Preserving Privacy and Quality of Service in Complex Event Processing through Event Reordering

Saravana Murthy Palanisamy, Frank Dürr, Muhammad Adnan Tariq, Kurt Rothermel
Institute for Parallel and Distributed Systems
University of Stuttgart, Stuttgart, Germany
firstname.lastname@ipvs.uni-stuttgart.de

ABSTRACT

The Internet of Things (IoT) envisions a huge number of networked sensors connected to the internet. These sensors collect large streams of data which serve as input to wide range of IoT applications and services such as e-health, e-commerce, and automotive services. Complex Event Processing (CEP) is a powerful tool that transforms streams of raw sensor data into meaningful information required by these IoT services. Often these streams of data collected by sensors carry privacy-sensitive information about the user. Thus, protecting privacy is of paramount importance in IoT services based on CEP.

In this paper we present a novel pattern-level access control mechanism for CEP based services that conceals private information while minimizing the impact on useful non-sensitive information required by the services to provide a certain quality of service (QoS). The idea is to reorder events from the event stream to conceal privacy-sensitive event patterns while preserving non-privacy sensitive event patterns to maximize QoS. We propose two approaches, namely an ILP-based approach and a graph-based approach, calculating an optimal reordering of events. Our evaluation results show that these approaches are effective in concealing private patterns without significant loss of QoS.

CCS CONCEPTS

• **Security and privacy** → **Privacy protections**; • **Theory of computation** → *Adversary models*; • **Mathematics of computing** → Solvers;

KEYWORDS

Privacy, Complex Event Processing, Pattern obfuscation, Access control

ACM Reference Format:
Saravana Murthy Palanisamy, Frank Dürr, Muhammad Adnan Tariq, Kurt Rothermel. 2018. Preserving Privacy and Quality of Service in Complex Event Processing through Event Reordering. In *DEBS '18: The 12th ACM International Conference on Distributed and Event-based Systems, June 25–29, 2018, Hamilton, New Zealand.* ACM, Hamilton, Newzealand, 12 pages. https://doi.org/10.1145/3210284.3210296

DEBS '18, June 25–29, 2018, Hamilton, New Zealand

ACM ISBN 978-1-4503-5782-1/18/06…$15.00
https://doi.org/10.1145/3210284.3210296

1 INTRODUCTION

With growing advancements in the Internet of Things (IoT), it is not an exaggeration to say that new IoT applications are released every day. The targets of these IoT applications are diverse ranging from industrial applications such as process automation (Industry 4.0) or smart power grids or e-health applications to consumer applications such as fitness trackers or smart homes. Gartner estimates that by 2020, around 20.8 billion IoT devices will be installed [9], many equipped with sensors to automatically capture huge amounts of sensor data.

Typically, the captured raw sensor data needs to be processed into meaningful information to be useful for applications. Complex Event Processing (CEP) is a popular state of the art technology for processing streams of basic events into meaningful "complex" events based on a set of processing rules [17]. For instance, from the basic events: *speed* > 10 *km/h* and *heart rate* > 120 *bpm*, a fitness tracker can infer the activity "doing sports".

The ability to automatically infer such meaningful information is a doubled-edged sword. On the one hand, meaningful information is required by the application to provide a certain *quality of service* (QoS). If a CEP system falsely detects non-existing events (false positives) or does not detect some actually existing events (false negatives), the QoS will degenerate. For instance, a fitness tracker not detecting activities accurately might annoy and drive away the user.

On the other hand, complex events might be highly *privacy-sensitive*. For instance, researchers from the University of California, San Francisco (UCSF) have shown recently in their *Health eHeart* study that a heart irregularity (*arrhythmia*) can be detected with 97 % accuracy from sensor data collected by an Apple watch [23] when it is paired with an AI algorithm. So although a user wants some complex events to be detected accurately, he/she might not want to share any other privacy-sensitive events. According to a recent survey, 32 % of the survey participants would be willing to measure and share fitness data with their health insurance provider, e.g., to get a premium rebate [24]. However, 73 % are afraid that their premiums would increase since the insurers could infer a certain lifestyle or disease from the shared data. Consequently, effective privacy protection mechanisms are needed to allow users or owners of data to selectively control the sharing of information by not sharing information about privacy-sensitive events while preserving the QoS by sharing information about desired events.

Access control is one prominent technique for protecting privacy in event processing systems. However, classic access control mechanisms protect privacy only at the level of single attributes of data or events [3, 19, 22]. But, sensitive information is often revealed through complex data *patterns* potentially spanning several

attributes. For example, blood sugar level and heart rate might not reveal any useful information when separately analyzed, but might suggest a disease when combined. We call these privacy-sensitive patterns that the user or data owner wants to protect from untrusted parties *private patterns*. In contrast *public patterns* are those patterns that are non-privacy-sensitive and needed by CEP-based services to deliver the offered services.

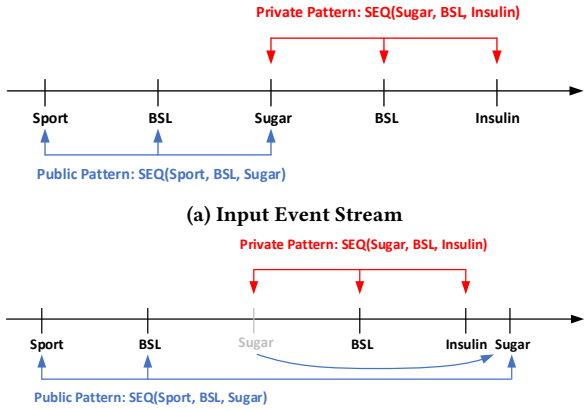
One trivial approach to protect private patterns is to share no data whatsoever. However, this would also make public patterns inaccessible, ruling out any service (zero QoS). Instead the event stream should be selectively *obfuscated* before it is shared with services such that private patterns are not detectable any more from the obfuscated stream while public patterns are preserved as much as possible (maximum QoS).

In this paper, we introduce a pattern-based access control mechanism for a CEP system concelaing private patterns while maximizing QoS by preserving as many public patterns as possible. The first pattern-based access control mechanisms was proposed by Wang et al.[26]. Their approach concealed private patterns by *suppressing* events that are part of the private pattern. However, suppressed events might also be part of public patterns, which results in complete loss of these public patterns, severely impacting QoS.

Therefore, we propose a less intrusive technique, namely, *event reordering* to conceal private patterns defined as sequences of events. For an intuitive explanation of event reordering, consider a CEP application for monitoring diabetes patients as described in [25]. On the one hand, the event stream consists of one private pattern defined as the following event sequence: the patient eats food with a high sugar content, leading to rise of blood-sugar level (BSL), followed by insulin intake (SEQ(High Sugar Intake, BSL high, Insulin)) as shown in Fig. 1a. This private pattern indicating an unhealthy lifestyle when revealed to the insurer might lead to an increase of the insurance premium. On the other hand, the event stream contains a public pattern, in which the patient does some sports deduced from the smart bracelet leading to a drop in BSL and thus eating food with high sugar content (SEQ(Sport, BSL low, High Sugar Intake)). This second pattern should be revealed as positive indication of a healthy lifestyle. Using event reordering, it is possible to conceal the private pattern while preserving the public pattern as shown in Fig. 1b.

In detail, we make the following contributions in this paper: (1) We define a formal *utility metric* that defines the quality of data in terms of public and private patterns. This metric provides the flexibility to specify individual importances for different kinds of patterns. (2) We propose two approaches for reordering the event stream, namely, an Integer Linear Programming (ILP) approach and a graph-based approach, both striving to maximize the utility of the reordered stream. (3) For the evaluation of these approaches, we define a strong adversary model that considers advanced background knowledge of event distributions gained from event histories.

Our evaluations shows that the proposed reordering strategies are better in terms of utility compared to the state-of-the-art suppression strategy. Moreover our evaluations also shows that the reordering strategies are robust against the above mentioned adversary model.



(a) Input Event Stream

(b) Reordered Event Stream

Figure 1: Event Stream Reordering

The rest of the paper is organized as follows. In Sec. 2 we discuss related work followed by the system model and problem statement in Sec. 3. In Sec. 4, we present reordering strategies that try to maximize the utility of reordered streams. In Sec. 5, we describe in detail how the adversary model uses the observations learned from history data to reveal event reordering and describe the evaluation results, before concluding the paper in Sec. 6.

