UniTraX: Protecting Data Privacy with Discoverable Biases

Reinhard Munz[1,✉], Fabienne Eigner[2], Matteo Maffei[3], Paul Francis[1], and Deepak Garg[1]

[1] MPI-SWS, Kaiserslautern and Saarbrücken, Germany
{munz,francis,dg}@mpi-sws.org
[2] CISPA, Saarland University, Saarbrücken, Germany
eigner@cs.uni-saarland.de
[3] TU Wien, Wien, Austria
matteo.maffei@tuwien.ac.at

Abstract. An ongoing challenge with differentially private database systems is that of maximizing system utility while staying within a certain privacy budget. One approach is to maintain per-user budgets instead of a single global budget, and to silently drop users whose budget is depleted. This, however, can lead to very misleading analyses because the system cannot provide the analyst any information about which users have been dropped.
This paper presents UniTraX, the first differentially private system that allows per-user budgets while providing the analyst information about the budget state. The key insight behind UniTraX is that it tracks budget not only for actual records in the system, but at all points in the domain of the database, including points that could exist but do not. UniTraX can safely report the budget state because the analyst does not know if the state refers to actual records or not. We prove that UniTraX is differentially private. UniTraX is compatible with existing differentially private analyses and our implementation on top of PINQ shows only moderate runtime overheads on a realistic workload.

1 Introduction

Differential Privacy (DP) is a model of anonymity that measures privacy loss resulting from queries made to a database [6]. A bound on privacy loss can be enforced by preventing queries after a privacy budget has been exceeded. An ongoing challenge with DP systems is that of maximizing system utility while staying within a privacy budget, where system utility is measured in terms of both number of queries and amount of distortion (noise) in query answers.

A simple but common approach to DP budgets is to maintain a single global budget. With this approach, all queries draw from the budget regardless of how many user records are used to answer a given query. In systems where users can specify their own individual budgets, the global budget is effectively the *minimum* of user budgets.

An alternative approach is to maintain per-user budgets. The idea here is that a given query draws only from the budgets of users whose records contribute to the answer. This can substantially improve system utility. An analysis that for instance targets smokers in a medical dataset would not reduce the budgets of non-smokers. Furthermore, per-user budgets maximize utility in systems where users specify their individual budgets because low-budget users do not constrain the queries that are made over only high-budget users.

In spite of the tremendous potential for increasing the utility of DP systems, we are aware of only a single system, ProPer [10], that tracks per-user budgets.[4] This is because of a fundamental difficulty with per-user budget systems. Namely, the system cannot report on the remaining budget of individual users without revealing private information. If budgets were made *public* in this way, then an analyst could trivially obtain information about users just by observing which users' budgets changed in response to a query.

Because of this, ProPer keeps user budgets *private*: it silently drops the record of a user from the dataset when the user's budget is depleted. This creates a serious usability problem for the analyst. Suppose there are two analysts, Alice and Bob. Alice wishes to learn about smokers, Bob wishes to learn about lung-cancer patients. Suppose Alice makes a set of queries about smokers, and as a result many smokers' budgets are depleted and these smokers' records are dropped from the dataset. Afterwards Bob asks the question: "What fraction of lung cancer patients are smokers?". Because many smokers have been dropped from the dataset, and non-smokers have not, Bob's answer is incorrect. Worse, Bob has no way of knowing whether the answer is incorrect, or how incorrect it is. Bob's answer is effectively useless. We call this *unknown dataset bias*.

To address this problem, this paper presents UniTraX, a DP system that allows for the benefits of keeping per-user budgets without the disadvantage of unknown dataset bias. The key insight of UniTraX is in how it tracks budget. Rather than *privately* tracking individual users' remaining budget, UniTraX *publicly* tracks the budget consumed by prior queries over regions of the data parameter space. In addition, UniTraX adds each user's initial budget to the dataset, making it a queryable parameter.

For example, assume a query asks for the count of users between the ages of 10 and 20. ProPer would *privately* deduct the appropriate amount from the individual remaining budget of *all users* in that age range. By contrast, UniTraX *publicly* records that a certain amount of budget was consumed for the *age range 10-20*. Because the consumed budget is public, the analyst can calculate how much initial budget any given point in the data parameter space would need in order to still have enough remaining budget for some specific query the analyst may wish to make. Because initial budgets are also a queryable parameter, the analyst can then explicitly *exclude* from the query any points whose initial budget is too small. This allows the analyst to control which points

[4] Other DP systems also permit per-user or per-field *initial* budgets [1, 15]. However, these systems do not track the *consumption* of budget on a per-user basis.

		budget attribution	
		global	*per-user*
consumed budget visibility	*private*		ProPer
	public	DP reference	**UniTraX**

Fig. 1. System comparison

are included in answers and therefore avoid unknown dataset bias. (See Sect. 2 for a detailed example.)

Internally, UniTraX utilizes the same calculation of required initial budget to reject any query that covers points without sufficient budget. Critically, such a rejection does not leak any private information as it solely depends on public budget consumption data and query parameters. In fact, the decision to reject a query does not even look at the actual data.

A significant practical concern is that tracking budgets across the entire parameter space, which will usually be substantially larger than the number of actual records in the database, can be quite expensive. To understand this cost, we built a prototype implementation of UniTraX on top of PINQ [17]. By carefully clubbing budgets over contiguous regions of the parameter space, we obtain average overheads of less than 80 % over a no-privacy baseline on a realistic workload.

The contributions of this paper are threefold:

1. A system model and design that maintains the advantages of per-user privacy budgets, while avoiding the problems due to unknown dataset bias.
2. A theoretical framework and proof that the design provides DP.
3. An implementation and evaluation showing that the system is able to efficiently track budgets with average overheads of less than 80 %.

In Sect. 2 we compare different system models for DP and provide an example to illustrate the effect of unknown dataset bias. We introduce the design of UniTraX in Sect. 3 and detail the theoretical framework and the proof of DP in Sect. 4. Our implementation and its evaluation are presented in Sects. 5 and 6. We discuss related work in Sect. 7 and conclude in Sect. 8.

2 System Comparison

To better understand the differences and advantages of UniTraX, we start with overviews of UniTraX and two prior system models, the classic DP "reference" model with a global budget, and ProPer with private per-user budgets. We use a simple running example to illustrate the differences. Figure 1 contrasts the public, per-user budget model of UniTraX with DP reference and ProPer.

For the example we assume that two analysts Alice and Bob want to analyze a dataset of patient records. These records contain a variety of fields among which is one that indicates whether a patient is a smoker, and one that indicates

whether the patient suffers from lung cancer. We assume that Alice is interested in smokers and wants to run various queries over different fields of smokers while Bob is interested in the fraction of lung cancer patients that are smokers. We assume that Alice does her analysis first, followed by Bob.

Regarding the setting of each patient's (user's) initial budget, we consider two cases: (1) all initial budgets are the same (uniform initial budgets), and (2) each budget is set by the user (non-uniform initial budgets). In the case of UniTraX, the initial budget is just another field in each record.

DP Reference. The DP reference mechanism uses a publicly visible global budget. In the case of uniform initial budgets, the global budget is set as the system default. In the case of non-uniform initial budgets, the global budget is set to the lowest initial budget among all users.

