \text{🔒}(M(\Sigma(e)))) \mid e \in \sigma_A \rangle$ and sends it to B ▷ sending over list of A
    B: $\overrightarrow{L_B} \leftarrow \langle (t(e), M(\Sigma(e))) \mid e \in \sigma_B \rangle$ ▷ list of B
    B: $\overrightarrow{L_A} \leftarrow \langle (t, \min(a, |M|)) \mid (t, a) \in \overrightarrow{L_A} \rangle$ ▷ sanitize $\overrightarrow{L_A}$
    B: $DFG \leftarrow DFG \uplus \text{DFG EDGES}(\overrightarrow{L_A}, \overrightarrow{L_B})$ ▷ compute encrypted edges
    B: $DFG \leftarrow DFG \uplus [\textit{if } \text{🔒}(t) \leq t' \textit{ then } (\text{🔒}(\blacktriangleright), \text{🔒}(a)) \textit{ else } (\text{🔒}(\blacktriangleright), \text{🔒}(a')) \mid$
      $\langle (\text{🔒}(t), \text{🔒}(a)), \ldots \rangle = \overrightarrow{T_A}, \langle (t', a'), \ldots \rangle = \overrightarrow{T_B}]$ ▷ add start activity
  end for
  B: shuffles DFG and sends it to A
  A: $DFG \leftarrow [(M(\min(\text{🔑}(a), |M|)), M(\min(\text{🔑}(b), |M|))) \mid (a, b) \in DFG]$ ▷ decrypt and sanitize
  A: $DFG \leftarrow DFG \uplus [(a, a') \mid \langle \ldots a, a', \ldots \rangle = \sigma, \sigma \in L_A, \text{id}(\sigma) \notin C_{\text{pub}}] \uplus [(\blacktriangleright, a) \mid \langle a, \ldots \rangle = \sigma, \sigma \in L_A, \text{id}(\sigma) \notin C_{\text{pub}}]$
    $\uplus [(a, \blacksquare) \mid \langle \ldots, a \rangle = \sigma, \sigma \in L_A, \text{id}(\sigma) \notin C_{\text{pub}}]$ ▷ A's own DFG edges
  A: verify $\forall_{a \in \Sigma} \sum_{a' \in \Sigma \cup \{\blacktriangleright\}} DFG((a', a)) = \sum_{a' \in \Sigma \cup \{\blacksquare\}} DFG((a, a')) = N(a)$
  A: **return** DFG
end procedure

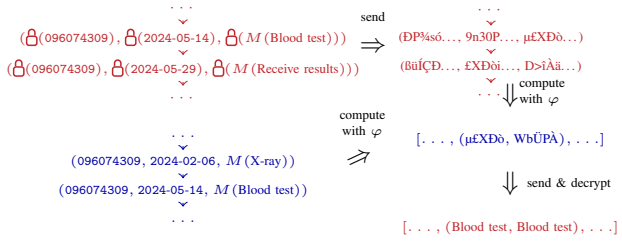


Fig. 6: Illustrative application of the BF protocol.

obtains the DFG, which shows a self-edge on the blood test. Without knowing which patient took a blood test twice, they can engage in a system-wide improvement effort to reduce the number of double procedures, for instance by improving communication between parties.

C. Practical complexity & correctness

We evaluate the practical complexity and correctness of our methods using 10 real-life event logs. First, we split each log into two partial event logs by distributing the events over both partial logs randomly.

To test complexity, we apply TB and BF with disabled encryption to the partial logs, to measure the number of FHE timestamp comparisons and conditional operations. For TB, the private set intersection (PSI) FHE operations were recorded separately. To illustrate the actual time required for a key algorithmic step, we measure the run time of TB with FHE encryption without PSI. For this step, we used 256 GB of RAM and an AMD Ryzen 5965WX CPU; networking delays were not taken into account.

To test correctness, the obtained DFGs are compared with the DFG of the original event log, by counting absolute edge differences using [23]. We count potentially incorrect edges, i.e., all pairs of consecutive events with identical timestamps.

As event logs, we consider the publicly available logs Hospital log, BPIC 12, Open Problems (BPIC 13), Closed Problems (BPIC 13), Incidents (BPIC 13), Road Traffic Fine Management Process, Sepsis Cases, BPIC12, BPIC18, BPIC17 Offers and BPIC19. For the original event logs, we refer to the IEEE task force on process mining. All split event logs are available in our repository.