2 RELATED WORK

Several access control mechanisms have been proposed to control the flow of information in stream processing systems [2, 16] and event processing systems [11, 19]. Most of these mechanisms are attribute-based ensuring that certain attributes in the stream of data are only visible to authorized processing operators [16, 19]. However, this is overly restrictive and simplifying at the same time, since certain attributes are either always accessible or not at all, independent of whether they contribute to private or public patterns. Some stream processing systems such as ACStream [3] provide context-based access control, however, still at the level of attributes rather than patterns. Moreover, there exists a rich literature on secure routing of information in event processing systems using the publish/subscribe paradigm. Nevertheless, these publish/subscribe systems [8] enforce access control only at the level of individual events. Patterns of events originated from the correlation of multiple streams are not considered.

In the area of data mining, hiding patterns in sequential databases was considered in [1] and [10] where pattern concealing is performed on a set of sequences that are independent of each other. In contrast, in the context of CEP, pattern-based access control is to be ensured on temporally related event streams. Therefore, their strategies cannot be applied to solve our problem.

Another class of privacy protection in event stream processing is differential privacy [5] and zero-knowledge privacy guarantees [20]. Again most works consider privacy only at the level of single attributes or events. Very few works consider pattern-level privacy. Specifically [13] provides differential privacy guarantees at the level of sequences (patterns). However, although these approaches

provide strong and provable privacy guarantees, they strive to protect the privacy of individual users whose data is part of a dataset from a larger population of users. The altered dataset is still useful for answering queries about the population, while it should be impossible to derive information about individual users. Our goal is fundamentally different. We consider the data of a single user rather than a population of users, and want to preserve uncritical information (public patterns) about this individual user, while concealing privacy-sensitive information (private patterns).

Only few works [11, 26] have been published in the area of pattern-based access control. In [11], He et al. studied the trade-off between quality and privacy in complex event processing (CEP) and analyzed the complexity of minimizing the corresponding degradation in quality theoretically. Their work is very relevant to our vision. However, they consider only suppression of events to conceal patterns. Event suppression was also used by Wang et al. [26] in their approach called Utility Maximizing Event Suppression Scheme. This approach also conceals patterns by suppressing events that are part of the private pattern while maximizing a given service utility. In contrast, we use event reordering rather than suppression resulting in higher QoS with respect to preserved public patterns as shown in Sec. 5.

3 SYSTEM MODEL AND PROBLEM STATEMENT

In this section, we introduce our system model, which also includes our assumptions about the adversary who tries to detect private patterns. Moreover, we define a utility metric defining the utility of an obfuscated event stream, which is then used as an objective in our problem statement.

3.1 System Model

Our system consists of the following components (cf. Fig. 2): producers, consumers, CEP middleware, and pattern-based access control. Next, we will describe these components in more detail.

Producers act as data sources producing basic events as input to the system. Producers can be any type of sensor or also (manual) user input, e.g., entered through the GUI of an application. Basic events generated by producers are typed and contain a set of attributes, which are specified by the corresponding event type, as well as a timestamp. For instance, a blood sugar level (BSL) sensor produce events of type BSL, which contain a blood sugar level attribute and a timestamp. There can be multiple producers but we assume that the event streams of the different producers are merged together into a single input event stream (sequence of events) in timestamp order.

We assume that the user controls the producers. In particular, he/she can control to which other component the producers send their event stream and be sure that no information is "leaked" from these streams. In other words, we do not consider sensors installed somewhere in the environment under the control of an untrusted third party such as a camera network observing pedestrians in public places.

The *CEP middleware* processes basic events into complex events which are then forwarded to the *consumers*, which could be an IoT service provider for example. As *CEP middleware*, we utilize an

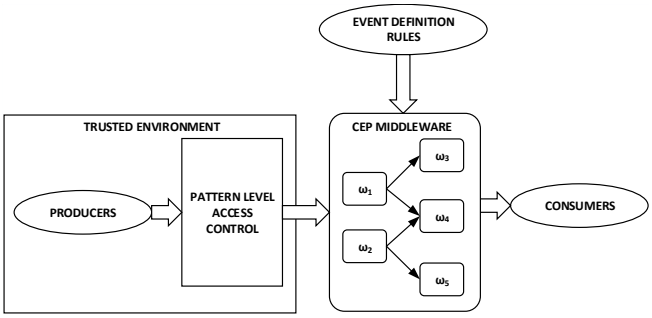


Figure 2: Simplified Model of a CEP System with Pattern-level Access Control component

unmodified standard CEP middleware, for instance, as described in [7]. The CEP middleware is built from a set of interconnected operators. An operator is one specific processing node in the CEP middleware. Its job is to process and transform the incoming event stream according to predefined rules into one or many outgoing event streams. These outgoing event streams are connected either to another processing operator or to a consumer. Typical operators include sequence, aggregation, negation, conjunction, etc. [6]. In this paper, we only consider the sequence operator. The cooperative processing of the different operators results in a complex event. For instance, a high BSL value and a high blood pressure value can result in the complex event "critical health condition".

Typically, operators do not consider the complete (theoretically infinite length) event stream, but rather process the stream in frames of limited duration or length called *windows*. We assume that private patterns fit within windows of given maximum size—to prevent private patterns spanning an arbitrary window size, we would need to store and delay events potentially infinitely, which is not practical. A window can either be time-based or count-based. A time-based window considers all events arriving in a specified period, while a count-based window considers a pre-defined number of events for a match. New windows can also be opened by certain events, in particular, the first event of a sequence of a pattern to be detected. Multiple windows might be open at the same time, e.g., to detect different instances of the same complex event type or different types of complex events. In general, windows might overlap.

A query represents a request to the CEP system and defines the type of a complex event that the consumer is interested in. In this work we only consider—similar to related CEP privacy protection mechanisms [26]—the *sequence* operator, which is one of the most popular CEP operators. A sequence operator captures queries where a set of events arrive in a specific order, thus, both, private and public patterns, are defined as sequences of events. An exemplary sequence denoting the unhealthy behaviour of a diabetes patient (i.e., a private pattern that should be hidden from the untrusted consumer and CEP middleware) could be:

$$Q = SEQ(Eating\_Sugar, High\_Blood\_Sugar\_Level, Insulin\_Intake) \quad (1)$$

This sequence query checks for three events: `Eating_Sugar`, `High_Blood_Sugar_Level`, and `Insulin_Intake`. Moreover the query also defines the ordering relations that the `Eating_Sugar`

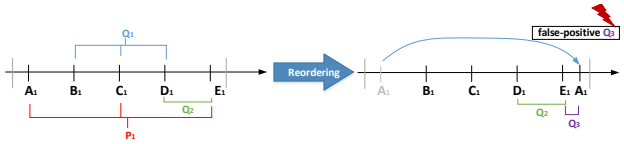


Figure 3: Reordering introducing false positive

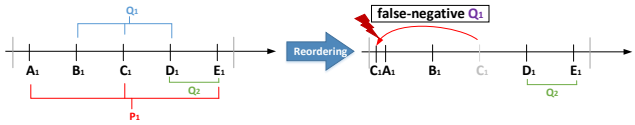


Figure 4: Reordering introducing false negative

event has to occur before the `High_Blood_Sugar_Level` event, and also the `High_Blood_Sugar_Level` event has to occur before the `Insulin_Intake` event. Such an unhealthy behaviour when revealed to the insurance company for example would lead to an increase of their premium rebate. In the following descriptions, we use the following short notation to denote event types and events: A, B, C, etc., denote event types, and A_i denotes the i-th event instance of type A in the event stream. If the type of event is not relevant, we also denote generic events as e_i.

We assume that the CEP middleware as well as the consumers are untrusted and operated by another entity than the user (e.g., the IoT service provider). Moreover, we assume that the user wants to hide certain private patterns expressed as sequences of events from the non-trusted CEP middleware or consumer. Obviously, feeding the unaltered event stream directly into the CEP middleware would enable the middleware to detect all possible sequence events including the private ones. Therefore, we place another component called *pattern-based access control* (PAC) between the producers and the CEP middleware and steer all producer events through the PAC as shown in Fig. 2. The PAC performs event reordering to hide private patterns from the CEP system. Both, producers and PAC, run in a trusted execution environment. There are several options to implement this environment: a trusted physical device, e.g., the smart phone of the user or a trusted server such as a private fog node controlled by the user; a trusted execution environment on a third-party node, e.g., secure "enclaves" in a Cloud server implementing Intel's Software Guard Extensions (SGX) [15] technology where only the authorized user-space process (deployed by the trusted user) can access confidential data (neither other applications, nor the operating system, nor the hypervisor).