The reference mechanism counts every query against this single global budget. First, Alice runs her queries against smokers. Since each query decrements from the global budget, this budget may well be depleted before Bob can even start. At this point no information about non-smokers will have left the system. Still, the system has to reject all further queries.

ProPer. ProPer tracks one budget per user but must keep it private. Users whose budgets are depleted are silently dropped from the dataset and not considered for any further queries. Nevertheless, each user's full budget can be used.

Staying in our example, Alice's queries use no budget of non-smokers under this tracking mechanism. Once Alice has finished her queries, Bob starts his analysis. Bob wishes to make two queries, one counting the number of smokers with lung cancer, and one counting the number of non-smokers with lung cancer. Bob may look at Alice's queries, and observe that she focused on smokers, and therefore know that there is a danger that his answers will be biased against smokers. In the general case, however, he cannot be sure if his answers are biased or not.

In the case of uniform budgets, if Alice requested histograms, then she would have consumed the smokers' budgets uniformly and depleted either all or none of the smokers' budgets. If Bob gets an answer that, keeping in mind the noise, is significantly larger than zero, then Bob's confidence that his answer is non-biased may be high. If on the other hand Alice focused some of her queries on specific ranges (e.g., certain age groups), or if budgets are non-uniform, then Bob knows that the answer for smokers with lung cancer may be missing users, while the answer for non-smokers with lung cancer will not. He may therefore have unknown dataset bias, and cannot confidently carry out his analysis.

Our System (UniTraX). UniTraX tracks public budgets that are computable from the history of previous queries. UniTraX is able to tell an analyst how much budget has been consumed by previous queries for any subspace of the parameter space. For example, the analyst may request how much budget has been consumed in the subspace defined by "age$\geq$10 AND age$<$20 AND gender=male AND smoker=1".

UniTraX tracks budget consumption over regions of the parameter space. For example, if a query selects records over the subspace "age≥10 AND age<20", then UniTraX records (publicly) that a certain amount of budget has been consumed from this region of the parameter space. Initial budgets are an additional dimension of the parameter space in UniTraX. In particular, the initial budget of an actual record in the database is stored in a field in the record. By comparing the (public) consumed budget of any point in the parameter space to the initial budget of that point, UniTraX can determine publicly whether that point's budget has been fully consumed or not. This allows UniTraX to reject a query safely: If, after the query, the consumed budget of any point selected by the query will exceed that point's initial budget, then the query is immediately rejected. This decision does not require looking at the actual data, and reveals no private information.

Critically, public consumed budgets combined with the ability to filter queries based on users' initial budgets allows analysts to control and eliminate unknown dataset bias. Returning to our example, when Bob is ready to start his analysis, he queries UniTraX to determine the consumed budgets for "smoker=1 AND disease=lungCancer", and "smoker=0 AND disease=lungCancer". Because no queries have been made for non-smokers, the consumed budget of the latter query's region would be zero. Suppose that UniTraX indicates that the consumed budget for the region "smoker=1 AND disease=lungCancer" is 50, and that Bob's two queries will further consume a budget of 10 each. Because the two groups are disjoint, Bob knows that any user with an initial budget of 60 or higher has enough remaining budget for his queries. (If the two queries were not known to have disjoint user populations, then Bob would need to filter for initial budgets of 70 or higher.)

Bob generates the following two queries:

- "count WHERE smoker=1 AND disease=lungCancer AND initBudget≥60",
- "count WHERE smoker=0 AND disease=lungCancer AND initBudget≥60".

In doing so, Bob is assured that that no users are excluded from either query, and avoids unknown dataset bias.[5]

So far, we have described how Bob may query only points with sufficient remaining budget. However, when this is not the case, UniTraX is able to simply reject Bob's queries. In fact, UniTraX can even inform him about which points are out of budget without leaking private information. Privacy is protected by the fact that Bob does not know whether these points exist in the dataset or not. UniTraX's rejection does not reveal this information to Bob as it solely depends on public consumed budgets and query parameters. Using the returned information, Bob is able to debug his analysis and retry.

[5] Note that if users select their own initial budgets, and there is some correlation between user attributes and initial budgets, then there may still be a specific bias in the data. For instance if smokers tend to choose high budgets and non-smokers tend to choose low budgets, then Bob's queries would be biased towards smokers. This problem appears fundamental to any system that allows individual user budgets.

UniTraX not only allows analysts to debug their analyses but is fully compatible with existing DP systems. Any analysis that successfully executes over a dataset protected by a global budget system requires only a simple initialization to run on the same dataset protected by UniTraX (see Sect. 5 for PINQ-based analyses). Thus, analysts can easily adapt to UniTraX and exploit the increased utility of per-user budgets.

3 Design Overview

Threat Model. UniTraX uses the standard threat model for DP. The goal is to prevent malicious analysts from discovering whether a specific record (user) exists in the queried database (dataset). We assume, as usual, that analysts are limited to the interface offered by UniTraX and that they do not have direct access to the database. We make no assumptions about the background or auxiliary knowledge of the analysts. Analysts may collude with each other offline.

Goals. We designed UniTraX with the following goals in mind.

Privacy: Users should be able to set privacy preferences (budgets) for their records individually. These preferences must be respected across queries.

Utility: Querying a parameter subspace should not affect the usability of records in a disjoint subspace.

Bias discovery: The system should allow the analyst to discover when there may be a bias in query answers because privacy budgets of some parts of the parameter space have been depleted by past queries.

Efficiency: The overhead of the system should be moderate.

In the following we describe the design of UniTraX, explaining how it attains the first three goals above. The fourth goal, efficiency, is justified by the experimental evaluation in Sect. 6.

Design Overview. For simplicity, we assume that the entire database is organized as a single table with a fixed schema. The schema includes a designated column for the initial privacy budget of each record. UniTraX is agnostic to how this initial budget is chosen—it may be a default value common to all records or it may be determined individually for each record by the person who owns the record. Higher values of initial budget indicate less privacy concerns for that record. Records may be added to the database or removed from it at any time.

The set of all possible records constitutes the *parameter space.*[6] We use the term *point* for any point in the parameter space; a point may or may not exist in the actual database under consideration. We use the terms *actual record* and *record* for the points that actually exist in the database under consideration.

Like most DP systems, UniTraX supports statistical or aggregate queries. The query model is similar to that of PINQ [17]. An analyst performs a query

[6] The parameter space is also sometimes called the "domain" of the database.

in two steps. First, the analyst *selects* a subspace of the parameter space using a SQL SELECT-like syntax. For example, the analyst may select the subspace "age≥10 AND age<20 AND gender=male AND smoker=1". Next, the analyst *runs* an aggregate query like count, sum or average on this selected subspace. To protect privacy, UniTraX adds random noise to the result of the query. The amount of noise added is determined by a privacy parameter, ε, that the analyst provides with the query. For lower values of ε, the result is more noisy, but the reduction of privacy budget is less (thus leaving more budget for future queries).

The novel aspect of UniTraX is how it tracks budgets. When an aggregate query with privacy parameter ε is made on a selected subspace S, UniTraX simply records that budget ε has been consumed from subspace S. The *remaining budget* of any point in the parameter space is the point's initial budget (from the point's designated initial budget field) minus the ε's of all past queries that ran on subspaces containing the point.