The results are shown in Table II. For TB, we observe that the PSI step, that is, figuring out what case IDs the organizations have in common, can be considerably more costly (3000 times) than the other parts of the protocol. However, if there are few cases of a long length (such as in the Hospital Log), the PSI step may require fewer operations. The number of operations required for BF is again a couple of orders of magnitude larger.

To put things into perspective, we measured the run time of TB, without the PSI step. The run time measures show that, if the PSI step can be avoided, for instance by negotiating it separately, TB is feasible in smaller real-life settings. On our machine, the number of FHE operations per second ranged from 9-16, with the larger logs being quicker, probably due to higher parallelization. While GPU-based FHE (~20% of run time per FHE operation) or dedicated FHE hardware (such as AMD Alveo v80; ~15% of runtime) may increase the number of FHE operations that can be performed per time unit, it is clear that for larger logs, considerably more hardware than our single workstation would be necessary. BF is infeasible even on small logs and requires further research, for instance to leverage lesser-than-fully homomorphic encryption schemes.

With respect to correctness, we manually verified that both TB and BF return the same DFGs consistently. Compared to the DFG of the original event logs, the table shows how many incorrect edges were present. We discuss one of the logs, BPIC12, in more detail: of the 29 differences, 11 were in `A_CANCELLED` being misclassified as an end activity. A manual inspection found several traces in which `A_CANCELLED` and `O_CANCELLED`, having equivalent timestamps, were the

TABLE II: Results on our practical complexity experiment.

Event log	traces	events	TB			BF			
			FHE operations (PSI)	FHE operations (/PSI)	runtime (/PSI)(s)	FHE operations	incorrect / correct edges		equal-cons.time.
Open Problems (BPIC 13)	819	2 351	785 478	9 788	1 094	64 851 674	0 /	3 170	0
Closed Problems (BPIC 13)	1 487	6 660	3 425 587	48 820	4 377	388 977 834	0 /	8 147	0
Sepsis Cases	1 050	15 214	2 178 827	486 155	30 789	1 033 507 494	88 /	16 264	4 447
BPIC 17 Offer Log	42 995	193 849	3 361 138 425	805 884	94 687	256 079 071 575	0 /	236 844	0
Incidents (BPIC 13)	7 554	65 533	105 306 262	1 307 654	110 803	24 587 532 746	0 /	73 087	0
Road Traffic Fines	150 370	561 470	34 303 791 270	2 306 834	195 892	3 239 335 973 589	7 681 /	711 840	12 018
BPIC 12	13 087	262 200	319 096 952	14 815 325	-	268 938 984 434	29 /	275 287	13 995
BPIC 19	251 734	1 595 923	114 280 960 276	74 786 011	-	17 911 107 075 734	20 132 /	1 847 657	233 464
Hospital Log	1 143	150 291	2 483 106	99 440 729	-	65 852 563 634	9 391 /	151 434	130 400
BPIC 18	43 809	2 514 266	3 838 500 771	287 667 431	-	19 891 900 452 605	20 334 /	2 558 075	206 041

last two events, and they were observed in both orders. This seems to suggest that the order in which they were logged was arbitrary. Nevertheless, the experiment confirms that in some cases, edge counts may be slightly different than if one would discover a DFG from the full log, though this happened on average for only 1 out of 10 consecutive equivalent timestamps.

VII. Conclusion

Our paper introduces two protocols that allow discovering a directly-follows graph for processes that are inter-organizationally distributed between two organizations. While both protocols allow discovering a DFG without the partner organization learning about cases, the ordering of events, or event-related attributes directly. They differ in their complexity and their security guarantees. The brute force protocol keeps case IDs and the number of events in each case secret, but has a high complexity, while the trace-based algorithm has a lower complexity but exposes these aspects. The paper also describes the information that is leaked by a DFG itself.

While our experiments show that the runtime for our algorithm is rather high, we expect the algorithm to improve in runtime by using graphics (GPU) or homomorphic processing units (HPU) and to exploit a non-fully homomorphic encryption scheme to reduce runtime. Another approach to further reduce runtime while keeping standards towards confidence might be to use a sliding window for timestamps to reduce the entropy in the data to be compared. However, this needs further investigations on its real-life impact towards *S4*. Further, we expect our introduced techniques to be generalizable to other process mining techniques.

Overall, this work leverages fully homomorphic encryption for inter-organizational process mining. Compared to existing work, it allows for process discovery without prior knowledge on the common process and how process instances interact with both organizations. Further, there is no need for mutual trust in the presented algorithms as no information of the partial event logs is leaked during discovery, nor can it be reconstructed, as the encrypting party and computing party are strictly separated.