It is also important to specify the assumptions about the adversary who tries to compromise the privacy of users by detecting private patterns. We assume a "honest-but-curious" adversary, a common model in security. In our context, this adversary follows all given protocols and does not attack trusted components, but tries to identify private patterns that have been concealed by the PAC. In other words, the adversary might be one of the non-trusted components (CEP middleware or consumer). Consequently, the adversary cannot observe the original event stream, but can observe the complete reordered event stream as sent by the PAC. We also assume that the adversary has some background knowledge including causal and statistical knowledge about event arrivals, e.g., learned from publicly available event streams. In the next sub-section, we will describe the relevant adversary background knowledge in more detail while defining the problem.

3.2 Problem Statement

Informally, the problem solved by our approach is to reorder events in the PAC such that private patterns are hidden from the CEP system while maximizing the QoS of the service acting as complex event consumer. This is a non-trivial problem since there might be different options to reorder events to hide private patterns, each with different impact on QoS. In general, QoS is impacted by false positive events (public events introduced by reordering that never happened) and false negative events (actual public events destroyed by reordering). Fig. 3 and Fig. 4 show examples of a false positive and false negative, respectively. In these examples, $Q_1 = (B, C, D)$, $Q_2 = (D, E)$, and $Q_3 = (E, A)$ are public patterns, and $P_1 = (A, C, E)$ is a private pattern. While concealing P1, the false positive public pattern $Q3$ is introduced into the modified event stream. Moreover public pattern $Q_1 = (B, C, D)$, which is no longer present in the modified event stream is a false negative since its ordering relation (B, C) is no longer satisfied.

In order to define the impact of reordering more precisely, we define the following *utility metric*:

$$\begin{aligned} Utility(U) \quad = \quad & \Sigma_{i=1}^{\#\ of\ matched\ public\ Patterns} w_i \\ & -2 * \Sigma_{j=1}^{\#\ of\ matched\ false\ positives} w_j \\ & -\Sigma_{k=1}^{\#\ of\ matched\ private\ patterns} w_k \end{aligned} \tag{2}$$

Note that the number of matched public patterns (first term) counts both, true and false positives. Therefore, we must subtract two times the number of false positives in the second term, once to remove false positives from the number of matched public patterns, and once to introduce a negative effect for each false positive. Thus, the first and second term increase utility for each true positive match of a public pattern and decrease it for each false positive or negative match of a public pattern. Moreover, we introduce weights w_i and w_j to express the impact of different types of public patterns onto the QoS. Matched private patterns are considered by the third term. Including private pattern matches into the utility metric rather than making it a hard constraint ("no private pattern matches") allows for trading off privacy against QoS, i.e., revealing some private information for more public pattern matches. Here, weight w_k defines the weight of private pattern k according to the following equation:

$$w_k \quad = \quad (\Sigma w_i + 1) * cp_k \tag{3}$$

Tuning parameter cp_k allows for trading off privacy against QoS by specifying a criticality percentage for private pattern k. If we set the private pattern criticality to 100 %, i.e., $cp_k = 1$, then a single match of that private pattern k would outweigh the effect of all (true and false) public pattern matches such that effectively, no

private patterns of that type will be revealed (100 % privacy). For $cp_k = 0$ that type of private pattern is always revealed.

The problem is now to find an event reordering strategy, maximizing the utility according to Equ. 2 and making it (ideally) impossible for the adversary to reveal private patterns that have been concealed (otherwise the number of matched private patterns as specified in Equ. 2 does not represent the private patterns actually revealed to the adversary). Solving this problem requires some assumptions about the background knowledge of the adversary to understand what possibilities the adversary has to detect and reverse event reorderings [21]. We consider two kinds of attacks: causal order constraint attacks and statistical attacks.

Causal order constraint attacks rely on knowledge about the causal relationship between events. We say the order of two events e_1 and e_2 is causally constrained if e_2 must never happen before e_1. For instance, in our e-health scenario the event "insulin injection" never happens before the event "high blood sugar level" but only as a reaction to a high sugar level, i.e., afterwards. Reordering e_1 and e_2 could be detected immediately as a violation of the causality constraints known to the adversary.

Statistical attacks consider the inter-arrival time distribution of events, either between the same type of events, or between different types of events. For instance, through analyzing a publicly available data set, an adversary might know that statistically for patients carefully following their therapy plan, in only 5 % of all cases the time between two insulin injections is smaller than 1h. In contrast to causal order constraints, there is no "binary" violation here, but only a certain probability that reordering has happened.

So besides maximizing the utility through reordering events, the performed event reorderings need to be obfuscated such that an adversary cannot detect and reverse reorderings from causal order constraints or make it unlikely that he detects a reordering with high probability ("confidence") for statistical attacks.

4 EVENT REORDERING APPROACHES

In this section, we describe our approaches to reorder event streams executed by the PAC such that the utility is maximized while not revealing concealed private patterns to adversaries. We present two reordering approaches: one based on Integer Linear Programming (ILP) and one based on graph processing. Both approaches share a common two-phase execution model, which we will present first in the next sub-section, before we describe the specific details of the two approaches in subsequent sub-sections.

4.1 Overview

Event reordering is performed in two phases, namely the utility maximization phase and the reorder obfuscation phase (cf. Fig. 5).

The goal of the *utility maximization phase* (Phase 1) is to find an optimal set of pairs of events that have to be reordered to maximize the utility metric, i.e., to conceal private patterns while also considering the negative effect of reordering on true and false positive public patterns as defined by Equ. 2. Note that it is sufficient to consider only all ordered event pairs that are part of private patterns as candidates for reordering since reordering a single pair of events in a sequence defining a private pattern is sufficient to conceal it. In particular, it is not necessary to iterate over all events

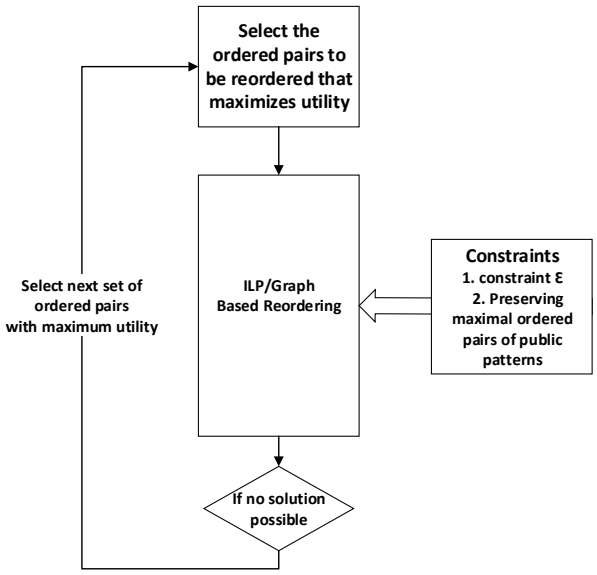


Figure 5: Overview of the Reordering Strategy

of a complete window and check a prohibitive large number of permutations of ordered events. Besides maximizing the utility, the PAC also counters causal order constraint attacks in Phase 1 by not reordering pairs whose order is causally constrained.

In detail, Phase 1 is invoked on a full window of events, whenever this window contains at least one private pattern—obviously, a window without private patterns can be forwarded as it is. The problem of finding an optimal set of event pairs to be reordered is translated to an ILP formulation with the utility metric as objective and exploring all ordered pairs of events of private patterns as reordering candidates. Causally constrained pairs are excluded from reordering by defining corresponding constraints in the ILP. In case of two solutions providing maximum utility, the solution with a smaller number of reordered event pairs is used to break the tie. The output of Phase 1 are two sets of event pairs, where each pair defines an event order: set N contains all pairs of events whose order has to be changed (parts of private patterns), while R includes all pairs of events whose order must be preserved (parts of public patterns).

The set of pairs to be reordered is the input to the *reorder obfuscation phase* (Phase 2). In Phase 2, the PAC shifts the timestamps of events in event pairs selected for reordering in Phase 1, i.e., it actually performs the reordering defined in Phase 1. Since moving event timestamps influences the inter-arrival time distribution of events, the PAC shifts timestamps carefully to avoid statistical attacks, i.e., it obfuscates reorderings such that an adversary cannot detect and reverse them. To this end, we have developed two strategies implementing Phase 2, one based on ILPs and another based on graphs presented in detail in Sec. 4.2 and Sec. 4.3, respectively.