The consumed budgets of all subspaces are *public*—analysts can ask for them at any time. This allows analysts to determine which subspaces have been heavily queried in the past and, hence, become aware of possible data biases. Moreover, analysts may select only subspaces with sufficient remaining budgets in subsequent queries, thus increasing their confidence in analysis outcomes, as illustrated in Sect. 2.

To respect privacy budgets, it is imperative that a query with privacy parameter ε does not execute on any points whose remaining budget is less than ε. This is enforced by query rejection, where a query is executed only if all points in the selected subspace have remaining budget at least ε. Note that this check is made on not only actual records but all points in the selected subspace. If any such point does not have sufficient remaining budget, the query is rejected and an error is returned to the analyst (who may then select a smaller subspace with higher initial budgets and retry the query). Whether a query is executed or rejected depends only on the consumption history, which is public, so rejecting the query provides no additional information to the analyst.

Initial Budgets. UniTraX is agnostic to the method used to determine initial budgets of actual records and supports any scheme for setting initial budgets on actual records. The simplest scheme would assign the same, fixed initial budget to every actual record. A more complex scheme may allow users to choose from a small fixed set of initial budgets for each record they provide, while the most complex scheme may let users freely choose any initial budget for every record.

4 Formal Description and Differential Privacy

In this section, we describe UniTraX using a formal model. We specify the differential privacy property that we expect UniTraX to satisfy and formally prove that the property is indeed satisfied. Our formalization is directly based on ProPer's formalization [10], which we find both elegant and natural.

4.1 Formal Model of UniTraX

Database. We treat the database as a table with n columns of arbitrary types $\mathcal{C}_1, \ldots, \mathcal{C}_n$ and an initial budget column—a non-negative real number. The type of each record, also called the parameter space, is $\mathcal{R} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_n \times \mathcal{C}_B$, where $\mathcal{C}_B = \mathbb{R}^{\geq 0}$ is the type of the initial budget column. At any point of time, the state of the database is a set E of records from the parameter space ($E \in 2^{\mathcal{R}}$).

UniTraX. UniTraX acts as a reference monitor between the database and the analyst. Its internal state consist of two components: (1) the consumption history H and (2) the select table T.

1. UniTraX tracks the budget consumed by past queries on every subspace of the parameter space. Formally, this is equivalent to storing a map from points in the parameter space to non-negative real numbers. We call this map the *consumption history*, denoted H. H has the type $\mathcal{H} = \mathcal{R} \to \mathbb{R}^{\geq 0}$. Intuitively, $H(r)$ is the amount of budget consumed by past queries that ran on subspaces containing the point r of the parameter space.
2. To run an aggregate query in UniTraX, the analyst must first select a subspace of the parameter space. To support selection of records that have at least a stipulated *remaining budget*, UniTraX allows selected subspaces to also span the consumption history. Consequently, a selected subspace is a subset of $\mathcal{R} \times \mathbb{R}^{\geq 0}$ (points extended with their *consumed* budgets). We represent such subspaces via logical predicates *sspace* of type $\mathcal{P} = \mathcal{R} \times \mathbb{R}^{\geq 0} \to \{\mathsf{true}, \mathsf{false}\}$. For the analyst's convenience, UniTraX allows storing a list of selected subspaces, indexed by *subspace variables* drawn from a set SVar. UniTraX stores the association between subspace variables and subspaces in a *select table*, T, of type $\mathsf{SVar} \to \mathcal{P}$.

Analyst. We model an adaptive analyst, who queries UniTraX based on an internal program and previously received answers. Formally, the analyst is a (possibly infinite) state machine with states denoted by P and its decorated variants, and state transitions defined by the relation $P \xrightarrow{a} P'$. Here a, b denote *interactions* between the analyst and UniTraX. Allowed interactions are summarized in Fig. 2. Note that interactions consist of either an instruction to, or an observable output from UniTraX, or both. In detail, the interactions are:

- $sv := sspace$ represents the instruction to UniTraX to associate the subspace variable sv with the subspace *sspace*, which must be in $\mathcal{P}$. This models the selection of a subspace (for use in later aggregation queries).
- $Q_\varepsilon(sv)?n$ models the instruction to UniTraX to run the aggregation query Q with privacy parameter ε on the subspace previously mapped to variable sv. The interaction also includes the noised result n of the query. If some point in subspace sv has remaining budget less than ε, the output n is 'reject'.
- update represents an output from UniTraX to the analyst indicating that the database has been updated. The output does not specify which records were added or deleted (else the analyst could trivially break DP).

a, b	$::= sv := sspace$	select subspace *sspace* and name it sv
	$Q_\varepsilon(sv)?n$	run aggregation query Q on sv, observe output n
	update	database update
	read?H	read consumption history, result is H

Fig. 2. Allowed interactions between analyst and UniTraX

- read?H models reading the entire current consumption history by the analyst. H is the history returned by UniTraX.

We make no assumptions about the analyst (i.e., its state machine). It may select any subspace, run any aggregation query, and read the consumption history at any time. However, for technical reasons we assume (like ProPer) that the analyst is internally deterministic and deadlock-free, meaning that it branches only on observable output from the database and that it can always make progress.[7] Our assumptions are formalized by the following condition: If $P \xrightarrow{a} P'$ and $P \xrightarrow{b} P''$, then

1. if $a = b$ then $P' = P''$
2. if $a = (sv := sspace)$ then $a = b$
3. if $a = Q_\varepsilon(sv)?n$ then $b = Q_\varepsilon(sv)?n'$ for some n' and for all n'' there exists P''' with $P \xrightarrow{Q_\varepsilon(sv)?n''} P'''$
4. if $a = \text{read?}H$ then $b = \text{read?}H'$ for some H' and for all H'' there exists P''' with $P \xrightarrow{\text{read?}H''} P'''$

Configuration. A configuration $\mathbb{C} = (P, E, H, T)$ represents the state of the complete system. It includes the state of the analyst (P), the database of actual records (E) and the internal state of UniTraX (consumption history H and select table T).

Execution Semantics. We model the evolution of the system using transitions $\mathbb{C} \xrightarrow{\alpha}_p \mathbb{C}'$. Here, $\alpha \in \mathsf{Act}$ denotes an *action label* describing an operation within the system and p is a transition probability (real number between 0 and 1). The transition $\mathbb{C} \xrightarrow{\alpha}_p \mathbb{C}'$ reads as follows: If, in configuration $\mathbb{C}$, the operation α happens, then, with probability p, the configuration changes to $\mathbb{C}'$. α may be any one of:

- τ: analyst selects a subspace
- $n \in \mathsf{Val}$: query by analyst that returns result n
- reject: query by analyst that is rejected
- $R_{in} : R_{del}$: database update that adds records R_{in} and removes records R_{del}
- H: analyst reads consumption history H

The transition system $\mathbb{C} \xrightarrow{\alpha}_p \mathbb{C}'$ is defined by the five rules shown in Fig. 3. These rules model the system's behavior as follows.