References

[1] T. Boymans, R. Vanwersch, Y. Bemelmans, D.-J. Hofstee, C. Wijnands, and L. van Rhijn, "Netwerkgeneeskunde voor mensen met beweegklachten," *Nederlands Tijdschrift voor Geneeskunde*, Nov. 2023.

[2] M. Fassnacht, C. Benz, D. Heinz, J. Leimstoll, and G. Satzger, "Barriers to data sharing among private sector organizations," in *HICSS*, pp. 3695–3704, ScholarSpace, 2023.

[3] A. Khan, A. Ghose, and H. K. Dam, "Cross-silo process mining with federated learning," in *ICSOC*, vol. 13121 of *LNCS*, pp. 612–626, 2021.

[4] M. Rafiei and W. M. P. van der Aalst, "An abstraction-based approach for privacy-aware federated process mining," *IEEE Access*, pp. 33697–33714, 2023.

[5] G. Elkoumy, S. A. Fahrenkrog-Petersen, M. Dumas, P. Laud, A. Pankova, and M. Weidlich, "Secure multi-party computation for inter-organizational process mining," in *BPMDS*, LNBIP, pp. 166–181, 2020.

[6] D. Evans, V. Kolesnikov, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Found. Trends Priv. Secur.*, vol. 2, no. 2-3, pp. 70–246, 2018.

[7] S. J. J. Leemans, E. Poppe, and M. T. Wynn, "Directly follows-based process mining: Exploration & a case study," in *ICPM*, pp. 25–32, IEEE, 2019.

[8] F. Mannhardt, "Responsible process mining," in *Process Mining Handbook*, vol. 448 of *LNBIP*, pp. 373–401, Springer, 2022.

[9] I. Ileri, T. G. Erdogan, and A. K. Tarhan, "Privacy for process mining: A systematic literature review," *IEEE Access*, vol. 13, pp. 83171–83194, 2025.

[10] G. Elkoumy, S. A. Fahrenkrog-Petersen, M. F. Sani, A. Koschmider, F. Mannhardt, S. N. von Voigt, M. Rafiei, and L. von Waldthausen, "Privacy and confidentiality in process mining: Threats and research challenges," *ACM Trans. Manag. Inf. Syst.*, vol. 13, no. 1, pp. 11:1–11:17, 2022.

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

[12] J. Rott, M. Böhm, and H. Krcmar, "Laying the ground for future cross-organizational process mining research and application: a literature review," *Bus. Process. Manag. J.*, vol. 30, no. 8, pp. 144–206, 2024.

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

[14] M. Rafiei, L. von Waldthausen, and W. M. P. van der Aalst, "Supporting confidentiality in process mining using abstraction and encryption," in *SIMPDA*, vol. 379 of *LNBIP*, pp. 101–123, Springer, 2019.

[15] R. Gatta, M. Vallati, J. Lenkowicz, C. Masciocchi, F. Cellini, L. Boldrini, C. Fernández-Llatas, V. Valentini, and A. Damiani, "On the feasibility of distributed process mining in healthcare," in *ICCS (5)*, vol. 11540 of *LNCS*, pp. 445–452, Springer, 2019.

[16] J. D. Hernandez-Resendiz, E. Tello-Leal, H. M. Marin-Castro, U. M. Ramirez-Alcocer, and J. A. Mata-Torres, *Merging Event Logs for Inter-organizational Process Mining*, pp. 3–26. Springer, 2021.

[17] H. Zhian, R. Buyya, and A. Polyvyanyy, "Federated stochastic process discovery using grammatical inference," in *CAiSE*, LNCS, 2025.

[18] V. Goretti, D. Basile, L. Barbaro, and C. D. Ciccio, "Trusted execution environment for decentralized process mining," in *CAiSE*, vol. 14663 of *LNCS*, pp. 509–527, Springer, 2024.

[19] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 79:1–79:35, 2018.

[20] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *J. Cryptol.*, vol. 33, no. 1, pp. 34–91, 2020.

[21] H. Chen, K. Laine, and P. Rindal, "Fast private set intersection from homomorphic encryption," in *CCS*, pp. 1243–1255, ACM, 2017.

[22] A. Küsters and W. M. P. van der Aalst, "Rust4pm: A versatile process mining library for when performance matters," in *BPM Demos*, pp. 91–95, CEUR-WS.org, 2024.

[23] S. J. J. Leemans, T. Li, and J. N. van Detten, "Ebi - a stochastic process mining framework," in *ICPM Demos*, vol. 3783, CEUR-WS.org, 2024.