While executing Phase 2, it might turn out that for the selected set of events to be reordered it is infeasible to effectively obfuscate reorderings. In that case, the PAC starts another iteration of Phase 1

and Phase 2, now selecting the second best set of events with respect to utility to be reordered in Phase 1.

4.2 ILP-based Reordering

In this and the next sub-sections, we present two algorithms for obfuscating event reorderings to counter statistical attacks, which detect and reverse event reorderings through an analysis of the event arrival distribution of the reordered event stream. Statistical attacks might become feasible since reordering changes the timestamps of events and thus the inter-arrival time between events.If the observed inter-arrival time between events deviates significantly from the typical inter-arrival time (adversary background knowledge), the adversary can infer with a certain probability ("confidence") that the order of events has been changed.

Obviously, the means to influence the inter-arrival time of events is the time distance $x(e_i)$ by which a reordered event e_i has been shifted. In general, smaller $x(e_i)$ influence the distribution less than greater $x(e_i)$. Therefore, in Phase 2 the PAC strives to minimize the sum of time distances $\sum_{e_i \in E} \mid x(e_i) \mid$ by which all reordered events have been shifted.

Minimizing the sum of time distances keeps the difference between the modified inter-arrival distribution after reordering and the original inter-arrival distribution before reordering small (although there is no hard bound on how much the distribution is actually changed, thus, this remains a heuristic). However, although on average events might not be shifted far, single events might still be shifted by longer time distances. To avoid extreme shifts, which could be detected easily as outliers, we introduce a maximum allowed time distance ε that limits the maximum distance by which each event can be shifted.

This concept can be formulated as ILP, leading to our first algorithm for obfuscating event reorderings in Phase 2:

$$minimize \sum_{e_i \in E} \mid x(e_i) \mid \tag{4}$$

$$subject\ to$$

$$\forall\ or(e_j, e_k) \in R: \quad e_j.time + x(e_j) < e_k.time + x(e_k) \tag{C1}$$

$$\forall\ or(e_j, e_k) \in N: \quad e_k.time + x(e_k) < e_j.time + x(e_j) \tag{C2}$$

$$-\varepsilon(e_i) \leq x(e_i) \leq \varepsilon(e_i) \tag{C3}$$

$$where\ i = 1, 2, ..., n$$

This ILP is executed on a window of events E. The objective is the minimization of event shifts. Constraint 3 limits the maximum shift of each event by ε. Constraint 1 and 2 ensure the desired order of events as defined by Phase 1 while shifting events by a certain distance $x(e_i)$. Sets N and R contain pairs of events whose order has to be changed to conceal private patterns, and pairs of events whose order must be preserved as part of public patterns, respectively (output of Phase 1).

This ILP can be solved by a standard ILP solver. If a feasible solution cannot be found due to the constraints, Phase 1 and Phase 2 are repeated with the second best reordering w.r.t. the utility metric.

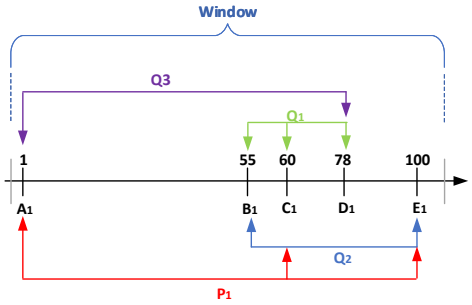


Figure 6: A window of Input event stream with arrival times

4.3 Graph-based Reordering

The second algorithm for assigning timestamps to reordered events and obfuscating reordering such that it cannot be detected is based on a graph representation. Again, this algorithm receives the sets N and R from Phase 1 containing pairs of events whose order is to be changed and preserved, respectively. The algorithm then first constructs a weighted directed acyclic graph (DAG) $G = (\mathcal{V}, \mathcal{E}, \mathcal{L}^{\mathcal{V}}, \mathcal{L}^{\mathcal{E}})$ from N and R. The graph vertices $\mathcal{V}$ represent events, and $\mathcal{E}$ denotes the order of events. Let v_{ei} and v_{ej} denote the vertices of the events e_i and e_j, respectively. Then a directed edge $(v_{ei}, v_{ej}) \in \mathcal{E}$ represents the event order (e_i, e_j). $\mathcal{L}^{\mathcal{V}}$ and $\mathcal{L}^{\mathcal{E}}$ represent vertex and edge labelings based on event timestamps and inter-arrival times, respectively.

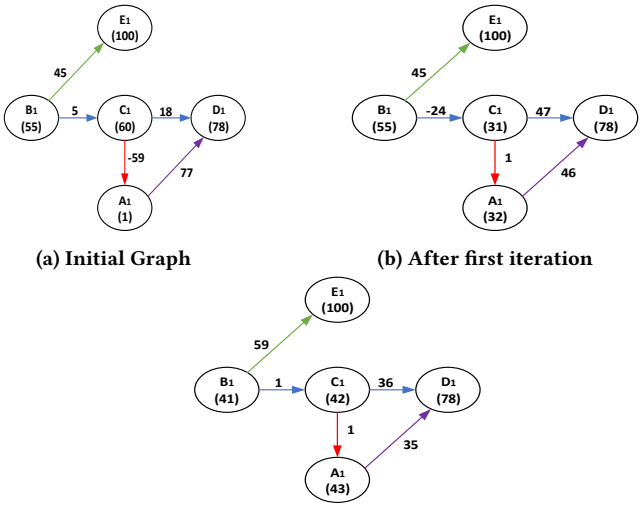
The initial graph is constructed as follows. For all ordered event pairs $(e_i, e_j) \in R$—i.e., events whose order is to be preserved—an edge (v_{ei}, v_{ej}) is added to $\mathcal{E}$. For all ordered event pairs $(e_i, e_j) \in N$—i.e., events whose order must be reversed—an edge (v_{ej}, v_{ei}) (reversed direction) is added to $\mathcal{E}$. Each vertex v_e is labeled with the original timestamp $t(e)$ of e, and each edge (v_{ei}, v_{ej}) is labeled with the original inter-arrival time $t(e_j) - t(e_i)$. Note that for event pairs in N with reversed order, the initial edge weights are negative since the timestamps have not been adjusted yet according to the reversed order.

Fig. 6 shows an example with the following three public patterns (Q_1, Q_2, Q_3) and one private pattern (P_1) and the timestamps shown in Fig. 6:

$$\begin{aligned} Q_1 &= SEQ(B, C, D) \qquad P_1 = SEQ(A, C, E) \\ Q_2 &= SEQ(B, E) \\ Q_3 &= SEQ(A, D) \end{aligned} \tag{5}$$

Assume that in Phase 1, sets N and R have been defined as follows: $R = \{(B_1, C_1), (C_1, D_1), (A_1, D_1), (B_1, E_1)\}$, $N = \{(A_1, C_1\})$. The initial graph for this example is shown in Fig. 7a.

The graph-based reordering algorithm shown in Alg. 1 takes the generated graph as input and adjusts vertex weights (event timestamps) and edge weights (inter-arrival times) until all inter-arrival times are positive, i.e., until timestamps are consistent with the event order defined in Phase 1. To increase the weight of a negative edge, we change both, the timestamp of the source and target vertex, equally by adding and subtracting half the edge weight representing the inter-arrival time between the two events, respectively. Equally changing the timestamps of both events ensures that the maximum



(a) Initial Graph **(b) After first iteration**

(c) Resulting Graph

Figure 7: Graph Transformation Algorithm

change of both timestamps is minimized to obfuscate performed event reorderings.

Algorithm 1 Graph Transformation

```
procedure GraphTransformation(g)
    while ContainsEdgeWithNegativeOrZeroWeight(g) do
        edge ← GetEdgeWithNegativeOrZeroWeight(g)
        sNode ← GetSourceNode(edge)
        tNode ← GetTargetNode(edge)
        ε − BoundCheck
        sNode.Time ← sNode.Time − ⌊|edge.weight| / 2⌋
        tNode.Time ← tNode.Time + ⌈|edge.weight| / 2⌉ + 1
        UpdateEdgeWeights(g)
    end while
end procedure
```

Also in the graph-based approach we never change timestamps by more than ε to obfuscate event reorderings. Before adjusting a negative edge by shifting the timestamps of the source vertex and target vertex, we check whether this would require a change greater than ε. If it is impossible to adjust timestamps without violating the ε bound, we stop Phase 2 and go back to Phase 1 to calculate another (second-best) order of events before repeating Phase 2.