[7] These restrictions do not affect the analyst's attack capabilities.

$$\textsc{Update}\quad \frac{P \xrightarrow{\text{update}} P' \qquad R_{in}, R_{del} \subseteq \mathcal{R}}{(P,E,H,T) \xrightarrow{R_{in}:R_{del}}_1 (P', (E \cup R_{in}) \setminus R_{del}, H, T)}$$

$$\textsc{Select}\quad \frac{P \xrightarrow{sv:=sspace} P' \qquad sspace \in \mathcal{P}}{(P,E,H,T) \xrightarrow{\tau}_1 (P', E, H, T[sv := sspace])} \qquad \textsc{Read-History}\quad \frac{P \xrightarrow{\text{read?}H} P'}{(P,E,H,T) \xrightarrow{H}_1 (P', E, H, T)}$$

$$\textsc{Query}\quad \frac{\begin{array}{c} P \xrightarrow{Q_\varepsilon(sv)?n} P' \qquad sspace := T(sv) \in \mathcal{P} \\ \forall r \in \mathcal{R}.sspace(r, H(r)) \Rightarrow\ H(r) + \varepsilon \leq r.c_B \qquad p = \text{Prob}[Q_\varepsilon(E|_{sspace,H}) = n] \end{array}}{(P,E,H,T) \xrightarrow{n}_p (P', E, H', T)}$$

$$\text{where } H'(r) := \begin{cases} H(r) + \varepsilon & \text{if } sspace(r, H(r)) \\ H(r) & \text{otherwise} \end{cases}$$

$$\text{and } E|_{sspace,H} \stackrel{\text{def}}{=} \{r \in E \mid sspace(r, H(r))\}$$

$$\textsc{Reject}\quad \frac{\begin{array}{c} P \xrightarrow{Q_\varepsilon(sv)?\,\text{reject}} P' \\ sspace := T(sv) \in \mathcal{P} \qquad \exists r \in \mathcal{R}.sspace(r, H(r)) \wedge\ H(r) + \varepsilon > r.c_B \end{array}}{(P,E,H,T) \xrightarrow{\text{reject}}_1 (P', E, H, T)}$$

Fig. 3. Semantics of UniTraX

(Update) Models a database update by adding some record set R_{in} and removing some record set R_{del} from the database E. This transition returns to the analyst the observable output 'update' (first premise).

(Select) Represents the analyst's selection of subspace *sspace*, naming it *sv*.

(Read-History) Denotes the analyst reading the current consumption history H. This rule forces our privacy proofs to internally show that the consumption history is indeed public.

(Query) Models the successful execution of aggregation query Q on subspace *sspace* identified by *sv* with privacy parameter ε. The execution requires all points in *sspace* to have a remaining budget of at least ε. A point r is in *sspace* if $sspace(r, H(r)) = \mathsf{true}$. (In the rule, $r.c_B$ is short-hand for the initial budget column of point r.) As a consequence of the query, two things happen. First, the consumption history of all points in the subspace is increased by ε, to record that a query with privacy parameter ε has run on the subspace. Second, the answer to query Q executed over those records that are both in the subspace and actually exist in the database E (selected by the operation $E|_{sspace,H}$) is returned to the analyst after adding differentially private noise for the parameter ε. The

transition's probability p is equal to the probability of getting the specific noised answer for the query (the noised answer is denoted n in the rule).

(REJECT) Represents UniTraX's rejection of query Q due to some point in the query's selected subspace not having sufficient remaining budget. The analyst observes a special response 'reject' (first premise).

With the notable exception of (QUERY), all rules are deterministic—they happen with probability 1 (the p in $\xrightarrow{\alpha}_p$ is 1).

Trace Semantics. The relation $\mathbb{C} \xrightarrow{\alpha}_p \mathbb{C}'$ describes a single step of system evolution. We lift this definition to multiple steps. A *trace* σ is a (possibly empty) finite sequence of labels $\alpha_1, \ldots, \alpha_n$. We write $\mathbb{C} \overset{\sigma}{\Longrightarrow}_q \mathbb{C}'$ to signify that configuration $\mathbb{C}$ evolves in multiple steps to configuration $\mathbb{C}'$ with probability q. The individual steps of the evolution have labels in σ. Formally, we have:

$$\frac{}{\mathbb{C} \overset{[]}{\Longrightarrow}_1 \mathbb{C}} \qquad \frac{\mathbb{C} \xrightarrow{\alpha}_p \mathbb{C}' \qquad \mathbb{C}' \overset{\sigma}{\Longrightarrow}_q \mathbb{C}''}{\mathbb{C} \overset{\alpha\,\sigma}{\Longrightarrow}_{p \cdot q} \mathbb{C}''}$$

We abbreviate $\mathbb{C} \overset{\sigma}{\Longrightarrow}_q \mathbb{C}'$ to $\mathbb{C} \overset{\sigma}{\Longrightarrow}_q$ when $\mathbb{C}'$ is irrelevant.

Note that from the transition semantics (Fig. 3) it follows that a trace σ records all updates to the database and all observations of the analyst (the latter is comprised of all responses from UniTraX to the analyst).

Extension to Silent Record Dropping. Up to this point, our design rejects a query whose selected subspace includes at least one point with insufficient remaining budget. This protects user privacy and prevents unknown dataset bias. However, in some cases, an analyst might prefer the risk of unknown dataset bias over modifying their existing programs to handle query rejections. This might be the case, for instance, if the analyst already knows by other means that the percentage of records with insufficient budget will be negligible. In this case, it would be preferable to *automatically drop records* with insufficient budget during query execution, as in ProPer. It turns out that we can provide silent record dropping without weakening the privacy guarantee. In the following paragraph, we detail a simple extension of UniTraX that allows the analyst to specify *for each query individually* whether the system should silently drop records with insufficient remaining budgets instead of rejecting the query.

In order to enable silent record dropping, we introduce an extended query interaction $Q_\varepsilon^{\mathsf{drop}}(sv)?n$ for the analyst's program. Unlike the previously described interaction, $Q_\varepsilon(sv)?n$, this interaction cannot fail (be rejected). The semantics of $Q_\varepsilon^{\mathsf{drop}}(sv)?n$ is defined by the new rule (QUERY-DROP) shown in Fig. 4. The query executes on those records in database E that (1) are in subspace *sspace*, and (b) have remaining budget at least ε. These records are selected by $E\|_{sspace,H,\varepsilon}$. As a consequence of the query, two things happen. First, the consumption history of all points in the parameter space satisfying (1) and (2) is increased by ε. Second, the answer of the query is returned to the analyst with probability p, which is determined by the same method used in (QUERY).

QUERY-DROP

$$\frac{P \xrightarrow{Q_\varepsilon^{\mathsf{drop}}(sv)?n} P' \qquad sspace := T(sv) \in \mathcal{P} \qquad p = \mathrm{Prob}[Q_\varepsilon^{\mathsf{drop}}(E\|_{sspace,H,\varepsilon}) = n]}{(P, E, H, T) \xrightarrow{n}_p (P', E, H', T)}$$

$$\text{where } H'(r) := \begin{cases} H(r) + \varepsilon & \text{if } sspace(r, H(r)) \wedge H(r) + \varepsilon \leq r.c_B \\ H(r) & \text{otherwise} \end{cases}$$

$$\text{and } E\|_{sspace,H,\varepsilon} \stackrel{\text{def}}{=} \{r \in E \mid sspace(r, H(r)) \wedge H(r) + \varepsilon \leq r.c_B\}$$

Fig. 4. Semantics extension for silent record dropping

4.2 Privacy Property and Its Formalization

UniTraX respects the initial privacy budget of every record added to the database in the sense of differential privacy. Before explaining this property formally, we recap the standard notion of differential privacy due to Dwork et al. [6].

Standard Differential Privacy. Let Q be a randomized algorithm on a database that produces a value in the set V. For example, the algorithm may compute a noisy count of the number of entries in the database. We say that Q is ε-differentially private if for any two databases D, D' that differ in one record and for any $V' \subseteq V$,

$$\left| \ln \left(\frac{Pr\left[Q(D) \in V'\right]}{Pr\left[Q(D') \in V'\right]} \right) \right| \leq \varepsilon.$$

In words, the definition says that for two databases that differ in only one record, the probabilities that the analyst running Q makes a specific observation are very similar. This means that any individual record does not significantly affect the probability of observing any particular outcome. Hence, the analyst cannot infer (with high confidence) whether any specific record exists in the database.

If the analyst runs n queries that are ε_1-, ..., ε_n-differentially private, then the total *loss* of privacy is defined as $\varepsilon_1 + \ldots + \varepsilon_n$. Typically, a maximum *privacy budget* is set when the analyst is given access to the database and after each ε-differentially private query, ε is subtracted from this budget. Once the budget becomes zero, no further queries are allowed. In this mode of use, DP guarantees that for any two possible databases D, D' that differ in at most one record, for any sequence of queries $\boldsymbol{Q}$, and for any sequence of observations $\boldsymbol{o}$,

$$\left| \ln \left(\frac{Pr\left[\boldsymbol{Q} \text{ results in } \boldsymbol{o} \text{ on } D\right]}{Pr\left[\boldsymbol{Q} \text{ results in } \boldsymbol{o} \text{ on } D'\right]} \right) \right| \leq \eta,$$

where η is the privacy budget.

Our Privacy Property. We use the same privacy property as ProPer. This privacy property generalizes differential privacy described above by accounting for

$$dist(\sigma, \sigma') \stackrel{\text{def}}{=} \begin{cases} \bigcup_{i \in [1,n]} dist(\alpha_i, \alpha'_i) & \text{if } \sigma = \alpha_1, \ldots, \alpha_n \text{ and } \sigma' = \alpha'_1, \ldots, \alpha'_n \\ (R_{in} \Delta R'_{in}) \cup (R_{del} \Delta R'_{del}) & \text{if } \sigma = R_{in} : R_{del} \text{ and } \sigma' = R'_{in} : R'_{del} \\ \emptyset & \text{if } \sigma = \sigma' \end{cases}$$

Fig. 5. Trace distance

dynamic addition and deletion of records and, importantly, allowing all new records to carry their own initial budgets. Informally, our privacy property is the following. Consider two possible traces σ_0 and σ_1 that can result from the same starting configuration. Suppose that σ_0 and σ_1 differ *only* in the updates made to the database and are otherwise identical. Let p_0 and p_1 be the respective probabilities of the traces. Then, $\left|\ln\left(\frac{p_0}{p_1}\right)\right| \leq \eta$, where η is the sum of the initial budgets of all records in which the database updates differ between σ_0 and σ_1.

Why is this a meaningful privacy property? We remarked earlier that a trace records all observations that the analyst (adversary) makes. Consequently, by insisting that the traces agree everywhere except on database updates, we are saying that the two traces agree on the analyst's observations. Hence, if an analyst makes a sequence of observations under database updates from σ_0 with probability p_0, then the probability that the analyst makes the *same* observations under database updates from σ_1 is very close to p_0. In fact, the log of the ratio of the two probabilities is bounded by the sum of the initial budgets of the records in which the updates differ. This is a natural generalization of DP's per-database budgets to per-record budgets.

To formalize this property, we define a partial function $dist(\sigma, \sigma')$ that returns the set of records in which database updates in σ and σ' differ if σ and σ' agree pointwise on all labels other than database updates. If σ and σ' differ at a label other than database update then $dist(\sigma, \sigma')$ is undefined. The formal definition is shown in Fig. 5.

Definition 1 (Privacy). *We say that UniTraX preserves privacy if whenever* $\mathbb{C} \stackrel{\sigma_0}{\Longrightarrow}_{p_0}$ *and* $\mathbb{C} \stackrel{\sigma_1}{\Longrightarrow}_{p_1}$ *and* $dist(\sigma_0, \sigma_1) = R$*, then* $\left|\ln\left(\frac{p_0}{p_1}\right)\right| \leq \sum_{r \in R} r.c_B$.

Our main result is that UniTraX is private in the sense of the above definition.

Theorem 1 (Privacy of UniTraX). *UniTraX preserves privacy in the sense of Definition 1.*

We prove this theorem by first proving a strong invariant of configurations that takes into account how UniTraX tracks the consumption history. The entire proof is in our technical report [19].

5 Implementation

We have implemented UniTraX on top of PINQ, an earlier framework for enforcing differential privacy with a global budget for the database [17]. We briefly review relevant details of PINQ before explaining our implementation.

PINQ Review. PINQ adds differential privacy to LINQ, a general-purpose database query framework. LINQ defines *Queryable* objects, abstractions over data sources, e.g., a database table. The Queryable object may be *transformed* by a SQL SELECT-like operation to obtain another Queryable object representing selected records from the table. One may run an aggregate query on this second object to obtain a specific value.

Building on LINQ, PINQ maintains a global privacy budget for the entire database. This budget is set when a Queryable object is initialized. Subsequently, differentially-private noise is added to every aggregation query on every object derived from this Queryable object and the global budget is appropriately reduced.

UniTraX Implementation. Our implementation currently supports only query execution with rejection. The main addition to PINQ is tracking of consumption budgets over subspaces. *In principle*, we must store the consumption budget for every point in the parameter space. *In practice*, queries tend to select contiguous ranges, so at any point of time, the parameter space splits into contiguous subspaces, each with a uniform consumption budget. Accordingly, our implementation tries to cluster contiguous subspaces with identical consumption and represents them efficiently.

Our interface defines a new object type, UQueryable, which represents a subspace. Like Queryable, this object can be transformed via SQL SELECT-like operations to derive other, smaller UQueryable objects. To run an analysis on a subspace, the analyst invokes a special function, GetAsPINQ, to convert a UQueryable object representing the subspace into a PINQ object representing the same subspace. This special function also takes as an argument a budget, which the analysis will eventually consume. The function first checks that this budget is larger than the remaining budget of all points in the subspace. If not, the function fails. Otherwise, this budget is immediately added to the consumption budget of the subspace and a fresh PINQ object initialized with this budget is returned. Subsequently, the analyst can run any queries on the PINQ object and PINQ's existing framework enforces the allocated budget.

We also provide a new interface to the analyst to ask for the maximum budget consumed in a given subspace.