Fig. 7b and Fig. 7c show how our initial example graph is transformed in multiple iterations. In the first iteration, the edge (C_1, A_1) with weight -59 is changed to weight 1 by shifting the timestamp of the source vertex C_1 from 60 to 31 and the timestamp of the target vertex A_1 from 1 to 32 as shown in Fig. 7b. However, the change of $t(C_1)$ introduces another negative edge between B_1 and C_1, which is adjusted in the following iterations to finally get a graph with all positive arrival times as shown in Fig. 7c.

4.4 Handling Overlapping Windows

Our two reordering approaches presented so far operate on windows. Private or public patterns are only detected and considered by the algorithms if the complete pattern is within the window. Executing these algorithms is straightforward if patterns are defined on *disjoint windows* such as tumbling windows. Note that similar to any CEP system, the user has to define patterns to be concealed or published together with a suitable windowing strategy. If patterns are defined on disjoint windows, reordering events in one window will not affect the next (disjoint) window.

However, if patterns are defined on *overlapping windows* such as sliding or hopping windows, events that are part of patterns might appear in different windows, and reordering in one window will affect all other overlapping windows [18]. Overlapping windows lead to a number of additional requirements, which are trivially fulfilled for disjoint windows but pose new challenges for overlapping windows:

- *Consistent reordering*: If events are reordered in one window, they must be reordered in the same order in other overlapping windows. Otherwise, inconsistent orders of the same events in different windows would give hints to the adversary that these events have been reordered.
- *Stable reordering*: Closely related to consistent reordering is the restriction that once a sequence of events has been forwarded, this order is stable, i.e., it cannot be reverted back and changed again by another window, which would lead to inconsistent event orders.
- *Non-redundant/exactly-once event forwarding*: If events are consistently reordered in all windows and the order of events is stable after forwarding, we forward each event exactly once.
- *Non-blocking forwarding of windows*: A window of events should be forwarded as soon as possible to support time-sensitive CEP applications whose QoS depends on the timely delivery of events. Although we do not strive for hard delay bounds, we would like to avoid blocking of windows by other windows, including (theoretically infinite) cascading blocking where window W_1 cannot be forwarded until the overlapping window W_2 is forwarded, which cannot be forwarded until the overlapping window W_3 is forwarded, etc..

Next, we explain how to meet these requirements with overlapping windows by using a greedy strategy that processes the window first which closes first. In general, multiple open windows $\{W_1, \ldots, W_m\}$ with overlapping events might exist at a time. Fig. 8 shows an example with two overlapping open windows W_1 and W_2. Whenever a window closes—e.g., after a certain time or reaching the maximum event count—the PAC executes Phase 1 and 2 on the closed window as described previously until a reordering is found. Let W_1 be the window that closes first and $W_1' = (e_1, \ldots, e_n)$ be the reordered window with the reordered event sequence $(e_1, \ldots, e_n)$ (in the example, $W_1' = (A_1, B_1, E_1, C_1, D_1, F_1, A_2, B_2)$). Moreover, let $p = (e_1, \ldots, e_k)$ with $\max k \leq e_n$ be the maximum prefix of events from W_1' that are not contained in any other open window from $\{W_2, \ldots, W_m\}$, i.e., $\forall W \in \{W_2, \ldots, W_m\} : p \cap W = \emptyset$ (in the example, $p = (A_1, B_1)$). Then we make p stable and forward p to the CEP middleware, i.e., the stable part is not blocked by other open windows. The remaining suffix $s = W_1' \setminus p$ of events from the closed window W_1' (in the example, $s = (E_1, D_1, C_1, F_1, A_2, B_2)$) contains events

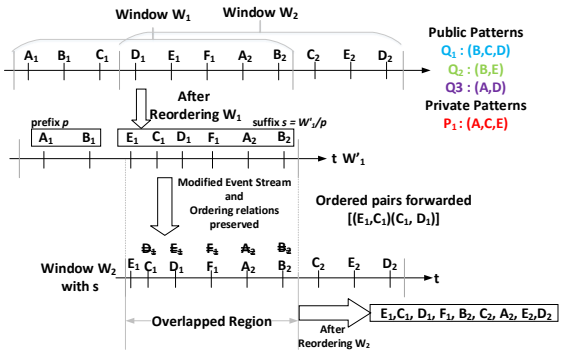


Figure 8: Solution for Overlapping Windows

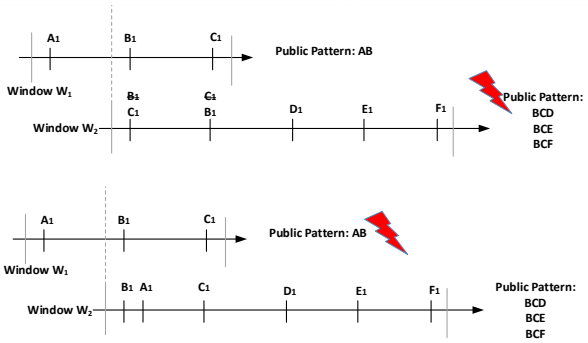


Figure 9: Reordering affecting the utility of the following window

that are already part of other open windows ($\{E_1, D_1, F_1, A_2, B_2\}$) and possibly also events that are not part of other open windows ($\{C_1\}$). In order not to drop events, we add all events from s that are not already part of the other open windows to these open windows. For instance, we add event C_1 to window W_2 in our example. As a last step, we also transfer ordering constraints that have been defined in Phase 1 to conceal private patterns in W_1 to the other open windows. This ensures that subsequent reorderings will not accidentally reveal already concealed private patterns from W_1' or contradict an already existing public pattern. In our example, we reordered the pair (C_1, E_1) from W_1 to (E_1, C_1) in W_1' and preserved the pair (C_1, D_1). Thus ordered constraints to preserve the pairs (E_1, C_1) and (C_1, D_1) are added to subsequent executions of Phase 1 while processing W_2.

Since windows are greedily reordered with *locally* optimal utility whenever a window closes, the constraints defined by already closed windows on the order of events in still open windows might prevent a *globally* optimal solution with maximum utility, which could be achieved if the whole history of events were considered for reordering. An example for such a sub-optimal reordering is shown in Fig. 9. The public patterns Q_i and the private pattern P_1 of this example are shown in Equ. 6.

$$\begin{aligned} Q_1 &= SEQ(B, C, D) \qquad P_1 = SEQ(A, B, C) \\ Q_2 &= SEQ(B, C, E) \\ Q_3 &= SEQ(B, C, F) \\ Q_4 &= SEQ(A, B) \end{aligned} \tag{6}$$

In W_1, the ordered pair (B_1, C_1) is reordered to conceal P_1. As a consequence, three public patterns Q_1, Q_2, and Q_3 are lost in W_2. If the ordered pair (A_1, B_1) were reordered instead to conceal P_1, then only Q_4 would be lost, i.e, providing greater global utility.

Thus, there is a trade-off between forwarding events with short delay and increasing global utility by collecting and processing a larger event history. Global utility can be increased by delaying the forwarding of events. This *forwarding delay* can be defined differently, e.g., as real time, number of events, or percentage of window size. In our evaluation we will show how the utility increases with increasing forwarding delay.

5 EVALUATION RESULTS

In this section, we evaluate our event reordering approach with respect to its ability to conceal private patterns (preserving privacy) with minimum impact onto public patterns (preserving QoS). Moreover, we evaluate the performance of our reordering algorithms with respect to latency.

5.1 Evaluation Setup

For the performance evaluation, we ran our experiments on a commodity server with Intel Core i5-5300U processor (2 cores / 4 threads at 2.3 GHz) and 12 GB of RAM. We implemented our reordering algorithms in Python using the GNU Linear Programming Kit (GLPK) as ILP solver in Phase 1 and for the ILP-based approach also in Phase 2.

We used both, a real-world data set and—where the real-world dataset could not be used—a synthetic dataset. As *real-world dataset (Dataset 1)* we used the publicly available Online Retail dataset from the UC Irvine Machine Learning Repository [4]. This dataset contains all the transactions occurring between 01/12/2010 and 09/12/2011 (around 1 year) of a UK-based online retailer mainly selling all-occasion gifts. It contains almost 500,000 purchased items along with timestamps and customer ids with around 3,200 different items spread over 20,000 transactions. To define patterns, we search this dataset for all event sequences of length 2 to 4 among the transactions (e.g., SEQ(Wooden Happy Birthday Garland, Wooden cake stand, Party Invites Woodland)), and selected the 100 most popular sequence patterns as private and public patterns of varying length. We searched for patterns in windows of size 4–800 events. For the evaluation of overlapping windows, we used hopping windows with hopping intervals of 5 events. Note that such an e-commerce dataset might seem less privacy-sensitive as, for instance, a health dataset—which is hard to find and use for evaluations due to its obvious criticality. However, by analyzing the shopping behavior of customers, privacy-sensitive information can be revealed indeed, for instance, the pregnancy of a customer [12].

For the *synthetic dataset*, we generated a sequence of 100,000 events with event instances generated from a set of 20 different event types. The arrival times of each event-type are normally distributed with mean and standard deviation of μ and σ time units.

The values of μ and σ are defined as variables and can be changed for each run. We selected 50 random public and private patterns with varying lengths from 2 to 10. Each pattern contains event types without repetition from the above mentioned set.

5.2 Adversary Model

For the evaluation of the effectiveness of protecting privacy, we first need to elaborate on the adversary model. In our system model presented in Sec. 3, we have made the assumption that the adversary has background information about causal constraints on event orders and statistical information about the true (typical) inter-arrival time distribution of events to perform causal order constraint attacks and statistical attacks, respectively. Since our approach by design ensures that causal order constraints are never violated by adding suitable constraints to the ILP in Phase 1, we focus here on the ability to resist statistical attacks.

For statistical attacks, the adversary observes the reordered event stream and calculates the probability that a certain private pattern has been concealed by reordering. If the probability is higher than a pre-defined threshold—i.e., the adversary has sufficient "confidence" that he correctly revealed a private pattern—, the adversary assumes that the pattern has occurred. So the basic question is, how can the adversary calculate these probabilities?

To this end, we assume that from publicly available data the adversary knows the true mean value $\mu(A)$ and true standard deviation $\sigma(A)$ of the interarrival time distribution for events of each event type A, B, C, etc. Moreover, for each ordered pair of event types (A, B), we assume that the adversary knows the true mean value $\mu((A, B))$ and standard deviation $\sigma((A, B))$ of the interarrival time distribution of pairs of events of these types. From observing the reordered event stream, the adversary knows the observed interarrival time between each pair of events, say (A_i, B_j). With this information, the adversary can calculate the so-called *z-score* $z(A_i, B_j) = \frac{t(B_j) - t(A_i) - \mu((A,B))}{\sigma((A,B))}$, i.e., the number of standard deviations by which the observed inter-arrival time is below or above the true mean inter-arrival time [14]. Similarly, the individual z-scores $z((A_{i-1}, A_i)) = \frac{t(A_i) - t(A_{i-1}) - \mu(A)}{\sigma(A)}$ can be calculated for the individual events A_i and B_j. Intuitively, these z-scores quantify how much the reordered stream deviates from the expectation of the adversary defined by his background knowledge ("ground truth" of the adversary).

When the adversary observes a potentially reordered sequence, say $(A_i, B_j, \ldots, Z_k)$, he calculates the *reorder indicator (RI)* for each private pattern individually. $\mathrm{RI}((A_i, B_j, C_k))$ for a private pattern $(SEQ(A, B, C))$ is calculated as the combination of the probability values (p_z) corresponding to the z-scores of the individual events as well as of the event pairs. Table 1 is called a z-table which shows p_z values corresponding to z-scores for a normal distribution. The number of different z-scores can be varied depending on the required resolution. It is sufficient to calculate *RI* for ordered pairs of private patterns individually and then combine them since reordering is also done at the level of ordered pairs. Thus *RI* for an ordered pair (A, B) is calculated as:

$$\mathrm{RI}(A_i, B_j) = 1 - (p_z(A_{i-1}, A_i) * p_z(B_{j-1}, B_j) * p_z(A_i, B_j))$$

Table 1: Z-table with probability values under a defined z-score

Z-score	Probability values (p_z)
less than 1	68.27%
between 1 and 2	27.18%
between 2 and 3	4.28%
greater than 3	≈ 0%

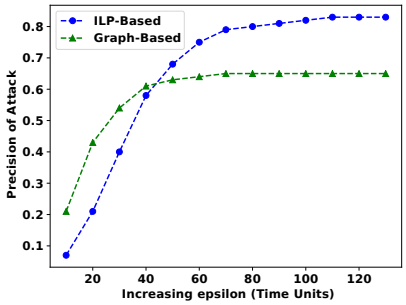
If RI is above a pre-defined threshold, the adversary assumes that the stream has been reordered. Note that RI is just a hint to the adversary that something has been reordered, but does not directly tell which pair(s) of events have been reordered. Thus, even an RI of 100 % will not suffice to provably reveal the true order, as, for instance, might be required in legal cases (in dubio pro reo). Still, high RI scores might have negative consequences for the producer. For instance, a customer could be denied an insurance contract or loan due to a negative risk assessment. Therefore, we even consider such hints of reordering critical and explicitly consider the RI in our evaluation.

5.3 Robustness to Statistical Attacks

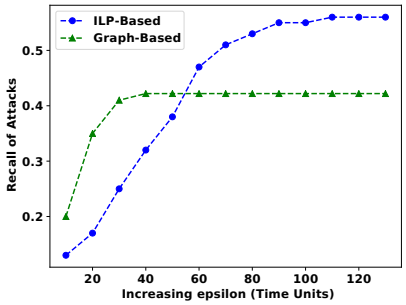
We start our evaluation by evaluating the robustness to statistical attacks of our reordering approaches. As already pointed out, private patterns will never occur in the reordered event stream (unless the utility metric deliberately defines a trade-off between privacy and QoS). So the major viable attacks are statistical attacks, where the adversary tries to calculate the probability that a private pattern has been concealed through reordering. In the adversary model in Sec. 5.2, we have already described how the adversary calculates the Reordering Indicator (RI) indicating that a private pattern has been reordered. If the RI is above a pre-defined threshold $\mathrm{th}_{\mathrm{RI}}$, the adversary assumes that a private pattern has been concealed. We will now evaluate the effectiveness of this attack by evaluating the *precision* (correctly identified private patterns divided by the overall number of correctly or incorrectly identified private patterns) and *recall* (correctly identified private patterns divided by total number of concealed private patterns in the reordered stream) of the attack. We set $\mathrm{th}_{\mathrm{RI}} = 0.9$, i.e., 90 % "confidence" that a private pattern has been concealed. We use the synthetic dataset with precisely known mean values μ and standard deviations σ, and assume that the adversary precisely knows the accurate values of μ and σ from background knowledge to provide perfect attack conditions.

One important parameter of our algorithms is ε, which limits the maximum time distance by which each event can be shifted (cf. Sec. 4.2). Small values of ε should lead to only minor changes of inter-arrival times providing little evidence to the adversary about reordering, however, too small values of ε will make it impossible for our algorithms to find a reordering. Fig. 10a and Fig. 10b show the influence of ε onto precision and recall. We see that smaller values of ε indeed decrease both, precision and recall of attacks, thus, making statistical attacks less effective.