Typical Analysis Workflow. We briefly describe the steps an analyst must follow to run an analysis on our implementation. Assume that the analyst wants to analyze records within a specific subspace with a set of queries that require a certain amount of budget to run successfully. Further assume that the analysis needs to run on a stipulated minimum number of user records for its results to be meaningful. The analyst would perform the following steps:

1. Obtain the initial UQueryable object representing the entire database.
2. Select the desired subspace obtaining another UQueryable object.
3. Obtain the maximum budget consumed on the second object.
4. Add the budget required for the analysis and a budget for a noisy count to the just-obtained maximum budget.
5. Select the subspace that has at least the just-calculated sum of budgets available, obtaining yet another UQueryable object.
6. Obtain a PINQ object from the last UQueryable object with the PINQ budget set to the budget of the count.
7. Perform a (noisy) count on the PINQ object. If it is too low, stop here.
8. Otherwise, obtain another PINQ object, this time with the budget required for the analysis.
9. Perform the analysis on the second PINQ object. All records in the PINQ object have enough budget for the full analysis.

Data Stream Analysis. UniTraX can be directly used for analysis on streams of data since its design and privacy proof already take record addition and deletion into account. To allow analysts to use the full budget of newly arriving records, we assume records to be timestamped on arrival; this timestamp is another column in our parameter space. At any time, all active analyses use points with timestamps in a specific window of time only. When the budgets of points in the window run out, the window is shifted to newer timestamps. Records with timestamps in the old window can be discarded. All analyses share the budgets of points in the active time window.

6 Preliminary Evaluation

This section presents a preliminary evaluation of the performance of our implementation of UniTraX. It is preliminary in that (1) it uses only one dataset (the New York City taxi ride dataset [18, 21]), and (2) we carry out only one "analysis session". The session consists of queries that perform the basic statistical operations of count, average, and median.

Objective. Of primary interest to us is the increase in end-to-end latency experienced by the analyst (time from query submission to answer reception) as compared to both PINQ (reference DP) and LINQ (baseline that provides no privacy). Additionally, we want to understand the overhead of storing UniTraX's budget consumption history data structure.

In absolute terms, these overheads are a function of the access pattern on the parameter space. The exact column names, the data in them or the precise queries do not matter for this. Nonetheless, we briefly describe the dataset we use and the queries we run. The queries are deliberately chosen to be simple since long-running, complex queries will mask UniTraX's relative overheads.

Dataset. We use all taxi rides of New York City reported for January 2013 (≈14 M records). We modify these records to only contain numerical data and add an additional initial budget for each. For the purpose of our measurements all budgets are chosen high enough so that no budgets expire.

Analysis Session. Our session is roughly patterned off of the analysis of the same dataset described in [12]. The session consists of 1213 queries split into three groups. The first group covers the entire geographic area, and consists of six histograms for different columns. The subsequent groups focus on a 16×16 grid of squares in Manhattan. The second group of queries counts the number of rides in each square, and takes averages over two different columns for squares that have more than 5000 rides with sufficient budget. The third group counts rides again and takes the median of one column for squares that have more than 1000 rides with sufficient budget.

Experimental Setups. We run the session over each of the following three setups:

1. Directly on LINQ using the LINQ-to-SQL interface (no privacy protection).
2. Through a PINQ object (DP protection with a global budget).
3. With UniTraX.

All numbers presented in this section are averages of five runs of the session.

Hardware. All experiments run on two identical commodity Dell PowerEdge M620 blade systems. Each is equipped with dual Intel Xeon E5-2667 v2 8-core CPUs with Hyperthreading (total of 32 hardware threads per machine) and 256 GB of main memory. Both systems are connected to the same top-of-rack switch with two bonded 1 Gbit/s connections each.

Software. We use Microsoft Windows Server 2016 on both systems. The first system runs both UniTraX as well as the client query program. Microsoft Visual Studio Community 2015 is the only additional software installed for these tasks. The second system runs Microsoft SQL Server 2016 Developer Edition as the remote database server. To optimize database performance we put data and index files of our database onto a RAM-disk, create indexes that fit our queries, and make the database read-only.

Absolute and Relative Latency Overheads. Figure 6 presents absolute end-to-end latencies for the three experimental setups: direct, only PINQ, and UniTraX. A random 5 % sample of the 1213 queries is shown, sorted on the x-axis by increasing latency with respect to the direct experiment. Overheads are moderate. As expected, UniTraX is usually slower than PINQ, which is slower than direct query execution without any privacy protection. In 3.2 % of the cases, UniTraX outperforms direct and PINQ. We verified that in these cases the database server chose to do a sequential table scan for direct and PINQ but a parallel and thus faster index scan for UniTraX. We were unable to force parallel execution for direct and PINQ.

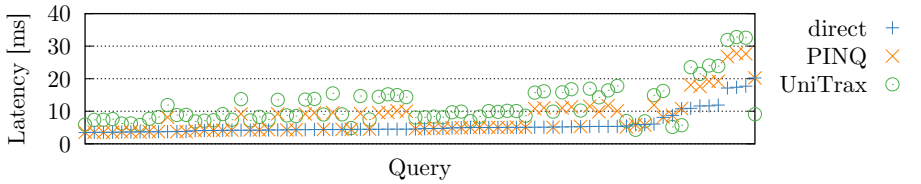

Fig. 6. End-to-end latencies of a 5 % sample of the 1213 queries ordered according to latencies of direct. The trend in the order of performance is evident. UniTraX is slower than PINQ, which is slower than direct. Where UniTraX outperforms the others, the database chose a better query plan for UniTraX's queries.

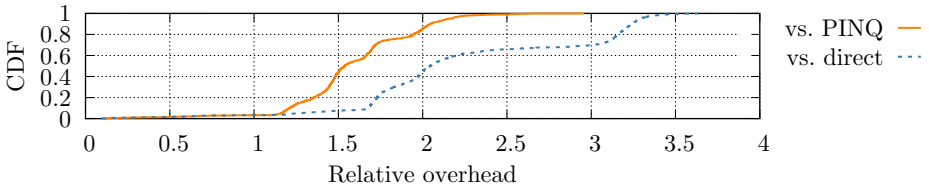

Fig. 7. CDF of relative overheads incurred by UniTraX across all 1213 queries. At the 99th-percentile UniTraX is 2.5 x slower than PINQ and 3.5 x slower than the direct case. The initial tail of inverse overhead before 1 consists of 3.2 % of queries where the database chooses sub-optimal query plans for PINQ and the direct case.

Figure 7 presents a CDF for all 1213 queries in terms of the overhead of UniTraX relative to direct and PINQ respectively. We observe that in half of the cases, UniTraX is 1.5 x slower than PINQ and 2 x slower than the direct case. At the 99th-percentile UniTraX is 2.5 x slower than PINQ and 3.5 x slower than the direct case. The figure includes a tail between 0 and 1, indicating that UniTraX is sometimes faster than PINQ or the direct case. As explained before, this behavior is due to the database choosing sub-optimal query plans for PINQ and the direct case. On average, UniTraX is 1.3 x slower than PINQ and 1.8 x slower than the direct case. In summary, latency overheads introduced by UniTraX are moderate.