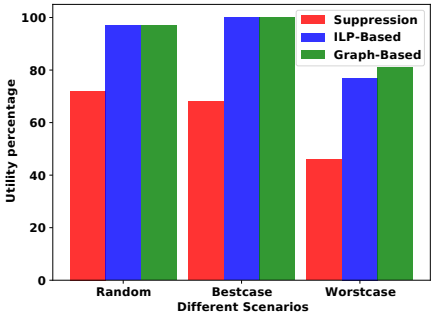
We also see that the graph-based approach provides better privacy, i.e., smaller recall and precision for the same values of ε, than the ILP-based approach. For each event pair to be reordered, the



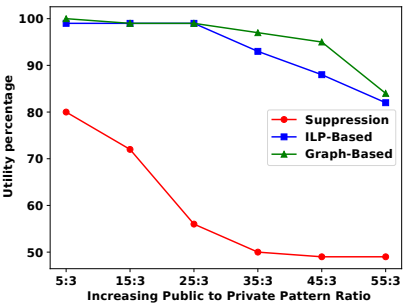
(a) Precision vs Epsilon Range



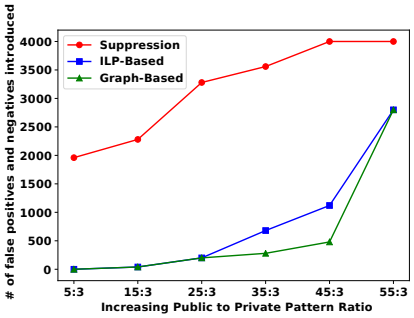
(b) Recall vs Epsilon Range



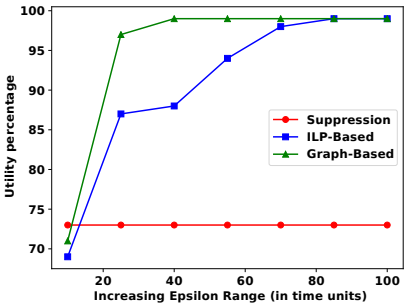
(c) Comparison of Suppression and Reordering



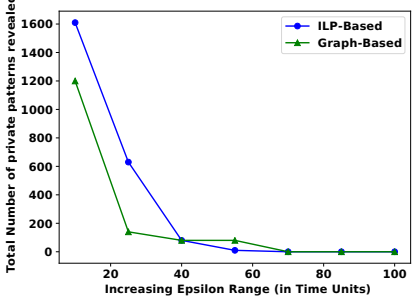
(d) Utility with # of Public Patterns



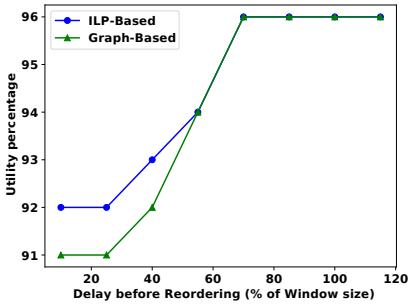
(e) False positives and false negatives vs # of Public Patterns



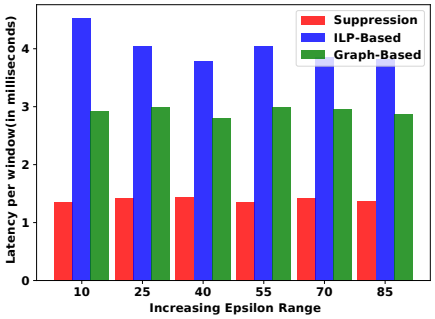
(f) Utility with increasing Epsilon



(g) # of Private Patterns revealed vs Epsilon



(h) Utility with increasing forwarding delay



(i) Latency comparison

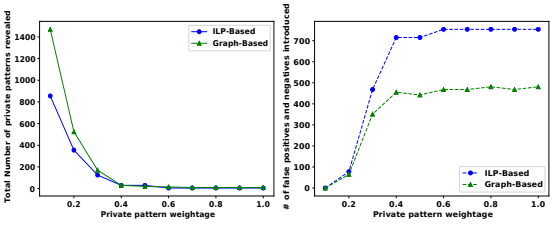
Figure 10: Evaluations Results

graph-based approach moves timestamps of both events equally. This balanced shifting of events minimizes the maximum time distance of individual events. In contrast, the ILP-based approach is not balancing shifts and might have to move events longer distances. Therefore, the graph-based approach in general impacts the inter-arrival time distributions less than the ILP-based approach making reordering harder to detect.

5.4 QoS Preservation

Next, we evaluate the negative impact of concealing private patterns onto QoS. Here, we mutually compare our ILP- and graph-based reordering approaches, and compare the reordering strategy to its closest competitor from related work, namely, the event supression strategy as proposed in [26]. All approaches conceal private patterns at the expense of potentially destroying some desired public patterns, i.e., by negatively impacting QoS.

For evaluating the privacy-QoS trade-off, we use a setting where the approaches do not reveal any private patterns, i.e., privacy takes strict precedence over QoS. As performance metric, we use the utility metric defined in Equ. 2. We configure the utility metric such that a) no private patterns are revealed (strict privacy precedence of concealed private patterns); b) all public patterns are assigned



(a) # of private patterns revealed vs Criticality **(b) # of false-positives and false negatives vs Criticality**

Figure 11: Evaluation for criticality of private patterns

uniformly random weights between 1 and 10 to express the impact of different types of public patterns onto QoS; c) consequently false positive and false negative public patterns get same weights in negative, corresponding to the public patterns. For a fair comparison, we consider three different scenarios: the best-case, worst-case, and average-case scenario from the point of view of the event reordering strategy. In the best case, public and private patterns do not share any ordered pairs of events. Note that this does not mean that private and public patterns do not share any *individual* events! In the worst-case scenario, public patterns are chosen such that all the ordered pairs of private patterns do have a match in the set of all ordered pairs of public patterns. In the average-case scenario, 50% ordered pairs of public patterns match with ordered pairs of private patterns. For this evaluation, we used the real-world dataset. Fig. 10c show the results for all scenarios. It is evident from the figure that event reordering is in all scenarios and for both reordering approaches superior to event suppression with respect to QoS, even in the worst case scenario. This is because of the obvious reason that suppression strategy introduces a greater number of false negatives when compared to the reordering approaches.

Since it becomes harder to preserve privacy with an increasing number of public patterns, because required changes in private patterns have a higher chance of also hitting events of public patterns, we furthermore show the impact on the reordering approaches with respect to the number of public patterns. Fig. 10d show the results for increasing number of public patterns with the number of private patterns maintained constant. We see that the number of public patterns indeed influences the QoS. We also see that the graph-based approach impacts QoS slightly less than the ILP-based approach. This is also because of the balanced shifting of events in the graph-based approach which leads to lesser impact on public patterns. Furthermore we show the impact of number of public patterns on the number of false positives and false negatives introduced for the same configuration in Fig. 10e.

It is also important to evaluate the impact of parameter ε onto the utility. Since ε limits the time distance by which events can be shifted to increase the robustness to statistical attacks as shown in Sec. 5.3, at the same time it might make it impossible to find a reordering within the given bounds. Thus, we expect the utility to be lower for smaller values of ε. Fig. 10f shows the change in utility achieved for varying ε, also in comparison to the suppression strategy. If ε is very low such that reordering becomes impossible, then the utility of the reordering strategy is less than the utility of the suppression strategy. However, note that the suppression strategy does not consider additional constraints such as the ε restriction, making it vulnerable to statistical attacks. In particular, suppressing event also impacts the inter-arrival time distribution of events, again giving the adversary hints about changes of the event stream.

So far, we only evaluated scenarios where privacy took strict precedence over QoS, i.e., with 0 % revealed private patterns. Next, we evaluate this privacy/QoS trade-off with our reordering approach. To define this trade-off, we introduced the criticality percentage cp_k in Equ. 3. Fig. 11a shows the number of revealed private patterns over cp (privacy). Fig. 11b shows the number of false positives and negatives over cp_k (QoS). As we can see, greater values of cp_k increase privacy and at the same time decrease QoS.

5.5 Delay-QoS Trade-off

In Sec. 4.4, we have stated the requirement to forward events from the PAC as soon as possible, i.e., to minimize the delay introduced by processing (reordering) events in windows. We also have already seen that overlapping windows are critical with respect to delay since the reordering performed in one window has to be considered by other overlapping windows. In general, we expect that by delaying events longer, we might be able to increase QoS (utility) by making optimal reordering decisions, i.e., there is a trade-off between delay and QoS. Next, we evaluate this trade-off. For these experiments, we again use the real-world dataset.

As discussed in Sec. 4.4 a short delay in forwarding events could provide a higher utility globally. Fig. 10h shows the utility over varying *forwarding delay*. We see the expected trend of increasing utility for longer delay. If a certain delay is reached, the utility does not increase any further by delaying events longer since the chosen delay is sufficient to cover the maximum overlap of windows—i.e., while processing one window, the pattern matches of the next overlapping window are already known.

5.6 Runtime Efficiency

In this subsection, we evaluate the runtime efficiency of our ILP- and graph-based reordering approaches. Computational overhead is introduced by finding an optimal reordering during Phase 1 that maximizes utility (same Phase 1 implementation for ILP-based and graph-based approach), and assigning timestamps such that reordering cannot be identified in Phase 2 (implemented differently for ILP-based and graph-based approach). Here, we want to quantify this computational overhead for both reordering approaches and also compare it to the overhead of the related suppression approach.

In this evaluation, we use the real-world dataset. With each approach, we processed 5000 windows and calculated the average processing latency for processing (reordering/suppression) one window. Fig. 10i shows the results. First of all, we see that suppression is the most efficient approach, followed by the graph-based reordering approach, and then the ILP-based approach. Suppression requires only about 40-50 % of the runtime of the graph-based reordering strategy on average. This gap can be explained by the additional overhead for maximizing utility—in particular, preserving more public patterns as shown in the QoS evaluation in Sec. 5.4—and

obfuscating changes made to the event stream to protect them from statistical attacks.

With respect to the ε constraint, the suppression strategy is obviously independent of this value since it does not know this constraint or a similar strategy to protect changes from being identified by statistical attacks. For the ILP-based reordering approaches, in Fig. 10i we can observe the expected influence of ε on runtime, i.e., longer runtime for smaller ε value. However, the influence is not very much pronounced (max. 15% higher runtime), and the runtime stabilizes quickly for greater ε values. For the graph-based approach, the influence of ε on the runtime efficiency is negligible.

6 SUMMARY AND FUTURE WORK

In this paper, we presented a novel event reordering approach for removing privacy-sensitive information, specified as sequences of events, from CEP event streams. Besides protecting privacy by concealing private event patterns through event reordering, our approach also preserves quality of services by preserving, as much as possible, desired public patterns, which should not be concealed to provide required input to CEP services. We defined a utility metric that allows for specifying the desired privacy/QoS trade-off. Moreover, we presented two reordering algorithms based on Integer Linear Programming and graph processing to find event reorderings maximizing utility while also protecting from sophisticated statistical attacks taking into account advanced background knowledge of event inter-arrival time distributions. Finally, we showed that the event reordering strategy is superior to the closest related strategy using event suppression with respect to QoS. Moreover, we showed that our event reordering approach is robust to advanced statistical attacks and can be implemented efficiently to allow for online processing of event streams.

As part of future work, further concepts for concealing private patterns could be considered. For instance, while reordering is a natural choice to conceal privacy-sensitive patterns defined as event sequences, it does not protect from revealing private information depending on values, e.g., a heart rate constantly greater than a given threshold value for a certain amount of time. To strike a balance between privacy and QoS in such scenarios would require to filter and modify (obfuscate) events and data based on values (here heart rate, duration) such that modifications cannot be recognized.

ACKNOWLEDGMENTS

This Paper is part of the PATRON research project. The PATRON research project is commissioned by the Baden-Württemberg Stiftung GmbH. The authors would like to thank the BW-Stiftung for the funding of PATRON.

REFERENCES

[1] Osman Abul, Francesco Bonchi, and Fosca Giannotti. 2010. Hiding Sequential and Spatiotemporal Patterns. *IEEE Transactions on Knowledge and Data Engineering* 22, 12 (dec 2010), 1709–1723.

[2] Raman Adaikkalavan, Indrakshi Ray, and Xing Xie. 2011. Multilevel Secure Data Stream Processing. In *Data and Applications Security and Privacy XXV*, Yingjiu Li (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 122–137.

[3] Jianneng Cao, Barbara Carminati, Elena Ferrari, and Kian-Lee Tan. 2009. ACStream: Enforcing Access Control over Data Streams. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE.

[4] Daqing Chen, Sai Laing Sain, and Kun Guo. 2012. Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining. *Journal of Database Marketing & Customer Strategy Management* 19, 3 (aug 2012), 197–208.

[5] Yan Chen, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2017. PeGaSus: Data-Adaptive Differentially Private Stream Processing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 1375–1388.

[6] Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information. *Comput. Surveys* 44, 3 (jun 2012), 1–62.

[7] Gianpaolo Cugola and Alessandro Margara. 2013. Deployment strategies for distributed complex event processing. *Computing* 95, 2 (01 Feb 2013), 129–156.

[8] Christian Esposito and Mario Ciampi. 2015. On Security in Publish/Subscribe Services: A Survey. *IEEE Communications Surveys & Tutorials* 17, 2 (2015), 966–997.

[9] Chet Geschickter and Kristin R. Moyer. 2016. *Measuring the Strategic Value of the Internet of Things for Industries*. Technical Report TR ID : G00298896. Gartner Inc.

[10] Aris Gkoulalas-Divanis and Grigorios Loukides. 2011. Revisiting sequential pattern hiding to enhance utility. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11*. ACM Press.

[11] Yeye He, Siddharth Barman, Di Wang, and Jeffrey F. Naughton. 2011. On the complexity of privacy-preserving complex event processing. In *Proceedings of the 30th symposium on Principles of database systems of data - PODS '11*. ACM Press.

[12] Kashmir Hill. 2012. How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did. (Feb. 2012). Retrieved 2018-02-26 from https://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/#2099483e6668

[13] Georgios Kellaris, Stavros Papadopoulos, Xiaokui Xiao, and Dimitris Papadias. 2014. Differentially private event sequences over infinite streams. *Proceedings of the VLDB Endowment* 7, 12 (aug 2014), 1155–1166.

[14] Erwin Kreyzig. 1979. *Advanced Engineering Mathematics* (fourth ed.). Wiley Publications. 880 pages.

[15] Kubilay Ahmet Küçük, Andrew Paverd, Andrew Martin, N. Asokan, Andrew Simpson, and Robin Ankele. 2016. Exploring the Use of Intel SGX for Secure Many-Party Applications. In *Proceedings of the 1st Workshop on System Software for Trusted Execution (SysTEX '16)*. ACM, New York, NY, USA, Article 5, 6 pages.

[16] Wolfgang Lindner and Jorg Meier. 2006. Securing the Borealis Data Stream Engine. In *2006 10th International Database Engineering and Applications Symposium (IDEAS'06)*. IEEE.

[17] David Luckham. 2008. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. In *Rule Representation, Interchange and Reasoning on the Web*, Nick Bassiliades, Guido Governatori, and Adrian Paschke (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 3–3.

[18] Ruben Mayer, Ahmad Slo, Muhammad Adnan Tariq, Kurt Rothermel, Manuel Gräber, and Umakishore Ramachandran. 2017. SPECTRE. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference on - Middleware '17*. ACM Press.

[19] Matteo Migliavacca, Ioannis Papagiannis, David M. Eyers, Brian Shand, Jean Bacon, and Peter Pietzuch. 2010. DEFCON: High-performance Event Processing with Information Security. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference (USENIXATC'10)*. USENIX Association, Berkeley, CA, USA, 1–1.

[20] Do Le Quoc, Martin Beck, Pramod Bhatotia, Ruichuan Chen, Christof Fetzer, and Thorsten Strufe. 2017. Privacy Preserving Stream Analytics: The Marriage of Randomized Response and Approximate Computing. *CoRR* abs/1701.05403 (2017). arXiv:1701.05403

[21] Zohaib Riaz, Frank Dürr, and Kurt Rothermel. 2017. Understanding Vulnerabilities of Location Privacy Mechanisms against Mobility Prediction Attacks. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. ACM. https://doi.org/10.4108/eai.7-11-2017.2273757

[22] B. Schilling, B. Koldehofe, K. Rotherme, and U. Ramachandran. 2013. Access Policy Consolidation for Event Processing Systems. In *2013 Conference on Networked Systems*. IEEE.

[23] Avesh Singh. 2017. Applying Artificial Intelligence in Medicine. (May 2017). Retrieved 2018-02-26 from https://blog.cardiogr.am/applying-artificial-intelligence-in-medicine-our-early-results-78bfe7605d32

[24] Andre Soldwedel and Jutta Rothmund. 2015. *Quantified Health*. Technical Report. YouGov Deutschland AG.

[25] Christoph Stach, Frank Dürr, Kai Mindermann, Saravana Murthy Palanisamy, and Stefan Wagner. 2018. How a Pattern-based Privacy System Contributes to Improve Context Recognition. In *Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (CoMoRea)*. IEEE Computer Society, 1–6.

[26] Di Wang, Yeye He, Elke Rundensteiner, and Jeffrey F. Naughton. 2013. Utility-maximizing event stream suppression. In *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*. ACM Press.