Size of Budget Tracking State. Figure 8 shows the number of subspaces tracked by UniTraX at the beginning of each query. Numbers are again ordered according to query latencies in the direct case (see Fig. 6). These numbers do not change across different runs. The two curves represent two analyst query strategies, one with and one without *re-balancing*. These two curves illustrate that the analyst can dramatically affect the size of the budget tracking state based on how queries are formulated.

In the "without re-balancing" strategy (w/o RB), the analyst queries data only within a range of interest. For instance, suppose that the analyst is interested in a histogram of fares between $0 and $100. The analyst may request 10 $10 bars. As long as each bar consumes the same budget, UniTraX will opti-

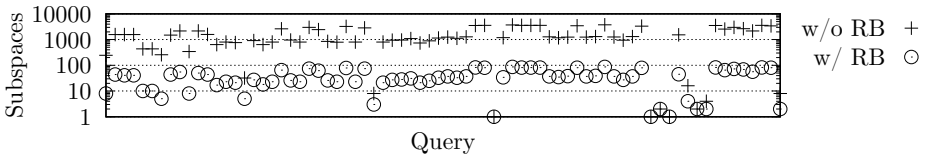

Fig. 8. Number of subspaces UniTraX tracks throughout the execution of the queries shown in Fig. 6. Reported numbers are obtained at the beginning of each query and do not change across different runs. The different curves represent two different analyst query strategies, one where the analyst only requests data of interest (w/o RB), and one where the analyst requests extra data in order to improve UniTraX's re-balancing (w/ RB). This shows that analysts can substantially reduce the overhead of UniTraX through careful selection of query parameters.

mize tracking state and merge the subspaces of these 10 bars into a single subspace. The range above the histogram (above $100), however, cannot be merged. As a result, UniTraX stores two subspaces for the fare column. The same happens with other columns, with the result that there is a combinatoric explosion in the number of subspaces because of the combinations of the columns' multiple subspaces.

In the "with re-balancing" strategy (w/ RB), the analyst instead queries data that covers the full range of a column, even though the analyst may not be interested in all of that range, or may even know that no data exists in some subrange (e.g., no taxi pickups over water). As a result, UniTraX is able to merge more subspaces, even those of different columns. At the cost of budget, this reduces the number of subspaces substantially, in this case by more than an order of magnitude. Re-balancing thus allows analysts to trade-off overheads against budget savings.

7 Related Work

Due to its age, the area of privacy-preserving data analytics has amassed a vast amount of work. The related work section of [16] provides a good overview of early work in this space. Around ten years ago Dwork et al. introduced differential privacy or DP [6], which quickly developed into a standard for private data analytics research (see [7] and [8]). In this section, we focus on research that investigates heterogeneous or personalized budgets, tracking of personalized budgets, private analytics on dynamic data sets, and PINQ, the system our implementation is based on.

Alaggan et al. [1] propose heterogeneous differential privacy (HDP) to deal with user-specific privacy preferences. They allow users to provide a separate *privacy weight* for each individual data field, a granularity finer than that supported by UniTraX. However, the total privacy budget is a global parameter. When computing a statistical result over the dataset, HDP perturbs each accessed data value individually according to its weight and the global privacy

budget. UniTraX can be extended to support per field rather than per record budgets at the cost of additional runtime latency. Further, UniTraX allows analysts to query parts of a dataset without consuming the privacy budget of other parts. UniTraX also supports a greater set of analytic functions, e.g., *median*. HDP does not provide these capabilities. Queries can only run over the whole dataset and, as privacy weights are secret, the exact amount of answer perturbation remains unknown to the analyst.

Jorgensen et al.'s personalized differential privacy (PDP) is a different approach to the same problem [15]. In contrast to UniTraX, PDP trusts analysts and assumes that per-user budgets are public. It tracks the budget globally but manages to avoid being limited to the most restrictive user's budget by allowing the analyst to sample the dataset prior to generating any statistical output. Depending on the sampling parameters the analyst is able to use more than the smallest user budget for a query (but on a subset of records). PDP only supports querying the entire dataset at once. Nevertheless, we believe that a combination of PDP and UniTraX could be useful, in particular to allow analysts to make high budget queries on low budget records. The combination could also do away with PDP's assumption that analysts be trusted.

In place of personalized privacy protection, Nissim et al. [20] and earlier research projects [5, 14] provide users different monetary compensation based on their individual privacy preferences. It is unclear whether these models can be combined with UniTraX as they do not provide any personalized privacy protection. Users with a higher valuation receive a higher compensation but suffer the same privacy loss as other users.

Despite allowing users to specify individual privacy preferences, all the above systems track budget globally and do not allow analysts to selectively query records and consume budget only from the queried records. To the best of our knowledge, ProPer [10] is the only system that allows this. We compared extensively to ProPer in Sect. 2. Our formal model in Sect. 4 is also based on ProPer's formal model. Google's RAPPOR [11] likewise provides differential privacy guarantees based on user-provided parameters, but the system model is significantly different from ours and the privacy guarantee holds only when certain cross-query correlations do not occur. In contrast, we (and ProPer) need no such assumptions.

Differential privacy is being increasingly applied to dynamic datasets rather than static databases. Since the first consideration of such scenarios in 2010 [9], numerous systems have emerged [2–4, 13, 22, 23] that aggregate dynamic data streams rather than static datasets in a privacy-preserving manner. UniTraX and ProPer can be immediately used for dynamic data streams since their designs and privacy proofs already take record addition and deletion into account.

As explained in Sect. 5, our UniTraX implementation is based on the Privacy Integrated Queries (PINQ) [17] platform, which offers privacy-preserving data analysis capabilities. PINQ, in turn, is based on the Language Integrated Queries (LINQ) framework, a well-integrated declarative extension of the .NET platform. LINQ provides a unified object-oriented data access and query inter-

face, allowing analysts data access independent of how the data is provided and where the answer is finally computed. Data providers can be switched without changing code and can be, e.g., local files, remote SQL servers, or even massive parallel cluster systems like DryadLINQ [24]. PINQ provides a thin DP wrapper over LINQ. For all queries, it ensures that sufficient budget is available and that returned answers are appropriately noised. The maximum budget must be provided during object initialization. Our implementation uses PINQ in an unconventional way—we initialize a new PINQ object prior to every data analysis, and use PINQ to enforce a stipulated budget. Additionally, we track budget consumption on subspaces of the parameter space across queries.

8 Conclusion and Future Work

This paper presented UniTraX, the first differentially private system that supports per-record privacy budgets, tells the analyst where (in the parameter space) budgets have been used in the past, and allows the analyst to query only those points that still have sufficient budget for the analyst's task. UniTraX attains this by tracking budget consumption not on actual records in the database, but on points in the parameter space. As a result, information about budget consumption reveals nothing about actual records to the analyst.

We have also presented a formal model of UniTraX and a formal proof that UniTraX respects differential privacy for all records. Our prototype implementation incurs moderate overheads on a realistic workload.

There are several directions for future work. First, our implementation is not very optimized and there is scope for reducing overheads even further. Second, UniTraX can be extended to track budgets at even finer granularity, e.g., a budget for every field. Third, one could investigate how queries can be optimized to reduce budget consumption.

References

1. Allagan, M., Gambs, S., Kermarrec, A.M.: Heterogeneous differential privacy. Journal of Privacy and Confidentiality 7(2), 127–158, Article 6 (2016), http://repository.cmu.edu/jpc/vol7/iss2/6/
2. Chan, T.H., Li, M., Shi, E., Xu, W.: Differentially private continual monitoring of heavy hitters from distributed streams. In: Fischer-Hübner, S., Wright, M.K. (eds.) Proceedings of the 12th International Symposium on Privacy Enhancing Technologies (PETS '12). Lecture Notes in Computer Science, vol. 7384, pp. 140–159. Springer, Berlin, Heidelberg, Germany (2012), https://doi.org/10.1007/978-3-642-31680-7_8
3. Chan, T.H., Shi, E., Song, D.: Private and continual release of statistics. ACM Transactions on Information and System Security (TISSEC) 14(3), 26:1–26:24 (2011), https://doi.org/10.1145/2043621.2043626
4. Chan, T.H., Shi, E., Song, D.: Privacy-preserving stream aggregation with fault tolerance. In: Keromytis, A.D. (ed.) Revised Selected Papers of the 16th International Conference on Financial Cryptography and Data Security (FC '12). Lecture

Notes in Computer Science, vol. 7397, pp. 200–214. Springer, Berlin, Heidelberg, Germany (2012), https://doi.org/10.1007/978-3-642-32946-3_15
5. Dandekar, P., Fawaz, N., Ioannidis, S.: Privacy auctions for recommender systems. In: Goldberg, P.W., Guo, M. (eds.) Proceedings of the 8th International Workshop on Internet and Network Economics (WINE '12). Lecture Notes in Computer Science, vol. 7695, pp. 309–322. Springer, Berlin, Heidelberg, Germany (2012), https://doi.org/10.1007/978-3-642-35311-6_23
6. Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP '06). Lecture Notes in Computer Science, vol. 4052, pp. 1–12. Springer, Berlin, Heidelberg, Germany (2006), https://doi.org/10.1007/11787006_1
7. Dwork, C.: Differential privacy: A survey of results. In: Agrawal, M., Du, D., Duan, Z., Li, A. (eds.) Proceedings of the 5th International Conference on Theory and Applications of Models of Computation (TAMC '08). Lecture Notes in Computer Science, vol. 4978, pp. 1–19. Springer, Berlin, Heidelberg, Germany (2008), https://doi.org/10.1007/978-3-540-79228-4_1
8. Dwork, C.: A firm foundation for private data analysis. Communications of the ACM (CACM) 54(1), 86–95 (2011), https://doi.org/10.1145/1866739.1866758
9. Dwork, C., Naor, M., Pitassi, T., Rothblum, G.N.: Differential privacy under continual observation. In: Mitzenmacher, M., Schulman, L.J. (eds.) Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC '10). pp. 715–724. ACM, New York, NY, USA (2010), https://doi.org/10.1145/1806689.1806787
10. Ebadi, H., Sands, D., Schneider, G.: Differential privacy: now it's getting personal. In: Rajamani, S.K., Walker, D. (eds.) Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15). pp. 69–81. ACM, New York, NY, USA (2015), https://doi.org/10.1145/2676726.2677005
11. Erlingsson, Ú., Pihur, V., Korolova, A.: RAPPOR: randomized aggregatable privacy-preserving ordinal response. In: Ahn, G., Yung, M., Li, N. (eds.) Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14). pp. 1054–1067. ACM, New York, NY, USA (2014), https://doi.org/10.1145/2660267.2660348
12. Espín Noboa, L., Lemmerich, F., Singer, P., Strohmaier, M.: Discovering and characterizing mobility patterns in urban spaces: A study of Manhattan taxi data. In: Bourdeau, J., Hendler, J., Nkambou, R., Horrocks, I., Zhao, B.Y. (eds.) Proceedings of the 25th International Conference Companion on World Wide Web (WWW '16 Companion). pp. 537–542. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland (2016), https://doi.org/10.1145/2872518.2890468
13. Friedman, A., Sharfman, I., Keren, D., Schuster, A.: Privacy-preserving distributed stream monitoring. In: Proceedings of the 21st Annual Symposium on Network and Distributed System Security (NDSS '14). ISOC (2014), https://doi.org/10.14722/ndss.2014.23128
14. Ghosh, A., Roth, A.: Selling privacy at auction. In: Shoham, Y., Chen, Y., Roughgarden, T. (eds.) Proceedings of the 12th ACM Conference on Electronic Commerce (EC '11). pp. 199–208. ACM, New York, NY, USA (2011), https://doi.org/10.1145/1993574.1993605
15. Jorgensen, Z., Yu, T., Cormode, G.: Conservative or liberal? Personalized differential privacy. In: Gehrke, J., Lehner, W., Shim, K., Cha, S.K., Lohman, G.M. (eds.)

Proceedings of the 31st IEEE International Conference on Data Engineering (ICDE '15). pp. 1023–1034. IEEE (2015), https://doi.org/10.1109/ICDE.2015.7113353
16. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: l-diversity: privacy beyond k-anonymity. ACM Transactions on Knowledge Discovery from Data (TKDD) 1(1), Article 3 (2007), https://doi.org/10.1145/1217299.1217302
17. McSherry, F.: Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In: Çetintemel, U., Zdonik, S.B., Kossmann, D., Tatbul, N. (eds.) Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '09). pp. 19–30. ACM, New York, NY, USA (2009), https://doi.org/10.1145/1559845.1559850
18. Monroy-Hernández, A.: NYC taxi trips (June 2014), http://www.andresmh.com/nyctaxitrips/
19. Munz, R., Eigner, F., Maffei, M., Francis, P., Garg, D.: UniTraX: Protecting data privacy with discoverable biases. Tech. Rep. MPI-SWS-2018-001, Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern and Saarbrücken, Germany (February 2018), https://www.mpi-sws.org/tr/2018-001.pdf
20. Nissim, K., Vadhan, S.P., Xiao, D.: Redrawing the boundaries on purchasing data from privacy-sensitive individuals. In: Naor, M. (ed.) Proceedings of the 5th Conference on Innovations in Theoretical Computer Science (ITCS '14). pp. 411–422. ACM, New York, NY, USA (2014), https://doi.org/10.1145/2554797.2554835
21. NYC Taxi & Limousine Commission: TLC trip record data (May 2017), http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
22. Rastogi, V., Nath, S.: Differentially private aggregation of distributed time-series with transformation and encryption. In: Elmagarmid, A.K., Agrawal, D. (eds.) Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '10). pp. 735–746. ACM, New York, NY, USA (2010), https://doi.org/10.1145/1807167.1807247
23. Shi, E., Chan, T.H., Rieffel, E.G., Chow, R., Song, D.: Privacy-preserving aggregation of time-series data. In: Proceedings of the Symposium on Network and Distributed System Security (NDSS '11). ISOC (2011), https://www.isoc.org/isoc/conferences/ndss/11/pdf/9_3.pdf
24. Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, Ú., Gunda, P.K., Currey, J.: DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In: Draves, R., van Renesse, R. (eds.) Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08). pp. 1–14. USENIX (2008), https://www.usenix.org/event/osdi08/tech/full_papers/yu_y/yu_y.pdf